Fixed-Size Least Squares Support Vector Machines: Scala Implementation for Large Scale Classification

Mandar Chandorkar, Raghvendra Mall, Oliver Lauwers, Johan A.K Suykens, Bart De Moor
ESAT-STADIUS KU Leuven
Kasteelpark Arenberg 10,
B-3001 Leuven, Belgium
Email: mandar2812@gmail.com,{raghvendra.mall,oliver.lauwers,johan.suykens,bart.demoor}@esat.kuleuven.be

***Abstract*—We propose *FS-Scala*, a flexible and modular *Scala* based implementation of the Fixed Size Least Squares Support Vector Machine (FS-LSSVM) for large data sets. The framework consists of a set of modules for (gradient and gradient free) optimization, model representation, kernel functions and evaluation of FS-LSSVM models. A kernel based *Fixed-Size Least Squares Support Vector Machine* (FS-LSSVM) model is implemented in the proposed framework, while heavily leveraging the parallel computing capabilities of *Apache Spark*. Global optimization routines like *Coupled Simulated Annealing* (CSA) and *Grid Search* are implemented and used to tune the hyper-parameters of the FS-LSSVM model. Finally, we carry out experiments on benchmark data sets like *Magic Gamma*, *Adult*, *Forest Cover Type* and *Higgs SUSY* and evaluate the performance of various kernel based FS-LSSVM models.**

I. Introduction

The 21st century stands out in how mankind learned the value of storing and making predictions/decisions from large volumes of data. A significant aspect of large scale data analysis is distributed computation frameworks like *High Performance Computing*, *Message Passing Interface* etc. Recently large scale commodity hardware clusters have replaced the two former frameworks as the most popular model for parallel data analysis. With this crucial change in hardware came a change in computational models as well. It is at this juncture that distributed *Map Reduce* became the de-facto computational philosophy for large scale data analysis and words such as *Hadoop* [1], [2], [3] and *Apache Spark* [4], [5] have become synonymous with large scale data analysis and machine learning.

Along with innovation in hardware design and distributed computing models, there came a need for good programming libraries and frameworks to work with various Machine Learning models on large data sets. It was demonstrated in [6] that a gigantic language corpus encapsulates almost all aspects of human language and speech. So far the prevalent 'motto' in the Internet industry has been "large data, simple models". Often, this is misunderstood as the Machine Learning translation of *Occam's Razor*. The bias-variance trade-off [7] is a far better mechanism to ensure the model does not become overly complex, and this, rather than restricting the user to simple models, is the real Occam's razor in training a model.

Therefore, in order to extract maximum value from large scale data, it is important to have the flexibility to train and compare different model families before arriving at the one that fits the requirement of the user. Therefore one must be able to train general nonlinear models and tweak them by changing the various components which they employ to learn (i.e., a model may be linear or kernel based, it can be optimized by various methods like *Stochastic Gradient Descent*, *Conjugate Gradient*, etc.). This is not possible in a rigid, monolithic programming framework. Modularity, extensibility and ease of usage are of paramount importance while designing Machine Learning software for large scale data applications.

978-1-4799-7560-0/15/$31 ©2015 IEEE

The current state of the art in distributed Machine Learning in Scala is the *MLLib* module in *Apache Spark* [8]. It has implementations of Linear SVM and Logistic Regression for solving binary classification problems. But a crucial component missing in *MLLib* and all distributed Machine Learning libraries is the ability to learn classification models with non-linear decision boundaries. FS-Scala aims to solve the problem of scalable non-linear classification models by implementing the *Fixed-Size Least Squares Support Vector Machine* (FS-LSSVM) algorithm [9], [10] with model tuning capabilities.

In recent literature we find sparse reductions to FS-LSSVM methods [11], [12]. The authors in [11], [12] explored the sparsity vs error trade-off for FS-LSSVM models. Even though they run experiments on large scale datasets like Forest Cover dataset, the scalability of these methods are restricted to available memory on a single machine. Moreover, they don't exploit the possibility of parallelism available in several components of the FS-LSSVM model. Another work [13] converts the Big Data into a Big Network and then uses a network based subset selection technique (*Fast and Unique Representative Subset selection* (FURS) [14]) to obtain a representative subset of the original data. It then builds a FS-LSSVM model using this subset. However, in this paper we showcase that we can parallelize the subset selection technique which maximizes the *Quadratic Rènyi Entropy* for Big datasets and use the generated subset as the set of prototype vectors (PV) essential for building the FS-LSSVM model.

