KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Sparsity in Large Scale Kernel Models

Raghvendra Mall

Promotor:
Prof. dr. ir. Johan A. K. Suykens

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering Science

June 2015

Sparsity in Large Scale Kernel Models

Raghvendra MALL

Jury Members:
Em. prof. dr. ir. Yves Willems, chair
Prof. dr. ir. Johan A. K. Suykens, promotor
Em. prof. dr. ir. Joos Vandewalle
Prof. dr. ir. Hugo Van hamme
Prof. dr. ir. Luc De Raedt
Prof. dr. ir. Renaud Lambiotte
(University of Namur, Belgium)
Prof. dr. ir. Jean-Charles Lamirel
(University of Strasbourg, France)

Dissertation presented in partial
fulfillment of the requirements for
the degree of Doctor
in Engineering Science

June 2015

© 2015 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Raghvendra Mall, Kasteelpark Arenberg 10, bus 2446, B-3001 Heverlee (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Preface

The work presented in this thesis is related to the research carried out during my doctoral studies at the ESAT-STADIUS research group. It has been a precious time full of rich experiences both at a professional and a personal front.

Poet Robert Frost said it best - **"Miles to go before I sleep"**. This has been the motto during the entire course of my doctoral studies. Fumbling, stumbling to being calm and steady, the journey across the novel and exciting problems in large scale machine learning has been enriching and enlightening. From being a complete novice to transform into a somewhat expert in certain directions has been worth the hard work and today I feel the efforts have paid-off.

However, it is not just an individual effort that makes this journey so special. It is the people who inspired me, helped me and collaborated with me, who made this voyage a fascinating one. I would first like to thank my promoter Professor Johan Suykens. He has been the best guide allowing me the flexibility to delve deep into research directions which fascinate me, helped me to enhance my abilities and knowledge from science to sports and played the role of a fatherly figure when it came to personal matters.

Besides my supervisor, I feel that I have to mention my assessors from ESAT (Em. Prof. Joos Vandewalle and Prof. Hugo Van Hamme) who gave me constructive criticism throughout my assessments which encouraged me to dig deeper and obtain a better understanding of the problems I faced. I deeply appreciate your suggestions and guidance.

I would like to thank my current and former colleagues from the STADIUS research group ranking them in order of importance (since I work on ranking) - Rocco, Vilen, Siamak, Mauricio, Gervasio, Antoine, Kim, Ricardo, Emanuele, Marco, Marko, Lynn, Oliver, Xiaolin, Zahra, Yunlong, Phillip, Kris, Carolina, Carlos, Andreas, Lei, Halima (QCRI), Mohammed (QCRI)- for being the constant source of inspiration, motivation and most importantly fun. I have to give special thanks to Rocco who has been the mentor below the promoter and

has been part of the majority of the research that I have conducted during my doctorate. I would also like to thank Vilen in whom I found a friend for life. I will never forget the discussions I had with all you guys be it about work, sports, life, politics or anything. I will miss our traditional poker nights. I will take this opportunity to thank my Indian friends here in Leuven - Parimal, Manish, Chandan, Dabas, Eshwar, Anjan, Sagnik, Abhijit, Bharat - for making Belgium feel like home. Finally, I am thankful to the administrative and technical staff at STADIUS, in particular to Ida, John, Wim, Elsy, Liesbeth and Maarten, for efficient management of all the bureaucratic and technical issues that I faced along the way.

I owe special gratitude to my family for all their love and support. Without their sacrifice and grace, I would never have been here at the first place.

Thank you all! Thank you for being part of this unbelievable voyage!

Raghvendra Mall
Leuven, June 2015.

Abstract

In the modern era with the advent of technology and its widespread usage there is a huge proliferation of data. Gigabytes of data from mobile devices, market basket, geo-spatial images, search engines, online social networks etc. can be easily obtained, accumulated and stored. This immense wealth of data has resulted in massive datasets and has led to the emergence of the concept of Big Data. Mining useful information from this big data is a challenging task. With the availability of more data the choices in selecting a predictive model decreases, because very few tools are computationally feasible for processing large scale datasets. A successful learning framework to perform various learning tasks like classification, regression, clustering, dimensionality reduction, feature selection etc. is offered by Least Squares Support Vector Machines (LSSVM) which is designed in a primal-dual optimization setting. It provides the flexibility to extend core models by adding additional constraints to the primal problem, by changing the objective function or by introducing new model selection criteria.

The goal of this thesis is to explore the role of sparsity in large scale kernel models using core models adopted from the LSSVM framework. Real-world data is often noisy and only a small fraction of it contains the most relevant information. Sparsity plays a big role in selection of this representative subset of data. We first explored sparsity in the case of large scale LSSVM using fixed-size methods with a re-weighted L_1 penalty on top resulting in very sparse LSSVM (VS-LSSVM).

An important aspect of kernel based methods is the selection of a subset on which the model is built and validated. We proposed a novel fast and unique representative subset (FURS) selection technique to select a subset from complex networks which retains the inherent community structure in the network. We extend this method for Big Data learning by constructing k-NN graphs out of dense data using a distributed computing platform i.e. Hadoop and then apply the FURS selection technique to obtain representative subsets on top of which models are built by kernel based methods.

We then focused on scaling the kernel spectral clustering (KSC) technique for big data networks. We devised two model selection techniques namely balanced angular fitting (BAF) and self-tuned KSC (ST-KSC) by exploiting the structure of the projections in the eigenspace to obtain the optimal number of communities k in the large graph. A multilevel hierarchical kernel spectral clustering (MH-KSC) technique was then proposed which performs agglomerative hierarchical clustering using similarity information between the out-of-sample eigen-projections.

Furthermore, we developed an algorithm to identify intervals for hierarchical clustering using the Gershgorin Circle theorem. These intervals were used to identify the optimal number of clusters at a given level of hierarchy in combination with KSC model. The MH-KSC technique was extended from networks to images and datasets using the BAF model selection criterion. We also proposed optimal sparse reductions to KSC model by reconstructing the model using a reduced set. We exploited the Group Lasso and convex re-weighted L_1 penalty to sparsify the KSC model.

Finally, we explored the role of re-weighted L_1 penalty in case of feature selection in combination with LSSVM. We proposed a visualization (Netgram) toolkit to track the evolution of communities/clusters over time in case of dynamic time-evolving communities and datasets.

Real world applications considered in this thesis include classification and regression of large scale datasets, image segmentation, flat and hierarchical community detection in large scale graphs and visualization of evolving communities.

Abbreviations

LSSVM	Least Squares Support Vector Machines
KKT	Karush Kuhn Tucker
FS-LSSVM	Fixed-Size Least Squares Support Vector Machines
PFS-LSSVM	Primal Fixed-Size Least Squares Support Vector Machines
SVM	Support Vector Machine
SD-LSSVM	Subsampled Dual Least Squares Support Vector Machines
SPFS-LSSVM	Sparse Primal Fixed-Size Least Squares Support Vector Machines
SSD-LSSVM	Sparse Subsampled Dual Least Squares Support Vector Machines
PV	Prototype Vectors
SV	Support Vectors
FURS	Fast and Unique Representative Subset
KSC	Kernel Spectral Clustering
k**-NN**	k-Nearest Neighbor
RBF	Radial Basis Function
ECOC	Error Correcting of Codes
BAF	Balanced Angular Fitting
BLF	Balanced Line Fitting
DD	Degree Distribution
CCF	Clustering Co-efficients
CC	Cut-Conductance
ARI	Adjusted Rand Index
VI	Variation of Information
DB	Davies-Bouldin Index
ST-KSC	Self Tuned Kernel Spectral Clustering
MH-KSC	Multilevel Hierarchical Kernel Spectral Clustering
AH-KSC	Agglomerative Hierarchical Kernel Spectral Clustering
HKSC	Hierarchical Kernel Spectral Clustering
SKSC	Soft Kernel Spectral Clustering
MKSC	Kernel Spectral Clustering with Memory

Contents

Abstract iii

Abbreviations v

Contents vii

1 Introduction 1

1.1 General Background 1

1.2 Role of Sparsity and Other Challenges 7

1.2.1 Classification & Regression 8

1.2.2 Community Detection 9

1.2.3 Clustering 10

1.2.4 Feature Selection 11

1.2.5 Visualization 12

1.3 Objectives and Motivations 13

1.4 Contributions of this work 15

2 Sparse Reductions to LSSVM for Large Scale Data 23

2.1 Very Sparse LSSVM Reductions 23

2.1.1 Introduction 24

2.1.2 Initializations 26
2.1.3 Sparsifications 30
2.1.4 Computational Complexity & Experimental Results . . 36
2.1.5 Sparsity versus Error Trade-off 44
2.1.6 Conclusion 47
2.2 Sparse Reductions to FS-LSSVM 47
2.2.1 Introduction 48
2.2.2 SV_L_0-norm PFS-LSSVM 50
2.2.3 Window reduced FS-LSSVM 52
2.2.4 Computational Complexity and Experimental Results . 53
2.2.5 Conclusion 58

3 Subset Selection **59**

3.1 FURS selection retaining community structure 59
3.1.1 Introduction 60
3.1.2 Related Work 63
3.1.3 Proposed Method 64
3.1.4 Evaluation Metrics 70
3.1.5 Experiments 71
3.1.6 Conclusion 74
3.2 Representative subsets for Big Data Learning using k-NN graphs 75
3.2.1 Introduction 75
3.2.2 Distributed k-NN graph generation framework 77
3.2.3 FURS for Weighted Graphs 80
3.2.4 Classification Experiments 80
3.2.5 Clustering Experiments 83
3.2.6 Conclusion 86

4 Large Scale Community Detection **89**

4.1 Kernel Spectral Clustering for Big Networks 90

4.1.1 Introduction 90

4.1.2 Kernel Spectral Clustering 91

4.1.3 Model Selection by means of Angular Similarity 95

4.1.4 Selecting a Representative Subgraph 97

4.1.5 Experiments and Analysis 98

4.1.6 Conclusion 103

4.2 Self-Tuned Kernel Spectral Clustering 104

4.2.1 Introduction 104

4.2.2 Self-Tuned Kernel Spectral Clustering 105

4.2.3 Experiments 108

4.2.4 Conclusion 114

4.3 Multilevel Hierarchical Kernel Spectral Clustering 114

4.3.1 Introduction 115

4.3.2 Multilevel Hierarchical KSC 117

4.3.3 Experiments 123

4.3.4 Visualization and Illustrations 129

4.3.5 Conclusion 131

5 Hierarchical & Sparse Kernel Spectral Data Clustering **137**

5.1 Identifying Intervals for Hierarchical Clustering 138

5.1.1 Introduction 138

5.1.2 Proposed Method 140

5.1.3 Spectral Clustering 144

5.1.4 Experiments 146

5.1.5 Conclusion 148

5.2 Agglomerative Hierarchical Kernel Spectral Data Clustering . . 150
5.2.1 Introduction 151
5.2.2 Agglomerative Hierarchical KSC approach 152
5.2.3 Experiments 155
5.2.4 Conclusion 159
5.3 Sparse Kernel Spectral Clustering 162
5.3.1 Introduction 162
5.3.2 Sparse reductions to KSC model 163
5.3.3 Experiments on Real World Datasets 169
5.3.4 Conclusion 173

6 Applications of Supervised & Unsupervised Kernel Methods 175

6.1 Reweighted L_1 LSSVM for Feature Selection 175
6.1.1 Introduction 176
6.1.2 Proposed Method 177
6.1.3 Experimental Results 180
6.1.4 Conclusion 183
6.2 Netgram: Visualizing Communities in Evolving Networks . . . 185
6.2.1 Introduction 185
6.2.2 Related Work 187
6.2.3 Evolution of Communities 190
6.2.4 Netgram Tool 196
6.2.5 Experiments 201
6.2.6 Conclusion 206

7 Conclusions & Future Work 209

7.1 General Conclusions 209
7.2 Major Contributions 213

7.3 Future Work 214

Bibliography **217**

Curriculum vitae **233**

List of publications **235**

Chapter 1

Introduction

1.1 General Background

We live in an era where with the advent of technology and its widespread usage, there is a huge proliferation of data. The advancement of information technologies has impacted both science and society. Gigabytes of data are easily available from mobile devices, sensory networks, market basket, geo-spatial images, search engines, online social networks etc. This data can be easily obtained, accumulated and stored efficiently. The immense wealth of available information has resulted in massive datasets and led to the emergence of the concept of Big Data.

Some of the characteristics of Big Data are **volume**, **variety**, **velocity** and **veracity**. While **volume** gives importance to the quantity of information, **variety** refers to the category to which the Big Data belongs. Depending on the source of information, the Big Data might be generated from health care industry, online social networks like Facebook or LinkedIn, videos uploaded and watched on Youtube etc. The **velocity** aspect refers to the rate at which the data is being generated and processed to identify the hidden patterns and make new scientific discoveries. For example, the New York stock exchange captures approximately 1 terabyte of trade information during each trading session (`http://www.ibmbigdatahub.com/infographic/four-vs-big-data`). Finally, **veracity** refers to the uncertainty in the data. Because data is generated from heterogeneous sources of information inconsistencies can result if the data is not handled and managed efficiently. For instance, poor data quality costs the US economy around 3.1 trillion dollars per year (`http://www.ibmbigdatahub.`

com/infographic/four-vs-big-data).

In this thesis we focus on the **volume** aspect of Big Data. Particularly, *the topic of the thesis was to explore the role of sparsity in advanced data driven black-box modeling techniques for large scale data.* The field of mining and recognizing complex patterns in large scale data is often referred as data mining [1] or machine learning [2]. The idea is to "learn" a model from a dataset and then make decisions about previously unseen data based on what it has learned. This task is of crucial importance to extract meaningful knowledge from large scale datasets. However, as the size of the data increases the choice of selection of modeling techniques decrease. This is because very few tools are computationally feasible for processing large scale datasets.

One direction of research has been to make use of the current stat-of-the-art machine learning techniques for small scale data and to extend these on a distributed computing environment. Recently, a tool named Mahout [3] (`http://www.manning.com/owen/`) was built such that several machine learning algorithms were implemented for Big Data using a distributed framework referred as Hadoop [4]. Oryx (`https://github.com/OryxProject/oryx`) is another machine learning tool for Big Data which uses the Apache Spark (`http://spark.apache.org/`) distributed computing framework. Several state-of-the-art machine learning techniques like Random forests [5], k-means [6] and principal component analysis (PCA) [7] are available in these tools.

The other direction is to design efficient model based learning techniques which are fast, scalable and might support parallelization or distributed computing. One such class of models which support this scheme are kernel based methods. Kernel methods, particularly those based on the least squares support vector machines (LSSVMs) [8, 9], belong to the class of machine learning techniques where we build the models in a primal-dual optimization framework. The task at hand is formulated such that modeling is performed in a learning framework with a proper training, validation and test phase which is essential for predictive purposes and good generalization capabilities. For kernel methods, during the training phase, we first map the original data to a high dimensional feature space. Then, the design of a learning algorithm in that space allows to discover complex and non-linear representations in the original input space [8, 10]. The optimal model parameters are obtained during the validation phase and the generalization performance of the model are tested on previously unseen data during the test phase. In this thesis a major role is played by the LSSVM [8, 9] which belongs to the class of support vector machines (SVM [10]).

The LSSVM [8, 9] is formulated in a constrained optimization framework with a squared loss function in the objective and equality constraints instead of the inequality constraints which are used in the SVM [10] formulation.

A major advantage of the LSSVM formulation is that by modifying the objective and/or adding additional new constraints to the core formulation, it is possible to develop models for a variety of learning tasks with the aid of a **systematic model selection** procedure (validation phase) resulting in **high generalization performance** using the **out-of-sample extensions property**. Figure 1.1 showcases the LSSVM working principle.

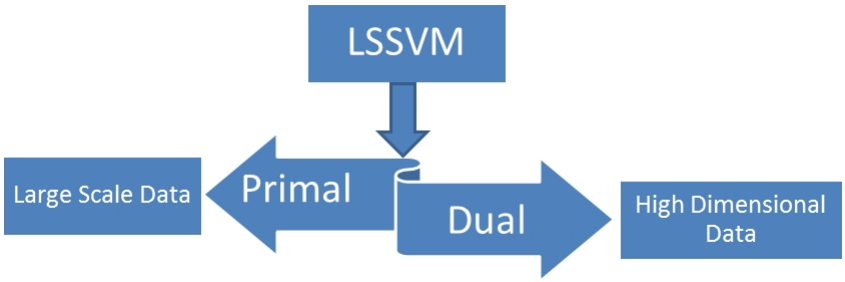


Figure 1.1: Linear LSSVM [8, 9] framework allows to solve both large scale and high dimensional problems by making use of the primal-dual optimization framework. However, the challenge is to develop simple and scalable non-linear LSSVMs which can solve large/massive size problems.

It was shown in [8] that kernel based models using the LSSVM framework can solve a variety of problems including supervised learning tasks like classification, regression, Bayesian inference [11] and unsupervised learning tasks like density estimation [12] and principal component analysis (PCA) [7]. In [13], the authors used an LSSVM based framework to develop an unsupervised clustering technique named kernel spectral clustering (KSC) for datasets which was extended for small scale complex networks in [14]. The authors of [15] proposed a semi-supervised formulation to the core KSC model by modifying the objective function. A kernel regularized correlation (KRC) technique was proposed in [16] using the LSSVM framework to obtain useful estimates of the canonical correlation in high dimensional feature space. A primal-dual framework for feature selection and anomaly detection using LSSVMs was proposed in [17] and [18] respectively. Figure 1.2 highlights the diverse range of machine learning problems that can be achieved using variations of LSSVM models.

These techniques show the power of kernel based methods using the LSSVM framework for performing a wide range of different machine learning tasks. These type of tasks are common in fields such as statistics, engineering, quantum mechanics, artificial intelligence, pattern recognition, optimization, linear algebra and data mining which are intersecting point of several disciplines

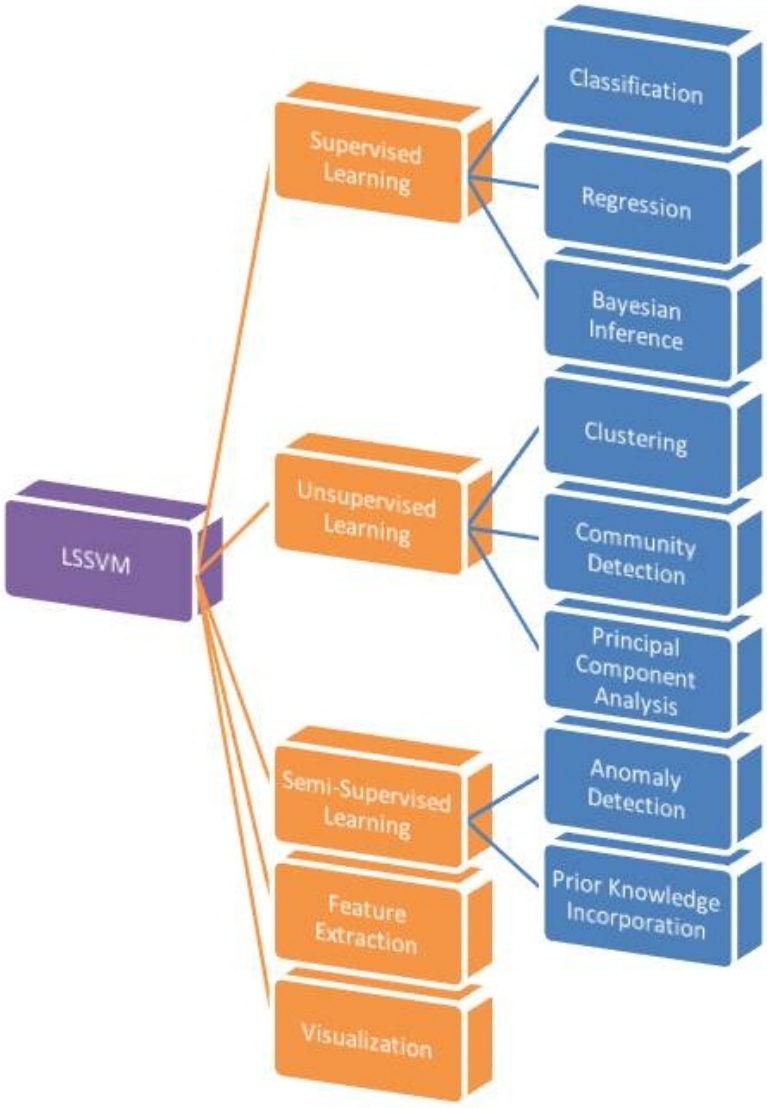


Figure 1.2: Wide variety of machine learning tasks that can be performed by using the LSSVM as the core model.

of science and mathematics. Figure 1.3 represents this quality of LSSVM models.

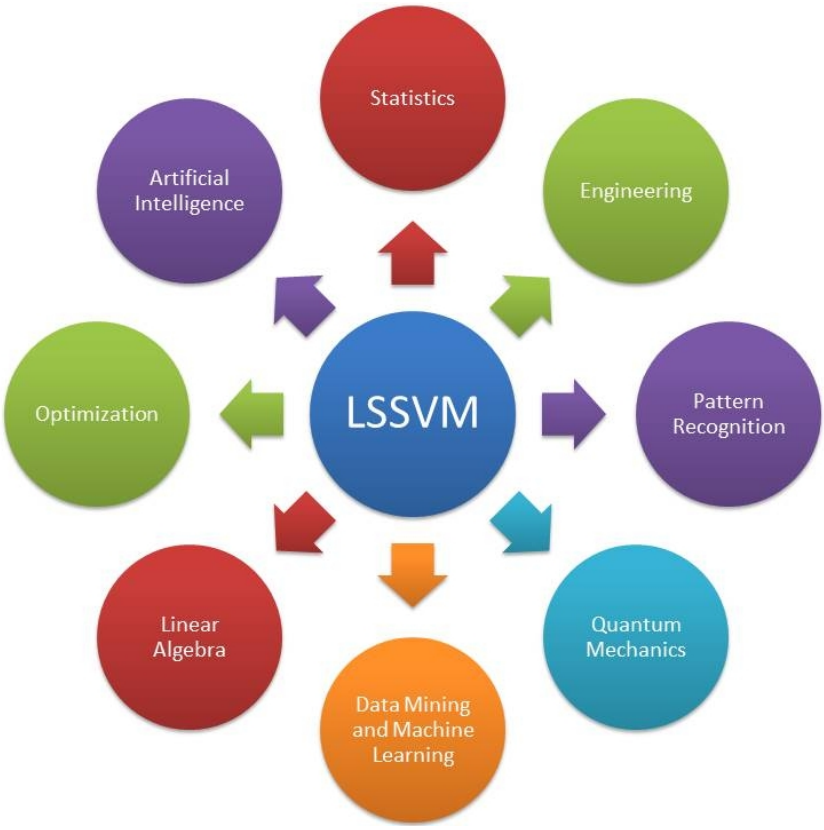


Figure 1.3: Kernel based methods especially LSSVMs are quite interdisciplinary as depicted in this figure.

Some of the practical applications where these kernel based methods using LSSVM framework have been very successful include ovarian cancer prediction [19], anomaly detection [18], community detection in large scale complex networks [20, 21], time-series prediction [22], colour based image segmentation [13], chemometrics [23], seizure detection [24] etc. Figure 1.4 illustrates the different types of data and their corresponding applications for which LSSVM based models have proven to be quite successful.

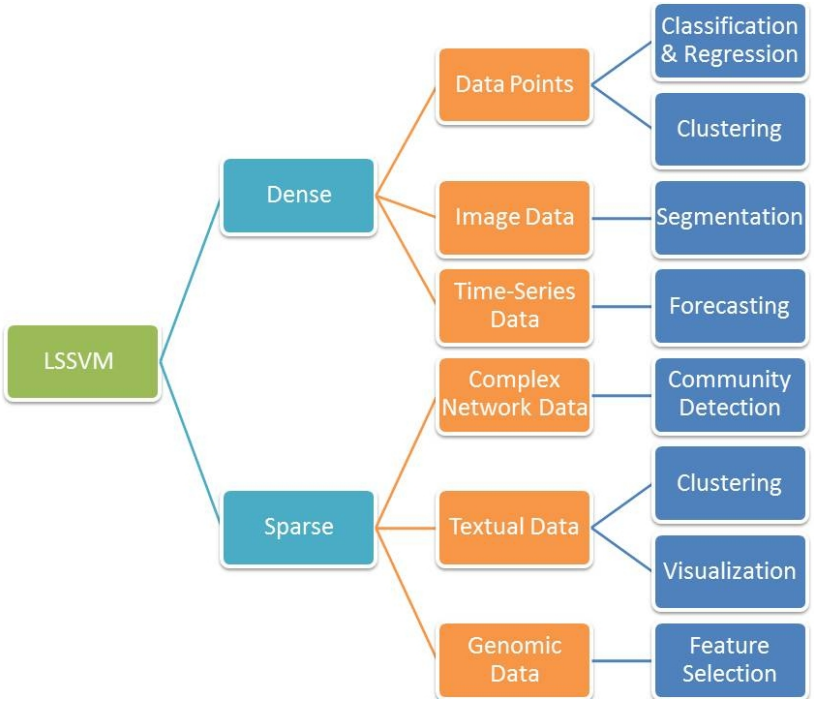


Figure 1.4: Different data types and their applications on which LSSVM based models have been applied successfully.

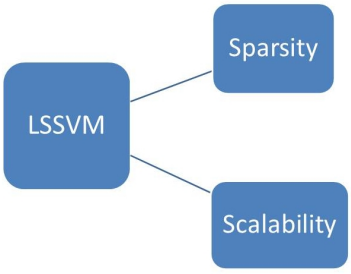


Figure 1.5: Challenges faced by LSSVM based methods and describe them in detail in Section 1.2.

1.2 Role of Sparsity and Other Challenges

Real world data is often noisy and only a small fraction of data contains the maximum amount of information necessary to build predictive models for pattern recognition and knowledge discovery. Often these large scale datasets are structured in the sense that they are sparse or compressible. In a biological experiment, one could measure changes of expression in $30,000$ genes and expect at most only a couple hundred of genes with a different expression level. Similarly, in signal processing, one could sample or sense signals which are known to be sparse (or approximately so) when expressed in the correct basis. Most real world complex networks like online social networks, collaboration networks, trust networks etc. are known to be sparse. This premise radically changes the problem, making the search for solutions feasible since the simplest solution now tends to be the right one.

Sparsity in machine learning algorithms [10] comes to the rescue and provides an effective way to design predictive models for large scale datasets. One benefit of sparsity is that it results in a model where among all the coefficients that describe the model only a small number are non-zero. Another aspect that can be considered as part of sparsity is the selection of representative subsets of data on which the model is built and validated. If this subset does not capture the intrinsic nature of the large scale dataset then it would not result in good generalization performance. It is relatively easier to scale simple models for Big Data learning. Thus, sparsity plays a big role in generating simple and scalable predictive kernel models which are trained on representative subsets of data for various machine learning tasks. Figure 1.6 illustrates several directions by means of which sparsity can be introduced in kernel based methods.

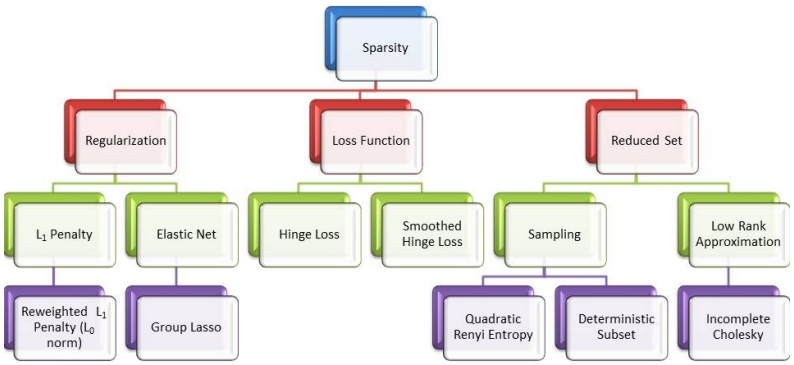


Figure 1.6: Some of the techniques by means of which sparsity can be introduced in kernel based models.

We describe the main issues tackled in this thesis with respect to sparsity for different machine learning tasks along with other challenges faced in the following Subsections.

1.2.1 Classification & Regression

Support vector machines [10] and least squares support vector machines [9] are state-of-the-art kernel based techniques for performing learning tasks like classification and regression. The standard SVM solves a convex optimization problem with inequality constraints whereas in the LSSVM formulation we have equality constraints and a L_2 loss function. The inequality constraints along with the complementarity condition result in zero values for the Lagrange multipliers in the SVM formulation thereby resulting in sparsity. The non-zero Lagrange multipliers results in the set of relevant input data referred as support vectors (SV). However, it was shown in [25] that the number of SV increases linearly with the size of the dataset i.e. N. The LSSVM model lacks sparsity and each input data is a SV. Another issue with SVM and LSSVM models is that the size of the kernel matrix required to store in memory is $O(N \times N)$ and the computational complexity of solving the problem is $O(N^3)$. Hence as the value of N increases these methods becomes computationally in-feasible and memory intensive.

Several type of methods have been proposed to handle sparsity and scalability issues in kernel methods:

- **Reduction methods**: These methods train the model on the dataset, prune the support vectors and then select the rest for retraining the model. Some works in this category include [26, 27, 28, 29, 25]. However, since they train the model on the dataset, they cannot overcome the original computational constraints and apply these models for large scale datasets.
- **Direct methods**: These methods enforce sparsity from the beginning. In these methods, the number of SV referred as prototype vectors (PV) are fixed in advance. One such method was introduced in [8] and was called fixed-size least squares support vector machines (FS-LSSVM). Methods based on this concept were extended for large scale datasets in [30, 31]. However, the number of prototype vectors required to obtain good generalization performance are not known beforehand and the subsets on which the model are built and validated need not necessarily be representative of the intrinsic characteristics of the large scale data.
- **Linear methods**: Several linear kernel methods [32, 33] exist which can scale up to large scale datasets. However, in this case we loose the ability

to map the input data to high dimensional feature space and tackle the non-linearity in the input space.

1.2.2 Community Detection

The problem of community detection in complex networks is formulated such that nodes belonging to one community are densely connected and between the communities the connections are sparse. Real world networks like social networks, collaboration networks, biological networks, communication networks etc. exhibit community like structure. The problem of community detection has also been framed as graph partitioning and graph clustering [34, 35]. Several methods have been proposed to handle the task of community detection [36, 37, 38, 39, 40, 41, 42, 13, 14] in complex networks. However, among the myriad variety of techniques available for community detection, few of them can scale for large scale networks [36, 42, 40].

We provide the main characteristics of some of these techniques:

- **Louvain method**: This method greedily optimizes the modularity criterion proposed in [37] resulting in a simple and scalable method to extract communities in large scale networks in a hierarchical fashion. Modularity measures the difference between a given partition of the network and the expectation of the same partition for a random network. This method first detects communities at lower level of hierarchy by greedily maximizing modularity and then considers these communities as nodes to re-perform the steps to obtain communities for the next level of hierarchy. The process stops until the modularity criterion cannot be further maximized. However, it was shown in [43] that modularity suffers from resolution limit i.e. it cannot identify modules beyond a scale.

- **Infomap method**: This method uses an information theoretic approach to hierarchical community detection. It uses the probability flow of random walks as a substitute for information flow in complex networks. It then fragments the network into modules by compressing a description of the information flow. However, the Infomap method generally works well when there are few levels of hierarchy in the network and its scalability to large networks is a challenge.

- **OSLOM method**: This method was proposed by [40] to avoid the issue of resolution limit and locate statistically significant communities in the network. The method is based on local optimization of a fitness function expressing the statistical significance of communities with respect

to random fluctuations. It can uncover hierarchical and overlapping community structure in large scale networks. However, in this thesis we show that this method works well for benchmark synthetic networks [44] but in case of real world networks it is unable to detect quality clusters at coarser levels of granularity.

There is no or a very limited role of sparsity in these community detection techniques. Each such technique tries to optimize a different criterion to unfold the underlying hierarchical community structure in large scale network. As a result, the communities obtained by these techniques vary a lot from each other. Hence, another challenge is the use of the right criteria to evaluate the quality of the obtained communities. Since community detection is an unsupervised learning task there is a need to have both good internal and external quality metrics [45] to evaluate the communities obtained by different community detection technique.

1.2.3 Clustering

Like community detection, clustering is an unsupervised learning task where the goal is to organize the data into natural groups for a given dataset. Clusters are defined such that the data present within the group are more similar to each other in comparison to the data between clusters. The most commonly used clustering technique is the k-means [6] technique. The algorithm is based on initial selection of k prototypes as cluster means and then calculation of distance of each point from those means. Each point is allocated a cluster based on its minimum distance from all the cluster means. Then, the mean of the cluster is updated. This approach is performed iteratively till the configuration of the clusters don't change. The k-means technique is simple, fast ($O(N)$) and scalable. However, it works best in case of equal sized spherical clusters. The cluster quality can suffer from initial randomization. As clustering is an unsupervised learning task the right value of k or ideal number of clusters is usually not known beforehand.

Spectral clustering techniques can overcome some of these issues. Spectral clustering comprises a family of clustering methods that make use of the eigenvectors of a normalized affinity matrix from the data to group points that are similar. These methods [46, 47, 48, 49] are formulated as relaxation of graph partitioning algorithms which are NP-complete. These methods can handle complex non-linear structure in the input space but require the number of clusters k to be known beforehand. Their time complexity is $O(N^3)$ as it needs to perform and eigen-decomposition and their space requirement is $O(N^2)$ to store the affinity matrix. Hence, it is not feasible to use these techniques for

clustering large scale datasets. Recently, a kernel-based modeling approach to spectral clustering was proposed in [13] and was referred as Kernel Spectral Clustering (KSC). The KSC formulation links spectral clustering with weighted kernel PCA via primal-dual insights in a constrained optimization framework typical of LSSVM. One of its main advantages is the powerful out-of-sample extensions property which allows cluster affiliation for previously unseen data. This makes KSC a good candidate for clustering large scale datasets.

Challenges faced by KSC - The KSC methodology [13] has been formulated in a learning framework with a proper training, validation and test phase. It is important to train and validate the model on representative subset of the data which captures the inherent clustering information in the large scale data. There is a need for proper model selection to estimate the right kernel parameter σ and the number of clusters k. The authors of [13] proposed a Balanced Line Fitting (BLF) criterion to obtain optimal model parameters. However, this criterion *works well only in case of well separated clusters*. The dual predictive model of KSC is based on non-sparse kernel expansions. So, the authors in [50, 51] proposed techniques to select reduced set of points on which the KSC model is built. These techniques result in *sparse solutions but not necessarily the optimal ones*. A hierarchical version of KSC (HKSC) was proposed in [50]. At each level of hierarchy they generate a KSC model using a different set of model parameters (σ, k). As a result when the clusters are being merged at two level of hierarchies some points from the merging clusters might go to different clusters as indicated in [52]. These points are forced to join the merging cluster of the majority and *the natural agglomerative hierarchical behavior is lost*.

1.2.4 Feature Selection

There are several benefits of performing feature selection like facilitating data interpretation, reducing storage requirements, decreasing computational complexity and defeating curse of dimensionality to improve prediction performance of models. Techniques such as *filter* methods exist to select variables based on correlations, mutual information, Fisher criterion etc. as depicted in [53, 54, 55]. Generally *filter* methods are used as pre-processing step and are not as powerful as *wrapper* methods.

Wrapper methods utilize a learning procedure to score subsets of variable according to their predictive power. Many of these *wrapper* methods work by direct objective minimization. Generally, the objective function consists of two terms - (1) a penalty on the number of variables to be selected which has to be minimized and (2) the predictive accuracy of the classifier which has to be

maximized. These methods try to formalize an objective function for variable selection which leads to algorithms to optimize it.

Methods which use the L_1-norm penalty for this purpose include [56, 57, 54]. The L_1-norm penalty also shrinks the fitted coefficients towards zero and under certain conditions reduces some of the fitted coefficients to *exactly* zero. However, the L_1-norm penalty suffers from two serious limitations - (1) In case of highly correlated variables, the L_1-norm penalty tends to pick only one or few of them instead of selecting all the correlated features as a group. (2) In case of $d \gg N$, it was proved in [58] that the L_1-norm penalty can keep at most N input features. In order to overcome these shortcomings a doubly regularized support vector machine was proposed in [59] which uses the *elastic net* [60] penalty i.e. a combination of L_2-norm and L_1-norm penalty. Similar approaches named the *group lasso* and *sparse group lasso* were proposed in [61] and [62] respectively to overcome these drawbacks. An adaptive L_1-norm penalty known as *Adaptive Lasso* was introduced in [63] and an *adaptive elastic net* penalty was introduced in [64]. However, whenever we use either the L_1-norm penalty or L_1-norm penalty in combination with other penalties, we end up solving a quadratic programming problem due to inequality constraints laid by the L_1-norm penalty. Since solving a linear system is easier than solving a QP, we prefer formulations which lead to solving a system of linear equations. One such method which uses the LSSVM for feature selection based on low rank updates was introduced in [65].

In [56], the FSV method for feature selection was proposed. In this method an approximation to the zero-norm was proposed such that $||w||_0 = \text{card}\{w|w \neq 0\} \approx \sum_i 1 - \exp\{1 - \alpha|w_i|\}$ where α is the parameter to be tuned. In an alternative approach namely AROM method [66], the authors explored $\sum_i \ln\{\epsilon + |w_i|\}$ as a surrogate of zero-norm in optimization, where $0 < \epsilon \ll 1$ is a parameter to be tuned. However, there are two shortcomings of these methods as well. Firstly, these methods are mere approximation of the true-zero norm and optimizing an approximation term instead of the true objective makes the methods computationally expensive. Secondly, we have to tune an additional parameter for each of the two methods.

1.2.5 Visualization

Visualizing evolution of communities: In many real-life applications the data or networks are non-stationary and change over time. Several dynamic clustering algorithms [67, 68, 69, 70, 71, 72] exist which allow detecting and tracking these evolving communities over time. However, there doesn't exist a tool which allows to visualize and track the evolution of the communities

obtained independently by any of these dynamic clustering techniques in a simple and structured manner.

1.3 Objectives and Motivations

We outline the objectives and motivations of this thesis below:

- **The first objective is to design very sparse least squares support vector machines (VS-LSSVM) for large scale data.** Sparse LSSVM models result in simpler models which are memory and computationally efficient. It is essential for practical purposes like scaling the algorithm on a laptop to large scale datasets comprising **millions of data points**. Sparse solutions means fewer support vectors and less time required for operations like out-of-sample extensions. By having an additional layer of sparsity in the case of primal FS-LSSVM (PFS-LSSVM), we overcome the problem of initial selection of the cardinality (M) for the prototype vector (PV) set. It also allows to study the trade-off between amount of sparsity and the predictive accuracy of the VS-LSSVM models.
- **The second objective is to build kernel based methods on representative subsets of data.** In case of clustering or community detection it is important to identify representative subsets from the large scale data/network in a fast and deterministic manner. The resulting subsets are used as training and validation set respectively when building the kernel based model. These subsets should retain the inherent community structure present in the Big data in order to obtain good generalization performance for previously unseen data.
- **The third objective to extend the kernel spectral clustering (KSC) technique to large scale complex networks for community detection**. There are several issues which arise in this context. One of the issues is the model selection i.e. identifying the natural number of communities present in the network along with the right kernel parameters. Another issue is that the KSC model should be efficient enough to infer community affiliation for previously unseen nodes in computationally effective way such that the entire community detection procedure is fast and scalable to networks with **tens of millions of nodes and hundreds of millions of edges on a laptop in less than few minutes**.
- **The fourth objective is to design a kernel based model for hierarchical community detection or clustering**. The goal is to use the KSC technique as a core model to have a natural multilevel

hierarchical clustering technique which is built in an agglomerative fashion. The clustering technique uses the local affinity between clusters at one level of hierarchy to generate communities at another level of hierarchy. The technique should be able to detect high quality clusters at coarser as well as finer levels of hierarchy thereby overcoming the problem of resolution limit. Finally, the model should be scalable for large scale complex networks with **millions of nodes and hundreds of millions of edges**. Another related objective is to obtain intervals to estimate the range for ideal value of k at a given level of hierarchy without performing computationally expensive eigen-decomposition step. This helps to reduce the search space for optimal model parameter i.e. number of clusters k at a given level of hierarchy.

- **The fifth objective is to design optimal reduced sets for sparse kernel spectral clustering.** The goal is to find reduced sets of training points which can best approximate the original solution. This can be done by having an additional layer of sparsity. The advantage of these sparse reductions is that it results in much simpler and faster predictive KSC models and reduce the time complexity of the out-of-sample extensions.

- **The final objective is to make a visualization toolkit to detect and track the evolution of communities in dynamic networks**. Real world complex networks are often non-stationary and evolve over time. In order to design such a toolkit we need to have a mechanism to track the community labels over time. This is because cluster labels assigned to communities at one time-stamp might change at another time-stamp. The toolkit should be able to handle significant events like birth, death, merge, split, continuation, growth and shrinkage of communities over time. Finally, the visualization should be such that they adhere to aesthetic conditions like minimal usage of screen space and minimization of line cross-overs during significant events like merge and split.

In the modern era with widespread availability of large scale data, it becomes easier to perform machine learning tasks on them. Here we provide information about several such repositories where large scale datasets are freely available and which we utilized for conducting experiments in this thesis:

- Datasets like **Forest Cover** ($\approx 600,000$ **points and** 54 **dimensions**) and **Year Prediction** ($\approx 500,000$ **points and** 90 **dimensions**) are used for classification and regression respectively and are available at [73].

- Datasets like **KDDCupBio** ($\approx 150,000$ **points and** 74 **dimensions**) and **Susy** ($\approx 5,000,000$ **points and** 18 **dimensions**) are used for

clustering and are available at `http://cs.joensuu.fi/sipu/datasets/` and [73] respectively.

- Image datasets where each **color image has a configuration of** 350×450 **pixels** and are used for tasks like color based image segmentation can be obtained from the Berkeley Segmentation Dataset `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/`.

- The Stanford Network Analysis Project (SNAP) `http://snap.stanford.edu/` provides massive scale anonymized networks like **Youtube (**$\approx 1,200,000$ **nodes and** $3,000,000$ **edges)** and **LiveJournal (**$\approx 4,000,000$ **nodes and** $35,000,000$ **edges)** that can be used for problems like flat and hierarchical community detection.

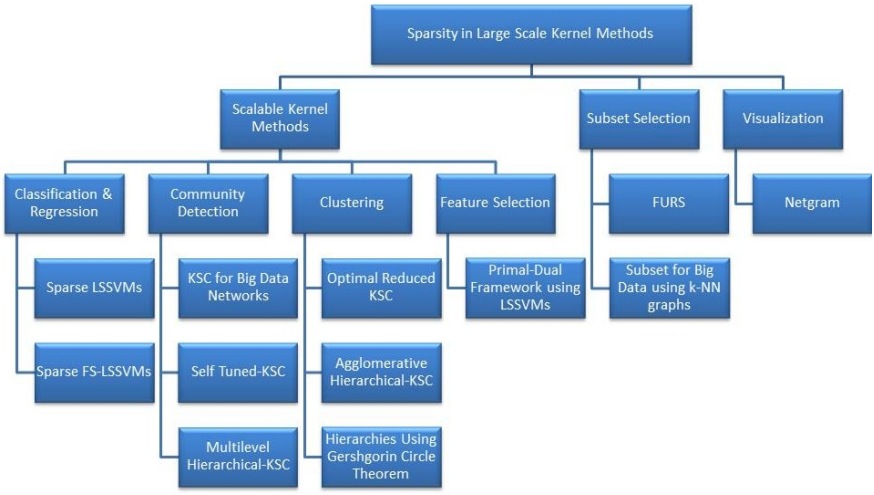


Figure 1.7: Outline of the contributions in this doctoral thesis

1.4 Contributions of this work

The main contributions of this thesis are summarized as the follows:

- **Exploring the sparsity vs error trade-off in case of LSSVM variants for large scale data**. We propose very sparse reductions to primal FS-LSSVM (PF-LSSVM) and a dual subsampled LSSVM (SD-LSSVM) by using a convex re-weighted L_1-norm penalty. We also explored

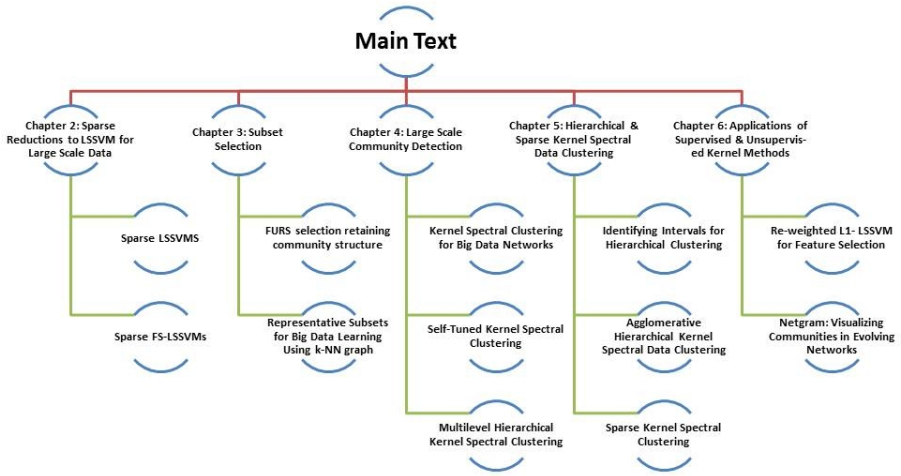


Figure 1.8: **Thesis Outline**: Chapter by Chapter Overview

a reduced set based technique for FS-LSSVM selecting prototype vectors (PV) closest and farthest from decision boundaries as support vectors. By using this additional layer of sparsity we overcome the challenge of selection of the right cardinality M for the initial PV set selection in case of FS-LSSVM. We are able to reduce the time complexity of out-of-sample extensions property, explore the sparsity versus error trade-off and illustrate the effectiveness of the proposed methods on large scale data. The related contributions are:

1. Mall R., Suykens J.A.K., **"Very Sparse LSSVM Reductions for Large Scale Data"**, IEEE Transactions on Neural Networks and Learning Systems, IEEE Transactions on Neural Networks and Learning Systems, 26(5), March 2015, pp. 1086-1097.
2. Mall R., Suykens J.A.K., **"Sparse Reductions for Fixed-Size Least Squares Support Vector Machines on Large Scale Data"**, in Proc. of the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2013), GoldCoast, Australia, Apr. 2013, pp. 161-173.

- **Selecting representative subsets from large scale networks and datasets**. Kernel based methods build their models on a training set and perform validation procedure on a separate validation set. Thus, it is extremely essential that the training and validation is performed on representative subsets of data particularly for a learning task like clustering. So, we proposed a fast and unique representative subset (FURS) selection

for community detection in large scale networks. The FURS method greedily selects nodes with high degree centrality from different dense regions in the graph retaining the natural community structure. It uses concepts of activation, de-activation and re-activation of topology of the graph for this purpose. We also use the FURS technique for Big Data learning. Here the Big Data is first converted into a k-NN graph using a distributed framework like Hadoop and then FURS is performed to obtain the training and validation sets. We also compared FURS with state-of-the-art sampling techniques like random sampling, stratified random sampling, subset selection using quadratic Rènyi entropy [30, 8] in case of datasets and Slashburn [74], Snowball expansion sampling [75], Metropolis sampling [76] and forest fire sampling [77] techniques in case of large scale networks. The related contributions are:

1. Mall R., Langone R., Suykens J.A.K., **"FURS: Fast and Unique Representative Subset selection retaining large scale community structure"**, Social Network Analysis and Mining, 3(4), Oct 2013, pp. 1075-1095.
2. Mall R., Jumutc V., Langone R., Suykens J.A.K., **"Representative Subsets For Big Data Learning using k-NN graphs"**, IEEE BigData, Washington D.C., U.S.A, 2014, pp. 37-42.

- **Extend the Kernel Spectral Clustering to Big Data Networks**. We extend the kernel spectral clustering (KSC [13]) to perform community detection for big data networks. We propose a new model selection technique namely Balanaced Angular Fitting (BAF) which uses the concept of angular similarity in the validation node projections to obtain the optimal number of clusters k in the network. The technique is fast, scalable and also acts as a metric to evaluate the quality of the communities discovered. Another metric was proposed which uses the concept entropy and balance to automatically identify the optimal value of k in large scale networks resulting in a self-tuned KSC model. We exploited the structure of the eigen-projections throughout the thesis to obtain flat as well as hierarchical clustering organization for real world big data networks. The related contributions include:

  1. Mall R., Langone R., Suykens J.A.K., **"Kernel Spectral Clustering for Big Data Networks"**, Entropy, Special Issue: Big Data, 15(5), May 2013, pp. 1567-1568.
  2. Mall R., Langone R., Suykens J.A.K., **"Self-Tuned Kernel Spectral Clustering for Large Scale Networks"**, in Proc of IEEE International Conference on Big Data (IEEE BigData), Santa Clara, USA, Oct 2013, pp. 385-393.

- **Agglomerative/Multilevel Hierarchical Kernel Spectral Clustering (AH-KSC/MH-KSC)**. We propose a novel multilevel hierarchical kernel spectral clustering (MH-KSC) technique for large scale networks which overcomes the resolution limit problem. The technique uses the eigen-projections of the validation nodes to generate a series of affinity matrices iteratively which result in distance thresholds determining the distance between communities. These distance thresholds are then used in combination with test projections to obtain a hierarchical clustering organization for large scale networks in an agglomerative fashion. The proposed method uses local similarity information to generate good quality clusters at both coarser as well as finer levels of granularity, a feature usually absent in hierarchical community detection methods like Louvain [36], Infomap [42] and OSLOM [40] methods. We extend the MH-KSC technique to datasets and images by first using the BAF criterion to obtain an optimal set of model parameters (σ, k) which generally correspond to one level of hierarchy. Then, we use the same procedure as that in MH-KSC to obtain an agglomerative clustering organization. We also evaluate the proposed technique with state-of-the-art agglomerative hierarchical clustering techniques like single-link, complete-link, median-link and average-link hierarchical clustering techniques. We also proposed a novel technique to determine intervals to estimate the ideal value of k at a given level of hierarchy for datasets using the Gershgorin Circle Theorem [78]. We exploit the piece-wise constant nature of the upper bounds to the eigenvalues of the Laplacian matrix obtained by Gershgorin Circle Theorem to reduce the search space for identifying the optimal number of clusters at a given level of hierarchy. We use these intervals in combination with a hierarchical kernel spectral [52] technique to show the effectiveness of the proposed method. The related contributions include:

  1. Mall R., Langone R., Suykens J.A.K., **"Multilevel Hierarchical Kernel Spectral Clustering for Real-Life Large Scale Complex Networks"**, PLoS ONE, 9(6):e99966, Jun 2014.
  2. Mall R., Langone R., Suykens J.A.K., **"Agglomerative hierarchical kernel spectral data clustering"**, IEEE SSCI CIDM, Dec 2014, pp. 9-16.
  3. Mall R., Mehrkanoon S., Suykens J.A.K., **"Identifying intervals for hierarchical clustering using the Gershgorin circle theorem"**, Pattern Recognition Letters, 55, April 2015, pp. 1-7.

- **Sparse Reductions to KSC model**. We proposed sparse reductions to the KSC model to make the clustering models simpler and reduce the out-of-sample extension time complexity. We use the group lasso [61] and re-weighted L_1-norm penalty [79, 80] in combination with the

reconstruction error to obtain optimal feasible sparse reduced sets. We compare the same with L_1-norm penalty proposed in [51]. The related contribution is:

  - Mall R., Mehrkanoon S., Langone R., Suykens J.A.K., **"Optimal reduced sets for sparse kernel spectral clustering"**, IEEE IJCNN, July 2014, pp. 2436-2443.

- **Applications of Supervised & Unsupervised kernel based methods**. We proposed a new primal dual framework for feature selection using the LSSVM framework. We add the convex relaxation of L_0-norm in the form of a re-weighted L_1 penalty term to the LSSVM objective to obtain a robust and efficient feature selection technique. The re-weighted L_1 penalty removes the noisy features and the L_2 penalty term supports grouping of essential features. After multiple randomizations we consistently obtain the same set of features and overcome this drawback suffered by LASSO [63] method. We also propose a novel method to detect and rank overlap and outlier points using the soft kernel spectral clustering [81] technique. Using structural as well as similarity information we rank these outlier and overlap points and show that the proposed ranking is different from traditional information retrieval ranking procedures. Contributions related to this part are:

  1. Mall R., El Anbari M., Bensmail H., Suykens J.A.K., **"Primal-Dual Framework for Feature Selection using Least Squares Support Vector Machines"**, in Proc. of the 19th International Conference on Management of Data (COMAD), Ahmedabad, India, Dec. 2013, pp. 105-108.
  2. Mall R., Langone R., Suykens J.A.K., **"Ranking Overlap and Outlier Points in Data using Soft Kernel Spectal Clustering"**, in Proc. of European Symposium on Artificial Neural Networks (ESANN), Bruges, Belgium, 2015.

- **Visualizing evolution of communities for dynamic complex networks**. We designed a toolkit to visualize evolution of communities in time-varying graphs. Real world networks are non-stationary and are dynamic in nature. The goal of the toolkit is to detect and track evolution of individual communities in a simple line-based visualization. We proposed a new tracking mechanism as clustering is abstract and the labels of the same community in one time-stamp might differ from its label at another time-stamp. The proposed tool can capture occurrence of significant events like birth, death, merge, split, growth, shrinkage and continuation of communities using a visualization scheme where we

try to minimize the line cross-overs between communities. The related contribution is:

– Mall R., Langone R., Suykens, J.A.K., **"Netgram: Visualizing Communities in Evolving Networks"**, submitted to PloS ONE.

Figures 1.9 and 1.10 represent the contributions of this thesis w.r.t. exploiting sparsity and providing scalability to kernel based methods using LSSVM as the core model for a wide variety of machine learning problems.

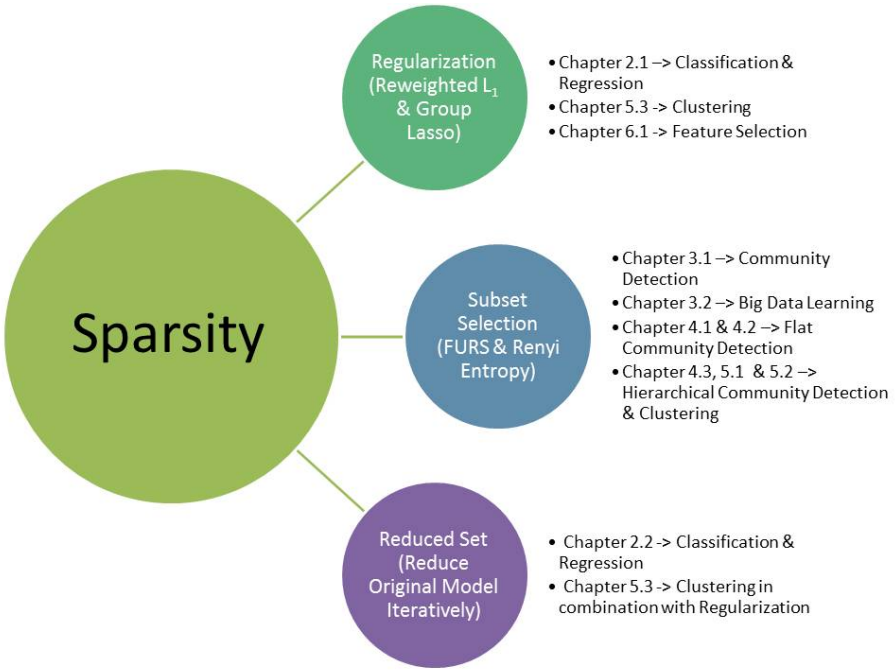


Figure 1.9: Exploiting sparsity in this thesis for various machine learning problems and in which section of the chapters it appears.

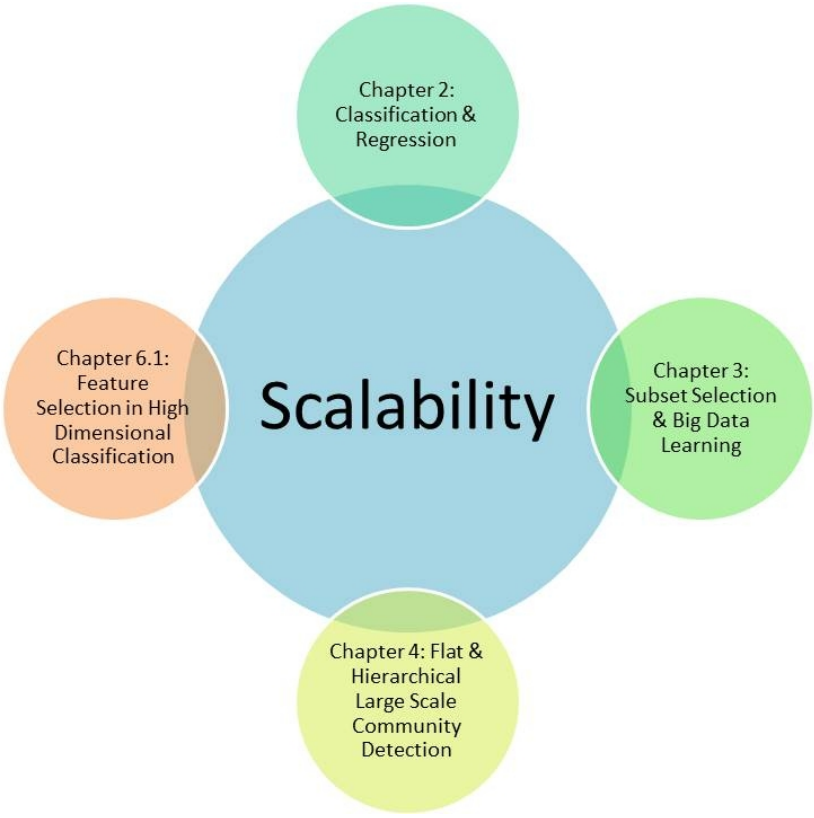


Figure 1.10: Scalability issues appearing in LSSVM based models proposed in this thesis have been tackled in the Chapters shown above for problems like classification, regression, community detection and feature selection.

Chapter 2

Sparse Reductions to LSSVM for Large Scale Data

This chapter comprises previously published articles including:

1) Mall R., Suykens J.A.K., **"Very Sparse LSSVM Reductions for Large Scale Data"**, IEEE Transactions on Neural Networks and Learning Systems, vol. 26, no. 5, Mar. 2015, pp. 1086 - 1097.

2) Mall R., Suykens J.A.K., **"Sparse Reductions for Fixed-Size Least Squares Support Vector Machines on Large Scale Data"**, in Proc. of the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2013), GoldCoast, Australia, Apr. 2013, pp. 161-173.

Keywords—L_0-norm; reduced models; classification & regression; sparsity;

2.1 Very Sparse LSSVM Reductions[1]

Abstract—Least Squares Support Vector Machines (LSSVM) have been widely applied for classification and regression with comparable performance to SVMs. The LSSVM model lacks sparsity and is unable to handle large scale data due

[1]This section consists of Section $1-6$ from Mall R., Suykens J.A.K., **"Very Sparse LSSVM Reductions for Large Scale Data"**, IEEE Transactions on Neural Networks and Learning Systems, vol. 26, no. 5, Mar. 2015, pp. 1086 - 1097.

to computational and memory constraints. A primal Fixed-Size LSSVM (PFS-LSSVM) was previously proposed in [8] to introduce sparsity using Nyström approximation with a set of prototype vectors (PV). The PFS-LSSVM model solves an over-determined system of linear equations in the primal. However, this solution is not the sparsest. We investigate the sparsity-error trade-off by introducing a second level of sparsity. This is done by means of L_0-norm based reductions by iteratively sparsifying LSSVM and PFS-LSSVM models. The exact choice of the cardinality for the initial PV set is not important then as the final model is highly sparse. The proposed method overcomes the problem of memory constraints and high computational costs resulting in highly sparse reductions to LSSVM models. The approximations of the two models allow to scale the models to large scale datasets. Experiments on real world classification and regression datasets from the UCI repository illustrate that these approaches achieve sparse models without a significant trade-off in errors.

2.1.1 Introduction

Least Squares Support Vector Machines (LSSVM) were introduced in [9] and have become a state-of-the-art learning technique for classification and regression. In the LSSVM formulation instead of solving a quadratic programming problem with inequality constraints as in the standard SVM [10], one has equality constraints and the L_2-loss function. This leads to an optimization problem whose solution in the dual is obtained by solving a system of linear equations.

A drawback of LSSVM models is the lack of sparsity as usually all the data points become support vectors (SV) as shown in [8]. Several works in the literature address this problem of lack of sparsity in the LSSVM model. They can be categorized as:

1. *Reduction methods*: - Training the model on the dataset, pruning support vectors and selecting the rest for retraining the model.
2. *Direct methods*: - Enforcing sparsity from the beginning.

Some works in the first category are [82, 27, 26, 28, 29, 83] and [84]. In [27], the authors provide an approximate SVM solution under the assumption that the classification problem is separable in the feature space. In [26] and [28], the proposed algorithm approximates the weight vector such that the distance to the original weight vector is minimized. The authors of [29] eliminate the support vectors that are linearly dependent on other support vectors. In [83] and [84], the authors work on a reduced set for optimization by pre-selecting a subset of data as support vectors without emphasizing much on the selection

methodology. The authors of [82] prune the support vectors which are farthest from the decision boundary. This is done recursively until the performance degrades. Another work [85] in this direction suggests to select the support vectors closer to the decision boundary. However, these techniques cannot guarantee a large reduction in the number of support vectors.

In the second category, the number of support vectors referred to as prototype vectors (PVs) are fixed in advance. One such approach is introduced in [8] and is referred to as *fixed-size least squares support vector machines* (FS-LSSVM). It provides a solution to the LSSVM problem in the primal space resulting in a parametric model and a sparse representation. The method uses an explicit expression for the feature map using the Nyström method [86] and [87]. The Nyström method is related to finding a low rank approximation to the given kernel matrix by choosing M rows or columns from the large $N \times N$ kernel matrix. In [8], the authors proposed searching for M rows or columns by maximizing the quadratic Rènyi entropy criterion. It was shown in [30] that the cross-validation error of primal FS-LSSVM (PFS-LSSVM) decreases with respect to the number of selected PVs until it does not change anymore and is heavily dependent on the initial set of PVs selected by quadratic Rènyi entropy. This point of "saturation" can be achieved for $M \ll N$ but this is not the sparsest solution. A sparse conjugate direction pursuit approach was developed in [88] where they iteratively build up a conjugate set of vectors of increasing cardinality to approximately solve the over-determined PFS-LSSVM linear system. The approach works most efficiently when few iterations suffice for a good approximation. However, when few iterations don't suffice for approximating the solution the cardinality will be M.

In recent years the L_0-norm has been receiving increasing attention. The L_0-norm is the number of non-zero elements of a vector. So when the L_0-norm of a vector is minimized it results into the sparsest model. But this problem is NP-hard. Therefore, several approximations to it are discussed in [66] and [79] etc. In this work, we modify the iterative sparsification procedure introduced in [80] and [89]. The major drawbacks of the methods described in [80] and [89] are that these approaches cannot scale to very large scale datasets due to memory ($N \times N$ kernel matrix) and computational ($O(N^3)$ time) constraints. We reformulate the iterative sparsification procedure for LSSVM and PFS-LSSVM methods to produce highly sparse models. These models can efficiently handle very large scale data. We discuss two different initialization methods for which in a next step the sparsification step is applied:

- Initialization by Primal Fixed-Size LSSVM: Sparsification of the primal fixed-size LSSVM (PFS-LSSVM) method leads to a highly sparse parametric model namely sparsified primal FS-LSSVM (SPFS-LSSVM).

- Initialization by Subsampled Dual LSSVM: The subsampled dual LSSVM (SD-LSSVM) is a fast initialization to the LSSVM model solved in the dual. Its sparsification results into a highly sparse non-parametric model namely sparsified subsampled dual LSSVM (SSD-LSSVM).

We compare the proposed methods with state-of-the-art techniques including C-SVC, ν-SVC from the LIBSVM [25] software, Keerthi's method [31], L_0-norm based method proposed by Lopez [89] and the L_0-reduced PFS-LSSVM method (SV_L_0-norm PFS-LSSVM) [90] on several benchmark datasets from the UCI repository [73]. Below we mention some motivations to obtain a sparse solution:

- Sparseness can be exploited for having more memory and computationally efficient techniques, e.g. in matrix multiplications and inversions.
- Sparseness is essential for practical purposes such as scaling the algorithm to very large scale datasets. Sparse solutions means fewer support vectors and less time required for out-of-sample extensions.
- By introducing two levels of sparsity, we overcome the problem of selection of the smallest cardinality (M) for the PV set faced by the PFS-LSSVM method.
- The two level of sparsity allows scaling to large scale datasets while having very sparse models.

We also investigate the sparsity versus error trade-off.

2.1.2 Initializations

In this work, we consider two initializations. One is based on solving the least squares support vector machine problem in the primal (PFS-LSSVM). The other is a fast initialization method solving a subsampled least squares support vector machines problem in the dual (SD-LSSVM).

Primal FS-LSSVM

Least Squares Support Vector Machine—We provide a brief summary of the Least Squares Support Vector Machines (LSSVM) methodology for classification and regression.

Given a sample of N data points $\{x_i, y_i\}$, $i = 1, ..., N$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$ for classification and $y_i \in \mathbb{R}$ for regression, the LSSVM primal

problem is formulated as follows:

$$\begin{aligned} \min_{w,b,e} \quad & \mathcal{J}(w,e) = \frac{1}{2} w^\intercal w + \frac{\gamma}{2} \sum_{i=1}^{N} e_i^2 \\ \text{s.t.} \quad & w^\intercal \phi(x_i) + b = y_i - e_i, i = 1, \ldots, N, \end{aligned} \tag{2.1}$$

where $\phi : \mathbb{R}^d \to \mathbb{R}^{n_h}$ is a feature map to a high dimensional feature space, where n_h denotes the dimension of the feature space (which can be infinite dimensional), $e_i \in \mathbb{R}$ are the errors and $w \in \mathbb{R}^{n_h}$, $b \in \mathbb{R}$.

Using the coefficients α_i for the Lagrange multipliers, the solution to (2.1) can be obtained by the Karush-Kuhn-Tucker (KKT) [91] conditions for optimality. The result is given by the following linear system in the dual variables α_i:

$$\left[\begin{array}{c|c} 0 & 1_N^\intercal \\ \hline 1_N & \Omega + \frac{1}{\gamma} I_N \end{array}\right] \left[\begin{array}{c} b \\ \hline \alpha \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline y \end{array}\right], \tag{2.2}$$

with $y = (y_1, y_2, \ldots, y_N)^\intercal$, $1_N = (1, \ldots, 1)^\intercal$, $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_N)^\intercal$ and $\Omega_{kl} = \phi(x_k)^\intercal \phi(x_l) = K(x_k, x_l)$, for $k, l = 1, \ldots, N$ with K a Mercer kernel function. From the KKT conditions we get that $w = \sum_{i=1}^{N} \alpha_i \phi(x_i)$ and $\alpha_i = \gamma e_i$. The second condition causes the LSSVM to be non-sparse as whenever e_i is non-zero then $\alpha_i \neq 0$. Generally, in real world scenarios the $e_i \neq 0, i = 1, \ldots, N$ for most data points. This leads to lack of sparsity in the LSSVM model.

Nyström Approximation and Primal Estimation—For large datasets it is often advantageous to solve the problem in the primal where the dimension of the parameter vector $w \in \mathbb{R}^d$ is smaller compared to $\alpha \in \mathbb{R}^N$. However, one needs an explicit expression for ϕ or the approximation of the nonlinear mapping $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}^M$ based on a sampled set of prototype vectors (PV) from the whole dataset. In [30], the authors provide a method to select this subsample of size $M \ll N$ by maximizing the Quadratic Rènyi entropy.

Williams and Seeger [92] uses the Nyström method to compute the approximated feature map $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}^M, i = 1, \ldots, M$ for a training point, or for any new point x^*, with $\hat{\phi} = (\hat{\phi}_1, \ldots, \hat{\phi}_M)^\intercal$, is given by

$$\hat{\phi}_i(x^*) = \frac{1}{\sqrt{\lambda_i^s}} \sum_{j=1}^{M} (u_i)_j K(z_j, x^*), \tag{2.3}$$

where λ_i^s and u_i denote the eigenvalues and the eigenvectors of the kernel matrix $\bar{\Omega} \in \mathbb{R}^{M \times M}$ with $\bar{\Omega}_{ij} = K(z_i, z_j)$, where z_i and z_j belong to the subsampled set $\mathcal{S}_{PV}$ which is a subset of the whole dataset $\mathcal{D}$. The matrix $\bar{\Omega}$ relates to a subset

of the big kernel matrix $\Omega \in \mathbb{R}^{N\times N}$. However, we should never calculate this big kernel matrix Ω in our proposed methodologies. The computation of the features corresponding to each point $x_i \in \mathcal{D}$ in matrix notation can be written as:

$$\hat{\Phi} = \begin{bmatrix} \hat{\phi}_1(x_1) & \dots & \hat{\phi}_M(x_1) \\ \vdots & \ddots & \vdots \\ \hat{\phi}_1(x_N) & \dots & \hat{\phi}_M(x_N) \end{bmatrix}. \tag{2.4}$$

Solving (2.1) with the approximate feature matrix $\hat{\Phi} \in \mathbb{R}^{N\times M}$ in the primal as proposed in [8] results into solving the following linear system of equations:

$$\left[\begin{array}{c|c} \hat{\Phi}^\intercal\hat{\Phi} + \frac{1}{\gamma}I & \hat{\Phi}^\intercal 1_N \\ \hline 1_N^\intercal\hat{\Phi} & 1_N^\intercal 1_N \end{array}\right] \left[\begin{array}{c} \hat{w} \\ \hline \hat{b} \end{array}\right] = \left[\begin{array}{c} \hat{\Phi}^\intercal y \\ \hline 1_N^\intercal y \end{array}\right], \tag{2.5}$$

where $\hat{w} \in \mathbb{R}^M$, $\hat{b} \in \mathbb{R}$ are the model parameters in the primal space with $y \in \{+1, -1\}$ for classification and $y \in \mathbb{R}$ for regression.

Parameter Estimation for Very Large Datasets—In [30], the authors propose a technique to obtain tuning parameters for very large scale datasets. We utilize the same methodology to obtain the parameters of the model ($\hat{w}$ and $\hat{b}$) when the approximate feature matrix $\hat{\Phi}$ given by (2.4) cannot fit into memory. The basic concept is to decompose the feature matrix $\hat{\Phi}$ into a set of S blocks. Thus, $\hat{\Phi}$ is not required to be stored into memory completely. Let l_s, where $s = 1, \dots, S$, denote the number of rows in the sth block such that $\sum_{s=1}^{S} l_s = N$. The matrix $\hat{\Phi}$ can be described as:

$$\hat{\Phi} = \begin{bmatrix} \hat{\Phi}_{[1]} \\ \vdots \\ \hat{\Phi}_{[S]} \end{bmatrix},$$

with $\hat{\Phi}_{[S]} \in \mathbb{R}^{l_s\times(M+1)}$ and the vector y is given by

$$y = \begin{bmatrix} y_{[1]} \\ \vdots \\ y_{[S]} \end{bmatrix},$$

with $y_{[S]} \in \mathbb{R}^{l_s}$. The matrix $\hat{\Phi}_{[S]}^\intercal\hat{\Phi}_{[S]}$ and the vector $\hat{\Phi}_{[S]}^\intercal y_{[S]}$ can be calculated in an updating scheme and stored efficiently in the memory since their sizes are $(M+1)\times(M+1)$ and $(M+1)\times 1$ respectively, provided that the size of each

block, i.e., l_s can fit into memory. Moreover, the following also holds:

$$\hat{\Phi}^\intercal \hat{\Phi} = \sum_{s=1}^{S} \hat{\Phi}_{[s]}^\intercal \hat{\Phi}_{[s]}, \hat{\Phi}^\intercal y = \sum_{s=1}^{S} \hat{\Phi}_{[s]}^\intercal y_{[s]}.$$

Algorithm 1 summarizes the overall idea.

Algorithm 1: PFS-LSSVM for very large scale data [30]

Divide the training data $\mathcal{D}$ into approximately S equal blocks such that $\hat{\Phi}_{[s]}$ with $s = 1, \ldots, S$, calculated using (2.4) can fit into memory.
Initialize matrix $A \in \mathbb{R}^{(M+1)\times(M+1)}$ and $c \in \mathbb{R}^{M+1}$.
for $s = 1$ *to* S **do**
    Calculate matrix $\hat{\Phi}_{[s]}$ for the sth block using Nyström approximation (2.4)
    $A \leftarrow A + \hat{\Phi}_{[s]}^\intercal \hat{\Phi}_{[s]}$
    $c \leftarrow c + \hat{\Phi}_{[s]}^\intercal y_{[s]}$
end
Set $A \leftarrow A + \frac{I_{M+1}}{\gamma}$
Solve the linear system (2.5) to obtain parameters $\hat{w}$,$\hat{b}$.

Algorithm 2: Primal FS-LSSVM method

Data: $\mathcal{D} = \{(x_i, y_i) : x_i \in \mathbb{R}^d, y_i \in \{+1, -1\}$ for classification & $y_i \in \mathbb{R}$ for regression, $i = 1, \ldots, N\}$.
Determine the kernel bandwidth using the multivariate rule-of-thumb.
Given the number of PV, perform prototype vector selection by maximizing the quadratic Rènyi entropy.
Determine the learning parameters σ and γ performing fast v-fold cross validation as described in [30].
if *the approximate feature matrix (2.4) can be stored into memory* **then**
    Given the optimal learning parameters, obtain the PFS-LSSVM parameters $\hat{w}$ and $\hat{b}$ by solving the linear equation (2.5).
else
    Use Algorithm 1 to obtain the PFS-LSSVM parameters $\hat{w}$ and $\hat{b}$.
end

Fast Initialization: Subsampled Dual LSSVM

In this case, we propose a different approximation instead of the Nyström approximation and solve a subsampled LSSVM problem in the dual (SD-LSSVM). We first use the active subset selection method as described in [30] to obtain an initial set of prototype vectors PV, i.e., $\mathcal{S}_{PV}$. This set of points is obtained by maximizing the quadratic Rènyi entropy criterion, i.e., approximate the information of the big $N \times N$ kernel matrix by means of a smaller $M \times M$ matrix and this can be considered as the set of representative points of the dataset.

The assumption for the approximation in the proposed approach is that this set of prototype vectors is sufficient to train an initial LSSVM model in the dual and is sufficient to obtain the tuning parameters σ and γ for the SD-LSSVM model. Here the major advantage is that it greatly reduces the computation time required for training and cross-validation ($O(M^3)$ in comparison to $O(NM^2)$ for PFS-LSSVM). This results in an approximate value for the tuning parameters close to the optimal values (for the entire training dataset). However, as the training of the LSSVM model is performed in the dual, we no longer need explicit approximate feature maps and can have the original feature map of the form $\phi : \mathbb{R}^d \to \mathbb{R}^{n_h}$ where n_h denotes the dimension of the feature space which can be infinite dimensional.

Thus, the SD-LSSVM problem of training on the M prototype vectors selected by Rènyi entropy is given by:

$$\begin{aligned} \min_{\bar{w},\bar{b},\bar{e}} \quad & \mathcal{J}(\bar{w},\bar{e}) = \frac{1}{2}\bar{w}^\intercal\bar{w} + \frac{\gamma}{2}\sum_{i=1}^{M}\bar{e}_i^2 \\ \text{s.t.} \quad & \bar{w}^\intercal\phi(z_i) + \bar{b} = y_i - \bar{e}_i, i = 1,\ldots,M, \end{aligned} \tag{2.6}$$

where $z_i \in \mathcal{S}_{PV}$ and $\mathcal{S}_{PV}$ is a subset of the whole dataset $\mathcal{D}$.

2.1.3 Sparsifications

We propose two L_0-norm reduced models starting from the initializations explained in Section 2.1.2: One for the primal FS-LSSVM method namely the sparsified primal FS-LSSVM (SPFS-LSSVM) and the other one for the SD-LSSVM method namely sparsified subsampled dual LSSVM (SSD-LSSVM). Both models can handle very large scale data efficiently.

L_0-norm Reduced PFS-LSSVM - SPFS-LSSVM

In this section, we propose an approach using the L_0-norm to introduce a second level of sparsity resulting in a reduced set of prototype vectors $\mathcal{S}_{SV}$ whose cardinality is M' and is giving a highly sparse solution. We modify the procedure described in [80] and [89]. The methodology used in [80] and [89] cannot be extended to large scale data due to memory constraints ($O(N \times N)$)

and computational costs ($O(N^3)$). The L_0-norm problem can be formulated as:

$$\begin{aligned} \min_{\tilde{w},\tilde{b},\tilde{e}} \quad & \mathcal{J}(\tilde{w},\tilde{e}) = \|\tilde{w}\|_0 + \frac{\gamma}{2}\sum_{i=1}^{N}\tilde{e}_i^2 \\ \text{s.t.} \quad & \tilde{w}^\intercal\hat{\phi}(x_i) + \tilde{b} = y_i - \tilde{e}_i, i = 1,\ldots,N, \end{aligned} \tag{2.7}$$

where $\tilde{w}$, $\tilde{b}$ and $\tilde{e}_i, i = 1,\ldots,N$ are the variables of the optimization problem and $\hat{\phi}$ is the explicit feature map as discussed in (2.3).

The weight vector $\tilde{w}$ can be approximated as a linear combination of the M prototype vectors, i.e., $\tilde{w} = \sum_{j=1}^{M}\tilde{\beta}_j\hat{\phi}(z_j)$ where $\tilde{\beta}_j \in \mathbb{R}$ which don't need to be the Lagrange multipliers. We apply the regularization weight λ_j on each of these $\tilde{\beta}_j$ to iteratively sparsify such that most of the $\tilde{\beta}_j$ move to zero leading to an direct L_0-norm solution as shown in [80]. The L_0-norm problem can then be re-formulated in terms of reweighting steps of the form:

$$\begin{aligned} \min_{\tilde{\beta},\tilde{b},\tilde{e}} \quad & J(\tilde{\beta},\tilde{e}) = \frac{1}{2}\sum_{j=1}^{M}\lambda_j\tilde{\beta}_j^2 + \frac{\gamma}{2}\sum_{i=1}^{N}\tilde{e}_i^2 \\ \text{s.t.} \quad & \sum_{j}^{M}\tilde{\beta}_j\hat{Q}_{ij} + \tilde{b} = y_i - \tilde{e}_i, i = 1,\ldots,N. \end{aligned} \tag{2.8}$$

The matrix $\hat{Q}$ is a rectangular matrix of size $N \times M$ and is defined by its elements $\hat{Q}_{ij} = \hat{\phi}(x_i)^\intercal\hat{\phi}(z_j)$ where $x_i \in \mathcal{D}$, $z_j \in \mathcal{S}_{PV}$. The set $\mathcal{S}_{PV}$ is a subset of the dataset $\mathcal{D}$. This problem (2.8) is similar to the one formulated for SVMs in [80] and guarantees sparsity and convergence. It is well known that the L_p-norm problem is non-convex for $0 < p < 1$. We obtain an approximate solution for $p \to 0$ by the iterative sparsification procedure and converge to a local minimum.

We propose to solve this problem in the primal which allows us to extend the sparsification procedure to large scale datasets along with incorporating the information about the entire training dataset $\mathcal{D}$. Thus, after eliminating the $\tilde{e}_i$ the optimization problem becomes:

$$\min_{\tilde{\beta},\tilde{b}} \quad J(\tilde{\beta},\tilde{b}) = \frac{1}{2}\sum_{j=1}^{M}\lambda_j\tilde{\beta}_j^2 + \frac{\gamma}{2}\sum_{i=1}^{N}(y_i - (\sum_{j=1}^{M}\tilde{\beta}_j\hat{Q}_{ij} + \tilde{b}))^2. \tag{2.9}$$

The solution to (2.9) resembles the ridge regression solution (in case of zero bias term) and is obtained by solving:

$$\left[\begin{array}{c|c} \hat{Q}^\intercal\hat{Q} + \frac{1}{\gamma}diag(\lambda) & \hat{Q}^\intercal 1_N \\ \hline 1_N^\intercal\hat{Q} & 1_N^\intercal 1_N \end{array}\right]\left[\begin{array}{c} \tilde{\beta} \\ \tilde{b} \end{array}\right] = \left[\begin{array}{c} \hat{Q}^\intercal y \\ \hline 1_N^\intercal y \end{array}\right] \tag{2.10}$$

where $diag(\lambda)$ is a diagonal $M \times M$ matrix with diagonal elements λ_j. The iterative sparsification method is presented in Algorithm 3.

Algorithm 3: SPFS-LSSVM method

Data: Solve PFS-LSSVM (2.5) to obtain initial $\hat{w}$ and $\hat{b}$
$\tilde{\beta} = \hat{w}$
$\lambda_i \leftarrow \tilde{\beta}_i$, $i = 1, \ldots, M$
if *the $\hat{Q}$ matrix can be stored into memory* **then**
  Calculate $\hat{Q}^\intercal\hat{Q}$ once and store into memory.
else
  Divide into blocks for very large datasets computing $\hat{Q}_{[s]}^\intercal\hat{Q}_{[s]}$ in an additive updating scheme similar to procedure in Algorithm 1.
  Calculate once and store the $M \times M$ matrix into memory.
end
while *not convergence* **do**
  $H \leftarrow \hat{Q}^\intercal\hat{Q} + diag(\lambda)/\gamma$;
  Solve system (2.10) to give $\tilde{\beta}$ and $\tilde{b}$;
  $\lambda_i \leftarrow 1/\tilde{\beta}_i^2, i = 1, \ldots, M$;
end
Result: $indices = find(|\tilde{\beta}_i| > 0)$, $\beta' = \tilde{\beta}(indices)$, $b' = \tilde{b}$.

The procedure to obtain sparseness involves iteratively solving the system (2.10) for decreasing values of λ. Considering the tth iteration, we can build the matrix $H \leftarrow \hat{Q}^\intercal\hat{Q} + diag(\lambda)/\gamma$ from the in-memory matrix $\hat{Q}^\intercal\hat{Q}$ and the modified matrix $diag(\lambda^t)$ and solve the system of linear equations. From this solution we get λ^{t+1} and the process is restarted. It was shown in [80] that as $t \to \infty$, $\tilde{\beta}_t$ converges to the L_0-norm solution asymptotically. This is shown in Algorithm 3. Since, this β' depends on the initial choice of weights, we set them to the PFS-LSSVM solution $\hat{w}$ and $\hat{b}$ to avoid ending up in very different local minima. For this procedure we need to calculate $\hat{Q}^\intercal\hat{Q}$ matrix just once and keep it into memory. The final predictive model is:

$$f(x^*) = \sum_{i=1}^{M'} \beta_i' \hat{\phi}(z_i)^\intercal \hat{\phi}(x^*) + b'.$$

We use $f(x^*)$ for regression and sign$[f(x^*)]$ for classification. Table 2.1 provides a conceptual and notational overview of the steps involved in SPFS-LSSVM model.

	Initial		1st **Reduction**		2nd **Reduction**
SV/Train	N/N		M/N		M'/N
Primal	w, $\mathcal{D}$	**Step 1** $\rightarrow$	$\hat{w}$, $\mathcal{S}_{\mathcal{PV}}$	**Step 2** $\rightarrow$	β', b', $\mathcal{S}_{\mathcal{SV}}$
	$\phi(x) \in \mathbb{R}^{n_h}$	**PFS-LSSVM**	$\hat{\phi}(x) \in \mathbb{R}^M$	**SPFS-LSSVM**	

Table 2.1: Given the dataset $\mathcal{D}$ we first perform the primal FS-LSSVM in Step 1. We obtain the prototype vector set $\mathcal{S}_{PV}$, the weight vector $\hat{w}$ along with the explicit feature map $\hat{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}^M$. In Step 2 we perform the SPFS-LSSVM, i.e., we use an iterative sparsifying L_0-norm procedure in the primal where $\tilde{w} = \sum_{j=1}^{M} \tilde{\beta}_j \hat{\phi}(z_j)$ and regularization weights λ_j are applied on $\tilde{\beta}_j$. We construct the $\hat{Q}$ matrix which has information about the entire training set $\mathcal{D}$. After the 2nd reduction we obtain the solution vector β' and b'. The solution vector relates to prototype vectors of the highly sparse solution set $\mathcal{S}_{SV}$.

L_0-norm reduced subsampled dual LSSVM - SSD-LSSVM

The subsampled dual LSSVM (SD-LSSVM) performs a fast initialization on a subsample obtained by maximizing the quadratic Rènyi entropy. The SD-LSSVM model as described in Section 2.1.2 results in $\bar{\alpha}, i = 1, \ldots, M$ and $\bar{b}$ as a solution in the dual. However, we have not seen all the data in the training set for this case. Also, we don't know beforehand the ideal value of the cardinality M for the initial PV set. In general, we start with a large value of M for fixing the initial cardinality of the PV set. But we propose an iterative sparsification procedure for the SD-LSSVM model leading to an L_0-norm based solution. This results in a set of reduced prototype vectors $\mathcal{S}_{SV}$ whose cardinality is M' which can be much less than M. We use these reduced prototype vectors along with the non-zero $\bar{\alpha}_i, i = 1, \ldots, M$ and $\bar{b}$ obtained as a result of the iterative sparsification procedure for the out-of-sample extensions (i.e. test data predictions). The L_0-norm problem for the SD-LSSVM model can then be formulated as:

$$\begin{aligned} \min_{\bar{w},\bar{b},\bar{e}} \quad & \mathcal{J}(\bar{w},\bar{e}) = \|\bar{w}\|_0 + \frac{\gamma}{2}\sum_{i=1}^{N} \bar{e}_i^2 \\ \text{s.t.} \quad & \bar{w}^\intercal \phi(x_i) + \bar{b} = y_i - \bar{e}_i, i = 1, \ldots, N, \end{aligned} \tag{2.11}$$

where $\bar{w}$, $\bar{b}$ and $\bar{e}_i, i = 1, \ldots, N$ are the variables of the optimization problem and ϕ is the original feature map.

One of the KKT conditions of (2.6) is $\bar{w} = \sum_{i=1}^{M} \bar{\alpha}_i \phi(z_i)$ where $\bar{\alpha}_i$ are Lagrange dual variables. We apply the regularization weight λ_j on each of these $\bar{\alpha}_j$

to iteratively sparsify such that most of the $\bar{\alpha}_j$ move to zero leading to an approximate L_0-norm solution as shown in [80]. In order to handle large scale datasets and obtain the non-zero $\bar{\alpha}_j$ leading to the reduced set of prototype vectors $\mathcal{S}_{SV}$, we formulate the optimization problem by eliminating each $\bar{e}_i$. Thus, the optimization problem can be reformulated as:

$$\min_{\bar{\alpha},\bar{b}} \quad J(\bar{\alpha},\bar{b}) = \frac{1}{2}\sum_{j=1}^{M}\lambda_j\bar{\alpha}_j^2 + \frac{\gamma}{2}\sum_{i=1}^{N}(y_i - (\sum_{j=1}^{M}\bar{\alpha}_j Q_{ij} + \bar{b}))^2. \tag{2.12}$$

The matrix Q is a rectangular matrix of size $N \times M$ and is defined by its elements $Q_{ij} = \phi(x_i)^\intercal\phi(z_j) = K(x_i, z_j)$ where $x_i \in \mathcal{D}$, $z_j \in \mathcal{S}_{PV}$. The variables x_j and z_j which are part of the PV set $\mathcal{S}_{PV}$ are used interchangeably. This marks the distinction between (2.12) and (2.9) where we use explicit approximate feature maps. The solution to (2.12) is obtained by solving:

$$\left[\begin{array}{c|c} Q^\intercal Q + \frac{1}{\gamma}diag(\lambda) & Q^\intercal 1_N \\ \hline 1_N^\intercal Q & 1_N^\intercal 1_N \end{array}\right]\left[\begin{array}{c} \bar{\alpha} \\ \bar{b} \end{array}\right] = \left[\begin{array}{c} Q^\intercal y \\ \hline 1_N^\intercal y \end{array}\right]. \tag{2.13}$$

Here $diag(\lambda)$ is a diagonal $M \times M$ matrix with diagonal elements λ_j. We train the SD-LSSVM only on the PV set (cardinality M) but we incorporate the information from the entire training dataset (cardinality N) in the loss function while performing the iterative sparsification algorithm. This results into an improvement in performance as more information is incorporated in the model. The approach to perform an iterative sparsification procedure for the SD-LSSVM model is presented in Algorithm 4.

Algorithm 4: SSD-LSSVM method

Data: Solve SD-LSSVM (2.6) on actively selected PV set to obtain the initial $\bar{\alpha}$ and $\bar{b}$
$\lambda_i \leftarrow \bar{\alpha}_i$, $i = 1, \ldots, M$
if *the Q matrix can be stored into memory* **then**
  Calculate $Q^\intercal Q$ once and store into memory.
else
  Divide into blocks for very large datasets computing $Q_{[s]}^\intercal Q_{[s]}$ in an additive updating scheme similar to procedure in Algorithm 1.
  Calculate once and store the $M \times M$ matrix into memory.
end
while *not convergence* **do**
  $H \leftarrow Q^\intercal Q + diag(\lambda)/\gamma$;
  Solve system (2.13) to give $\bar{\alpha}$ and $\bar{b}$;
  $\lambda_i \leftarrow 1/\bar{\alpha}_i^2, i = 1, \ldots, M$;
end
Result: $indices = find(|\bar{\alpha}_i| > 0)$, $\alpha' = \bar{\alpha}(indices), b' = \bar{b}$

The procedure to obtain sparseness works similarly as described in Section 2.1.3. Once we obtain the *indices* corresponding to the non-zero $\bar{\alpha}_i$, we can obtain the reduced set of prototype vectors SV corresponding to those non-zero $\bar{\alpha}_i$.

We can then use α' and b' (as defined in Algorithm 4) along with these SV to perform the test predictions. The final predictive model is:

$$f(x^*) = \sum_{i=1}^{M'} \alpha_i' K(z_i, x^*) + b'.$$

We use $f(x^*)$ for regression and sign$[f(x^*)]$ for classification. Table 2.2 provides a conceptual and notational overview of the steps involved in SSD-LSSVM model.

	Initial		1st **Reduction**		2nd **Reduction**
SV/Train	N/N		M/M		M'/N
Dual	α, $\mathcal{D}$	**Step 1** $\rightarrow$	$\bar{\alpha}$, $\mathcal{S}_{PV}$	**Step 2** $\rightarrow$	α', b', $\mathcal{S}_{SV}$
	$\phi(x) \in \mathbb{R}^{n_h}$	**SD-LSSVM**	$\phi(x) \in \mathbb{R}^{n_h}$	**SSD-LSSVM**	

Table 2.2: Given the dataset $\mathcal{D}$ we first perform the SD-LSSVM as a fast initialization in Step 1. We obtain the dual Lagrange variables $\bar{\alpha}_i$, $i = 1, \ldots, M$. In Step 2 we perform the SSD-LSSVM, i.e., we use an iterative sparsifying L_0-norm procedure in the primal resulting in a reduced set of vectors $\mathcal{S}_{SV}$. We construct the rectangular matrix Q which incorporates information about the entire training set. After the 2nd reduction we select the non-zero $\bar{\alpha}_i$ and $\bar{b}$ to obtain the solution vector α' and b'.

Figure 2.1 and 2.2 compare the proposed SPFS-LSSVM method and SSD-LSSVM method with the normal PFS-LSSVM method for classification on the Ripley dataset and for regression on the Boston Housing data. For the Boston Housing dataset we display the projection of all the data points on the first eigenvector (for visualization purpose) along the *x-axis* while the estimator value is plotted along the *y-axis* since the dataset has dimensions $d > 3$. From Figure 2.1, we can observe that the PFS-LSSVM method results in a better decision boundary with lower prediction error. However, the cardinality of the SV set is much higher in comparison with the SPFS-LSSVM and SSD-LSSVM methods. The proposed approaches result in a much sparser model without any significant trade-off in error and have good decision boundaries.

From Figure 2.2, we observe that the SPFS-LSSVM method results in better prediction errors than the SSD-LSSVM using a fewer number of prototype vectors. In the SSD-LSSVM method during the training phase, we train just over the SV set, in comparison to the SPFS-LSSVM technique where we train over the entire training set. So, we might end up with an under-determined model in the SSD-LSSVM case as less information is incorporated in the model. Figure 2.2 also shows that the proposed methods select points with high and small predictor values as prototype vectors for the set $\mathcal{S}_{SV}$. However, the

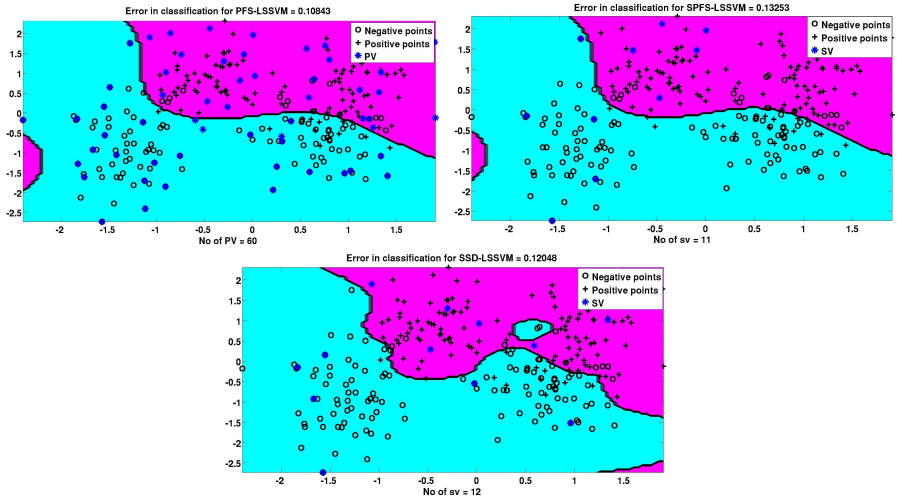


Figure 2.1: Comparison of best results out of 10 randomizations for the PFS-LSSVM method with the proposed approaches for the Ripley dataset.

difference in errors between the proposed approaches is not very significant and when compared to the PFS-LSSVM method, the trade-off in error with respect to the amount of sparsity gained is not significant.

2.1.4 Computational Complexity & Experimental Results

The *convergence* of Algorithm 3 and 4 is assumed when the difference $\parallel \tilde{\beta}^t - \tilde{\beta}^{t+1} \parallel /M$ and $\parallel \bar{\alpha}^t - \bar{\alpha}^{t+1} \parallel /M$ respectively is lower than 10^{-4} or when the number of iterations t exceeds 50. The result of the two approaches is the *indices* of those SVs for which $|\tilde{\beta}_i| > 10^{-6}$ and $|\bar{\alpha}_i| > 10^{-6}$ which provides us the reduced set of prototype vectors $\mathcal{S}_{SV}$. We perform an analysis of the computation time required for the proposed approaches and are futher described.

Computational Complexity

The computation time of the PFS-LSSVM method involves:

- Solving a linear system of size $M+1$ where M is the number of prototype vectors selected initially (PV).

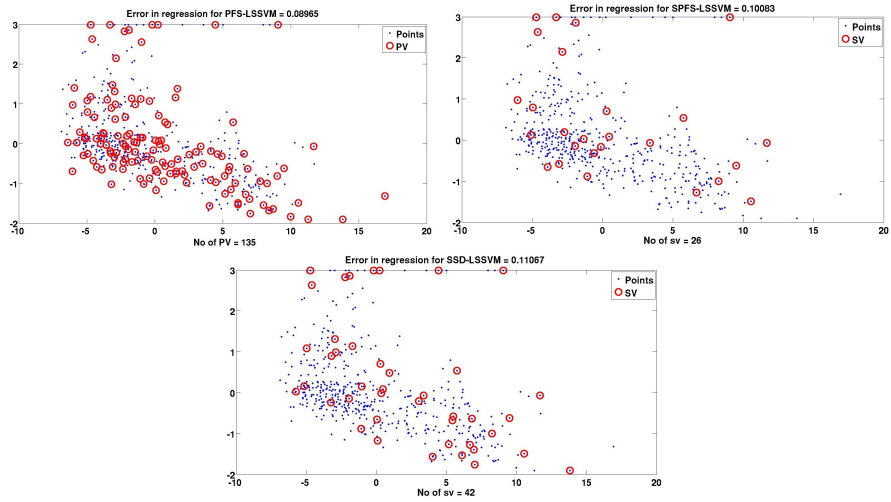


Figure 2.2: Comparison of best results out of 10 randomizations for the PFS-LSSVM method with the proposed approaches for the Boston Housing dataset projected on the first eigenvector as the dimensions of the dataset ($d > 3$). We use 337 training points whose projections are plotted for all the methods. This projection is only for visualization purposes.

- Calculating the Nyström approximation and eigenvalue decomposition of the kernel matrix of size M once.
- Forming the matrix product $\Phi^\intercal\Phi$.

This computation time is $O(NM^2)$ where N is the dataset size as shown in [30].

The computation time for the SPFS-LSSVM method comprises a v-fold cross-validation time ($O(vNM^2)$), time for matrix multiplication $\hat{Q}^\intercal\hat{Q}$ ($O(NM^2)$) once and iteratively solving a system of linear equations whose complexity is ($O(M^3)$). Since $M \ll N$, the overall complexity of SPFS-LSSVM is $O((v+1)NM^2)$.

The computation time for the SD-LSSVM method of training on M prototype vectors is given by the time required to solve a system of linear equations with M variables and is equivalent to $O(M^3)$. For the proposed SSD-LSSVM approach the computation time is $O(NM^2)$ for the matrix multiplication $Q^\intercal Q$ once and then iteratively solving a system of M linear equations ($O(M^3)$). Thus the overall time complexity of SSD-LSSVM method is much less in comparison with the SPFS-LSSVM technique. The major part of the computation time is

required for cross-validation to obtain the tuning parameters γ and σ for the proposed approaches.

Since there is no widely accepted approach for selecting an initial model selection value M, in our experiments we selected $M = \lceil k \times \sqrt{N} \rceil$ where $k \in \mathbb{N}$, the complexity of SPFS-LSSVM can be re-written as $O(k^2N^2)$. The parameter k is a user-defined parameter and is not a tuning parameter. However, k should be chosen carefully such that the $N \times M$ matrices can fit into memory. For smaller datasets like Ripley, Breast-Cancer, Diabetes, Spambase, Magic Gamma, Boston Housing, Concrete, Slice Localization and Adult, we experimented with different values of k. For each value of k, we obtained the optimal tuning parameters (σ and γ) along with the prediction errors. In Table 2.3, we report the value of k corresponding to which we get the lowest classification and regression errors for these smaller datasets.

Dataset Description

All the datasets have been obtained from the UCI benchmark repository [73]. A brief description about the datasets is given in Table 2.3.

Classification					
Dataset	N	Dims	N_{test}	PV	k
Ripley(RIP)	250	2	84	60	4
Breast-Cancer(BC)	682	10	227	155	6
Diabetes(DIB)	768	8	256	167	6
Spambase(SPAM)	4061	57	1354	204	3
Magic Gamma(MGT)	19020	11	6340	414	3
Adult(ADU)	48842	14	16281	664	3
Forest Cover(FC)	531012	54	177004	763	1
Regression					
Dataset	N	Dims	N_{test}	PVs	k
Boston Housing(BH)	506	14	169	135	6
Concrete(Con)	1030	9	344	192	6
Slice Localization(SLC)	53500	385	17834	694	3
Year Prediction(YP)	515345	90	171780	718	1

Table 2.3: Classification & Regression Data Description and initial number of prototype vectors (PV) selected such that $M = \lceil k \times \sqrt{N} \rceil$.

Numerical Experiments

All experiments are performed on a PC machine with Intel Core i7 CPU and 8 GB RAM under Matlab 2008a. We use the RBF-kernel for the kernel matrix construction in all cases. As a pre-processing step, all records containing

unknown values are removed from consideration. All given inputs are normalized to zero mean and unit variance.

We compare the performance of our proposed approaches with methods including normal PFS-LSSVM classifier/regressor, SV_L_0-norm PFS-LSSVM proposed in [90], C-SVM and ν-SVM, L_0-norm method of Lopez [89] and Keerthi's method [31] for classification. The latter SVM and ν-SVM methods are implemented in the LIBSVM software. All methods use a cache size of 8 GB. Shrinking is applied in the SVM case. All comparisons are made on the same 10 randomizations of the methods. The SV_L_0-norm PFS-LSSVM method tries to sparsify the PFS-LSSVM solution by an iterative sparsification procedure but its loss function only incorporates information about the set of PV vectors (M) while performing this operation. Thus, it results in solutions having more variations in error and more variations in the number of reduced prototype vectors (SV). Details of the method are provided in [90]. The Lopez method cannot scale to large scale data. Keerthi's method greedily finds a set of basis functions of a specified size using a Newton optimization technique. However, if the number of iterations required for the Newton method to converge increases, the time complexity increases and becomes even worse than the time required for the SVM methods as shown in Table 2.4.

For all the approaches, we use the method of coupled simulated annealing (CSA) as described in [93] to obtain the optimal tuning parameters namely γ, σ for PFS-LSSVM, LSSVM, SVM, Lopez, Keerthi, SV_L_0-norm PFS-LSSVM, SPFS-LSSVM and SSD-LSSVM methods. We start by using 5 multiple random starters for each tuning parameter. For each combination of tuning parameters we evaluate the cost. The cost is defined as the accuracy in the case of classification and mean squared error (mse) in the case of regression. This cost is obtained by performing one iteration of 10-fold cross-validation of the corresponding method. These costs along with the combination of parameters are provided to CSA to obtain the optimal tuning parameters as illustrated in [93]. We fixed the ν parameter in ν-SVM to a value of 0.5 because if we tuned for ν then the method becomes computationally very expensive. Thus, the number of 10-fold cross-validations performed for PFS-LSSVM, LSSVM, SVM, Lopez, Keerthi and the two proposed approaches is 25 (5×5) for each randomization of these methods. The time reported in Table 2.4 includes the time required for performing all these cross-validations.

All comparisons are performed on an out-of-sample test set depicted as N_{test} in Table 2.3 consisting of 1/3 of the data. The first 2/3 of the data is reserved for training and cross-validation. Several techniques [94, 95, 96] use cross-validation for estimating the model parameters as it optimizes the bias-variance trade-off. For each algorithm, the average set performances and sample standard deviations on the same 10 randomizations are reported. Also the mean total

time (training, cross-validation and testing) and the corresponding standard deviation, the mean number of prototype vectors for each method is depicted in Table 2.4.

Table 2.4 provides a comparison of the mean estimated error, mean value of cardinality of prototype vectors (PV or SV, denoted in Table 2.4 by SV) and a comparison of the mean run time computations of the proposed approaches with Keerthi's, Lopez method, PFS-LSSVM and SVM methods for various classification and PFS-LSSVM and SVR methods for different regression datasets. Figure 2.3 represents the estimated error, run time performance and variations in the number of prototype vectors for Adult dataset (ADU). From Figure 2.3, we observe that Keerthi's method result in best prediction errors but requires the maximum computation time as well. The variations in the prediction errors by the proposed SPFS-LSSVM and SSD-LSSVM method are insignificant and their estimated errors are comparable to that of the PFS-LSSVM method though they require a much smaller prototype vectors set. An important observation was that the SSD-LSSVM method produces the maximum amount of sparsity and has the least computation time without significant trade-off in error and thus making it highly suitable for very large scale datasets. The SSD-LSSVM method works well because the initial set of prototype vectors (PV) that it selects is only helping to construct a basis set where this basis set is maximizing the quadratic Rènyi entropy criterion. We also observe that the number of reduced prototype vectors (SV) can vary a lot for L_0-norm based methods as during each randomization we start with a different initialization and after performing the iterative sparsification procedure we end up in a different local minimum. This results in variations in the number of reduced support vectors (SV) for these methods.

Performance Analysis

From Table 2.4, we observe that the variations in prediction errors is not much for all of these methods. This suggests that none of the methods over-fit for a particular randomization and the cross-validation errors are nearly similar for each randomization of every algorithm.

In case of classification datasets, the proposed approaches called SPFS-LSSVM and SSD-LSSVM work much better in comparison with SVM methods both in terms of sparsity and prediction errors as observed from Table 2.4. Their errors are comparable to those of PFS-LSSVM and Keerthi's method and in the case of the Forest Cover dataset even better than PFS-LSSVM method. The amount of sparsity introduced is quite high for the proposed approaches, which is consistent with the fact that the L_0-norm leads to highly sparse solutions.

Test Classification(Error and Mean SV)																
	RIP		BC		DIB		SPAM		MGT		ADU		FC	
Algorithm	Error	SV	Error	SV	Error	SV	Error	SV	Error	SV	Error	SV	Error	SV
PFS-LSSVM [30]	**0.102 ± 0.008**	58	0.023 ± 0.004	153	**0.21 ± 0.008**	167	0.08 ± 0.002	204	**0.134 ± 0.001**	414	0.147 ± 0.0	664	0.2048 ± 0.026	763
C-SVC [25]	0.15 ± 0.07	81	0.0348 ± 0.02	414	0.332 ± 0.02	509	0.075 ± 0.067	800	0.144 ± 0.015	7000	0.151(∗)	11085(∗)	0.185(∗)	185000(∗)
ν-SVC [25]	0.167 ± 0.03	112	0.028 ± 0.011	398	0.338 ± 0.017	512	0.113 ± 0.07	1525	0.158 ± 0.014	7252	0.161(∗)	12205(∗)	184(∗)	165205(∗)
Lopez [89]	0.183 ± 0.1	7	0.04 ± 0.011	17	0.248 ± 0.034	10	0.0726 ± 0.005	340	-	-	-	-	-	-
Keerthi [31]	0.129 ± 0.033	26	0.0463 ± 0.037	30	0.228 ± 0.035	53	**0.07 ± 0.005**	175	0.135 ± 0.002	260	**0.145 ± 0.002**	494	0.2054(*)	762(*)
SV_L_0-norm [90]	0.1651 ± 0.13	15	0.023 ± 0.01	27	0.27 ± 0.05	39	0.127 ± 0.09	152	0.15 ± 0.009	75	0.149 ± 0.001	236	0.2346 ± 0.025	352
SPFS-LSSVM	0.147 ± 0.03	11	**0.021 ± 0.01**	26	0.273 ± 0.04	8	0.082 ± 0.002	169	0.141 ± 0.004	163	0.148 ± 0.0	464	0.2195 ± 0.03	505
SSD-LSSVM	0.13 ± 0.02	11	0.0256 ± 0.007	8	0.227 ± 0.02	6	0.0836 ± 0.005	63	0.135 ± 0.0	280	0.1488 ± 0.001	102	**0.1905 ± 0.009**	635

Test Regression(Error and Mean SV)								
	Boston Housing		Concrete		Slice Localization		Year Prediction	
Algorithm	Error	SV	Error	SV	Error	SV	Error	SV
PFS-LSSVM [30]	**0.133 ± 0.002**	113	**0.112 ± 0.006**	193	**0.0554 ± 0.0**	694	**0.402 ± 0.034**	718
ϵ-SVR [25]	0.16 ± 0.05	226	0.23 ± 0.02	670	0.102(*)	13012(*)	0.465(*)	192420(*)
ν-SVR [25]	0.16 ± 0.04	195	0.22 ± 0.02	330	0.092(*)	12524	0.472(*)	175250(*)
Lopez [89]	0.16 ± 0.05	65	0.165 ± 0.07	215	-	-	-	-
SV_L_0-norm [90]	0.192 ± 0.015	38	0.265 ± 0.02	25	0.156 ± 0.13	381	0.495 ± 0.03	422
SPFS-LSSVM	0.176 ± 0.03	50	0.2383 ± 0.04	46	0.058 ± 0.002	651	0.44 ± 0.05	595
SSD-LSSVM	0.189 ± 0.022	43	0.1645 ± 0.017	54	0.056 ± 0.0	577	0.42 ± 0.01	688

Train, Cross-validation & Test Classification(Computation Time)							
Algorithm	RIP	BC	DIB	SPAM	MGT	ADU	FC
PFS-LSSVM [30]	3.16 ± 0.18	27.36 ± 1.33	36.14 ± 2.4	182 ± 7	3142 ± 61	16523 ± 540	118500 ± 9294
C-SVC [25]	4.6 ± 0.83	14.76 ± 1	61 ± 75	1010 ± 53	20603 ± 396	13973(*)	58962(*)
ν-SVC [25]	5.67 ± 11	14.91 ± 1	83.5 ± 115	785 ± 22	13901 ± 189	13992.7(*)	53478(*)
Lopez [89]	5.12 ± 0.9	28.2 ± 5.4	39.1 ± 5.9	950 ± 78.5	-	-	-
Keerthi [31]	21.1 ± 0.6	74.34 ± 3.9	86.7 ± 5.12	1070 ± 37.7	16608 ± 397	20359 ± 762	9149.8(*)
SV_L_0-norm [90]	3.19 ± 0.16	27.384 ± 1.32	36.18 ± 2.4	182.2 ± 7	3143 ± 61	16527 ± 539	118540 ± 9267.9
SPFS-LSSVM	3.21 ± 0.16	27.42 ± 1.33	36.266 ± 2.4	182.3 ± 7	3155 ± 61	16588 ± 529	118570 ± 9247.6
SSD-LSSVM	**0.955 ± 0.05**	**8.67 ± 0.14**	**11.82 ± 0.14**	**81.5 ± 0.6**	**2192 ± 1.9**	**13482 ± 15.6**	**86460 ± 2.8**

Train, Cross-validation & Test Regression(Computational Time)				
Algorithm	Boston Housing	Concrete	Slice Localization	Year Prediction
PFS-LSSVM [30]	14.28 ± 0.2	58.41 ± 1.41	33769 ± 1571	248046 ± 15330
ϵ-SVR [25]	63 ± 1	168 ± 3	52438(*)	67231(*)
ν-SVR [25]	61 ± 1	131 ± 2	42724(*)	59899(*)
Lopez [89]	158.6 ± 3.2	753 ± 35.5	-	-
SV_L_0-norm [90]	14.58 ± 0.2	58.6 ± 1.48	33775 ± 1574	248178 ± 15262
SPFS-LSSVM	14.5 ± 0.2	58.76 ± 1.44	33790 ± 1571	248412 ± 15248
SSD-LSSVM	**3.75 ± 0.06**	**21.35 ± 0.137**	**25991 ± 11**	**86652 ± 24**

Table 2.4: Comparison in performance of the different methods for UCI repository datasets. The (*) represents no cross-validation and performance on a fixed value of tuning parameters due to computational burden and (-) represents the method cannot scale for this dataset.

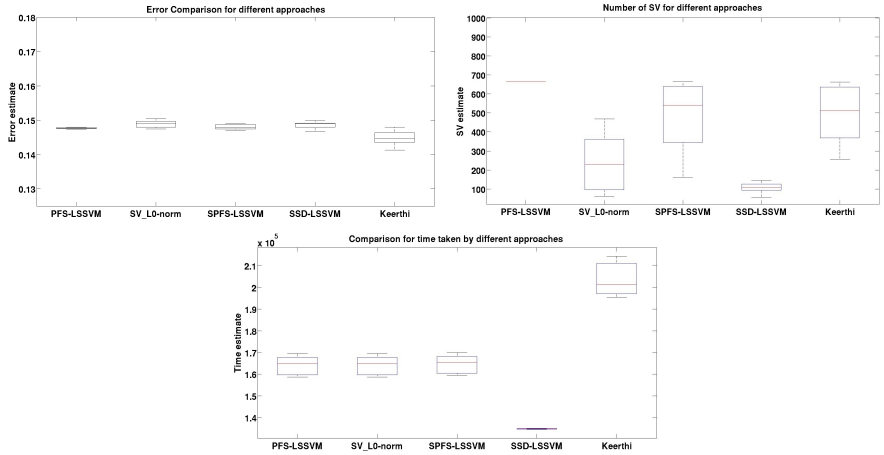


Figure 2.3: Comparison of the proposed approaches with PFS-LSSVM, SV_L_0-norm PFS-LSSVM and Keerthi's method for the Adult dataset.

However, an observation shows that the number of prototype vectors is reduced to a maximum extent in most of the datasets for the SSD-LSSVM approach. The SPFS-LSSVM method results in best prediction error for Breast-Cancer(BC) dataset while the SSD-LSSVM method gives best results for the very large scale FC dataset as highlighted in Table 2.4. The amount of sparsity introduced varies for different datasets. For example, for the BC dataset the PFS-LSSVM method uses 33.65%, Keerthi's method uses 6.59%, SPFS-LSSVM and SSD-LSSVM method use 5.71% and 1.76% of the training data respectively as SV without significant trade-off in errors.

From Figure 2.4, we observe the performance for the Forest Cover (FC) dataset. PFS-LSSVM uses 0.22% while SV_L_0-norm PFS-LSSVM uses 0.1%, SPFS-LSSVM uses 0.14% and SSD-LSSVM method uses 0.18% of the training data as SV. This suggests that for the FC dataset the amount of sparsity achieved by the proposed approaches is much less in comparison to that for the BC dataset. The SSD-LSSVM results have less variations while the SV_L_0-norm PFS-LSSVM method has maximum variance in error predictions which can be attributed to the lack of information in the loss function that the technique was incorporating while performing the iterative sparsification procedure. This results in more variations in the number of SV obtained for the 10 randomizations. The computation time for PFS-LSSVM, SV_L_0-norm PFS-LSSVM, SPFS-LSSVM are higher in comparison to that of SSD-LSSVM which follows the line of the reasoning provided in Section 2.1.4. Another important observation is that since SPFS-LSSVM and SSD-LSSVM result in much fewer number of prototype

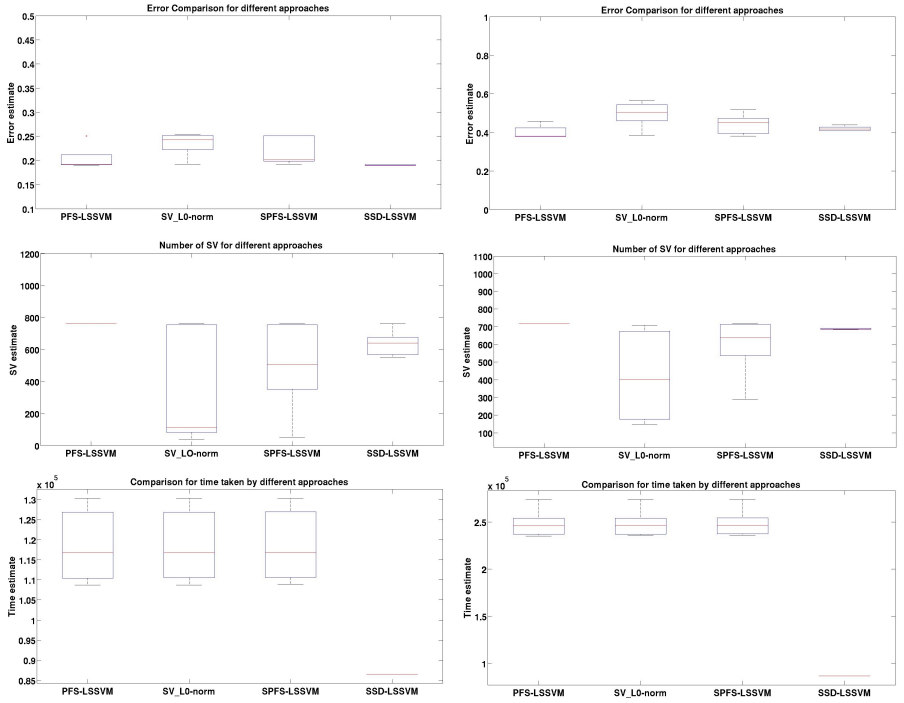


Figure 2.4: Comparison of performance of the proposed approaches with PFS-LSSVM, SV_L_0-norm PFS-LSSVM method for FC & Year Prediction datasets.

vectors SV, the time required for out-of-sample predictions (testing time) for these methods is much less in comparison to the other methods.

For datasets like Boston Housing and Concrete, the estimated error by the proposed methods is larger than for the PFS-LSSVM method, but the amount of sparsity introduced is quite significant. This is because regression in general requires a large number of support vectors as observed for the different SVR approaches in Table 2.4. From Table 2.4, we observe that for the proposed approaches the higher the mean value of SV, the lower is the estimated error. An exception is the case of Slice Localization dataset which can be attributed to the non-parametric nature of SSD-LSSVM as we are not using an explicit feature map in this method and thus information about all the features (dimensions) are intact. From Figure 2.4, we observe the performance of the various approaches for the large scale Year Prediction dataset. The PFS-LSSVM method has the least error estimates, but the SSD-LSSVM approach results in comparable prediction errors with a much lower computation cost. We also observe that the

number of reduced prototype vectors (SV) can vary a lot for the L_0-norm based methods as during each randomization, we start with a different initialization and after performing the iterative sparsification procedure, we end up in a different local minimum.

2.1.5 Sparsity versus Error Trade-off

In this section, we perform additional experiments to illustrate the sparsity versus error trade-off between PFS-LSSVM and SPFS-LSSVM techniques and between SD-LSSVM and SSD-LSSVM methods.

Additional Experiments

Table 2.5 provides a comparison between the error variations and the mean number of prototype vectors for PFS-LSSVM and SPFS-LSSVM and SD-LSSVM and SSD-LSSVM techniques. For PFS-LSSVM and SD-LSSVM methods, we report the value of M as the number of prototype vectors PV. For the proposed techniques we report the mean value of M' as the number of reduced prototype vectors SV. From Table 2.5, we observe that the PFS-LSSVM performs the best with respect to error variations but the amount of sparsity introduced by the proposed techniques is quite significant without much trade-off in error. An interesting observation is that the SD-LSSVM method results in poor prediction errors in comparison with the SSD-LSSVM method for most of the datasets. This is because the SD-LSSVM works with only M PV vectors while training, cross-validating and testing. But the SSD-LSSVM uses the information about the entire training set N in its loss function while performing the iterative sparsification procedure. Thus, it results in a set of prototype vectors SV which improves prediction errors particularly for large scale datasets like Adult, Forest Covertype for classification, Slice Localization and Year Prediction datasets for regression.

Trade-off Analysis

Table 2.6 illustrates the sparsity versus error trade-off for the proposed L_0-norm based reductions to PFS-LSSVM and SD-LSSVM methods for various classification and regression datasets of different sizes.

Sparsity is calculated as the fraction between the change in the number of prototype vectors to the total number of prototype vectors, i.e., $\frac{|\mathcal{S}_{PV}|-|\mathcal{S}_{SV}|}{|\mathcal{S}_{PV}|}$.

Classification								
	Intial Step		Final Step		Initial Step		Final Step	
Dataset	PFS-LSSVM		SPFS-LSSVM		SD-LSSVM		SSD-LSSVM	
	Error	PV	Error	SV	Error	PV	Error	SV
RIP	0.1 ± 0.01	58	0.15 ± 0.03	11	0.12 ± 0.03	58	0.13 ± 0.02	11
BC	0.02 ± 0.0	153	0.02 ± 0.01	26	0.02 ± 0.03	153	0.03 ± 0.01	8
DIB	0.2 ± 0.01	167	0.27 ± 0.04	8	0.22 ± 0.01	167	0.23 ± 0.02	6
SPAM	0.08 ± 0.0	204	0.082 ± 0.0	169	0.15 ± 0.01	204	0.083 ± 0.01	63
MGT	0.13 ± 0.0	414	0.14 ± 0.0	163	0.19 ± 0.01	414	0.135 ± 0.0	280
ADU	0.15 ± 0.0	664	0.15 ± 0.0	464	0.16 ± 0.0	664	0.15 ± 0.0	102
FC	0.20 ± 0.03	763	0.22 ± 0.03	505	0.26 ± 0.0	763	0.19 ± 0.01	635
Regression								
	Intial Step		Final Step		Initial Step		Final Step	
Dataset	PFS-LSSVM		SPFS-LSSVM		SD-LSSVM		SSD-LSSVM	
	Error	PV	Error	SV	Error	PV	Error	SV
BH	0.13 ± 0.0	113	0.18 ± 0.03	50	0.12 ± 0.0	113	0.19 ± 0.02	43
Con	0.11 ± 0.0	193	0.24 ± 0.04	46	0.16 ± 0.2	193	0.16 ± 0.02	54
SLC	0.06 ± 0.0	694	0.06 ± 0.0	651	0.08 ± 0.0	694	0.06 ± 0.0	577
YP	0.4 ± 0.03	718	0.44 ± 0.05	595	0.49 ± 0.0	718	0.42 ± 0.0	688

Table 2.5: Error & mean SV comparisons for PFS-LSSVM (Initial) and SPFS-LSSVM (After Sparsification), SD-LSSVM (Initial) and SSD-LSSVM (After Sparsification).

Classification				
	Intial Step	Final Step	Initial Step	Final Step
Dataset	PFS-LSSVM & SPFS-LSSVM		SD-LSSVM & SSD-LSSVM	
	Sparsity	Fractional Change in Error	Sparsity	Fractional Change in Error
RIP	0.81	0.50	0.81	0.08
BC	0.63	0.00	0.95	0.50
DIB	0.95	0.35	0.96	0.05
SPAM	0.17	0.03	0.69	**-0.45**
MGT	0.61	0.08	0.32	**-0.29**
ADU	0.30	**0.00**	0.85	**-0.06**
FC	0.34	0.10	0.17	**-0.27**
Regression				
	Intial Step	Final Step	Initial Step	Final Step
Dataset	PFS-LSSVM & SPFS-LSSVM		SD-LSSVM & SSD-LSSVM	
	Sparsity	Change in Error	Sparsity	Change in Error
BH	0.56	0.05	0.62	0.07
Con	0.76	0.13	0.72	**0.00**
SLC	0.07	**0.00**	0.17	**-0.02**
YP	0.17	0.04	0.04	**-0.07**

Table 2.6: Sparsity versus Error trade-off for PFS-LSSVM (Initial) and SPFS-LSSVM (After Sparsification), SD-LSSVM (Initial) and SSD-LSSVM (After Sparsification). Negative change in error means that by introducing sparsity the performance of the model has actually improved or remained unchanged. These results are highlighted. This is explained in detail in Section 2.1.5.

We use the metric fractional change in error for classification. For the PFS-LSSVM and SPFS-LSSVM it is defined as $(err_{SPFS} - err_{PFS})/err_{PFS}$ where

err_{SPFS} is the error corresponding to SPFS-LSSVM and err_{PFS} is the error corresponding to PFS-LSSVM. Similarly, for the SD-LSSVM and SSD-LSSVM it is defined as $(err_{SSD} - err_{SD})/err_{SD}$ where err_{SSD} is the error corresponding to SSD-LSSVM and err_{SD} is the error corresponding to SD-LSSVM. For regression we report the mean squared error (mse). It is sufficient to investigate the change in error, i.e., $err_{SPFS} - err_{PFS}$ and $err_{SSD} - err_{SD}$ w.r.t. the sparsity introduced by the proposed approaches.

From Table 2.6 we observe that for some datasets the fractional change in error or the change in error takes negative values. This means that by introducing sparsity the performance of the model has improved. This is seen only for the case of the SD-LSSVM and SSD-LSSVM methods. The reason for this improvement is that the subsampled dual LSSVM (SD-LSSVM) model is built by incorporating information about only the M prototype vectors selected by maximizing the quadratic Rènyi entropy. On the other hand for the sparsified subsampled dual LSSVM (SSD-LSSVM) during the sparsification process we incorporate the information about the entire training set (N data points) and thus it results in a better predictive model. This method works particularly well for large scale datasets as can be observed from the results corresponding to SPAM, MGT, ADU, FC, SLC and YP datasets. In case of classification, the sparsity introduced for these datasets by the SD-LSSVM and SSD-LSSVM methods is quite substantial as shown in Table 2.6.

For the PFS-LSSVM and SPFS-LSSVM methods, if the amount of sparsity increases then the change in error also increases. This is reflected for datasets like RIP, DIB and Con in Table 2.6. However, for large scale datasets like MGT, ADU, FC, SLC and YP a considerable amount of sparsity is introduced without much change in error estimation of the model as highlighted in Table 2.6.

The PFS-LSSVM can scale to large datasets but the resulting models are not the sparsest due to the problem of selection of smallest cardinality value (M) for the PV set. We overcome this problem by the means of the proposed SPFS-LSSVM. The LSSVM model in general cannot scale for large scale datasets as it creates and $N \times N$ kernel matrix. To overcome this problem we introduced the SD-LSSVM which trains on the smaller PV set. We further incorporate the information about large training set (N) while performing the L_0-norm on top of it. This results in the SSD-LSSVM. By doing this we prevent loss of information in the final model. Thus, the two levels of sparsity allows to scale to large scale datasets while having very sparse models.

2.1.6 Conclusion

In this work, we proposed two techniques namely SPFS-LSSVM and SSD-LSSVM resulting in very sparse solutions for mining large scale datasets. The problem of selection of smallest cardinality M for the PV set faced by the PFS-LSSVM method is solved by using an iterative sparsifying L_0-norm procedure resulting in very sparse solutions. The computation time required by one of the proposed approaches (SSD-LSSVM) is quite low due to lower training, cross-validation and most importantly out-of-sample extension time because of fewer prototype vectors. We compared the error predictions of proposed approaches with the recent approaches like Keerthi's method, SV_L_0-norm PFS-LSSVM, Lopez method and traditional approaches like ν-SVM and C-SVM with good performance and sparser models. We also investigated the trade-off between sparsity versus performance and showed that a significant amount of sparsity can be obtained without much change in error.

2.2 Sparse Reductions to FS-LSSVM[2]

Abstract—FS-LSSVM is a powerful tool for solving large scale classification and regression problems. FS-LSSVM solves an over-determined system of M linear equations by using Nyström approximations on a set of prototype vectors (PVs) in the primal. This introduces sparsity in the model along with ability to scale for large datasets. But there exists no formal method for selection of the right value of M. In this work, we investigate the sparsity-error trade-off by introducing a second level of sparsity after performing one iteration of FS-LSSVM. This helps to overcome the problem of selecting a right number of initial PVs as the final model is highly sparse and dependent on only a few appropriately selected prototype vectors (SV) is a subset of the PVs. The first proposed method performs an iterative approximation of L_0-norm which acts as a regularizer. The second method belongs to the category of threshold methods, where we set a window and select the SV set from correctly classified PVs closer and farther from the decision boundaries in the case of classification. For regression, we obtain the SV set by selecting the PVs with least minimum squared error (mse). Experiments on real world datasets from the UCI repository illustrate that highly sparse models are obtained without significant trade-off in error estimations scalable to large scale datasets.

[2]This chapter contains Sections 1−5 from Mall R., Suykens J.A.K., **"Sparse Reductions for Fixed-Size Least Squares Support Vector Machines on Large Scale Data"**, in Proc. of the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2013), GoldCoast, Australia, Apr. 2013, pp. 161-173.

2.2.1 Introduction

The L_0-norm counts the number of non-zero elements of a vector. It results in very sparse models by returning models of low complexity and acts as a regularizer. However, obtaining this L_0-norm is an NP-hard problem. Several approximations to it are discussed in [66]. In this work, we modify the iterative sparsifying procedure introduced in [80] and used for LSSVM the technique as shown in [89] and reformulate it for the FS-LSSVM. We apply this formulation on FS-LSSVM because for large scale datasets like Magic Gamma, Adult and Slice Localization ([73]) we are overwhelmed with memory $O(N^2)$ and computational time $O(N^3)$ constraints when applying the L_0-norm scheme directly on LSSVM [89] or SVM [80]. The second proposed method performs an iteration of FS-LSSVM and then based on a user-defined window selects a subset of PVs as SV. For classification, the selected vectors satisfy the property of being correctly classified and are either closer or farther from the decision boundary since they well determine the extent of the classes. But for regression, the SV set comprises those PVs which have the least mse and are best-fitted by the regressor. Once the SV set is determined we re-perform FS-LSSVM resulting in highly sparse models without significant trade-off in accuracy and scalable to large scale datasets.

The contribution of this work involves providing smaller solutions which use $M' < M$ PVs for FS-LSSVM, obtaining highly sparse models with guarantees of low complexity (L_0-norm of $\tilde{w}$) and overcoming the problem of memory and computational constraints faced by L_0-norm based approaches for LSSVM and SVM on large scale datasets. Sparseness enables exploiting memory and computationally efficiency, e.g. matrix multiplications and inversions. The solutions that we propose utilize the best of both FS-LSSVM and sparsity inducing measures on LSSVM and SVM resulting in highly sparse and scalable models. Table 2.7 provides a conceptual overview of LSSVM, FS-LSSVM and proposes SV_L_0-norm Primal FS-LSSVM along with the notations used in the rest of this work. Figures 2.5 and 2.6 illustrate our proposed approaches on the Ripley and Motorcycle dataset.

Algorithm 5: Fixed-Size LSSVM method

Data: $\mathcal{D} = \{(x_i, y_i) : x_i \in \mathbb{R}^d, y_i \in \{+1, -1\}$ for classification & $y_i \in \mathbb{R}$ for regression, $i = 1, \ldots, N\}$.

1 Determine the kernel bandwidth using the multivariate rule-of-thumb [97].
2 Given the number of PVs, perform prototype vector selection using quadratic Rènyi entropy criterion.
3 Determine the tuning parameters σ and γ performing fast v-fold cross validation as described in [30].
4 Given the optimal tuning parameters, get FS-LSSVM parameters $\tilde{w}$ & $\tilde{b}$.

	LSSVM		**FS-LSSVM**		SV_L_0**-norm PFS-LSSVM**
SV/Train	N/N		M/N		M'/M
Primal	w		$\tilde{w}$		$\boxed{w'}$
	$\phi(\cdot) \in \mathbb{R}^{N_h}$	Step 1	$\tilde{\phi}(\cdot) \in \mathbb{R}^{M}$	Step 2	$\boxed{\phi'(\cdot) \in \mathbb{R}^{M'}}$
Dual	α	$\rightarrow$	$\boxed{\tilde{\alpha}}$	$\rightarrow$	α'
	$K \in \mathbb{R}^{N \times N}$		$\boxed{\tilde{K} \in \mathbb{R}^{M \times M}}$		$K' \in \mathbb{R}^{M' \times M'}$

Table 2.7: For SV_L_0-norm PFS-LSSVM, we first perform FS-LSSVM in the Primal. Then, a sparsifying procedure is performed in the Dual of FS-LSSVM as highlighted in the box in the middle resulting in a reduced SV set. Then FS-LSSVM is re-performed in the Primal as highlighted by the box on the right. We propose two sparsifying procedures namely SV_L_0-norm PFS-LSSVM and Window reduced FS-LSSVM.

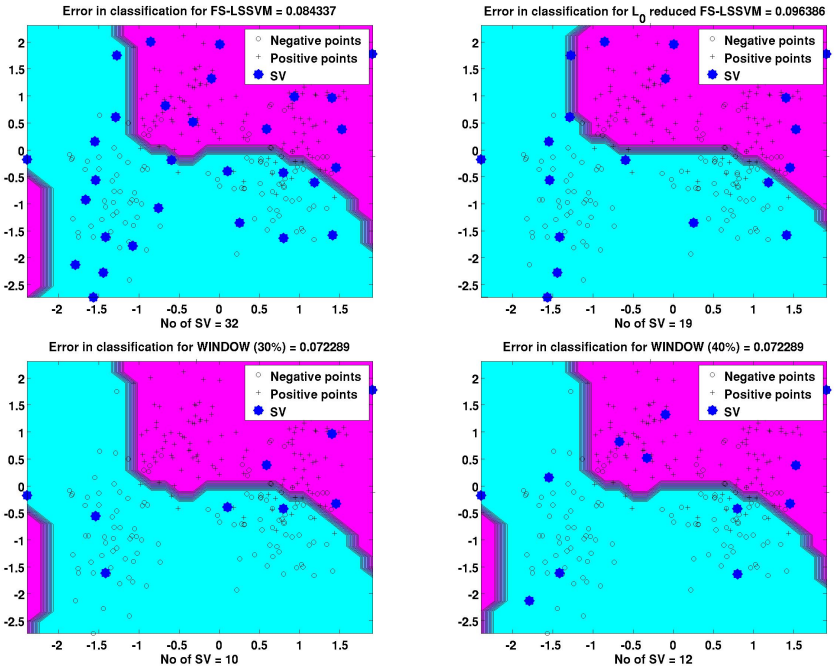


Figure 2.5: Comparison of the best randomization result out of 10 randomizations for the proposed methods with FS-LSSVM for Ripley Classification data

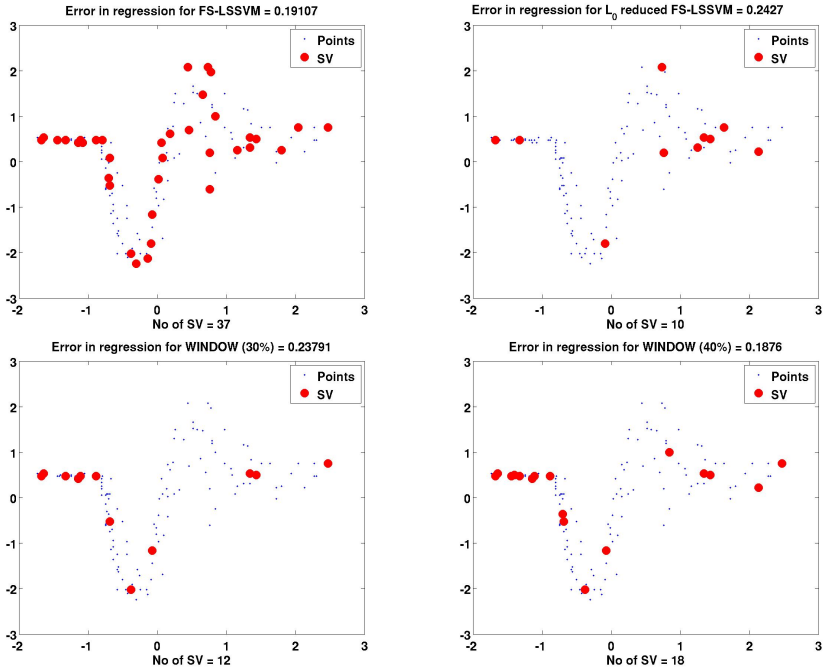


Figure 2.6: Comparison of the best randomization result out of 10 randomizations for the proposed methods with FS-LSSVM for Motorcycle Regression data

2.2.2 SV_L_0-norm PFS-LSSVM

Algorithm 5 gives a brief summary of the FS-LSSVM method. We first propose an approach using the L_0-norm to reduce $\tilde{w}$ and acting as a regularizer in the objective function. It tries to estimate the optimal subset of PVs leading to sparse solutions. For our formulation, the objective function is to minimize the error estimations of these prototype vectors regulated by L_0-norm of $\tilde{w}$. We modify the procedure described in [80], [89] and consider the following generalized primal problem:

$$\begin{aligned} \min_{\tilde{\alpha},\tilde{b},\tilde{e}} \quad & J(\tilde{\alpha},\tilde{e}) = \frac{1}{2}\sum_j \lambda_j \tilde{\alpha}_j^2 + \frac{\gamma}{2M}\sum_i \tilde{e}_i^2 \\ \text{s.t.} \quad & \sum_j \tilde{\alpha}_j \tilde{K}_{ij} + \tilde{b} = y_i - \tilde{e}_i,\ i = 1,\ldots,M \end{aligned} \tag{2.14}$$

where $\tilde{w} \in \mathbb{R}^M$ and can be written as $\tilde{w} = \sum_j \tilde{\alpha}_j \tilde{\phi}(x_j)$. The regularization term is now not on $\| \tilde{w} \|^2$ but on $\| \tilde{\alpha} \|^2$. The regularization weights are given by the prefix λ_j coefficients. This formulation is similar to [89] with the difference that it is made applicable here to large scale datasets. These $\tilde{\alpha}_j$ are coefficients of linear combination of the features which result in $\tilde{w}$ vector. The set of PVs is represented as $\mathcal{S}_{PV}$. The $\tilde{e_i}$ are error estimates and are determined only for the vectors belonging to the set $\mathcal{S}_{PV}$. Thus, the training set comprises of the vectors belonging to the set $\mathcal{S}_{PV}$.

Introducing the coefficient β for Lagrangian $\mathcal{L}$ one obtains: $\partial\mathcal{L}/\partial\tilde{\alpha}_j = 0 \Rightarrow \tilde{\alpha}_j = \sum_j \beta_i K_{ij}/\lambda_j, \partial\mathcal{L}/\partial\tilde{b} = 0 \Rightarrow \sum_i \beta_i = 0, \partial\mathcal{L}/\partial\tilde{e}_i = 0 \Rightarrow \beta_i = \gamma\tilde{e}_i, \partial\mathcal{L}/\partial\beta_i = 0 \Rightarrow \sum_j \tilde{\alpha}_j K_{ij} + \tilde{b} = y_i - \tilde{e}_i, \forall i$. Combining the conditions $\partial\mathcal{L}/\partial\tilde{\alpha}_i = 0$, $\partial\mathcal{L}/\partial\tilde{e}_i = 0$ and $\partial\mathcal{L}/\partial\beta_i = 0$ and after little algebraic manipulation yields $\sum_k \beta_k H_{ik} + \tilde{b} = y_i$, with $H = \tilde{K} diag(\lambda)^{-1}\tilde{K} + I_M/\gamma$ and $\tilde{K}$ is a kernel matrix. The kernel matrix $\tilde{K}$ is defined as $\tilde{K}_{ij} = \tilde{\phi}(x_i)^\intercal \tilde{\phi}(x_j)$ where $x_i \in \mathcal{S}_{PV}, x_j \in \mathcal{S}_{PV}$, $\tilde{\phi}(x_i) \in \mathbb{R}^M$ and $H \in \mathbb{R}^{M \times M}$.

This, together with $\partial\mathcal{L}/\partial\tilde{b} = 0$, results in the linear system

$$\left[\begin{array}{c|c} 0 & 1_k^\intercal \\ \hline 1_k & H \end{array}\right] \left[\begin{array}{c} \tilde{b} \\ \hline \beta \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline y \end{array}\right] \tag{2.15}$$

The procedure to obtain sparseness involves iteratively solving the system (2.15) for different values of λ and is described in Algorithm 6. Considering the tth iteration, we can build the matrix $H^t = \tilde{K} diag(\lambda^t)^{-1}\tilde{K} + I_M/\gamma$ and solve the system of linear equations to obtain the value of β^t and $\tilde{b}^t$. From this solution we get $\tilde{\alpha}^{t+1}$ and most of its element tend to zero, the $diag(\lambda^{t+1})^{-1}$ will end up having many zeros along the diagonal due to the values allocated to λ^{t+1}. It was shown in [80] that as $t \to \infty$, $\tilde{\alpha}^t$ converges to a stationary point $\tilde{\alpha}^*$ and this model is guaranteed to be sparse and result in set SV. This iterative sparsifying procedure converges to a local minimum as the L_0-norm problem is NP-hard. Since this $\tilde{\alpha}^*$ depends on the initial choice of weights, we set them to the FS-LSSVM solution $\tilde{w}$, so as to avoid ending up in different local minimal solutions.

The *convergence* of Algorithm 6 is assumed when the $\| \tilde{\alpha}^t - \tilde{\alpha}^{t+1} \| /M'$ is lower than 10^{-4} or when the number of iterations t exceeds 50. The result of the approach is the *indices* of those PVs for which $|\tilde{\alpha}_i| > 10^{-6}$. These *indices* provide the set of most appropriate prototype vectors (SV). The FS-LSSVM method (Algorithm 5) is re-perfomed using only this set SV. We are training only on the set $\mathcal{C}_{PV}$ and not on the entire training data because the H matrix becomes $N \times N$ matrix which cannot in memory for large scale datasets.

Algorithm 6: *SV_L*$_0$-norm PFS-LSSVM method

Data: Solve FS-LSSVM to obtain initial $\tilde{w}$ & $\tilde{b}$
$\tilde{\alpha} = \tilde{w}(1:M)$
$\lambda_i \leftarrow \tilde{\alpha}_i, i = 1, \ldots, M$
while *convergence* **do**
  $H \leftarrow \tilde{K} diag(\lambda)^{-1} \tilde{K} + I_M/\gamma$
  Solve system (2.15) to give β and $\tilde{b}$
  $\tilde{\alpha} \leftarrow diag(\lambda)^{-1} \tilde{K} \beta$
  $\lambda_i \leftarrow 1/\tilde{\alpha}_i^2, i = 1, \ldots, M'$
end
Result: $indices = \text{find}(|\tilde{\alpha}_i| > 0)$

The time complexity of the proposed methods is bounded by solving the linear system of equations (2.15). An interesting observation is that the H matrix becomes sparser after each iteration. This is due to the fact that $diag(\lambda)^{-1} = diag(\tilde{\alpha}_1^2, \ldots, \tilde{\alpha}_M^2)$ and most of these $\tilde{\alpha}_i \to 0$. Thus the H matrix becomes sparser in each iteration such that after some iterations inverting H matrix is equivalent to inverting each element of the H matrix. The computation time is dominated by matrix multiplication to construct the H matrix. The H matrix construction can be formulated as multiplications of two matrices i.e. $P = \tilde{K} diag(\lambda)^{-1}$ and $H = P\tilde{K}$. The P matrix will become sparser as it multiplies the $\tilde{K}$ matrix with $diag(\lambda)^{-1}$. Let $\tilde{M}$ be the number of columns in P matrix with elements $\neq 0$. This number i.e. $\tilde{M}$ can be much less than M. Thus, for the *SV_L*$_0$-norm PFS-LSSVM the time required for the sparsifying procedure is given by $O(M^2\tilde{M})$ and the average memory requirement is $O(M^2)$.

2.2.3 Window reduced FS-LSSVM

In [82], it was proposed to remove the support vectors with smaller $|\alpha_i|$ for the LSSVM method. But this approach doesn't greatly reduce the number of support vectors. In [85], the authors proposed to remove support vectors with the larger $y_i f(x_i)$ as they are farther from decision boundary and easiest to classify. But these support vectors are important as they determine the true extent of a class.

We propose window based SV selection method for both classification and regression. For classification, we select the vectors which are correctly classified and closer and farther from the decision boundary. An initial FS-LSSVM method determines the $\tilde{y}$ for the PVs. To find the reduced set SV, we first remove the prototype vectors which were misclassified ($\tilde{y} \neq y$) as shown in [27]. Then, we sort the estimated $f(x_j)$, $\forall j \in CorrectPV$ where $CorrectPV$ is the set of correctly classified PVs to obtain a sorted vector S. This sorted vector is divided into two halves one containing the sorted positive estimations ($\tilde{y}$)

corresponding to positive class and the other containing sorted negative values ($\tilde{y}$) corresponding to negative class. The points closer to the decision boundary have smaller positive and smaller negative estimations ($|\tilde{y}|$) and the points farther from the decision boundary have the larger positive and larger negative estimations ($|\tilde{y}|$) as depicted in Figure 2.5. So these vectors corresponding to these estimations are selected. Selecting correctly classified vectors closer to decision boundary prevents over-fitting and selecting vectors farther from the decision boundary helps to identify the extent of the classes.

For regression, we select the prototype vectors which have least mse after one iteration of FS-LSSVM. We estimate the squared errors for the PVs and out of these prototype vectors select those vectors which have the least mse to form the set SV. They are the most appropriately estimated vectors as they have the least error and so are most helpful in estimating a generalization of the regression function. By selection of prototype vectors which have least mse we prevent selection of outliers as depicted in Figure 2.6. Finally, a FS-LSSVM regression is re-performed on this set SV.

The percentage of vectors selected from the initial set of prototype vectors is determined by the window. We experimented with various window size i.e. $(30, 40, 50)$ percent of the initial prototype vectors (PVs). For classification, we selected half of the window from the positive class and the other half from the negative class. In case the classes are not balanced and number of PVs in one class is less than half the window size then all the correctly classified vectors from those PVs are selected. In this case, we observe that the number of selected prototype vectors (SV) can be less than window size. The methodology to perform this Window reduced FS-LSSVM is presented in Algorithm 7.

This method result in better generalization error with smaller variance achieving sparsity. The trade-off with estimated error is not significant and in several cases it leads to better results as will be shown in the experimental results. As we increase the window size the variation in estimated error decreases and estimated error also decreases until the median of the estimated error becomes nearly constant as is depicted in Figure 2.7.

2.2.4 Computational Complexity and Experimental Results

Computational Complexity

The computation time of FS-LSSVM method involves:

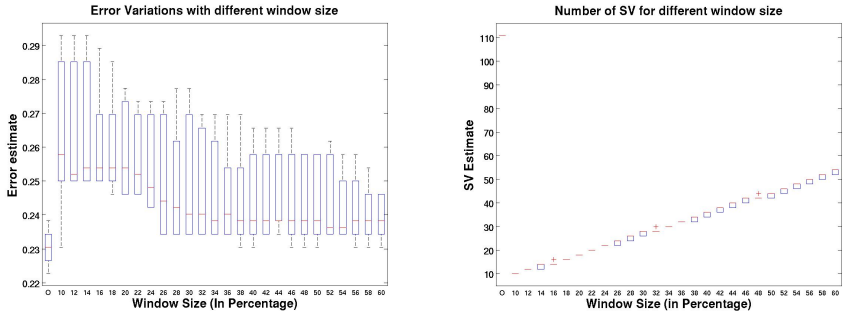


Figure 2.7: Trends in error & SV with increasing window size for Diabetes dataset compared with the FS-LSSVM method represented here as 'O'

Algorithm 7: Window reduced FS-LSSVM

Data: $\mathcal{D} = \{(x_i, y_i) : x_i \in \mathbb{R}^d, y_i \in \{+1, -1\}$ for $Classification$ & $y_i \in \mathbb{R}$ for $Regression$, $i = 1, \ldots, N\}$.