This paper is organized as follows. Section II introduces the FS-LSSVM algorithm [9]. Section III outlines the various modules that comprise FS-Scala and their roles. An implementation of the FS-LSSVM model is constructed using the framework and tested on various data sets, with the findings outlined in IV. Finally, we conclude in Section V.

II. Least Squares Support Vector Machines

A. Formulation

Least Squares Support Vector Machines (LSSVM) [10] [15] modifies the SVM formulation to include the *squared error* loss function and equality constraints with respect to the error variables e_i, as shown in (1).

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

with w and b the weights and bias parameters of the model, γ a regularization parameter, ϕ the feature map, x_i a training data point and y_i the corresponding label. N is the size of the training set. Applying the KKT conditions on (1) gives us the solution in the dual [10]. As compared to the classical SVM this solution looses sparsity since each point becomes a support vector, however we solve a linear system and not a *Quadratic Programming* problem.

B. FS-LSSVM

For large scale data analysis, solving (1) in the dual is not advantageous as the size of the solution matrix is equal to the size of the original data. In order to make training large scale kernel SVM models feasible, one needs to make approximations to the computation of the kernel matrices. The Fixed-Size LSSVM (FS-LSSVM) as proposed by De Brabanter, Suykens et. al [9], [10] consists of solving the LSSVM problem in the primal as follows.

$$\min_{w,b} \ \frac{1}{2}w^\intercal w + \frac{\gamma}{2}\sum_{i=1}^{n}\left(y_i - w^\intercal \hat{\phi}(x_i) - b\right)^2. \tag{2}$$

The solution to equation 2 is given by:

$$\begin{pmatrix} \hat{w} \\ \hat{b} \end{pmatrix} = \left(\hat{\Phi}_e^\intercal \hat{\Phi}_e + \frac{I_{m+1}}{\gamma}\right)^{-1} \hat{\Phi}_e^\intercal y, \tag{3}$$

$$\text{where} \quad \hat{\Phi}_e = \begin{pmatrix} \hat{\phi}_1(x_1) & \cdots & \hat{\phi}_m(x_1) & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \hat{\phi}_1(x_n) & \cdots & \hat{\phi}_m(x_n) & 1 \end{pmatrix}.$$

In the above formulation, $\hat{\phi}(x_k)$ is an approximation to the true feature map $\phi(x_k)$ (likewise, the hats on other variables indicate estimates of the corresponding objects in the earlier equations), which is related to the kernel $K(x_i, x_j) = \phi(x_i)^\intercal \phi(x_j)$ (Mercer's theorem). The approximate feature map $\hat{\phi}(x_k)$ is calculated using the Nyström method as outlined in [9], [11] and [12]. A low rank approximation to the kernel matrix is constructed by iteratively calculating a subset of the original data which maximizes the *Quadratic Rènyi Entropy*. This procedure of extracting $\hat{\phi}(x_k)$ from a data set, given a kernel function, is called *Automatic Feature Extraction* (AFE).

Kernel based models are sensitive to hyper-parameters. In the case of FS-LSSVM we have to tune the model with respect to γ the regularization parameter and the parameters of the kernel chosen. Models are generally compared with their cross-validation performance in which case the objective cost function with respect to the hyper-parameters is in general non-smooth and non-convex. Gradient free methods like Grid Search, Nelder Mead [16] and Coupled Simulated Annealing [17] are suitable to tackle the problem of model selection for FS-LSSVM based kernel models.

Algorithm 1 explains the steps involved in tuning the FS-LSSVM model with the bold part representing our contributions in this paper, which have been implemented in a MapReduce setting. As per 1, after preprocessing the original data, *FS-Scala* performs Rènyi Entropy based greedy subset selection to choose a prototype set of the prescribed size. Model tuning is carried out using either grid search or Coupled Simulated Annealing, by first initializing a grid on the hyper-parameters of the model (i.e. regularization constant and kernel parameters). Every point on the defined grid defines a unique model instance. Each instance is scored according to its n-fold cross-validation score and at the end of this procedure the best performing model instance is chosen to train on the entire training data and evaluated against a test set that was held out initially.