Perform FS-LSSVM using the initial set of PVs of size M on training data.
if *Classification* **then**
  $CorrectPV$ = Remove misclassified prototype vectors
  $S = sort(f(x_i))\ \forall i \in CorrectPV$;
  $A = S(:) > 0$; $B = S(:) < 0$;
  $begin = windowsize/4$;
  $endA = size(A) - windowsize/4$;
  $endB = size(B) - windowsize/4$;
  SV = $[A[begin\ \ endA]; B[begin\ \ endB]]$;
end
if *Regression* **then**
  Estimate the squared error for the initially selected PVs
  SV = Select the PVs with least mean squared error
end
Re-perform the FS-LSSVM method using the reduced set SV of size M' on training data

- Solving a linear system of size $M+1$ where M is the number of prototype vectors selected initially (PV).
- Calculating the Nyström approximation and eigenvalue decomposition of the kernel matrix of size M once.
- Forming matrix product $[\tilde{\phi}(x_1), \tilde{\phi}(x_2), \ldots, \tilde{\phi}(x_n)]^\intercal [\tilde{\phi}(x_1), \tilde{\phi}(x_2), \ldots, \tilde{\phi}(x_n)]$.

The computation time is $O(NM^2)$ where N is dataset size as shown in [30]. We already presented the computation time for the iterative sparsifying procedure for L_0 reduced FS-LSSVM. For this approach, the computation time $O(M^3)$. So, it doesn't have an impact on the overall computational complexity as we will observe from the experimental results. In our experiments, we selected

$M = \lceil k \times \sqrt{N} \rceil$ where $k \in \mathbb{N}$, the complexity of L_0 reduced FS-LSSVM can be re-written as $O(k^2N^2)$. We experimented with various values of k and observed that after certain values of k, the change in estimated error becomes nearly irrelevant. In our experiments, we choose the value of k corresponding to the first instance after which the change in error estimations becomes negligible.

For the window based method, we have to run the FS-LSSVM once and based on window size obtain the set SV which is always less than PVs i.e. $M' \leq M$. The time-complexity for re-performing the FS-LSSVM on the set SV is $O(M'^2N)$ where N is the size of the dataset. The overall time complexity of the approach is $O(M^2N)$ required for Nyström approximation and the average memory requirement is $O(NM)$.

Experiments

All the experiments are performed on a PC machine with Intel Core i7 CPU and 8 GB RAM under Matlab 2008a. We use the RBF-kernel for kernel matrix construction in all cases. As a pre-processing step, all records containing unknown values are removed from consideration. Input values have been normalized. We compare the performance of our proposed approaches with the normal FS-LSSVM classifier/regressor, L_0 LSSVM [89], SVM and ν-SVM. The last two methods are implemented in the LIBSVM software with default parameters. All methods use a cache size of 8 GB. Shrinking is applied in the SVM case. All comparisons are made on 10 randomizations of the methods.

The comparison is performed on an out-of-sample test set consisting of 1/3 of the data. The first 2/3 of the data is reserved for training and cross-validation. The tuning parameters σ and γ for the proposed FS-LSSVM methods and SVM methods are obtained by first determining good initial starting values using the method of coupled simulated annealing (CSA) in [93]. After that a derivative-free simplex search is performed. This extra step is a fine tuning procedure resulting in more optimal tuning parameters and better performance.

Table 2.8 provides a comparison of the mean estimated error $\pm$ its deviation, mean number of selected prototype vectors SV and a comparison of the mean computation time $\pm$ its deviation for 10 randomizations of the proposed approaches with FS-LSSVM and SVM methods for various classification and regression data sets. Figure 2.8 represents the estimated error, run time and variations in number of selected prototype vectors for Adult (ADU) and Slice Localization (SL) datasets respectively.

Test Classification(Error and Mean SV)												
	RIP		BC		DIB		SPAM		MGT		ADU	
Algorithm	Error	SV	Error	SV	Error	SV	Error	SV	Error	SV	Error	SV
FS-LSSVM	**0.0924 ± 0.01**	32	0.047 ± 0.004	155	0.233 ± 0.002	111	0.077 ± 0.002	204	**0.1361 ± 0.001**	414	0.146 ± 0.0003	664
L_0 LSSVM	0.183 ± 0.1	7	0.04 ± 0.011	17	0.248 ± 0.034	10	**0.0726 ± 0.005**	340	-	-	-	-
C-SVC	0.153 ± 0.04	81	0.054 ± 0.07	318	0.249 ± 0.0343	290	0.074 ± 0.0007	800	0.144 ± 0.0146	7000	0.1519(∗)	11,085
ν-SVC	0.1422 ± 0.03	86	0.061 ± 0.05	330	0.242 ± 0.0078	331	0.113 ± 0.0007	1525	0.156 ± 0.0142	7252	0.161(∗)	12,205
L_0 reduced FS-LSSVM	0.1044 ± 0.0085	17	**0.0304 ± 0.009**	19	0.239 ± 0.03	36	0.1144 ± 0.053	144	0.1469 ± 0.004	115	0.1519 ± 0.002	78
Window (30%)	0.095 ± 0.009	10	**0.0441 ± 0.0036**	46	**0.2234 ± 0.006**	26	0.1020 ± 0.002	60	0.1522 ± 0.002	90	**0.1449 ± 0.003**	154
Window (40%)	**0.091 ± 0.015**	12	**0.0432 ± 0.0028**	59	**0.2258 ± 0.008**	36	0.0967 ± 0.002	78	0.1481 ± 0.0013	110	**0.1447 ± 0.003**	183
Window (50%)	**0.091 ± 0.009**	16	**0.0432 ± 0.0019**	66	**0.2242 ± 0.005**	45	0.0955 ± 0.002	98	0.1468 ± 0.0007	130	**0.1446 ± 0.003**	213
Test Regression(Error and Mean SV)												
	Motorcycle		Boston Housing		Concrete				Slice Localization			
Algorithm	Error	SV	Error	SV	Error	SV			Error	SV		
FS-LSSVM	**0.2027 ± 0.007**	37	**0.1182 ± 0.001**	112	**0.1187 ± 0.006**	192			**0.053 ± 0.0**	694		
L_0 LSSVM	0.2537 ± 0.0723	9	0.16 ± 0.05	65	0.165 ± 0.07	215			-	-		
ϵ-SVR	0.2571 ± 0.0743	69	0.158 ± 0.037	226	0.23 ± 0.02	670			0.1017(*)	13,012		
ν-SVR	0.2427 ± 0.04	51	0.16 ± 0.04	226	0.22 ± 0.02	330			0.0921(*)	12,524		
SV_L_0-norm FS-LSSVM	0.62 ± 0.29	5	0.1775 ± 0.057	48	0.2286 ± 0.03	43			0.1042 ± 0.06	495		
Window (30%)	0.32 ± 0.03	12	0.1465 ± 0.02	34	0.2 ± 0.005	58			0.1313 ± 0.0008	208		
Window (40%)	0.23 ± 0.03	15	0.1406 ± 0.006	46	0.1731 ± 0.006	78			0.116 ± 0.0005	278		
Window (50%)	0.23 ± 0.02	21	**0.1297 ± 0.009**	56	0.1681 ± 0.0039	96			0.1044 ± 0.0006	347		

Train & Test Classification(Computation Time)						
Algorithm	RIP	BC	DIB	SPAM	MGT	ADU
FS-LSSVM	4.346 ± 0.143	28.2 ± 0.691	15.02 ± 0.688	181.1 ± 7.75	3105.6 ± 68.6	23,565 ± 1748
L_0 LSSVM	5.12 ± 0.9	28.2 ± 5.4	39.1 ± 5.9	950 ± 78.5	-	-
C-SVC	13.7 ± 4	43.36 ± 3.37	24.8 ± 3.1	1010 ± 785	20,603 ± 396	139,730(*)
ν-SVC	13.6 ± 4.5	41.67 ± 2.436	23.23 ± 2.3	785 ± 22	35,299 ± 357	139,927(*)
SV_L_0-norm FS-LSSVM	4.349 ± 0.28	28.472 ± 1.07	15.381 ± 0.5	193.374 ± 8.35	3185.7 ± 100.2	22,485 ± 650
Window (30%)	4.28 ± 0.22	28.645 ± 1.45	14.5 ± 0.5	171.78 ± 4.82	3082.9 ± 48	22,734 ± 1520
Window (40%)	4.33 ± 0.27	27.958 ± 1.21	14.6 ± 0.75	172.62 ± 5.3	3052 ± 42	22,466 ± 1705
Window (50%)	4.36 ± 0.27	28.461 ± 1.3	14.255 ± 0.5	167.39 ± 2.31	3129 ± 84	22,621 ± 2040
Train & Test Regression(Computational Time)						
Algorithm	Motorcycle	Boston Housing		Concrete	Slice Localization	
FS-LSSVM	3.42 ± 0.12	14.6 ± 0.27		55.43 ± 0.83	37,152 ± 1047	
L_0 LSSVM	10.6 ± 1.75	158.86 ± 3.2		753 ± 35.5	-	
ϵ-SVR	24.6 ± 4.75	63 ± 1		55.43 ± 0.83	252,438(*)	
ν-SVR	25.5 ± 3.9	61 ± 1		168 ± 3	242,724(*)	
L_0 reduced FS-LSSVM	3.426 ± 0.18	15.4 ± 0.35		131 ± 2	36,547 ± 1992	
Window (30%)	3.52 ± 0.244	15.04 ± 0.17		56.75 ± 1.35	36,087 ± 1066	
Window (40%)	3.425 ± 0.153	15.04 ± 0.27		55.67 ± 0.81	35,906 ± 1082	
Window (50%)	3.5 ± 0.226	15.04 ± 0.15		55.06 ± 1.088	36,183 ± 1825	

Table 2.8: Comparison of performance of different methods for UCI repository datasets. The '*' represents no cross-validation and performance on fixed tuning parameters due to computational burden and '-' represents cannot run due to memory constraints.

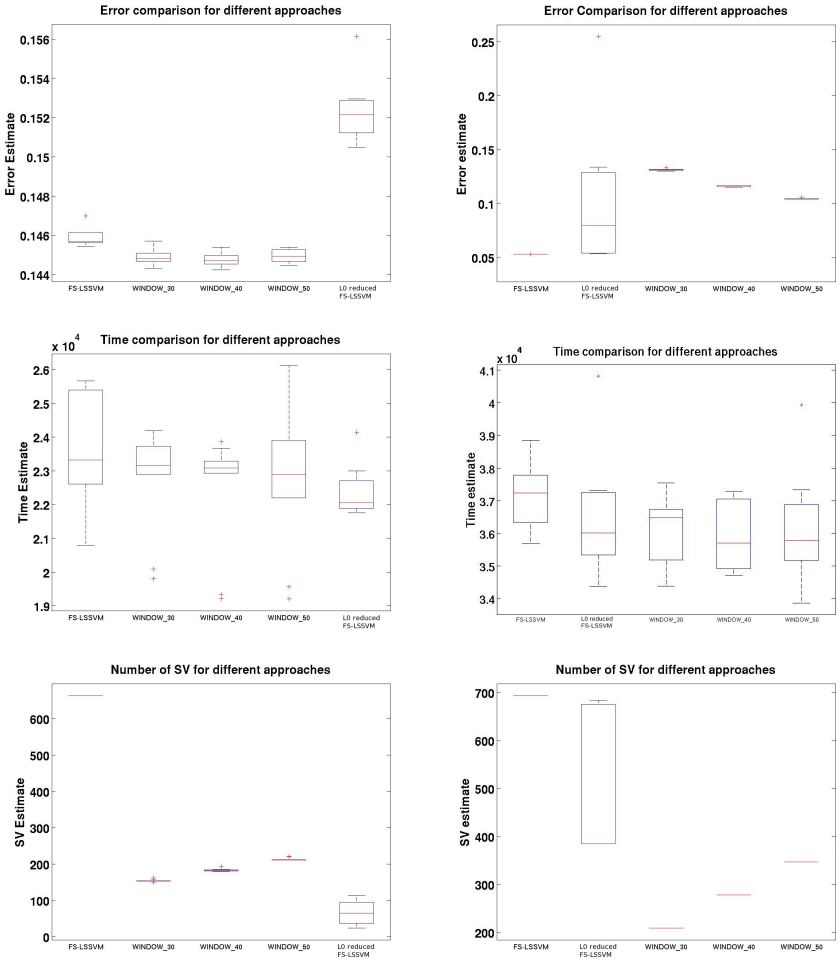


Figure 2.8: Comparison of performance of proposed approaches with FS-LSSVM method for Adult & Slice Localization datasets

Performance Analysis

The proposed approaches i.e SV_L_0-norm PFS-LSSVM and Window reduced FS-LSSVM method introduce more sparsity in comparison to FS-LSSVM and SVM methods without significant trade-off for classification. For smaller datasets L_0 LSSVM produces extremely few support vectors but for datasets like SPAM, Boston Housing and Concrete it produces more support vectors. For some

datasets like breast-cancer and diabetes, it can be seen from Table 2.8 that proposed approaches results in better error estimations than other methods with much smaller set SV. For datasets like SPAM and MGT, the trade-off in error is not significant considering the reduction in number of PVs (only 78 prototype vectors required by SV_L_0-norm PFS-LSSVM for classifying nearly 20,000 points). From Figure 2.8, we observe the performance for Adult dataset. The window based methods result in lower error estimate using fewer but more appropriate SV. Thus the idea of selecting correctly classified points closer and farther from the decision boundary results in better determining the extent of the classes. These sparse solutions typically lead to better generalization of the classes. The mean time complexity for different randomizations is nearly the same.

For regression, as we are trying to estimate a continuous function, if we greatly reduce the PVs to estimate that function, then the estimated error would be higher. For datasets like boston housing and concrete, the estimated error by the proposed methods is more than FS-LSSVM method but the amount of sparsity introduced is quite significant. These methods result in reduced but more generalized regressor functions and the variation in the estimated error of window based approach is lesser as in comparison to L_0-norm based method. This is because in each randomization, the L_0-norm reduces to different number of prototype vectors whereas the reduced number of prototype vectors (SV) for window based method is fixed and is uninfluenced by variations caused by outliers as the SV have least mse. This can be observed for the Slice Localization dataset in Figure 2.8. For this dataset, SV_L_0-norm PFS-LSSVM estimates lower error than window approach. This is because for this dense dataset, the SV_L_0-norm PFS-LSSVM requires more SV (495) than window based method $(208, 278, 347)$ which signifies more vectors are required for better error estimation. The proposed models are of magnitude $(2 - 10)$x sparser than the FS-LSSVM method.

2.2.5 Conclusion

In this work, we proposed two sparse reductions to FS-LSSVM namely SV_L_0-norm and Window reduced PFS-LSSVM. These methods are highly suitable for mining large scale datasets overcoming the problems faced by L_0 LSSVM [89] and FS-LSSVM. We developed the SV_L_0-norm PFS-LSSVM based on iteratively sparsifying L_0-norm training on the initial set of PVs. We also introduced a Window reduced FS-LSSVM trying to better determine the underlying structure of model by selection of more appropriate prototype vectors (SV).

Chapter 3

Subset Selection

This chapter comprises previously published articles including:

1) Mall R., Langone R., Suykens J.A.K., **"FURS: Fast and Unique Representative Subset selection retaining large scale community structure"**, Social Network Analysis and Mining, 3(4), Oct 2013, pp. 1075-1095.

2) Mall R., Jumutc V., Langone R., Suykens J.A.K., **"Representative Subsets For Big Data Learning using k-NN graphs"**, IEEE BigData, Washington D.C., U.S.A, 2014, pp. 37-42.

Keywords—sampling graphs;subset selection; k-NN graphs; Map-Reduce;

3.1 FURS selection retaining community structure[1]

Abstract — We propose a novel algorithm, FURS (Fast and Unique Representative Subset selection) to deterministically select a set of nodes from a given graph which retains the underlying community structure. FURS greedily selects nodes with high-degree centrality from most or all the communities in the network. The nodes with high-degree centrality for each community are

[1]This section consists of Sections $1-5$ and Section 8 from Mall R., Langone R., Suykens, J.A.K., **"FURS: Fast and Unique Representative Subset selection retaining community structure"**, Social Network Analysis and Mining, 3(4), Oct 2013, pp. 1075-1095.

usually located at the center rather than the periphery and can better capture the community structure. The nodes are selected such that they are not isolated but can form disconnected components. The FURS is evaluated by means of quality measures like coverage, clustering coefficients, degree distributions and variation of information. Empirically, we observe that the nodes are selected such that most or all of the communities in the original network are retained. We compare our proposed technique with state-of-the-art methods like SlashBurn, Forest-Fire, Metropolis and Snowball Expansion sampling techniques. We evaluate FURS on several synthetic and real-world networks of varying size to demonstrate the high quality of our subset while preserving the community structure. The subset generated by the FURS method can be effectively utilized by model based approaches with out-of-sample extension properties for inferring community affiliation of the large scale networks.

3.1.1 Introduction

In the modern era graphs have become universal. Their applications span from social network analysis, bio-informatics, telecommunication networks to even software engineering. With the advancement of technology, widespread use of Internet and availability of cheap sensors, the amount of information that can be collected is only increasing. This leads to large scale graphs with hundreds of thousands to even millions of nodes. There are several internet based organizations from Facebook to LinkedIn which produces graphs ranging from online social networks to professional networks scaling to 100 million users [98], [99] and [100] and captures the interactions between these users. In the telecommunication field, the cell phone interactions produces large scale graphs and provide insight that groups of people prefer to converse with which other groups of people [36]. In biological systems, graphs are generated from interactions between various entities which reflect the associations between these entities. For example the interactions between neurons in the brain to associations between the proteins in food synthesis [101] and [102].

Real world graphs exhibit community structure where the nodes are densely connected within the community and sparsely connected between the communities. The problem of community detection has received a lot of attention in recent years [39], [103], [38], [104], [105], [106] and [42]. These communities are of great importance as they help to shed light on behavior and functioning of the networks, like buyer or seller behavior during times of crisis. However, the modern day networks are extremely large and detecting communities from these networks can become impractical and intractable due to memory and time constraints. The question to ask then is how to overcome the challenge of scale of the networks and perform data analysis of these networks. One direction to

proceed is to develop efficient algorithms which are fast, accurate, scalable [42], [36] and might use parallelization or distributed computing. The other approach which has been receiving some attention lately is the method of *sampling*.

Sampling is conventionally done by a stochastic algorithm when one is interested in performing computations that are too expensive for the large graph. A sample of the network can be a set of nodes from the large graph along with their edges. Another sample can be a set of edges from the large graph along with the corresponding vertices. The simplest technique for obtaining such a sample would be to perform a random sampling. However, the subgraph obtained by random sampling does not retain the inherent community structure and is essentially useless for any meaningful analysis. Thus, the sampling of the network should be performed such that the obtained subgraph is a *good representative* of the original network. But how does one measure if a subgraph is a 'good representative' of the larger network? Existing work using graph properties like degree distributions and clustering coefficients are [107] and [77]. Another work argues that the measure of representativeness varies and depends on the analysis to be performed [75]. In this work, we use several evaluation metrics like Coverage (Cov), fraction of communities preserved (Frac), clustering-coefficients (CCF), degree distributions (DD) and variation of information (VI) to determine the quality of the subset generated by FURS.

Motivation & Contributions

Recent work [108] showed that egonets can exhibit conductance scores as good as the Fiedler cut and provide a good seed sample for a partitioning method like PageRank clustering. However, another work [74] suggests that real world scale-free networks follow power-law degree distributions and have 'no good cuts'. They provide an ordering of the nodes of the graph (SlashBurn algorithm) to obtain a good compression of the real world graphs. We concur with [74] and observe that nodes with high degree centrality or hubs tend to be part of dense regions of a graph.

The aim of this work is to select a subset of nodes which are located at the center of the communities in the large scale network without explicitly performing community detection. The nodes which are located at the center are good representative of the underlying community structure. The concept is parallel to the concept of identification and selection of k centroids for the k-means clustering technique [109]. For this purpose we want to locate and select nodes with high degree centrality. This is because nodes with high PageRank centrality [110, 111], eigenvector centrality [110, 111] and betweenness centrality [112] can be influential nodes in the large scale network but need not necessarily be

at the center of the communities. This problem of selection of a subset where the nodes are central to the communities present in the large network without explicitly perform community detection is NP-hard.

We propose a Fast and Unique Representative Subset (FURS) selection technique which is a greedy approximation of the above criterion. The basic idea is to first order the nodes based on their degree in descending order during each iteration and pick the node with highest degree centrality. Once such a node is selected its immediate neighbors are deactivated (as they can be reached directly from this node) during that iteration and the node is placed in the selected subset without changing the graph topology. We then select the node with highest degree centrality among the active nodes and the process is repeated until we reach the subset size. Once all nodes are deactivated, a new iteration is started and the deactivated nodes are re-activated. They are ordered according to their degree centrality in descending order and the process of node selection, deactivation and reactivation is repeated till we obtain the desired subset. The proposed approach greedily selects nodes with high degree centrality from different dense regions of the graph.

Thus, we propose a Fast and Unique Representative Subset (FURS) selection algorithm which deterministically obtains a representative subset of nodes while retaining the community structure of the large graph. The contributions in this work are listed as the follows:

- The sample set of nodes has high degree centrality. We observe that these nodes span the different communities in the graph capturing the community structure of the large network. This is evaluated by the metric fraction of communities of the large network preserved in the subset generated by the FURS. We experimentally demonstrate that the quality of the subset generated by FURS is better for several evaluation metrics than previous techniques.
- We compare and show that the proposed subset selection technique is **faster** than the state-of-the-art sampling techniques like SlashBurn, Metropolis and Snowball Expansion sampling.
- We show that the subset obtained by FURS is also a good candidate set for simple diffusion model. The spread of information over time using FURS is generally better than the candidate set obtained by random node selection, hubs selection, spokes selection, high eigenvector centrality, high Pagerank, high betweenness centrality and low betweenness centrality based representative subset selection.

3.1.2 Related Work

Sampling techniques can be broadly divided into two categories:

- **Node sampling** - Node sampling involves selecting nodes which form a representative subset of the graph. The selected set of nodes can either be connected or disconnected. The subgraph obtained from the subset containing disconnected nodes comprises disconnected components and can even have isolated nodes (w.r.t. the subgraph and not the large scale network). Some node sampling techniques include randomly selecting nodes based on degree centrality, random walk model and forest-fire model [77]. In [77], they evaluate the quality of the samples for these methods based on their ability to match various properties of the original graph structure like degree distributions, clustering coefficients and component sizes. They conclude that the sample obtained by the forest-fire approach is better than other methods. Recently, a new approach was proposed to provide an ordering of the nodes of the graph (SlashBurn algorithm) to obtain a good compression of the real world graphs in [74]. The SlashBurn algorithm can be used to obtain a subset of nodes which contain information about the inherent community structure. For the SlashBurn algorithm after selection of the k-hubset the connections are burnt and a new graph is constructed. The giant connected component is discovered in this new graph and the process of selection is performed recursively.
- **Subgraph sampling** - In subgraph sampling a new node is always selected from the neighborhood of an already selected node based on a criterion. As a result the obtained subgraph is always connected. This is a hard constraint and has to be followed making the problem more difficult and computationally expensive. In [107], the Metropolis algorithm [76] was used for a sample subgraph selection. Recently, two sampling technique using the concepts of expander graph was published in [75] where the obtain subgraph is connected.

We compare our proposed algorithm with the Forest-Fire [77] and SlashBurn [74] techniques from Node Sampling methods. We also compare our approach with the Metropolis Sampling using Degree Distribution (MDD) [76] and Snowball Expansion Sampling (XSN) [75] which is better of the two methods that had been proposed in [75], from the Subgraph sampling methods.

There have been other contributions involving sampling graphs for purposes like visualization [113], compression [114], [74], [115], [116], sociology [117] and epidemiology [118]. There is another work [119] which assumes that a network

sample is already generated and contains nodes from a single community. With this assumption, they propose a method to grow the network such that it includes all the members of this community. However, the aim of this work is to come up with a fast technique to obtain a unique subset of nodes which represents *all* or *most* of the communities in the network.

3.1.3 Proposed Method

All the graphs considered in this work are assumed to be undirected and unweighted unless otherwise mentioned. We first provide details about the notations used.

Notations

1. A graph is mathematically represented as $G = (V, E)$ where V represents the set of nodes and $E \subseteq V \times V$ represents the set of edges in the network. Physically, the nodes represent the entities in the network and the edges represent the relationship between these entities.
2. The cardinality of the set V is denoted as N.
3. The cardinality of the set E is denoted as e.
4. For unweighted graphs, A is called the adjacency matrix and $A_{ij} = 1$ if $(v_i, v_j) \in E$ else $A_{ij} = 0$.
5. The subgraph generated by the subset of nodes S is represented as $G(S)$. Mathematically, $G(S) = (S, Q)$ where $S \subset V$ and $Q = (S \times S) \cap E$ represents the set of edges in the subgraph.
6. The degree distribution function is given by $f(V)$. For the graph G it can written as $f(V)$ while for the subgraph S it can be presented as $f(S)$. Each vertex $v_i \in V$ has a degree represented as $f(v_i)$.
7. The degree matrix is represented as D, a diagonal matrix with diagonal entries $d_{i,i} = \sum_j A_{ij}$.
8. The adjacency list corresponding to each vertex $v_i \in V$ is given by $x_i = A(:, i)$.
9. The set of neighboring nodes of a given node v_i is represented by $Nbr(v_i)$.
10. The median degree of the graph is represented as m.

Core Concept

Nodes which have a high degree centrality or hubs represent the existence of more interaction in the network and have the tendency of being located at the center of a community. However, it is essential to select several such nodes of high degree centrality from the different communities in the large network. But this problem of selection of such a subset S without explicitly performing community detection is NP-hard. Mathematically it can be formulated as:

$$\begin{aligned} \max_{S} \quad & J(S) = \sum_{j=1}^{s} D(v_j) \\ \text{s.t.} \quad & v_j \in C_i, \\ & C_i \in \{C_1, \ldots, C_k\} \end{aligned} \tag{3.1}$$

where $D(v_j)$ represents the degree centrality of the node v_j, s is the size of the subset, C_i represents the ith community and k represents the number of communities in the network which cannot be obtained explicitly.

A greedy solution to the problem can be formulated in an optimization framework by maximizing the sum of the degree centrality of the nodes in selected subset S such that the neighbors of the selected nodes are deactivated for that iteration. By deactivating the neighbors we move from one dense region of the network to another dense region thereby approximately covering most or all the communities in the network. Till the subset size s is achieved, the deactivated nodes are activated in the next iteration and the procedure is re-performed. Algorithmically, it can be represented as,

$J(S) = 0$

While $|S| <$ s

$$\begin{aligned} \max_{S} \quad & J(S) := J(S) + \sum_{j=1}^{s^t} D(v_j) \\ \text{s.t.} \quad & Nbr(v_j) \rightarrow \text{deactivated, iteration t,} \\ & Nbr(v_j) \rightarrow \text{activated, iteration t+1,} \end{aligned} \tag{3.2}$$

where s^t is the size of the set of nodes selected by FURS during iteration t.

FURS Procedure

The FURS algorithm can be divided into three steps namely Hub Selection, Deactivation and Reactivation of nodes. We describe the FURS procedure in detail below:

1. **Hub Selection** - We first sort all the nodes on the basis of their degree centrality in descending order. We maintain the identity of the node and its corresponding degree centrality in a list. An important observation is that if two nodes have the same degree centrality, then after sorting they are maintained in an order which remains constant. Thus no matter how many times one runs the sorting procedure the nodes after sorting are always maintained in the same order. For example, consider a network of 5 nodes $((v_1, 5), (v_2, 3), (v_3, 5), (v_4, 4), (v_5, 3))$ where the first term in each tuple represents the node identifier and the second term represents the corresponding degree centrality. After sorting, the list is always represented as $((v_1, 5), (v_3, 5), (v_4, 4), (v_2, 3), (v_5, 3))$. The technique for subset selection is inspired by the greedy algorithm used for maximum coverage problem in graphs as introduced in [120].

   Before subset selection, we remove all the nodes from the graph whose degree centrality is less than the minimum of a user-defined threshold t and median degree centrality m of the network i.e. $D(v_i) < \min(t, m)$ as we wish to select nodes of higher degree centrality and prevent the selection of outliers. By putting this condition, we remove all cliques of size $\min(t, m)$ and discard such cliques as outlier community w.r.t. the size of other communities in the network. However, their connection with corresponding nodes is retained i.e. the degree distribution of the graph is retained. The median degree centrality m is the median value in the list of degree centrality values of all the nodes and is not affected by outliers. So, we prefer to use the median degree centrality instead of the mean degree centrality which is heavily influenced by outliers. After removal of all the nodes with degree centrality $D(v_i) < \min(t, m)$, we pop the node with highest degree centrality from the list and select it as the new node say v_j.

2. **Deactivation** - All the neighbors of v_j obtained by $A(v_j)$ are deactivated from the maintained list. By deactivating these nodes $Nbr(v_j)$, we simply don't consider these nodes for selection for the time being without affecting the graph topology. Thus, the degree distribution of the remaining nodes $V \backslash (Nbr(v_j) \cup v_j)$ stays unaffected.

   We then select the node with the next highest degree centrality from the list say v_p after deactivating the neighbors of v_j and we deactivate

$Nbr(v_p)$. By performing this operation, we ensure that the newly selected node will not appear in the neighbors of the existing subset of nodes for that iteration. This enables us to select nodes from different dense regions of the graph and thus have a representative subset containing nodes from *most* or *all* of the communities in the large network.

3. **Reactivation** - This process of selection of a node based on degree centrality and deactivating its immediate neighbors is performed iteratively until we obtain the required number of nodes which is equivalent to the subset size s. We observe empirically from our experiments that generally, it requires 2 iterations to obtain the required subset S.

We sort these nodes based on their degree centrality, maintain a list and iteratively re-perform all the operations. By performing this operation, we end up selecting several nodes from each dense region of the graph. The subgraph obtained from the subset selected by FURS can have disconnected components. We put the constraint that the resulting subgraph $G(S)$ does not contain isolated nodes as isolated nodes cannot capture underlying community structure. If the subgraph $G(S)$ contains isolated nodes then the subset size s is increased iteratively to $s := s + \lceil 0.05 \times N \rceil$ and FURS is re-performed. Thus nodes selected from each dense region are connected and the subset selected by FURS is not a maximal independent set of the large scale network. Algorithm 8 summarizes the FURS technique.

Algorithm 8: FURS Algorithm

Data: A list of nodes with their corresponding degree values $L = (V, f(V))$, the median degree m and the adjacency matrix A containing information about neighbors $Nbr(v_i)$, $\forall v_i \in V$.
Result: A subset of representative nodes S whose cardinality is N_S

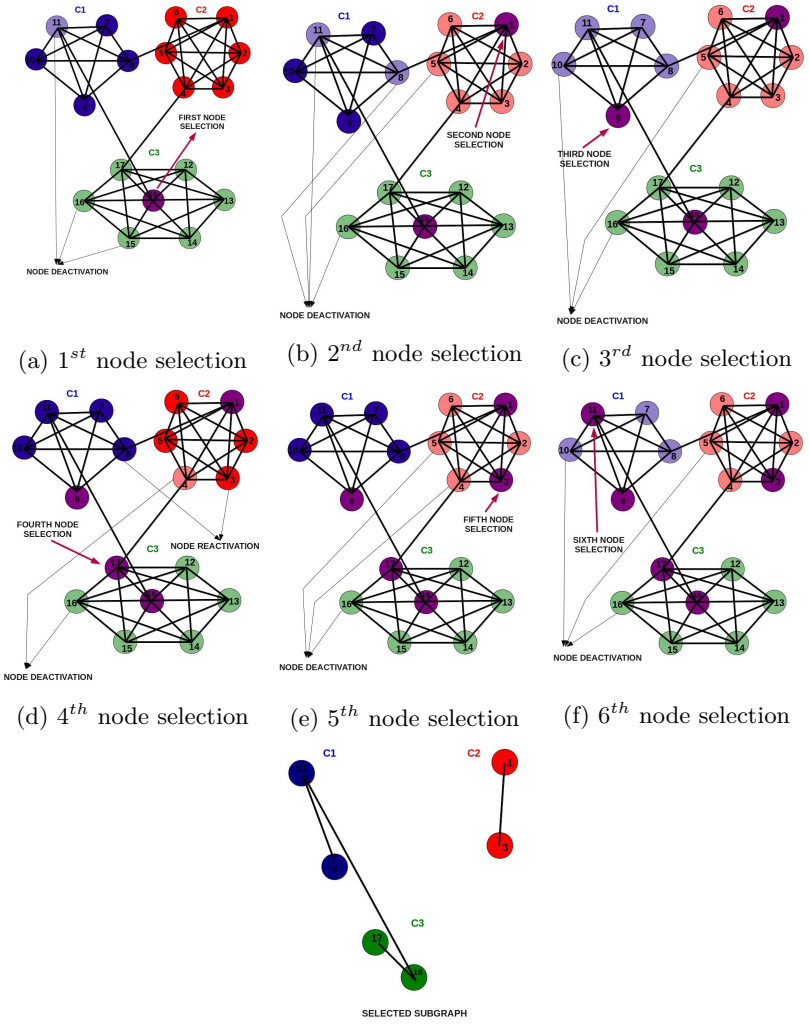
$L := (V, f(V))$, $\forall v_i \in V$ such that $f(v_i) > M$
$L :=$ sort(L) `// Based on the degree values in descending order`
while $|S| < N_S$ **do**
  `// REACTIVATION Step`
  if $L == \{\}$ **then**
    $L := L \cup \{v_i, f(v_i)\}$, $\forall v_i \in V$ that was deactivated.
    $L :=$ sort(L) `// Based on the degree values in descending order`
  end
  `// HUB SELECTION`
  $v_1 := L.pop()$ `// pop out the highest degree node`
  $S := S \cup v_1$ `// Add to output set S`
  $Nb \leftarrow Nbr(v_1)$ `// Neighboring nodes of` v_1
  `// Create a temporary list and add` Nb `along with their corresponding degree, if` $Nbr(v_1)$ `is not already present in the list.`
  `// DEACTIVATION Step`
  $L := L.deactivate(Nb, f(Nb))$ `// Deactivate the neighbors of` v_1
end
if $\sim$ *isempty(Isolated Nodes(S))* **then**
  $s := s + \lceil 0.05 \times N \rceil$.
  Re-perform FURS.
end

Figure 3.1 explains the working mechanism of FURS selection procedure on a small network of 18 nodes. FURS selects 6 nodes from this network and the subgraph corresponding to this subset contains nodes from *all* the communities in the network. We observe from Figure 3.1 the presence of 3 cliques $C1$, $C2$ and $C3$ of size 5, 6 and 7 respectively with few interconnections between them. We calculate the degree centrality values and maintain a sorted list L of the node identifier and the degree centrality of the corresponding node. Here, $L = \{(v_{18}, 7), (v_{17}, 7), (v_{16}, 6), (v_{15}, 6), (v_{14}, 6), (v_{13}, 6), (v_{12}, 6), (v_1, 6), (v_4, 6), (v_3, 5), (v_2, 5), (v_6, 5), (v_5, 5), (v_{11}, 5), (v_8, 5), (v_9, 4), (v_{10}, 4), (v_7, 4)\}$. In Figure 3.1a, we select the 1st node or the node with the highest degree centrality i.e. v_{18} and deactivate all the nodes of clique $C3$ along with node v_{11} which are neighbors of node v_{18}. After that we select node v_1 whose degree centrality 6 is maximum among the activated nodes. We deactivate all the nodes of clique $C2$ and node v_8 which are neighbors of v_1. This is depicted in Figure 3.1b. We then select v_9 which has maximum degree centrality ($D(v_9) = 4$) among the currently activated nodes. We observe that all the other nodes in the network are deactivated as observed in Figure 3.1c.

Since the required subset size ($s = 6$) is not equal to the current subset size ($s^1 = 3$ i.e. the size of subset after iteration 1 is 3), so we activate all the deactivated nodes. We then select node v_{17} whose degree centrality is 7. It is followed by deactivating all the nodes in clique $C3$ and node v_4 which are immediate neighbors of the node v_{17}. This step is depicted in Figure 3.1d. Figure 3.1e shows the selection of node v_3 from clique $C2$ as it has maximum degree centrality among the activated nodes. Finally, Figure 3.1f highlights the selection of node v_{11} as $D(v_{11}) = 5$. The resulting subgraph is shown in Figure 3.1g and contains a disconnected component corresponding to clique $C2$. Thus the resulting subgraph $G(S)$ captures community information about all the three communities present in the network.

Time Complexity

The FURS algorithm results in a unique representative subset of the entire network as the selection process is deterministic. The initial seed node is selected such that it has the highest degree centrality in the graph. In order to maintain the list L of nodes along with their corresponding degree centrality in the ranking of largest to smallest degree centrality value, we need to sort L. This is computationally the most expensive step of our proposed algorithm. The minimum time required to perform this sorting is $O(N.\log(N))$. Every time L becomes empty, we reinitialize the list L with the nodes and degree centrality values of the nodes which were deactivated in the previous iteration. Let the number of such iterations required be *iter*. Thus, the overall computations



(a) 1st node selection (b) 2nd node selection (c) 3rd node selection

(d) 4th node selection (e) 5th node selection (f) 6th node selection

(g) FURS Subgraph

Figure 3.1: Steps involved in FURS for subset size 6 from network of 18 nodes.

required for sorting becomes $O(iterN\log(N))$. In general, we observe that 2-3 iterations are sufficient to obtain the required subset S.

Apart from sorting the list L, the other computation that is being performed is deactivating the neighbors of the winning node. Let $S = (p_1, p_2, \ldots, p_s)$ be the

set of nodes sampled by the proposed algorithm. For each node $p_i \in S$, we have to deactivate all its neighbors $Nbr(p_i)$. Deactivating each neighbor of a node p_i takes unit computation time. The computational time required for the purpose of deactivation can then be represented as $O(\sum_{i=1}^{s} Nbr(p_i))$. Thus, the overall computational complexity of the algorithm is $O(iterN\log(N) + \sum_{i=1}^{s} Nbr(p_i))$.

3.1.4 Evaluation Metrics

Current community detection algorithms generate different partitions in each iteration for a given large scale network. For a fair comparison, we first generate a partition of the large graph using a scalable community detection algorithm and then run the same algorithm on the subgraphs generated by various sampling techniques. In order to obtain method-independent results we experimented with three different community detection algorithms namely CNM [38], Infomap [42] and Louvain [36] as these approaches can handle large scale networks. We then evaluate the subgraph generated by each selection technique on various metrics like time required to generate the subgraph, clustering coefficients, degree distributions, coverage, variation of information and fraction of communities preserved. The results reported are the mean values for the various evaluation metrics. The measures like variation of information, clustering coefficients, degree distribution compare the extent of similarity of the generated subgraph $G(S)$ with respect to the subgraph for the same set of nodes in the original graph $G(S')$. A summary of the various evaluation metrics is mentioned below.

Variation of Information: Variation of Information (VI) is an information theoretic measure and is used to compare two different partitions as depicted in [121]. Mathematically VI can be formulated as:

$$VI(U,V) = \sum_{i=1}^{k}\sum_{j=1}^{r} \frac{n_{ij}}{n} log(\frac{n_i.n_j/n^2}{n_{ij}/n^2}),$$

where n_i represents the number of nodes in cluster i in partitioning U and n_j represents the number of nodes in cluster j in partitioning V and n_{ij} is the joint distribution of the cluster memberships in U and V. The VI measure is not normalized and the range for variation of information is $[0; 2\log(\max(k; r))]$ [122] and lower the VI better the match between the cluster memberships.

Clustering Coefficient: The clustering coefficient (CCF) is defined as a vector with values ranging between $[0, 1]$ both inclusive. We compare it using the L_1-norm. In order to prevent any bias like a single degree dominating the distance, we prevent the use of higher order L-norms including L_∞. We calculate the average (absolute) difference between the clustering coefficients which is

mathematically formulated as $\frac{\sum_{v \in S} |G(v) - S(v)|}{|S|}$. Once we obtain this average distance, we convert it into similarity measure by subtracting the distance from 1 as in [75].

Degree Distribution: We compare the degree distributions (DD) of the large graph and the subgraph generated by the selection technique using the Kolmogorov-Smirnov D-Statistics as employed in [107], [75].The Kolmogorov-Smirnov D-Statistics corresponds to the maximum difference between the two cumulative distribution functions F_Y of G and $F_{Y'}$ of S over the range of random variables Y and Y'. Y and Y' are distributed according to G and S respectively. The distance $D(G, S)$ is formulated as: $D(G, S) = \max_{v \in S} |F_Y(v) - F_{Y'}(v)|$. We convert this distance into a similarity measure by subtracting the distance from 1 as in [75].

Coverage: Coverage (Cov) is a simple evaluation metric which is defined as the ratio of the total number of unique nodes directly reachable from the nodes in the selected subset to the total number of nodes in the graph. It can be represented as the ratio of cardinality of the set of all the nodes directly reachable from the nodes in the selected subset to the total number of nodes in the graph and mathematically be formulated as $\frac{|\cup_{s_i \in S} N(s_i)|}{n}$. Coverage varies between 0 and 1 and higher values result in better coverage.

Fraction of Communities: We determine the fraction of total communities in the larger network represented by the subgraph generated by the selection technique as the fraction of communities preserved (Frac). This number ranges between 0 and 1 and was also used in [75].

3.1.5 Experiments

Real World Networks

We compare our proposed sampling technique on several real-world networks ranging from social networks, communication networks, citation networks, collaboration networks, web graphs, internet peer to peer networks to road networks. These networks are available at the `http://snap.stanford.edu/data/index.html`. Table 3.1 reflects a few keys statistics of each network.

Experimental Setup

We compare our proposed FURS method with Forest-Fire sampling (FF) [77], Metropolis using Degree Distribution (MDD) [107], Snowball Expansion (XSN)

Network	**Nodes**	**Edges**	**CCF**
p2p	10,876	39,994	0.008
Cond-mat	23,133	186,936	0.6334
HepPh	34,401	421,578	0.1457
Enron	36,692	367,662	0.497
Epinions	75,879	508,837	0.2283
Web-Stanford	281,903	2,312,497	0.619
roadCA	1,965,206	5,533,214	0.0464
Livejournal	3,997,962	34,681,189	0.3538

Table 3.1: Nodes (V), Edges (E) and Clustering Coefficients (CCF) for each network

sampling [75] and SlashBurn algorithm [74]. These are the state-of-the-art techniques for sampling community structure. For MDD the produced samples try to mimic the degree distribution of the original network. In XSN, the sample set S is selected such that it maximizes the expansion factor: $\frac{|N(S)|}{|S|}$ and the concept behind SlashBurn algorithm was explained earlier.

We perform all the experiments on a computer with 12 Gb RAM and 2.4 GHz Intel Xeon processor. We perform 5 randomizations of community detection algorithms (Louvain, Infomap, CNM) on the large network and for each randomization, we perform community detection on the subgraph generated by each of the subset selection method. Thus, we report mean and standard deviation values for the various evaluation metrics. The subset size is maintained as 15% of the nodes in the network as per the experimental analysis in [77]. For the Metropolis algorithm based MDD, we perform $1,000$ iterations to produce each sample.

Experimental Results

We perform exhaustive experiments on 8 benchmark real world networks using various evaluation metrics. It is depicted in Table 3.2. Some of the abbreviated metrics in Table 3.2 are VI_LN i.e. variation of information for Louvain method, Frac_LN i.e. fraction of communities preserved by Louvain method. Other abbreviations include (VI_IP) for variation of information for Infomap method, (VI_CNM) for variation of information for CNM method, (Frac_IP) for fraction of communities captured by Infomap method and (Frac_CNM) for fraction of communities captured by CNM. We observe that the FURS approach performs well with respect to computation time, clustering coefficients, coverage and fraction of communities preserved by Louvain and Infomap method for most of the networks. FURS is better than at least three other sampling methods on most of the networks. However, FURS performs worst for Cond-mat network w.r.t. the metric VI_LN, HepPh network w.r.t. the metric Frac_CNM and

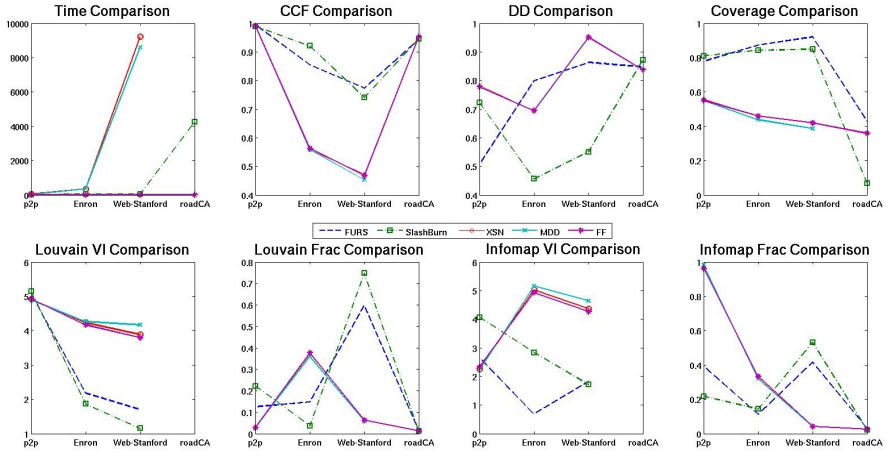


Figure 3.2: Evaluation of various subset selection methods on 4 real world networks of increasing size

Enron network w.r.t. the quality metric Frac_IP. However, the other sampling techniques are worse on one or more properties for each network. This is highlighted in Table 3.2 for the SlashBurn approach which is our primary competitor. The SlashBurn performs worst for CCF, DD, Coverage, VI_LN, Frac_LN, VI_IP, Frac_IP, VI_CNM and Frac_CNM for one or more network. The SlashBurn method performs the worst for the p2p network. However, in general it can better capture the evaluation metric - variation of information for the different community detection algorithms.

Figure 3.2 refers to the application of various subset selection techniques on 4 real world networks of increasing scale. We observe that the XSN and MDD technique become computationally infeasible for the roadCA network. We observe that the FURS selection technique is fast, has high clustering coefficients, coverage, smaller variation of information and better preserves the fraction of community in the large networks. However, the internet peer to peer network network (p2p) is an exception on which the XSN, MDD and Forest-Fire (FF) sampling perform better. From Figure 3.2, we observe that the SlashBurn algorithm can effectively capture the variation of information for the large web network of Stanford University (web-Stanford) w.r.t. both Louvain and Infomap community detection methods. The VI metric can be high even when the Frac values are high. This is because size of the partitions in the subgraphs is not necessarily uniform. Hence, higher entropy and higher VI value as observed in some cases for FURS. We cannot evaluate the VI metric for massive scale networks like roadCA and Livejournal as it is computationally very expensive.

		p2p		Cond-mat		HepPh		Enron		Epinions		Web-Stanford		roadCA		Livejournal	
Algo	Properties	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
	Time	**0.45**	0.0	**4.92**	0.0	**17.05**	0.0	**14.01**	0.0	**19.0**	0.0	**35.862**	0	**49.4**	0	**499**	0.0
	CCF	**0.995**	0.0	0.73	0.0	**0.87**	0.0	0.85	0.0	0.87	0.0	**0.77**	0	0.94	0	**0.9051**	0.0
	DD	0.5	0.0	0.853	0.0	0.81	0	**0.8**	0.0	**0.83**	0	0.86	0.0	0.85	0	0.79	0.0
F	Coverage	0.78	0.0	**0.83**	0.0	0.882	0	**0.875**	0.0	0.66	0	**0.92**	0	**0.43**	0	**0.75**	0.0
U	VI_LN	5.0	0.06	4.67*	0.1	1.22	0.1	2.18	0.06	3.66	0.05	1.7	0.03	-	-	-	-
R	Frac_LN	0.125	0.01	**0.33**	0.0	0.16	0.01	0.15	0.003	**0.84**	0.13	0.6	0.03	**0.014**	0.0	0.023	0.0
S	VI_IP	2.66	0.02	3.22	0.1	**0.52**	0.10	**0.68**	0.04	5.06	2.19	1.82	0.03	-	-	-	-
	Frac_IP	0.4	0.0	**0.32**	0.0	0.075	0.0	0.11*	0.0	0.03	0.0	0.42	0.0	**0.03**	0.0	-	-
	VI_CNM	4.57	0.0	3.53	0.0	1.58	0.0	**1.95**	0.0	3.45	0	-	-	-	-	-	-
	Frac_CNM	0.72	0.0	**0.78**	0.0	0.03*	0.0	0.103	0.0	0.17	0	-	-	-	-	-	-
	Time	1.61	0.0	5.18	0	31.2	0	35.6	0	115.16	0	641.4	0	4251.2	0	85596	0.0
S	CCF	0.99*	0.0	**0.86**	0	0.86	0	**0.92**	0	**0.95**	0	0.74	0	**0.95**	0.0	0.77	0.0
L	DD	0.723	0.0	0.64*	0	0.63*	0	0.46*	0	0.56	0	0.55*	0	**0.87**	0.0	0.68*	0.0
A	Coverage	**0.81**	0.0	0.82	0	**0.9**	0	0.84	0	**0.81**	0	0.84	0	0.07*	0	0.68	0.0
S	VI_LN	5.16*	0.07	**3.4**	0.1	**1.07**	0.08	**1.86**	0.3	**2.37**	0.22	**1.15**	0.07	-	-	-	-
H	Frac_LN	**0.223**	0.015	0.08*	0.0	**0.19**	0.0	0.036*	0.0	0.14*	0.03	**0.75**	0.045	0.01	0	**0.2**	0.0
B	VI_IP	4.07*	1.55	**2.20**	0.02	0.55	0.02	2.83	1.38	**2.31**	2.1	**1.72**	0.08	-	-	-	-
U	Frac_IP	0.22*	0.12	0.07*	0.0	0.09	0.0	0.143	0.07	**0.04**	0.02	**0.53**	0.0	0.02	0	-	-
R	VI_CNM	4.62*	0.0	**2.77**	0.0	**1.35**	0	2.22	0	**2.17**	0	-	-	-	-	-	-
N	Frac_CNM	**0.75**	0.0	0.56	0.0	0.06	0	0.03*	0	0.065*	0.0	-	-	-	-	-	-
	Time	37.2	10.7	270.9	2.9	312.5	8.44	355.4	16.13	1453.1	30.0	9225	1980	-	-	-	-
	CCF	0.992	0.0	0.44	0.01	0.76	0.0	0.56	0.007	0.87	0.0	0.47	0.02	-	-	-	-
	DD	**0.783**	0.0	**0.91**	0.0	**0.96**	0.0	0.7	0.0	0.53	0.003	**0.95**	0.0	-	-	-	-
X	Coverage	0.56	0.01	0.57	0.0	0.81	0.0	0.46	0.02	0.38	0.007	0.42	0.03	-	-	-	-
S	VI_LN	**4.9**	0.05	4.28	0.05	3.5	0.07	4.23	0.13	5.63	0.124	3.89	0.2	-	-	-	-
N	Frac_LN	0.028	0.0	0.32	0.004	0.07	0.0	0.37	0.018	0.143	0.03	0.06	0.0	-	-	-	-
	VI_IP	2.24	0.09	4.63	0.07	3.3	0.17	5.03	0.21	7.73	0.98	4.36	0.2	-	-	-	-
	Frac_IP	0.97	0.01	**0.32**	0.0	**0.33**	0.02	0.32	0.0	0.00	0.0	0.042	0.0	-	-	-	-
	VI_CNM	**4.56**	0.07	3.76	0.07	2.91	0.18	2.68	0.18	3.33	0.085	-	-	-	-	-	-
	Frac_CNM	0.22	0.02	0.43	0.01	**0.86**	0.04	0.21	0.011	**0.18**	0.01	-	-	-	-	-	-
	Time	21.9	0.2	273.6	1.8	323.08	13.8	358.7	15.4	1487.4	44.0	8608	273.7	-	-	-	-
	CCF	0.992	0.0	0.44	0.01	0.76	0.0	0.56	0.01	0.87	0.002	0.45	0.016	-	-	-	-
	DD	0.78	0.01	**0.91**	0.0	**0.96**	0.0	0.7	0.01	0.53	0.003	**0.95**	0.0	-	-	-	-
	Coverage	0.55	0.01	0.57	0.0	0.8	0.0	0.44	0.014	0.37	0.005	0.39	0.03	-	-	-	-
M	VI_LN	**4.9**	0.04	4.3	0.04	3.43	0.04	4.27	0.06	5.66	0.05	4.1693	0.3	-	-	-	-
D	Frac_LN	0.027	0.0	0.32	0.0	0.07	0.01	0.36	0.01	0.142	0.03	0.058	0.0	-	-	-	-
D	VI_IP	**2.2**	0.03	4.66	0.07	3.23	0.11	5.15	0.23	7.8	0.85	4.64	0.273	-	-	-	-
	Frac_IP	**0.98**	0.0	**0.32**	0.01	0.06	0.0	0.32	0.008	0.0	0.0	0.04	0.002	-	-	-	-
	VI_CNM	**4.56**	0.12	3.8	0.12	2.8	0.08	2.64	0.04	3.4	0.05	-	-	-	-	-	-
	Frac_CNM	0.2	0.01	0.324	0.02	0.83	0.05	**0.22**	0.008	**0.18**	0.02	-	-	-	-	-	-
F	Time	0.48	0.01	4.95	0.01	17.15	0.03	14.1	0.05	20.1	0.07	37.8	0.1	50.24	0.5	501	1.0
O	CCF	0.992	0.0	0.44	0.01	0.76	0.0	0.6	0.01	0.87	0.0	0.47	0.0	0.95	0.0	0.73	0.01
R	DD	0.77	0.01	**0.91**	0.0	**0.96**	0.0	0.7	0.0	0.53	0.0	**0.95**	0.0	0.84	0.0	**0.8**	0.01
E	Coverage	0.55	0.01	0.57	0.0	0.80	0.0	0.46	0.02	0.38	0.01	0.42	0.04	0.35	0.0	0.51	0.02
S	VI_LN	4.92	0.06	4.27	0.06	3.49	0.1	4.17	0.123	5.63	0.08	3.8	0.40	-	-	-	-
T	Frac_LN	0.028	0.0	0.32	0.0	0.07	0.0	**0.38**	0.019	0.144	0.03	0.06	0.0	0.013	0.0	0.012	0.0
F	VI_IP	2.32	0.17	4.68	0.06	3.3	0.06	4.93	0.24	8.4	0.05	4.27	0.4	-	-	-	-
I	Frac_IP	0.96	0.04	**0.32**	0.0	0.06	0.0	**0.33**	0.014	0.0	0.0	0.042	0.0	0.025	0.0	-	-
R	VI_CNM	4.58	0.17	3.78	0.06	2.89	0.15	2.64	0.115	3.5	0.16	-	-	-	-	-	-
E	Frac_CNM	0.22	0.02	0.21	0.03	0.85	0.07	**0.22**	0.014	**0.18**	0.0144	-	-	-	-	-	-

Table 3.2: Statistics of real world networks for various subset selection techniques. Here '-' represents not calculated as computationally too expensive and '*' represents the cases for which FURS & SlashBurn algorithms perform worst.

3.1.6 Conclusion

We proposed a novel representative subset selection technique namely FURS which selects a set of nodes retaining the inherent community structure. FURS greedily selected nodes with high degree centrality from different dense regions of the graph thereby spanning *most* or *all* the communities in the network. For this subset selection technique, we used the concept of node activation and node deactivation while retaining the topology of the graph. We compared FURS with state-of-the-art techniques like SlashBurn, Forest-Fire, Metropolis and Snowball Expansion sampling methodologies for various evaluation criteria including coverage, degree distribution, clustering coefficients, variation of information and fraction of communities covered. The subset generated by FURS can be efficiently used for community affiliation for unseen nodes in a network. This was shown in combination with a model based kernel spectral clustering technique (KSC) in Chapter 4.

3.2 Representative subsets for Big Data Learning using k-NN graphs[2]

Abstract—In this work, we propose a deterministic method to obtain subsets from big data which are a good representative of the inherent structure in the data. We first convert the large scale dataset into a sparse undirected k-NN graph using a distributed network generation framework that we propose in this work. After obtaining the k-NN graph we exploit the fast and unique representative subset (FURS) selection method [20, 123] to deterministically obtain a subset for this big data network. The FURS selection technique selects nodes from different dense regions in the graph retaining the natural community structure. We then locate the points in the original big data corresponding to the selected nodes and compare the obtained subset with subsets acquired from state-of-the-art subset selection techniques. We evaluate the quality of the selected subset on several synthetic and real-life datasets for different learning tasks including big data classification and big data clustering.

3.2.1 Introduction

In the modern era with the advent of new technologies and its widespread usage there is a huge proliferation of data. This immense wealth of data has resulted in massive datasets and has led to the emergence of the concept of Big Data. However, the choices for selecting a predictive model for Big Data learning is limited as only a few tools scale to large scale datasets. One way is to develop efficient learning algorithms which are fast, scalable and might use parallelization or distributed computing. Recently, a tool named Mahout (`http://www.manning.com/owen/`) was built which implemented several machine learning techniques for big data using a distributed Hadoop [4] framework. The other direction is sampling [124] and [125]. There are several machine learning algorithms which build predictive models on a small representative subset of the data [30, 20, 123, 13, 126] with out-of-sample extensions properties. This property allows inference for previously unseen part of the large scale data. The methods which belong to this class include kernel based methods, similarity based methods, prototype learning methods, instance based methods, manifold learning etc.

Sampling [127] is concerned with selection of points as a subset which can be used to estimate characteristics of the whole dataset. The main disadvantage of

[2]This section consists of Sections $1-6$ from Mall R., Jumutc V., Langone R., Suykens J.A.K., **"Representative Subsets For Big Data Learning using k-NN graphs"**, IEEE BigData, Washington D.C., U.S.A, 2014, pp. 37-42.

probabilistic sampling techniques is that every time the algorithm runs different subsets are obtained. It often results in large variations in the performance. Another disadvantage is that most probabilistic sampling techniques cannot capture some characteristic of the data like the inherent cluster structure unless the cluster information is available in advance. However, in case of real-life datasets this information is not known beforehand and is learnt by unsupervised learning techniques. In this work, we propose a framework to overcome these problems and select representative subsets that retain the natural cluster structure present in the data.

We first convert the big data into an undirected and weighted k-Nearest Neighbor (k-NN) [128, 129] graph where each node represents a data point and each edge represents the similarity between the data points. In this work, we propose a distributed environment to convert big data into this k-NN graph. After obtaining the k-NN graph we use the fast and unique representative subset (FURS) selection technique proposed in [123] and [20]. FURS selects nodes from different dense regions in the graph while retaining the inherent community structure. Finally, we map these selected nodes to the points in the original data. These points capture the intrinsic cluster structure present in the data.

We compare and evaluate the resulting subset with other sampling techniques like simple random sampling [124], stratified-random sampling [125] and a subset selection technique based on maximizing the Rènyi entropy criterion [92] and [30]. For classification, we use the subset to build a subsampled-dual least squares support vector machine (SD-LSSVM) model as proposed in [130] and use the out-of-sample extensions property to determine the class labels for points in the big data. For clustering we utilize the kernel spectral clustering (KSC) method proposed in [13]. We build the training model on the subset and again use the out-of-sample extension property of the model to infer cluster affiliation for the entire dataset. Figure 3.3 represents the flow chart of the steps undertaken.

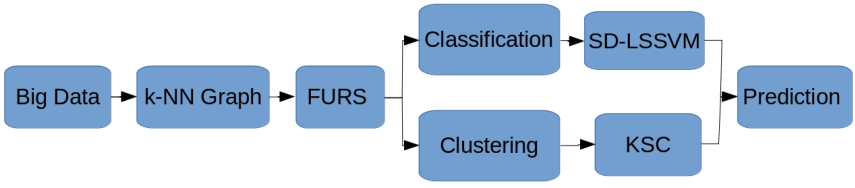


Figure 3.3: Steps involved in obtaining the subset from big data and its evaluation w.r.t. learning performance.

3.2.2 Distributed k-NN graph generation framework

In this section we describe a parallel approach for network generation from the kernel matrix. The kernel matrix is in general a full matrix and a full graph can be generated corresponding to the kernel matrix. However, most real-life datasets have underlying sparsity i.e. each point in the dataset is similar to only a few other points in the big data. Hence, we propose to use the k-NN graph [128] and [129] for representing the big data. Now, we will present a resilient way of handling big and massive datasets just by sequencing and distributing computations in a smart way.

Initial Setup

Our approach is based on an emerging Julia language (`http://julialang.org/`) and the model of the asynchronous management of co-routines [131]. To generate an undirected network where each point has connection with its top k most similar points we need a k-NN graph. This k-NN graph is obtained by sorting columns of the corresponding kernel matrix. The kernel matrix consists of similarity values between every pair of points in the data. Calculation of the entire kernel matrix for big data is not feasible or might be prohibitively expensive on a single machine or a supercomputer with enough storage and RAM. To resolve this problem we address the computation via a cluster-based approach.

Before proceeding with the distributed computational model one has to select a proper bandwidth for the Radial-Basis function (RBF) kernel which we are using to precompute the entries of the kernel matrix. There are several methods which have been proposed to tune the bandwidth [94, 96, 132] most of which are computationally expensive. Since the structure of the original data is not known in advance, we choose the Silverman's Rule of Thumb [133] which results in an approximate k-NN graph for the data. The bandwidth $\sigma \in \mathbb{R}^d$ used in the RBF kernel is computed as follows:

$$\sigma = \hat{\sigma} N^{-1/(d+4)}, \tag{3.3}$$

where $\hat{\sigma}$ is a mean standard deviation across all dimensions d and N is the total number of observations in dataset $\mathcal{D}$.

Kernel Matrix Evaluations

The computation of the kernel matrix is quite straightforward after obtaining σ and is given by:

$$\Omega = \begin{pmatrix} K(x_1, x_1) & \dots & K(x_1, x_N) \\ \vdots & \ddots & \vdots \\ K(x_N, x_1) & \dots & K(x_N, x_N) \end{pmatrix}, \tag{3.4}$$

where $K(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}$ is the Radial-Basis function. To compute Ω efficiently without loading the entire dataset one has to consider a batch cluster-based approach where for every node we load a batch subset $\mathcal{D}_p \subset \mathcal{D}$, $p \in \{1, \dots, P\}$ of data such that $\cup_{p=1}^{P} \mathcal{D}_p = \mathcal{D}$. The corresponding matrix slice is X_p. $\mu_p \in \mathbb{R}^d$ and $Var_p \in \mathbb{R}^d$ are the mean vector and variance vector of the data in set $\mathcal{D}_p$. We obtain an average $\mu_X = \frac{1}{P}\sum_{p=1}^{P} \mu_p$ and an average $Var_X = \frac{1}{P}\sum_{p=1}^{P} Var_p$, where X is the matrix representation for the dataset $\mathcal{D}$.

Finally, we obtain the dimension-wise standard deviation $\sigma_X = \sqrt{Var_X} \in \mathbb{R}^d$. To obtain $\hat{\sigma}$ we simply take an average across all dimensions d of σ_X as $\hat{\sigma} = \frac{1}{d}\sum_{i=1}^{d} \sigma_X^{(i)}$. The overall setup is a Map-Reduce and All-Reduce settings [134] and can be implemented using platforms like Hadoop [4] or Spark [135].

After obtaining σ we proceed with the batch-computation of the kernel matrix where for every node in a cluster we assign a matrix slice X_p over which we are performing calculations. For each batch p, we estimate $\Omega^{(p)}$ which consists of $\Omega_{ij}^{(p)}$, where $i \in \{1, \dots, N\}$, $p \in \{1, \dots, P\}$, $j \in \{m \times (p-1) + 1, \dots, m \times p\}$ and m is the batch size. To compute $\Omega^{(p)}$ we load the subset X_p first and then feed chunks of the entire dataset to construct rows of $\Omega^{(p)}$ which span index $i \in \{1, \dots, N\}$. After calculating the corresponding slice $\Omega^{(p)}$ of the kernel matrix we sort in ascending order the columns of $\Omega^{(p)}$ and pick indices corresponding to the top k values. The reduction step is performed by joining indices $j \in \{m \times (p-1) + 1, \dots, m \times p\}$ with the latter picked indices of k nearest neighbors for each j. By aggregating the tuples (j, k) and (k, j) one can obtain the edge list for a k-NN graph. This concept is explained in Figure 3.4.

Sparsity & Analysis of k-NN graphs

We generate k-NN graphs for different numbers of neighbors k as the amount of sparsity in data is not known in advance. In our experiments, we generate graphs $\forall k \in \{10, 100, 500\}$. For smaller values of k we obtain graphs that are sparse representations of the data. For small k fewer edges are obtained whereas

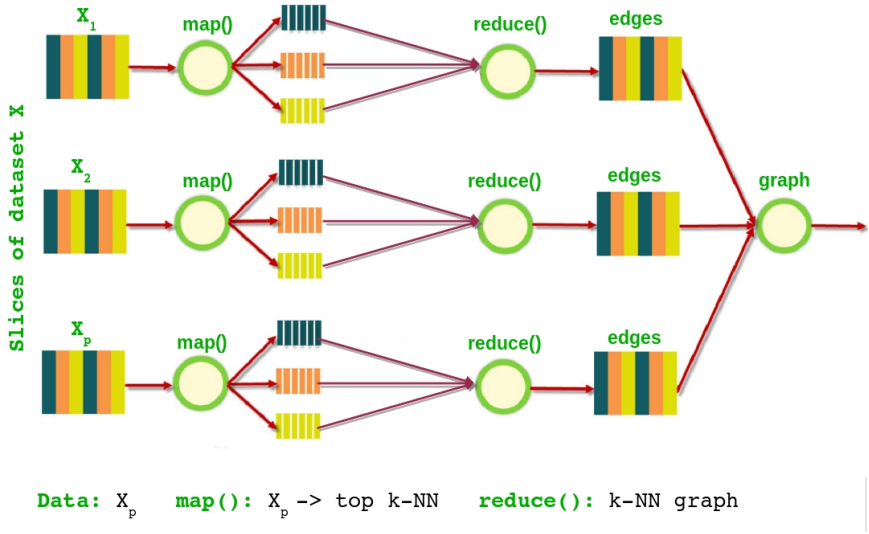


Figure 3.4: Map-Reduce setting for the distributed generation of an undirected and weighted k-NN graph.

for large values of k we obtain dense graphs. Since the graph is undirected the total number of edges in a k-NN graph is equal to $2k \times N$ where N is the total number of nodes in the network. The total number of connections in the densest graph would be $N \times (N-1)$ excluding self edges. Hence, the amount of sparsity in the graph can be mathematically be represented as: $Sparsity = 1 - \frac{2k \times N}{N \times (N-1)}$.

We need to only create a k-NN graph for the largest value of k. From this k-NN graph we can further obtain k-NN graphs for smaller values of k. We use a notion based on median degree to determine whether a sparse or a dense k-NN graph is a better representative of the original data. We calculate the median degree (m_k) for each k-NN graph and determine the number of nodes with degree $\geq m_k$. We choose m_k as it is central to the degree distribution and not influenced by outliers. The larger the number of nodes whose degree is greater than m_k for a k-NN graph the better is its representation for the big dataset. The rationale is that if a dataset has an underlying sparse representation then each point has fewer points in its vicinity. However, with large values of k in the k-NN graph we are enforcing additional connections between less similar points. As a result there will be more nodes with degrees smaller than m_k. Hence, a k-NN graph with large values of k is not a good representative for a sparse dataset. We also illustrate this in section 3.2.4.

Complexity analysis

In the computational complexity of the distributed k-NN graph generation, the most expensive part is the construction of the kernel matrix. The latter inherits the complexity of $O(N^2)$. After distributing the calculations we can achieve an almost linear speedup in terms of the number p of corresponding nodes/workers in a cluster as $O(N^2/p)$. The second important computationally expensive part is sorting of columns for $\Omega^{(p)}$ slices of the kernel matrix. Its complexity is $O(N \log N)$ which corresponds to the complexity of the merge-sort [136] method as a default choice for `sortperm` operation in Julia (`http://julia.readthedocs.org/en/latest/stdlib/sort/#sorting-algorithms`). After taking into account the total number of nodes p in a cluster and the total number of columns N we have to process the computational complexity grows as $O(\frac{N^2 \log N}{p})$. The final computational complexity is given as $O(N^2(1+\log N)/p)$.

Machine Configuration for Experiments

In our experiments for generation of the k-NN graph we use a computer with 64-bit, 132 Gb RAM, 2.2GHz processor and 40 cores. We utilize all these 40 cores for parallel computations. Once we have obtained the subset, the rest of the learning procedures like classification and clustering are performed on a PC machine with Intel Core i7 CPU and 8 GB RAM under Matlab 2013b. This is to showcase the power of model based techniques [13, 20, 130] for big data learning.

3.2.3 FURS for Weighted Graphs

The FURS selection technique is described in detail in 3.1.3.

3.2.4 Classification Experiments

There are several supervised learning techniques [30, 130] which build a predictive model on a small representative subset of the data and use its out-of-sample extension property to obtain the required class information about the unseen part of the data. In this work, we showcase the effectiveness of representative subset selection for the subsampled-dual LSSVM (SD-LSSVM) model proposed in [130].

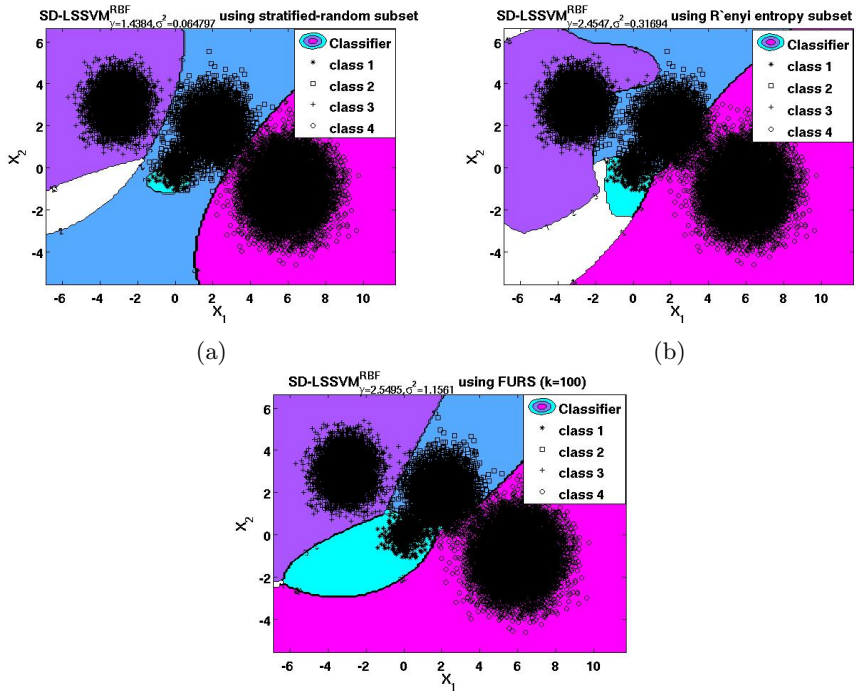


Figure 3.5: Generalization results using different subset selection techniques for the synthetic 4G dataset. We can observe from Figure 3.5a and 3.5b that the stratified-random and stratified-Rènyi entropy based selection technique cannot result in good generalization for the classification problem. However, the FURS selection technique (on k-NN graph with $k = 10$) performs much better as depicted in Figure 3.5c. This can also be observed from the results in Table 3.4.

Experimental setup

We conduct experiments on 2 synthetic datasets. The first synthetic dataset (4G) comprises 4 Gaussians with $20,000$, $5,000$, $3,000$ and 500 2-dimensional points respectively. The Gaussian with the smallest density is considered negative $(-)$ class while the other 3 Gaussians form the positive $(+)$ class. There is a small overlap between the positive and negative class. The second synthetic dataset (5G) consists of 3 Gaussians with $50,000$, $20,000$, $10,000$ points. They form the $+$ class. There are 2 separate Gaussian of 500 points which form the $-$ class. We also performed experiments on 3 real-life datasets obtained from

the UCI repository [73]. Table 3.3 gives a summary of the datasets used in the experiments.

Dataset	N	d	N_+	N_-	M	M_+	M_-	N_{test}
4G	28,500	2	28,000	500	114	112	2	28,386
5G	81,000	2	80,000	1,000	324	320	4	80,676
Magic	19,020	11	12,363	6657	100	65	35	18,920
Shuttle	58,000	9	45,820	12,180	100	79	21	57,900
Skin	245,057	3	193,595	51,462	500	395	105	244,557

Table 3.3: Dataset description and usage. N_+ and N_- represents the number of points in the $(+)$ and $(-)$ class respectively. We select as few points (M) as possible while still retaining good generalization. M_+ and M_- represents the number of points in the selected subset from the $(+)$ and $(-)$ class respectively. We maintain the ratio of $(+)$ to $(-)$ class in the selected subset.

For each dataset we run the Map-Reduce procedure once and from a big k-NN graph obtain graphs for different values of k, *i.e.* $k \in \{10, 100, 500\}$. Since FURS is a deterministic algorithm, we run the subset selection only once on the k-NN graph emerged from Map-Reduce process. To leverage effects of the unbalanced datasets we perform all subset selections separately w.r.t. each class. We generate separate k-NN graphs for each class and select points from each of these classes. The total number of data points selected from each class by the different subset selection techniques is given in Table 3.3. We compare the performance of the subset selected by FURS algorithm with stratified-random [125] and stratified-Rènyi entropy-based [30] subset selection methodologies.

In all our experiments regarding classification and supervised learning we use a 2-step procedure for tuning the hyperparameters of the SD-LSSVM model [130], such as γ and the bandwidth σ of RBF kernel. This procedure consists of Coupled Simulated Annealing (CSA)[93] initialized with 5 random sets of parameters for the first step and the simplex method [137] for the second step. After CSA converges to some local minima we select the tuple of parameters that attains the lowest error and start the simplex procedure to refine our selection. On every iteration step for CSA and simplex method we perform a 10-fold cross-validation. We run SD-LSSVM model 50 times to leverage effects of randomizations and test the model on nearly test set for predictions. We average the error results and report it along with the standard deviations. Figure 3.5 depicts the results of different subset selection technique on the 4G synthetic dataset.

Numerical results

In this subsection we present the generalization errors obtained for several public and artificial datasets described in Table 3.3. It can be observed from Table 3.4 that for most datasets FURS algorithm results in the improved classification rates and lower standard deviations. We can observe that for some difficult artificial datasets, like 4G and 5G stratified-random (SR) sampling and stratified-Rènyi entropy (SRE) based subset selection techniques might fail because classes are highly unbalanced and selected subset may contain outliers or boundary cases. On the other hand for some datasets, like Magic, stratified-random sampling is performing best.

Dataset	SR	SRE	$\text{FURS}_{k=10}$	$\text{FURS}_{k=100}$	$\text{FURS}_{k=500}$
	Generalization or Test Error ± Standard Deviation				
4G	0.395±0.235	0.898±0.375	**0.252±0.023**	**0.331±0.063**	0.414±0.070
5G	0.264±0.225	0.298±0.142	0.358±0.296	**0.082±0.016**	**0.086±0.009**
Magic	**25.32±3.913**	**28.44±4.446**	33.07±2.076	31.05±4.143	36.13±3.062
Shuttle	2.437±1.104	2.330±0.958	4.223±0.998	**1.482±0.600**	**1.980±0.715**
Skin	0.578±0.387	**0.254±0.078**	3.277±1.133	**0.494±0.078**	0.772±0.080

Table 3.4: Averaged generalization errors along with their standard deviations for SD-LSSVM model. Generalization errors are expressed as percentage. The best and second-best results are highlighted.

In general, we can observe improved generalization errors for $\text{FURS}_{k=100}$ and $\text{FURS}_{k=500}$. The SRE based selection technique performs better in some cases like Magic and Skin dataset. This might indicate that we need to increase size of subset for FURS selection technique for these datasets. The performance of SRE can be explained by the characteristics of the Rènyi entropy criterion which selects points uniformly from the dataset. Figure 3.6 shows the variations in the error estimations of the different subset selection methods for the Shuttle dataset. From Figure 3.6 we can observe that the $\text{FURS}_{k=10}$ performs worse indicating that a dense k-NN graph is better representative of the original dataset. We can also observe that the FURS selection method has lower standard deviations in the error estimations.

3.2.5 Clustering Experiments

A powerful model based clustering technique with out-of-sample extensions property is the kernel spectral clustering (KSC) [13] method. We use it in our clustering experiments.

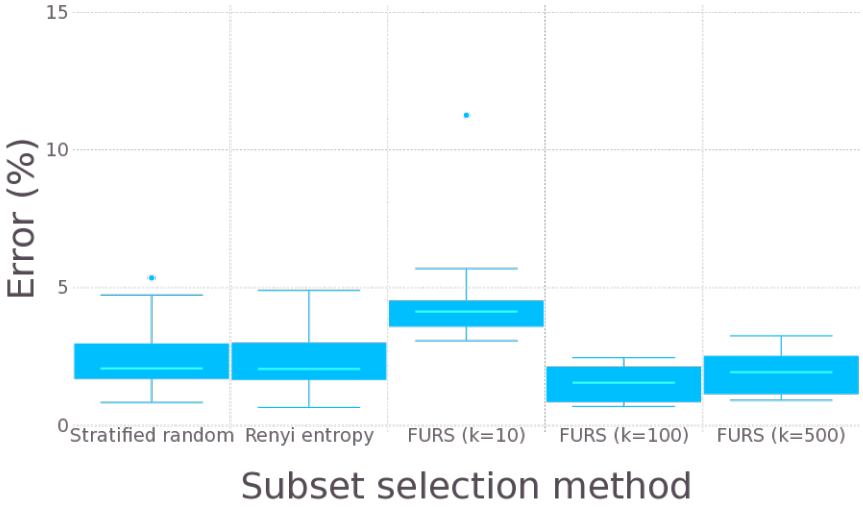


Figure 3.6: Error estimations for classification by different subset selection techniques on Shuttle dataset. $\text{FURS}_{k=100}$ results in the best mean performance with the lowest variance.

Experimental Setup

We conducted clustering experiments on the 2 synthetic datasets (4G and 5G) which we also used for supervised learning. However, we generate the k-NN graph for the whole data in the unsupervised case followed by applying the FURS algorithm on the whole network. We also experimented on 5 real-life datasets, 3 (House, Mopsi Finland and KDDCupBio) of which are obtained from `http://cs.joensuu.fi/sipu/datasets/` and others (Gas Sensory Array Drift (Batch) and Power Consumption (Power)) is obtained from UCI repository [73]. In our experiments the training set and validation set are obtained by the subset selection technique. We use the same size for the training and validation sets (M) and the same kernel parameter σ for different subset selection technique. We generate k-NN graphs with different k ($k \in \{10, 100, 500\}$).

We use the median degree (m_k) to estimate the nodes from which the representative set can be selected by FURS algorithm (section 3.1.3). The idea of calculating the number of nodes whose degree is greater than m_k helps to determine whether a sparse or dense representation is better for the k-NN graph as mentioned in section 3.2.2 and depicted in Table 3.5. In Table 3.5 $\mathcal{S}_k$ represents the set containing nodes whose degree is greater than m_k. From Table

3.5 we observe that $|\mathcal{S}_{k=10}|, |\mathcal{S}_{k=100}|$ and $|\mathcal{S}_{k=500}|$ represents the cardinality or the number of points whose degree is above median degree m_k for the respective k-NN graph. Table 3.5 also provides a summary of the datasets.

Dataset	N	d	M	N_{test}	$\lvert\mathcal{S}_{k=10}\rvert$	$\lvert\mathcal{S}_{k=100}\rvert$	$\lvert\mathcal{S}_{k=500}\rvert$
4G	28,500	2	100	28,500	15,852	**19,760**	*19,384*
5G	81,000	2	810	81,000	45,328	**51,304**	*47,156*
House	34,112	3	342	34,112	**20,556**	*19,400*	18,864
Mopsi Finland (MF)	13,467	2	135	13,467	*7,202*	**7,388**	6,231
Batch	13,910	128	139	13,910	**7,171**	*5,534*	4,649
KDDCupBio	145,751	74	1,458	145,751	**64,034**	*60,307*	55,700
Power	2,049,280	7	2,050	2,049,280	*1,096,668*	**1,292,203**	1,083,454

Table 3.5: Dataset description and usage. We use the entire dataset as test set for clustering. We select 1% of the total points as training set and validation set (M). For each dataset we bold and emphasize the sets from which we obtain the training set and validation set respectively. The Power dataset results 20Gb size k-NN graph for $k = 500$.

We compare the FURS selection technique with simple random sampling and the Rènyi entropy based selection method. We run FURS only once whereas we perform other subset selection methods 10 times. We compare clustering results based on several internal (Silhouette (SIL), Davies-Bouldin (DB)) [45] and external quality (ARI, VI) [45] metrics. Figure 3.7 reflects the clustering results for different subset selection technique on 4G synthetic dataset.

Results & Analysis

Table 3.6 provides a detailed comparison of the FURS selection technique with other subset selection methods on various internal and external quality metrics. From Table 3.6 we observe that the FURS algorithm completely outperforms other techniques on the two synthetic datasets. For the Batch dataset, FURS performs best w.r.t. internal quality metrics like SIL and DB while Rènyi entropy based subset selection technique gives best results w.r.t. VI. Higher values of SIL criterion and lower values of DB index represent better quality clusters. From Table 3.6 we observe that for the large scale real-world datasets (House and Mopsi Finland) where the ground truth is unknown, the FURS selection technique performs better than random sampling and Rènyi entropy based subset. For big datasets (KDDCupBio and Power) we use the DB quality measure as the silhouette measure is computationally very expensive. Random sampling technique gives better results on KDDCupBio but it also shows large variations in the results.

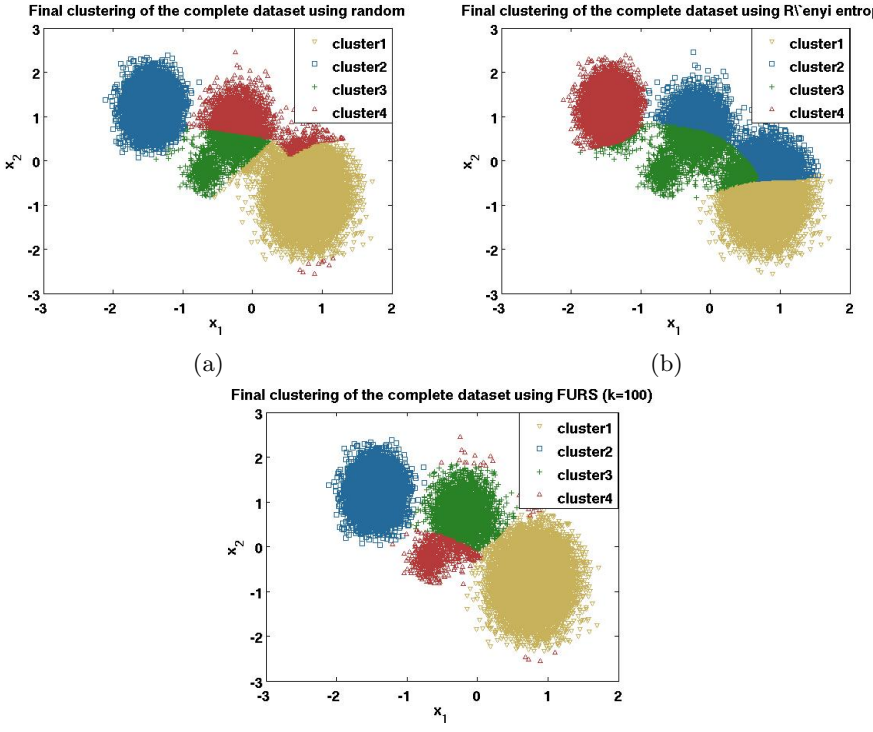


Figure 3.7: Comparison of clustering performance of KSC model using the same kernel parameter ($\sigma = 0.188$) but different subsets for building the model. We can observe poor generalizations for random and Rènyi entropy based selection techniques in 3.7a and 3.7b. However, the FURS algorithm results in good generalization performance as depicted in 3.7c and Table 3.6.

3.2.6 Conclusion

We proposed a method to obtain representative subsets of the big data for model based learning techniques. We proposed to convert the big data into a k-NN graph using a distributed framework and then selected the required representative subset using the FURS algorithm. The selected subset retained the natural cluster structure of the data. We illustrated the effectiveness of this selected subset for big data learning.

Dataset	**Random**				**Rènyi entropy**				**FURS**			
	ARI	VI	DB	SIL	ARI	VI	DB	SIL	ARI	VI	DB	SIL
4G	0.743 ± 0.157	0.485 ± 0.213	0.804 ± 0.167	0.667 ± 0.059	0.402 ± 0.236	1.223 ± 0.412	2.909 ± 2.820	0.594 ± 0.148	**0.985**	**0.097**	**0.616**	**0.765**
5G	0.664 ± 0.233	0.638 ± 0.402	2.011 ± 0.998	0.649 ± 0.049	0.471 ± 0.143	0.911 ± 0.263	0.885 ± 0.276	0.601 ± 0.054	**0.988**	**0.081**	**0.808**	**0.764**
Batch	**0.064 ± 0.029**	2.633 ± 0.099	4.287 ± 0.764	0.503 ± 0.066	0.006 ± 0.008	**1.978 ± 0.115**	3.969 ± 0.783	0.460 ± 0.134	0.039	2.452	**1.868**	**0.669**
	DB		**SIL**		**DB**		**SIL**		**DB**		**SIL**	
House	0.612 ± 0.154		0.679 ± 0.073		0.507 ± 0.028		0.579 ± 0.006		**0.407**		**0.751**	
Mopsi Finland	0.897 ± 0.935		0.824 ± 0.085		**0.526 ± 0.223**		0.886 ± 0.010		0.568		**0.920**	
	DB				**DB**				**DB**			
KDDCupBio	**2.932 ± 1.205**				4.233 ± 1.82				3.883			
Power	2.558 ± 1.374				2.0619 ± 0.760				**1.921**			

Table 3.6: Comparison of FURS algorithm with other subset selection technique w.r.t. various quality metrics.

Chapter 4

Large Scale Community Detection

This chapter comprises previously published articles including:

1) Mall R., Langone R., Suykens J.A.K., **"Kernel Spectral Clustering for Big Data Networks"**, Entropy, Special Issue: Big Data, 15(5), May 2013, pp. 1567-1568.

2) Mall R., Langone R., Suykens J.A.K., **"Self-Tuned Kernel Spectral Clustering for Large Scale Networks"**, in Proc of IEEE International Conference on Big Data (IEEE BigData), Santa Clara, USA, Oct 2013, pp. 385-393.

3) Mall R., Langone R., Suykens J.A.K., **"Multilevel Hierarchical Kernel Spectral Clustering for Real-Life Large Scale Complex Networks"**, PLoS ONE, 9(6):e99966, Jun 2014.

Keywords—kernel spectral clustering; community detection; out-of-sample extensions; angular similarity; hierarchical community detection; self-tuned spectral clustering; parameter-free spectral clustering;

4.1 Kernel Spectral Clustering for Big Networks[1]

Abstract — We show the feasibility of utilizing the Kernel Spectral Clustering (KSC) method for the purpose of community detection in big data networks. KSC employs a primal-dual framework to construct a model. It results in a powerful property of effectively inferring the community affiliation for out-of-sample extensions. The original large kernel matrix cannot fit into memory. Therefore, we select a smaller subgraph which preserves the overall community structure to construct the model. It makes use of the out-of-sample extension property for community membership of the unseen nodes. We provide a novel, memory and computationally efficient model selection procedure based on angular similarity in the eigenspace. We demonstrate the effectiveness of KSC on large scale synthetic networks and real world networks like the YouTube network, a road network of California and the Livejournal network. These networks contain millions of nodes and several million edges.

4.1.1 Introduction

The problem of community detection has also been framed as graph partitioning and graph clustering [138] and has received a lot of attention lately [39, 103, 38, 104, 106, 42, 105, 36, 46, 48, 139, 47]. Among the myriad variety of algorithms for community detection, the KSC method is related to the spectral clustering methods [48, 46, 139, 47]. Spectral clustering methods are standard techniques for graph partitioning. The underlying model is based on eigen-decomposition of the Laplacian matrix derived from the affinity matrix of the nodes in the community. The major drawback of these spectral clustering methods is the construction of the large affinity matrix ($N \times N$), where N is the number of nodes in the network which requires to calculate similarity between every pair of nodes in the network. As the size of the network increases, computing $O(N^2)$ and storing this affinity ($O(N \times N)$) matrix becomes in-feasible.

Recently, a spectral clustering formulation based on weighted kernel PCA with primal-dual framework was proposed in [13]. The formulation resulted in a model with a powerful property of inferring community memberships for out-of-sample nodes. The model is built on a small subset of the big data network which captures the inherent community structure. One then solves an affordable eigenvalue problem on this smaller subgraph and use the out-of-sample extension property. The KSC method was extended for community detection by [105].

[1]This section consists of Sections $1 - 6$ from Mall R., Langone R., Suykens J.A.K., **"Kernel Spectral Clustering for Big Data Networks"**, Entropy, SI: Big Data, 15(5), May 2013, pp. 1567-1568.

However, the two important steps, the subset selection and the model selection proposed by [105] are computationally expensive and memory inefficient. The two major contributions in this work are:

- We propose a novel, memory and computationally efficient model selection technique to tune the model parameter k using angular similarity in the eigenspace between the projected vectors of the nodes in validation set.
- We show that kernel spectral clustering is applicable for community detection in big data networks.

The authors in [105] utilized Modularity [37] for the purpose of model selection. However, they need to keep a square, non-sparse $N \times N$ matrix for validation of the model using Modularity. This makes the approach in-feasible when the networks are large scale. We provide a novel model selection technique based on angular similarity between the projected vectors in the eigen-spectrum. The authors in [105] used Expansion sampling [75] technique for subset selection. The method is based on the principle of expander graphs and optimizes the expansion factor function. However, the method is computationally expensive as shown in [123] and stochastic by nature. Therefore, for the given big data network, for each iteration of Expansion sampling the obtained subgraph would be different. We use the Fast and Unique Representative Subset (FURS) selection technique [123] for obtaining a representative subset of the big data network. The method is computationally less expensive and is a better representative of the community structure than most of the state-of-the-art sampling techniques like SlashBurn [74], Metropolis [76], Snowball Expansion [75], Random Node [77] sampling etc. as shown through extensive comparisons in [123]. The KSC method was recently extending to evolving networks as described in [22].

In this work, all the graphs considered are unweighted and undirected unless otherwise specified. All the experiments were performed on a machine with 12Gb RAM, 2.4 GHz Intel Xeon processor. The maximum size of the kernel matrix that can be stored in the memory of our PC is 5000×5000. The approach which we present can be easily implemented in a distributed environment as well and can handle big data networks.

4.1.2 Kernel Spectral Clustering

General Background

Spectral clustering uses the information contained in the affinity matrix to detect structures in the given network. In case of data points, the similarity between

the points is measured with respect to the mutual distance (e.g. Euclidean, cosine, RBF distance) between the points. Thus, the obtained similarity matrix can be considered as a weighted graph where each point has a certain extent of similarity with other points in the dataset. For our case, we already have a weighted graph A, where $A_{ij} = 1$ if $(v_i, v_j) \in E$ else $A_{ij} = 0$. So in our case the spectral clustering method can be applied directly. However, for the KSC method, we need to build a graph over the network to represent the similarity between the nodes in a kernel based framework.

The spectral graph theory can provide insights into several properties of the graph like partitions existing in the graph. These insights are obtained by studying the eigen-spectrum of the Laplacian matrix [48], [46], [49]. Several Laplacian matrices exist including the unnormalized Laplacian defined as $L = D - A$, the symmetric normalized Laplacian $L_{SYM} = D^{-1/2}LD^{-1/2} = \mathrm{I}_N - D^{-1/2}AD^{-1/2}$, the non-symmetric normalized Laplacian $L_{RW} = D^{-1}L = \mathrm{I}_N - D^{-1}A$. This Laplacian is related to the random walk on the graph, hence denoted by L_{RW}. For a random walk Laplacian, clustering can be understood as detecting the partitions such that a random walker will spend maximum time in that partition with a few jumps to other clusters. This minimizes the transition probability between the clusters. The transition matrix of the random walker on the graph can be obtained by normalizing the adjacency matrix A such that the sum of its row ends up being 1. Thus the transition matrix can be depicted as $P = D^{-1}A$, where the entry P_{ij} represents the probability of a jump from node v_i to node v_j. The corresponding eigenvalue problem becomes $Pr = \xi r$, and was shown to be similar to dual problem of a constrained optimization problem typical of least squares support vector machines (LSSVM) [8].

Primal-dual formulation

The KSC method is described by a primal-dual framework. The model is determined during the training phase and the parameter of the model i.e. k (number of clusters) is estimated during the validation stage. Finally, the model is tested on the test data to provide community affiliation to the unseen nodes.

In case of networks, the training data comprises the adjacency list of all the vertices $v_i \in X_{tr}$, where X_{tr} represents the set of nodes used for training the model. Let the cardinality of the set X_{tr} be N_{tr}. Big data networks can be stored into memory in sparse format as these networks are highly sparse. However, if the graph does not fit in the main memory, it can be split into blocks of adjacency lists and stored on the hard disk. We can then iterate over these blocks by selecting the adjacency lists corresponding to the nodes in the training set X_{tr}. This set of adjacency lists can be efficiently stored in the memory as

real world networks are highly sparse and there are very few connections for each node $v_i \in X_{tr}$. The maximum length of the adjacency list of a node in the training data can be equal to N, when the node is connected to all the other nodes in the network. During the test phase, the cluster memberships for each of the unseen nodes can be predicted by uploading the adjacency list of the test node in the memory and using the out-of-sample extension property of the model. This makes the approach feasible for big data networks unlike approaches which require the entire graph to be stored in the main memory. Moreover, the computational complexity of our approach is dominated by the time required to construct the kernel matrix. It is given by $O(N_{tr}^2 N)$, i.e. the time required to calculate similarity between sparse adjacency lists of the nodes in the training data.

Given X_{tr} training nodes $\mathcal{D} = \{x_i\}_{i=1}^{N_{tr}}$, $x_i \in \mathbb{R}^N$ and $v_i \in X_{tr}$. Here x_i represents the adjacency list of the ith training node and the number of nodes in the training set is N_{tr}. Given $\mathcal{D}$ and the number of communities k, the primal problem of the spectral clustering via weighted kernel PCA is formulated as follows [13]:

$$\begin{aligned} \min_{w^{(l)}, e^{(l)}, b_l} \quad & \frac{1}{2}\sum_{l=1}^{k-1} w^{(l)\intercal} w^{(l)} - \frac{1}{2N}\sum_{l=1}^{k-1} \gamma_l e^{(l)\intercal} D_\Omega^{-1} e^{(l)} \\ \text{such that} \quad & e^{(l)} = \Phi w^{(l)} + b_l 1_{N_{tr}}, l = 1, \ldots, k-1 \end{aligned} \tag{4.1}$$

where $e^{(l)} = [e_1^{(l)}, \ldots, e_{N_{tr}}^{(l)}]^\intercal$ are the projections onto the eigenspace, $l = 1, \ldots, k-1$ indicates the number of score variables required to encode the k communities, $D_\Omega^{-1} \in \mathbb{R}^{N_{tr} \times N_{tr}}$ is the inverse of the degree matrix associated to the kernel matrix Ω. Φ is the $N_{tr} \times d_h$ feature matrix, $\Phi = [\phi(x_1)^\intercal; \ldots; \phi(x_{N_{tr}})^\intercal]$ and $\gamma_l \in \mathbb{R}^+$ are the regularization constants. We note that $N_{tr} \ll N$ i.e. the number of nodes in the training set is much less than the total number of nodes in the big data network. The kernel matrix Ω is the obtained by calculating the similarity between the adjacency list of each pair of nodes in the training set. Each element of Ω, denoted as $\Omega_{ij} = K(x_i, x_j) = \phi(x_i)^\intercal \phi(x_j)$ is obtained by calculating the cosine similarity between the adjacency lists of x_i and x_j which has been shown to be an effective similarity measure for large scale data [140]. Thus, $\Omega_{ij} = \frac{x_i^\intercal x_j}{\|x_i\| \|x_j\|}$ and can be calculated efficiently using notions of set unions and intersections. This corresponds to using a normalized linear kernel function $K(x, z) = \frac{x^\intercal z}{\|x\| \|z\|}$ [8]. The clustering model is then represented by:

$$e_i^{(l)} = w^{(l)\intercal} \phi(x_i) + b_l, i = 1, \ldots, N_{tr} \tag{4.2}$$

where $\phi : \mathbb{R}^N \to \mathbb{R}^{d_h}$ is the mapping to a high-dimensional feature space d_h,

b_l are the bias terms, $l = 1, \ldots, k-1$. However, for big data networks we can utilize the explicit expression of the underlying feature map and $d_h = \mathbb{R}^N$. The projections $e_i^{(l)}$ represent the latent variables of a set of $k-1$ binary cluster indicators given by $\text{sign}(e_i^{(l)})$ which can be combined with the final groups using an encoding/decoding scheme. The dual problem corresponding to this primal formulation is:

$$D_\Omega^{-1} M_D \Omega \alpha^{(l)} = \lambda_l \alpha^{(l)} \tag{4.3}$$

where M_D is the centering matrix which is defined as $M_D = \text{I}_{N_{tr}} - (\frac{1}{1_{N_{tr}}^\intercal D_\Omega^{-1} 1_{N_{tr}}})(1_{N_{tr}} 1_{N_{tr}}^\intercal D_\Omega^{-1})$. The $\alpha^{(l)}$ are the dual variables and the kernel function $K : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ plays the role of similarity function. This dual problem is closely related to the random walk model.

Encoding/decoding scheme

In ideal situations when the communities are non-overlapping, we will obtain k well separated clusters and the matrix $D^{-1} M_D \Omega$ has $k-1$ piece-wise constant eigenvectors. This is because the multiplicity of the largest eigenvalue i.e. 1 is $k-1$ as depicted in [49]. In the eigenspace every cluster A_p, $p = 1, \ldots, k$ is a point and is represented with a unique codebook $c_p \in \{-1, 1\}^{k-1}$. To obtain this codebook $\mathcal{CB} = \{c_p\}_{p=1}^k$ we transform the rows of the projected vector matrix obtained from the training data by binarizing it i.e. $[\text{sign}(e^{(1)}), \ldots, \text{sign}(e^{(k-1)})]$. However, in real world networks the communities do overlap and we have nearly piece-wise constant eigenvectors. The codebook set $\mathcal{CB}$ is also obtained by selecting the top k most frequent codebook vectors. Due to the centering matrix M_D the eigenvectors have zero mean and the optimal threshold for binarizing the eigenvectors is self-determined (equal to 0). Taking into account that the first eigenvector $\alpha^{(1)}$ already provides a binary clustering the number of score variables needed to encode k clusters is $k-1$. The decoding scheme consists of comparing the cluster indicators obtained in the validation/test stage with the codebook and selecting the nearest codebook based on Hamming distance. This scheme corresponds to the ECOC decoding procedure [141] and is used in out-of-sample extensions as well. The proposed extension is based on the score variables which correspond to the projections of the mapped out-of-sample points onto the eigenvectors found in the training stage. The cluster indicators can be obtained by binarizing the score variables for the out-of-sample points as follows:

$$\text{sign}(e_{test}^{(l)}) = \text{sign}(\Omega_{test} \alpha^{(l)} + b_l 1_{Ntest}) \tag{4.4}$$

where $l = 1, \ldots, k-1$, Ω_{test} is the $N_{test} \times N_{tr}$ kernel matrix evaluated using

the test points with entries $\Omega_{test,ri} = K(x_r^{test}, x_i)$, $r = 1, \ldots, N_{test}$ and $i = 1, \ldots, N_{tr}$. Here N_{test} represents the number of nodes in the test set. This natural extension to out-of-sample points is the main advantage of the kernel spectral clustering framework. In this way the clustering model can be trained, validated and tested in an unsupervised learning procedure. If the Ω_{test} cannot fit the memory then we divide the test set into blocks and perform the testing operations iteratively on a single computer. However, this operation can easily be performed in parallel on a distributed environment.

4.1.3 Model Selection by means of Angular Similarity

Model selection is a crucial step in KSC. In order to obtain correct cluster parameters for the model, we propose a novel and computationally efficient mechanism using the concept of angular similarity. In [105], the authors used Modularity as a model selection criterion. However, the validation matrix required for Modularity calculation can blow up as the size of the network becomes large and might be in-feasible to store into memory. So, we need to work with the sparse adjacency lists of the nodes in the validation set instead of creating a square validation matrix. Since we are using the cosine similarity metric to estimate each element of the kernel matrix Ω, the model is free of a tuning parameter σ unlike the original formulation of KSC [13] when using a Gaussian RBF kernel. Thus, the only parameter to be determined for our procedure is the number of clusters k in the network.

We propose a criterion *Balanced Angular Fit* for estimating the number of clusters $k > 2$ in the network. This criterion exploits the projections of the training and validation nodes in the eigenspace. For a given value of $k > 2$, we estimate the cluster memberships of the nodes in the training set X_{tr} based on the codebook $\mathcal{CB}$. We assign each training node to the cluster corresponding to the codebook vector for which its Hamming distance is minimum. Thus, for a given value of $k > 2$ we can obtain the clustering $\Delta = \{C_1^{tr}, \ldots, C_k^{tr}\}$, where C_i contains the set of training nodes belonging to the ith cluster. We calculate the clustermean (μ_i) for each cluster w.r.t. to the projections of the training nodes belonging to the ith cluster in the eigenspace.

Once we obtain the clustermeans for all the clusters, we use the projections of the validation nodes in the eigenspace. The idea is to calculate the angular similarity between the projection of each validation node and each of the clustermeans. The clustermean which makes the least angle with the projection of the validation node is the closest or most similar for that node. Thus, we assign the cluster corresponding to that clustermean to the given validation node. For each validation node ($valid_i$), we want to obtain $\max_j \cos(\theta_{j,valid_i})$,

where

$$cos(\theta_{j,valid_i}) = \frac{\mu_j^\intercal e_{valid_i}}{\|\mu_j\|\|e_{valid_i}\|}, j = 1, \ldots, k. \tag{4.5}$$

The closer the projection of the validation node is to that of a given clustermean, the smaller the angle it has with it and the larger the cosine value of that angle. So, for each validation node we have the cluster to which it is assigned along with the maximum cosine similarity value. We maintain the maximum cosine similarity value for each validation node in a dictionary $MaxSim$. The dictionary is maintained as $MaxSim(valid_i) = \cos(\theta_{j,valid_i})$, where $\cos(\theta_{j,valid_i}) = \max_j \cos(\theta_{j,valid_i}), j = 1, \ldots, k$ is the maximum cosine similarity value for the validation node $(valid_i)$. The obtained clustering is $\Delta_{valid} = \{C_1^{valid}, \ldots, C_k^{valid}\}$, where C_i^{valid} contains the set of validation nodes belonging to cluster i. We now define *Balanced Angular Fit* (BAF) as:

$$BAF(k) = \sum_{i=1}^{k} \sum_{valid_j \in Q_i} \frac{1}{k} . \frac{MaxSim(valid_j)}{|C_i^{valid}|}. \tag{4.6}$$

The BAF simply sums up the cosine similarity values of all the validation nodes belonging to each cluster divided by the cardinality of that cluster. It then divides this overall value by the total number of clusters k in the network. The range of values that BAF can take is $[-1, 1]$. This is because the maximum similarity value that a validation node can have w.r.t. a clustermean is 1 (i.e. the angle between them is zero). Because we sum up the cosine similarity values of all the validation nodes in a cluster and divide it by the cardinality of that cluster, the metric is inherently balanced and this fraction cannot exceed one. This process is repeated for each cluster and in order to normalize the metric we divide it by the number of clusters in the network. In the worst case scenario, when all the validation nodes are wrongly assigned to clusters with whose clustermean it makes an angle of π, the BAF can end up being -1. However, we observed empirically that the BAF value ranges between $[0, 1]$ in all our experiments. Higher values of BAF indicates better graph partitions found in the validation set. To estimate the parameter k, we iteratively increase the value of k from 3 till a maximum value and select the value of k for which the BAF is maximum. The BAF will not work for $k = 2$ as the clustermean obtained are one-dimensional, one value is positive and other is negative. The projections of validation nodes are also one-dimensional and either positive or negative and thus the minimum angle between the projections of the validation nodes and the best clustermean will always be zero. Thus, the BAF value will turns out to be 1. The BAF measure can not just be used as a model selection criterion but can also serve the purpose of an evaluation metric for testing the quality of the clustering. There can be several values of k for which the BAF

values can be high. If the interval between these values of k is large then it can be identified with the underlying hierarchy in the network.

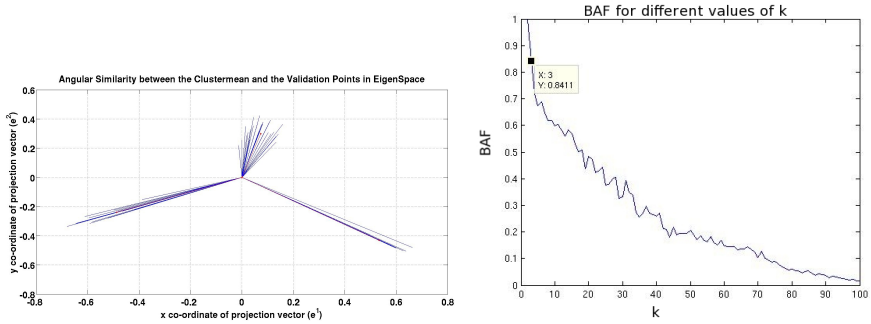


Figure 4.1: Model Selection for KSC based on BAF

In Figure 4.1, the red colored '*' marked points represent the clustermeans in 2-D projected eigenspace where the number of cluster k in the network is equal to 3. The projections of the validation nodes are represented as blue colored '.' points in 2-D. From Figure 4.1, one can easily conclude the presence of three clusters and the same can be inferred by BAF. We can observe that angle between the projections of the validation points and the corresponding best clustermean is quite small. This is also inferred from the high BAF value of 0.8411 for $k = 3$. We also observe that the nodes in the validation set whose projections are in the first quadrant have maximum deviation from their clustermean which affects the overall BAF value. From the figure on the right we observe that for $k = 2$, the BAF value is equal to 1 as suggested earlier and thus ignored while reporting the correct number of clusters.

4.1.4 Selecting a Representative Subgraph

We used the FURS selection technique described in Chapter 3 for identifying a representative subgraph of the large scale complex network.

We provide an algorithm which summarizes the KSC approach for big data networks in Algorithm 9.

Algorithm 9: KSC Algorithm for Big Data Networks

Data: Given a graph $G = (V, E)$, which might not necessarily fit in memory.
Result: The partitions of the graph G i.e. divide graph into k clusters.

Convert and store the graph G in sparse format.
if *If G fits in the main memory* **then**
  Select training set X_{tr} (i.e. maximum size = $5,000$ nodes) using FURS.
  Select the validation set of nodes (same size as X_{tr}) after removing the subgraph S corresponding to X_{tr} using FURS.
  Calculate the kernel matrix Ω by applying cosine similarity operations on sparse adjacency lists of $\forall i, j\ v_i, v_j \in X_{tr}$.
  Perform eigen-decomposition of Ω to obtain the model i.e. $\alpha^{(l)}, b_l$.
  Use out-of-sample extension property to obtain the projection values for the validation set i.e. $e^{(l)}_{valid}$.
  Use BAF and $e^{(l)}_{valid}$ to estimate the number of clusters k.
  After estimating k, use the out-of-sample property and the decoding scheme to assign clusters to unseen test nodes.
else
  Divide G into blocks where each block contains an adjacency list of maximum $5,000$ nodes.
  To obtain training and validation nodes from all parts of graph divide the size of the subset to be selected by the number of blocks used to store G.
  Select this estimated number of training nodes from each block by running the FURS selection technique on each block. This results in a subset of nodes which approximately maximizes the coverage of the graph.
  Run an iteration over all the blocks and remove all the edges corresponding to the selected subset of training nodes.
  Calculate the kernel matrix Ω by applying cosine similarity operations on sparse adjacency lists of $\forall i, j\ v_i, v_j \in X_{tr}$.
  Perform eigen-decomposition of Ω to obtain the model i.e. $\alpha^{(l)}, b_l$.
  Re-perform the operations specified for selecting training nodes to select the validation set.
  If the validation set size $\gg$ training set size, it might become in-feasible to store Ω_{valid} in the main memory. To overcome this problem, divide the validation set into blocks containing adjacency list of maximum $5,000$ nodes. The maximum size of Ω_{valid} would be 5000×5000 which can be stored in memory as mentioned earlier.
  Use out-of-sample extension property to obtain the projection values for each block of validation set i.e. $e^{(l)}_{valid_{block_i}}$.
  Use BAF on all the blocks of $e^{(l)}_{valid_{block_i}}$ to estimate the value of k.
  After estimating k, use the out-of-sample property and the decoding scheme to assign clusters to unseen test nodes.
  Each block of G is used to as test data and cluster memberships are assigned to the nodes in the corresponding blocks.
end

4.1.5 Experiments and Analysis

We conducted experiments on several big datasets including a synthetic network with $500,000$ nodes, a social network with about 4 million nodes, a road network of city with around 2 million nodes and YouTube network with more than 1 million nodes. These real world datasets are obtained from `http://snap.stanford.edu/data/index.html`.

Dataset Description

- Synthetic Network - The synthetic network is generated from the software provided in [106] with a mixing parameter of 0.1. The number of nodes in the network is $500,000$, the number of edges in the network is $23,563,412$ and the number of communities in the network is 6.
- YouTube Network - In YouTube social network, users form friendship with each other and the users can create groups which others can join. This network has more than 1 million nodes and around 2.5 million edges. Thus the network is highly sparse.
- roadCA Network - A road network of California. Intersections and endpoints are represented by nodes and the roads connecting these intersections are represented as edges. The number of nodes in the network is around 2 million and the number of edges in the network is more than 5.5 million.
- Livejournal Network - It is a free online social network where users are bloggers and they declare friendship among themselves. The number of nodes in the network is around 4 million and the network has around 35 million edges.

FURS on these datasets

We first apply the FURS technique on these datasets to select a training set and validation set of nodes. As mentioned earlier, the maximum number of nodes that we select using FURS selection technique is $5,000$ nodes in order to have a kernel matrix which adheres to the memory restrictions. The step that we follow is to first select the training set using FURS, then these set of nodes are removed from the graph. We then run another iteration of FURS to select the validation set of nodes. Table 4.1 highlights the values of various evaluation metrics for these datasets for the FURS selection technique for the training set of nodes.

Dataset	**Nodes**	**Edges**	**Time**	**Coverage**	**Clustering Coefficient**	**Degree Distribution**
Synthetic	500,000	23,563,412	103.49	0.44	0.906	0.786
YouTube	1,134,890	2,987,624	20.546	0.28	0.975	0.959
roadCA	1,965,206	5,533,214	30.816	0.135	0.89	0.40
Livejournal	3,997,962	34,681,189	181.21	0.173	0.92	0.953

Table 4.1: Various evaluation metrics on FURS for each dataset

From Table 4.1, we can observe that even after selecting just $5,000$ nodes which is equivalent to 0.01, 0.0045, 0.0025, 0.00125 fraction of nodes in the

four networks respectively, the coverage value is quite high. The clustering coefficient and degree distribution values have been converted into similarity metric as described in [75]. The maximum value for the clustering coefficient and degree distribution can be 1. So the higher the value the better the quality of the subset selected. We observe from Table 4.1 that these quality metrics have quite high values with the exception of 0.4 value of degree distribution for roadCA network. We also notice that the time required for subset selection for the Synthetic network is more than that for networks like YouTube and roadCA which have more nodes. But this is because the Synthetic network is much more dense than these real world networks with around 23 million edges in comparison to nearly 2.5 and 5.5 million edges in the YouTube and roadCA network respectively.

Results and Analysis

The kernel spectral clustering results into a model based on the subgraph selected by the FURS subset selection technique and uses its out-of-sample extension property to provide cluster labels for new unseen nodes. Since, most of the spectral clustering methods, graph partitioning methods and graph clustering methods require to take into consideration the full network into the main memory, it would be unfair to compare KSC with these methods. There are several graph clustering techniques like Louvain method, Infomap method, CNM [38] which can scale up to million nodes on a single computer. However, if the size of the network increases such that it can no longer fit in the memory then these methods would become in-feasible.

Figure 4.2 shows the model selection or identification of the number of clusters k for the different datasets. We plot 3 dominant peaks for each of the 4 datasets. Currently, we select the peaks based on an adhoc procedure. We sort BAF values and select the maximum and restrict from selecting peaks in the immediate neighborhood as we might miss the hierarchical structure in that case. As mentioned earlier, the BAF metric is not only suitable for model selection but can also be used as an effective cluster evaluation metric. The higher the BAF value, the better the partitioning of the network. We observe that for the Synthetic network the BAF values are high for small number of clusters. Thus, in this case we make an exception and select the peaks close to each other. For the Synthetic network we do locate a peak at $k = 6$ with $BAF = 0.6817$, which is the actual number of clusters present in the network. For the YouTube network, we observe that we reach the first dominant peak at $k = 9$ and then the BAF value starts to decrease before it starts to increase again and locates the next dominant peak at $k = 97$. This shows the effectiveness of the BAF metric for identification of hierarchical

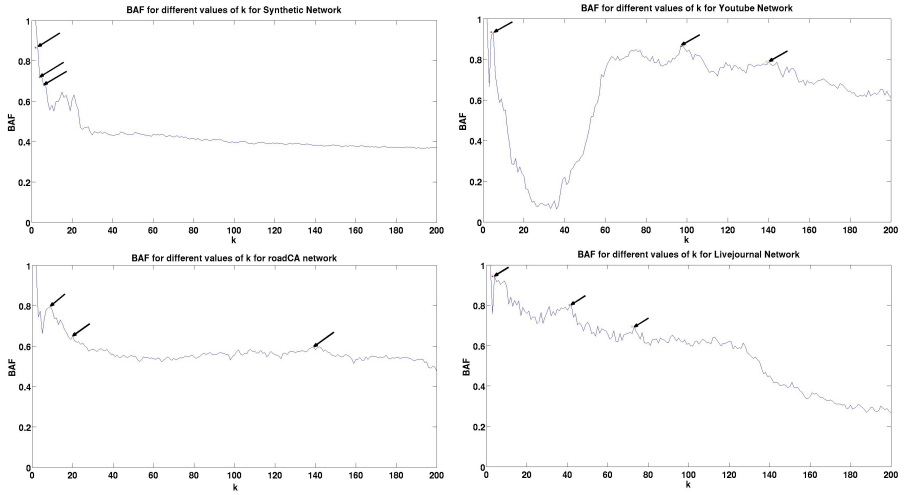


Figure 4.2: BAF for different values of k for the various big data networks

structure in the big data network. For the YouTube network the BAF values decrease initially as the value of k increases indicating that the clusters are not well-formed. However, its value start to increase from $k = 40$ as we are now obtaining the second level of hierarchy. The BAF value is 0.8654 for $k = 97$ which indicates that there are 97 well partitioned clusters at the second level of hierarchy for the YouTube network. For the roadCA and Livejournal networks, the BAF generally follows a decreasing trend with the increment in the number of clusters with a few spurious dominant peaks as depicted in Figure 4.2. The maximum number of clusters in the network was fixed as 200 in our experiments and we observe that it is sufficient to provide a general trend for BAF versus the number of clusters (k) curve.

Table 4.2 provides details about three dominant peaks, time required for training i.e. time required for building the kernel matrix and obtaining the model $(\alpha^{(l)}, b_l)$, validation time and the time required for testing. For testing, we use the entire big data network as test set. We noticed that it is not possible to store the big Ω_{test} in the main memory and we divide the test data into blocks containing adjacency list of maximum $5,000$ nodes as described in Algorithm 9. The maximum number of nodes we selected for training set is $5,000$ taking into account the memory constraints.

From Table 4.2, we observe that the training time for building the model for these big data networks is quite small. The training time comprises of constructing the kernel matrix (Ω) and performing eigen-decomposition to obtain the model

Dataset	Peak_1	BAF_1	Peak_2	BAF_2	Peak_3	BAF_3	Train Time (Seconds)	Validation Time (Seconds)	Test Time (Seconds)
Synthetic	3	0.8605	4	0.719	6	0.6817	103.45	111.42	10234.5
YouTube	5	0.9348	97	0.8654	139	0.7877	88.64	97.86	20055.45
roadCA	9	0.798	20	0.6482	138	0.5917	97.412	105.13	38960
Livejournal	4	0.94355	41	0.8014	73	0.6951	121.43	134.56	97221.4

Table 4.2: Experimental results for various Big Data Networks

$(\alpha^{(l)}, b_l)$. The validation time comprises of creating Ω_{valid}, using out-of-sample extensions to obtain $e^{(l)}_{valid}$ and obtain the model parameter i.e. the number of clusters k using the BAF metric. The maximum time required is the test time i.e. time required to create Ω_{test} and cluster memberships using out-of-sample extensions for each block of the large network. We observe that the number of blocks into which the Synthetic network, YouTube network, roadCA network and Livejournal network are divided is 100, 227, 394 and 800 respectively. The average testing time for each block of each dataset is nearly the same as the time required for training the model for that dataset. The testing operation can easily be performed on a distributed environment. Thus, the time required for testing can be scaled down by a factor of 20-40 depending on the number of computers in the cluster of the distributed environment. We observe that the time required for training the Synthetic network is more than YouTube and roadCA network though they have more number of nodes. This is because the Synthetic network is much more denser than these two real world datasets.

We also compare the proposed KSC method using BAF metric (BAF-KSC) with original KSC formulation using BLF criterion (BLF-KSC), Louvain, Infomap and CNM on evaluation metrics like Modularity (Q) and conductance (Con) for several small scale real world networks to provide a comparison of the performances. We have considered a wide range of networks varying from flight network (Openflights), network based on trust (PGPnet), biological network (Metabolic), citation networks (HepTh, HepPh), communication network (Enron), review based network (Epinion) to collaboration network (Condmat). Most of these networks can be found at `http://snap.stanford.edu/data/`.

Dataset	BAF-KSC			BLF-KSC [13]			Louvain [36]			Infomap [42]			CNM [38]		
	Clusters	Q	Con	Clusters	Q	Con	Clusters	Q	Con	Clusters	Q	Con	Clusters	Q	Con
Openflights	5	0.533	**0.002**	5	0.523	0.002	109	**0.61**	0.02	18	0.58	0.005	84	0.60	0.016
PGPnet	8	0.58	**0.002**	9	0.53	0.002	105	**0.88**	0.045	84	0.87	0.03	193	0.85	0.041
Metabolic	10	0.22	0.028	5	0.215	**0.012**	10	**0.43**	0.03	41	0.41	0.05	11	0.42	0.021
HepTh	6	0.45	**0.0004**	5	0.432	0.0004	172	**0.65**	0.004	171	0.3	0.004	6	0.423	0.0004
HepPh	5	0.56	**0.0004**	5	0.41	0.001	82	**0.72**	0.007	69	0.62	0.06	6	0.48	0.0007
Enron	10	0.4	0.002	7	0.21	**0.001**	1272	**0.62**	0.05	1099	0.37	0.27	6	0.25	0.0045
Epinion	10	**0.22**	0.0003	10	0.22	**0.0**	33	0.006	0.0003	17	0.18	0.0002	10	0.14	**0.0**
Condmat	6	0.28	**0.0002**	13	0.43	0.0003	1030	**0.79**	0.03	1086	0.79	0.025	8	0.38	0.0003

Table 4.3: Performance comparison of BAF-KSC method with BLF-KSC, Louvain, Infomap and CNM methods on quality metrics like Modularity (Q) and Conductance (Con) for several real world networks.

From Table 4.3 we observe that the BAF-KSC and BLF-KSC methods results in small number of clusters. This is because we consider the first peak corresponding to the highest BAF and BLF value respectively. We also observe that since the KSC method results in small number of clusters the conductance (Con) value is the quite low for most of the networks. The lower the conductance value the better the quality of the clusters. We also observe that the Modularity (Q) metric value is generally high for the BAF-KSC and BLF-KSC methods but less in comparison to Louvain method. We also observe the Q value is higher for the proposed KSC method (BAF-KSC) over the original KSC formulation (BLF-KSC) for 7 real world networks. Higher values of Modularity (Q) mean more modular graph partitions. The Louvain method is the best w.r.t. the Q metric as it optimizes the Modularity function. However, the conductance for the Louvain method for the various real world networks is generally 10 times more than the conductance for the KSC method. The CNM and BLF-KSC methods results in best conductance for Epinion networks as it identifies small number of well separated clusters for these networks.

4.1.6 Conclusion

In this work, we showed that the Kernel Spectral Clustering (KSC) method is a powerful tool for community detection for big data networks. While the problem of community detection has received a lot of attention in the past but the state-of-the-art approaches work under the assumption that the entire network can fit in the main memory. However, with the increasing amount of information the size of the networks will only increase and big data networks need not necessarily fit in the main memory.

The KSC method employs an optimization based framework to construct the model which has a very useful out-of-sample extension property. But the model that is constructed should be such that it adheres to the memory restrictions. Therefore, there is a need to select a representative subgraph of the big data network on which the model can be built. We use the FURS selection procedure for this purpose. It selects nodes from different dense regions of the graph while maximizing the coverage and preserves the inherent community structure of the big data network.

In order to obtain the model parameters i.e. in case of large complex networks the number of clusters k in the network, we propose a novel metric *Balanced Angular Fit*. The BAF metric works with a codebook $\mathcal{CB}$ and the projections of the validation set to determine the ideal number of clusters k in the big data network. The out-of-sample extensions property allows inferring community affiliation for unseen nodes. It is because of this property that we can handle

large scale networks relatively easily. Finally, we show that the KSC method works effectively on a big Synthetic Network and several real world big data networks. Future work may focus on automatically determining the dominant peaks in BAF versus number of clusters curve.

4.2 Self-Tuned Kernel Spectral Clustering[2]

Abstract — We propose a parameter-free kernel spectral clustering model for large scale complex networks. The kernel spectral clustering (KSC) method works by creating a model on a subgraph of the complex network. The model requires a kernel function which can have parameters and the number of communities k has be detected in the large scale network. We exploit the structure of the projections in the eigenspace to automatically identify the number of clusters. We use the concept of entropy and balanced clusters for this purpose. We show the effectiveness of the proposed approach by comparing the cluster memberships w.r.t. several large scale community detection techniques like Louvain, Infomap and Bigclam methods. We conducted experiments on several synthetic networks of varying size and mixing parameter along with large scale real world experiments to show the efficiency of the proposed approach.

4.2.1 Introduction

Recently, a kernel spectral clustering formulation based on weighted kernel principal component analysis (PCA) with a primal-dual framework was proposed in [13]. The method was extended for community detection in [105] and [22]. The KSC model requires a kernel function and the number of clusters k has to be detected. For model selection, the proposed approach in [105] and [22] uses the concept of Modularity [37]. The Modularity criteria needs to store a $N \times N$ matrix which makes it in-feasible for large scale networks. An approach to apply KSC for big data networks was proposed in [20]. We use the concepts proposed in [20] to build the model. In this work, we propose to make the kernel spectral clustering approach parameter-free or in other words

- Use the normalized linear kernel function.

[2]This section consists of Section 1 and $3-5$ from Mall R., Langone R., Suykens, J.A.K., **"Self-Tuned Kernel Spectral Clustering for Large Scale Networks"**, in Proc of IEEE International Conference on Big Data (IEEE BigData), Santa Clara, USA, Oct 2013, pp. 385-393.

- Automatically detect the number of communities (k) in the large scale network.

4.2.2 Self-Tuned Kernel Spectral Clustering

In Section 4.1.2, we have shown that for large scale networks the normalized linear kernel function is sufficient. The only parameter left is the number of clusters to be identified (k). We propose a novel technique to determine the number of clusters (k) automatically in a few steps. We exploit the projections of the validation nodes in the eigenspace. The proposed approach is based on the simple idea that the projections of the validation nodes which belong to the same cluster have very small or nearly zero angular distance from each other in comparison with the nodes belonging to different clusters.

Before we explain the technique, we mention a few pre-requisites. All the experiments were performed on a single machine with 8 Gb RAM, 2.4Ghz Intel I-7 processor. The maximum size matrix that can be stored in memory is $5,000 \times 5,000$. The training set size is fixed as $N_{tr} = \min(15\% \text{ of } N, 5000)$ i.e. minimum between 15% of the total nodes in the network (N) and $5,000$ nodes. The subset size of 15% of the nodes in the network is chosen as per experimental analysis in [77]. We set the validation set size N_{valid} to be the same as N_{tr}. We put the condition that minimum size of a cluster is $MinCsize = \max(0.01\% \text{ of } N_{valid}, 5)$ i.e. maximum between 0.01% of the number of nodes in the validation set and 5 nodes. Since for estimation of the number of clusters we use the validation set, we put the minimum size constraint on the validation set. This constraint is to prevent the selection of outlier groups of small size as a cluster. Thus, suppose the training set size is $5,000$ nodes then the minimum size of cluster has to be 5. The $maxk$ value is determined as $\lceil \frac{N_{tr}}{MinCsize} \rceil$. Thus, if the minimum size of the cluster is 5 and maximum size of the validation set is $5,000$ then the maximum number of clusters that can be detected is $1,000$. The runtime complexity of KSC approach is $\mathrm{O}(N_{tr}^3) + \mathrm{O}(N_{tr} \times N_{test})$. However, testing can be done in parallel and the second term in time complexity can be reduced significantly.

Identifying the number of clusters

We obtain the projection of the validation nodes $v_j \in V_{valid}$ in the eigenspace using the out-of-sample extension and binarize the score variables as follows:

$$\text{sign}(e_{valid}^{(l)}) = \text{sign}(\Omega_{valid}\alpha^{(l)} + b_l 1_{N_{valid}}) \tag{4.7}$$

Let the latent variable matrix be represented as P. The size of the this matrix is $N_{valid} \times (maxk - 1)$. The matrix P is defined as $[e_1, \ldots, e_{N_{valid}}]^\intercal$. We then create a new symmetric affinity matrix R from the latent variable matrix P. Since we already stated that the maximum size of the validation set is $5,000$ nodes, the maximum size of the R matrix is $5,000 \times 5,000$. The affinity matrix is build using cosine distance as:

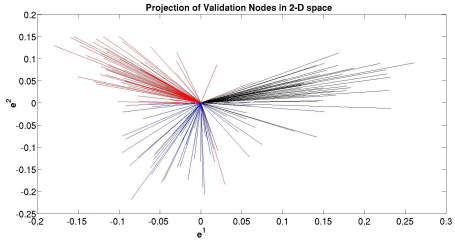
$$R(i,j) = CosDist(e_i, e_j) = 1 - \cos(e_i, e_j) = 1 - \frac{e_i^\intercal e_j}{\|e_i\|\|e_j\|}$$

The $CosDist()$ function can take values between $[0, 2]$. We use the concept that nodes which belong to the same community have smaller or nearly zero angular distance w.r.t the nodes within the community and more angular distance w.r.t nodes belonging to different communities. Thus, the nodes which belong to the same cluster will have very small or nearly zero $CosDist(e_i, e_j), \forall i, j$ in the same cluster.

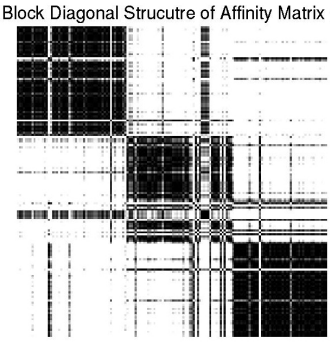
Figure 4.3 highlights this idea on a synthetic network of $1,000$ nodes generated by the software proposed in [103] with mixing parameter $\mu = 0.1$. The mixing parameter implies the extent of overlap in the communities with smaller values representing less overlap. In Figure 4.3a, we obtain the projections of the validation nodes in 2-dimensional space as we set $maxk = 3$ for the purpose of visualization. Figure 4.3b showcases the presence of a block diagonal structure for the affinity matrix R. To obtain this block diagonal matrix we used the groundtruth cluster memberships for the validation set of nodes. We then sort the validation nodes using the groundtruth information on the basis of their cluster affiliation and construct the affinity matrix R. The black shaded regions represent the part of the affinity matrix where the $CosDist(e_i, e_j)$ values are zero and nearly equal to zero. The aim is to identify the number of such block diagonals which determine the number of clusters k in the given network.

Real world networks have overlapping communities. However, the extent of overlap in these communities is not known beforehand. Also the groundtruth is not available in most real world large scale networks. Thus the affinity matrix R, though contains a block-diagonal structure, need not necessarily be ordered in block-diagonal format. We observe from Figure 4.3a, that projection of nodes which belong to the same cluster do not necessarily have zero angular distance due to presence of overlap. But the angle between two projection vectors say e_i and e_j belonging to the pth cluster (C_p) is mostly less than $90°$ leading to $0 \leq \cos(e_i, e_j) \leq 1$. After little algebraic manipulation we obtain that $0 \leq CosDist(e_i, e_j) \leq 1$ for two nodes i and j belonging to cluster p (C_p).

Since the extent of overlap in terms of angular distance is not known beforehand we setup a threshold $td = [0.1, 0.2, \ldots, 1]$. We need to investigate only 10 different threshold values which is much less than iteratively solving for each



(a) Validation node projections



(b) Affinity matrix R w.r.t. groundtruth

Figure 4.3: First two steps for identifying the k clusters in a given network

$k = 2$ to $k = maxk - 1$ in increments of 1. Generally as the value of threshold increases the number of clusters decreases as more validation nodes satisfy the threshold and fall in the same cluster. For a given value of threshold $t \in td$, we first identify that validation node (e_i) for which the number of nodes with $CosDist(e_i, e_j) \leq t$ is maximum. We keep the count of this number of nodes in a vector. We obtain the indices of all those validation nodes and shrink the affinity matrix R by removing the rows and columns corresponding to these indices. We repeat this process recursively till it results in an empty matrix. This procedure is equivalent to identifying the block diagonal structure. As a result of this procedure for each value of threshold t, we obtain a vector $(SizeC_t)$ containing information about the size of clusters for that value of threshold t.

In order to determine the ideal number of clusters k, we use the notion of entropy and balance w.r.t. the size of the clusters. For each value of threshold t we have the vector $SizeC_t$. We also put the constraint that the minimum size of a cluster is greater than or equal to $MinCsize$. So we remove from $SizeC_t$ vector all the size of clusters which are less than $MinCsize$. The Shannon entropy for a discrete set of probabilities is defined as:

$$H(s)^t = -\sum_{i=1}^{k} p(s_i)^t \log(p(s_i^t)), \forall t \in td \tag{4.8}$$

where $H(s)^t$ is the entropy corresponding to the vector $SizeC_t$ for the threshold t. Here s_i^t is the size of the ith cluster and $p(s_i^t)$ is defined as ratio between the size of ith cluster corresponding to threshold t and the size of the validation set

(N_{valid}). Entropy values are higher when there is a larger number of uniformly sized clusters.

Balance is generally defined as the ratio between the minimum size cluster in $SizeC_t$ and the maximum size cluster in $SizeC_t$. The aim of using balance is to prevent one or more large community from being superior over all other communities. Higher balance promotes uniformly sized clusters and prevents few extremely large communities from being dominant. This leads to more number of communities. In this work, we use the overall balance criterion which is the sum of balance for all the clusters in $SizeC_t$ and defined as:

$$B(s)^t = \sum_{i=1}^{k} \frac{s_i^t}{\max(s_1^t, \ldots, s_k^t)}, \forall t \in td \tag{4.9}$$

where s_i^t represents the size of the ith cluster corresponding to threshold t. The denominator in (4.9) refers to maximum sized cluster for the threshold t. The higher the overall balance in the clusters, the closer the value of $B(s)^t$ is to k.

We calculate the entropy and overall balance for each of these $SizeC_t$ and use the harmonic mean or an F-measure to determine the best of both entropy and overall balance as their values are comparable. The F-measure is defined as:

$$F(s)^t = \frac{2H(s)^t B(s)^t}{H(s)^t + B(s)^t}, \forall t \in td \tag{4.10}$$

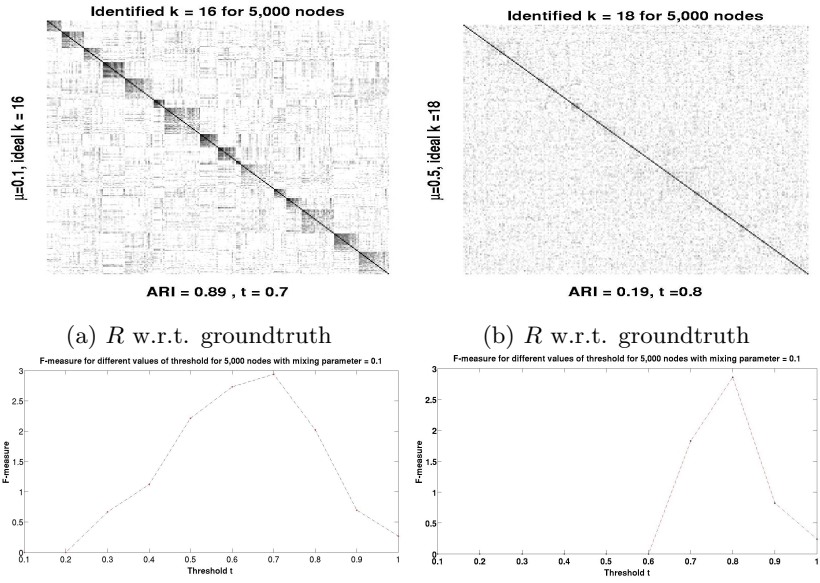
where $H(s)^t$ and $B(s)^t$ represents the entropy and expected balance for threshold t. We evaluate $F(s)^t$ for all values of threshold $t \in td$ and output that value of threshold t say $maxt$ for which $F(s)^t$ is maximum. The number of clusters (k) is obtained from the vector $SizeC_{maxt}$ and is equal to the number of terms in this vector. Algorithm 10 summarizes the proposed method for identifying the number of clusters (k) in a given large scale network.

4.2.3 Experiments

We conducted experiments on several synthetic and real world networks. The synthetic networks are generated from the software proposed in [103] with varying the size of the network and the mixing parameter μ. The mixing parameter contains information about the extent of overlap between the communities in the large scale network.

Algorithm 10: Algorithm to automatically identify k communities

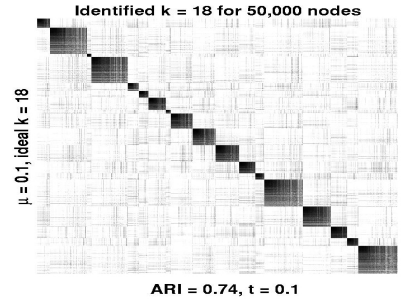
Data: $P = [e_1, e_2, \ldots, e_{N_{valid}}]$
Result: The number of clusters k in the given large scale network.
Construct an affinity matrix R using the projection vectors $e_i \in P$ and the similarity function $CosDist()$.
Set $td = [0.1, 0.2, \ldots, 1]$.
for *each* $t \in td$ **do**
    Save R in a temporary variable B i.e. $B := R$.
    Initialize the $SizeC_t$ as an empty vector.
    while B *is not an empty matrix* **do**
        Locate the projection of validation node e_i for which the number of nodes with $CosDist(e_i, e_j) \leq t$ is maximum.
        Count the number of these nodes and locate the indices of these nodes.
        Append the count of these nodes to the vector $SizeC_t$.
        Remove rows and columns corresponding to these indices.
    end
    Calculate the entropy $H(s)^t$ from $SizeC_t$.
    Calculate the expected balance $B(s)^t$ from $SizeC_t$.
    Calculate the F-measure $F(s)^t$ using $H(s)^t$ and $B(s)^t$.
end
Obtain the threshold corresponding to which the F-measure is maximum as $maxt$.
Estimate k as the number of terms in the vector $SizeC_{maxt}$.



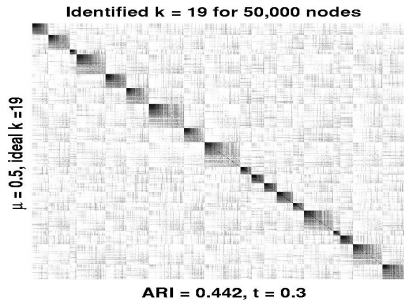
(a) R w.r.t. groundtruth (b) R w.r.t. groundtruth

(c) F-measure vs Threshold (d) F-measure vs Threshold

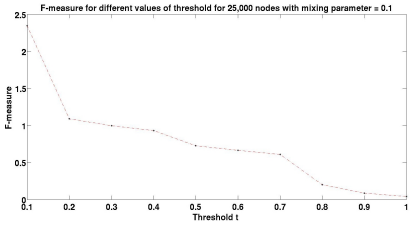
Figure 4.4: Self-Tuned KSC on a synthetic network of $5,000$ nodes for mixing parameter values 0.1 and 0.5



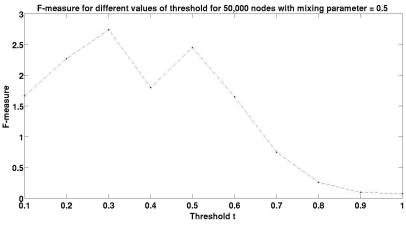
(a) R w.r.t. groundtruth



(b) R w.r.t. groundtruth



(c) F-measure vs Threshold



(d) F-measure vs Threshold

Figure 4.5: Self-Tuned KSC on a synthetic network of 50,000 nodes for mixing parameter values 0.1 and 0.5

Experiments on Synthetic networks

Figures 4.4 and 4.5 show the result of our proposed approach for identifying the number of clusters k on synthetic networks made of $5,000$ and $50,000$ nodes respectively. We show the results for two different mixing parameters $\mu = 0.1$ and $\mu = 0.5$. For smaller μ the extent of overlap is less in comparison to higher value of μ and detecting communities is less difficult.

From Figures 4.4a, 4.4b, 4.5a and 4.5b, we show that the proposed method identifies exactly or close to the exact number of clusters present in the network even when the extent of overlap is high. The adjusted rand index (ARI) [142] values are also quite high suggesting that the clusters obtained via KSC [13] are meaningful. We also observe that the threshold corresponding to which we obtain the maximum F-measure is related to the mixing parameter μ. For the smaller mixing parameter (0.1) the threshold value t for which the F-measure is maximum is less than or equal to the threshold value corresponding to higher mixing parameter 0.5 as observed from Figures 4.4c, 4.4d,4.5c and 4.5d.

Experiments on Real World Datasets

We conducted experiments on several real-world large scale networks ranging from trust networks (PGPnet), collaboration networks (Cond-mat), citation networks (HepPh), communication networks (Enron), social networks (Epinion), web graphs (Youtube), actor networks (Imdb-actor) to road networks (roadCA). Details about these networks are available at `http://snap.stanford.edu/data/index.html`. Table 4.4 provides a few key statistics of each dataset. We

Dataset	**Vertices**	**Edges**	**CCF**
PGPnet	10,876	39,994	0.008
Cond-mat	23,133	186,936	0.6334
HepPh	34,401	421,578	0.1457
Enron	36,692	367,662	0.497
Epinions	75,879	508,837	0.2283
Imdb-actor	383,640	1,342,595	0.453
Youtube	1,134,890	2,987,624	0.173
roadCA	1,965,206	5,533,214	0.0464

Table 4.4: Vertices (V), Edges (E) and Clustering Coefficients (CCF).

use the entire large scale network as test set in our experiments. We use the clusters obtained by Louvain [36], Infomap [42] and Bigclam [143] methods as groundtruth for these networks. We compare the clusters obtained via the Original KSC method w.r.t. these groundtruth clusters. Then we modify the Original KSC method by automatically determining the number of clusters (k). We combine our proposed approach with the original out-of-sample extension property of the KSC model to obtain the Self-Tuned KSC (ST-KSC) method.

In order to perform a comprehensive evaluation, once we automatically obtain the value of k, we perform k-means on the projections in the eigenspace for the validation set to obtain the centroids. We obtain the projections of the test set in the eigenspace and assign the cluster membership corresponding to the centroid to which it is the closest. This method is referred as Self-Tuned k-means KSC (ST k-means KSC). We then compare the cluster memberships obtained via ST-KSC and ST k-means KSC w.r.t. the community affiliations via Louvain, Infomap and Bigclam method respectively.

For evaluating the cluster memberships we use three information theoretic measures: mutual information (MI), variation of information (VI) and adjusted rand index (ARI). These are standard cluster quality evaluation metrics given two list of cluster memberships and are described in detail in [45].

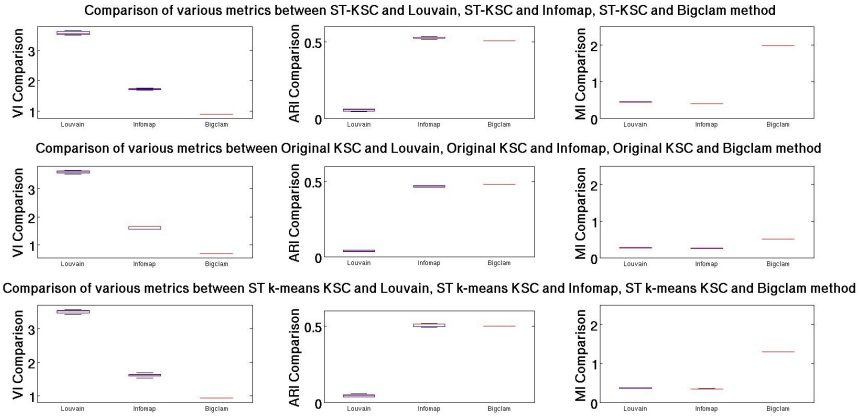
Table 4.5 evaluates the clusters obtained via the original KSC method, the proposed ST-KSC method and ST k-means KSC methods w.r.t. the clusters obtained by Louvain, Infomap and Bigclam methods. Since the partition obtained by methods like Louvain and Infomap method are not unique, we

		Original KSC				ST-KSC				ST k-means KSC			
		k	VI	ARI	MI	k	VI	ARI	MI	k	VI	ARI	MI
PGPnet	Louvain	4	4.022	0.004	0.124	30	**3.94**	**0.073**	**1.102**	30	4.08	0.041	0.87
	Infomap	4	3.583	0.006	0.1	30	**3.7**	**0.1**	**0.98**	30	3.77	0.063	0.78
	Bigclam	4	**1.57**	0.011	0.029	30	3.4	**0.12**	**1.026**	30	3.462	0.07	0.78
Cond-mat	Louvain	3	4.021	0.01	0.225	52	**3.91**	**0.011**	0.324	52	4.5	0.04	**0.68**
	Infomap	3	3.132	0.05	0.222	52	**3.1**	0.051	0.32	52	3.63	**0.224**	**0.68**
	Bigclam	3	**0.5**	**0.49**	0.562	52	1.47	0.155	**2.49**	52	1.22	0.331	1.99
HepPh	Louvain	3	**3.22**	0.028	0.093	76	5.7395	0.014	0.216	76	5.23	**0.09**	**0.64**
	Infomap	3	**5.72**	0.006	0.166	76	7.224	0.014	0.78	76	6.3	**0.1**	**1.4**
	Bigclam	3	**0.95**	0.0004	0.0005	76	6.44	0.001	0.51	76	6.25	**0.09**	**0.8**
Enron	Louvain	3	3.6	0.038	0.274	16	**3.57**	**0.053**	**0.442**	16	3.51	0.0441	0.37
	Infomap	3	**1.6**	0.467	0.261	16	1.73	**0.53**	**0.4**	16	1.62	0.51	0.344
	Bigclam	3	**0.7**	0.48	0.51	16	0.91	**0.51**	**1.97**	16	0.94	0.50	1.3
Epinion	Louvain	3	3.1	0.0006	0.005	24	**3.08**	0.001	0.015	24	3.451	**0.03**	**0.06**
	Infomap	3	4.163	0.017	0.007	24	**4.16**	**0.031**	0.013	24	4.47	0.014	**0.1**
	Bigclam	3	**0.11**	**0.1**	0.007	24	0.84	0.02	**0.02**	24	0.74	0.014	0.016
Imdb-actor	Louvain	3	**3.05**	0.0005	0.006	59	4.4	**0.114**	**1.09**	59	3.84	0.093	0.673
	Infomap	3	**2.7**	0.0006	0.004	59	4.2	**0.11**	**0.99**	59	3.58	0.08	0.63
	Bigclam	3	**1.1**	0.0	0.0	59	3.0405	**0.025**	**1.07**	59	3.14	0.02	0.83
Youtube	Louvain	3	**3.64**	0.0003	0.004	21	4.1	0.016	0.047	21	4.55	**0.017**	**0.096**
	Infomap	3	**3.71**	0.0003	0.0035	21	4.16	0.014	0.03	21	4.7	**0.016**	**0.07**
	Bigclam	3	**0.056**	0.0	2.6e-06	21	1.6	0.035	0.06	21	1.25	**0.09**	**0.1**
roadCA	Louvain	3	**6.12**	0.0004	0.141	39	6.38	0.0005	**0.41**	39	6.42	**0.0006**	0.242
	Infomap	3	8.0	7.6e-05	0.21	39	**7.9**	0.0001	**0.64**	39	8.14	**0.00011**	0.40206
	Bigclam	3	**1.075**	0.0	0.0	39	2.62	0.044	**1.26**	39	1.56	**0.17**	1.01

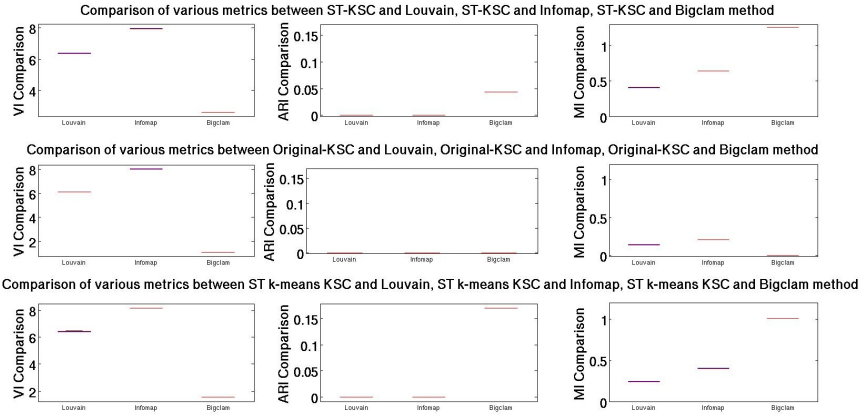
Table 4.5: Evaluation of the cluster memberships for Original KSC, ST-KSC and k-means KSC for several real world networks w.r.t. Louvain, Infomap and Bigclam methods. The highlighted numbers represent the best results. Original KSC method performs better w.r.t. VI while the ST-KSC and ST k-means KSC methods perform well w.r.t. ARI and MI.

perform 10 randomizations and report the mean VI, ARI and MI values. Figure 4.6a and 4.6b shows the variations for these evaluation metrics in case of Enron and roadCA networks respectively.