Algorithm 1: Tuning FS-LSSVM

```
Data: Data Set, Kernel, Global Optimization routine, grid parameters
Result: Proposed Tuned FS-LSSVM model
Pre-process the data by mean scaling.;
Calculate the prototype set by maximizing the Quadratic Rènyi Entropy in parallel using MapReduce.;
Initialize a grid for the hyper-parameters;
while termination of global optimization routine do
    Initialize the kernel using the hyper-parameters. Do AFE on the kernel matrix constructed from the prototypes, using the Nystrom method;
    evaluate the cross validation score for the particular hyper-parameter values;
end
```

III. FS-LSSVM Implementation

Our Scala-based [18] software, called FS-Scala, tackles three major issues w.r.t. the implementation of the FS-LSSVM:

- **Tuning Kernel Models**: Since the performance of kernel based models is sensitive with respect to the choice of hyper-parameters, one has to choose a mechanism of model selection or hyper-parameter optimization. In FS-Scala, we implement the Grid Search and Coupled Simulated Annealing global optimization algorithms for model tuning.
- **Parallel Computation**: Big Data analysis requires the distribution of computational work load, *MapReduce* is the dominant paradigm employed for writing distributed data processing programs. In FS-Scala we leverage *MapReduce* to distribute the computation in the pre-processing, training and cross-validation tasks.
- **Infrastructure Flexibility**: The big data landscape has many tools which enable the storage and analysis of

large streams of data, they consist of technologies such as, but not limited to *Apache Spark*, *Hadoop*, Graph Databases like *Titan* [19], *OrientDB* [20], *Neo4j* [21]. Creating a powerful framework for model training and evaluation requires the decoupling of storage and processing infrastructure from the actual logic that implements the architecture of learning models.

The implementation of the FS-LSSVM in FS-Scala, outlined in Algorithm 1 is as described in the original article of De Brabanter et al. [9]. Kernel based models all implement the interface *GloballyOptimizable* in the optimization module (see Figure 2). Since the *GlobalOptimizer* and its subclasses (i.e. *GridSearch* and *CoupledSimulated Annealing*) all optimize models which implement the GloballyOptimizable interface, it enables tuning of models with a variety of global optimization algorithms convenient.

Architecture

Figure 1 shows the organization of modules in FS-Scala. It can be decomposed into five principal modules:

- Model Classes: This is the core set of classes which form the heart of the library, a number of abstract model categories are defined each with its own set of defined behaviours.
- Optimization application programming interface (API): A module which houses the implementation of common optimization methods (i.e. Gradient and Gradient free). Currently FS-Scala has implementations for Conjugate Gradient (algorithm 3), Gradient Descent, Grid Search and Coupled Simulated Annealing [17] (CSA).
- Kernels: FS-Scala is equipped with a powerful abstract API for representing kernel functions. The module has two abstract classes to outline the behaviors of kernels used in SVM based applications as well as density estimation. The library comes bundled with an implementation for AFE as well as for common SVM kernels i.e. Linear, Radial Basis Function (RBF), Polynomial, Laplace, Exponential. New kernel functions can be easily added to the library by extending the base classes in this module.
- Evaluation Metrics: We have implemented evaluation metrics for Binary Classification and Regression problems. Further more, the implementation of binary classification performance expressed as the area under Receiver Operating Characteristic (ROC), is carried out using *MapReduce* in a *single pass* fashion through the evaluation data points, which can be seen in algorithm 4. Calculating the area under the ROC curve in a *single pass* fashion greatly increases the speed of the eventual FS-LSSVM source code.
- Miscellaneous Utilities: This module contains code to carry out auxiliary tasks for model learning and optimization. It contains the implementation of entropy calculation, summary statistics, prototype selection as well as a set of various functions which can be required for implementing new model classes using the library.

The FS-Scala software is available at [22].