From Table 4.5 we observe that the absolute values of ARI are quite small for most networks as it is heavily dependent on the number of clusters. For methods like Louvain and Infomap method, the number of clusters found are much more than that for KSC methods which lead to smaller ARI. From Table 4.5 we observe that clusters obtained by the Original KSC method generally have better variation of information (VI) corresponding to the clusters obtained via Louvain, Infomap and Bigclam methods. This is because the Original KSC method using the BAF evaluation metric is biased to produce small number of clusters as observed from Table 4.5. However, due to this bias it results in poor ARI and MI values for most of the real world large scale networks. The clusters obtained by the two self-tuned methods have better ARI and MI values. The ST-KSC method results in better quality clusters for PGPnet, Enron, Imdb-actor and roadCA networks when the clusters are evaluated w.r.t. the clusters obtained by Louvain, Infomap and Bigclam methods. The evaluation



(a) For Enron network



(b) For roadCA network

Figure 4.6: Variation of evaluation metrics (VI, ARI and MI) for various methods for Enron 4.6a and roadCA 4.6b network. It shows that ST-KSC method gives best results for Enron network and ST k-means KSC method provides best results for roadCA network.

metrics for which ST-KSC method performs better is ARI and MI. The ST k-means KSC method results in better quality clusters for Cond-mat, HepPh and Youtube networks w.r.t to the quality metrics ARI and MI.

Figure 4.6 showcases the results of the various quality metrics for the Enron and roadCA networks. Figure 4.6a shows that the clusters obtained by all the KSC methods are dissimilar from those obtained by Louvain method because of

the high VI, low ARI and low MI values. However, the clusters obtained by the KSC methods are more similar to the ones obtained by Infomap and Bigclam methods. We observe this from 4.6a due to the low VI, high ARI and better MI values. In case of the Bigclam method, the clusters obtained are highly similar to that obtained by all the KSC methods. The clusters produced by the ST-KSC method in particular with VI (0.91), ARI (0.51) and MI (1.97) is the most similar to the clusters produced by Bigclam method.

From Figure 4.6b we observe that the clusters obtained by all the KSC methods have high VI and low ARI values in comparison to the clusters obtained by Louvain and Infomap method. This is because the mean number of clusters obtained by Louvain and Infomap method are $3,000$ and $65,807$ respectively over 10 randomizations. This is much larger in comparison to the number of clusters obtained by Original KSC (3) and 39 obtained by the proposed approach. However, the Bigclam method identifies 40 clusters. The clusters obtained by Bigclam method have best result w.r.t. VI with Original KSC method (1.075), w.r.t. ARI with ST k-means KSC (0.17) and w.r.t. MI with ST-KSC (1.26).

4.2.4 Conclusion

In this work, we proposed an approach to make the kernel spectral clustering method free of parameters for large scale networks. We used the normalized linear kernel for large scale networks and devised an approach to automatically identify the number of clusters k in the given network. For achieving this, we exploit the projections of the validation nodes in the eigenspace by creating an affinity matrix which had block-diagonal structure. We used the concepts of entropy and balance to identify these block-diagonals and obtain the number of clusters. We compared the resulting KSC methods with large scale community detection methods like Louvain, Infomap and Bigclam methods.

4.3 Multilevel Hierarchical Kernel Spectral Clustering[3]

Abstract–In this work we exploit the structure of the projections generated by KSC [13] in the eigenspace during the validation stage to automatically

[3]This section consists of Section 1 and Sections $3-5$ from Mall R., Langone R., Suykens J.A.K., **"Multilevel Hierarchical Kernel Spectral Clustering for Real-Life Large Scale Complex Networks"**, PLoS ONE, 9(6):e99966, Jun 2014.

determine a set of increasing distance thresholds. We use these distance thresholds in the test phase to obtain multiple levels of hierarchy for the large scale network. The hierarchical structure in the network is determined in a bottom-up fashion. We empirically showcase that real-world networks have multilevel hierarchical organization which cannot be detected efficiently by several state-of-the-art large scale hierarchical community detection techniques like the Louvain, OSLOM and Infomap methods. We show a major advantage our proposed approach i.e. the ability to locate good quality clusters at both the coarser and finer levels of hierarchy using internal cluster quality metrics on 7 real-life networks.

4.3.1 Introduction

The community structure exhibited by the real world complex networks often have an inherent hierarchical organization. This suggests that there should be multiple levels of hierarchy in these real-life networks with good quality clusters at each level. In other words, there exist meaningful communities at coarser as well as refined levels of granularity in this multilevel hierarchical system of the real-life complex networks.

A state-of-the-art hierarchical community detection technique for large scale networks is the Louvain method [36]. It uses a popular quality function namely *modularity* (Q) [104, 38, 39, 37] for locating modular structures in the network in a hierarchical fashion. Modularity measures the difference between a given partition of a network and the expectation of the same partition for a random network. By optimizing modularity, they obtain the modular structures in the network. However, it suffers from a drawback namely the resolution limit problem [43, 144, 145]. The issue of resolution limit arises because the optimization of modularity beyond a certain resolution is unable to identify modules even as distinct as cliques which are completely disconnected from the rest of the network. This is because modularity fixes a global resolution to identify modules which works for some networks but not others.

Recently the authors of [146] show that methods trying to use variants of modularity to overcome the resolution limit problem, still suffer from the resolution limit. They propose an alternative algorithm namely OSLOM [40] to avoid the issue of resolution. However, in our experiments we observe that OSLOM works well for benchmark synthetic networks [103] but in case of real-life networks it is unable to detect quality clusters at coarser levels of granularity. We also evaluate another state-of-the-art hierarchical community detection technique called the Infomap method [42]. The Infomap method uses an information theoretic approach to hierarchical community detection. It uses

the probability flow of random walks as a substitute for information flow in real-life networks. It then fragments the network into modules by compressing a description of the probability flow.

The KSC method was applied for community detection in graphs by [105]. However, their subset and model selection approach was computationally expensive and memory inefficient. Recently, the KSC method was extended for big data networks in [20]. The method works by building a model on a representative subgraph of the large scale network. This subgraph is obtained by the fast and unique representative subset (FURS) selection technique as proposed in [123]. During the model selection stage, the model parameters are estimated along with determining the number of clusters k in the network. A self-tuned KSC model for big data networks was proposed in [126]. The major advantage of the KSC method is that it creates a model which has a powerful out-of-sample extensions property. Using this property, we can infer community affiliation for unseen nodes of the whole network.

In [52], the authors used multiple scales of the kernel parameter σ to determine the hierarchical structure in the data using KSC approach. However, in this approach the clustering model is trained for different values of (k, σ) and evaluated for the entire dataset using the out-of-sample extension property. Then, a map is created to match the clusters at two levels of hierarchy. As stated by the authors in [52], during a merge there might be some data points of the merging clusters that go into a non-merging cluster which is then forced to join the merging cluster of the majority. In this work, we overcome this problem and generate a natural hierarchical organization of the large scale network in an agglomerative fashion.

The purpose of hierarchical community detection is to automatically locate multiple levels of granularity in the network with meaningful clusters at each level. The KSC method has been used effectively to obtain flat partitioning in real-world networks [105, 20, 126]. In this work, we exploit the structure of the eigen-projections derived from the KSC model. The projections of the validation set nodes in the eigenspace is used to create an iterative set of affinity matrices resulting in a set of increasing distance thresholds ($\mathcal{T}$). Since the validation set of nodes is a representative subset of the large scale network [123], we use these distance thresholds ($t_i \in \mathcal{T}$) on the projections of the entire network obtained as a result of the out-of-sample extension property of the KSC model. These distance thresholds, when applied in an iterative manner, provide a multilevel hierarchical organization for the entire network in a bottom-up fashion. We show that our proposed approach is able to discover good quality coarse as well as refined clusters for real-life networks.

There are some methods that optimize weighted graph cut objectives [34, 35, 147]

to provide multilevel clustering for the large scale network. However, these methods suffer from the problem of determining the right value of k which is user defined. In real-world networks the value of k is not known beforehand. So in our experiments, we evaluate the proposed multilevel hierarchical kernel spectral clustering (MH-KSC) algorithm against the Louvain, Infomap and OSLOM methods. These methods automatically determine the number of clusters (k) at each level of hierarchy. Figure 4.7 provides an overview of steps involved in the MH-KSC algorithm and Figure 4.8 depicts the result of our proposed MH-KSC approach on email network (Enron).

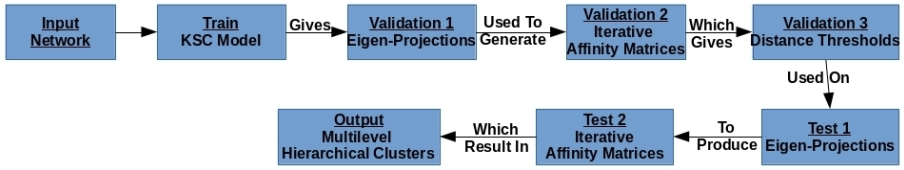


Figure 4.7: Steps undertaken by the MH-KSC algorithm

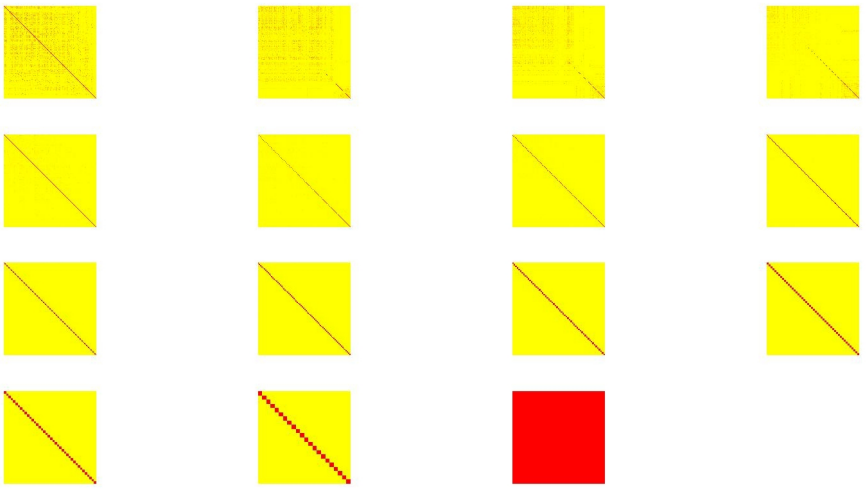
In all our experiments we consider unweighted and undirected networks. All the experiments were performed on a machine with 12Gb RAM, 2.4 GHz Intel Xeon processor. The maximum size of the kernel matrix that is allowed to be stored in the memory of our PC is $10,000 \times 10,000$. Thus, the maximum cardinality of our training and validation sets can be $10,000$. We use 15% of the total nodes as size of training and validation set (if less than $10,000$) based on experimental findings in [77]. We make use of the procedure provided in [20] to divide the data into chunks in order to extend our proposed approach to large scale networks. There are several steps in the proposed methodology which can be implemented on a distributed environment.

4.3.2 Multilevel Hierarchical KSC

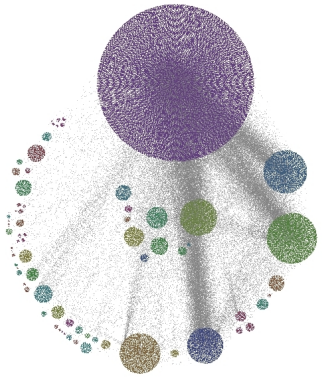
We use the predictive KSC model in the dual to get the latent variable matrix for the validation set V_{valid} represented as $P_{valid} = [e_1, \ldots, e_{N_{valid}}]^\intercal$ and the test set V_{test} (entire network) denoted by P_{test}. In [126] the authors create an affinity matrix R_{valid} using the latent variable matrix P_{valid} which is a $N_{valid} \times (maxk\text{-}1)$ matrix, as:

$$R_{valid}(i,j) = CosDist(e_i, e_j) = 1 - \cos(e_i, e_j) = 1 - \frac{e_i^\intercal e_j}{\|e_i\|\|e_j\|}, \qquad (4.11)$$

where $CosDist(\cdot,\cdot)$ function calculates the cosine distance between 2 vectors and takes values between $[0, 2]$. Nodes which belong to the same community



(a) Affinity matrix created at different levels of hierarchy in left to right order. The number of block-diagonals in each subgraph represents k at that level of hierarchy.



(b) Result of MH-KSC algorithm on Enron dataset. Circles which have the same colour are part of the same cluster at the coarsest level of hierarchy. We depict clusters at 2 different levels of hierarchy using the toolbox provided in [40].

Figure 4.8: Result of proposed MH-KSC approach on the Enron network

will have $CosDist(e_i, e_j)$ closer to 0, $\forall i, j$ in the same cluster. It was shown in [126] that a rotation of the R_{valid} matrix has a block diagonal structure. This block diagonal structure was used to identify the ideal number of clusters k in the network using the concept of entropy and balanced clusters.

Determining the Distance Thresholds

We propose an iterative bottom-up approach on the validation set to determine the set of distance thresholds $\mathcal{T}$. In our approach, we refer to the affinity matrix at the ground level of hierarchy as $R^{(0)}_{valid}$. The $R^{(0)}_{valid}$ matrix is obtained by calculating the $CosDist(\cdot,\cdot)$ between each element of the latent variable matrix P_{valid} as mentioned earlier. After several empirical evaluations, we observe that distance threshold at level 0 of hierarchy can be set to values between $[0.1, 0.2]$. In our experiments we set $t^{(0)} = 0.15$. This allows to make the approach tractable to large scale networks which will be explained later.

We then use a greedy approach to select the validation node with maximum number of similar nodes in the latent space i.e. we select the projection e_i which has a maximum number of projections e_j satisfying $R^{(0)}_{valid}(i,j) < t^{(0)}$. We put the indices of these nodes in $C^{(0)}_1$ representing the 1st cluster at level 0 of hierarchy. We then remove these nodes and corresponding entries from $R^{(0)}_{valid}$ to obtain a reduced matrix. This process is repeated iteratively until $R^{(0)}_{valid}$ becomes empty. Thus, we obtain the set $C^{(0)} = \{C^{(0)}_1, \ldots, C^{(0)}_q\}$ where q is the total number of clusters at ground level of hierarchy. The set $C^{(0)}$ has communities along with the indices of the nodes in these communities.

To obtain the clusters at the next level of hierarchy we treat the communities at the previous levels as nodes. We then calculate the average cosine distance between these nodes using the information present in them. At each level h of hierarchy we create a new affinity matrix as:

$$R^{(h)}_{valid}(i,j) = \frac{\sum_{m \in C^{(h-1)}_i} \sum_{l \in C^{(h-1)}_j} R^{(h-1)}_{valid}(m,l)}{|C^{(h-1)}_i| \times |C^{(h-1)}_j|}, \tag{4.12}$$

where $|\cdot|$ represents the cardinality of the set. In order to determine the threshold at level h of hierarchy, we estimate the minimum cosine distance between each individual cluster and the other clusters (not considering itself). Then, we select the mean of these values as the new threshold for that level to combine clusters. This makes the approach different from the classical single-link clustering where we combine two clusters which are closest to each other at a given level of hierarchy and the average-link agglomerative clustering where we combine based on the average distance between all the clusters.

The reason for using mean of these minimum cosine distance values as the new threshold is that if we consider the minimum of all the distance values then there is a risk of only combining 2 clusters at that level. However, it is desirable to combine multiple sets of different clusters. Thus, the new threshold $t^{(h)}$ at

level h is set as:

$$t^{(h)} = \text{mean}(\min_j(R^{(h)}_{valid}(i,j))), i \neq j. \tag{4.13}$$

We use this process iteratively till we reach the coarsest level of hierarchy where we have 1 cluster containing all the nodes. As a consequence we obtain the hierarchical clustering $\mathcal{C} = \{C^{(0)}, \ldots, C^{(maxh)}\}$ automatically. As we move from one level of hierarchy to another the value of distance threshold increases since we are merging large clusters at coarser levels of hierarchy. We finally end up with a set of increasing distance thresholds $\mathcal{T} = \{t^{(0)}, \ldots, t^{(maxh)}\}$.

Requirements for Feasibility to Large Scale Networks

The whole large scale network is used as test set. The latent variable matrix for the test set is obtained by out-of-sample extensions of the predictive KSC model and defined as $P_{test} = [e_1, \ldots, e_{N_{test}}]^\intercal$. Since we use the entire network as test set, therefore, $N_{test} = N$. The P_{test} matrix is a $N \times (maxk\text{-}1)$ dimensional matrix. So, we can store this P_{test} matrix in memory but cannot create an affinity matrix of size $N \times N$ due to memory constraints.

To make the approach feasible to large scale network we put a condition that the maximum size of a cluster at ground level cannot exceed $10,000$ (depending on the available computer memory) and the maximum number of clusters allowed at the ground level is $10,000$. This limits the size of the affinity matrix at that level of hierarchy to be less than $10,000 \times 10,000$. It also effects the choice of the initial value of the distance threshold $t^{(0)}$. If we set $t^{(0)}$ too high ($\gg 0.2$) then majority of the nodes at the ground level in the test case will fall in one community resulting in one giant connected component. If we set the value of $t^{(0)}$ too low ($\ll 0.1$) then we will end up with lot of singleton clusters at the ground level in the test case. In our experiments, we observed that the interval any value between $[0.1, 0.2]$ is good choice for the initial threshold value at level 0 of hierarchy. To be consistent we chose $t^{(0)} = 0.15$ for all the networks.

Multilevel Hierarchical KSC for Test Nodes

The validation set is a representative subset of the whole network as shown in [123]. Thus, the threshold set $\mathcal{T}$ can be used to obtain a hierarchical clustering for the entire network. To make the proposed approach self-tuned, we use $t^{(i)} > t^{(0)} > 0.15$, $i > 0$, during the test phase.

In order to prevent creating the affinity matrix for the large network we follow a greedy procedure. We select the projection of the first test node and calculate

its similarity with the projections of all the test nodes. We then locate the indices (j) of those projections s.t. $CosDist(e_1, e_j) < t^{(1)}$. If the total number of such indices is less than $10,000$ then we put them in cluster $C_1^{(1)}$ otherwise we select the first $10,000$ indices and place them in cluster $C_1^{(1)}$. This is due to the constraint that the size of a cluster $(C_1^{(1)})$ at ground level cannot exceed $10,000$. We then remove entries corresponding to those projections in P_{test} to obtain a reduced matrix. We perform this procedure iteratively until P_{test} is empty to obtain $C^{(1)} = \{C_1^{(1)}, \ldots, C_r^{(1)}\}$ where r is the total number of clusters at hierarchical level 1. After the 1st level, we use the same procedure that was for validation set i.e. creating an affinity matrix at each level using the cluster information along with the threshold set $\mathcal{T}$ to obtain the hierarchical structure in an agglomerative fashion. The cluster memberships are propagated iteratively from the 1st level to the highest level of hierarchy. The multilevel hierarchical kernel spectral clustering (MH-KSC) method is described in Algorithm 11.

Algorithm 11: MH-KSC Algorithm

Data: Graph $G = (V, E)$ representing large scale network.
Result: Multilevel Hierarchical Organization of the network.

Divide data into train,validation and test set, V_{tr},V_{valid},V_{test}.
Construct dataset $\mathcal{D} = \{x_i\}_{i=1}^{N_{tr}}$, $x_i \in \mathbb{R}^N$ from training set V_{tr}.
Perform KSC on $\mathcal{D}$ to obtain the predictive model as in (4.3).
Obtain $P_{valid} = [e_1, \ldots, e_{N_{valid}}]^\intercal$ using predictive model and V_{valid}.
Construct $R_{valid}^{(0)}(i,j) = CosDist(e_i, e_j) = 1 - \frac{e_i^\intercal e_j}{\|e_i\|\|e_j\|}$, $\forall e_i, e_j \in P_{valid}$.
Begin validation stage with: $h = 0$, $t^{(0)} = 0.15$.
$[C^{(0)}, k] = GreedyMaxOrder(R_{valid}^{(0)}, t^{(0)})$. `/* Algorithm 12 */`
Add $t^{(0)}$ to the set $\mathcal{T}$ and $C^{(0)}$ to the set $\mathcal{C}$.
while $k > 1$ **do**
    $h := h + 1$.
    Create $R_{valid}^{(h)}$ using $R_{valid}^{(h-1)}$ and $C^{(h-1)}$ as shown in (4.12).
    Calculate $t^{(h)}$ using equation (4.13).
    $[C^{(h)}, k] = GreedyMaxOrder(R_{valid}^{(h)}, t^{(h)})$.
    Add $t^{(h)}$ to the set $\mathcal{T}$ and $C^{(h)}$ to the set $\mathcal{C}$.
end
`/* Iterative procedure to get the set` $\mathcal{T}$. `*/`
Obtain P_{test} like P_{valid} and begin with: $h = 1$, $t^{(1)} \in \mathcal{T}$.
$[R_{test}^{(2)}, C^{(1)}, k] = GreedyFirstOrder(P_{test}, t^{(1)})$. `/* Algorithm 13 */`
Add $C^{(1)}$ to the set $\mathcal{C}$.
foreach $t^{(h)} \in \mathcal{T}$, $h > 1$ **do**
    $[C^{(h)}, k] = GreedyMaxOrder(R_{test}^{(h)}, t^{(h)})$.
    Add $C^{(h)}$ to the set $\mathcal{C}$.
    Create $R_{test}^{(h+1)}$ using $R_{test}^{(h)}$ and $C^{(h)}$ as shown in (4.12).
end
Obtain the set $\mathcal{C}$ for test set and propagate cluster memberships iteratively from 1st to coarsest level of hierarchy.

Algorithm 12: *GreedyMaxOrder* Algorithm

Data: Affinity matrix R and threshold t.
Result: Clustering information C and number of clusters k.

$k = 1$ and $totinst = 0$.
while $totinst \neq |R|$ **do**
  Find i in range $(1, |R|)$ for which number of instances j, s.t. $R(i, j) < t$, $j = 1, \ldots, |R|$, is maximum.
  Put indices of instance i and all instances j, s.t. $R(i, j) < t$, to C_k.
  $k := k + 1$ and $totinst := totinst + |C_k|$.
  Set all elements corresponding to the indices in C_k to ∞ in R.
  Add C_k to the set C.
end
$k := k - 1$.

Algorithm 13: *GreedyFirstOrder* Algorithm

Data: Projection matrix P_{test}, threshold $t^{(1)}$.
Result: Affinity matrix $R^{(2)}_{test}$, clustering information $C^{(1)}$ and k.

$k = 1$.
while $|P_{test}| \neq 0$ **do**
  Select 1st node and locate all nodes j for which $CosDist(e_1, e_j) < t^{(1)}$.
  Put all these instances in $C^{(1)}_k$ and to set $C^{(1)}$.
  $k := k + 1$.
  Remove these instances from P_{test} to have a reduced P_{test}.
end
/* The affinity matrix ($R^{(1)}_{test}$) is not calculated as it would be unfeasible to store an $N \times N$ matrix in memory. */
$k := k - 1$.
for $i = 1$ *to* $|C^{(1)}|$ **do**
  for $j = i + 1$ *to* $|C^{(1)}|$ **do**
    Calculate $R^{(2)}_{test}(i, j)$ as the average $CosDist(\cdot, \cdot)$ between the eigen-projections of the instances in $C^{(1)}_i$ and $C^{(1)}_j$.
  end
end
/* Affinity Matrix calculated for the first time. */

Time Complexity Analysis

The two steps in our proposed approach which require the maximum computation time are the out-of-sample extensions for the test set and the creation of the affinity matrix from the ground level clusters.

Since we use the entire network as test set the time required for out-of-sample extension is $O(N_{tr} \times N)$. Our greedy procedure to obtain the clustering information at the ground level $C^{(1)}$ requires $O(r \times N)$ computations where r is the number of clusters at 1st level of hierarchy for the test set. This is because for each cluster $C^{(1)}_1 \in C^{(1)}$ we remove all the indices belonging in that cluster from the matrix P_{test}. As a result the size of P_{test} decreases till it reduces to zero resulting in $O(r \times N)$ computations. The affinity matrix $R^{(1)}_{test}$

is a symmetric matrix so we only need to compute the upper of lower triangular matrix. The number of cluster-cluster similarities that we have to calculate is $\frac{r\times(r-1)}{2}$ where the size of each cluster at ground level can be maximum $10,000$.

However, as shown in [20], we can perform the out-of-sample extensions in parallel on n computers and rows of the affinity matrix can also be calculated in parallel thereby reducing the complexity by $\frac{1}{n}$.

4.3.3 Experiments

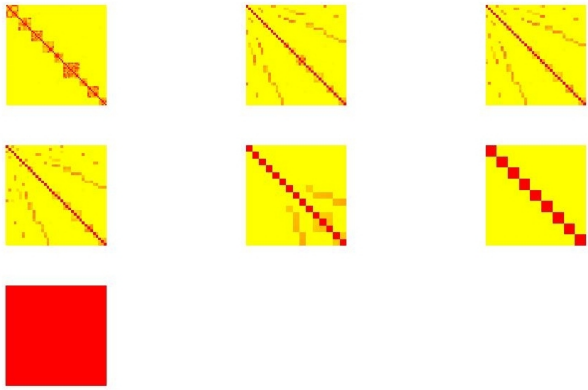
We conducted experiments on 2 synthetic datasets obtained from the toolkit in [103] and 7 real-world networks obtained from `http://snap.stanford.edu/data/index.html`.

Synthetic Network Experiments

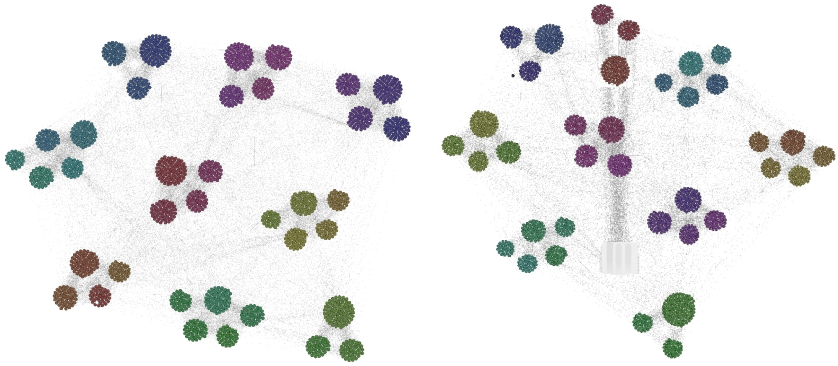
The synthetic networks are referred as Net_1 and Net_2 and have $2,000$ and $50,000$ nodes respectively. The ground truth for these 2 benchmark networks are known at 2 levels of hierarchy. These 2 levels of hierarchy for these benchmark networks are obtained by using 2 different mixing parameters i.e. μ_1 and μ_2 for macro and micro communities. We fixed $\mu_1 = 0.1$ and $\mu_2 = 0.2$ in our experiments. Since the ground truth is known beforehand, we evaluate the communities obtained by our proposed MH-KSC approach using an external quality metric like Adjusted Rand Index (ARI) and Variation of Information (VI) [45, 148]. We also evaluate the cluster information using internal cluster quality metrics like Modualrity (Q) [104] and Cut-Conductance (CC) [34]. We compare MH-KSC with Louvain, Infomap and OSLOM.

Figures 4.9 and 4.10 showcases the result of MH-KSC algorithm on the Net_1 and Net_2 respectively. From Figures 4.9a and 4.10a, we observe the affinity matrices generated corresponding to the test set for Net_1 and Net_2 respectively. From Figures 4.9b and 4.10b, we can observe the communities prevalent in the original network and the communities estimated by MH-KSC method for Net_1 and Net_2 respectively. In Net_1 there are 9 macro communities and 37 micro communities while in Net_2 there are 13 macro communities and 141 micro communities as depicted by Figures 4.9b and 4.10b.

Table 4.6 illustrates the first 10 levels of hierarchy for Net_1 and Net_2 and evaluates the clusters obtained at each level of hierarchy w.r.t. quality metrics ARI, VI, Q and CC. Higher values of ARI (close to 1) and lower values of VI (close to 0) represent good quality clusters. Both these external quality metrics are normalized as shown in [45]. Higher values of modularity (Q close to 1)



(a) Affinity matrices created at different levels of hierarchy for Net_1 network. The number of block-diagonals in each subgraph represents k at that level of hierarchy.

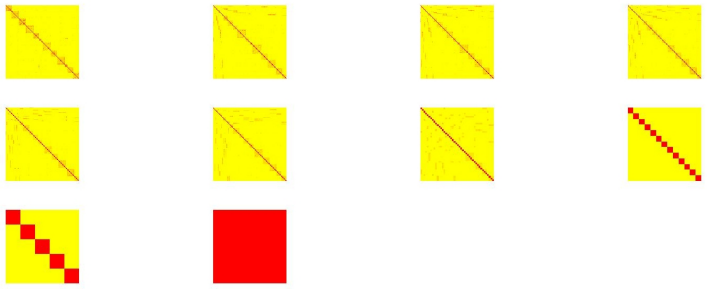


(b) Original hierarchical network (left) and estimated hierarchical network (right) for synthetic network with $10,000$ nodes. The orientation and position of the communities might vary in the two plots. Both plots have 3 clusters with 5 micro communities, 4 clusters with 4 micro communities and 2 clusters with 3 micro communities.

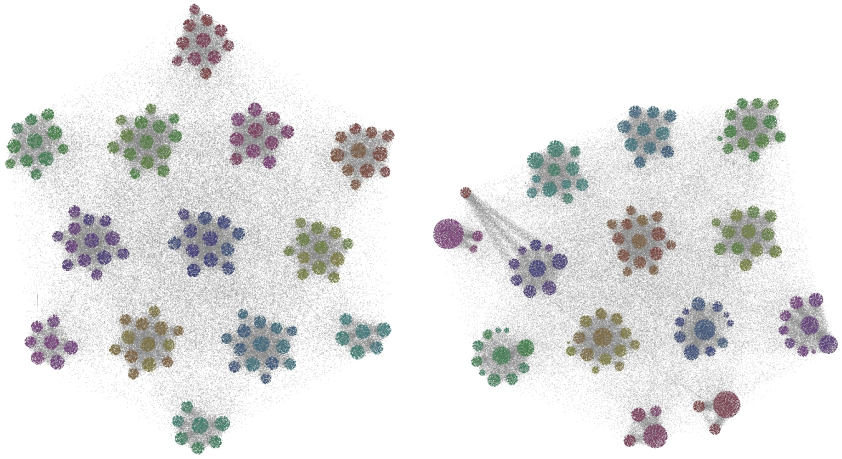
Figure 4.9: Result of MH-KSC algorithm on benchmark Net_1 network.

and lower values of cut-conductance (CC close to 0) indicate better clustering information.

Table 4.7 provides the result of Louvain, Infomap and OSLOM methods and compares it with the best levels of hierarchy for Net_1 and Net_2. The Louvain, Infomap and OSLOM methods require multiple runs as in each iteration they result in a different partition. We perform 10 runs and report the mean results in Table 4.7. From Table 4.7, it can be observed that the best results for Louvain



(a) Affinity matrices created at different levels of hierarchy for Net_2 network. The number of block-diagonals in each subgraph represents k at that level of hierarchy.



(b) Original hierarchical network (left) and estimated hierarchical network (right) for synthetic network with $50,000$ nodes. The orientation and position of the communities might vary in the two plots. Original network has 3 clusters with 11 micro communities, 2 clusters with 14, 13, 12 and 7 micro communities each, 1 cluster with 10 and another 1 with 6 micro communities. Estimated network has 3 clusters with 11 micro communities, 2 clusters 13, 10 and 3 micro communities each and 1 cluster with 14, 12, 9 and 4 micro communities respectively.

Figure 4.10: Result of MH-KSC algorithm on benchmark Net_2 network.

and Infomap methods generally occur at coarse levels of hierarchy w.r.t. to ARI, VI and Q metric. Thus, these two methods work well to identify macro communities. The Louvain method works the better than MH-KSC for Net_2 at macro and micro level. However, it cannot obtain similar quality micro

	Net_1					Net_2				
Hierarchy	k	ARI	VI	Q	CC	k	ARI	VI	Q	CC
10	-	-	-	-	-	**134**	**0.685**	**0.612**	0.66	**1.98e-05**
9	-	-	-	-	-	112	0.625	0.643	0.685	1.99e-05
8	-	-	-	-	-	106	0.61	0.667	0.691	1.99e-05
7	63	0.972	0.11	0.62	**4.74e-04**	103	0.595	0.692	0.694	**1.98e-05**
6	40	0.996	0.018	0.668	4.86e-04	97	0.53	0.77	0.706	1.99e-05
5	39	**0.996**	**0.016**	0.669	**4.834e-04**	87	0.47	0.90	0.722	1.99e-05
4	**37**	0.965	0.056	0.675	4.856e-04	44	0.636	0.74	**0.773**	1.99e-05
3	15	0.878	0.324	**0.765**	5.021e-04	**13**	**1.0**	**0.0**	**0.82**	2.0e-05
2	**9**	**1.0**	**0.0**	**0.786**	5.01e-04	5	0.12	1.643	0.376	2.12e-05
1	1	0.0	2.19	0.0	5.0e-04	1	0.0	2.544	0.0	2.0e-05

Table 4.6: Number of clusters (k) for top 10 levels of hierarchy by MH-KSC method. The number of clusters close to the actual number, the best and second best results are highlighted. For Net_1 only 7 levels of hierarchy are identified by MH-KSC, rest are represented by '-'. The MH-KSC method provides more insight by identifying several meaningful levels of hierarchy with good clusters w.r.t. quality metrics like ARI, VI, Q and CC.

Method	Net_1						Net_2					
	Level	k	ARI	VI	Q	CC	**Level**	k	ARI	VI	Q	CC
Louvain	2	32	0.84	0.215	**0.693**	4.87e-05	3	135	0.853	0.396	**0.687**	**1.98e-05**
	1	9	**1.0**	**0.0**	**0.786**	**5.01e-04**	1	13	**1.0**	**0.0**	**0.82**	**2.0e-05**
Infomap	2	8	0.915	0.132	0.771	5.03e-04	3	590	0.003	8.58	0.003	**1.98e-05**
	1	6	0.192	1.965	0.487	5.07e-04	1	13	**1.0**	**0.0**	**0.82**	**2.0e-05**
OSLOM	2	38	0.988	0.037	0.655	4.839e-04	2	141	**0.96**	**0.214**	0.64	2.07e-05
	1	9	**1.0**	**0.0**	**0.786**	**5.01e-04**	1	29	0.74	0.633	076	2.08e-05
MH-KSC	5	39	**0.996**	**0.016**	0.67	**4.83e-04**	10	134	0.685	0.612	0.66	**1.98e-05**
	2	9	**1.0**	**0.0**	**0.786**	**5.01e-04**	3	13	**1.0**	**0.0**	**0.82**	**2.0e-05**

Table 4.7: 2 best level of hierarchy obtained by Louvain, Infomap, OSLOM and MH-KSC methods on Net_1 and Net_2 benchmark networks. The best results w.r.t. various quality metrics when compared with the ground truth communities for each benchmark network is highlighted.

communities when compared with MH-KSC method for Net_1 as inferred from Table 4.7. The Infomap method performs the worst among all the methods w.r.t. detection of communities at finer levels of granularity. OSLOM performs well w.r.t. to locating both macro communities for Net_1 and micro communities for Net_2 as observed from Table 4.7. It performs better than any method w.r.t. locating micro communities for Net_2 w.r.t. ARI and VI metric. However, it performs worst while trying to identify the macro communities for the same benchmark network. The MH-KSC performs best on Net_1 while it performs better w.r.t. locating macro communities for Net_2.

Real-Life Network Experiments

We experimented on 7 real-life networks from the Stanford SNAP datasets `http://snap.stanford.edu/data/index.html`. These networks are anonymous networks and are converted to undirected and unweighted networks before performing experiments on them. Table 4.8 provides information about topological characteristics of these real-life networks. The Fb and Epn networks are social networks, PGP is a trust based network, Cond is a collaboration network between researchers, Enr is an email network, Imdb is an actor-actor collaboration network and Utube is a web graph depicting friendship between the users of Youtube.

Network	Nodes	Edges	CCF
Facebook (Fb)	4,039	88,234	0.6055
PGPnet (PGP)	10,876	39,994	0.008
Cond-mat (Cond)	23,133	186,936	0.6334
Enron (Enr)	36,692	367,662	0.497
Epinions (Epn)	75,879	508,837	0.1378
Imdb-Actor (Imdb)	383,640	1,342,595	0.453
Youtube (Utube)	1,134,890	2,987,624	0.081

Table 4.8: Nodes (V), Edges (E) and Clustering Coefficients (CCF) for each network

In case of real-life networks the true hierarchical structure is not known beforehand. Hence, it is important to show whether they exhibit hierarchical organization which can be tested by identifying good quality clusters w.r.t. internal quality metrics like Q and CC at multiple levels of hierarchy.

We showcase the results for 10 levels of hierarchy in a bottom-up fashion for the MH-KSC method in Table 4.9. The coarsest level of hierarchy has all nodes in one community and is not very insightful. Clusters at very coarse levels of granularity comprises giant connected components. So, it is more meaningful to give more emphasis to fine grained clusters at lower levels of hierarchy. To show that real-life networks exhibit hierarchy we evaluate our proposed MH-KSC approach in Table 4.9.

We compare MH-KSC algorithm with Louvain [36], Infomap [42] and OSLOM [40]. We perform 10 runs for each of these methods as they generate a separate partition each time when they are executed. The mean results of Louvain method is reported in Table 4.10. Table 4.11 showcases the results for Infomap and OSLOM method.

From Table 4.10 it is evident that the Louvain method works best w.r.t. the modularity (Q) criterion. This aligns with methodology as it is trying to optimize for Q. However, the Louvain method always performs worse than

		Hierarchical Organization									
Network	**Metrics**	**Level 12**	**Level 11**	**Level 10**	**Level 9**	**Level 8**	**Level 7**	**Level 6**	**Level 5**	**Level 4**	**Level 3**
Fb	k	358	192	152	121	105	90	71	43	37	21
	Q	0.604	0.764	0.769	0.789	0.792	0.81	0.812	0.818	0.821	**0.83**
	CC	**2.47e-05**	1.56e-04	2.38e-04	1.91e-04	1.95e-04	1.63e-04	2.16e-04	1.76e-04	2.44e-04	2.4e-04
PGP	k	345	274	202	156	129	83	59	46	24	19
	Q	0.682	0.693	0.705	0.715	0.725	0.727	0.728	**0.729**	0.701	0.698
	CC	8.48e-05	9.84e-05	5.88e-05	1.38e-04	7.2e-05	8.03e-05	1.0e-04	1.07e-04	4.13e-04	**4.89e-05**
Cond	k	2676	1171	621	324	171	102	80	58	41	24
	Q	0.5	0.567	0.586	0.611	**0.615**	0.614	0.582	0.582	0.574	0.515
	CC	2.49e-05	2.6e-05	3.7e-05	3.52e-05	3.6e-05	5.86e-05	2.37e-05	3.45e-05	1.43e-05	**1.4e-05**
Enr	k	2208	1002	464	303	211	163	119	76	59	48
	Q	0.30	0.388	0.444	0.451	**0.454**	0.427	0.43	0.325	0.328	0.271
	CC	**1.19e-05**	3.18e-05	3.1e-05	5.3e-05	7.04e-05	2.69e-04	2.2e-03	1.651e-04	2.56e-05	5.46e-05
Epn	k	8808	3133	1964	957	351	220	166	97	66	26
	Q	0.105	0.156	0.158	0.176	0.184	0.183	**0.186**	0.184	0.146	0.006
	CC	**1.4e-06**	3.1e-06	6.4e-06	7.0e-06	9.5e-06	1.26e-05	7.0e-06	9.0e-06	2.42e-05	7.8e-06
Imdb	k	7431	1609	890	468	313	200	130	72	46	21
	Q	0.357	0.47	0.473	0.485	0.503	**0.521**	0.508	0.514	0.513	0.406
	CC	1.43e-06	2.78e-06	2.79e-06	5.6e-06	4.24e-06	5.6e-06	6.42e-06	1.99e-06	7.46e-06	**9.2e-07**
Utube	k	9984	2185	529	274	180	131	100	71	46	26
	Q	0.524	0.439	0.679	**0.682**	0.599	0.491	0.486	0.483	0.306	0.303
	CC	**2.65e-07**	3.0e-07	1.3e-06	2.4e-06	1.0e-06	7.6e-06	1.03e-5	1.07e-05	2.33e-05	1.55e-04

Table 4.9: Results on MH-KSC algorithm on 7 real-life networks using quality metrics Q and CC. The best results corresponding to each metric for individual networks are highlighted.

		Hierarchical Organization					
Network	**Metrics**	**Level 6**	**Level 5**	**Level 4**	**Level 3**	**Level 2**	**Level 1**
Fb	k	-	-	-	225	155	151
	Q	-	-	-	0.82	0.846	**0.847**
	CC	-	-	-	**9.88e-05**	1.33e-04	1.32e-04
PGP	k	-	-	2392	566	154	100
	Q	-	-	0.705	0.857	0.882	**0.884**
	CC	-	-	**4.95e-05**	8.66e-05	6.8e-05	1.0e-04
Cond	k	-	-	6732	1825	1066	1011
	Q	-	-	0.56	0.7	0.731	**0.732**
	CC	-	-	**1.56e-05**	2.97e-05	3.49e-05	4.15e-05
Enr	k	-	-	4001	1433	1237	1230
	Q	-	-	0.546	0.608	0.613	**0.614**
	CC	-	-	**1.28e-05**	1.88e-05	4.58e-05	6.48e-05
Epn	k	10351	2818	1574	1325	1301	1300
	Q	0.287	0.319	0.323	0.324	0.324	**0.324**
	CC	**1.86e-06**	4.2e-06	4.25e-06	5.57e-06	6.75e-06	1.13e-05
Imdb	k	-	22613	4544	3910	3815	3804
	Q	-	0.591	0.727	0.729	0.729	**0.729**
	CC	-	**1.0e-06**	1.0e-06	1.85e-06	2.5e-06	2.82e-06
Utube	k	33623	11587	6964	6450	6369	6364
	Q	0.696	0.711	0.714	0.715	0.715	**0.715**
	CC	**1.38e-06**	2.22e-06	3.25e-06	3.98e-06	4.06e-06	9.96e-06

Table 4.10: Results of Louvain method on 7 real-life networks indicating the top 6 levels of hierarchy. The best results are highlighted and '-' is used in case the metric is not applicable due to absence of partitions.

MH-KSC algorithm w.r.t. cut-conductance CC as observed from Tables 4.9 and 4.10. Another issue with the Louvain method is that except for the Fb and PGP networks it is not able to detect (< 1000 clusters) high quality clusters at coarser levels of granularity. This is attributed to the resolution limit problem suffered by Louvain method. From Table 4.11 we observe that the Infomap

		Infomap		OSLOM				
		Hierarchical Info		Hierarchical Info				
Network	**Metrics**	**Level 2**	**Level 1**	**Level 5**	**Level 4**	**Level 3**	**Level 2**	**Level 1**
Fb	k	325	131	-	161	50	27	21
	Q	0.055	**0.763**	-	0.045	0.133	0.352	**0.415**
	CC	**2.86e-05**	2.3e-04	-	**2.0e-04**	2.0e-04	3.0e-04	3.0e-04
PGP	k	85	65	431	143	51	48	45
	Q	0.041	**0.862**	0.748	**0.799**	0.709	0.709	0.709
	CC	1.66e-04	**1.40e-04**	1.74e-04	**5.32e-05**	2.06e-04	1.56e-04	6.64e-05
Cond	k	1009	173	4092	2211	1745	1613	1468
	Q	**0.648**	0.027	0.483	0.574	0.615	**0.615**	0.05
	CC	**1.71e-05**	2.78e-05	1.77e-05	2.48e-05	3.04e-05	6.56e-05	**1.16e-05**
Enr	k	1920	1084	-	3149	2177	2014	1970
	Q	0.015	**0.151**	-	0.317	0.382	0.412	**0.442**
	CC	**1.83e-05**	8.39e-04	-	**1.75e-05**	4.96e-05	9.92e-05	7.22e-05
Epn	k	14170	50	1693	584	206	30	25
	Q	5.3e-06	**4.48e-04**	0.162	0.226	**0.239**	0.098	0.019
	CC	**3.97e-06**	4.63e-05	1.23e-05	9.75e-06	2.45e-05	8.2e-06	**7.9e-06**
Imdb	k	14308	3238	-	7469	2639	2017	2082
	Q	0.04	**0.707**	-	0.045	0.092	0.1	0.115
	CC	**1.23e-06**	4.72e-06	-	**1.35e-06**	2.03e-06	7.95r-06	1.17e-05
Utube	k	10703	976	18539	6547	4184	2003	1908
	Q	0.035	**0.698**	0.396	0.53	**0.588**	0.487	0.027
	CC	**1.38e-06**	5.56e-06	1.52e-06	3.1e-07	**2.72e-07**	6.1e-06	5.69e-06

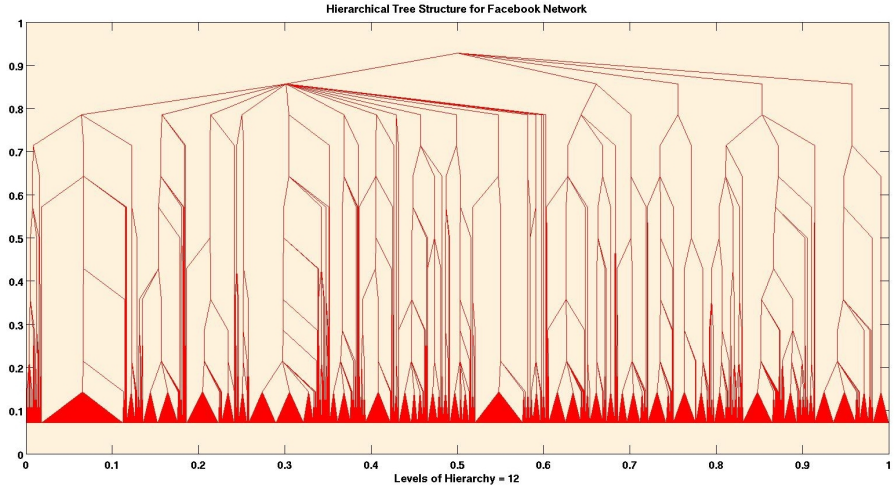
Table 4.11: Results of Infomap and OSLOM methods. The best results for each method corresponding to each network is highlighted and '-' represent not applicable cases.

method produces only 2 levels of hierarchy. In most of the cases, the clusters at one level of hierarchy perform good w.r.t. only 1 quality metric except the PGP and Cond networks. The difference between the quality of the clusters at the 2 levels of hierarchy is quite drastic. This reflects that the Infomap method is not very consistent w.r.t. various quality metrics.

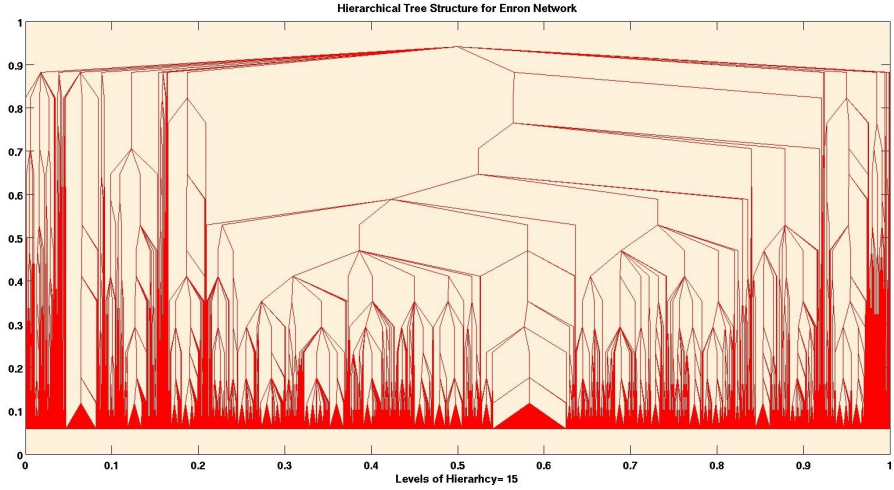
We compare the performance of MH-KSC method with OSLOM in detail. From Tables 4.9 and 4.10 we observe that the MH-KSC technique outperforms OSLOM w.r.t. both quality metrics for Fb, Enr, Imdb and Utube networks while OSLOM does the same only for Cond network. In case of PGP, Cond and Epn networks OSLOM results in better Q than MH-KSC. However, MH-KSC approach has better CC value for PGP and Epn networks. For large scale networks like Enr, Imdb and Utube, OSLOM cannot identify good quality coarser clusters i.e. number of clusters detected are always > 1000.

4.3.4 Visualization and Illustrations

We provide a tree based visualization of the multilevel hierarchical organization for Fb and Enr networks in Figure 4.11. The hierarchial structure is depicted as tree for Fb and Enr network in Figures 4.11a and 4.11b respectively.



(a) Multilevel Hierarchical Organization for Fb network



(b) Multilevel Hierarchical Organization for Enr network

Figure 4.11: Tree based visualization of the multilevel hierarchical organization prevalent in 2 real-life networks .

We plot the results corresponding to fine, intermediate and coarse levels of hierarchy for PGP network using the software provided in [40]. The software requires all the nodes in the network along with 2 levels of hierarchy. In Figure

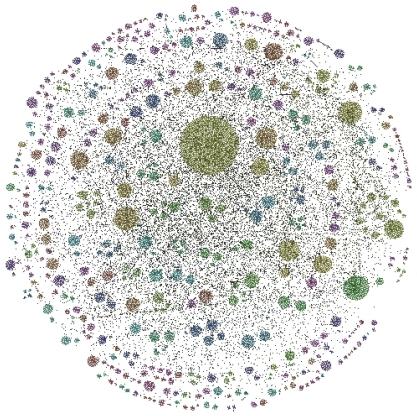
4.12 we plot the results for PGP net corresponding to MH-KSC algorithm using 2 fine, 4 intermediate and 2 coarse levels of the hierarchical organization. For Louvain method we use 4th and 3rd level of hierarchy as inputs for the finest level, 3rd and 2nd level of hierarchy as inputs for intermediate level and 2nd and 1st level of hierarchy as inputs for coarsest level plot. The Infomap method only generates 2 level of hierarchy which correspond to a coarse level plot. Similarly, for OSLOM we plot a fine and coarse level plot. The results for Louvain, Infomap and OSLOM methods are depicted in Figure 4.13.

Figures 4.12 and 4.13 shows that MH-KSC algorithm allows to depict richer structures than the other methods. It has more flexibility and allows the visualization at coarser, intermediate and finer levels of granularity. From Figures 4.13a,4.13b, 4.13c and Table 4.10, we observe that the Louvain method can only detect quality clusters at finer levels of granularity and cannot detect less than $1,000$ communities. While the Infomap method can only locate giant connected components for the PGP network as observed from Figure 4.13d and Table 4.11. The OSLOM method also seems to work reasonably well as observed from Figures 4.13e and 4.13f. However, it detects fewer levels of hierarchy and thus has less flexibility in terms of selection for the level of hierarchy than the proposed MH-KSC approach.

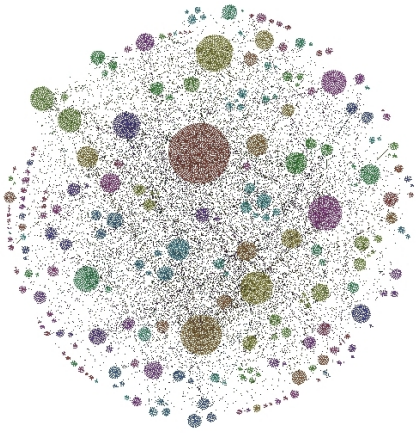
We provide a visualization of the 2 best layers of hierarchy for Epn network based on the Q and the CC criterion for MH-KSC, Louvain, Infomap and OSLOM methods respectively in Figures 4.14 and 4.15. The result for Infomap method in both the figures is the same as it only generates 2 levels of hierarchy.

4.3.5 Conclusion

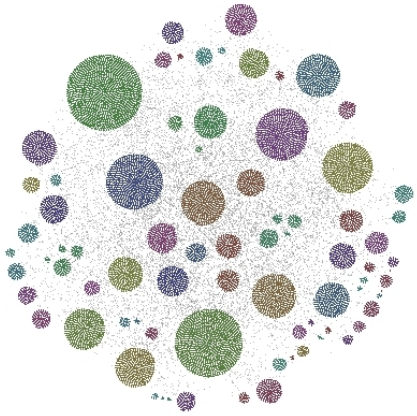
We proposed a new multilevel hierarchical kernel spectral clustering (MH-KSC) algorithm. The approach relies on the KSC primal-dual formulation and exploits the structure of the projections in the eigenspace. The projections of the validation set provided a set ($\mathcal{T}$) of increasing distance thresholds. These distance thresholds were used along with affinity matrix obtained from the projections in an iterative procedure to obtain a multilevel hierarchical organization in a bottom-up fashion. We highlighted some of the necessary conditions for the feasibility of the approach to large scale networks. We showed that many real-life networks exhibit hierarchical structure. Our proposed approach was able to identify good quality clusters for both coarse as well as fine levels of granularity. We compared and evaluated our MH-KSC approach against several state-of-the-art large scale hierarchical community detection techniques.



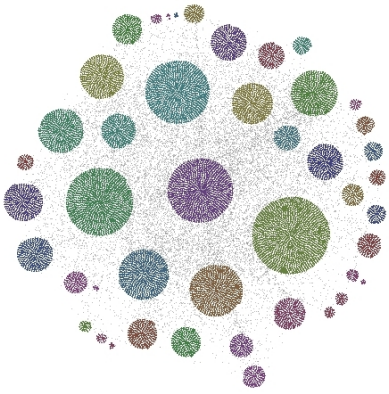
(a) Many Micro Communities at Finer Levels



(b) Less Micro Communites at Intermediate Levels

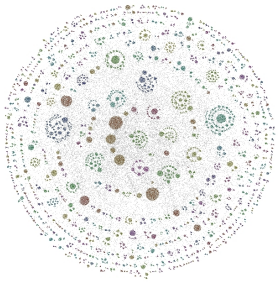


(c) Few Micro and Few Macro Communities at Intermediate Levels

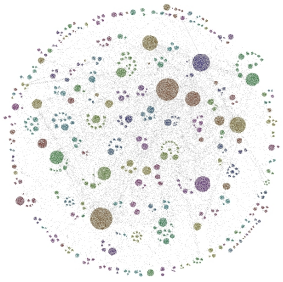


(d) Some Pre-dominant Macro Communities at Coarser Levels

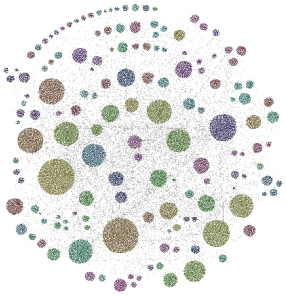
Figure 4.12: Results of the **MH-KSC** algorithm for the PGP network. Clusters with same colour are part of one community.



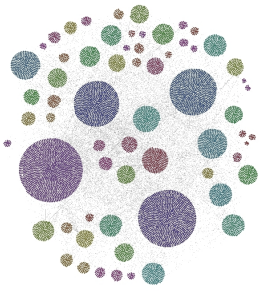
(a) Many Micro Communities at Finer Levels for **Louvain** Method



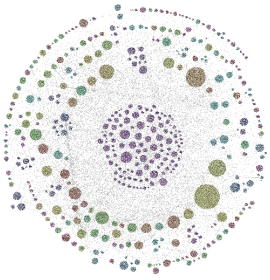
(b) Few Micro Communities at Intermediate Levels for **Louvain** Method



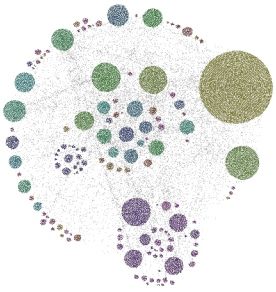
(c) Few Micro and Macro Communities at Coarser Levels for **Louvain** Method



(d) Some Macro Communities at Coarser Levels for **Infomap** Method



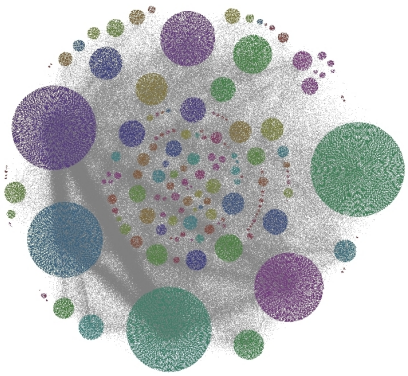
(e) Many Micro Communities at Finer Levels for **OSLOM** Method



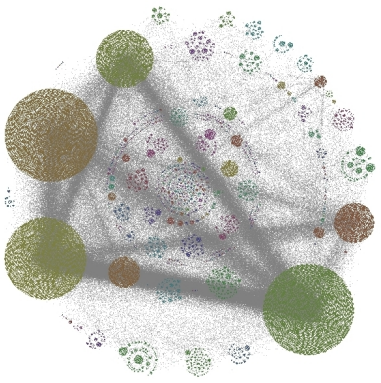
(f) Few Micro and Macro Communities at Coarser Levels for **OSLOM** Method

Figure 4.13: Results of Louvain, Infomap and OSLOM methods for PGP network. Clusters with same colour are part of one community. Louvain method can only provide results with few macro and many micro communities. Infomap method only has 2 coarse levels of hierarchy. OSLOM can provide both micro and macro communities but cannot detect as many meaningful intermediate layers as the MH-KSC method.

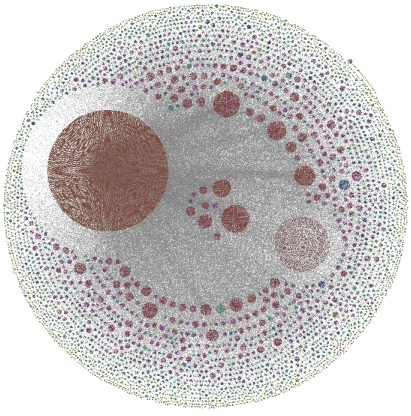
.



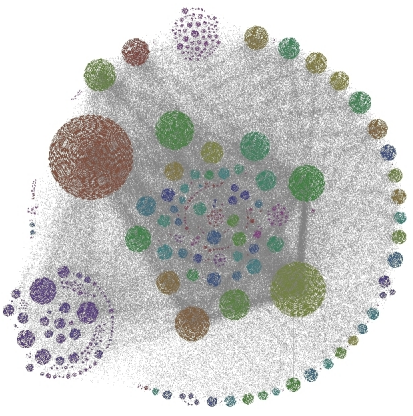
(a) Best Result for MH-KSC algorithm



(b) Best Result for Louvain method

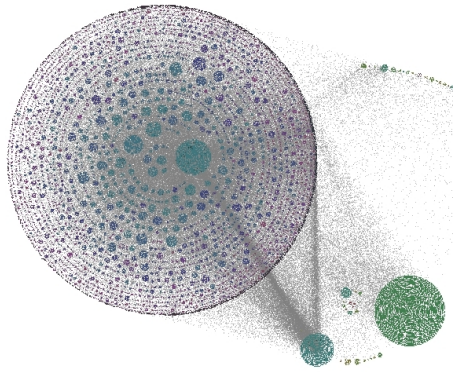


(c) Best Result for Infomap approach

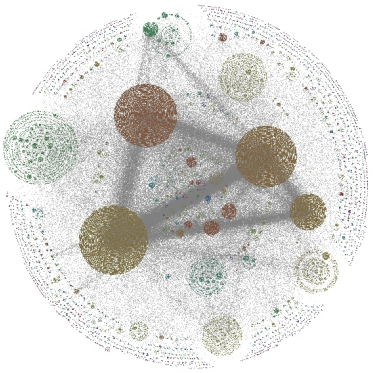


(d) Best Result for OSLOM

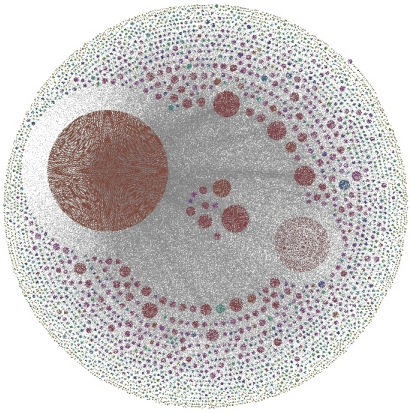
Figure 4.14: Representing the 2 best levels of hierarchy for Epn network w.r.t. the modularity (Q) criterion for various techniques.



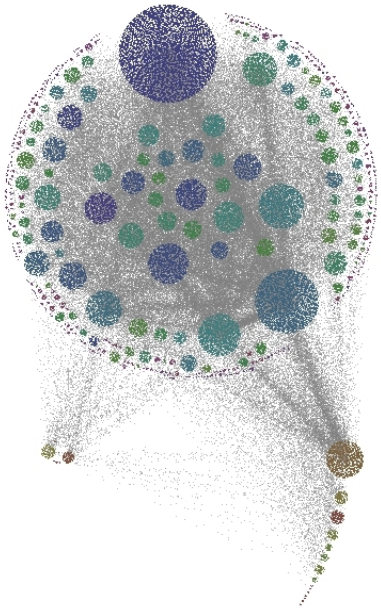
(a) Best Result for MH-KSC algorithm



(b) Best Result for Louvain method



(c) Best Result for Infomap approach



(d) Best Result for OSLOM

Figure 4.15: Representing the 2 best levels of hierarchy for Epn network w.r.t. the modularity (CC) criterion for various techniques.

Chapter 5

Hierarchical & Sparse Kernel Spectral Data Clustering

This chapter comprises previously published articles including:

1) Mall R., Mehrkanoon S., Suykens J.A.K., **"Identifying intervals for hierarchical clustering using the Gershgorin circle theorem"**, Pattern Recognition Letters, 55, April 2015, pp. 1-7.

2) Mall R., Langone R., Suykens J.A.K., **"Agglomerative hierarchical kernel spectral data clustering"**, IEEE SSCI CIDM, Dec 2014, pp. 9-16.

3) Mall R., Mehrkanoon S., Langone R., Suykens J.A.K., **"Optimal reduced sets for sparse kernel spectral clustering"**, IEEE IJCNN, July 2014, pp. 2436-2443.

Keywords—Gershgorin circle theorem; k clusters; eigengap; agglomerative hierarchical kernel spectral clustering; sparse kernel spectral clustering; group lasso; L_0 norm penalty;

5.1 Identifying Intervals for Hierarchical Clustering[1]

Abstract—In this work, we present a novel method for unraveling the hierarchical clusters in a given dataset using the Gershgorin circle theorem. The Gershgorin circle theorem provides upper bounds on the eigenvalues of the normalized Laplacian matrix. This can be utilized to determine the ideal range for the number of clusters (k) at different levels of hierarchy in a given dataset. The obtained intervals help to reduce the search space for identifying the ideal value of k at each level. Another advantage is that we don't need to perform the computationally expensive eigen-decomposition step to obtain the eigenvalues and eigenvectors. The intervals provided for k can be considered as input for any spectral clustering method which uses a normalized Laplacian matrix. We show the effectiveness of the method in combination with a spectral clustering method to generate hierarchical clusters for several synthetic and real world datasets.

5.1.1 Introduction

Clustering algorithms are widely used tools in fields like data mining, machine learning, graph compression, probability density estimation and many other tasks. The aim of clustering is to organize data into natural groups in a given dataset. Clusters are defined such that the data present within the group are more similar to each other in comparison to the data between clusters. Clusters are ubiquitous and application of clustering algorithms span from domains like market segmentation, biology (taxonomy of plants and animals), libraries (ordering books), WWW (clustering web log data to identify groups) and study of the universe (grouping stars based on similarity) etc. A variety of clustering algorithms exist in literature [149, 150, 151, 152, 153, 154, 46, 48, 47, 13, 20, 126, 21] etc. In this work, we show the effectiveness of the intervals provided by our proposed approach in combination with KSC [13] to obtain inference about the hierarchical structure of a given dataset.

Most clustering algorithms require the end-user to provide the number of clusters (referred as k). This is also applicable for KSC. Though for KSC, we have several model selection methods like Balanced Line Fit (BLF) [13], Balanced Angular Fit (BAF)[20] and Fisher criterion to estimate the number of clusters k which are computationally expensive. However, it is not always obvious to

[1]This section consists of Section $1-5$ from Mall R., Mehrkanoon S., Suykens J.A.K., **"Identifying intervals for hierarchical clustering using the Gershgorin circle theorem"**, Pattern Recognition Letters, 55, April 2015, pp. 1-7.

determine the ideal value for k. It is best to choose an ideal value for k based on prior information about the data. But such information is not always available and it makes exploratory data analysis quite difficult particularly when the dimension of the input space is large.

A hierarchical kernel spectral clustering method was proposed in [52]. In order to determine the optimal number of clusters (k) at a given level of hierarchy the authors in [52] searched over a grid of values for each kernel parameter σ. They select the value of k corresponding to which the model selection criterion (BLF) is maximum. A disadvantage of this method is that for each level of hierarchy a grid search has to be performed on all the grid values for k. In [20], the authors showed that the BAF criterion has multiple peaks for different values of k corresponding to a given value of σ. These peaks correspond to optimal value of k at different levels of hierarchy. In this work, we present a novel method to determine the ideal range for k at different levels of hierarchy in a given dataset using the Gershgorin circle theorem [78].

A major advantage of the approach proposed in this work is that we provide intervals for different levels of hierarchy before applying any clustering algorithm (or using any quality metric) unlike other hierarchical clustering algorithms. The Gershgorin circle theorem provides lower and upper bounds to the eigenvalues of a normalized Laplacian matrix. Using concepts similar to the eigengap, we can use these upper bounds on the eigenvalues to estimate the number of clusters at each level of hierarchy. Another advantage of this method is that we overcome the computationally expensive eigen-decomposition step. We show the efficiency of the proposed method by providing these discretized intervals (range) as input to KSC for identifying the hierarchy of clusters. These intervals can be used as starting point for any spectral clustering method which works on a normalized Laplacian matrix to identify the k clusters in the given dataset. The method works effectively for several synthetic and real-world datasets as observed from our experiments.

Several approaches have been proposed to determine the ideal value of k for a given dataset [155, 156, 157, 158, 159, 160, 161, 162, 46, 48, 163, 164, 165]. Most of these methods extend the k-means or expectation maximization and proceed by splitting or merging techniques to increase or decrease the number of clusters respectively.

In this work, we propose a novel method for providing an interval (a range) for the number of clusters (k) in a given dataset. This interval helps to reduce the search space for the ideal value of k. The method uses the Gershgorin circle theorem along with upper bounds on the eigenvalues for this purpose. There are several advantages of the proposed approach. It allows us to identify intervals for the number of clusters (k) at different levels of hierarchy. We overcome

the requirement of performing the eigen-decomposition step, thereby reducing the computational cost. There is no underlying assumption or prior knowledge requirement about the data.

5.1.2 Proposed Method

We consider the normalized Laplacian matrix (L) related to the Random Walk model as defined in [166]. In this model, the Laplacian matrix is defined as the transition matrix. This can mathematically be represented as $L = D^{-1}S$ where S is the affinity matrix and D is the diagonal degree matrix such that $D_{ii} = \sum_j S_{ij}$. For this model, the highest eigenvalue (equal to 1) has a multiplicity of k in case of k well-separated clusters and a gap between the eigenvalues indicates the existence of clusters. But in real world scenarios there is presence of overlap between the clusters and the eigenvalues deviate from 1. Then it becomes difficult to identify the threshold values to determine the k clusters. Therefore, we utilize the Gershgorin circle theorem to use the upper bounds on the eigenvalues to construct intervals for determining the ranges for the number of clusters (k) at each level of hierarchy in a given dataset. (If we use the normalized Laplacian $[L = I - D^{-1}S]$ matrix then it would be required to use the lower bounds on the eigenvalues to construct the intervals). The actual eigenvalues are obtained by performing eigen-decomposition on Laplacian matrix L

$$Lv_j = \lambda_j v_j, j = 1, \ldots, N \tag{5.1}$$

where N is the number of eigenvalues.

Let $L \in \mathbb{R}^{N \times N}$ be a square matrix which can be decomposed into the sum $L = C + R$ where C is a diagonal matrix and R is a matrix whose diagonal entries are all zero. Let also $c_i = C_{ii}$, $r_{ij} = R_{ij}$ and $\bar{r}_i = \sum_{j=1}^{N} |r_{ij}|$. Then, according to the Gershgorin circle theorem [78]:

- The ith Gershgorin disc associated to the ith row of L is defined as the interval $I_i = [c_i - \bar{r}_i, c_i + \bar{r}_i]$. The quantities c_i and r_i are respectively referred to as the center and the radius of disc I_i respectively.
- Every eigenvalue of L lies within at least one of the Gershgorin discs I_i.
- The following condition holds:

  $$c_j - \bar{r}_j \leq \bar{\lambda}_j \leq c_j + \bar{r}_j \tag{5.2}$$

  with $\bar{\lambda}_j$ corresponding to disc I_j. For each eigenvalue of L, λ_i, $i = 1, \ldots, N$ there exists an upper bound $\bar{\lambda}_j$, $j = 1, \ldots, N$ where i need not necessarily be equal to j. Thus, we have $\lambda_i \leq \bar{\lambda}_j$.

We are provided with a dataset $\mathcal{D} = \{x_1, x_2, \ldots x_N\}$ where $x_i \in \mathbb{R}^d$. We then construct the affinity matrix S by calculating similarity between each x_i and x_j. It was shown in [165] that the eigenvalues are positively correlated to the degree distribution in case of real world datasets. This relation can be approximated by a linear function. We empirically observe similar correlations between the degree distribution and these upper bounds i.e. $\bar{\lambda}_j$ generated by the Gershgorin circle theorem. In [167], the authors perform stability analysis of clustering across multiple levels of hierarchy. They analyze the dynamics of the Potts model and conclude that hierarchical information for multivariate spin configuration could be inferred from spectral significance of a Markov process. In [167] it was suggested that for every stationary distribution (a level of hierarchy) the spins of the whole system reach the same value. These spin values are dependent on the different eigenvalues and the difference between the eigenvalues of the system. Inspired from this concept we propose a method to use the distance between the upper bounds to determine the intervals to search for optimal values of k for different levels of hierarchy.

We sort these $\bar{\lambda}_j$ in descending order such that $\bar{\lambda}_1 \geq \bar{\lambda}_2 \geq \ldots \geq \bar{\lambda}_N$. Similarly, all the eigenvalues are sorted in descending order such that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N$. The relation $\lambda_1 \leq \bar{\lambda}_1$ holds in accordance to the Gershgorin circle theorem. We propose a heuristic i.e. we calculate the distance of each $\bar{\lambda}_j$ from $\bar{\lambda}_1$ to obtain δ_j and maintain this value in a *dist* vector. The distance value is defined as:

$$\delta_j = Dist(\bar{\lambda}_1, \bar{\lambda}_j) \tag{5.3}$$

where $Dist(\cdot, \cdot)$ is the Euclidean distance function.

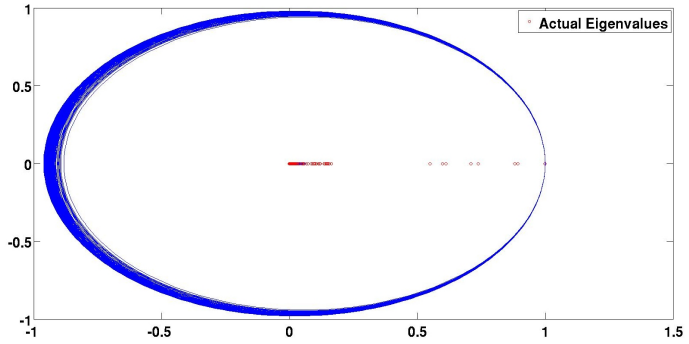
We then sort this *dist* vector in descending order. In order to estimate the intervals, we use a concept similar to the notion of eigengap. We first try to locate the number of terms which are exactly the same as $\bar{\lambda}_1$. This can be obtained by calculating the number of terms in the *dist* vector such that $Dist(\bar{\lambda}_1, \bar{\lambda}_j) = 0$. This gives the lower limit for the first interval say $l_1 = n_1$. If there is no $\bar{\lambda}_j$ which is exactly equal to $\bar{\lambda}_1$ then the lower limit for the first interval is 1. We then move to the first term say $\bar{\lambda}_p$ in the sorted *dist* vector which is different from $\bar{\lambda}_1$. We calculate the number of terms say n_2 in the *dist* vector which are at the same distance as $\bar{\lambda}_p$ from $\bar{\lambda}_1$. The upper limit for the first interval is then defined as the sum of the lower limit and the number of terms at the same distance as $\bar{\lambda}_p$ i.e. $u_1 = n_1 + n_2$. This upper limit is also considered as the lower limit for the second interval. We continue this process till we obtain all the intervals. Since we are using the bounds on the eigenvalues $(\bar{\lambda}_j)$ instead of the actual eigenvalues (λ_j), it is better to estimate intervals rather than the exact number of clusters. If the length of an interval is say 1 or 2, the search space will be too small. On the other hand, if the length of an interval is too large then we might miss hierarchical structure. So we put

a heuristic that the minimum length of an interval should be 3. The intervals provide a hierarchy in a top-down fashion i.e. the number of clusters increase as the level of hierarchy increases. Algorithm 14 provides details of the steps involved to obtain the intervals for each level of hierarchy of a given dataset.

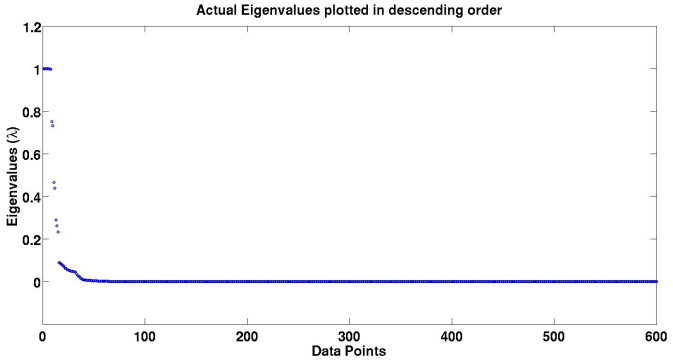
Algorithm 14: Algorithm for estimation of intervals for k

Data: Dataset $\mathcal{D} = \{x_1, x_2, \ldots, x_N\}$
Result: Intervals for number of clusters (k) for different levels of hierarchy
Construct the affinity matrix S which comprises S_{ij}
Calculate the diagonal degree matrix $D_{ii} = \sum_{j=1}^{N} S_{ij}$.
Obtain the Laplacian matrix $L = D^{-1}S$.
Obtain the matrices C and R from L matrix using Gershgorin theorem.
Calculate $\bar{\lambda}_j = c_j + \bar{r}_j$ using C and R matrices.
Sort these $\bar{\lambda}_j$, $j = 1, \ldots, N$.
Obtain the $dist$ vector by appending the distance (δ_j) of each $\bar{\lambda}_j$ from $\bar{\lambda}_1$.
Sort the $dist$ vector and initialize $i = 1$ for the count of number of terms explored & $h = 1$ for the level of hierarchy.
`// Initial Condition`
Calculate $\delta_i = Dist(\bar{\lambda}_1, \bar{\lambda}_i)$.
l_h = Number of terms which have same distance as δ_i.
`// Lower limit for` 1st `level of hierarchy`
Increase i by l_h i.e $i := i + l_h$.
Recalculate $\delta_i = Dist(\bar{\lambda}_1, \bar{\lambda}_i)$.
$u_h = l_h +$ Number of terms which have same distance as δ_i.
`// Upper limit for` 1st `level of hierarchy`
while $i \leq N - 1$ **do**
    while $u_h - l_h < 3$ **do**
        Change i such that $i := u_h + 1$.
        Calculate $\delta_i = Dist(\bar{\lambda}_1, \bar{\lambda}_i)$ & $l_h = u_h$.
        Increase u_h such that $u_h = u_h +$Number of terms which have same distance as δ_i.
    end
    Increase h by 1 such that $h := h + 1$.
    $l_h = u_{h-1}$.
    Convert i to $i := u_{h-1} + 1$.
    Calculate $\delta_i = Dist(\bar{\lambda}_1, \bar{\lambda}_i)$.
    $u_h = l_h +$ Number of terms which have same distance as δ_i.
end

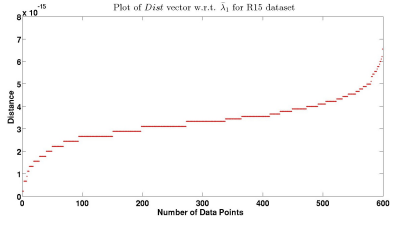
Figure 5.1 depicts the steps involved in determining the intervals for estimating the number of clusters (k) at different levels of hierarchy for the R15 [168]. The R15 dataset contains 600 2-dimensional points. There are 15 clusters in this dataset. In Figure 5.1d, we depict the lower limit of the intervals as $l1, l2, l3, l4, l5$ and $l6$ and the upper limit of the intervals as $u1, u2, u3, u4$ and $u5$ respectively. Using these limits the first 5 intervals that we obtain for the R15 dataset are $1-8$, $8-12$, $12-19$, $19-29$ and $29-40$ respectively. These intervals are obtained using Algorithm 14. From Figure 5.1, we show that first we obtain the Gershgorin discs (Figures 5.1a) which provides us the upper bounds on the eigenvalues. This is followed by the plot of the actual eigenvalues in descending order to show that the actual number of clusters cannot be obtained by directly



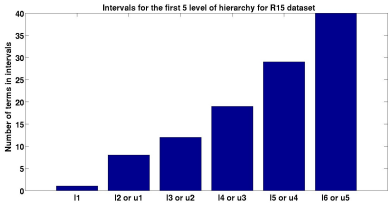
(a) Gershgorin Circles Step



(b) Actual Eigenvalues Plot



(c) Plot of *dist* vector elements



(d) Interval determining Step

Figure 5.1: Steps involved in determining the range for the number of clusters (k) at different levels of hierarchy for R15 Dataset.

using the concept of eigengap (Figures 5.1b) We observe from Figure 5.1b that the number of eigenvalues close to 1 equals 8 and the actual number of clusters in the synthetic R15 dataset is 15. The Gershgorin discs (Figures 5.1a allow us to calculate the *dist* vector (Figures 5.1c). This enables us to determine the

intervals for each level of hierarchy (Figures 5.1d)

In all our experiments, the affinity matrix S was constructed using the RBF-kernel. In order to handle non-linear structures, we use a kernel function to construct the affinity matrix S such that $S_{ij} = K(x_i, x_j) = \phi(x_i)^\intercal \phi(x_j)$. Here $\phi(x_i) \in \mathbb{R}^{n_h}$ and n_h can be infinite dimensional when using the RBF-kernel. One parameter of the RBF-kernel is σ. We use the mean of the multivariate rule-of-thumb proposed in [97] i.e. $\sigma = \text{mean}(\sigma(T) \times N^{-1/(d+4)})$ to estimate σ. Here $\sigma(T)$ is the standard deviation of the dataset, d is the number of dimensions in the dataset and mean is the mean value of all the $\sigma_i, i = 1, \ldots, d$.

5.1.3 Spectral Clustering

Once we obtain the intervals, we want to know the ideal value of k at each level of hierarchy. For this purpose, we provide these intervals as input to the model selection part of the Kernel Spectral Clustering (KSC) [13] method.

Hierarchical Kernel Spectral Clustering (HKSC)

The original KSC formulation [13] uses the Balanced Line Fit (BLF) criterion for model selection i.e. for selection of k and σ. This criterion works well only in case of well separated clusters. So, we use the Balanced Angular Fit (BAF) criterion proposed in [20] for cluster evaluation. It was shown in [20] that the BAF criterion has multiple peaks corresponding to different values of k for a given kernel parameter σ. In our experiments, we use the σ from the rule-of-thumb [97] as explained in Section 5.1.2. BAF is defined as:

$$\begin{aligned} &\text{BAF}(k,\sigma) = \sum_{p=1}^{k} \sum_{valid_{(i,\sigma)} \in Q_p} \frac{1}{k} . \frac{MS(valid_{(i,\sigma)})}{|Q_p|} + \eta \frac{\min_l |Q_l|}{\max_m |Q_m|}, \\ &MS(valid_{(i,\sigma)}) = \max_j \cos(\theta_{j,valid_{(i,\sigma)}}), j = 1, \ldots, k \\ &cos(\theta_{j,valid_{(i,\sigma)}}) = \frac{\mu_j^\intercal e_{valid_{(i,\sigma)}}}{\|\mu_j\| \|e_{valid_{(i,\sigma)}}\|}, j = 1, \ldots, k. \end{aligned} \tag{5.4}$$

where $e_{valid_{(i,\sigma)}}$ represents projection of ith validation point for the given σ, μ_j is mean projection of all validation points in cluster j and Q_p represents the set of validation points belonging to cluster p and $|Q_p|$ is its cardinality. BAF works on the principle of angular similarity. Validation points are allocated to the clusters to which (μ_j) they have the least angular distance. We use a regularizer η to have a convex combination of angular fitting and balance. The BAF criterion varies from [-1, 1] and higher values are better for a given value of k.

So this criterion works on the intervals provided by the proposed approach to detect the ideal number of clusters (k) for each level of hierarchy in the given dataset. We then build the KSC model using that value of k and obtain the cluster memberships for all the points using the out-of-sample extensions property. In constructing the hierarchy we start with smaller values of k before moving to intervals with larger value of k. Thus, the hierarchy of clusters are obtained in a top-down fashion. One advantage of performing the KSC method is that if the actual eigenvalues are too small for a particular interval of hierarchy the KSC method will stop automatically. It suggests that KSC cannot find any more clusters for this interval and future intervals. Thus, we have reached the final level where each individual data point is a cluster.

We then use the linkage criterion introduced in [52] to determine the split of the clusters based on the evolution of the cluster memberships as the hierarchy goes down. The idea is to find the set of points belonging to different clusters at a higher level of hierarchy which are descendants of the same cluster at a lower level of hierarchy. Then, a parent-child relationship is established between these set of clusters. An important point to note is that the splits might not be perfect. For each value of k, the KSC model is run independently and nested partitions are not always guaranteed. A cluster at higher level of hierarchy is considered as child of a cluster at lower level of hierarchy if majority of the points in this child cluster are coming from the parent cluster. A visualization of the hierarchical tree structure generated by HKSC for S1 dataset is depicted in Figure 5.2.

Algorithm 15 explains the steps of hierarchical kernel spectral clustering (HKSC) algorithm that we are using in this work.

Algorithm 15: Hierarchical Clustering Algorithm

Data: Dataset $\mathcal{D} = \{x_1, x_2, \ldots, x_N\}$ and the intervals for k provided by Gershgorin Circle Theorem.
Result: Hierarchical cluster organization for the dataset $\mathcal{D}$

```
Divide the dataset into training, validation and test set as shown in [13].
Use the mean of the multivariate rule-of-thumb [97] as kernel parameter σ.
for Each Interval from Algorithm 14 do
    Use the kernel parameter σ to train a KSC model using the training set.
    Select the k from this interval corresponding to which the BAF [20] criterion is maximum
    and build a KSC model for k clusters.
    Use the out-of-sample extensions property of the clustering model to obtain cluster
    memberships for the test set.
end
Stack all the cluster memberships obtained from the different intervals.
Create a linkage matrix as proposed in [52] by identifying which clusters split starting from the
top of the hierarchy.
```

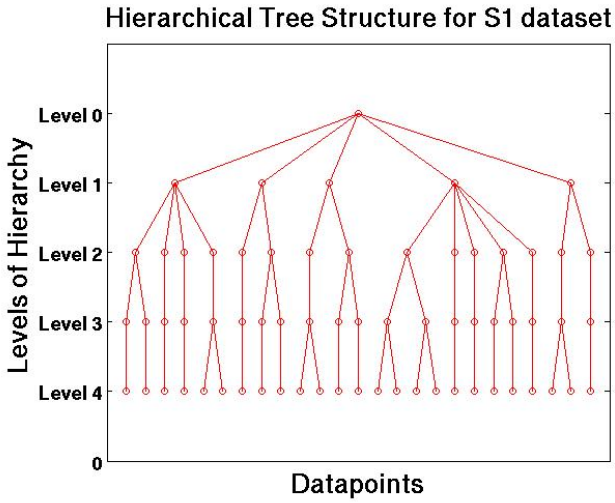


Figure 5.2: Hierarchical tree structure representing the top 5 levels of hierarchy for S1 dataset using HKSC methodology.

5.1.4 Experiments

We conducted experiments on several synthetic and real world datasets. These datasets were obtained from `http://cs.joensuu.fi/sipu/datasets/`. Table 5.1 provides details about these datasets along with the lower (li) and upper (ui) limit for each interval identified by our proposed method.

For HKSC method, we randomly select 30% of the data for training and validation respectively and the entire dataset as test set. We perform 10 randomizations of HKSC and report the mean results in Table 5.2. From Table 5.2, we observe that the HKSC method identifies the ideal number of clusters for most of the datasets including the Dim064, Dim512, Dim1024, Glass, Iris, Pathbased, R15, Spiral, S1 and Wine datasets. In most cases, the Balanced Angular Fit (BAF) values are maximum for the number of clusters identified by HKSC method which are closest to ideal number of clusters. Since the HKSC method requires to construct a kernel matrix ($N_{tr} \times N_{tr}$) in the dual, this method works best when the number of dimensions for a given dataset is large with fewer number of points.

In Figure 5.3 and Figure 5.4, we depict the clusters identified by the HKSC method for the intervals by our proposed approach at different levels of hierarchy. Figure 5.3 shows the results on S1 dataset whereas Figure 5.4 shows the results

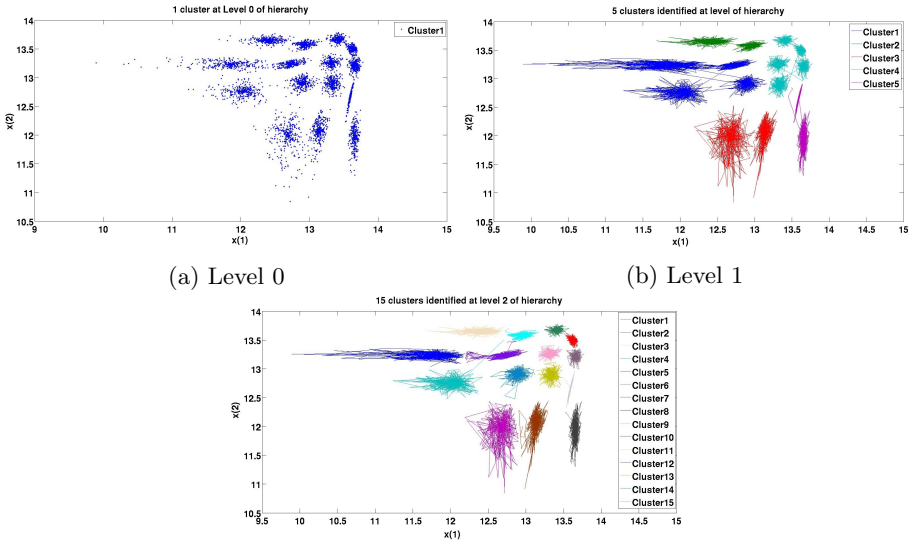
Dataset	Points	Dim	Ideal k	Level 1		Level 2		Level 3		Level 4		Level 5	
				l1	u1	l2	u2	l3	u3	l4	u4	l5	u5
Aggregation	788	2	7	2	5	5	15	15	21	21	26	26	31
D31	3100	2	31	1	13	13	16	16	22	22	27	27	34
DIM032	1024	32	16	1	6	6	14	14	32	32	64	64	152
DIM064	1024	64	16	1	13	13	42	42	169	169	445	445	663
DIM512	1024	512	16	1	6	6	22	22	99	99	300	300	526
DIM1024	1024	1024	16	3	35	35	188	188	426	426	641	641	768
Glass	214	9	7	1	6	6	17	17	32	32	56	56	83
Iris	150	4	3	2	6	6	15	15	35	35	49	49	83
Pathbased	300	2	3	1	6	6	13	13	22	22	37	37	58
R15	600	2	15	1	8	8	12	12	19	19	29	29	40
Sprial	312	2	3	1	17	17	30	30	49	49	85	85	137
S1	5000	2	15	1	6	6	16	16	23	23	27	27	32
Wine	178	13	3	1	5	5	10	10	22	22	34	34	59
Yeast	1484	8	10	1	10	10	15	15	21	21	27	27	38

Table 5.1: Details of various datasets used for experimentation. Ideal k represents the groundtruth number of clusters available for these datasets. However, in case of real-world datasets this ideal k is not always known beforehand.

Dataset	Ideal k	Level 1		Level 2		Level 3		Level 4		Level 5	
		k_1	BAF	k_2	BAF	k_3	BAF	k_4	BAF	k_5	BAF
Aggregation	7	3	0.934	**6**	**0.821**	16	0.695	21	0.5925	26	0.564
D31	31	4	0.829	13	0.755	19	0.837	26	0.655	**29**	**0.679**
DIM032	16	3	0.782	13	0.825	**15**	**0.841**	33	0.32	NA	NA
DIM064	16	13	0.818	**16**	**0.895**	42	0.2625	NA	NA	NA	NA
DIM512	16	3	0.721	**16**	**0.975**	22	0.5225	NA	NA	NA	NA
DIM1024	16	**16**	**0.998**	35	0.325	NA	NA	NA	NA	NA	NA
Glass	7	6	0.658	**7**	**0.677**	18	0.558	NA	NA	NA	NA
Iris	3	**3**	**0.71**	6	0.655	NA	NA	NA	NA	NA	NA
Pathbased	3	**3**	**0.888**	9	0.709	14	0.623	24	0.522	NA	NA
R15	15	7	0.844	9	0.879	**15**	**0.99**	19	0.60	NA	NA
Spiral	3	**3**	**0.818**	21	0.541	32	0.462	NA	NA	NA	NA
S1	15	5	0.842	**15**	**0.876**	16	0.805	23	0.76	NA	NA
Wine	3	**3**	**0.685**	6	0.624	10	0.5025	22	0.406	NA	NA
Yeast	10	3	0.824	**11**	**0.64**	15	0.629	26	0.651	NA	NA

Table 5.2: Hierarchical KSC (HKSC) results on various datasets used for experimentation. 'NA' here means that the eigenvalues are too small and no further clusters are detected i.e. at this level all the points are individual clusters.

for R15 dataset. For the S1 dataset we identified 5 clusters at level 1 and 15 clusters at level 2 of hierarchy. Similarly, for the R15 dataset we identified 7 clusters at level 1, 9 clusters at level 2 and 15 clusters at level 3 of hierarchy. The clusters identified by the HKSC method for each level of hierarchy for both the datasets captures the underlying hierarchical structure. Figure 5.5 highlights the result of HKSC on the intervals provided by our proposed method for 2 real world images.



(a) Level 0

(b) Level 1

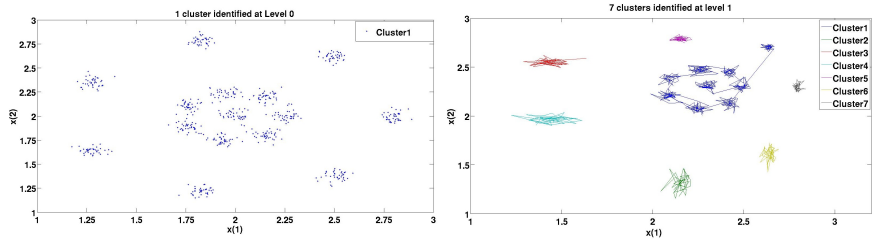
(c) Level 2

Figure 5.3: Clusters identified by HKSC method at Level 1 and Level 2 of hierarchy from the intervals provided in Table 5.1 by proposed method for the S1 dataset.

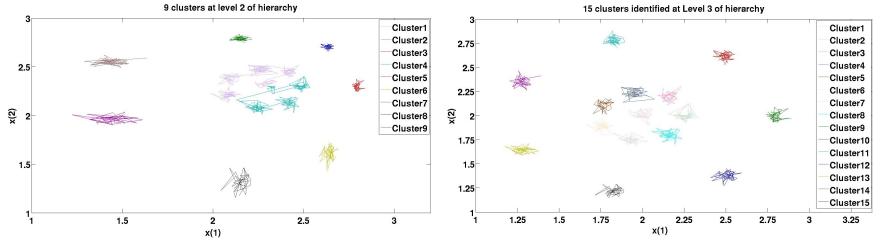
We compare HKSC results with linkage [169] based hierarchical clustering techniques including Single Link (SL), Complete Link (CL) and average Link (AL). The time complexity of proposed approach for identifying the intervals along with HKSC is $O(N^2 + hNN_{tr} + hN_{tr}^3)$, where h is the maximum level of hierarchy and $O(N^2)$ term is for affinity matrix calculation. But since $N_{tr} \ll N$ the overall complexity can be given as $O(N^2)$. The time complexity of SL, CL and AL are $O(N^2)$, $O(N^2 \log(N))$ and $O(N^2 \log(N))$ respectively. Since BAF criterion uses eigen-projections and is catered towards spectral clustering methods, we use another quality metric namely silhouette (SIL) [45] criterion. Higher SIL values correspond to better quality clusters. For all these methods, we compare that level of hierarchy which results in maximum SIL value as shown in Table 5.3.

5.1.5 Conclusion

We proposed a novel method for identifying the ideal range for the number of clusters (k) at different levels of hierarchy in a given dataset. The proposed approach provided these intervals before applying any clustering algorithm.

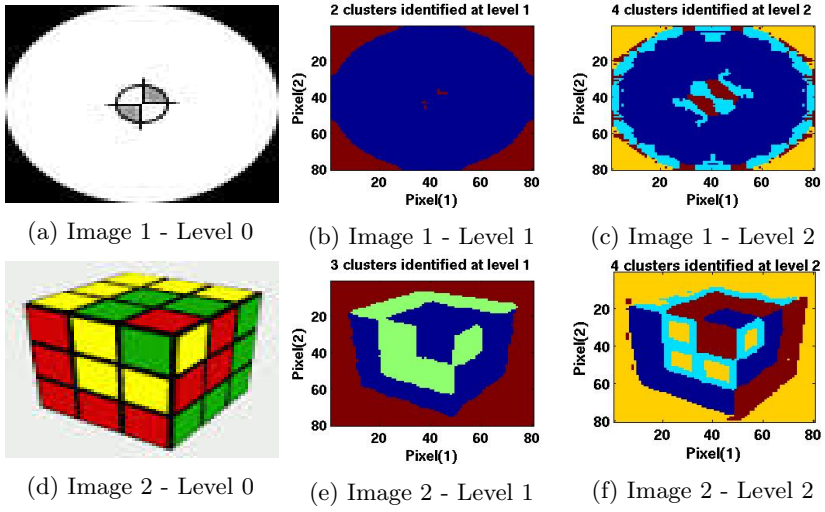


(a) Level 0 (b) Level 1



(c) Level 2 (d) Level 3

Figure 5.4: Clusters identified by HKSC method at Levels 1, 2 and 3 of hierarchy from the intervals provided in Table 5.1 by proposed method for the R15 dataset.



(a) Image 1 - Level 0 (b) Image 1 - Level 1 (c) Image 1 - Level 2

(d) Image 2 - Level 0 (e) Image 2 - Level 1 (f) Image 2 - Level 2

Figure 5.5: Clusters identified by HKSC method at Level 1, Level 2 of hierarchy by the proposed method for the two images.

Dataset	HKSC			SL			CL			AL		
	Best k	**SIL**	**Time(s)**	**Best** k	**SIL**	**Time(s)**	**Best** k	**SIL**	**Time(s)**	**Best** k	**SIL**	**Time(s)**
Aggregation	6	**0.70**	1.29	7	0.55	**1.28**	7	0.67	3.74	7	0.69	3.81
D31	29	**0.71**	22.12	30	0.64	**21.18**	30	0.68	59.56	30	**0.71**	61.12
DIM032	15	**0.86**	**2.91**	14	0.80	3.12	15	0.83	10.15	15	**0.86**	11.22
DIM064	16	**0.78**	**3.55**	16	0.64	4.23	16	0.71	12.12	16	0.74	13.87
DIM512	16	**0.68**	**5.24**	16	0.60	6.53	16	0.64	16.44	16	0.66	18.56
DIM1024	16	**0.62**	**6.72**	16	0.53	8.12	16	0.60	24.21	16	**0.62**	26.45
Glass	7	0.74	0.11	7	0.67	**0.09**	7	0.72	0.21	7	**0.75**	0.22
Iris	3	**0.95**	0.08	3	0.85	**0.05**	3	0.89	0.15	3	0.92	0.15
Pathbased	3	**0.89**	0.33	3	0.84	**0.31**	3	0.87	0.62	3	0.88	0.65
R15	15	0.78	**1.34**	15	0.74	1.35	15	0.77	2.52	15	**0.90**	2.84
Spiral	3	**0.82**	0.38	3	0.76	**0.35**	3	0.78	0.71	3	0.80	0.73
S1	15	**0.88**	**64.12**	15	0.54	65.23	15	0.79	187.9	15	0.81	191.2
Wine	3	0.65	0.10	3	0.62	**0.08**	3	0.64	0.18	3	**0.68**	0.19
Yeast	11	**0.84**	1.01	10	0.64	**0.99**	10	0.76	18.25	10	0.82	18.7

Table 5.3: Comparison of various hierarchical clustering techniques. We compare that level of hierarchy corresponding to which the **SIL** quality metric is maximum. We show the number of clusters for that level of hierarchy as **Best** k. We also compare computational time (in seconds) required by the different clustering techniques. The HKSC method generally results in best quality clusters (**SIL**) along with the AL clustering technique. The HKSC and SL methods are computationally cheaper. The SL technique, though fast, results in the worst quality clusters. The best results are highlighted in bold.

The proposed technique used the Gershgorin circle theorem on a normalized Laplacian matrix to obtain the upper bounds on the eigenvalues without performing the actual eigen-decomposition step. This helps to reduce the computational cost. We then obtained intervals for ideal value of k at each level of hierarchy using these bounds. We can then provide these intervals to any clustering algorithm which uses a normalized Laplacian matrix. We showed that the method works effectively in combination with HKSC for several synthetic and real world datasets.

5.2 Agglomerative Hierarchical Kernel Spectral Data Clustering[2]

Abstract—In this work, we extend the agglomerative hierarchical kernel spectral clustering (AH-KSC [170]) technique from networks to datasets and images. We first estimate the optimal model parameters for KSC [13] using

[2]This section consists of Section 1 and Sections $3-5$ from Mall R., Langone R., Suykens J.A.K., **"Agglomerative hierarchical kernel spectral data clustering"**, IEEE SSCI CIDM, Dec 2014, pp. 9-16.

the balanced angular fitting (BAF) [20] criterion. We then exploit the eigen-projections corresponding to these parameters to automatically identify a set of increasing distance thresholds. These distance thresholds provide the clusters at different levels of hierarchy in the dataset which are merged in an agglomerative fashion as shown in [21]. We showcase the effectiveness of the AH-KSC method on several datasets and real world images. We compare the AH-KSC method with several agglomerative hierarchical clustering techniques and overcome the issues of hierarchical KSC technique proposed in [52].

5.2.1 Introduction

The KSC methodology [13] solves an eigen-decomposition problem in an optimization framework. The dual clustering model is expressed as linear combinations of kernel evaluations with the eigenvectors used as the coefficients. We perform the model selection using the balanced angular fitting (BAF) criterion [20]. The BAF criterion provides the optimal kernel parameter σ for the radial basis kernel function used in case of datasets and σ_{χ^2} kernel function used in case of images. However, in case of networks a linear kernel is sufficient. In this work, we showcase that the BAF criterion can capture multiple peaks corresponding to different levels of hierarchy for an optimal σ. It also identifies the optimal number of clusters k for which the BAF criterion is maximum. This generally corresponds to the highest level of hierarchy.

In order to estimate these peaks and obtain a hierarchical organization for the given dataset we exploit the structure of the eigen-projections for the validation set obtained from the dual clustering model using the optimal kernel parameter (σ/σ_{χ^2}) and the out-of-sample extension property. The concept of exploiting the structure of the eigen-projections was first introduced in [126] for automatically determining the number of communities k in a network. It was then extended to identify multilevel hierarchical organizations in large scale networks in [21]. However, in the case of networks the normalized linear kernel is used [21] and there is no kernel parameter (σ). We used the same concept to obtain a hierarchical structure in a bottom-up fashion after obtaining the optimal σ from the BAF model selection criterion.

A hierarchical kernel spectral clustering technique was proposed in [52]. There the authors used multiple scales of the kernel parameter σ to obtain a KSC model with different k at each level of hierarchy. Then, for each KSC model they obtain the cluster memberships for the full dataset using the out-of-sample extension property. A special linkage criterion was used to determine which clusters at a given level merged based on cluster memberships of the full dataset. One of the issues with this method is that during the merging, there might be

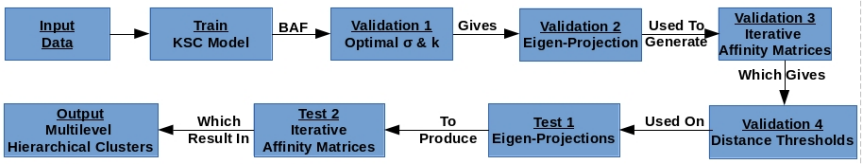


Figure 5.6: Steps of AH-KSC method as described in [21, 170] with addition of the step where the optimal σ and k are estimated.

some points of the merging clusters which go to a different clusters as mentioned in [52]. These points are forced to join the merging cluster of the majority. Thus, the hierarchical connections between the layers are more ad-hoc. We overcome this problem in the AH-KSC method as it provides a natural agglomerative bottom-up hierarchical organization.

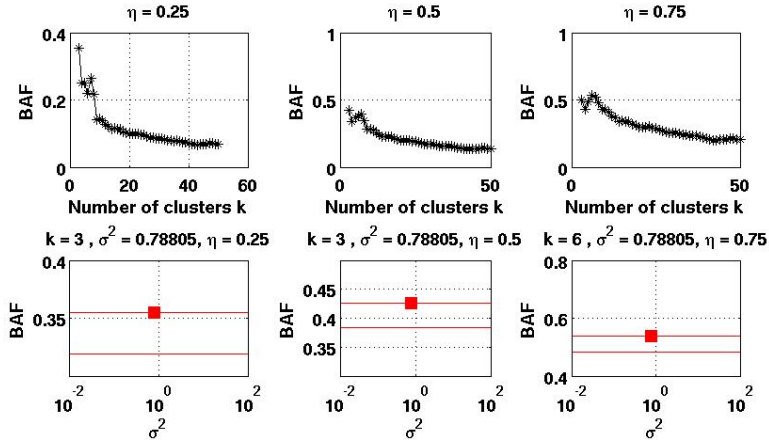
Figure 5.6 provides an overview of the steps involved in AH-KSC method and Figure 5.7 depicts the result of AH-KSC approach on a synthetic dataset and Figure 5.8 showcases the same for a real world image (Id:119082).

5.2.2 Agglomerative Hierarchical KSC approach

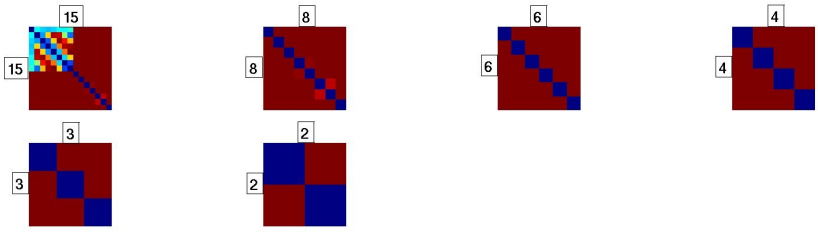
In [21], the authors show that the dual predictive KSC model can be used to obtain the latent variable matrix for the validation set as $P_{valid} = [e_1; \ldots; e_{N_{valid}}]$. The P_{valid} matrix is a $N_{valid} \times (maxk\text{-}1)$ matrix. The test projection set is denoted by P_{test}. In [126], the authors created an affinity matrix S_{valid} using P_{valid} as:

$$S_{valid}(i,j) = CosDist(e_i, e_j) = 1 - \cos(e_i, e_j) = 1 - \frac{e_i^\intercal e_j}{\|e_i\|\|e_j\|}, \tag{5.5}$$

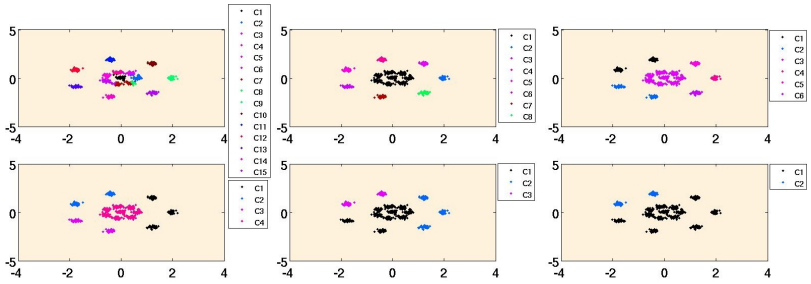
where $CosDist(\cdot,\cdot)$ function calculates the cosine distance between 2 vectors and takes values between $[0, 2]$. Data points which belong to the same cluster have $CosDist(e_i, e_j)$ closer to 0, $\forall i, j$ in the same cluster. It was shown in [126] that a rotation of the S_{valid} matrix has a block diagonal structure where the number of block diagonals is equal to number of clusters (k) in the dataset. In [21], the authors showed that the affinity matrix generated at one level of hierarchy depends on the affinity matrix of the previous level and a kernel



(a) Selection of optimal σ using BAF criterion and illustation of the multiple peaks in the BAF curve for different values of k.

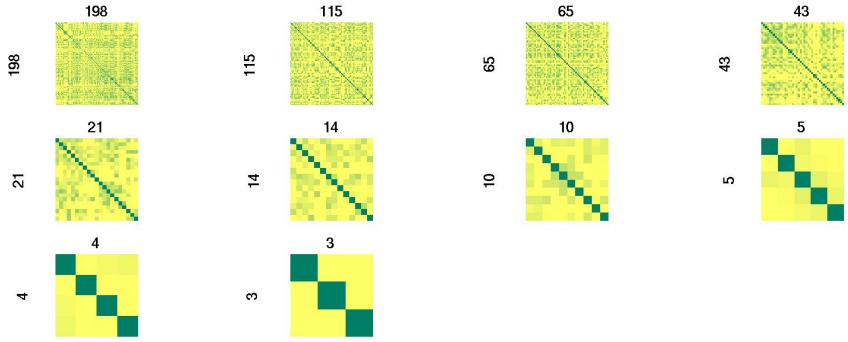


(b) Affinity matrices created at different levels of hierarchy in left to right order. The x, y-axis represent the size of the affinity matrix and the number of clusters k at each level of hierarchy.

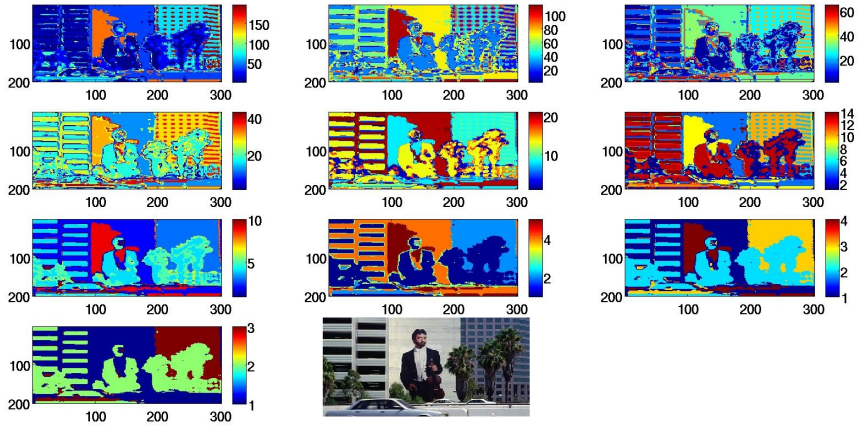


(c) The cluster memberships at different levels of hierarchy

Figure 5.7: Result of AH-KSC method on a synthetic dataset comprising 15 Gaussians.



(a) Affinity matrices created at different levels of hierarchy from left to right. The x, y-axis represent the size of the affinity matrix and number of clusters k at each level of hierarchy.



(b) The image segments at different levels of hierarchy along with the original image.

Figure 5.8: Result of AH-KSC method on a real world image dataset (Id:119082).

function. This can be depicted in the form of an equation as:

$$S^{(h)}(i,j) = \sum_{m \in C_i^{(h-1)}} \sum_{l \in C_j^{(h-1)}} \kappa(i,j) S^{(h-1)}(m,l)$$
$$\kappa(i,j) = \frac{1}{|C_i^{(h-1)}| \times |C_j^{(h-1)}|}, \tag{5.6}$$

where $S^{(h)}(i,j)$ represents the $(i,j)^{th}$ element of the affinity matrix at level h of hierarchy. $C_i^{(h-1)}$ represents the set of points in the ith cluster at level h-1 of hierarchy, $|\cdot|$ represents cardinality and $\kappa(i,j)$ is the kernel function which acts as a normalization constant.

$S^{(0)}_{valid}$ is obtained by calculating the $CosDist(\cdot,\cdot)$ between the elements of P_{valid} matrix as shown in equation (5.5). It was shown in [21] and [126] that $t^{(0)} = 0.15$ is most suitable threshold for the lowest level of hierarchy. A detailed description of the AH-KSC approach can be obtained from [21, 170]. Algorithm 16 summarizes the AH-KSC approach.

Algorithm 16: AH-KSC Method for Data Clustering

Data: Given a dataset $\mathcal{D}$ of data points or images.
Result: Agglomerative Hierarchical KSC of the dataset.

1 Divide data into train,validation and test set.
2 Use $\mathcal{D} = \{x_i\}_{i=1}^{N_{tr}}$, $x_i \in \mathbb{R}^d$ as the training set.
3 Perform KSC on $\mathcal{D}$ to obtain the predictive model.
4 **Use BAF criterion to obtain optimal kernel parameter σ which results in multiple peaks in the BAF curve for different values of k.** `/* Additional Step required for datasets & images.` `*/`
5 Obtain $P_{valid} = [e_1; \ldots; e_{N_{valid}}]$.
6 $S^{(0)}_{valid}(i,j) = CosDist(e_i, e_j) = 1 - \frac{e_i^\intercal e_j}{\|e_i\|\|e_j\|}$, $\forall e_i, e_j \in P_{valid}$ using equation (5.5).
7 Begin validation stage with: $h = 0$, $t^{(0)} = 0.15$.
8 $[C^{(0)}, k] = GreedyMaxOrder(S^{(0)}_{valid}, t^{(0)})$. Add $t^{(0)}$ to the set $\mathcal{T}$ and $C^{(0)}$ to the set $\mathcal{C}$.
9 **while** $k > 1$ **do**
10 $h := h + 1$.
11 Create $S^{(h)}_{valid}$ from $S^{(h-1)}_{valid}$ and $C^{(h-1)}$ using the concept in equation (5.6).
12 Calculate $t^{(h)} = \text{mean}(\min_j(A^{(h)}_{valid}(i,j))), i \neq j$.
13 $[C^{(h)}, k] = GreedyMaxOrder(S^{(h)}_{valid}, t^{(h)})$.
14 Add $t^{(h)}$ to the set $\mathcal{T}$ and $C^{(h)}$ to the set $\mathcal{C}$.
15 **end**
`/* Iterative procedure to get` $\mathcal{T}$. `*/`
16 Obtain P_{test} like P_{valid} and begin with: $h = 1$, $t^{(1)} \in \mathcal{T}$.
17 $[S^{(2)}_{test}, C^{(1)}, k] = GreedyFirstOrder(P_{test}, t^{(1)})$. Add $C^{(1)}$ to the set $\mathcal{C}$.
18 **foreach** $t^{(h)} \in \mathcal{T}$, $h > 1$ **do**
19 $[C^{(h)}, k] = GreedyMaxOrder(S^{(h)}_{test}, t^{(h)})$.
20 Add $C^{(h)}$ to the set $\mathcal{C}$.
21 Create $S^{(h+1)}_{test}$ from $S^{(h)}_{test}$ & $C^{(h)}$ using equation (5.6).
22 **end**
23 Obtain the set $\mathcal{C}$ for test set and propagate cluster memberships from 1st to topmost level of hierarchy.

5.2.3 Experiments

We conducted experiments on several synthetic datasets available at `http://cs.joensuu.fi/sipu/datasets/` and some real world images from `http:`

//www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/. For the synthetic datasets the ground truth or cluster memberships corresponding to flat clustering are known beforehand. Hence, we can use external cluster quality measure like Adjusted Rand Index (ARI) [45] for evaluating the quality of the clusters. However, for these datasets, only flat clustering or cluster membership for one level of hierarchy is known.

We use 30% and 40% of the data as training and validation set. We use the entire dataset as test set. To evaluate the quality of the clusters obtained at multiple levels of hierarchy we utilize internal quality metrics like silhouette (Sil) and Davies-Bouldin index (DB) [45]. Silhouette values are normalized between [0,1] and higher values correspond to better quality clusters while DB has opposite behavior. Lower values of DB close to 0 represent better quality clusters. We define an F-score as ratio between Sil and DB and observe the level of hierarchy for which the F-score is maximum. Table 5.4 provides a brief description of the datasets used in the experiments.

Datasets			
Name	Total Points (N)	Dimensions	Optimal k
Aggregation	788	2	7
Dim2	1351	2	9
R15	600	2	15
S1	5000	2	15
Images			
Number	Pixels		Optimal k
Id:10001	6,400		4
Id:10002	6,400		5
Id:100007	154,401		4
Id:119082	154,401		5

Table 5.4: Datasets used in the experiments. For synthetic datasets the optimal k is equivalent to ground truth number of clusters. For images optimal k is selected based on the maximum F-score value for SL, CL, ML, WL and AL agglomerative clustering techniques. This optimal k for images is usually not the same as that for AH-KSC method.

Experiments on Datasets

We compare AH-KSC method with traditional agglomerative hierarchical clustering techniques which merge clusters based on a linkage measure that specifies how dissimilar 2 clusters are. We compare against linkage techniques [169] including single (SL), complete (CL), median (ML) and weighted linkage (WL). We also compare against average link (AL) technique for datasets. One of the issues with these techniques is that they calculate pairwise similarities and at the lowest level of hierarchy the complexity becomes $O(N^2)$ which is

computationally expensive. The AH-KSC method overcomes this problem by obtaining a model on a subset of the data. It then estimates the optimal model parameters (σ) using the BAF criterion following which it performs the agglomerative hierarchical clustering. It uses the *GreedyFirstOrder* approach [21] which overcomes the issue of pairwise similarity calculation.

For all our experiments we perform 10 randomizations and report the mean results for various quality metrics. In Table 5.5 we provide a comparison of the AH-KSC method with other hierarchical clustering techniques w.r.t. the various quality measures like F-score, DB, Sil and ARI. For each method we present the best value for a given quality measure which corresponds to a clustering for that level of hierarchy which has the maximum F-score. From Table 5.5, we observe that the AH-KSC method gives the best results on 3 datasets for F-score and DB measure and 2 datasets for Sil internal quality measure.

Dataset	AH-KSC	SL	AL	CL	ML	WL
	F-score					
Aggregation	**1.34**	0.84	1.08	1.01	0.98	0.92
Dim2	2.27	**7.37**	5.30	4.54	5.40	5.40
R15	**4.75**	0.92	3.32	3.28	3.18	3.15
S1	**2.96**	0.0	1.81	1.55	1.58	1.76
	DB					
Aggregation	**0.466**	0.65	0.64	0.66	0.68	70
Dim2	0.37	**0.12**	0.17	0.20	0.17	0.17
R15	**0.17**	0.81	0.27	0.27	0.28	0.29
S1	**0.29**	1.7	0.45	0.51	0.50	0.46
	Sil					
Aggregation	**0.69**	0.55	**0.69**	0.67	0.66	0.65
Dim2	0.85	**0.90**	**0.90**	**0.90**	**0.90**	**0.90**
R15	0.78	0.74	**0.90**	0.90	0.90	0.90
S1	**0.88**	0.54	0.81	0.79	0.79	0.81
	ARI					
Aggregation	**0.94**	0.81	0.93	0.78	0.67	0.71
R15	0.87	0.54	**0.99**	0.98	0.98	0.99

Table 5.5: Comparison of AH-KSC method with single, average, complete, median and weight linkage techniques [169] based on various quality measures. The bold represent the best approach among the methods for a given dataset.

Figure 5.9 shows the tuning procedure for AH-KSC and 5.10 compares AH-KSC method with other clustering techniques for Aggregation dataset. We plot the tree based hierarchical structure for AH-KSC method in Figure 5.11a. Figure 5.11 also showcases the dendogram plots for SL, AL, CL, ML and WL hierarchical clustering techniques. The dendogram plots (5.11b,5.11c,5.11d,5.11e,5.11f) combine only 2 clusters at a given level of hierarchy. The tree based plot doesn't suffer from this drawback and can combine multiple set of clusters at a given level of hierarchy.

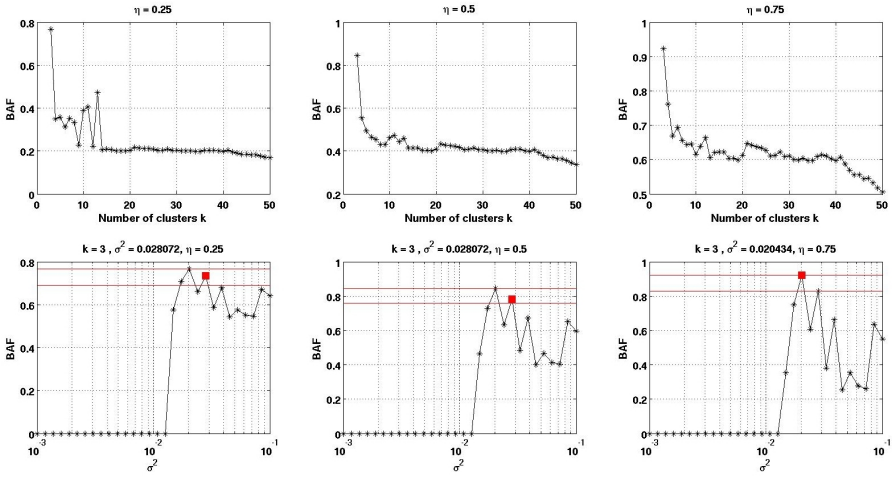


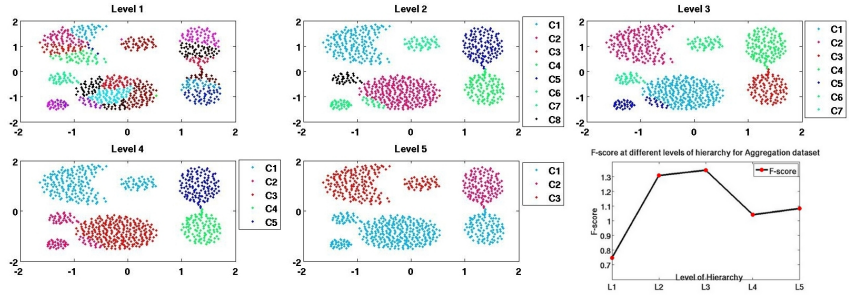
Figure 5.9: Optimal σ selection using BAF criterion. η determines the weight of the balance part. The σ which results in highest BAF value for maximum number of times for all η values is selected. In this example, optimal $\sigma = 0.028$ for $k = 3$. $k = 3$ should occur at one level of hierarchy for the AH-KSC method.

Experiments on Images

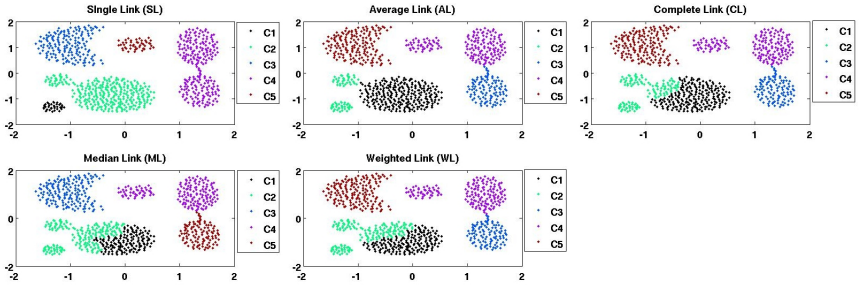
We illustrate segmentation on an image (Id:100007) such that each pixel is transformed into a histogram and the χ^2 distance is used in the RBF kernel with bandwidth σ_{χ^2}.

Figure 5.12 represents the image segmentation results for AH-KSC method at various level of hierarchy and the results for SL, CL, ML and WL agglomerative clustering techniques for the layer corresponding to which the F-score is highest. For the AH-KSC method we scale the image to 200×250 i.e. $50,000$ pixels. For the other hierarchical clustering techniques the image is scaled down to 100×150 i.e. $15,000$ pixels. This is because otherwise they become memory extensive. The Average Link technique is computationally very expensive and therefore not used during evaluation. From Figure 5.12a we observe that hierarchical segments obtained by AH-KSC method seem much more natural than those obtained by other clustering techniques as shown in Figure 5.12b. This is also verified by the highest Sil value for AH-KSC method in comparison to other techniques depicted in Table 5.6.

From Table 5.6, we can observe that the AH-KSC method provides the best F-score for 2 images, best DB value for 1 image and best Sil value for 3 images.



(a) 5 levels of hierarchy produced by AH-KSC method and best F-score at $k = 7$ for Level 3.



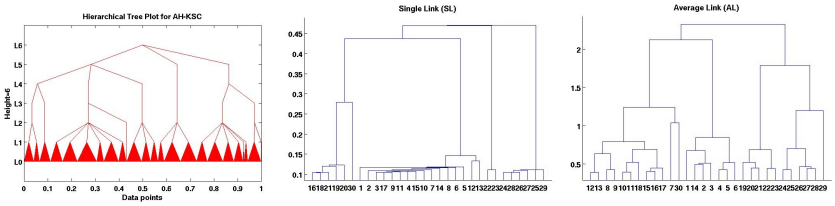
(b) Best results for other hierarchical techniques where F-score is maximum for $k = 5$.

Figure 5.10: Results of different hierarchical clustering technique on Aggregation dataset.

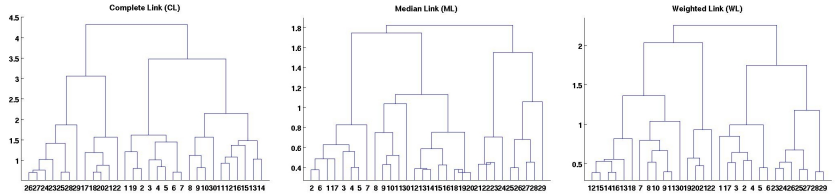
We observe that for image Id:100007 AH-KSC approach has the best Sil value but the weighted link technique has the lowest DB value which is quite small and results in better F-score for WL method in comparison to AH-KSC approach.

5.2.4 Conclusion

We extended the AH-KSC method [21, 170] from networks to data clustering. We obtained the optimal kernel parameter σ during the model selection stage using the BAF criterion. We used this optimal σ to generate the KSC clustering model. We then generated the eigen-projections using the out-of-sample extension property. These were further utilized to estimate a set of distance thresholds ($\mathcal{T}$). We iteratively built affinity matrices from these eigen-projections and used these distance thresholds to obtain an agglomerative hierarchical organization in a bottom-up fashion. We compared the AH-KSC method with other agglomerative



(a) Tree plot for AH-KSC method (b) Dendogram for Single Link (c) Dendogram for Average Link



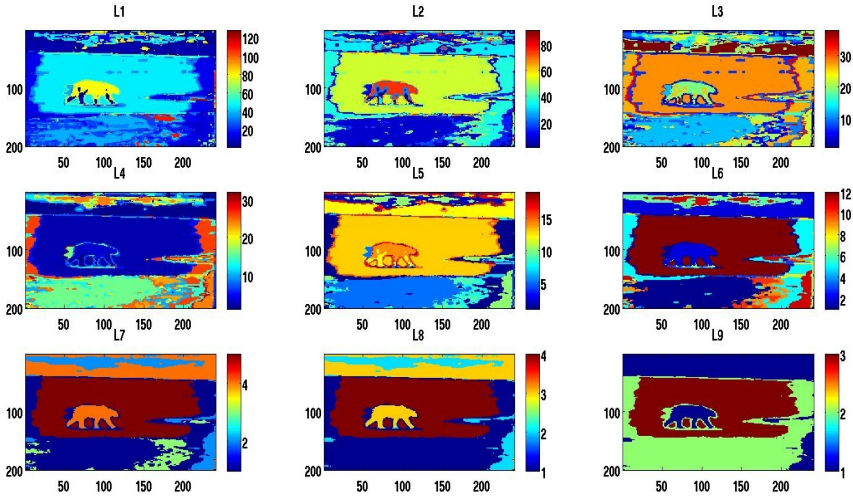
(d) Dendogram for Complete Link (e) Dendogram for Median Link (f) Dendogram for Weighted Link

Figure 5.11: Plots of different hierarchical structures for Aggregation dataset. AH-KSC method allows multiple set of clusters to combine at a given level of hierarchcy which is not permitted by the dendogram structure.

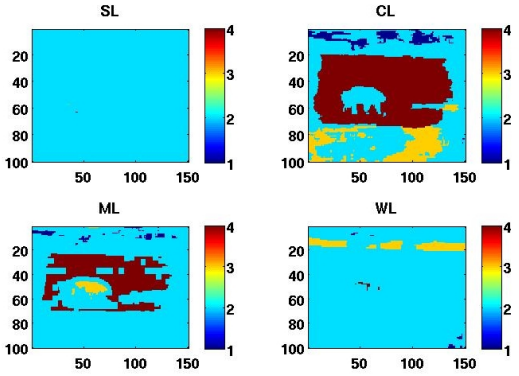
Image	AH-KSC	SL	CL	ML	WL
	F-score				
Id:10001	**2.08**	0.394	1.57	1.35	1.35
Id:10002	1.02	0.70	1.37	1.2	**1.40**
Id:100007	0.73	0.38	0.41	0.68	**1.00**
Id:119082	**0.59**	0.57	0.44	0.56	0.59
	DB				
Id:10001	**0.41**	0.68	0.55	0.62	0.62
Id:10002	0.73	0.80	0.57	0.63	**0.55**
Id:100007	0.89	1.02	1.41	0.75	**0.48**
Id:119082	0.97	**0.80**	1.1	0.88	0.81
	Sil				
Id:10001	**0.85**	0.27	0.85	0.84	0.84
Id:10002	0.75	0.56	**0.77**	0.76	0.77
Id:100007	**0.65**	0.39	0.58	0.51	0.48
Id:119082	**0.57**	0.45	0.48	0.49	0.49

Table 5.6: Comparison of AH-KSC method with other Agglomerative Hierarchical techniques based on various quality measures for images. The bold represent the best approach among the methods for a given image.

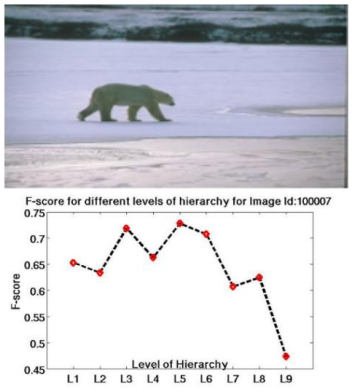
hierarchical clustering techniques and showed its effectiveness on several datasets and images.



(a) Hierarchical Segmentation by AH-KSC method from levels L1 to L9.



(b) Best results for other hierarchical techniques where F-score is maximum for $k = 4$



(c) Original Id:100007 Image & Best F-score at L5 when $k = 19$ for AH-KSC method.

Figure 5.12: Hierarchical segmentation results by different methods on Id:100007 image.

5.3 Sparse Kernel Spectral Clustering[3]

Abstract—Kernel spectral clustering (KSC) solves a weighted kernel principal component analysis problem in a primal-dual optimization framework. It has a powerful out-of-sample extension property leading to good clustering generalization w.r.t. the unseen data points. The out-of-sample extension property allows to build a sparse model on a small training set and introduces the first level of sparsity. The clustering dual model is expressed in terms of non-sparse kernel expansions where every point in the training set contributes. The goal is to find reduced set of training points which can best approximate the original solution. In this work, a second level of sparsity is introduced in order to reduce the time complexity of the computationally expensive out-of-sample extension. In this work, we investigate various penalty based reduced set techniques including the Group Lasso, L_0, $L_1 + L_0$ penalization and compare the amount of sparsity gained w.r.t. a previous L_1 penalization technique. We observe that the optimal results in terms of sparsity corresponds to the Group Lasso penalization technique in majority of the cases. We showcase the effectiveness of the proposed approaches on several real world datasets and an image segmentation dataset.

5.3.1 Introduction

A Kernel Spectral Clustering (KSC) algorithm based on weighted kernel PCA formulation was proposed in [13]. The method was based on a model built in a primal-dual optimization framework. The model had a powerful out-of-sample extension property which allows to infer cluster affiliation for unseen data. The KSC methodology has been extensively applied for task of data clustering [13], [50], [51] and community detection [20], [126], [21] in large scale networks.

The data points are projected to the eigenspace and the projections are expressed in terms of non-sparse kernel expansions. In [50], a method to sparsify the clustering model was proposed by exploiting the line structure of the projections when the clusters are well formed and well separated. However, the method fails when the clusters are overlapping and for real world datasets where the projections in the eigenspace do not follow a line structure as mentioned in [51]. In [51], the authors used an $L_2 + L_1$ penalization to produce a reduced set to approximate the original solution vector. Although the authors propose it as an $L_2 + L_1$ penalization technique, the actual penalty on the weight vectors

[3]This section consists of Section 1 and Section 3 – 5 from Mall R., Mehrkanoon S., Langone R., Suykens J.A.K., **"Optimal reduced sets for sparse kernel spectral clustering"**, IEEE IJCNN, July 2014, pp. 2436-2443.

is L_1 penalty and the loss function is squared loss function and hence the name. Therefore, in this work, we refer to the previous proposed approach as L_1 penalization technique. It is well known that the L_1 regularization introduces sparsity as shown in [171]. However, the resulting reduced set is neither the sparsest nor the most optimal w.r.t. the quality of clustering for the entire dataset. In this work, we propose alternative penalization techniques like Group Lasso [61] and [62], L_0 and $L_1 + L_0$ penalizations. The Group Lasso penalty is ideal for clusters as it results in groups of relevant data points. The L_0 regularization calculates the number of non-zero terms in the vector. The L_0-norm results in a non-convex and NP-hard optimization problem. We modify the convex relaxation of L_0-norm based iterative sparsification procedure introduced in [80] for classification. We apply it to obtain the optimal reduced sets for sparse kernel spectral clustering.

The main advantage of these sparse reductions is that it results in much simpler and faster predictive models. It allows to reduce the time complexity for the computationally expensive out-of-sample extensions and also reduces the memory requirements for building the test kernel matrix.

5.3.2 Sparse reductions to KSC model

Related Work

In classical spectral clustering one needs to store the $N \times N$ matrix where N is the total number of points in the dataset. One then has to perform an eigen-decomposition of this matrix. The time complexity of this eigen-decomposition is $O(N^3)$. In the case of KSC we can build the training model using a training set ($N_{tr} \ll N$) and use the out-of-sample extension property to predict the cluster affiliation for unseen data. This leads to the first level of sparsity. However, the projections of the data points in the eigenspace are expressed in terms of non-sparse kernel expansions as reflected in:

$$\hat{e}^{(l)}(x) = \sum_{i=1}^{N_{tr}} \alpha_i^{(l)} K(x, x_i) + b_l. \tag{5.7}$$

This non-sparsity is a result of the KKT condition: $w^{(l)} = \sum_{i=1}^{N_{tr}} \alpha_i^{(l)} \phi(x_i)$. Here $w^{(l)}$ represents the optimal representation of the primal weight vectors and comprises of linear combination of the mapped training data points in the feature space. When using a universal kernel like the RBF kernel the feature space comprises infinite dimensions. Thus, we first create an explicit feature map using the Nyström approximation as in [86] and [92]. This explicit feature

map is created using the training points X_{tr} and the feature mapping becomes: $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^{N_{tr}}$.

The objective is to find a reduced set of training points $\mathcal{RS} = \{\tilde{x}_i\}_{i=1}^{R}$ such that it approximates $w^{(l)}$ by a new weight vector $\tilde{w}^{(l)} = \sum_{i=1}^{R} \beta_i^{(l)} \phi(\tilde{x}_i)$ while minimizing the reconstruction error $||w^{(l)} - \tilde{w}^{(l)}||_2^2$ where $\tilde{x}_i$ is the ith point in the reduced set $\mathcal{RS}$ whose cardinality is R. In [51], it was shown by the authors that if the reduced set $\mathcal{RS}$ is known then the $\beta^{(l)}$ co-efficients can be obtained by solving the linear system:

$$\Omega^{\psi\psi}\beta^{(l)} = \Omega^{\psi\phi}\alpha^{(l)}, \tag{5.8}$$

where $\Omega_{mn}^{\psi\psi} = K(\tilde{x}_m, \tilde{x}_n)$, $\Omega_{mi}^{\psi\phi} = K(\tilde{x}_m, x_i)$, $m, n = 1, \ldots, R, i = 1, \ldots, N_{tr}$ and $l = 1, \ldots, k-1$.

In the past literature including the works in [50] and [51], it was shown that this reduced set can be built by selecting points whose projections in the eigenspace occupy certain positions or by using an L_1 penalization. The first method works only when the clusters are well formed and well separated and cannot be generalized to real world datasets. The second method using L_1 penalization cannot introduce significant sparsity. In this work, we investigate other penalization techniques including the Group Lasso [61] and [62], L_0 and $L_1 + L_0$ penalizations.

Group Lasso Penalization

The Group Lasso was first proposed for regression in [61] where it solves the convex optimization problem:

$$\min_{\beta \in \mathbb{R}^p} \quad \|y - \sum_{l=1}^{L} X_l \beta_l\|_2^2 + \lambda \sum_{l=1}^{L} \sqrt{\rho_l} \|\beta_l\|_2,$$

where the $\sqrt{\rho_l}$ accounts for the varying group sizes, $\|\,.\,\|_2$ is the Euclidean norm. This procedure acts like Lasso [171] at a group level: depending on λ, an entire group of predictors may drop out of the model. We now utilize this to obtain the formulation for our optimization problem as:

$$\min_{\beta \in \mathbb{R}^{N_{tr} \times (k-1)}} \quad \|\Phi^\intercal \alpha - \Phi^\intercal \beta\|_2^2 + \lambda \sum_{l=1}^{N_{tr}} \sqrt{\rho_l} \|\beta_l\|_2, \tag{5.9}$$

where $\Phi = [\phi(x_1), \ldots, \phi(x_{N_{tr}})]$, $\alpha = [\alpha^{(1)}, \ldots, \alpha^{(k-1)}]$, $\alpha \in \mathbb{R}^{N_{tr} \times (k-1)}$ and $\beta = [\beta_1, \ldots, \beta_{N_{tr}}]$, $\beta \in \mathbb{R}^{N_{tr} \times (k-1)}$. Here $\alpha^{(i)} \in \mathbb{R}^{N_{tr}}$ while $\beta_j \in \mathbb{R}^{k-1}$ and we

set $\sqrt{\rho_l}$ as the fraction of training points belonging to the cluster to which the lth training point belongs. By varying the value of λ we control the amount of sparsity introduced in the model as it acts as a regularization parameter. In [62], the authors show that if the initial solutions are $\hat{\beta}_1, \hat{\beta}_2, \ldots, \hat{\beta}_{N_{tr}}$ then if $\|X_l^\intercal(y - \sum_{i \neq l} X_i \hat{\beta}_i)\| < \lambda$, then $\hat{\beta}_l$ is zero otherwise it satisfies: $\hat{\beta}_l = (X_l^\intercal X_l + \lambda / \|\hat{\beta}_l\|)^{-1} X_l^\intercal r_l$ where $r_l = y - \sum_{i \neq l} X_i \hat{\beta}_i$.

Analogous to this, the solution to the group lasso penalization for our problem can be defined as: $\|\phi(x_l)^\intercal(\Phi^\intercal \alpha - \sum_{i \neq l} \phi(x_i)\hat{\beta}_i)\| < \lambda$ then $\hat{\beta}_l$ is zero otherwise it satisfies: $\hat{\beta}_l = (\Phi^\intercal \Phi + \lambda / \|\hat{\beta}_l\|)^{-1} \phi(x_l) r_l$ where $r_l = \Phi^\intercal \alpha - \sum_{i \neq l} \phi(x_i)\hat{\beta}_i$. The Group Lasso penalization technique can be solved by a blockwise co-ordinate descent procedure as shown in [61]. The time complexity of the approach is $O(\text{maxiter} * k^2 N_{tr}^2)$ where maxiter is the maximum number of iterations specified for the co-ordinate descent procedure and k is the number of clusters obtained via KSC. From our experiments we observed that on an average 10 iterations suffice for convergence.

An important point to remember here is that $\hat{\beta}_l \in \mathbb{R}^{k-1}$ and is a vector. When this $\hat{\beta}_l$ is zero it means that it is equivalent to zero vector or the corresponding l^{th} training point is not part of the reduced set $\mathcal{RS}$. In our experiments, we set the initial value of β as $\hat{\beta}_{ij} = \alpha_{ij} + \mathcal{N}(0,1)$ where $\mathcal{N}(0,1)$ represents Gaussian noise with mean 0 and standard deviation 1.

L_0 Penalization

We modify the iterative sparsification procedure for classification as shown in [80] and use it for obtaining the reduced set. The optimization problem ($\mathcal{J}$) which is solved iteratively is formulated as:

$$\begin{aligned} \min_{\beta \in \mathbb{R}^{N_{tr} \times (k-1)}} \quad & \|\Phi^\intercal \alpha - \Phi^\intercal \beta\|_2^2 + \rho \sum_{i=1}^{N_{tr}} \epsilon_i + \|\Lambda.\beta\|_2^2 \\ \text{such that} \quad & \|\beta_i\|_2^2 \leq \epsilon_i, i = 1, \ldots, N_{tr} \\ & \epsilon_i \geq 0, \end{aligned} \tag{5.10}$$

where Λ is matrix of the same size as the β matrix i.e. $\Lambda \in \mathbb{R}^{N_{tr} \times (k-1)}$. The term $\|\Lambda.\beta\|_2^2$ along with the constraint $\|\beta_i\|_2^2 \leq \epsilon_i$ corresponds to the L_0-norm penalty on β matrix. Λ matrix is initially defined as a matrix of ones so that it gives equal chance to each element of β matrix to reduce to zero. The constraints on the optimization problem forces each element of $\beta_i \in \mathbb{R}^{(k-1)}$ to reduce to zero. This helps to overcome the problem of sparsity per component which is

explained in [51]. The ρ variable is a regularizer which controls the amount of sparsity that is introduced by solving this optimization problem.

The optimization problem stated in (5.9) is a convex Quadratically Constrained Quadratic Programming (QCQP) problem. Its computational complexity is $O(k^3N_{tr}^3)$ and we solve it iteratively using the CVX software:[172]. We obtain a β matrix as a solution for each iteration such that $\beta_{ij}^{t+1} = \arg\min_\beta \mathcal{J}(\Lambda_{ij}^t)$. For each iteration, the Λ matrix is re-weighted as: $\Lambda_{ij}^t = \frac{1}{\beta_{ij}^t}$, $\forall\, i = 1, \ldots, N_{tr}, j = 1, \ldots, k-1$. It was shown in [80] that this iterative procedure results in a convex approximation to the L_0-norm. But as the L_0-norm is a non-convex problem it results in a local minimum. We stop this iterative procedure when the rate of change of the β matrix is below a threshold such that $\|\beta^{t+1} - \beta^t\|_2^2/N_{tr} < 10^{-4}$. We then select those indices i for which $\|\beta_i^{t+1}\|_2^2 > 10^{-6}$ and put the corresponding training points in the reduced set $\mathcal{RS}$. In our experiments we observe that the number of iterations required to reach this convergence is usually less than 20.

$L_1 + L_0$ Penalization

The $L_1 + L_0$ penalization formulation is quite similar to the formulation of L_1 penalization as defined in [51]. We add an additional regularization matrix Λ on the β matrix and the problem formulation becomes:

$$\begin{aligned} \min_{\beta \in \mathbb{R}^{N_{tr} \times (k-1)}} \quad & \|\Phi^\intercal \alpha - \Phi^\intercal \beta\|_2^2 + \rho \sum_{i=1}^{N_{tr}} \epsilon_i + \|\Lambda.\beta\|_2^2 \\ \text{such that} \quad & |\beta_i| \leq \epsilon_i, i = 1, \ldots, N_{tr} \\ & \epsilon_i \geq 0, \end{aligned} \tag{5.11}$$

The difference between (5.10) and (5.11) is the set of constraints for both the optimization problems. In (5.11) the constraint $|\beta_i| \leq \epsilon_i$ corresponds to the L_1 penalization.

This problem formulation results in a convex Quadratic Programming (QP) problem due to linear constraints. Its computational complexity is $O(k^3N_{tr}^3)$. It is also solved iteratively using the CVX software. We initialize Λ matrix as ones and after each iteration we modify each element of Λ^t matrix such that $\Lambda_{ij}^t = \frac{1}{\beta_{ij}^t}$, $\forall\, i = 1, \ldots, N_{tr}, j = 1, \ldots, k-1$. We show in the experimental results that this penalization often results similarly to the L_1 penalization outcomes. This suggests that the L_1 penalization is driving the amount of sparsity in this penalization to obtain the reduced set $\mathcal{RS}$.

Choice of Tuning parameter

The choice of the right tuning parameter is essential to obtain optimal reduced set. The tuning parameter λ influences the amount of sparsity in the model for the Group Lasso penalization technique while in case of other penalization techniques this is handled by the tuning parameter ρ. Sparsity is defined as $1 - \frac{|\mathcal{RS}|}{N_{tr}}$.

The procedure for selection of this tuning parameter is quite simple. For Group Lasso penalization technique, we obtain the $\lambda_{\max}$ initially which is defined as: $\arg_{\max} \|\phi(x_l)^\intercal(\Phi^\intercal \alpha - \sum_{i \neq l} \phi(x_i)\hat{\beta}_i)\|$, $\forall\ l = 1, \ldots, N_{tr}$. In order to tune the value of the regularizer λ for varying the amount of sparsity for the reduced set $\mathcal{RS}$, we use different fractional values of $\lambda_{\max}$ as λ. The values of λ are set such that the value of sparsity covers the entire range $[0, 1]$ i.e. we vary the value of λ such that there is no sparsity (sparsity$= 0$) in the model to the case when there is no data point in the reduced set $\mathcal{RS}$ (sparsity $= 1$).

For the L_1, L_0 and $L_1 + L_0$ penalization techniques we have to tune the parameter ρ. To have a fair comparison we use the same range and same values for tuning parameter ρ in case of these techniques. However, the best results for different penalization techniques can occur for different value of tuning parameter ρ. The choice of ρ is again dependent on the amount of sparsity it generates. We aim to select the smallest range of values for ρ such that the value of sparsity covers the entire range $[0, 1]$. From our experiments, we observe that the smallest possible range for ρ corresponding to which sparsity varies from $[0, 1]$ is $[1, 10]$. Thus, we vary the value of ρ in logarithmic steps between the range $[1, 10]$ to obtain the optimal reduced sets.

Out-of-Sample Extension Time Complexity

For the KSC method [13] we consider the entire dataset as the test set. The cardinality of the entire dataset is N. The computational complexity for the out-of-sample predictions for KSC method is $O(N_{tr}N)$ where $N_{tr} \ll N$. This is because for the out-of-sample extension we need to create the test kernel matrix of size $N_{tr} \times N$. When this kernel matrix is too large to be stored in memory then we divide the test data into chunks such that each chunk can fit in memory. Test cluster membership prediction is then done for each chunk.

For the reduced set based methods we can greatly reduce the computational cost for out-of-sample extensions. Let the cardinality of the reduced set corresponding to the Group Lasso, L_0 and $L_1 + L_0$ penalization methods be R_1, R_2, R_3 respectively. Since these methods introduce sparsity, the amount of sparsity

introduced corresponding to these penalization methods can be defined as: R_1/N_{tr}, R_2/N_{tr} and R_3/N_{tr} respectively. The cardinality of the reduced set $\mathcal{RS}$ is much lesser than the size of the training set i.e. $R_i \ll N_{tr}$, $i = 1, 2, 3$. Thus the time complexity for the out-of-sample extension corresponding to the three proposed reduced sets is $O(R_i N)$, $i = 1, 2, 3$. This also reduces the constraint on the memory as the size of the test kernel matrix for the reduced sets becomes $R_i \times N$, $i = 1, 2, 3$ which is much less than the size of the original test kernel matrix ($N_{tr} \times N$).

Synthetic Example

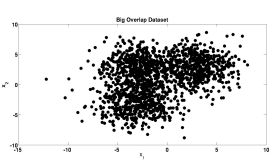
We show the results of an experiment on a synthetic dataset using RBF kernel in Figure 5.13. The dataset consists of 3 overlapping Gaussian clouds in 2-dimensions for a total number of $1,500$ data points. We select 450 data points for training and 600 data points for validation using the quadratic Rènyi entropy criterion.

Figure 5.13 shows the results on this synthetic dataset corresponding to Group Lasso, L_0, L_1,L_1+L_0 penalizations. We vary the regularization parameter λ for Group Lasso and ρ for the other penalization methods. In Figures 5.13b, 5.13d, 5.13f and 5.13h, the 'o'-shaped, red-bodied black-outlined points correspond to the reduced set. In these Figures the training set is constant but the reduced set changes in accordance to the penalization technique. Since the dataset is synthetic, the groundtruth is known beforehand, the quality of the clusters are evaluated using an external quality metric - Adjusted Rand Index (ARI) as defined in [45]. The ARI metric compares the cluster memberships obtained using the reduced set w.r.t. the groundtruth of the test points and higher value of ARI signifies better match between the cluster memberships.