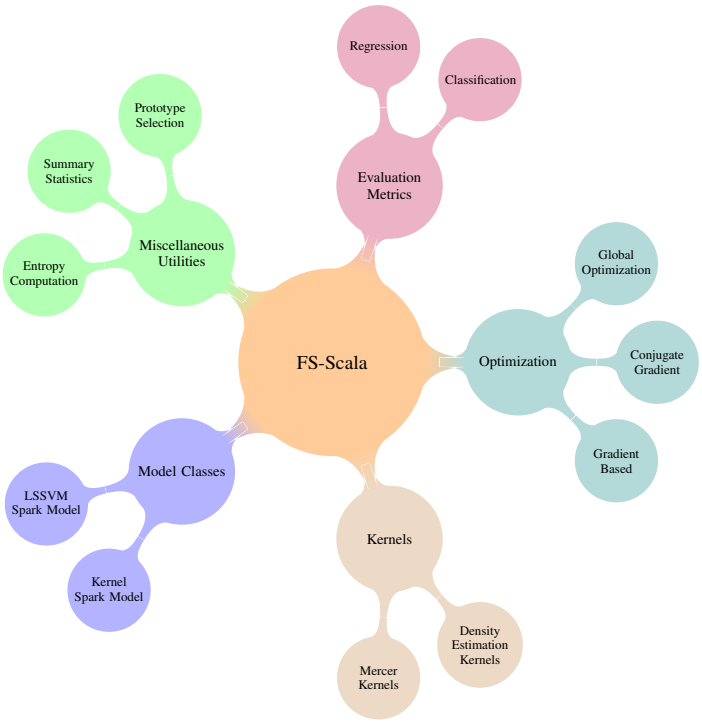


Fig. 1. Schematic structure of FS-Scala

Map Reduce

As discussed above, we use *MapReduce* wherever possible in order to distribute the workload using *Apache Spark*. Due to the primal formulation of the FS-LSSVM, the size of matrix $A = \hat{\Phi}_e^\intercal \hat{\Phi}_e + \frac{I_{m+1}}{\gamma}$, in the linear system in (3) is $(m+1) \times (m+1)$, where m is the number of prototypes selected to construct the kernel matrix in kernel based FS-LSSVM. Procedure 3 outlines the procedure to estimate the parameters $\hat{w}$, $\hat{b}$ of the FS-LSSVM model discussed in section II-B, using the *Conjugate Gradient* algorithm.

Using *MapReduce*, we calculate $A = \hat{\Phi}_e^\intercal \hat{\Phi}_e + \frac{I_{m+1}}{\gamma}$ and $\hat{\Phi}_e^\intercal Y$. We use these results to carry out iterations of the Conjugate Gradient updates until the maximum number of iterations is reached.

Optimization/Hyper-parameter tuning

Figure 2 depicts the class hierarchy structure of the Optimization module of FS-Scala. The FS-LSSVM model class has an embedded optimization object which it inherits from the *Optimizer* interface. Implementations of Conjugate Gradient and Gradient Descent are provided in the optimization module. New optimization algorithms can be added by inheriting from the top level *Optimizer* interface or the *RegularizedOptimizer* abstract class in case one is working with parametric models which involve regularization. Another important component of the optimization module is the *GlobalOptimizer* interface which acts as a skeleton for implementing gradient free global optimization algorithms. We have implemented simple Grid Search and CSA, for tuning kernel based models. CSA as proposed by De Souza et al. [17] creates a grid (simplex) of

Algorithm 2: Calculate feature matrices from data using MapReduce: $FeatureMat$

Data: $X = [x^i],\ x^i \in \mathbf{R}^n,\ \hat{\phi} : \mathbf{R}^n \longrightarrow \mathbf{R}^m,$
$Y = [y^i], y^i \epsilon \mathbf{R}$
Result: $\left(\hat{\Phi}_e^\intercal \hat{\Phi}_e\right),\ \hat{\Phi}_e^\intercal Y$
begin
  $MapFn(x, y)$:
    $M \longleftarrow \hat{\phi}(x)\hat{\phi}(x)^T$
    $v \longleftarrow \hat{\phi}(x)\ y$
  $emit(M, v)$
begin
  $RedFn((M, v), (M', v'))$:
  $emit(M + M', v + v')$
begin
  $(F, v) \longleftarrow MapReduce(X, MapFn, RedFn)$
  return (F, v)

Algorithm 3: Conjugate Gradient: CG

Data: $X = [x^i],\ x^i \in \mathbf{R}^n,\ \hat{\phi} : \mathbf{R}^n \longrightarrow \mathbf{R}^m,$
$Y = [y^i], y^i \epsilon \mathbf{R},\ \gamma,\ \epsilon$
Result: $\begin{pmatrix} \hat{w} \\ \hat{b} \end{pmatrix} = \left(\hat{\Phi}_e^\intercal \hat{\Phi}_e + \frac{I_{m+1}}{\gamma}\right)^{-1} \hat{\Phi}_e^\intercal Y$
begin
  $(F, v) \longleftarrow FeatureMat(X, Y, \hat{\phi}, \gamma)$
  $A \longleftarrow F + \frac{1}{\gamma}\mathbf{I}_{m \times m}$
  $(\hat{w}, \hat{b}) \longleftarrow CG(\hat{w}_i, \hat{b}_i, A, v)$

hyper-parameter values and treats each point as a Simulated Annealing (SA) process.

IV. Experiments

The experiments are performed on a 40 core 64GB RAM machine at the Department of Electrical Engineering, KU Leuven. The experiment parameters are summarized in table II. We use the *Magic Gamma* Telescope, *Adult*, *Forest Cover Type* and *Higgs Susy* data sets available from the UCI Machine Learning Repository [23].

- Magic Gamma: The data is generated by the registration of high speed gamma particles measured by a ground based atmospheric Cherenkov gamma telescope. Each entry consists of 10 numerical attributes and a binary class attribute.
- Adult: This is based on a a census study carried out in 1994, the data consists of 6 numerical attributes and 8 categorical attributes. The target attribute is binary class value, which indicates if the given individual has an annual income more than 50000\$.
- Forest Cover Type: A data set which consists of observations of cartographic variables on 30×30 metre cells at various locations with the task of predicting the forest type in the respective cells. This is a multi-class classification problem. This is transformed to a binary classification task of distinguishing forest type 2 from the others.
- Higgs Susy: A classification data set where one must distinguish between a signal process which produces super-symmetric particles and a background process which does not.

Algorithm 4: Evaluate performance for fold: $evaluateFold$

Data: $X_f = [x^i],\ x^i \in \mathbf{R}^n,\ \hat{\phi} : \mathbf{R}^n \longrightarrow \mathbf{R}^m,$
$Y_f = [y^i], y^i \in \mathbf{R},\ \hat{w},\ \hat{b}.$
Result: score for given fold
begin
  $predictLabel(\hat{w}, \hat{b})(x, y)$:
  $emit(\hat{w} \cdot x + \hat{b}, y)$
begin
  $Vector.fill(length)(IndicatorFn)$:
  $vec \longleftarrow (0, ..., 0)_{length}\ map(IndicatorFn)$
  $return(vec)$
begin
  $MapScore(score, label)$:
  if *label = 1.0* **then**
    $Pos \longleftarrow Pos + 1$
    $tpv \longleftarrow$
    $Vector.fill(l)(IndicatorFn(sign(score - thresholds(i)) == 1.0))$
    $fpv \longleftarrow Vector.fill(l)(IndicatorFn(false))$
  else
    $Neg \longleftarrow Neg + 1$
    $tpv \longleftarrow Vector.fill(l)(IndicatorFn(false))$
    $fpv \longleftarrow$
    $Vector.fill(l)(IndicatorFn(sign(score - thresholds(i)) == 1.0))$
  $emit(tpv, fpv)$
begin
  $RedScore((u, v), (u', v'))$:
  $emit(u + u', v + v')$
begin
  $thresholds \longleftarrow List(t_1, t_2, \ldots t_l)$
  $Pos \longleftarrow 0$
  $Neg \longleftarrow 0$
  $scoresLabels \longleftarrow$
  $(X_f, Y_f)\ map\ predictLabel(\hat{w}, \hat{b})$
  $(tp, fp) \longleftarrow$
  $scoresLabels\ map(MapScore)\ reduce(RedScore)$
  $tp \longleftarrow tp/Pos\ fp \longleftarrow fp/Neg$
  $roc \longleftarrow thresholds\ zip(tp\ zip\ fp)$
  return $1 - area(roc)$

Linear and kernel based FS-LSSVM models are trained, tuned and tested for various values of the experimental parameters shown in table II, each set of experiments is repeated four times, the mean and standard deviation of the classification accuracy are recorded and presented in figures IV, IV, 5 and 6. Note that the diameters of the circles in the figures indicate the standard deviation of the respective accuracy value over

TABLE I. METADATA

Data Set	Training samples	Test Samples	Features
Magic Gamma	18792	228	10
Adult	29310	3251	13
Forest Cover	523076	57936	53
Higgs Susy	4500000	500000	18

TABLE II. EXPERIMENT PARAMETERS

Name	Meaning	Values
Kernel	The type of kernel used	RBF, Polynomial, Laplacian, Exponential, Linear
Prototypes	Size of prototype set	50, 100, 200, N.A (Linear)
Global Opt.	Hyper-parameter optimization algorithm	gs: Grid Search, csa: Coupled Simulated Annealing
Grid Size	Number of points (per hyper-parameter) in the grid	2,3,4
Accuracy	avg. measure of classification accuracy	-