From Figures 5.13b, 5.13c and 5.13j, we observe that the best result for Group Lasso penalization occurs when the regularization parameter $\lambda = 0.8\lambda_{\max}$. It introduces maximal amount of sparsity (sparsity = 0.9933, cardinality of reduced set is 4) while obtaining the best generalization (ARI = 0.56). The best result for L_0 penalization technique takes place for $\rho = 10$ and produces a sparse reduced set (sparsity = 0.9911). But the generalization (ARI = 0.478) is not as good as Group Lasso. This can be observed from Figures 5.13d, 5.13e and 5.13k.

The L_1 and $L_1 + L_0$ penalization techniques produce the same generalization and sparsity for several values of regularizer ρ as depicted in Figures 5.13k and 5.13l. Figure 5.13l indicates that as we increase the value of ρ the amount of sparsity increases. However, when we increase the value of ρ from 8 to 10, then the quality of the clusters decrease as observed from Figure 5.13k. The best

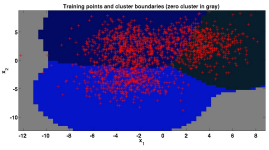
result for the L_1 and $L_1 + L_0$ penalization techniques (sparsity = 0.93, ARI = 0.44) is worse than Group Lasso and L_0 penalization technique both in terms of quality (ARI) and amount of sparsity introduced.



(a) Synthetic Dataset



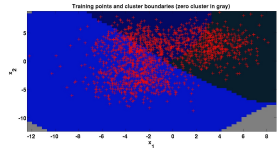
(b) Best GroupLasso Penalization



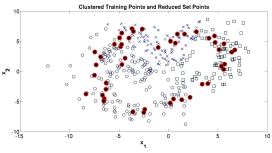
(c) Best GroupLasso Generalization



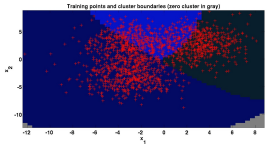
(d) Best L_0 Penalization



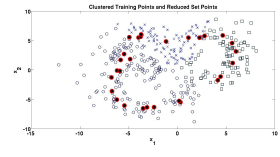
(e) Best L_0 Generalization



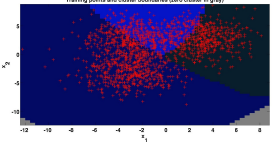
(f) Best L_1 Penalization



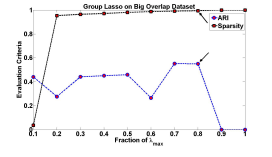
(g) Best L_1 Generalization



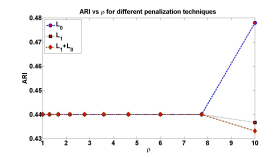
(h) Best $L_1 + L_0$ Generalization



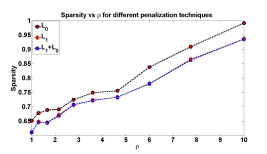
(i) Best $L_1 + L_0$ Generalization



(j) Group Lasso Evaluation



(k) ARI versus ρ



(l) Sparsity versus ρ

Figure 5.13: Results on Synthetic Dataset corresponding to the reduced sets obtained for different penalization techniques.

5.3.3 Experiments on Real World Datasets

Experimental Setup

We conducted experiments on several real world datasets which are available at `http://cs.joensuu.fi/sipu/datasets/`. We provide a brief description of

these datasets in Table 5.7. Since the cluster memberships of these datasets are not known beforehand, we use internal clustering quality metrics for evaluation of the resulting clusters. These internal quality metrics include the widely used silhouette (*sil*) index and the Davies Bouldin (*db*) index as described in [45]. Larger the values of *sil* better the clustering quality and lower the value of *db* better the clustering quality.

Dataset	Points	Dimensions	Clusters
Breast	699	9	2
Bridge	4096	16	-
Europe	169308	2	-
Glass	214	9	7
Iris	150	4	3
Mopsi Location Finland (MLF)	13467	2	-
Mopsi Location Joensuu (MLJ)	6014	2	-
Thyroid	215	5	2
Wdbc	569	32	2
Wine	178	13	3
Yeast	1484	8	10

Table 5.7: Real world datasets. Here '-' means the number of clusters are not known previously

Experimental Results

Table 5.8 showcases the sparsest feasible solution for the different penalization methods and evaluates it on quality metrics like *sil* and *db*. We represent the amount of sparsity as percentage of sparsity rather than fractions (i.e. fraction of sparsity $\times 100$). By feasible solutions we refer to the cases when the cardinality of the reduced set $|\mathcal{RS}| > 0$. For higher values of regularization parameters the cardinality of the reduced set can become zero and these solutions are not part of the feasible solutions.

From Table 5.8 we observe that the Group Lasso penalization introduces the maximum amount of sparsity in general and in the obtained cases the cluster quality by corresponding reduced set is better than the other penalization methods. The Group Lasso penalization performs best for the Europe, Mopsi Location Joensuu (MLJ), Thyroid, Wine and Yeast datasets. We also observe that the proposed L_0 penalization technique generally results in sparser solution than the L_1 penalization method. In some cases it also results in better quality clusters, for example in the cases of Bridge, Europe and Mopsi Location Finland (MLF) datasets. An important observation is that the results corresponding to the proposed $L_1 + L_0$ penalization are quite similar to the results of L_1 penalization. This suggests that the L_1 penalization dominates in each step of the iterative sparsification procedure for the $L_1 + L_0$ penalization method.

	Group Lasso					L_0 **Penalization**					L_1 **Penalization**				
Dataset	*sil*	*db*	Sparsity	λ	Secs	*sil*	*db*	Sparsity	ρ	Secs	*sil*	*db*	Sparsity	ρ	Secs
Breast	0.6824	0.85	**99.5**%	$0.7\lambda_{max}$	**0.99**	0.6898	0.833	96.6%	7.74	19.5	0.6898	0.833	96.6%	7.74	6.7
Bridge	0.423	1.436	**99.0**%	$0.7\lambda_{max}$	**34.8**	**0.596**	1.3	97.1%	4.642	701.0	0.559	1.72	98.5%	5.995	238.4
Europe	**0.437**	2.1	**99.9**%	$0.9\lambda_{max}$	**4509**	0.352	**1.145**	99.75%	10	210,512	0.352	1.148	99.7%	10	61,456
Glass	0.33	3.133	14.0%	$1.0\lambda_{max}$	**0.12**	0.3223	1.913	93.75%	2.155	2.42	0.408	**1.813**	**96.9**%	4.64	0.68
Iris	0.611	1.333	**93.33**%	$0.9\lambda_{max}$	**0.03**	0.605	1.323	84.45%	2.78	1.02	0.6184	1.3063	86.67%	3.598	0.28
MLF	0.7219	1.2735	**99.7**%	$0.6\lambda_{max}$	**301.2**	**0.7976**	**1.142**	99.6%	10	6,586	0.7946	1.158	99.5%	10	2,315
MLJ	**0.911**	**0.64**	**99.5**%	$0.7\lambda_{max}$	**80.2**	0.88	0.67	96.5%	10	1,720	0.8814	0.6684	95.5%	10	612
Thyroid	**0.6345**	**0.844**	**98.4**%	$6\lambda_{max}$	**0.13**	0.538	1.04	93.75%	5.995	2.48	0.538	1.04	93.75%	5.995	0.7
Wdbc	0.5585	1.304	96.5%	$0.9\lambda_{max}$	**0.78**	0.56	1.303	97.6%	5.995	13.2	0.56	1.303	97.6%	5.995	4.11
Wine	**0.291**	**1.943**	5.6%	$1.0\lambda_{max}$	**0.05**	0.29	1.96	85.0%	1.668	1.28	0.29	1.96	86.8%	2.154	0.31
Yeast	**0.81**	2.7	97.76%	$0.9\lambda_{max}$	**4.01**	0.258	2.3	**97.9**%	7.74	79.1	0.2637	**2.2**	83.0%	10	25.2

Table 5.8: Comparison of the sparsest feasible solution for the different penalization methods. Here we highlight the unique best results i.e. the best results are highlighted if they correspond to a single penalization method. In most of the cases the L_1 and $L_1 + L_0$ penalization result in the same sparsest solution.

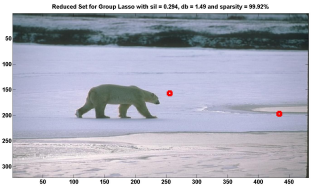
Real World Image Segmentation Dataset

We also perform an image segmentation experiment such that each pixel is transformed into a histogram and the χ^2 distance is used in the RBF kernel with bandwidth σ_χ as shown in Figure 5.14. The total number of pixels is $154,401$ (321×481). The training set consists of $N_{tr} = 7,500$ pixels and the validation set consists of $10,000$ pixels. Both these set are selected by maximizing the quadratic Rènyi entropy. After validation we obtain $k = 3$ and kernel parameter $\sigma_\chi = 2.807$. We performed this experiment on a 2.4 GHz Core 2 Duo, 12 Gb RAM machine using MATLAB 2012b.

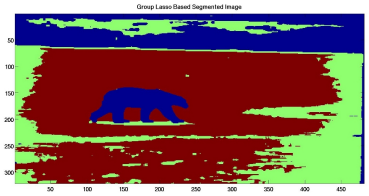
The Group Lasso based penalization method reduces the reduced set to just two data points for $\lambda = 0.7\lambda_{max}$ and still has the best $sil = 0.294$ value as shown in Figure 5.14a. In KSC [13] there is a possibility of a null cluster i.e. the cluster which is beyond the generalization boundary of all the clusters. The Group Lasso penalization technique produces 2 points in the reduced set, one corresponding to each cluster. The third cluster corresponds to the null cluster. Hence, it results in good segmentation as observed from Figure 5.14b. The L_0 penalization also results in a highly sparse model (sparsity $= 0.9645$) and has the smallest $db = 0.141$ value as observed from Figure 5.14c.

The results corresponding to L_1 and $L_1 + L_0$ penalization techniques are same for this image dataset. Thus, we only show the result for L_1 penalization technique in Figures 5.14e and 5.14f. The optimal value of the tuning parameter ρ for these penalization techniques was $\rho = 10$. From Figures 5.14d and 5.14f, we observe that the best image segmentation results for the L_0 and L_1 penalization technique is the same. However, the L_0 penalization technique produces more sparsity (0.9645) than L_1 penalization technique (0.948) to obtain the same

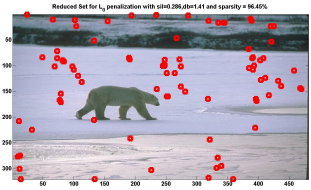
segmentation. We obtain good image segmentation in case of both the Group Lasso and the L_0 penalization technique.



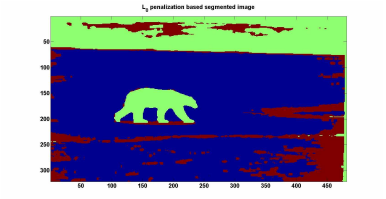
(a) Group Lasso based Reduced Set



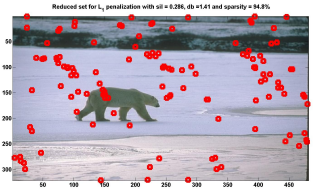
(b) Best GroupLasso based Image segmentation



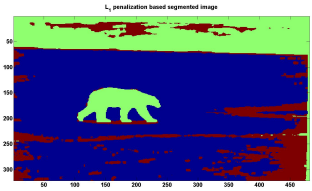
(c) Best L_0 penalization based Reduced Set



(d) Best L_0 penalization based Image segmentation



(e) Best L_1 penalization based Reduced Set [51]



(f) Best L_1 penalization based Image segmentation [51]

Figure 5.14: Results on Image Dataset corresponding to the reduced sets obtained via different penalization techniques. The red-colored circular boxes represents the points selected as reduced set points

Discussion

We have used several penalization techniques to obtain optimal reduced sets for kernel spectral clustering. We observe that the Group Lasso based penalization technique results in maximum sparsity in many cases and is computationally the most efficient as shown in Table 5.8. The Group Lasso based penalization technique is also ideal for clustering as it retains groups of relevant data points. The L_0 penalization technique results in sparser solution than L_1 penalization

technique in general but at the expense of more computational time. This is because it iteratively solves a QCQP for each sparsification step whereas there is no such iterative procedure for L_1 penalization technique. This can also be concluded from the computation time shown for the two methods in Table 5.8. We also observe that as the size of the dataset increases the L_0, L_1 and $L_1 + L_0$ penalization based techniques become less feasible. This is because CVX is meant for smaller size optimization problems and cannot handle very large scale problems efficiently.

5.3.4 Conclusion

We proposed several methods for obtaining sparse optimal reduced sets for kernel spectral clustering. The formulation is based on weighted kernel PCA for a specific choice of weights. Several techniques like Group Lasso, L_0, $L_1 + L_0$ penalization methods had been proposed to obtain the reduced set along with the modified weight vectors. The methodologies were aimed to tackle different datasets in a computationally and memory efficient way. We observed that the Group Lasso resulted in the sparsest models with good clustering quality in least computation time followed by reduced models by the L_0 penalization. The reduced models obtained by $L_1 + L_0$ penalization technique are quite similar to the reduced models obtained by a previous L_1 penalization method.

Chapter 6

Applications of Supervised & Unsupervised Kernel Methods

This chapter comprises of previously published and submitted articles including:

1) Mall R., El Anbari M., Bensmail H., Suykens J.A.K., **"Primal-Dual Framework for Feature Selection using Least Squares Support Vector Machines"**, in Proc. of the 19th International Conference on Management of Data (COMAD), Ahmedabad, India, Dec. 2013, pp. 105-108.

2) Mall R., Langone R., Suykens, J.A.K., **"Netgram: Visualizing Communities in Evolving Networks"**, submitted for publication.

Keywords—evolving communities; dynamic community detection; tracking communities; visualization; L_0-norm; feature selection;

6.1 Reweighted L_1 LSSVM for Feature Selection[1]

Abstract—LSSVM performs classification using L_2-norm on the weight vector and a squared loss function with linear constraints. The major advantage over

[1]This section consists of Sections 1 – 4 from Mall R., El Anbari M., Bensmail H., Suykens J.A.K., **"Primal-Dual Framework for Feature Selection using Least Squares Support Vector Machines"**, in Proc. of the 19th International Conference on Management of Data (COMAD), Ahmedabad, India, Dec. 2013, pp. 105-108.

classical L_2-norm support vector machine (SVM) is that it solves a system of linear equations rather than solving a quadratic programming problem. The L_2-norm penalty on the weight vectors is known to robustly select features. The zero-norm or the number of non-zero elements in a vector is an ideal quantity for feature selection. The L_0-norm minimization is a computationally intractable problem. However, a convex relaxation to the direct zero-norm minimization was proposed recently in form of reweighted L_1 penalization. In this work, we propose a combination of L_2-norm penalty and the convex relaxation of the L_0-norm penalty for feature selection in classification problems. We propose a primal-dual framework for feature selection using the combination of L_2-norm and L_0-norm penalty resulting in closed form solution. A series of experiments on microarray data and UCI data demonstrates that our proposed method results in better performance.

6.1.1 Introduction

Least Squares Support Vector Machines (LSSVM) [9] is an alternative to the standard support vector machines (SVM) [10]. It is a widely used tool for classification and regression problems. Given a dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, where the input $x_i \in \mathbb{R}^d$ is a vector with d features and the class label $y_i \in \{-1, +1\}$, the LSSVM finds an optimal hyperplane to separate the two classes using the following optimization problem:

$$\begin{aligned} \min_{w,e_k,b} \quad & \frac{1}{2}\lambda||w||^2 + \frac{1}{2}\sum_{k=1}^{N} e_k^2 \\ \text{such that} \quad & e_k = y_k - w^\intercal \phi(x_k) - b, k = 1, \ldots, N, \end{aligned} \tag{6.1}$$

where λ is a regularization constant, e_k is the error corresponding to the kth point and b is the bias term. Here $\phi(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}^{n_h}$ is a mapping to a high dimensional feature space as in the standard SVM [10] case. Throughout this work, we use the *linear kernel*. This means that $\phi(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}^d$ or $\phi(x) = x$. This allows to have interpretable models as the feature space is known beforehand. Finally the classifier in the primal is defined as: $y(x) = \text{sign}[w^\intercal \phi(x) + b]$.

The corresponding dual classifier is defined as: $y(x) = \text{sign}[\sum_{k=1}^{N} \alpha_k K(x_k, x) + b]$. Here $K(x_k, x_j) = \phi(x_k)^\intercal \phi(x_j)$, K is a positive definite kernel function and α_k are the Lagrange multipliers which can be positive or negative due to equality constraints. The KKT conditions lead to $w = \sum_{k=1}^{N} \alpha_k \phi(x_k)$ and $e_k = \frac{1}{\gamma}\alpha_k$. The second KKT condition makes the LSSVM solutions non-sparse i.e. each data point is considered as a support vector. Thus, the LSSVM formulation in [9] has the form of a *penalty+loss* with the λ playing the role of regularizer.

The major advantage of the LSSVM formulation over a standard SVM is that the equality constraints and the squared loss function leads to solving a system of linear equations instead of a quadratic programming (QP) problem as in the case of classical SVM. It is widely known [9, 173] that solving a system of linear equations is computationally easier than solving QPs. In this work, we take this into consideration and the proposed method always solves a system of linear equations and have closed form solutions.

The zero-norm defined as $||w||_0 = \text{card}\{w_i|w_i \neq 0\}$ counts the number of non-zero elements in the vector w. When the zero-norm is minimized it results in very sparse models. Recently, the zero norm has been receiving a lot of attention in the machine learning community [66, 79, 80, 89]. The minimization of the zero-norm is a computationally intractable problem as shown in [174]. This is because the zero-norm minimization is non-convex and NP-hard problem. However, recently a direct zero-norm optimization method based on reweighting the L_1 penalty was proposed in [175] which can achieve the true zero-norm asymptotically under Bayesian interpretation. This is closely related to Automatic Relevance Determination (ARD) for feature selection [176].

Motivations & Contributions

The role of L_2-norm in feature selection for SVM classifiers is to select the similar set of features upon different randomizations of the data. This leads to robustness in selection of features [177]. The L_2-norm penalty also results in shrinkage. It fits the coefficients toward zero but cannot make the coefficients *exactly* zero. So, in this work, we combine the L_2-norm penalty along with the convex relaxation for direct zero-norm penalty as formulated in [175, 80] for feature selection using LSSVM classifiers. The proposed method selects groups of essential features for classification and eliminates the unnecessary variables. We propose a primal-dual framework for sparse feature selection using a combination of L_2-norm and L_0-norm penalty while taking into consideration both the cases when $N \gg d$ and when $d \gg N$. Due to space limitations we refer the readers to [66, 175, 61, 62, 54, 56, 58, 60, 63, 64] for related work.

6.1.2 Proposed Method

The direct zero-norm optimization method results in an iterative convex formulation for L_0-norm based classifiers [80, 175]. It results in a local minimum to the non-convex zero-norm problem with good predictive capabilities and sparsity in both the feature and input space [175, 80]. However, the L_0-norm penalty doesn't guarantee the selection of the same set of variables for different

randomizations. Thus, we use the L_2-norm penalty in combination with L_0-norm penalty along with a squared loss function in our formulation.

Primal Formulation

We pre-process the dataset $\mathcal{D}$ to be mean-centered and have unit norm along each dimension d. Since the data is mean-centered we don't have the intercept term b. The constrained optimization problem for the proposed approach at iteration t is given by:

$$\begin{aligned} \min_{w^{(t)},e_k} \quad & \frac{1}{2}\lambda||w||^2 + \frac{1}{2}w^\intercal \Lambda^{(t-1)} w + \frac{1}{2}\sum_{k=1}^{N} e_k^2 \\ \text{such that} \quad & e_k = y_k - w^\intercal x_k, k = 1, \ldots, N, \end{aligned} \tag{6.2}$$

where λ is the regularization parameter and $\Lambda^{(t-1)} = \text{diag}(\frac{1}{|w_1^{(t-1)}|^2}, \ldots, \frac{1}{|w_d^{(t-1)}|^2})$. The $w^\intercal \Lambda^{(t-1)} w$ term in the optimization function is the convex relaxation to the $||w||_0$ minimization. The L_0-norm penalty term is based on reweighted L_1-norm penalty as that introduced in [175, 80]. After elimination of e_k in (6.2), we can obtain the following convex unconstrained optimization problem:

$$\min_{w^{(t)}} \quad \frac{1}{2}\lambda||w||^2 + \frac{1}{2}w^\intercal \Lambda^{(t-1)} w + \frac{1}{2}\sum_{k=1}^{N} (y_k - w^\intercal x_k)^2 \tag{6.3}$$

The solution to (6.3) at each iteration t can be obtained by directly differentiating the convex optimization function in (6.3) w.r.t to w. It results in a iteratively weighted ridge regression [178] like solution:

$$w^{(t)} = (\lambda \mathcal{I} + \Lambda^{(t-1)} + \mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal Y \tag{6.4}$$

where $\mathbf{X} = [x_1, x_2, \ldots, x_N]^\intercal$ and $Y = [y_1, y_2, \ldots, y_N]^\intercal$. This solution corresponds to the primal and is more appropriate for the case when $N \gg d$. The final classifier in the primal is then defined as: $y(x) = \text{sign}[w^{(t)\intercal} x]$.

Since the proposed approach follows an iterative procedure to a local minimum, it is needed to have a good starting value. We initially solve the LSSVM problem to obtain the weight vector $w^{(0)}$. The regularization parameter λ is also obtained by solving the LSSVM problem via coupled simulated annealing (CSA) [93]. Thus, the initial value of $\Lambda^{(0)} = \text{diag}(\frac{1}{|w_1^{(0)}|^2}, \ldots, \frac{1}{|w_d^{(0)}|^2})$. The L_0-norm penalty doesn't introduce any additional tuning parameters as in [175], performs direct zero-norm objective minimization and is advantageous over AROM and FSV methods.

Dual Formulation

One of the KKT conditions of LSSVM provides the connection between the primal weight vector w and the dual Lagrange multipliers α_k. The relation is given by $w = \sum_{k=1}^{N} \alpha_k x_k = \mathbf{X}^\intercal \alpha$ where $\alpha = [\alpha_1, \ldots, \alpha_N]^\intercal$. In the case when the number of points in the dataset is much less than the number of features in the dataset i.e. $d \gg N$, it is more suitable to solve the problem in the dual. Given the connection between w and α, replacing α in (6.3) results in the following convex unconstrained optimization problem:

$$\min_{\alpha^{(t)}} \quad \frac{1}{2}\lambda \alpha^\intercal \mathbf{X}\mathbf{X}^\intercal \alpha + \frac{1}{2}\alpha^\intercal \mathbf{X}\Lambda^{(t-1)}\mathbf{X}^\intercal \alpha + \frac{1}{2}\sum_{k=1}^{N}(y_k - \alpha^\intercal \mathbf{X} x_k)^2 \tag{6.5}$$

where λ is the regularization parameter and $\Lambda^{(t-1)} = \text{diag}(\frac{1}{|w_1^{(t-1)}|^2}, \ldots, \frac{1}{|w_d^{(t-1)}|^2})$. The $\alpha^\intercal \mathbf{X}\Lambda^{(t-1)}\mathbf{X}^\intercal \alpha$ term in the optimization function is the convex relaxation to the $||w||_0$ minimization. The solution to (6.5) at each iteration t can be obtained by directly differentiating the convex optimization function in (6.5) w.r.t to α. In (6.5), we can replace $\mathbf{X}\mathbf{X}^\intercal$ by the kernel matrix K as it is the linear kernel case. The solution to (6.5) is given by:

$$\alpha^{(t)} = (\lambda K + \mathbf{X}\Lambda^{(t-1)}\mathbf{X}^\intercal + KK^\intercal)^{-1} K^\intercal Y \tag{6.6}$$

Once we obtain the solution vector $\alpha^{(t)}$ for iteration t, we recalculate the weight vector $w^{(t)} = \sum_{k=1}^{N} \alpha_k^{(t)} x_k$ and re-evaluate $\Lambda^{(t)}$ using the aforementioned procedure. The initial coefficients $\alpha^{(0)}$ and the regularization parameter λ are obtained by solving the LSSVM classifier in the dual. The initial weight vectors $w^{(0)} = \sum_{k=1}^{N} \alpha_k^{(0)} x_k$ and $\Lambda^{(0)} = \text{diag}(\frac{1}{|w_1^{(0)}|^2}, \ldots, \frac{1}{|w_d^{(0)}|^2})$.

Stopping Criteria

The iterative procedure proposed for the primal and dual formulation is executed till we either reach convergence or we reach a maximum number of iterations (*max iterations*). In case of the primal, we define a threshold $\theta = \frac{||w^{(t)} - w^{(t-1)}||_2^2}{d}$. For the dual this threshold is defined as $\theta = \frac{||w^{(t)} - w^{(t-1)}||_2^2}{N}$. We continue the iterative procedure until this threshold θ reaches machine precision (denoted by ϵ). Empirically, we observed that generally 5 to 10 iterations suffice. Once the iterative procedure stops, we follow the setup in [66] and select the top r features from the weight vector such that $||w||_0 = r$.

Algorithm 17: Primal-Dual framework for feature selection using LSSVM

Data: $\mathcal{D} = \{(x_i, y_i) : x_i \in \mathbb{R}^d, y_i \in \{+1, -1\}$ for classification, $i = 1, \ldots, N\}$.
Result: The optimal feature vector $w \in \mathbb{R}^d$ s.t. $|w_i| \geq 0$, $i = 1, \ldots, r$.

```
if N ≫ d then
    Solve LSSVM classifier in primal to obtain w^(0) and λ.
    Initialize Λ^(0) = diag(1/|w_1^(0)|^2, ..., 1/|w_d^(0)|^2), θ = inf & cnt = 0.
    while θ > ε and cnt < max iterations do
        Solve (6.4) to obtain w^(t).
        Calculate Λ^(t) = diag(1/|w_1^(t)|^2, ..., 1/|w_d^(t)|^2).
        Estimate θ = ||w^(t) − w^(t−1)||_2^2 / d.
        Increment cnt to cnt + 1.
    end
end
else if d ≫ N then
    Solve the LSSVM classifier in the dual to obtain α^(0) and λ.
    Calculate w^(0) = Σ_{k=1}^{N} α_k^(0) x_k.
    Initialize Λ^(0) = diag(1/|w_1^(0)|^2, ..., 1/|w_d^(0)|^2), θ = inf & cnt = 0.
    while θ > ε and cnt < max iterations do
        Solve (6.6) to obtain α^(t).
        Estimate w^(t) = Σ_{k=1}^{N} α_k^(t) x_k.
        Calculate Λ^(t) = diag(1/|w_1^(t)|^2, ..., 1/|w_d^(t)|^2).
        Evaluate θ = ||w^(t) − w^(t−1)||_2^2 / d.
        Increment cnt to cnt + 1.
    end
end
Sort the final weight vector w based on its absolute values.
Select the top r features s.t. ||w||_0 = r and set rest to 0.
```

6.1.3 Experimental Results

In this section, we compare our proposed L_2-norm and L_0-norm (L2+L0) penalty based feature selection method with FSV method [56] and L_1-SVM [58] from the LibLinear library (`http://www.csie.ntu.edu.tw/~cjlin/liblinear/`) in the primal as these methods have formulations in the primal. We compare the proposed approach with AROM method [66] and Recursive Feature Elimination (RFE) [179] in the dual as these methods are computationally cheaper in the dual. We utilize the implementation of the afore-mentioned methods from the matlab toolbox of Spider (`http://www.kyb.tuebingen.mpg.de/bs/people/spider/index.html`). We also compare the proposed methodology with a primal-dual formulation of direct zero-norm minimization based LSSVM (D-L0) [175] and the original LSSVM [9].

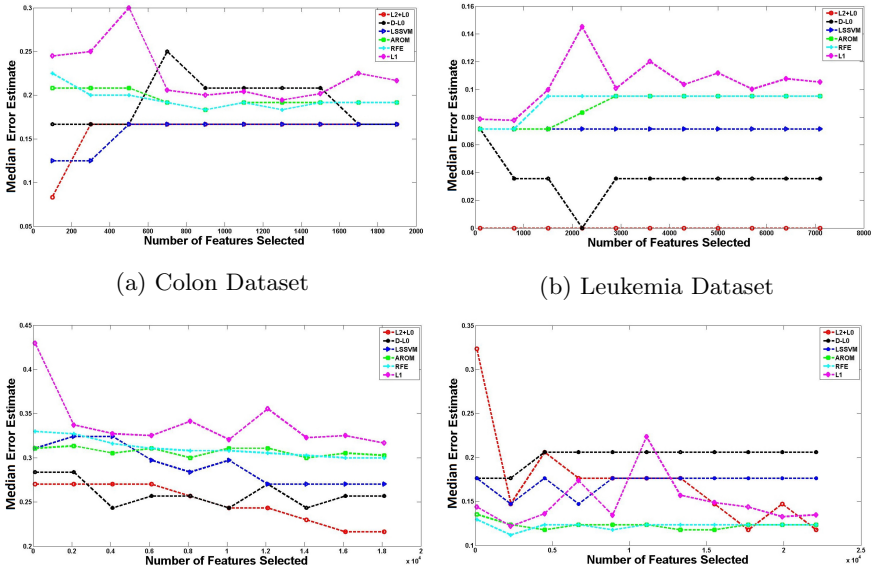
Experiments

We demonstrate our results on 4 microarray gene datasets in the dual. Out of these 4 datasets, two datasets are cancer microarray datasets namely Colon and Leukemia which are obtained from UCI repository [73]. The other two microarray datasets are obtained from `http://featureselection.asu.edu/datasets.php`. We also illustrate the effectiveness of our proposed approach over 6 datasets in the primal. These datasets are also obtained from the UCI repository. We randomly partition the dataset into 80% as the training set and 20% as the test set. In order to estimate the value of the hyper-parameter λ, we perform 50 cross-validations of LSSVM using CSA [93]. We first use the training set for feature selection by specifying a given number of features (r). After obtaining the desired weight vector w, classification is performed over the test set using this reduced weight vector. All the experiments are conducted on a PC with 4 Gb RAM, 3Ghz CPU using Matlab 2009a.

Dual Experimental Results

We evaluate the predictive performance of various feature selection methods in the dual on the 4 microarray gene datasets as shown in Figure 6.1. From Figure 6.1 we can observe that the $L_2 + L_0$-norm penalty based proposed approach results in lower or equal error estimates than the original LSSVM in most cases for different value of r. This justifies the need of feature selection before prediction is done. For all the datasets, the L_2-norm and L_0-norm penalty (L2+L0) based method and the direct L_0-norm (D-L0) based method perform better than AROM, RFE, L_1-norm SVM and standard LSSVM for different values of r with the exception of **GLI** dataset. For the **GLI** dataset, the AROM, RFE and L_1-norm SVM performs better but they are computationally more expensive methods. In general, between the proposed approach (L2+L0) and D-L0, our method performs better for all the 4 microarray datasets.

Table 6.1 contains information about the largest common subset size over 10 randomizations for different feature selection methods. This indicates the features that appeared consistently during each randomization for a given value of r s.t. $||w||_0 = r$. Higher values indicate the presence of a set of variable which is consistently being selected. Thus, it corresponds to the *robustness* of the proposed approach. We observe that the proposed (L2+L0) approach is more robust in selection of features than the other methods for Col and Leu (cancer microarray datasets). However, for the SMK dataset, the LSSVM method shows more *robustness* in general. As we mentioned earlier that it is the L_2-norm penalty which leads to robustness in selection of similar sets of variables, the



(a) Colon Dataset

(b) Leukemia Dataset

(c) SMK-CAN-187 Dataset

(d) GLI-85 Dataset

Figure 6.1: Results of various feature selection methods for different subset size on several microarray datasets. The red line corresponds to proposed L2+L0 method.

standard LSSVM formulation also uses L_2-norm penalty and hence shows this *robustness*.

Primal Experimental Results

We conducted experiments on 6 UCI datasets in the primal i.e. when $N \gg d$. Table 6.2 demonstrates the effectiveness of the proposed approach in comparison to methods like L_1 SVM, FSV method, direct zero-norm based LSSVM and the standard LSSVM. Table 6.2 contains information about the value of r corresponding to which each method performs best in terms of predictive power.

From Table 6.2 we observe that the proposed approach (L2+L0) outperforms other methods in terms of accuracy for 3 datasets. It showcases that our method leads to maximum sparsity w.r.t. feature selection. However, for the Musk1 (Mus) dataset feature selection is not beneficial. This can be observed from Table 6.2 since the best results correspond to LSSVM for $r = 166$. We only

Data	Method	Largest Common Feature subset size									
		100	**300**	**500**	**700**	**900**	**1100**	**1300**	**1500**	**1700**	**1900**
	L2+L0	**26**	**66**	**120**	**196**	**290**	**395**	**508**	**648**	**870**	**1412**
C	**D-L0**	2	13	41	83	145	236	350	502	751	1327
O	**AROM**	3	12	39	78	148	225	352	498	753	1330
L	**RFE**	7	18	37	90	135	240	346	498	771	1350
	L1	8	16	36	84	140	238	351	501	768	1346
	LSSVM	15	52	87	150	214	322	435	568	810	1354
		100	**800**	**1500**	**2200**	**2900**	**3600**	**4300**	**5000**	**5700**	**7100**
	L2+L0	**17**	**229**	**491**	**766**	**1134**	**1487**	**1866**	**2325**	**3165**	**7037**
L	**D-L0**	9	193	434	676	1022	1370	1795	2267	2859	6882
E	**AROM**	10	183	431	667	1015	1410	1850	2238	2901	6866
U	**RFE**	8	178	429	669	1018	1312	1750	2256	2714	6737
	L1	7	169	422	671	1001	1332	1772	2301	2702	6797
	LSSVM	13	219	454	704	1032	1376	1773	2205	2788	6872
		100	**2300**	**4500**	**6700**	**8900**	**11100**	**13300**	**155000**	**17700**	**22100**
	L2+L0	**100**	229	856	1905	**3400**	**5420**	**7798**	**10715**	**14026**	**21920**
G	**D-L0**	2	380	671	1066	1610	2469	3653	5357	7572	20853
L	**AROM**	12	278	651	1166	1810	2579	3573	5735	8127	19959
I	**RFE**	8	292	701	1256	1610	2456	3842	5912	7601	20129
	L1	8	288	699	1244	1700	2501	3678	5882	7812	20259
	LSSVM	4	**452**	**1137**	**1938**	2899	4048	5305	6785	8633	20749
		100	**2100**	**4100**	**6100**	**8100**	**10100**	**12100**	**14100**	**16100**	**18100**
	L2+L0	**10**	34	210	545	1064	1902	2989	4545	6778	**10772**
S	**D-L0**	7	54	215	534	1017	1664	2584	3926	5906	9849
M	**AROM**	6	33	201	526	999	1676	2612	4010	6091	9958
K	**RFE**	5	32	212	536	1010	1767	2588	3992	5990	10100
	L1	6	28	221	522	1009	1812	2489	3891	6019	9845
	LSSVM	6	**293**	**741**	**1308**	**2001**	**2908**	**3918**	**5157**	**6995**	10340

Table 6.1: Comparison of largest common feature set sizes over 10 randomizations for different feature selection methods in the dual corresponding to various values of r

highlight those results which are unique and correspond to best performance and least number of features used. We also infer that the FSV method is computationally quite expensive and is infeasible for datasets like Tit and TN. Hence in Table 6.2 the results aligning to the FSV method for these datasets are represented by '-'.

6.1.4 Conclusion

In this work, we proposed a combination of L_2-norm penalty and the convex relaxation of the L_0-norm penalty for feature selection in classification problems. The proposed method was formulated in a primal-dual framework by iteratively solving a system of linear equations. It is computationally easier than standard QP-based SVM solvers. The L_2-norm penalty helped in robustly selecting variables during each randomization whereas the L_0-norm penalty reduced the noisy feature coefficients to zero. We demonstrated the efficiency of the

Table 6.2: Performance comparison over 6 datasets in the primal

Datasets	L2+L0			D-L0			L1			FSV			LSSVM		
	Err	Time	r	Err	Time	r	Err	Time	r	Err	Time	r	Err	Time	r
BC	0.02 ± 0.01	0.01 ± 0.0	8	0.02 ± 0.01	0.01 ± 0.0	8	0.04 ± 0.01	0.11 ± 0.01	8	0.03 ± 0.01	424.3 ± 20	10	0.02 ± 0.01	0.01 ± 0.0	8
GER	**0.32 ± 0.03**	0.01 ± 0.01	**7**	0.33 ± 0.03	0.02 ± 0.01	9	0.36 ± 0.01	0.15 ± 0.02	16	0.33 ± 0.02	1040 ± 13.0	16	0.29 ± 0.02	0.01 ± 0.01	16
Mus	0.25 ± 0.04	0.04 ± 0.0	**16**	0.19 ± 0.06	0.04 ± 0.0	166	0.28 ± 0.03	0.14 ± 0.02	166	0.18 ± 0.03	10.5 ± 0.9	166	**0.17 ± 0.03**	0.03 ± 0.0	166
Son	**0.24 ± 0.05**	0.01 ± 0.0	**4**	0.24 ± 0.14	0.01 ± 0.0	32	0.25 ± 0.06	0.12 ± 0.02	60	0.26 ± 0.05	1.44 ± 0.11	32	0.28 ± 0.07	0.01 ± 0.0	60
Tit	**0.21 ± 0.02**	0.0 ± 0.0	3	0.22 ± 0.02	0.0 ± 0.0	3	0.22 ± 0.0	0.12 ± 0.03	3	-	-	-	0.22 ± 0.02	0.0 ± 0.0	3
TN	0.02 ± 0.0	0.02 ± 0.0	20	0.02 ± 0.0	0.02 ± 0.0	20	0.02 ± 0.0	0.18 ± 0.03	20	-	-	-	0.02 ± 0.0	0.02 ± 0.0	20

proposed approach on 10 real world datasets and evaluated it against several state-of-the-art feature selection based SVM classifiers.

6.2 Netgram: Visualizing Communities in Evolving Networks[2]

Abstract—Real-world complex networks are dynamic in nature and change over time. The change is usually observed in the interactions within the network over time. Complex networks exhibit community like structures. A key feature of the dynamics of complex networks is the evolution of communities over time. Several methods have been proposed to detect and track the evolution of these groups over time. However, there is no generic tool which visualizes all the aspects of group evolution in dynamic networks including birth, death, splitting, merging, expansion, shrinkage and continuation of groups. In this paper, we propose Netgram: a tool for visualizing evolution of communities in time-evolving graphs. Netgram maintains evolution of communities over 2 consecutive time-stamps in tables which are used to create a query database using the sql outer-join operation. It uses a line-based visualization technique which adheres to certain design principles and aesthetic guidelines. Netgram uses a greedy solution to order the initial community information provided by the evolutionary clustering technique such that we have fewer line cross-overs in the visualization. This makes it easier to track the progress of individual communities in time evolving graphs. Netgram is a generic toolkit which can be used with any evolutionary community detection algorithm as illustrated in our experiments. We use Netgram for visualization of topic evolution in the NIPS conference over a period of 11 years and observe the emergence and merging of several disciplines in the field of information processing systems.

6.2.1 Introduction

Large scale complex networks are ubiquitous in the modern era. Their presence spans a wide range of domains including social networks, biological networks, collaboration networks, trust networks, financial networks etc. These complex networks have a natural temporal aspect. Social networks evolve over time with addition and deletion of members, formation of friendships between people in different social circles or disappearance of friendship of people over time. In

[2]This section consists of Section 1 – 6 from Mall R., Langone R., Suykens, J.A.K., **"Netgram: Visualizing Communities in Evolving Networks"**, submitted for publication.

collaboration networks, a group of researchers working on a particular topic might collaborate intensively if they are working on an emerging topic whereas a group of researchers working together on an outdated topic might completely disappear over time.

Complex networks can be represented as graphs $G = (V, E)$ where V represent the vertices or nodes and E represents the edges or interaction between these nodes in the network. Most real-life networks exhibit community like structure i.e. nodes within a community are more densely connected to each other and sparsely connected to nodes outside that cluster. Traditionally, community detection methods [104, 103, 39, 38, 42, 146, 46, 47, 48, 49, 139, 36, 37, 40, 13] have focused on identifying communities in static representations of graphs.

However, by representing a time-evolving graph as a static network [146] it becomes difficult to detect and track the intrinsic changes in the community structures over time. By performing community detection on static snapshots [146] of the dynamic network, we loose the property of temporal smoothness which is essential to capture the evolution of communities over time. Temporal smoothness [67, 68] allows to preserve the long-term trend in the dynamic graph while smoothing out short-term variations due to noise. This property is similar to the property of moving averages in time-series analysis [67, 68]. In recent years, the problem of evolutionary community detection and its tracking in large scale dynamic social networks has received a lot of attention [67, 68, 42, 69, 70, 71, 180, 181, 182, 183, 72, 184]. The goal of evolutionary community detection is to identify and track communities at different snapshots of time in non-stationary graphs. Throughout this paper we use the notation t and T interchangeably to represent time, we use C_i^t to represent the ith cluster at time-stamp t and $\mathcal{C}^t$ to represent the set consisting of all the clusters C_i^t at time-stamp t.

In this paper, we propose a new tool Netgram which allows to visualize the evolution of communities in dynamic graphs. In order to visualize the progress of communities over time, Netgram follows a series of steps. Netgram takes as input the information about the individual identifier for all the nodes i.e. the number/label with which that node is represented in the network and its corresponding cluster membership at different time-stamps. It can be used as a post-processing step to any evolutionary community detection algorithm. After obtaining the input, in order to capture the significant events namely birth, death, merge, split, growth, shrinkage and continuation of communities, Netgram uses a modified version of the tracking algorithm proposed in [184]. We provide a visualization of the weighted network (W^t) at time-stamp t generated as a result of the tracking procedure. Once the evolution of communities is captured between two successive time-stamps, it is stored in a table. We then perform sql [185] join operations on these sets of tables to construct a query

database. This query database contains information about evolution of all the communities over the different time-stamps. We then visualize this query database using separate colors for cluster identifiers and lines which track the evolution of individual communities.

Netgram tries to adhere to certain design principles and a set aesthetic guidelines including minimizing the line cross-overs between the evolving communities during events like merge and split making it easier to track the evolution of individual communities in the dynamic network. The problem of minimizing the cross-talk between communities during different time-stamps is combinatorial by nature and Netgram uses a greedy solution for the same. Figure 6.2 illustrates the steps undertaken by the Netgram toolkit for visualizing evolution of communities. Figure 6.3 showcases the result that we get from the Netgram toolkit on a synthetic Birthdeath dataset of $1,000$ nodes over 5 time-stamps generated from the software `https://github.com/derekgreene/dynamic-community`.

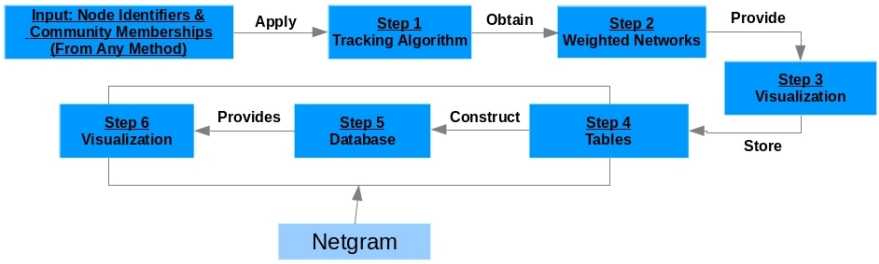
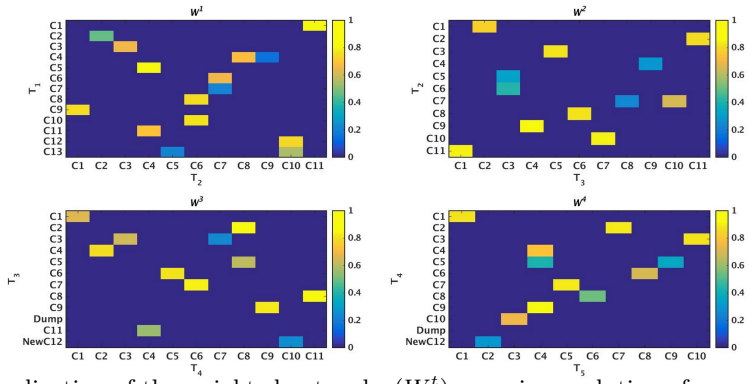


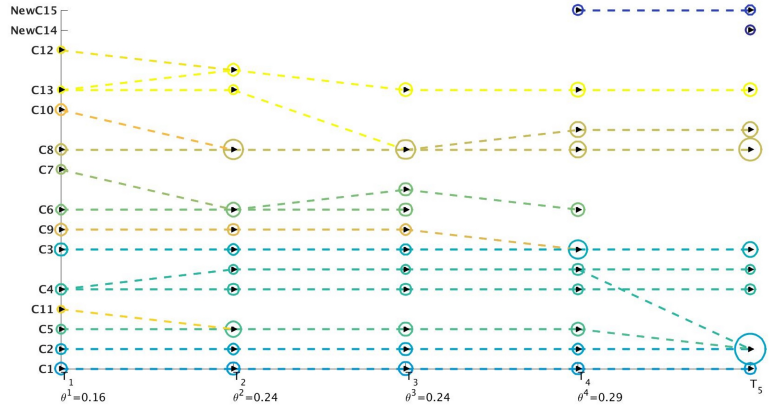
Figure 6.2: Steps undertaken by Netgram for visualizing evolution of communities in dynamic networks.

6.2.2 Related Work

We briefly mention here some of the methods that have been used in the past for detecting and tracking changes in complex networks. In [186] GraphScope was introduced. Graphscope is an efficient adaptive mining tool in time evolving graph for detecting communities. It requires no user-defined parameter and operates completely in a standalone mode using the principle of Minimum Description Length (MDL) from information theory. Moreover, it can automatically detect communities and determine good *change-points* over time. The intuition is that communities do not change much over 2 consecutive time-stamps and thus have similar description lengths. This allows to group them together into a time segment in order to achieve better compression.



(a) Visualization of the weighted networks (W^t) mapping evolution of communities over 5 time-stamps. W^t tracks evolution of a cluster between two consecutive time-stamps. The colors represent the weight of the edges in W^t. The weights can take value in the range $[0, 1]$ i.e. $0 \leq w(\mathrm{V}^t(j,k)) \leq 1$.



(b) Visualization and tracking of community evolution by Netgram for the clusters obtained from the Kernel Spectral Clustering with Memory Effect (MKSC) algorithm [72] for Birthdeath dataset.

Figure 6.3: Netgram showcases the birth, death, merge, split, expansion, shrinkage and continuation of communities for the Birthdeath dataset over 5 time-stamps (T_1,T_2,T_3,T_4 and T_5). We represent each community with a different colour circle and the size is $\propto$ the number of nodes in that community. From Figure 6.3b, we can observe the death of clusters **C6** & **C7** at time-stamps T_3 and T_4 respectively. Similarly, we can see birth of clusters **NewC14** & **NewC15** at time-stamp T_4 and T_5 respectively. We also observe that cluster **C11** merges with **C5** at time-stamp T_2 and cluster **C8** splits into 2 clusters at time-stamp T_4. We can observe that cluster **C8** has expanded at time-stamp T_3. The cluster **C8** also contracts at time-stamp T_4 as it splits into 2 clusters. Cluster **C1** demonstrates continuation over time.

Whenever a new snapshot cannot fit well into the old segment (in terms of compression), it introduces a change-point and starts a new segment at that time-stamp. These change-points detect drastic discontinuities in time.

In [181], the authors provided a framework called FaceNet. In this technique, the authors deviated from the traditional two-step approach to analyze community evolutions. In the traditional approach, communities are first detected for each time-stamp and then compared to determine correspondences. In this approach, the authors used a framework called FaceNet for analyzing communities and their evolutions through a robust unified process. This framework allowed the discovery of communities and captured their evolution with temporal smoothness [67] given by the historic community structures. The authors formulated their problem in terms of maximum a posteriori (MAP) estimation, where the community structure was estimated both by the observed networked data and by the prior distribution given by historic community structures. Then an iterative algorithm was developed which guaranteed convergence to an optimal solution.

In both the aforementioned techniques namely Graphscope [186] and FaceNet [181], the primary focus was on detection of communities in time evolving graphs rather than tracking the evolution of communities. These techniques can identify significant events like birth, death and continuation of communities. However, it is difficult to detect significant events like merge and split using these techniques.

In [71], a framework was provided which used the clique percolation method (CPM) for tracking evolution of communities in successive time-stamps. However, this technique is prone to be affected by noisy events. For example, if very few nodes from a community at one time-stamp split then this method detects it as a split event whereas these nodes might have been removed due to random fluctuations in community detection technique. In [69], a model was introduced which tracked the progress of communities over time in a dynamic network such that each community is characterized by a series of significant evolutionary events. By only keeping track of significant evolutionary events they overcome noisy events. However, none of these methods [71, 69] provide a visualization tool to observe and get insight into the evolution of communities in dynamic networks.

Recently, a method which maps changes in networks using alluvial diagrams was proposed in [42]. The method uses bootstrap sampling accompanied by significance clustering in order to distinguish meaningful structural changes from random fluctuations. This technique is based on the principle introduced in [69]. However, this method [42] doesn't take into consideration the property of temporal smoothness [67]. The alluvial diagrams help to visualize events like

merge, split, continuation, expansion and shrinkage of communities. However, they cannot showcase the birth and death of individual communities separately. This is because in this method [42] a new community can only emerge at a given time-stamp t as a split of some previous community at time-stamp $t-1$ and it is difficult to detect the death/dissolution of a particular community at one given time-stamp t. Moreover, this visualization using surfaces uses a large portion of the screen and makes it difficult to track the evolution of individual communities when there is a series of merge and split events. Netgram allows to separately identify birth of a new community, highlight the death of a community and makes it easy to identify and track individual community evolution in presence of multiple merge and split events using a simple line-based visualization system. The tracking algorithm of Netgram is based on similar principles as that proposed in [69, 42, 184] i.e. trying to identify significant events and differentiate it from random noisy events. However, the main goal of Netgram is to come up with a simple line-based visualization tool which allows to track evolution of individual communities, capture and visualize significant events like birth, death, merge, split, continuation, shrinkage and growth of communities in dynamic time-evolving graphs. Netgram overcomes the aforementioned issues and provides a simple line-based visualization system for tracking evolution of communities over time.

6.2.3 Evolution of Communities

Significant events that happen during evolution of communities in a dynamic network can be defined as:

- *Birth*: The emergence of a new community C^t_{new} at time t comprising nodes which were previously unseen at time $t-1$ i.e. $C^t_{new} \cap C^{t-1}_i = \emptyset$ with all the clusters C^{t-1}_i in the set $\mathcal{C}^{t-1}$ at time $t-1$.
- *Death*: The disappearance of one or more communities at time t. It suggests that the majority of the nodes which had appeared as a community C^{t-1}_j at a previous time have disappeared now i.e. $\frac{|C^{t-1}_j \cap C^t_i|}{|C^{t-1}_j \cup C^t_i|} < \theta^{t-1}$ for all the communities C^{t-1}_i in the set $\mathcal{C}^{t-1}$ at time $t-1$.
- *Continuation*: The majority of the nodes in the community C^{t-1}_i remains intact for the next time t i.e $\frac{|C^{t-1}_i \cap C^t_j|}{|C^{t-1}_i \cup C^t_j|} \geq \theta^{t-1}$. Therefore, the community structure of C^{t-1}_i doesn't change much and continues to become community C^t_j.

- *Merging*: When the majority of the nodes from 2 or more communities at time $t-1$ for example C_j^{t-1} and C_k^{t-1} combine together to form one cluster C_i^t at time t. i.e. $\frac{|C_j^{t-1} \cap C_i^t|}{|C_j^{t-1} \cup C_i^t|} \geq \theta^{t-1}$ and $\frac{|C_k^{t-1} \cap C_i^t|}{|C_k^{t-1} \cup C_i^t|} \geq \theta^{t-1}$, all such conditions hold true. These clusters subsequently start to share a common time-line starting from time t.
- *Splitting*: When the majority of the nodes from a community C_i^{t-1} splits into 2 or more communities at time t say C_j^t and C_k^t i.e. $\frac{|C_i^{t-1} \cap C_j^t|}{|C_i^{t-1} \cup C_j^t|} \geq \theta^{t-1}$ and $\frac{|C_i^{t-1} \cap C_k^t|}{|C_i^{t-1} \cup C_k^t|} \geq \theta^{t-1}$, all such conditions hold true.
- *Growth*: When the number of nodes in the community C_i^{t-1} at time $t-1$ increase significantly for the corresponding community C_j^t at time t. For example, if the size of the community C_i^{t-1} increases by 20% at time t.
- *Shrinkage*: When the number of nodes in the community C_i^{t-1} at the time $t-1$ decreases significantly for the corresponding community C_j^t at time t. For example, if the size of the community C_i^{t-1} decreases by 20% at time t.

  The definition of majority or θ^{t-1} is provided in the next subsection.

Tracking Algorithm & Weighted Networks

In order to recognize these events we need a tracking algorithm that matches the communities found by the evolutionary clustering algorithms at each time step. Several such tracking algorithms have been developed including [69, 71, 180, 184]. In this work, we use a modified version of the tracking algorithm introduced in [184].

We first generate a weighted directed bipartite network W^t from the clusters at two consecutive time-stamps t and $t+1$. In case of T time-stamps, we generate a set $\mathcal{W} = \{W^1, \ldots, W^{\mathrm{T}-1}\}$ of weighted directed bipartite networks. Each bipartite network W^t creates a map between the set of clusters at time-stamp t i.e. $\mathcal{C}^t$ and time-stamp $t+1$ i.e. $\mathcal{C}^{t+1}$. Here $\mathcal{C}^t = \{C_1^t, \ldots, C_n^t\}$, where n represents total number of clusters at time-stamp t. This map corresponds to the edges of the network. The weight $w(\mathrm{V}^t(j,k))$ of an edge $\mathrm{V}^t(j,k)$ between two clusters C_j^t and C_k^{t+1} corresponds to the fraction of nodes in cluster C_j at time-stamp t and cluster C_k at time-stamp $t+1$ which are assigned to cluster C_k at time-stamp $t+1$ and is represented as:

$$w(\mathrm{V}^t(j,k)) = \frac{|C_j^t \cap C_k^{t+1}|}{|C_j^t \cup C_k^{t+1}|} \tag{6.7}$$

where the numerator is equal to the number of nodes of cluster C_j^t which are also part of C_k^{t+1} and $|C_j^t \cup C_k^{t+1}|$ represents the total number of distinct nodes in clusters C_j^t and C_k^{t+1}. This edge weight calculation scheme is different from the one proposed in [184] and gives importance to both the nodes in C_j^t at time-stamp t and the nodes in C_k^{t+1} at time-stamp $t+1$.

An edge exists between two clusters C_j^t and C_k^{t+1} only if $w(\mathrm{V}^t(j,k)) > 0$. We then create an empty list L^t. Here we keep the information about the maximum weighted outgoing edge from each cluster C_i^t at time-stamp t i.e. $\mathrm{argmax}_j w(\mathrm{V}^t(i,j))$ and the maximum weighted incoming edge for each cluster C_j^{t+1} at time-stamp $t+1$ i.e. $\mathrm{argmax}_i w(\mathrm{V}^t(i,j))$. The list L becomes:

$$L^t = [w(\mathrm{V}^t(i, \mathrm{argmax}_j w(\mathrm{V}^t(i,j)))), \ldots, w(\mathrm{V}^t(\mathrm{argmax}_i w(\mathrm{V}^t(i,j)), j)), \ldots],$$

where $i = 1, \ldots, |\mathcal{C}^t|$ and $j = 1, \ldots, |\mathcal{C}^{t+1}|$. Here i and j vary from 1 to the total number of clusters in the set $\mathcal{C}^t$ and $\mathcal{C}^{t+1}$ respectively. List L^t has all the maximum weighted edges from all $C_i^t \in \mathcal{C}^t$ and all the maximum weighted edges to $C_j \in \mathcal{C}^{t+1}$. We then define a threshold $\theta^t = L_k^t$ where $k = \mathrm{argmin}_l L_l^t$, $l = 1, \ldots, |L|$ i.e. it is the minimum weight in the list L^t of maximum weighted edges in the bipartite graph W^t. We use this minimum weight θ^t to define the concept of majority or only those edges are kept in W^t whose edge weight is greater than or equal to θ^t. This criterion prevents random noisy fluctuations from being detected as split or merge events, a concern faced by the tracking algorithm in [184].

If the number of edges going out of a node of W^t is greater than 1, it indicates the possibility of a split at time $t+1$, whereas if the number of edges entering a node of W^t at time $t+1$ is greater than 1 then it indicates the possibility of a merge. A split will only happen if the corresponding clusters say C_k^{t+1} and C_l^{t+1} have an edge weight $w(\mathrm{V}^t(j,k)) \geq \theta^t$ and $w(\mathrm{V}^t(j,l)) \geq \theta^t$ respectively coming from cluster C_j^t. Similarly, a merge will only happen if weight of the edges from clusters say C_i^t and C_j^t to cluster C_k^{t+1} are greater than θ^t.

We observe continuation of a community when majority of the nodes from a cluster C_j^t are part of the some cluster at time $t+1$ say cluster C_k^{t+1} i.e. $w(\mathrm{V}^t(j,k)) \geq \theta^t$. In order to tackle the death of one or more communities, we add a *Dump* cluster to the set of clusters $\mathcal{C}^t$ for each time-stamp t (except time 1 when we are first identifying the communities). The *Dump* cluster represents death of a community. Similarly, in order to handle birth of a new cluster from nodes which were previously unseen at time t, we add a cluster C_0^t to the set of clusters $\mathcal{C}^t$ for each time-stamp t except the first time-stamp. If more than one cluster is born at a given time-stamp t then we use identifiers C_0^t, C_{-1}^t, C_{-2}^t etc. for the newborn clusters. This allows to overcome the problem of detection of events like birth and death faced by the tracking algorithm in [184]. In [184],

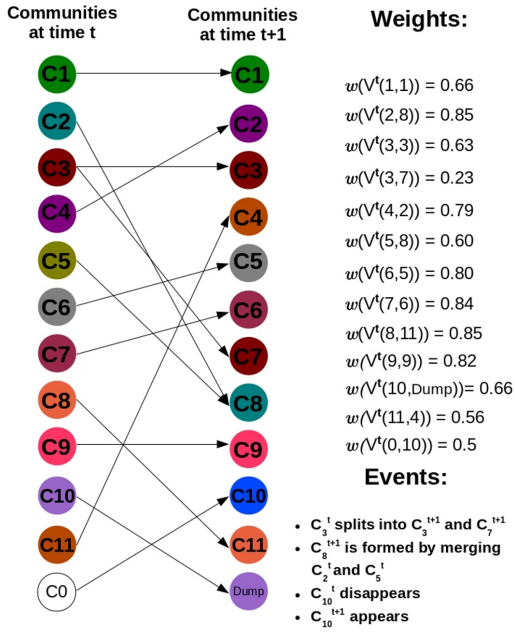


Figure 6.4: Weighted directed bipartite network W^t corresponding to 2 consecutive time-stamps for the synthetic Birthdeath dataset.

the authors used the same identifier for birth and death of communities which can lead to confusion when there is birth and death of a community at a given time-stamp. Moreover, this technique [184] cannot identify birth of multiple communities at a given time-stamp.

For the network W^t, if C_0^t is isolated then no new clusters were generated at time t and if C_0^t has an outgoing edge with weight $\geq \theta^t$ then a new community has emerged at time $t+1$. Similarly, if we have incoming edges to *Dump* at time $t+1$ in W^t then some clusters have disappeared. An advantage of having separate identifiers for tracking the birth and the death of communities is that it becomes easier to distinguish the two events when we are performing sql join operations [185] to generate the query database. Figure 6.4 gives an example of the mapping mechanism for 2 snapshots of Birthdeath dataset. This tracking procedure is summarized in Algorithm 18.

Algorithm 18: Community Tracking Algorithm

Data: The set of clusters $\mathcal{C}^t$ and $\mathcal{C}^{t+1}$ at a given time-stamp t and $t+1$ respectively.
Result: A weighted directed bipartite network W^t tracking the relationship between the clusters at time-stamp t and $t+1$ and the minimum weight θ^t that an edge should satisfy in W^t.

$L^t = []$
foreach $C_j^t \in \mathcal{C}^t$ **do**
  foreach $C_k^{t+1} \in \mathcal{C}^{t+1}$ **do**
    if *Nodes with labels* C_j^t *at* t *have the label* C_k^{t+1} *at* $t+1$ **then**
      Create an edge $\mathrm{V}^t(j,k)$ between C_j^t and C_k^{t+1}.
      $|C_j^t \cap C_k^{t+1}|$ = Number of nodes with labels C_j^t at t that have the label C_k^{t+1} at $t+1$.
      Weight of the edge $w(\mathrm{V}^t(j,k)) = \frac{|C_j^t \cap C_k^{t+1}|}{|C_j^t \cup C_k^{t+1}|}$.
      Add the weighted edge $V^t(j,k)$ to the graph W^t.
    end
  end
  $k = \mathrm{argmax}_k w(\mathrm{V}^t(j,k))$.
  Append $w(\mathrm{V}^t(j,k))$ to L^t. `/* Add the edge weight to the list` L^t. `*/`
end
foreach $C_k^{t+1} \in \mathcal{C}^{t+1}$ **do**
  foreach $C_j^t \in \mathcal{C}^t$ **do**
    if *Nodes with labels* C_k^{t+1} *at time* $t+1$ *have the label* C_j^t *at time* t **then**
      Create and edge $\mathrm{V}^t(j,k)$ between C_j^t and C_k^{t+1}.
      $|C_j^t \cap C_k^{t+1}|$ = Number of nodes with labels C_j^t at t that have the label C_k^{t+1} at $t+1$.
      Weight of the edge $w(\mathrm{V}^t(j,k)) = \frac{|C_j^t \cap C_k^{t+1}|}{|C_j^t \cup C_k^{t+1}|}$.
      Add the weighted edge $V^t(j,k)$ to the graph W^t.
    end
  end
  $j = \mathrm{argmax}_j w(\mathrm{V}^t(j,k))$.
  if $w(V^t(j,k))$ *not in* L^t **then**
    Append $w(\mathrm{V}^t(j,k))$ to L^t. `/* Add the edge weight to the list` L^t. `*/`
  end
end
$\theta^t = L_k^t$ where $k = \mathrm{argmin}_l L_l^t$, $l = 1, \ldots, |L|$.
`/* Find the minimum weight of all the weights added to the list` L^t. `*/`
foreach *edge* $V^t(j,k) \in W^t$ **do**
  if $w(V^t(j,k)) < \theta^t$ **then**
    Remove edge $\mathrm{V}^t(j,k)$ from W^t.
    `/* Only keep significant edges in the weighted bipartite network` W^t. `*/`
  end
end

Visualizing Weighted Bipartite Networks

After obtaining the set $\mathcal{W}$, we visualize the weighted bipartite networks W^t for each time-stamp t. The steps involved in the visualization include using each bipartite network W^t as an adjacency matrix A^t and plotting the adjacency matrix as an image.

An edge in the adjacency matrix $A^t(i,j)$ indicates the connection between cluster C_i^t at time t and cluster C_j^{t+1} at time $t+1$ and the weight of the edge is $A^t(i,j) = w(\mathrm{V}^t(i,j))$ obtained from W^t. For the purpose of visualization we remove all the edges directed to the *Dump* cluster. Let us say that cluster C_i^t disappeared at time $t+1$. Then, sum of the weight of all the edges from community C_i^t to all the clusters at time-stamp $t+1$ is 0. Hence, it becomes easier to identify communities which have disappeared. In the plot of A^t, we replace cluster C_i^t with *Dump*. In case of birth of one or more new communities we look up the entries corresponding to $A^t(0,i)$, $A^t(-1,j)$ etc. in the adjacency matrix at time-stamp $t+1$. In the plot of A^t, we indicate these newborn clusters with the prefix 'NewC'.

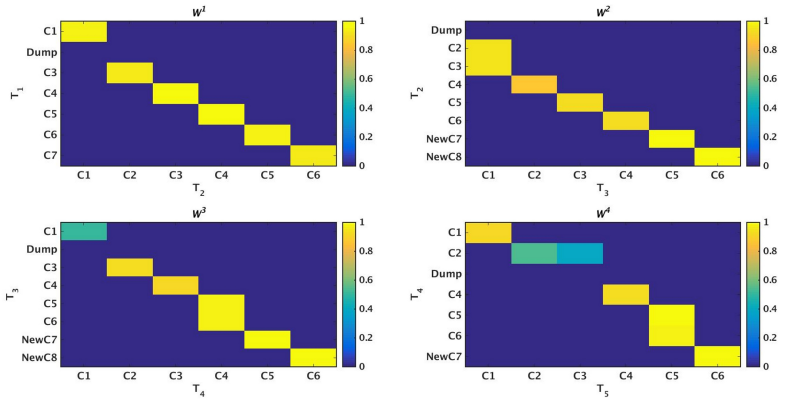


Figure 6.5: Visualization of the weighted networks W^t mapping the evolution of communities over 5 time-stamps. W^t tracks the evolution of a cluster between two successive time-stamps. We can observe that at each time-stamp T_i there is death of one community. We can also detect the birth of one or more communities at each time-stamp T_i, $i > 2$. The x-axis and y-axis represent the set of clusters at two consecutive time-stamps. The colors represent the weight of the edges for W^t.

An example of this visualization procedure is shown in Figure 6.5 in the case of a synthetic Hide dataset. This Hide dataset comprises 5 snapshots where one community disappears at each time-stamp t and one or more communities appear at each time-stamp $t > 1$. We use the evolutionary community detection algorithm introduced in [70] (namely Evolutionary Spectral Clustering) to illustrate that the proposed visualization also works for another evolutionary clustering algorithm.

T1	**T2**	**T3**	**T4**	**T5**
C_1	C_7	C_3	C_1	C_3
C_2	C_1	C_5	C_7	C_5
C_2	C_2	C_5	C_7	C_5
C_3	C_4	C_2	C_2	C_1
C_3	C_4	C_2	C_3	C_1
C_3	C_3	C_8	C_6	C_2
C_3	C_3	C_1	C_9	C_7
C_3	C_3	C_1	C_9	C_8
C_4	C_6	C_9	C_4	C_6
C_4	C_6	C_9	C_{10}	C_4
C_4	C_6	C_9	C_{10}	C_{11}
C_5	C_8	C_6	C_1	C_3
C_6	C_8	C_6	C_1	C_3
C_7	C_5	C_4	C_5	C_9
C_7	C_5	C_7	C_8	C_{10}

Table 6.3: Tracking the 7 communities detected by the OSLOM method [40] at time-stamp T_1 for Mergesplit dataset. This tracking information is stored in database $\mathcal{D}$ which is depicted here.

6.2.4 Netgram Tool

Once we have obtained the set $\mathcal{W}$, we store the connections between nodes of the bipartite network W^t at time t in tabular format. For the sake of simplicity, we just keep the information about the source and the sink of the edge i.e. for an edge $V^t(j,k)$, we keep (j,k) in table P^t for time t. This results in a set of tables $\mathcal{P} = \{P^1, \ldots, P^{\mathrm{T}-1}\}$ for T time-stamps.

We then construct a query database $\mathcal{D}$ by performing an outer-join sql operation [185, 187] between table P^1 and P^2 using the unique identifiers in 2nd column of P^1 and 1st column of P^2 as keys. We then repeat the process with query database $\mathcal{D}$ and succeeding tables in set $\mathcal{P}$ i.e. $P^3, \ldots, P^{T-1}$ using the unique identifiers of last column of database $\mathcal{D}$ and 1st column of table P^i $(i > 2)$ as keys on which the outer-join operation [185] is performed. By keeping separate unique identifiers for a dead cluster and a newborn community, it becomes easier to track birth and death of communities over time.

An example of the query database $\mathcal{D}$ obtained as a result of this process is shown in Table 6.3 for a synthetic Mergesplit dataset using the OSLOM [40] method. Since the OSLOM method is a hierarchical community detection algorithm, we only track large size communities at coarser levels of hierarchy for the 5 snapshots of the Mergesplit dataset.

We use a simple line-based tracking mechanism to visualize the query database $\mathcal{D}$. This line-based tracking is inspired by dendograms [188] which are used for visualization of layers of hierarchy in hierarchical clustering algorithms [170, 21]. Similarly the concept can be applied in case of dynamic networks where the

layers represent the individual time-stamps for the evolving network.

While building the Netgram tool, we considered the following design principles:

- Each community is represented by a circle whose size is proportional to the number of nodes in that community at a given time-stamp t.
- The evolution of communities between 2 time-stamps is represented by a dashed line.
- Lines follow a straight path if there is no merge or split event during the entire course of evolution for a given community.
- If a part of a community say C_i^t merges into another community say C_j^{t+1} i.e. a merge event has occurred at time $t+1$ then a dashed line is drawn from C_i^t at time t to the community it merges at time $t+1$.
- If a part of a community say C_i^t splits into another community say C_j^{t+1} i.e. a split event has occurred at time t then a dashed line is drawn from C_i^t at time t to the community it merges at time $t+1$.

We also take into account the following aesthetic conditions as suggested in [189] when designing a visualization tool:

- Minimize line cross-overs.
- Minimize screen space.

These layout guidelines lead to a combinatorial problem with respect to the aesthetic conditions under the constraints of the design principles [189]. A similar problem for storyline visualization for streaming data was proposed in a constrained optimization framework in [189, 190]. In [189, 190] the authors try to visualize individuals (or groups) in a story. However, there is no event like merging or splitting of groups rather the groups come close together or move far from each other. Moreover, we can display additional information like size/density of communities at different time-stamps from the size of the circle representing those communities at these time-stamps. Figure 6.6 visualizes the evolution of communities over 5 time-stamps for the synthetic Mergesplit dataset using the community information obtained from OSLOM [40] method along with the original order of clusters in $\mathcal{C}^1$ by providing this information to Algorithm 19.

The evolutionary clustering algorithm initially provides the community labels for all the nodes at a given time-stamp t. Using the query database $\mathcal{D}$ and the initial order of the partitions in $\mathcal{C}^1$ i.e. $\mathcal{O} = \{1, 2, \ldots, n\}$ where n represents the total number of communities in the 1st time-stamp, we propose a simple

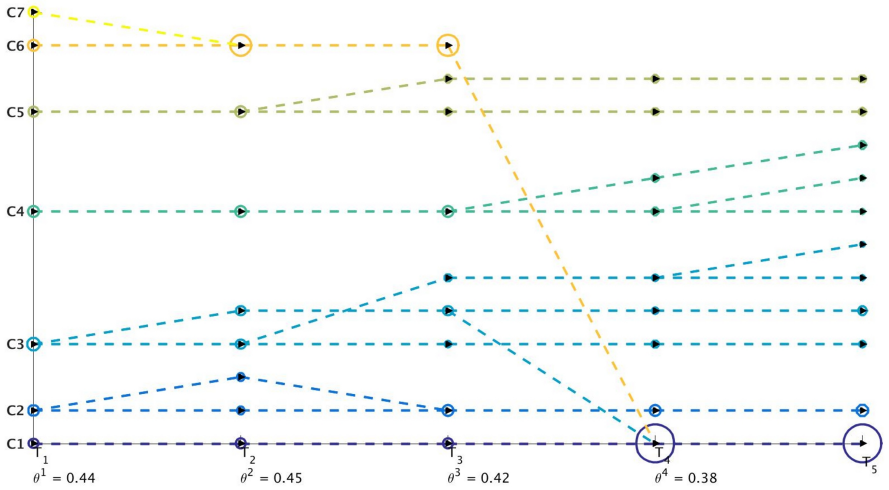


Figure 6.6: Visualization of evolution of communities obtained from OSLOM [40] method for Mergesplit dataset using Netgram. There are 11 cross-overs in this figure which is generated by passing the original order of clusters i.e $\mathcal{O} = \{1, 2, 3, 4, 5, 6, 7\}$ to Algorithm 19.

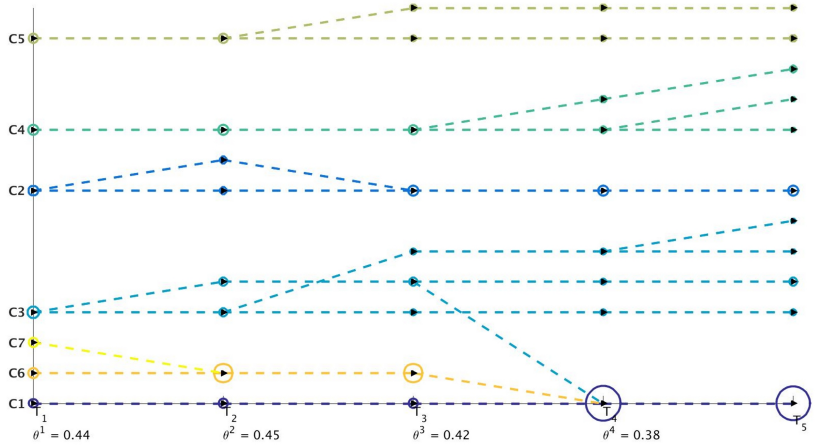


Figure 6.7: Visualization of evolution of communities for Mergesplit dataset using the refined order of partition $\mathcal{O}$ obtained from Algorithm 20 and then passed to Algorithm 19. The revised ordering of partitions results in just 2 cross-overs as depicted in this figure.