Algorithm 5: Distributed v-Fold Cross-Validation

Data: $X = [x^i],\ x^i \epsilon \mathbf{R}^n$, $\hat{\phi} : \mathbf{R}^n \longrightarrow \mathbf{R}^m$, $Y = [y^i], y^i \epsilon \mathbf{R}$, γ, folds
Result: Cross Validation Performance
begin
  $(A, v) \longleftarrow FeatureMat(X, Y, \hat{\phi}, \gamma)$
  $score \longleftarrow 0$
  for $i \longleftarrow 1\ to\ folds$ **do**
    $(X_i, Y_i) \longleftarrow$ fold i
    $(A_i, v_i) \longleftarrow FeatureMat(X_i, Y_i, \hat{\phi}, \gamma)$
    $(\hat{w}, \hat{b}) \longleftarrow CG(A - A_i + \frac{1}{\gamma}\mathbf{I}_{m\times m}, v - v_i)$
    $score \longleftarrow score + evaluateFold(\hat{w}, \hat{b}, X_i, Y_i)$
  return $score/folds$

the trials.

The performance of binary FS-LSSVM classifiers on the *MAGIC Gamma* Telescope Data Set obtained from the UCI Machine Learning Repository, are summarized in figure IV. FS-LSSVM models trained with polynomial kernels give better classification performance than the RBF and Linear counterparts, on the *MAGIC Gamma* data.

Results on the *Adult* Data Set, are summarized in figure IV. FS-LSSVM models trained with exponential kernels give better classification performance than the RBF and Linear counterparts, on the *Adult* data. For both the data sets one sees a pattern emerging that tuning kernel models with CSA gives better results than naive *Grid Search* based hyper-parameter optimization.

A. Execution time and Scalability

In figures 7 and 8 the execution time taken for model tuning and testing while using *FS-Scala* on the *Forest Cover Type* and *Higgs SUSY* data sets respectively. We can compare the execution time taken for model tuning on *Forest Cover Type* with corresponding results of the *LSSVMLab* software used in [9]. It is observed that *FS-Scala* gives an $80\% - 90\%$ improvement in execution time as compared to *LSSVMLab* on the *Forest Cover Type* data.

V. CONCLUSION

In this paper we propose *FS-Scala*, a Scala-based implementation of kernel based FS-LSSVM models, which is capable of scaling the the learning process to large data sets. As a use case, the kernel based FS-LSSVM model is tested on benchmark data sets. We observed that our implementation enables a substantial speed up over the existing FS-LSSVM implementations, while still providing flexibility to tweak various underlying data processing infrastructure.