Algorithm 19 that satisfies all the design principles and the aesthetic criteria except the minimization of line cross-overs. By using line based visualization instead of surface based visualization we minimize the screen space.

Algorithm 19: Netgram Visualization Layout

Data: Query Database $\mathcal{D}$, total number of time-stamps T and the order of the clusters i.e. $\mathcal{O}$.
Result: Line-based visualization satisfying the design principles.

```
1  i = 1, y = 0, n = |O|, M = {M_2, ..., M_T}
   /* M is a memory-map which keeps information about the communities present at all the
      time-stamps t from t = 2, ..., T. Initialize each of M_t as an empty list.        */
2  while i ≤ n do
3      cid = O_i. /* Select the i^th id from O to represent which community will be visualized
          at the i^th iteration.                                                          */
4      j = 2 & draw a circle at co-ordinates (j − 1, y) representing community C_cid ∝ its size.
5      List_cid = {C_cid}. /* Add C_cid to a list.                                       */
6      while j ≤ T do
7          while List_cid ≠ [] do
8              C_cid = Pop first element from List_cid.
9              R = Result of the query to D finding the rows corresponding to C_cid at time j − 1.
10             SubC_cid = Unique list of communities connected to C_cid at time j obtained from R.
11             foreach c_k ∈ SubC_cid & c_k ∉ M_j do
12                 if c_k = Dump then
13                     Stop the time-line for C_cid. /* Handles the death event.        */
14                 end
15                 else
16                     Draw a circle at (j, y) representing community c_k ∝ its size.
17                     Add c_k to M_j.
18                     Draw a dashed line from C_cid at time j − 1 to c_k at time j.
19                     y := y + 1.
20                 end
21             end
               /* Handles the growth, shrinkage, continuation and split events.       */
22             foreach c_k ∈ SubC_cid & c_k ∈ M_j do
23                 Draw a dashed line from C_cid at time j − 1 to an existing community c_k at
                   time j.
24             end
               /* Handles the merge event.                                            */
25         end
26         j := j + 1 & List_cid = SubC_cid.
27     end
28     List_cid = {} & i := i + 1 & y := y + 1.
29 end
30 For new communities follow the same procedure as above using the memory-map M generated
      from the existing communities and query database D. /* Handles birth event.    */
```

We now explain the concept of a line cross-over. A line cross-over generally occurs in case of a split or merge event. For example, in case of Figure 6.6, community **C6** merges into community **C1** at time-stamp T_4. However, the line showing the merge event crosses over the time-line of communities **C5, C4, C3, C2** and its branches i.e. 8 lines in total. Similarly, community **C3** has one cross-over at time-stamp T_3 during a split event and 2 cross-overs at time-stamp T_4 due to a merge event with community **C1**. Thus, in total there are 11 cross-overs using the initial order of clusters i.e. $\mathcal{O} = \{1, 2, 3, 4, 5, 6, 7\}$.

However, if we place **C6** next to cluster **C1** and community **C7** next to **C6** and maintain the order of the remaining communities i.e. $\mathcal{O} = \{1, 6, 7, 2, 3, 4, 5\}$, we can already reduce the number of line cross-overs to 3. We provide a greedy solution in Algorithm 20 to sequence the order of clusters/partitions in $\mathcal{C}^1$ such that there are fewer line cross-overs in comparison to the initial order of the clusters i.e. $\mathcal{O} = \{1, \ldots, n\}$ provided by the evolutionary community detection technique.

Algorithm 20: Greedy Solution to Handle Cross-Overs

Data: Original order of clusters $\mathcal{O}$, total number of time-stamps T and query database $\mathcal{D}$.
Result: New order of partitions $\mathcal{O}$

$i = 2$, $\mathcal{S} = \{\}$, $M = \{\}$ and $Queue = L = []$. /* Initialize an empty map M which will be used to map for each cluster $c_j \in \mathcal{C}^1$ the sequence of clusters which had interaction with c_j at different time-stamps. */
while $i \leq T$ **do**
    R = Result of the query to $\mathcal{D}$ to find which community or communities $c_j \in \mathcal{C}^1$ at time $i-1$ has experienced a merge or a split event from which community or communities $c_k \in \mathcal{C}^1$ at time i.
    Generate tuples (c_j, c_k, i) from R and append to $Queue$.
end
while $Queue \neq []$ **do**
    P = Pop first tuple out of queue. /* This allows us to handle communities which merged or splitted at earlier time-stamps first. */
    Obtain c_j and c_k and corresponding ids j, k from tuple P.
    if $k \notin M(j)$ **then**
        Append k to the sequence of j.
    end
end
/* M maps for each cluster $c_j \in \mathcal{C}^1$ the clusters $c_k \in \mathcal{C}^1$ with which it interacted ordering them based on the time-stamp of interaction i.e. the cluster with which c_j interacted first is placed first in the $M(j)$ and the cluster with which it interacted last is placed last. */
foreach $j \in \mathcal{O}$ **do**
    Insert j and all elements of $M(j)$ to L (j appearing first). /* $j = 1, \ldots, n$ */
end
foreach $j \in L$ **do**
    if $j \notin \mathcal{S}$ **then**
        Append j to $\mathcal{S}$.
    end
end
/* $\mathcal{S}$ is the greedy solution to handle cross-overs. It uses the simple principle to keep together those clusters which had interacted earlier in time in order to reduce cross-overs. */
$\mathcal{O} = \mathcal{S}$.

Figure 6.7 illustrates the effect of applying Algorithm 20 on the initial order of partitions in $\mathcal{C}^1$ obtained from OSLOM method [40] for Mergesplit dataset and providing this new order $\mathcal{O}$ to Algorithm 19 to perform visualization using Netgram. Algorithm 21 summarizes the steps undertaken by the Netgram tool for visualization of evolution of communities.

Algorithm 21: Steps undertaken by the Netgram tool for visualization.

Data: Node identifiers and community affiliation for nodes appearing in a dynamic network. The community affiliation can be obtained from any evolutionary clustering algorithm.

Result: A visualization of the evolution of communities over time.

1 Apply the modified version of Tracking Algorithm 18.
2 Generate a set $\mathcal{W}$ comprising weighted directed bipartite networks W^t capturing the evolution of communities between 2 successive time-stamps.
3 Visualize these weighted bipartite networks (W^t).
4 Store these weighted bipartite networks (W^t) into tables P^t.
5 Perform sql outer-join operation [185, 187] on these tables to get a query database $\mathcal{D}$.
6 Re-order the initial partition $\mathcal{C}^1$ provided by evolutionary community detection algorithm to have few cross-overs using Algorithm 20.
7 Visualize $\mathcal{D}$ satisfying all design principles and greedily fulfilling aesthetic guidelines using Algorithm 19.

6.2.5 Experiments

We first provide a brief description of the datasets used in this work. In the previous sections, we have seen the use of several synthetic datasets including:

- **Birthdeath** - A dataset comprising 5 dynamic networks. There are 13 communities in the network at time-stamp t_1. At every time-stamp t_i, $i > 2$ there is death of one community and at each time-stamp $t_i > 3$ there is birth of one community. The number of nodes in the networks decrease from $1,000$ to 886 over time. We illustrate the evolution of communities for this dataset using the MKSC algorithm [72, 184] in the Figure 6.3.
- **Hide** - A dataset comprising of 5 dynamic networks. There are 7 communities in the network at time t_1. At each time-stamp t_i, one community disappears as was shown in the visualization of the weighted bipartite networks in Figure 6.5.
- **Mergesplit** - A dataset comprising 5 dynamic networks. There are 7 communities in the network at time t_1 and $1,000$ nodes in each of the 5 snapshots of this dataset. At each time-stamp t_i, there is 1 merge and 2 split events. A visualization of the evolution of the communities obtained by OSLOM [40] method for this dataset was provided in Figure 6.6.

These networks were generated using the software available at `https://github.com/derekgreene/dynamic-community`. We also experimented on real-life

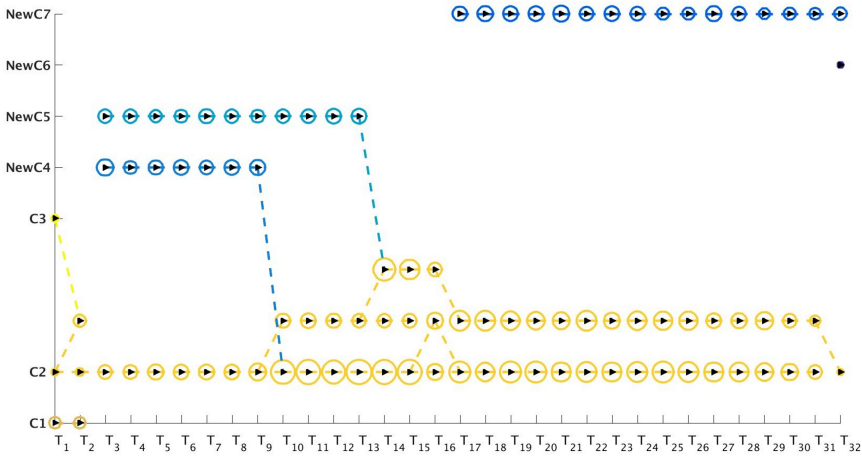


Figure 6.8: Visualization of evolution of communities generated by Evolutionary k-means [70] method using the Netgram tool for the Reality dataset. We mostly observe continuation of communities. However, communities **C1** and **C3** both disappear at time-stamp T_2 and new communities **NewC4** and **NewC5** appear at time-stamp T_3. By time-stamp T_{14} all communities merge into branches of community **C2**. New communities **NewC6** and **NewC7** appear at time-stamps T_{18} & T_{32} respectively.

datasets and a synthetic large scale dataset (to show the scalability of the Netgram tool) which are described below:

- **Reality** - This dataset monitors the cellphone activity of people from 2 different labs in MIT [191]. This dataset consists of networks where the node represents a user whose cellphone periodically scans for other cellphones via Bluetooth. The weight of the edge between two nodes is equal to the number of intervals for which the 2 users are in close proximity to each other. Each snapshot corresponds to a weighted network which records the activities of the users over a period of 1 week. There is a total 32 meaningful snapshots and total number of users monitored over this time-span is 94. However, not all users are present in each weighted network. The smallest network comprised of 21 people and largest network had 88 people.

- **NIPS** - This dataset consists of information about $1,500$ papers published in the Advances in Neural Information Processing Systems (NIPS) conference starting from 1987 to 1998. The dataset is part of the Bag of

Words Dataset [73] from the UCI repository `http://archive.ics.uci.edu/ml/`. From this dataset, we first separate out the papers published in each year starting from 1987 till 1998. We then create a TF-IDF model [192] using which we remove out irrelevant words from the documents. We then create a word-word graph for each time-stamp where the weight of the edges in the network is proportional to occurrence of 2 words together in all the documents for that year.

- **Weather** - We collected weather information for about 9 months from `http://www.wunderground.com/` for 23 European cities. The time period over which this data spanned was from January 2012 to October 2012. For each city for each day we collected information about 20 attributes. The goal was to cluster these cities using weekly information about these cities i.e. one snapshot consists of 23 cities and 140 variables where 20 variables from each day are concatenated together. Thus in total we have 40 snapshots corresponding to 40 weeks.
- **Big** - This dataset consists of 5 networks where the network at the first time-stamp has 1 million nodes. There are 10 communities in the network at time t_1. The number of nodes decreases over time. The dataset exhibits several significant events like merge, split and death of communities. It was also generated from the software provided in [69].

The Netgram toolkit is most suitable for visualization of a small number of large sized clusters. This is because in case of a large number of clusters the palette for plotting will become too cluttered and it becomes difficult to keep track of the evolution of individual communities. Thus, in case of a hierarchical evolutionary clustering algorithm like OSLOM [40] or Louvain [36] method, Netgram is most suitable for visualization of giant-connected components at coarser levels of hierarchy.

Figures 6.8 and 6.9 depict the evolution of communities for the Reality and the Big dataset respectively. In case of the Reality dataset, we use the Evolutionary k-Means technique introduced in [70] to obtain the community affiliation for all the nodes at different time-stamps. In case of the Big dataset, we use the Louvain method [36] to generate the community memberships for all the nodes in the network over different periods of time. We use the Louvain method as it can easily scale to 1 million nodes for community detection which is otherwise a difficult task for evolutionary clustering algorithms.

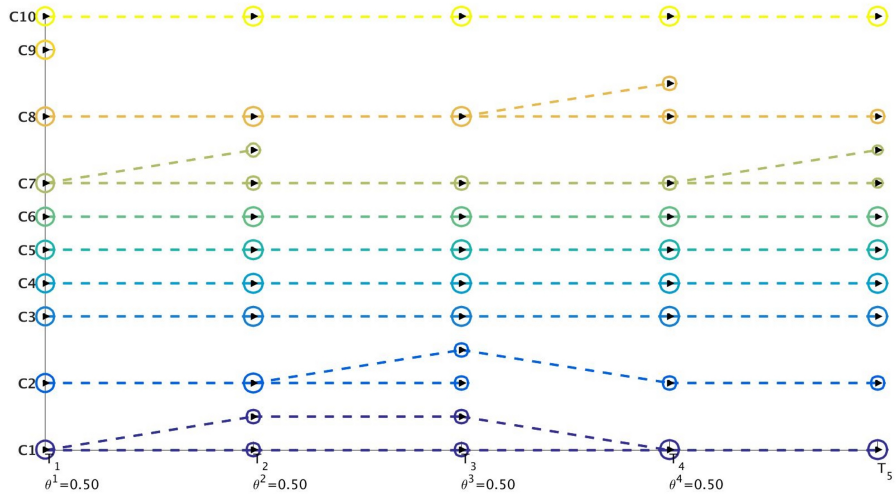


Figure 6.9: Visualization of evolution of communities generated by the Louvain [36] method using the Netgram tool for the Big dataset. We can observe significant events like merge, split, death and shrinkage of communities. For example, we can observe the split of communities **C7** and **C8** at time-stamp T_2 and T_3 respectively. Similarly, we can observe a merge event for cluster **C1** at time T_4. We observe a general pattern of shrinkage for most of the communities in this dataset.

NIPS Results

We obtain the communities from the word-word graph for the NIPS dataset using the MKSC [72] algorithm. The MKSC technique identified 5 communities at each time-stamp for this dataset. However, as we observe from Figure 6.10 there was birth of several new communities that was detected by the MKSC algorithm. We observe the appearance of disciplines like Speech Recognition and Computer Vision as distinguishable and distinct communities (in comparison to Supervised Learning Techniques) in 1991 and 1992 respectively. Figure 6.10 illustrates that the Netgram tool also detected the inception of NIPS Workshops in the year 1993. By the year 1998, we have specialized disciplines like Neural Networks, Robotics, Kernel Methods and Bayesian Methods which are combinations of Supervised Learning Techniques and Unsupervised Learning Techniques as observed from Figure 6.10.

Since we use a TF-IDF model [192] to obtain the word-word graph on which we perform the community detection for the NIPS dataset, we can identify the

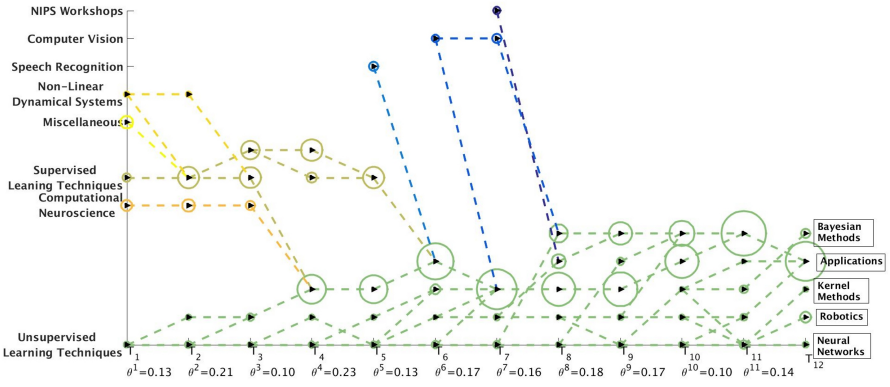


Figure 6.10: Visualization of evolution of communities for the NIPS dataset by Netgram toolkit.

top words (based on TF-IDF [192]) in these communities. In Figures 6.11 and 6.12, we showcase these top words corresponding to the communities detected by MKSC algorithm [72, 184].

Weather

For the weather dataset for each day we gathered information about attributes like maximum, minimum and mean temperature, dew point, humidity, sea level pressure, visibility, wind speed respectively and precipitation and wind direction for each city. The 23 European cities for which the data was gathered included Amsterdam, Antwerpen, Athens, Berlin, Brussels, Dortmund, Dublin, Eindhoven, Frankfurt, Groningen, Hamburg, Liege, Lisbon, London, Madrid, Milan, Nantes, Paris, Prague, Rome, Toulouse, Vienna and Zurich. We run the MKSC algorithm to obtain the communities for different time-stamps. The MKSC technique takes into account temporal smoothness [67] using the memory. Figure 6.13 depicts the result obtained for the weather dataset using MKSC algorithm.

Initially community **C1** consists of all the 23 European cities. However, after beginning of April (T_{13}) cluster **C1** splits into 2 or more prominent communities for all later time-stamps. This suggests that winter pattern in most of the European cities are nearly same. During the first week of April (T_{13}) the smaller cluster comprises cities like Milan, Rome, Nantes, Paris, Prague, Toulouse and Vienna which indicated that outbreak of spring in these cities are similar to each other and different from other European cities. During mid May (T_{19}), we



(a) Top words for Unsupervised Learning Techniques cluster



(b) Top words for Supervised Learning Techniques cluster



(c) Top words for Computational Neuroscience cluster



(d) Top words for Non-Linear Dynamics cluster



(e) Top words for Speech Recognition cluster



(f) Top words for Computer Vision cluster.

Figure 6.11: Word clouds which correspond to the top words in the communities detected by MKSC algorithm [72, 184] over the first few time-stamps for the NIPS dataset.

observe 3 communities where one cluster was just the city of Dublin, the other 2 clusters consisted of Amsterdam, Antwerpen, Brussels, Berlin, Dortmund, Eindhoven, Frankfurt, Groningen, Hamburg, Liege, London, Nantes, Paris and Athens, Lisbon, Madrid, Milan, Prague, Rome, Toulouse, Vienna, Zurich respectively. This clustering information clearly distinguishes the weather pattern of western European cities from cities of southern Europe (except Prague).

The Netgram tool is available for usage at `http://www.esat.kuleuven.be/stadius/ADB/mall/downloads/Netgram_Tool.zip`.

6.2.6 Conclusion

In this paper we proposed a visualization toolkit Netgram which can be used to depict the evolution of communities in dynamic networks over time. Netgram was used to illustrate the occurrence of significant events like birth, death, merge, split, expansion, shrinkage and continuation of communities over time. Netgram can be used as a post-processing step to any evolutionary clustering



(a) Top words for NIPS Workshops cluster. This cluster is dominated by names of authors who were referred multiple times in several workshop papers.



(b) Top words for cluster corresponding to Neural Networks



(c) Top words for Robotics cluster



(d) Top words for cluster corresponding to Kernel Methods.



(e) Top words for the cluster corresponding to Applications.



(f) Top words for Bayesian Methods cluster

Figure 6.12: Word clouds which correspond to the top words in the communities detected by MKSC algorithm [72, 184] over the last few time-stamps for the NIPS dataset.

algorithm. Netgram provides a simple line-based visualization tool for tracking the evolution of communities for various datasets. The tracking of the evolution of communities was performed in such a way that there were only a few line cross-overs and efficient screen space usage (since we used dashed lines). We proposed a greedy solution to have fewer line cross-overs in comparison to the plot obtained by using the original order or partitions ($\mathcal{O}$) as provided by the evolutionary clustering algorithm.

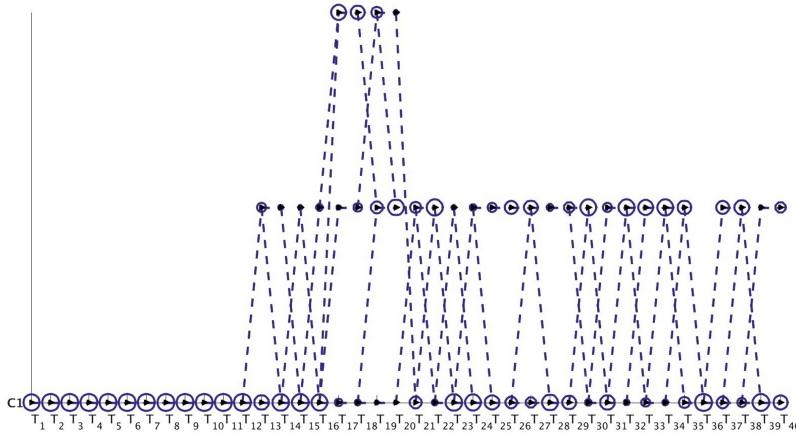


Figure 6.13: Visualization of evolution of clusters obtained using MKSC [72] algorithm for the weather dataset by Netgram toolkit.

Chapter 7

Conclusions & Future Work

7.1 General Conclusions

In this thesis we have explored the role of sparsity in large scale kernel models. The two primary goals have been to observe the extent of information required to obtain good generalization power for supervised and unsupervised predictive models using the LSSVM primal-dual optimization framework and the scalability of these models for large scale datasets.

In Chapter 2, we first proposed two techniques namely sparse primal fixed-size LSSVM (SPFS-LSSVM) and sparse subsampled dual LSSVM (SSD-LSSVM) that resulted in very sparse LSSVM models for large scale classification and regression problems. We overcame the problem of selecting the smallest cardinality M for the prototype vector set faced by PFS-LSSVM using a convex re-weighted L_1 penalization scheme. We showed that the SSD-LSSVM technique has the least computation time in comparison to state-of-the art approaches like Keerthi's method, ν-SVM and C-SVM while resulting in very sparse solutions and good performance. We investigated the trade-off between amount of sparsity generated and the predictive performance by the proposed very sparse LSSVM (VS-LSSVM) methods. We also proposed a Window reduced FS-LSSVM technique where we removed points which are closest and farthest from the decision boundaries to better determine the underlying structure of the model by facilitating the selection of more appropriate prototype vectors (PV). We showed the scalability of these methods to large scale classification (Forest Cover ($500,000$ data points)) and regression (Year Prediction ($500,000$

data points)) tasks on a single desktop computer.

One of the important aspects of kernel based methods is to build a good predictive model on a representative subset of data. A model can have good generalization w.r.t. a learning task if the subset on which it is built captures the inherent structure in the data. In Chapter 3, we proposed the Fast and Unique representative subset (FURS) [123] selection procedure. FURS greedily selected nodes with high degree centrality from different dense regions of the graph thereby spanning *most or all* the communities in the large scale network. It used the concept of node activation, deactivation and re-activation while retaining the topology of the graph. FURS was compared with state-of-the art sampling techniques like SlashBurn, Forest-Fire, Metropolis and Snowball sampling methodologies for various quality criteria like degree distribution, clustering co-efficients, variation of information (VI) and fractions of communities covered. We also proposed a method to obtain representative subsets for big data which can be used in combination with model based learning techniques. The idea in this work was to convert the big data into a k-NN graph using a distributed framework and then select the required subset using the FURS [123] algorithm. The selected subset retained inherent clustering information present in the data. We depicted the effectiveness of the proposed approach for several large scale real-world datasets like KDDCupBio and Skin datasets.

In Chapter 4 we illustrated that the Kernel Spectral Clustering (KSC) method can be used as an efficient tool for community detection in case of big data networks. We overcame the assumption that the entire network can still fit in main memory, an issue faced by most of the state-of-the-art community detection techniques like Louvain, Infomap and OSLOM methods. The advantage of the KSC method is that it builds the model on a small representative subset of the data employing a primal-dual optimization setting in a learning framework. As a result of which KSC has a very powerful out-of-sample extensions property which allows community affiliation for previously unseen nodes. We showed that an important condition was for the model to adhere to the memory restrictions. Therefore, there was a need to select a representative subgraph of the big data network in a computationally cost-effective way. We used the FURS [123] selection technique for this purpose.

We also proposed two model selection techniques in Chapter 4 to obtain the optimal model parameters i.e. k or the number of clusters in the large scale complex network. The first criterion called Balanced Angular Fitting (BAF) works with the codebook $\mathcal{C}B$ and the projections of the validation set. It worked on the principle of angular distance of the validation projections from the mean projection of the cluster to which they belong. The metric was both memory and computationally effective and could easily scale to big data networks. The second criterion made the KSC method free of parameters. It exploited the

projections of the validation nodes in the eigenspace by creating an affinity matrix which had block-diagonal structure. We then used the concept of entropy and balance to come up with an F-measure which identified the number of such block-diagonals resulting in the optimal k for the large scale network. We showed the scalability of the proposed approaches to large scale networks like Youtube, RoadCA and Livejournal networks ($\approx 10^7$ nodes and 10^8 edges).

Finally, in Chapter 4, we devised a new multilevel hierarchical kernel spectral clustering (MH-KSC) technique. This approach overcame the problem of resolution limit suffered by most of the state-of-the-art techniques like Louvain and Infomap methods by identifying good quality clusters for both coarse as well as fine levels of granularity. MH-KSC method exploited the structure of the projections in the eigenspace such that the validation set provided a set ($\mathcal{T}$) of increasing distance thresholds. These distance thresholds are used along with iterative affinity matrices generated from the projections to obtain an agglomerative hierarchical clustering organization in a bottom-up fashion. We demonstrated that the feasibility of the proposed method to large scale real-life networks like Imdb `http://www.imdb.com/` and Youtube networks.

In Chapter 5, we first proposed a novel method to identify an ideal range for the number of clusters (k) at different levels of hierarchy for a given dataset. An advantage of the proposed approach is that it provided these intervals before the application of a clustering algorithm. In the proposed method, we used the Gershgorin Circle theorem on the normalized Laplacian matrix to obtain upper bounds on the eigenvalues without performing the computationally expensive eigen-decomposition step. Using the piece-wise constant nature of these upper bounds we proposed an algorithm to obtain intervals for ideal value of k at each level of hierarchy. These intervals were then provided to clustering algorithms which used the normalized Laplacian matrix. We demonstrated the effectiveness of these intervals to determine k at each level of hierarchy in combination with a hierarchical kernel spectral clustering (HKSC) technique for several synthetic and real-world datasets. We then extended the MH-KSC from network clustering to data clustering and image segmentation. In case of networks, the normalized linear kernel was used to obtain the kernel matrix. The normalized linear kernel is independent of any tuning parameter. However, in case of datasets and images, we used the radial basis kernel where we are required to estimate the optimal bandwidth parameter σ. So, in the proposed approach we first estimated the optimal bandwidth parameter σ during validation stage using the BAF criterion. This optimal σ was used to generate a KSC clustering model which provided the out-of-sample eigen-projections. We then follow the steps used in MH-KSC i.e. utilize the eigen-projections to obtain a set of distance thresholds ($\mathcal{T}$) and iteratively built up the affinity matrices to obtain a hierarchical clustering organization in a bottom-up fashion. We compared

and illustrated the effectiveness of AH-KSC method over several agglomerative hierarchical clustering techniques like single link, complete link, median link and weighted link hierarchical clustering techniques on several datasets and real-world images.

In this chapter, we also proposed several methods for obtaining sparse reductions to KSC model. We used penalization terms like Group Lasso, convex relaxation of L_0-norm and combination of $L_1 + L_0$-norm penalties for reconstructing the original model with minimum error using a reduced set of prototype vectors (PV). The methodologies were aimed to generate sparse and simpler KSC models with efficient out-of-sample extension complexity. We observed that the Group Lasso penalty resulted in the sparsest models with good clustering generalization in the most computationally cost-effective way followed by the reduced models obtained via the convex relaxation to L_0-norm penalty. We demonstrated the proposed methods on several datasets and real-life images.

In Chapter 6 we focused more on the application aspect of the supervised and unsupervised kernel based models. We first proposed a primal-dual framework for feature selection in classification problems using a combination of the convex relaxation of the L_0-norm penalty and the L_2-norm penalty in the LSSVM setting. The proposed method resulted in iteratively solving a system of linear equations. The L_2-norm penalty helped in robust selection of variables whereas the L_0-norm penalty reduced the unwanted feature co-efficients to 0. We showcased the efficiency of the proposed approach on several real-life classification problems including Leukemia and Colon cancer problems.

Finally, in chapter 6, a new visualization toolkit named Netgram was proposed whose purpose was to visualize evolving communities in dynamic networks over time. Using Netgram we illustrated the occurrence of significant events like birth, death, merge, split, growth, shrinkage and continuation of communities over time. An advantage of the Netgram toolkit was that it could be used as a post-processing step to any evolutionary clustering method. Netgram provided a simple visualization for tracking the evolution of communities for various datasets over time using line-structures. The tracking of the evolution was performed such that we adhere to certain design principles and aesthetic conditions like minimizing the screen space and minimizing the line cross-overs [189].

Thus, in summary we proposed several supervised and unsupervised kernel based methods for large scale machine learning tasks like classification, regression, community detection and clustering. We illustrated that sparsity is an integral part when designing the model for large scale datasets and is obtained either via the representative subset selection or through simplifying the model complexity by addition of the regularization terms to the objective function. The design of

the models proposed in this thesis were such that they can easily scale to 10^6 points on a laptop scale independent of the learning task to be performed.

7.2 Major Contributions

The main goal of this thesis was to explore the role of sparsity to produce large scale kernel models for different learning tasks like classification, regression, clustering, community detection and feature selection. This was achieved through the following set of contributions:

1. Introduction of sparsity in the LSSVM [8] framework. This was achieved by including a convex relaxation of the L_0-norm penalty in the Primal Fixed-Size LSSVM [30] and the Subsampled dual LSSVM [130] objective functions.

2. We also explored the role of sparsity versus error trade-off for the very sparse LSSVM models introduced in Chapter 2 for large scale classification and regression problems.

3. We proposed a representative subset selection technique namely FURS [123] on which kernel based models are built for tasks like large scale clustering and community detection.

4. We showcased the effectiveness of kernel based methods for Big Data learning by converting the data into a k-NN graph using a distributed environment and performing subset selection using FURS in Chapter 3. We then built kernel based models for Big Data classification and clustering problems using this subset and inference for unseen data was done using the out-of-sample extensions property of the models.

5. We exploited the out-of-sample extensions property of kernel spectral clustering to perform community detection for Big Data networks in a tractable manner as described in Chapter 4.

6. We proposed two scalable and efficient model selection techniques to identify the right number of communities (k) in the large scale complex network.

7. In Chapter 4, we proposed a multilevel hierarchical kernel spectral clustering (MH-KSC) technique based on the idea of agglomerative generation of affinity matrices which overcomes the problem of **resolution limit** faced by state-of-the-art large scale community detection techniques like Louvain [36] and Infomap [42].

8. Proposed MH-KSC technique identifies good quality communities at coarser as well as finer levels of granularity.

9. In Chapter 5, we proposed a technique to identify a range in which the number of clusters k lie at a given level of hierarchy using the Gershgorin Circle Theorem [78] without performing the computationally expensive eigen-decomposition step for a normalized Laplacian matrix.

10. We modified and extended the MH-KSC technique [21] from networks to images and datasets resulting in agglomerative hierarchical kernel spectral data clustering (AH-KSC) method using the concept of distance thresholds for building iterative affinity matrices, one at each level of hierarchy, in combination with the Balanced Angular Fitting (BAF) [20] model selection criterion.

11. In Chapter 5, we also proposed sparse reductions to the KSC [13] using convex relaxation of L_0-norm and group lasso based penalties in combination with the objective function to generate reduced sets. This results in simpler and sparser models with computationally efficient out-of-sample time complexities.

12. We proposed a feature selection technique in a primal-dual optimization framework using the LSSVM [8] method for classification problems. We used a combination of the L_2-norm and a convex relaxation of L_0-norm penalty in the objective function to obtain sparse and robust LSSVM classification models for high dimensional datasets.

13. Finally, we designed a visualization tool named Netgram which tracks and visualizes the evolution of communities in dynamic networks. It adheres to certain design principles and aesthetic conditions [189] and keeps track of significant events like birth, death, split, merge, continuation, shrinkage and growth of communities in time-evolving networks. The advantage of Netgram tool is that it can work in combination with any evolutionary clustering model as a post-processing step.

7.3 Future Work

Some possible future research directions may consider the adaptation of the proposed algorithms i.e. for example exploring different penalty terms like Group Lasso [61], elastic-net [60] etc. in combination with the FS-LSSVM methods to obtain sparser kernel machines which retain certain structure like grouping of variables and removal of irrelevant features in high-dimensional

space along with selection of fewer points as support vectors in case of large scale datasets for tasks like time-series prediction.

Another direction of research would be to use the community detection techniques proposed in this thesis and extend them to large scale graphs which not only contain the topographic information like the adjacency list (neighborhood information) for each node but also has an additional dimension of information. This information can be social, personal and professional information like in Facebook, Google+ and LinkedIn networks respectively. The network now transforms into a tensor with the addition of this dimension. Kernel based methods employed in this thesis for community detection and clustering can be combined to obtain communities which are based not only on topography i.e. nodes within a community are more connected to other nodes within the community but the preferences of individual nodes within the community are also similar.

Also, a new generation of kernel-based methods can be cast in a deep architecture which can learn several levels of non-linearities. Deep learning was first proposed in artificial neural networks (ANN) in [193, 194] and has attracted much attention lately. Deep learning allows different levels of abstraction of the original data by stacking together several layers of neurons. Each layer performs an unsupervised learning task taking as input the output of the previous layer. In this way a rich understanding of the patterns at different scales can be obtained and meaningful features can be extracted. In the final layer, the supervised learning task is performed along with back-propagation. This results in very high accuracy in case of classification tasks like image classification. Some directions to explore more powerful models like kernel methods for deep learning have already been introduced in [195]. In relation to the methods devised in this thesis, the multilevel hierarchical kernel spectral clustering model can be extended to perform deep learning. In particular, the specific models built at each layer of hierarchy can be combined such that the final model produces a result which takes into account the outcomes of the models in the previous layers. Thus, in this case a model analogous to MH-KSC can be designed for deep learning.

Bibliography

[1] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From Data Mining to Knowledge Discovery in Databases," *AI Magazine*, vol. 17, pp. 37–54, 1996.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] S. Owen, R. Anil, T. Dunning, and S. Friedman, *Mahout in Action*. Manning Publications Co., 1st Edition, 2011.

[4] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media Inc, 3rd edition ed., 2009.

[5] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[6] A. K. Jain, "Data Clustering: 50 years beyond k-means," *Pattern Recognition Letters*, vol. 31, no. 8, 2010.

[7] I. T. Jolliffe, *Principal Component Analysis*. Springer Series in Statistics, 2nd Edition, 2002.

[8] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, *Least Squares Support Vector Machines*. World Scientific, Singapore, 2002.

[9] J. A. K. Suykens and J. Vandewalle, "Least Squares Support Vector Machine Classifiers," *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, 1999.

[10] V. N. Vapnik, *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[11] G. E. P. Box and G. C. Tiao, *Bayesian Inference in Statistical Analysis*. Wiley, 1973.

[12] M. Rosenblatt, "Remarks on some Nonparametric Estimates of a Density Function," *Annals of Mathematical Sciences*, vol. 27, no. (3):832, 1956.

[13] C. Alzate and J. A. K. Suykens, "Multiway spectral clustering with out-of-sample extensions through weighted kernel PCA," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 2, pp. 335–347, 2010.

[14] R. Langone, C. Alzate, and J. A. K. Suykens, "Kernel spectral clustering for community detection in complex networks," in *Proc of IJCNN*, pp. 1–8, 2012.

[15] C. Alzate and J. A. K. Suykens, "A semi-supervised formulation to binary kernel spectral clustering," in *Proc of IJCNN*, pp. 1–8, 2012.

[16] C. Alzate and J. A. K. Suykens, "ICA through an LS-SVM based kernel CCA measure for independence," in *Proc of IJCNN*, pp. 2920–2925, 2007.

[17] R. Mall, M. El Anbari, H. Bensmail, and J. A. K. Suykens, "Primal-Dual Framework for Feature Selection using Least Squares Support Vector Machines," in *Proc of* 19th *International Conference on Management of Data*, pp. 105–108, 2013.

[18] V. Jumutc and J. A. K. Suykens, "Multiclass Supervised Novelty Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 12, pp. 2510–2523, 2014.

[19] V. Van Belle, B. Van Calster, D. Timmerman, T. Bourne, C. Bottomley, L. Valentin, P. Neven, S. Van Huffel, J. A. K. Suykens, and S. Boyd, "A Mathematical Model for Interpretable Clinical Decision Support with Applications in Gynecology," *PloS ONE*, vol. 7, no. 3, 2010.

[20] R. Mall, R. Langone, and J. A. K. Suykens, "Kernel Spectral Clustering for Big Data Networks," *Entropy, Special Issue: Big Data*, vol. 15, no. 5, pp. 1567–1586, 2013.

[21] R. Mall, R. Langone, and J. A. K. Suykens, "Multilevel Hierarchical Kernel Spectral Clustering for Real-Life Large Scale Complex Networks," *PLoS ONE*, vol. 9, no. 6, p. e99666, 2014.

[22] R. Langone, C. Alzate, and J. A. K. Suykens, "Kernel Spectral Clustering with Memory Effect," *Physica A: Statistical Mechanics and its Applications*, vol. 392, no. 10, pp. 2588–2606, 2013.

[23] J. Luts, F. Ojeda, R. Van de Plas, B. De Moor, S. Van Huffel, and J. A. K. Suykens, "A tutorial on support vector machines based methods

for classification problems in chemometrics," *Analytica Chimica Acta*, vol. 665, no. 2, pp. 129–145, 2010.

[24] B. Huyandi, M. De Vos, M. Signoretto, J. A. K. Suykens, W. Van Paesschan, and S. Van Huffel, "Automatic Seizure Detection Incorporating Structural Information," in *Proc of* 21st *International Conference on Artificial Neural Networks*, pp. 233–240, 2011.

[25] C. C. Chang and C. J. Lin, "LIBSVM : A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 27, pp. 1–27, 2011.

[26] C. J. C. Burges, "Simplified support vector decision rules," in *In Proc of* 13th *International Conference on Machine Learning*, pp. 71–77, 1996.

[27] D. Geebelen, J. A. K. Suykens, and J. Vandewalle, "Reducing the number of Support Vectors of SVM Classifiers using the Smoothed Separable Case Approximation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 4, pp. 682–688, 2012.

[28] B. Schölkopf, S. Mika, C. J. C. Burges, P. Knirsch, K. Müller, G. Rätsch, and A. J. Smola, "Input space verus feature space in kernel-based methods," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 10, no. 5, pp. 1000–1017, 1999.

[29] T. Downs, K. E. Gates, and A. Masters, "Exact Simplification of Support Vector Solutions," *Journal of Machine Learning Research*, vol. 2, pp. 293–297, December 2001.

[30] K. De Brabanter, J. De Brabanter, J. A. K. Suykens, and B. De. Moor, "Optimized Fixed-Size Kernel Models for Large Data Sets," *Computational Statistics & Data Analysis*, vol. 54, no. 6, pp. 1484–1504, 2010.

[31] S. S. Keerthi, O. Chapelle, and D. DeCoste, "Building support vector machines with reduced classifier complexity," *Journal of Machine Learning Research*, vol. 7, pp. 1493–1515, July 2006.

[32] R. E. Fan, K. W. Chang, C. J. Hsieh, and C. J. Lin, "LIBLINEAR: A library for large scale classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[33] T. Joachims, *Making large-scale support vector machine learning practical.* MIT Press, 1999.

[34] I. Dhillon, Y. Guan, and B. Kulis, "Weighted Graph Cuts without Eigenvectors: A Multilevel Approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007.

[35] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1999.

[36] V. Blondel, J. Guillaume, R. Lambiotte, and L. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statisitcal Mechanics: Theory and Experiment*, vol. 10, no. P10008, 2008.

[37] M. E. J. Newman, "Modularity and community structure in networks," *PNAS*, vol. 103, no. 23, pp. 8577–8582, 2006.

[38] A. Clauset, M. Newman, and C. Moore, "Finding community structure in very large scale networks," *Physical Review E*, vol. 70, no. 066111, 2004.

[39] L. Danaon, A. Diáz-Guilera, J. Duch, and A. Arenas, "Comparing community structure identification," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 09, no. P09008+, pp. 75–174, 2005.

[40] A. Lanchichinetti, F. Radicchi, J. Ramasco, and S. Fortunato, "Finding statistically significant communities in networks," *PloS One*, vol. 6, no. e18961, 2011.

[41] J. Duch and A. Arenas, "Community detection in complex networks using external optimization," *Physical Review E*, vol. 72, no. 2, pp. 027104+, 2005.

[42] M. Rosvall and C. Bergstrom, "Maps of random walks on complex networks reveal community structure," *PNAS*, vol. 105, pp. 1118–1123, 2008.

[43] S. Fortunato and M. Barthélemy, "Resolution limit in community detection," *PNAS*, vol. 104, no. 36, 2007.

[44] A. Lancichinetti and S. Fortunato, "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities," *Physical Review E*, vol. 80, pp. 116–118, July 2009.

[45] R. Rabbany, M. Takaffoli, J. Fagnan, O. R. Zaiane, and C. R. J. G. B., "Relative Validity Criteria for Community Mining Algorithms," in *International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 258–265, 2012.

[46] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: analysis and an algorithm," in *Proc of the Advances in Neural Information Processing Systems*, pp. 849–856, 2002.

[47] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[48] U. von Luxburg, "A tutorial on Spectral clustering," *Stat. Comput*, vol. 17, pp. 395–416, 2007.

[49] F. R. K. Chung, *Spectral Graph Theory*. American Mathematical Society, 1997.

[50] C. Alzate and J. A. K. Suykens, "Highly Sparse Kernel Spectral Clustering with Predictive Out-of-sample extensions," in *ESANN*, pp. 235–240, 2010.

[51] C. Alzate and J. A. K. Suykens, "Sparse kernel spectral clustering models for large-scale data analysis," *Neurocomputing*, vol. 74, no. 9, pp. 1382–1390, 2011.

[52] C. Alzate and J. A. K. Suykens, "Hierarchical kernel spectral clustering," *Neural Networks*, vol. 35, pp. 21–30, 2012.

[53] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. John Wiley & Sons 2nd edition, 2001.

[54] T. Hastie, R. Tibshirani, and J. Friedman, *Elements of Statistical Learning*. Springer series in Statistics, 2001.

[55] R. Bekkermanm, R. El-Yaniv, N. Tishby, and Y. Winter, "Distributional word clusters versus words for text categorization," *Journal of Machine Learning Research*, vol. 3, pp. 1183–1208, 2003.

[56] P. Bradley and O. Mangasarian, "Feature selection via concave minimization and support vector machines," in *In Proc of the Fifteenth International Conference on Machine Learning*, pp. 82–90, 1998.

[57] M. Song, C. Breneman, J. Bi, N. Sukumar, K. Bennett, S. Cramer, and N. Tugcu, "Prediction of protein retention times in anion-exchange chromatography systems using support vector regression," *Journal of Chemical Information and Computer Sciences*, vol. 42, no. 6, pp. 1347–1357, 2002.

[58] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani, "1-norm SVMs," in *Neural Information Processing Systems*, pp. 1–8, 2003.

[59] L. Wang, J. Zhu, and H. Zou, "The doubly regularized support vector machine," *Statistica Sinica*, vol. 16, pp. 589–615, 2006.

[60] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of Royal Statistics*, vol. 67, pp. 301–320, 2005.

[61] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of Royal Statistical Society*, vol. 68, no. 1, pp. 49–67, 2006.

[62] J. Friedman, T. Hastie, and R. Tibshirani, "A note on the group lasso and a sparse group lasso," *arXiv:1001.0736*, 2010.

[63] H. Zou, "The Adaptive Lasso and its Oracle Properties," *Journal of American Statistical Association*, vol. 101, no. 476, pp. 1418–1429, 2006.

[64] H. Zou and H. H. Zhang, "On the adaptive elastic net with diverging number of parameters," *Annals of Statistics*, vol. 37, no. 4, pp. 1733–1751, 2009.

[65] F. Ojeda, J. A. K. Suykens, and B. De Moor, "Low rank updated LS-SVM classifiers for fast variable selection," *Neural Networks Special Issue on Advances in Neural Networks Research*, vol. 21, no. (2-3), pp. 437–449, 2008.

[66] J. Weston, A. Elisseeff, B. Schölkopf, and M. Tipping, "Use of the Zero Norm with Linear Models and Kernel Methods," *Journal of Machine Learning Research*, vol. 3, pp. 1439–1461, 2003.

[67] D. Chakrabarti, R. Kumar, and A. Tomkins, "Evolutionary clustering," in *In Proc of the* 12th *ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pp. 554–560, 2006.

[68] Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng, "Evolutionary spectral clustering by incorporating temporal smoothness," in *In Proc of KDD*, p. 153–162, 2007.

[69] D. Greene, D. Doyle, and P. Cunningham, "Tracking the evolution of communities in dynamic social networks," in *In Proc of International Conference on Advances in Social Network Analysis and Mining*, 2010.

[70] K. S. Xu, M. Kliger, and A. O. Hero III, "Adaptive Evolutionary Clustering," *Data Mining and Knowledge Discovery*, vol. 28, no. 2, pp. 304–336, 2014.

[71] G. Palla, A. L. Barabási, T. Vicsek, and B. Hungary, "Quantifying social group evolution," *Nature Letters*, vol. 446, 2007.

[72] R. Langone, C. Alzate, and J. A. K. Suykens, "Kernel spectral clustering with memory effect," *Physica A*, vol. 392, no. 10, pp. 2588–2606, 2013.

[73] C. L. Blake and C. J. Merz, "UCI repository of machine learning databases," in *http://archive.ics.uci.edu/ml/datasets.html*.

[74] U. Kang and C. Faloutsos, "Beyond 'caveman communities': Hubs and Spokes for graph compression and mining," in *Proc of ICDM*, pp. 300–309, 2011.

[75] A. Maiya and T. Berger-Wolf, "Sampling community structure," in *Proc. of WWW*, pp. 701–710, 2010.

[76] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *Journal of Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, 1953.

[77] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proc. of KDD*, pp. 631–636, 2006.

[78] S. Gershgorin, "Uber die Abgrenzung der Eigenwerte einer Matrix," *Izv. Akad. Nauk. USSR Otd. Fiz.-Mat. Nauk*, vol. 6, pp. 749–754, 1931.

[79] E. J. Candes, M. B. Wakin, and S. Boyd, "Enhancing Sparsity by Reweighted l1 Minimization," *Journal of Fourier Analysis and Applications, special issue on sparsity*, vol. 14, no. 5, pp. 877–905, 2008.

[80] K. Huang, D. Zheng, J. Sun, Y. Hotta, K. Fujimoto, and S. Naoi, "Sparse Learning for Support Vector Classification," *Pattern Recognition Letters*, vol. 31, no. 13, pp. 1944–1951, 2010.

[81] R. Langone, R. Mall, and J. A. K. Suykens, "Soft Kernel spectral clustering," in *Proc. of the IJCNN 2013*, pp. 1028 – 1035, 2013.

[82] J. A. K. Suykens, L. Lukas, and J. Vandewalle, "Sparse approximation using least squares support vector machines," in *In Proc of IEEE International Symposium on Circuits and Systems ISCAS*, pp. 757–760, 2000.

[83] Y. J. Lee and O. L. Mangasarian, "RSVM: Reduced Support Vector Machines," in *In Proc of the* 1st *SIAM International Conference on Data Mining*, 2001.

[84] Y. J. Lee and O. L. Mangasarian, "SSVM: A smooth support vector machine for classification," *Computational Optimization and Applications*, vol. 20, no. 1, pp. 5–22, 2001.

[85] Y. Li, C. Lin, and W. Zhang, "Improved Sparse Least-Squares Support Vector Machine Classifiers," *Neurocomputing*, vol. 69, no. 13, pp. 1655–1658, 2006.

[86] E. J. Nyström, "über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben," *Acta Mathematica*, vol. 54, pp. 185–204, 1930.

[87] C. T. H. Baker, *The Numerical Treatment of Integral Equations*. Oxford Claredon Press, 1983.

[88] P. Karsmakers, P. Pelckmans, K. De Brabanter, H. Van Hamme, and J. A. K. Suykens, "Sparse conjugate directions pursuit with application to fixed-size kernel methods," *Machine Learning, Special Issue on Model Selection and Optimization in Machine Learning*, vol. 85, no. 1, pp. 109–148, 2011.

[89] J. Lopez, K. De Brabanter, J. R. Dorronsoro, and J. A. K. Suykens, "Sparse LSSVMs with l_0-norm minimization," in *In Proc of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pp. 189–194, 2011.

[90] R. Mall and J. A. K. Suykens, "Sparse Variations to Fixed-Size Least Squares Support Vector Machines for Large Scale Data," in *In Proc of* 17th *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pp. 161–173, 2013.

[91] R. Fletcher, *Practical methods of Optimization.* John Wiley & Sons, 1987.

[92] C. K. I. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," *Advances in Neural Information Processing Systems*, vol. 13, pp. 682–688, 2001.

[93] S. Xavier de Souza, J. A. K. Suykens, J. Vandewalle, and D. Bollé, "Coupled Simulated Annealing for Continuous Global Optimization," *IEEE Transactions on Systems, Man, and Cybertics*, vol. 40, no. 2, pp. 320–335, 2010.

[94] M. Rudemo, "Empirical choice of histograms and kernel density estimators," *Scandinavian Journal of Statistics*, vol. 9, pp. 65–78, 1982.

[95] L. Du, Y. Yang, D. He, R. G. Harley, T. G. Habetler, and B. Lu, "Support Vector Machines Based Methods For Non-Intrusive Identification of Miscellaneous Electric Loads," in *In Proc of* 38th *Annual Conference of the IEEE Industrial Electronics Society (IECON)*, 2012.

[96] A. W. Bowman, "An alternative method of cross-validation for smoothing of density estimators," *Biometrika*, vol. 71, pp. 353–360, 1984.

[97] D. W. Scott and S. R. Sain, "Multi-dimensional Density Estimation," *Data Mining and Computational Statistics*, vol. 23, pp. 229–263, 2004.

[98] D. Crandall, D. Cosley, D. Huttenlocher, J. Kleinberg, and S. Suri, "Feedback effects between similarity and social influence in online communities," in *KDD'08*, pp. 160–168, 2008.

[99] E. Ferrera, "A large-scale community structure analysis in Facebook," *EPJ Data Science*, vol. 1, pp. 1–30, November 2012.

[100] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins, "Microscopic evolution of social networks," in *KDD'08*, pp. 462–470, 2008.

[101] H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A. L. Barabasi, "The large-scale organization of metabolic networks," *Nature*, vol. 407, pp. 651–654, October 2000.

[102] E. Bullmore and O. Sporns, "Complex brain networks: graph theoretical analysis of structural and functional systems," *Nature Reviews. Neuroscience*, vol. 10, February 2009.

[103] S. Fortunato, "Community detection in graphs," *Physics Reports*, no. 486, pp. 75–174, 2009.

[104] M. Girvan and M. Newman, "Community structure in social and biological networks," *PNAS*, vol. 99, no. 12, pp. 7821–7826, 2002.

[105] R. Langone, C. Alzate, and J. A. K. Suykens, "Kernel spectral clustering for community detection in complex networks," in *Proc IEEE WCCI/IJCNN*, pp. 2596–2603, 2012.

[106] A. Lancichinetti and S. Fortunato, "Community detection algorithms: a comparitive analysis," *Physical Review E*, vol. 80, no. 056117, 2009.

[107] C. Hubler, H. P. Kriegel, K. Borgwardt, and Z. Ghahramani, "Metropolis algorithms for representative subgraph sampling," in *ICDM'08*, pp. 283–292, 2008.

[108] D. F. Gleich and C. Seshadhri, "Vertex Neighborhoods, Low Conductance Cuts, and Good Seeds for Local Community Methods," in *Proc of KDD'12*, pp. 597–605, 2012.

[109] J. B. MacQueen, "Some methods for classification and analysis of Multivariate Observations," in *Proc of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, 1967.

[110] L. Katz, "A New Status Index Derived from Sociometric Index," *Psychometrika*, pp. 39–43, 1953.

[111] P. Bonacich, "Power and Centrality: A Family of Measures," *The American Journal of Sociology*, vol. 92, no. 5, pp. 1170–1182, 1987.

[112] L. C. Freeman, "Centrality in Social Networks:Conceptual Clarification," *Social Networks*, vol. 1, no. 3, pp. 215–239, 1979.

[113] D. Rafiei, "Effecively visualizing large networks through sampling," in *Proceedings of VIS 05*, pp. 375–382, 2005.

[114] M. Adler and M. Mitzenmacher, "Towards compressing web graphs," in *Proceedings of IEEE DCC*, pp. 203–212, 2000.

[115] T. Feder and R. Motwani, "Clique partitions, graph compression and speeding-up algorithms," in *Journal of Computer and System Sciences*, pp. 123–133, 1991.

[116] A. C. Gilbert and K. Levchenko, "Compressing network graphs," in *Proc. LinkKDD workshop at the* 10th *ACM Conference on KDD*, August 2004.

[117] O. Frank, *Network Sampling and Model Fitting*. Cambridge Press University, February 2005.

[118] S. Goel and M. Salganik, "Respondent-driven sampling as Markov Chain Monte Carlo," *Statistics in Medicine*, vol. 17, no. 28, pp. 2202–2229, 2009.

[119] A. Mehler and S. Skiena, "Expanding network communities from representative examples," *ACM Transactions on Knowledge Discovery from Data*, vol. 3, no. 2, pp. 1–27, 2009.

[120] U. Feige, "A threshold of ln for approximating set cover," *Journal of the ACM*, vol. 45, no. 4, pp. 634–652, 1998.

[121] M. Meil, "Comparing clustering information based distance," *Journal of Multivariate Analysis*, vol. 98, no. 5, pp. 873–895, 2007.

[122] J. Wu, H. Xiong, and J. Chen, "Adapting the right measures for k-means clustering," in *Proc of SIGKDD'09*, pp. 877–886, 2009.

[123] R. Mall, R. Langone, and J. A. K. Suykens, "FURS: Fast and Unique Representative Subset selection for large scale community structure," *Social Network Analysis and Mining*, vol. 3, no. 4, pp. 1075–1095, 2013.

[124] D. S. Yates, D. S. Moore, and D. S. Starnes, *The Practice of Statistics*. Freeman, 3rd edition ed., 2008.

[125] C. E. Särdnal, B. Swenson, and J. Wretman, *Model assisted survey sampling*. Springer-Verlag, 1992.

[126] R. Mall, R. Langone, and J. A. K. Suykens, "Self-tuned kernel spectral clustering for large scale networks," in *Proc of IEEE BigData*, pp. 385–393, 2013.

[127] D. S. Moore and G. P. McCabe, *Introduction to the practice of statistics*. W.H. Freeman & Co., 5th Edition, 2005.

[128] D. Eppstein, M. S. Paterson, and F. Yao, "On-nearest neighbor graphs," *Discrete and Computational Geometry*, vol. 17, no. 3, pp. 263–282, 1997.

[129] G. L. Miller, S. Teng, W. Thurston, and S. A. Vavasis, "Separators for sphere-packings and nearest neighbor graphs," *J. ACM*, vol. 44, no. 1, pp. 1–29, 1997.

[130] R. Mall and J. A. K. Suykens, "Very Sparse LSSVM Reductions for Large Scale Data," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 26, no. 5, pp. 1086–1097, 2015.

[131] J. Liu, A. Kimbell, and A. C. Myers, "Interruptible iterators," *SIGPLAN Not.*, vol. 41, no. 1, pp. 283–294, 2006.

[132] A. Kleiner, A. Talwalker, P. Sarkar, and M. I. Jordan, "The Big Data Bootstrap," in *In Proc of ICML*, 2012.

[133] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*. London: Chapman & Hall/CRC, 1998.

[134] A. Agarwal, O. Chapelle, and J. Langford, "A Reliable Effective Terascale Linear Learning System," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1111–1133, 2014.

[135] M. Zaharia, M.Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with Working Sets," in *Proc. of the* 2nd *USENIX Conference on Hot Topics in Cloud Computing*, 2010.

[136] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd Edition, 2001.

[137] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, pp. 308–313, 1965.

[138] S. Schaffer, *Algorithms for Nonuniform Networks*. PhD thesis, Helsinki University of Technology, 2006.

[139] L. Zelnik-Manor and P. Perona, "Self-tuning spectral clustering," in *Advances in Neural Information Processing Systems*, pp. 1601–1608, 2005.

[140] L. Muflikhah, "Document Clustering using Concept Space and Cosine Similarity Measurement," in *Proc of ICCTD*, pp. 58–62, 2009.

[141] J. Baylis, *Error Correcting Codes: A Mathematical Introduction*. FL: CRC Press, 1988.

[142] L. Hubert and P. Arabie, "Comparing Partitions," *Journal of Classification*, vol. 2, no. 1, pp. 193–218, 1985.

[143] J. Yang and J. Leskovec, "Overlapping community detection at scale: a non-negative matrix factorization approach," in *In Proc. of WSDM*, pp. 587–596, 2013.

[144] M. Kumpula, J. Saramaki, K. Kaski, and J. Kertesz, "Limited resolution and multiresolution methods in complex network community detection," *Fluctuation Noise Letters*, vol. 7, no. 209, 2007.

[145] H. Good, A. de Montjoye, and A. Clauset, "Performance of modularity maximization in practical contexts," *Physical Review E*, vol. 81, no. 046106, 2010.

[146] A. Lanchichinetti and S. Fortunato, "Limits of modularity maximization in community detection," *Physical Review E*, vol. 84, no. 066122, 2011.

[147] D. Kushnir, M. Galun, and A. Brandt, "Fast multiscale clustering and manifold identification," *Pattern Recognition*, vol. 39, no. 10, pp. 1876–1891, 2006.

[148] J. C. Lamirel, R. Mall, P. Cuxac, and G. Safi, "A New Efficient and Unbiased Approach for Clustering Quality Evaluation," in *Proc of PAKDD Workshops*, pp. 209–220, 2011.

[149] H. Steinhaus, "Sur la division des corp material en parties," *Bulletin of Acad. Polon. Sci.*, vol. 4, no. 3, pp. 801–804, 1956.

[150] S. Llyod, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, pp. 129–137, 1982.

[151] M. Ester, K. Peter, J. Hans, and X. Xiaowel, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc of the* 2nd *International Conference on Knowledge Discovery and Data Mining*, pp. 226–231, AAAI Press, 1996.

[152] G. E. Mclachlan and K. E. Basford, *Mixture Models: Inference and applications to clustering*. Marcel Dekker, 1987.

[153] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet Allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[154] M. Welling, M. Rosen-zvi, and G. Hinton, "Exponential family harmoniums with an application to information retrieval," *Advances in Neural Information Processing Systems*, vol. 17, pp. 1481–1488, 2005.

[155] D. Pelleg and A. Moore, "X-means: Extending k-means with efficient estimation of the number of clusters," in *Proc of International Conference on Machine Learning*, pp. 727–734, Morgan Kaufmann, 2000.

[156] R. E. Kass and L. Wasserman, "A reference bayesian test for nested hypothesis and its relationship to the Schwarz criterion," *Journal of the American Statistical Association*, vol. 90, no. 431, pp. 928–934, 2000.

[157] G. Schwarz, "Estimating the dimension of a model," *The Annals of Statistics*, vol. 6, no. 2, pp. 461–464, 2001.

[158] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, pp. 465–471, 1978.

[159] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. 19, pp. 716–723, 1974.

[160] G. Hamerly and C. Elkan, "Learning the k in k-means," in *Proc of* 17th *annual conference on neural information processing systems (NIPS)*, pp. 281–288, 2003.

[161] P. Sand and A. W. Moore, "Repairing faulty mixture models using density estimations," in *Proc of* 18th *International Conference on Machine Learning*, pp. 457–464, 2001.

[162] Y. Feng and G. Hamerly, "PG-means: Learning the number of clusters in data," in *Advances in Neural Information Processing Systems*, 2006.

[163] M. Polito and P. Perona, "Grouping and dimensionality reduction by locally linear embedding," *Advances in NIPS*, vol. 14, 2002.

[164] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the number of clusters in a data set via the gap statistic," *Journal of Royal Statistical Society*, vol. 63, no. 2, pp. 411–423, 2001.

[165] J. Chen, J. Lu, C. Zhan, and G. Chen, "Laplacian Spectra and Synchronization Processes on Complex Networks," *Handbook of Optimization in Complex Networks. Optimization and its Applications*, pp. 81–113, 2012.

[166] M. Meila and J. Shi, "A Random Walks View of Spectral Segmentation," in *Proc of International Conference on Artificial Intelligence and Statistics*, 2001.

[167] H. J. Li and X. S. Zhang, "Analysis of stability of community structure across multiple hierarchical levels," *EPL*, vol. 103, no. 58002, 2013.

[168] C. J. Veenman, M. J. T. Reinders, and E. Backer, "A maximum variance cluster algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 9, pp. 1273–1280, 2002.

[169] A. K. Jain and P. Flynn, "Image segmentation using clustering," in *In Advances in Image Understanding*, pp. 65–83, IEEE Computer Society Press, 1996.

[170] R. Mall, R. Langone, and J. A. K. Suykens, "Agglomerative Hierarchical Kernel Spectral Clustering for Large Scale Networks," in *Proc of European Symposium on Artificial Neural Networks (ESANN)*, 2014.

[171] R. Tibshirani, "Regression shrinkage and Selection via the Lasso," *Journal of Royal Statistical Society*, vol. 58, no. 1, pp. 267–288, 1996.

[172] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming," in *http://cvxr.com/cvx*, 2010.

[173] A. R. R. Neto and G. A. Barreto, "A novel heuristic for Building Reduced-Set SVMs using the Self-Organizing Map," *Advances in Computational Intelligence*, vol. 6691, pp. 97–104, 2001.

[174] E. Amaldi and V. Kann, "On the approximability of minimizing non zero variables or unsatisfied relations in linear systems," *Theoritical Computer Science*, vol. 209, pp. 237–260, 1998.

[175] K. Huang, I. King, and M. R. Lyu, "Direct Zero-norm Optimization for Feature Selection," in *In Proc of the* 8th *International Conference on Data Mining*, pp. 845–850, 2008.

[176] Y. Li, C. Campbell, and M. Tipping, "Bayesian automatic relevance determination algorithms for classifying gene expression data," *Bioinformatics*, vol. 18, no. 10, pp. 1332–1339, 2002.

[177] O. Bousquet and A. Elisseeff, "Stability and Generalization," *Journal of Machine Learning Research*, vol. 2, pp. 499–526, 2002.

[178] A. E. Hoerl and R. W. Kennard, "Ridge Regression: Biased Estimation for Non-othrogonal Problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[179] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine Learning*, pp. 389–422, 2002.

[180] P. Bródka, S. Saganowski, and P. Kazienko, "GED: the method for group evolution discovery in social networks," *Social Network Analysis and Mining*, vol. 3, no. 1, pp. 1–14, 2013.

[181] Y. R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng, "Analyzing communities and their evolutions in dynamic social networks," *ACM Transactions on Knowledge Discovery in Data*, vol. 3, no. 2, 2009.

[182] P. J. Mucha, T. Richardson, K. Macon, M. A. Porter, and J. P. Onnela, "Community structure in time-dependent, multiscale and multiplex networks," *Science*, vol. 328, no. 5980, pp. 876–878, 2010.

[183] Y. Sun, Y. Yu, and J. Han, "Ranking-Based Clustering of Heterogenous Networks with Star Network Schema," in *Proc of KDD*, 2009.

[184] R. Langone, R. Mall, and J. A. K. Suykens, "Clustering data over time using kernel spectral clustering with memory," in *In Proc of IEEE SSCI CIDM*, pp. 1–8, 2014.

[185] P. J. Pratt, *A Guide To SQL*. Seventh Edition, Thomson Course Technology, 2005.

[186] J. Sun, C. Faloutsos, S. Papadimitriou, and Y. P. S., "Graphscope: parameter-free mining of large time-evolving graphs," in *Proc. of the* 13th *ACM SIGKDD International conference on Knowledge discovery and Data Mining*, pp. 687–696, 2007.

[187] C. T. Yu and W. Meng, *Principles of Database Query Processing for Advanced Applications*. Morgan Kaufmann, 1998.

[188] F. Murtagh, "Counting dendograms: A survey," *Discrete Applied Mathematics*, vol. 7, no. 2, pp. 191–199, 1984.

[189] Y. Tanahashi, C. H. Hsueh, and K. Ma, "An Efficient Framework for Generating Storyline Visualizations from Streaming Data," *IEEE Transactions on Visualization & Computer Graphics*, no. doi:10.1109/$TVCG$.2015.2392771, 2015.

[190] Y. Tanahashi and K. Ma, "Design Considerations for optimal storyline visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 2, pp. 2679–2688, 2012.

[191] N. Eagle, A. S. Pentland, and D. Lazer, "Inferring social network structure using mobile phone data," *PNAS*, vol. 106, no. 1, pp. 15274–15278, 2009.

[192] S. Robertson, "Understanding inverse document frequency: On theoretical arguments for IDF," *Journal of Documentation*, vol. 60, no. 5, p. 503–520, 2004.

[193] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[194] Q. V. Le, M. Ranzato, R. Monga, M. Devin, G. Corrado, K. Chen, J. Dean, and A. Y. Ng, "Building high-level features using large scale unsupervised learning," in *Proc of ICML*, pp. 1–12, 2012.

[195] S. Bengio, L. Deng, H. Larochelle, H. Lee, and R. Salakhutdinov, "Special section on learning deep architectures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1795–1797, 2013.

Curriculum vitae

Raghvendra Mall was born in Kolkata, India on March 5th, 1988. He received his Bachelors in Technology in 2010 and Masters by Research in Computer Science with primary focus on Data Mining in 2011 from the International Institute of Information Technology, Hyderabad, India.

During his Masters, he worked as a Research Intern at the TALARIS team, INRIA, LORIA, Nancy, France from August 2010 to January 2012 under the guidance of Prof. Jean-Charles Lamirel. There he worked on devising new techniques for incremental clustering for biological and textual datasets.

After finishing his Masters, Raghvendra worked at Microsoft R & D Private Limited, Bangalore, India as a research intern from August 2011 to January 2012. There he worked on large scale keyword-advertiser graphs and came up with hierarchical clustering techniques to identify micro markets using this graph and methods to optimize the revenue from the Bing search engine by efficiently ordering the advertisements.

He started his doctorate at ESAT-STADIUS, KU Leuven, Belgium in June, 2012 under the guidance of Prof. Johan A.K. Suykens. During his doctorate, Raghvendra explored multiple aspects of sparsity in large scale machine learning using kernel methods. He published 6 first author peer-reviewed journal papers and more than 10 first author conference papers.

His current research interests include sampling, flat and hierarchical community detection in complex networks, exploring the role of sparsity in classification, clustering, regression, anomaly detection and designing visualization techniques.

He also accomplished a number of professional duties, including teaching, supervision of master thesis students, reviewer work for several journals and active participation in various conferences and research meetings.

List of publications

International Journal Papers:

- **Mall R.**, Mehrkanoon S., Suykens J.A.K., “Identifying Intervals for Hierarchical Clustering using the Gershgorin Circle Theorem", Pattern Recognition Letters, vol. 55, April. 2015, pp. 1-7.
- **Mall R.**, Suykens J.A.K., "Very Sparse LSSVM Reductions for Large Scale Data", IEEE Transactions on Neural Networks and Learning Systems, vol. 26, no. 5, Mar. 2015, pp. 1086 - 1097.
- Mehrkanoon S., Alzate C., **Mall R.**, Langone R., Suykens J.A.K., “Multi-class semi-supervised learning based upon kernel spectral clustering", IEEE Transactions on Neural Networks and Learning Systems, DOI:10.1109/TNNLS.2014.2322377, 2014.
- **Mall R.**, Langone R., Suykens J.A.K., "Multilevel Hierarchical Kernel Spectral Clustering for Real-Life Large Scale Complex Networks", PLOS One, e99966, vol. 9, no. 6, Jun. 2014.
- **Mall R.**, Langone R., Suykens J.A.K., "FURS: Fast and Unique Representative Subset selection retaining large scale community structure", Social Network Analysis and Mining, vol. 3, no. 4, Oct. 2013, pp. 1075-1095.
- **Mall R.**, Langone R., Suykens J.A.K., "Kernel Spectral Clustering for Big Data Networks", Entropy, Special Issue: Big Data, vol. 15, no. 5, May 2013, pp. 1567-1586.

International Conference Papers :

- **Mall R.**, Suykens J.A.K., “Kernel Spectral Document Clustering Using Unsupervised Precision-Recall Metrics", in Proc. of IJCNN, Killarney, Dublin, 2015.
- Mehrkanoon S., Agudelo M., **Mall R.**, Suykens J.A.K., “Hierarchical Semi-Supervised Clustering using KSC based model", in Proc. of IJCNN, Killarney, Dublin, 2015.
- **Mall R.**, Langone R., Suykens J.A.K., “Ranking Overlap and Outlier Points in Data using Soft Kernel Spectal Clustering", in Proc. of European Symposium on Artificial Neural Networks (ESANN), Brugges, Belgium, Apr. 2015.
- **Mall R.**, Jumutc V., Langone R., Suykens J.A.K., “Representative Subsets For Big Data Learning using kNN graphs", in Proc. of IEEE BigData, Washington D.C., 2014, pp. 37-42.
- **Mall R.**, Langone R., Suykens J.A.K., “Agglomerative Hierarchical Kernel Spectral Data Clustering", in Proc. of IEEE Symposium on Computational Intelligence (SSCI-CIDM), Orlando, Florida, Dec. 2014, pp. 9-16.
- Langone R., **Mall R.**, Suykens J.A.K., “Clustering data over time using kernel spectral clustering with memory", in Proc. of the Symposium Series on Computational Intelligence (SSCI-CIDM), Orlando,Florida, Dec. 2014, pp. 1-8.
- **Mall R.**, Mehrkanoon S., Langone R., Suykens J.A.K., “Optimal Reduced Sets for Sparse Kernel Spectral Clustering", in Proc. of IJCNN 2014, Beijing, China, July 2014, pp. 2436-2443.
- **Mall R.**, Langone R., Suykens J.A.K., “Agglomerative Hierarchical Kernel Spectral Clustering for Large Scale Networks", in Proc. of the European Symposium on Artitficial Neural Networks (ESANN), Brugges, Belgium, Apr. 2014.
- **Mall R.**, El Anbari M., Bensmail H., Suykens J.A.K., “Primal-Dual Framework for Feature Selection using Least Squares Support Vector Machines", in Proc. of the 19th International Conference on Management of Data (COMAD), Ahmedabad, India, Dec. 2013, pp. 105-108.

- **Mall R.**, Langone R., Suykens J.A.K., "Highly Sparse Reductions to Kernel Spectral Clustering", in Proc. of the 5th International Conference on Pattern Recognition and Machine Intelligence (PReMI), Kolkata, India, Dec. 2013, pp. 163-169.
- **Mall R.**, Langone R., Suykens J.A.K., "Self-Tuned Kernel Spectral Clustering for Large Scale Networks", in Proc. of the IEEE International Conference on Big Data (IEEE BigData 2013), Santa Clara, United States of America, Oct. 2013, pp. 385-393.
- Langone R., **Mall R.**, Suykens J.A.K., "Soft Kernel Spectral Clustering", in Proc. of the (IJCNN 2013), Dallas, Texas, Aug. 2013, pp. 1028-1035.
- **Mall R.**, Suykens J.A.K., "Sparse Reductions for Fixed-Size Least Squares Support Vector Machines on Large Scale Data", in Proc. of the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2013), GoldCoast, Australia, Apr. 2013, pp. 161-173.

Book Chapters :

- **Mall R.**, Suykens J.A.K., "KSC-net: Community Detection for Big Data Networks", in Big Data: Algorithms, Analytics, and Applications, (Li K., Jiang H., Yang L., and Cuzzocrea A., eds.), CRC Press, Taylor & Francis Group, ISBN 9781482240559, 2015.

Forthcoming Papers & Submissions:

- **Mall R.**, Langone R., Suykens J.A.K., "Netgram: Visualizing Communities in Evolving Networks", submitted for publication.
- Langone R., **Mall R.**, Vandewalle J., Suykens J. A. K., "Discovering cluster dynamics using kernel spectral methods", submitted as Chapter to a book.
- Langone R., **Mall R.**, Alzate C., Suykens J.A.K. "Kernel Spectral Clustering and applications", submitted as Chapter to a book.



FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING (ESAT)
STADIUS
Kasteelpark Arenberg 10, bus 2446
B-3001 Heverlee
raghvendra.mall@esat.kuleuven.be
http://homes.esat.kuleuen.be/~rmall