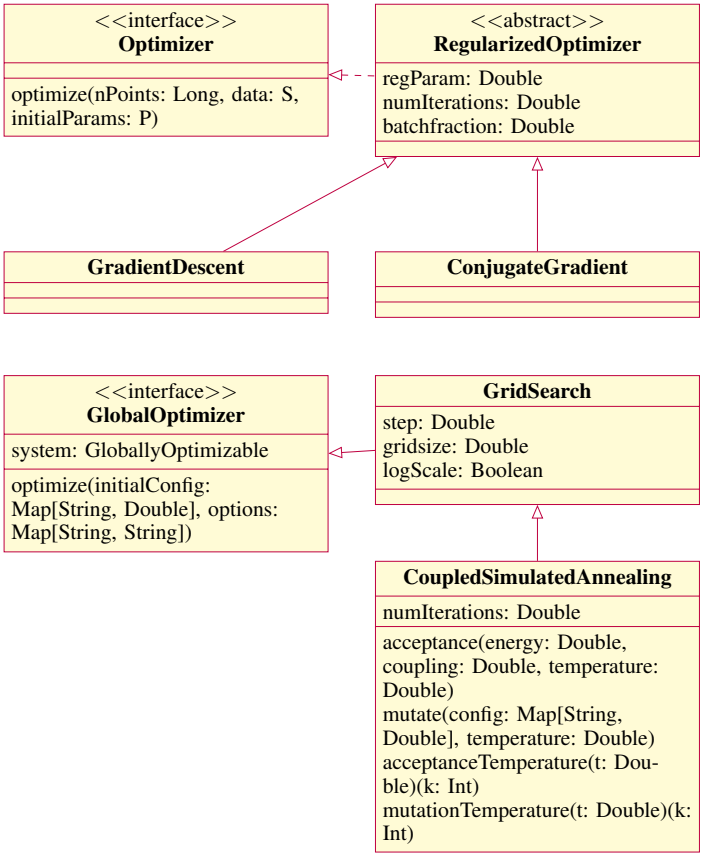


Fig. 2. Class Hierarchy of Optimization API

ACKNOWLEDGMENT

This work was supported by EU: ERC AdG A-DATADRIVE-B (290923), Research Council KUL: GOA/10/-/09 MaNet , CoE PFV/10/002 (OPTEC), BIL12/11T; PhD/Postdoc grants-Flemish Government; FWO: projects: G.0377.12 (Structured systems), G.088114N (Tensor based data similarity); PhD/Postdoc grants; IWT: projects: SBO POM (100031); PhD/Postdoc grants; iMinds Medical Infor-

Fig. 3. Magic Gamma Telescope: Comparison of Accuracy CSA vs GS

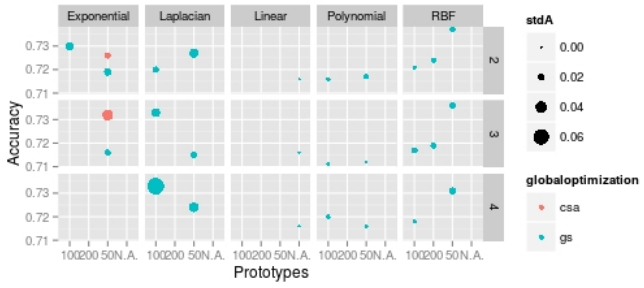


Fig. 4. Adult: Comparing Accuracy across kernels

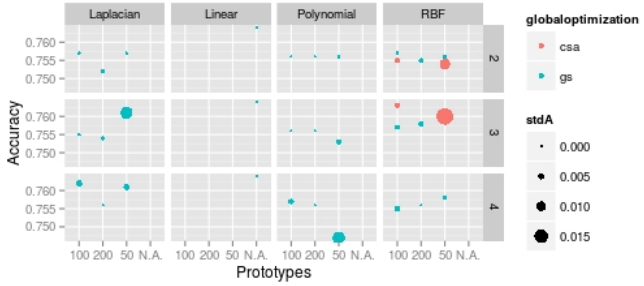


Fig. 5. Forest Cover Type: Comparison between kernel and linear LSSVM models

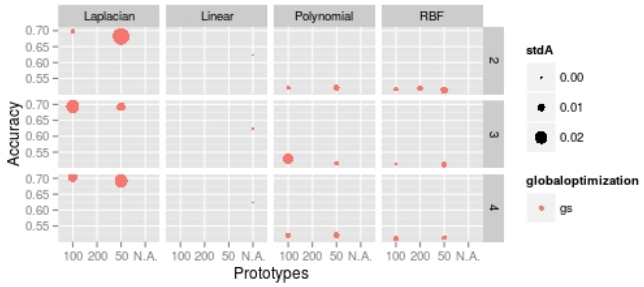
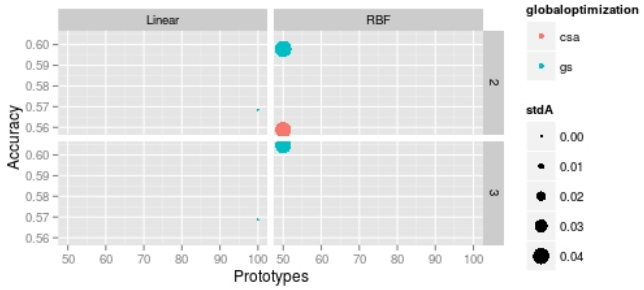


Fig. 6. Higgs Susy: Performance of LSSVM models



mation Technologies SBO 2014 and 2015-Belgian Federal Science Policy Office: IUAP P7/19 (DYSCO, Dynamical systems, control and optimization, 2012-2017).

REFERENCES

[1] "Apache hadoop: Lightning-fast cluster computing," 2005 (accessed July 6, 2015). [Online]. Available: http://hadoop.apache.org/

[2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 1–26, 2008.

[3] D. Borthakur, S. Rash, R. Schmidt, A. Aiyer, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, and A. Menon, "Apache hadoop goes realtime at Facebook," *SIGMOD '11 - Proceedings of the 2011 international conference on Management of data*, p. 1071, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1989323.1989438

[4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark : Cluster Computing with Working Sets," *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, p. 10, 2010.

[5] "Apache spark: Lightning-fast cluster computing," 2010 (accessed July 6, 2015). [Online]. Available: http://http://spark.apache.org/

[6] A. Halevy, P. Norvig, and F. Pereira, "The unreasonable effectiveness of data," *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8–12, 2009.

[7] G. Valentini and T. G. Dietterich, "Bias-variance analysis of support vector machines for the development of svm-based ensemble methods," *J. Mach. Learn. Res.*, vol. 5, pp. 725–775, Dec. 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=1005332.1016783

[8] X. Meng, J. K. Bradley, B. Yavuz, E. R. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "MLlib : Machine Learning in Apache Spark," *CoRR*, vol. abs/1505.06807, 2015. [Online]. Available: http://arxiv.org/abs/1505.06807

[9] 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 and Data Analysis*, vol. 54, no. 6, pp. 1484–1504, Jun. 2010. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167947310000393

[10] J. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, *Least Squares Support Vector Machines*. World Scientific, 2002.

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

[12] ——, "Sparse Reductions for Fixed-Size Least Squares Support Vector Machines on Large Scale Data," in *Proc. of 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2013)*, 2013, pp. 161–173.

[13] R. Mall, V. Jumutc, R. Langone, and J. A. K. Suykens, "Representative subsets for big data learning using k-NN graphs," in *Proc. of IEEE BigData*, 2014, pp. 37–42.

[14] R. Mall, R. Langone, and J. A. K. Suykens, "FURS: Fast and Unique Representative Subset selection retaining large scale community structure," *Social Network Analysis and Mining*, vol. 3, no. 4, pp. 1075–1095, 2013.

[15] J. A. K. Suykens and J. Vandewalle, "Least Squares Support Vector Machine Classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[16] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, Jan. 1965. [Online]. Available: http://comjnl.oxfordjournals.org/cgi/content/long/7/4/308

[17] S. Xavier-De-Souza, J. A. K. Suykens, J. Vandewalle, and D. Bolle, "Coupled simulated annealing," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 40, no. 2, pp. 320–335, 2010.

[18] M. Odersky and al., "An overview of the scala programming language," EPFL Lausanne, Switzerland, Tech. Rep. IC/2004/64, 2004.

[19] "Titan: Distributed graph database," 2014 (accessed July 6, 2015). [Online]. Available: http://thinkaurelius.github.io/titan/

[20] "Orientdb," 2010 (accessed July 6, 2015). [Online]. Available: http://orientdb.com/orientdb/

[21] "Neo4j: The worlds leading graph database," 2007 (accessed July 6, 2015). [Online]. Available: http://neo4j.com/

[22] "Fs-scala: Apache spark implementation of fixed size least squares support vector machines," 2015 (accessed July 12, 2015). [Online]. Available: https://github.com/mandar2812/FS-Scala.git

Fig. 7. Forest Cover Type: Execution time of FS-Scala FS-LSSVM implementation (training, tuning and test)

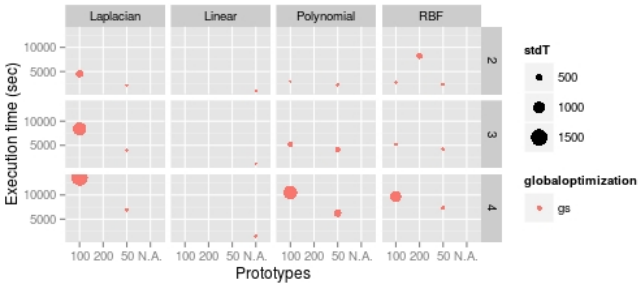
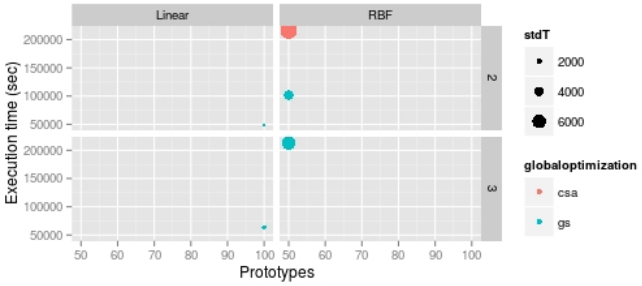


Fig. 8. Forest Cover Type: Execution time of FS-Scala FS-LSSVM implementation (training, tuning and test)



[23] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml