Check for updates

Received: 18 November 2019 | Revised: 7 January 2021 | Accepted: 14 February 2021

DOI: 10.1002/nla.2371

RESEARCH ARTICLE

WILEY

Two-level preconditioning for Ridge Regression

Joris Tavernier[1,2] | **Jaak Simm**[2] | **Karl Meerbergen**[1] | **Yves Moreau**[2]

[1]Departement of Computer Science, NUMA, KU Leuven, Belgium

[2]Department of Electrical Engineering, ESAT - STADIUS, KU Leuven, Belgium

Correspondence
Joris Tavernier, Departement of Computer Science, NUMA, KU Leuven, Belgium.
Email: Joris.Tavernier@kuleuven.be

Funding information
Fonds Wetenschappelijk Onderzoek, Grant/Award Number: G079016

Abstract

Solving linear systems is often the computational bottleneck in real-life problems. Iterative solvers are the only option due to the complexity of direct algorithms or because the system matrix is not explicitly known. Here, we develop a two-level preconditioner for regularized least squares linear systems involving a feature or data matrix. Variants of this linear system may appear in machine learning applications, such as ridge regression, logistic regression, support vector machines and Bayesian regression. We use clustering algorithms to create a coarser level that preserves the principal components of the covariance or Gram matrix. This coarser level approximates the dominant eigenvectors and is used to build a subspace preconditioner accelerating the Conjugate Gradient method. We observed speed-ups for artificial and real-life data.

KEYWORDS

clustering, Krylov subspace methods, large-scale, machine learning, preconditioning, ridge regression, Tikhonov regularization

1 | INTRODUCTION

The computational bottleneck in machine learning applications is often solving a linear system. These systems are usually solved iteratively due to the complexity of direct algorithms. Here we focus on problems from machine learning with variations of a least squares linear system leading to the regularized normal equations

$$(X^TX + \beta I)\mathbf{w} = X^T\mathbf{b}, \tag{1}$$

with β a given regularization parameter. The feature matrix $X \in \mathbb{R}^{N\times F}$ consists of N data samples with F features. This matrix is often high dimensional and sparse. Hence, Krylov subspace methods are appealing, since they only require the matrix-vector multiplication and not the matrix itself. For arbitrary sparse feature matrices X, the matrix product X^TX in (1) is likely to be much less sparse than X. We focus on underdetermined systems with $N < F$, referred to as small sample problems. This results in ill-posed linear systems with a nontrivial null space $\mathcal{N}(X^TX) \neq \emptyset$.

The number of iterations required to reach a certain tolerance for Krylov subspace methods depends on the spectrum of the matrix, more specifically the distribution of the eigenvalues.[1,2] The condition number of the matrix is often an adequate indication. The art of preconditioning aims at reducing the number of iterations by transforming the problem to an easier one. One simple approach is to divide the matrix by its main diagonal and is called Jacobi preconditioning.[1,3] Diagonal or Jacobi preconditioning is often very efficient if the matrix is diagonally dominant. Other existing preconditioning techniques such as incomplete Cholesky factorization[4,5] or incomplete LU factorization[6] are useless since the matrix product in our linear system is often close to dense. It is possible that the covariance matrix or Gram matrix X^TX follows some special structure for which the inverse is sparse. In this case, the Sparse Approximate Inverse[7] can be computed based on this sparse structure. However, in general the inverse of the covariance matrix is dense.

 © 2021 John Wiley & Sons, Ltd.

For the solution of partial differential equations (PDEs), geometric Multigrid,[8] and its algebraic extensions are often successful.[9-11] The crucial concept is that the eigenvectors that slow down the convergence of iterative methods, can be accurately represented on a coarser grid. The components of these eigenvectors are then eliminated on coarser grids.[12-14] Currently, the most generic preconditioners are variants of bootstrap AMG.[15,16] Bootstrap AMG designs the hierarchy of grids such that a certain set of slow converging eigenvectors are exactly interpolated from the coarser grids. Bootstrap additionally exploits the existing hierarchy of grids to improve this set of slow converging eigenvectors. In this way, the hierarchy of grids is recursively improved.

The multilevel idea was also developed for the Fredholm integral equations and the resulting ill-posed problems.[17] The unregularized problem is solved by applying the Conjugate Gradient method (CG) for the normal equations on different levels. Regularization is applied by restricting the number of iterations on the different levels. In addition, principal eigenvectors can be used for preconditioning LSQR for ill-posed problems.[18,19] For PDEs, multilevel preconditioning exploits the underlying idea that the operator is based on a discretized grid, often resulting in M-matrices. As for the Fredholm integral equations, the operator is based on a discretization of continuous functions on a grid. Data matrices, however, do not have this property.

Randomized algorithms such as stochastic gradient descent are popular for large-scale optimization problems in data sciences.[20-22] These randomized algorithms have been used effectively for least-squares solves of sparse, well-conditioned matrices.[23] For an overview of randomized iterative algorithms for linear systems, see Reference 24 and the references therein.

In machine learning, several linear systems of the form (1) may have to be solved for one application. Firstly, the regularization parameter β in (1) is often unknown and several values are tried in a cross-validation process. Secondly, different right-hand sides are often available or the linear system to be solved is part of a sampling algorithm. This means that even high computational costs for the construction of the preconditioner is often negligible compared to the many solves.

For applications from data science, state-of-the-art multilevel methods cannot be used in their standard form. As opposed to grid-based operators in PDEs, the matrices consist of samples and features and a hierarchy has to be designed on the data level. The resulting linear operator is a product of matrices and often dense. This means that the operator itself is not always available due to memory restrictions. There is no natural way to design the level hierarchy and the system matrix may not be available to design the coarse level directly from the operator. We thus propose to build a multilevel preconditioner based on the Multigrid idea by clustering the features of the data matrix, resulting in coarser levels. Figure 1 shows an example for dense matrices with clustering for both rows and columns. This can be done such

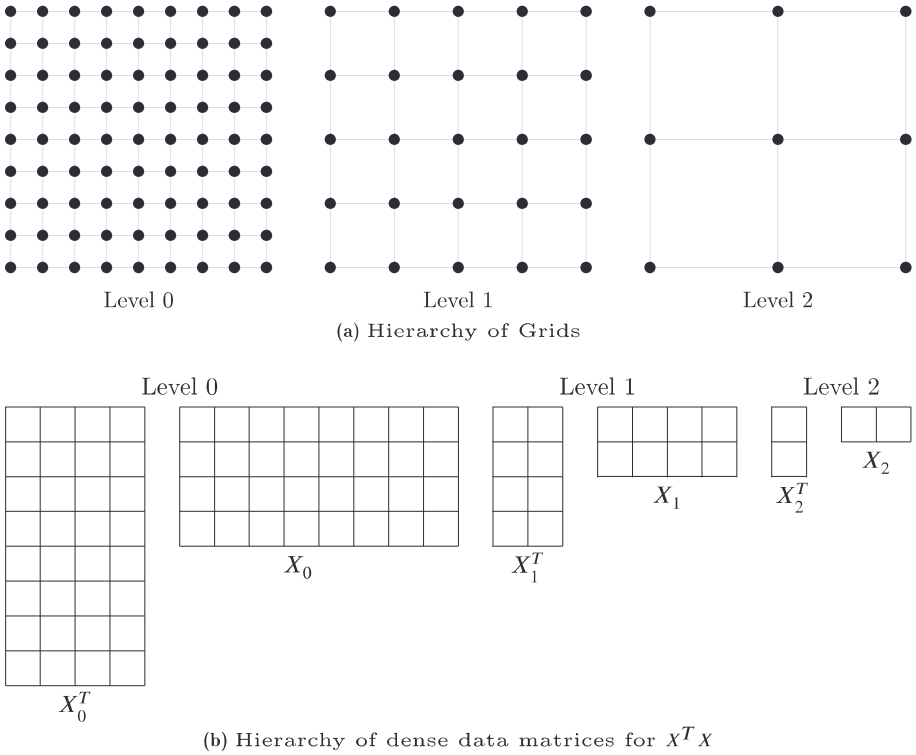


FIGURE 1 Examples of the level hierarchies in partial differential equations (PDEs) and data. (a) The hierarchy of grids for the discretization of a PDE. (b) The hierarchy of levels for X^tX using a data matrix X and clustering both the rows and columns

that the principal eigenvectors are represented on the coarser levels. In practice, it is however more efficient to directly project to the coarsest level and only use the finest and the coarsest level and design a two-level preconditioner.

The paper is organized as follows. Section 2 describes different algorithms from machine learning and identifies the linear systems to be solved. Next, Section 3 describes the two-level preconditioner. In Section 4, the numerical results are presented. Finally, conclusions are given in Section 5.

2 | APPLICATIONS

Data in machine learning consists of data samples and their features. Feature matrices can take any form, from dense or sparse and binary or real valued. The sample characteristics $\mathbf{x}_i$ are collected in a feature matrix $X \in \mathbb{R}^{N\times F}$ with N the number of samples or data points and F the number of features. Using the collected feature matrix, one tries to predict certain properties for new and unknown samples. These properties can be real valued and then the task at hand is regression or they can be limited to certain classes and then the task is classification. In this section, we will describe some techniques from machine learning where the linear system (1) is the main computational bottleneck, with β the regularization parameter. The techniques described here are mainly intended as illustrations and a comprehensive understanding is not required for the remaining sections. These applications require several linear systems with different right-hand sides and justify the cost of more computational intensive preconditioners such as the preconditioner proposed in Section 3.

2.1 | Ridge regression

Suppose we have more samples than features resulting in an overdetermined system

$$\mathbf{b} = X\mathbf{w} + \mathbf{e},$$

with $\mathbf{b}$ the property of interest for the samples.[25] The ridge regression solution is determined by minimizing the squared error on $\mathbf{b}$ ($\mathbf{e} = X\mathbf{w} - \mathbf{b}$). Adding regularization to the solution vector $\mathbf{w}$, the optimization problem is given by

$$\min_{\mathbf{w}} \mathbf{e}^T\mathbf{e} + \beta\mathbf{w}^T\mathbf{w},$$

and the solution is

$$\mathbf{w} = (X^TX + \beta I)^{-1}X^T\mathbf{b}. \tag{2}$$

This technique is known in the field of numerical linear algebra as Tikhonov regularization for linear systems. The value of the regularization parameter β is unknown and several values are tried in a cross-validation process.

2.2 | Bayesian regression

In ordinary ridge regression[25,26] the residuals $\mathbf{e}$ are assumed uncorrelated and normally distributed in (2.1), for example, $\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \tau^{-1}I_N)$ with I_N the identity matrix of size $\mathbb{R}^{N\times N}$ and $\tau > 0$ the precision parameter. In Bayesian learning, the observations $\mathbf{b}$ are considered probabilistic

$$p(\mathbf{b}|X, \mathbf{w}) = \prod_{i=1}^{N} \mathcal{N}(y_i|\mathbf{x}_i\mathbf{w}, \tau^{-1}),$$

with $\mathbf{x}_i \in \mathbb{R}^{1\times F}$ the features of sample i. The conditional posterior distribution of $\mathbf{w}$ given τ, X, β and $\mathbf{b}$ is then

$$p(\mathbf{w}|\tau, \mathbf{b}, X) \sim \mathcal{N}((X^TX + \beta I_F)^{-1}X^T\mathbf{b}, ((X^TX + \beta I_F)\tau)^{-1}),$$

with β a regularization parameter. Computing the covariance matrix $(X^TX + \beta I_F)\tau$ is inconvenient and often computationally infeasible. Simm et al.[27] however, describe a Gibbs sampler for Bayesian regression and using their noise injection trick, a sample of $\mathbf{w}$ can be taken by solving

$$\left(X^TX + \frac{\lambda}{\tau}I_F\right)\mathbf{w} = X^T(\mathbf{y} + \mathbf{e}_1) + \frac{\mathbf{e}_2}{\tau}, \tag{3}$$

with $\mathbf{e}_1 \sim \mathcal{N}(\mathbf{0}, \tau^{-1}I_N)$ and $\mathbf{e}_2 \sim \mathcal{N}(\mathbf{0}, \lambda I_F)$. A zero mean and uncorrelated normal distribution is used as prior for $\mathbf{w}$:

$$p(\mathbf{w}, \lambda|\alpha, \beta) \sim \mathcal{N}(\mathbf{w}|\mathbf{0}, \lambda^{-1}I_F)\mathcal{G}(\lambda|\alpha, \beta),$$

with $\mathcal{G}(\lambda|\alpha, \beta)$ the gamma distribution as conjugate prior for the precision parameter $\lambda > 0$. The parameters λ and τ are sampled and the fraction $\frac{\lambda}{\tau}$ thus changes each iteration. The noise injection trick was originally developed for Bayesian Probabilistic Matrix Factorization[28] with side information as detailed in Reference 27.

2.3 | Optimization by the Newton method

For certain optimization problems

$$\min_{\mathbf{w}} f(w),$$

the solution can be found by the Newton method which requires the inverse of the Hessian $(\nabla^2 f(\mathbf{w}))$. The Newton approach takes, at iteration k, the step $\mathbf{s}_k = -(\nabla^2 f(\mathbf{w}_{k-1}))^{-1}\nabla f(\mathbf{w}_{k-1})$ and the iterative solution is updated as $\mathbf{w}_k = \mathbf{w}_{k-1} + \mathbf{s}_k$ with $\nabla f(\mathbf{w})$ the gradient. For large-scale data the inverse is not directly computed but a linear system is solved. Binary logistic regression[29] and L2 loss function Support Vector Machines[30] for example can be solved by the Newton Method and the Hessians

$$\nabla^2 f(\mathbf{w}) = X^T D(\mathbf{w})X + \beta I,$$

are variants of (1) with the system matrix a product and $D(\mathbf{w})$ a diagonal matrix. More details can be found in the paper by Lin et al.[31] and the references therein.

2.4 | Linear discriminant analysis

Dimension reduction techniques such as Principal Component Analysis (PCA) or Linear discriminant analysis (LDA) try to reduce the number of variables used in further analysis such as classification.[25] LDA is a supervised method that finds a linear mapping that maximally separates the classes while minimizing the within class variance. Defining the total-class-scatter matrix as $S_T = X^TX$ and the between-class-scatter matrix as S_B the LDA optimization problem is

$$\max_{\mathbf{w}\in\mathbb{R}^N} \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_T \mathbf{w}},$$

and can be solved by finding the eigenvectors corresponding to the largest eigenvalues of

$$S_B\mathbf{w}_i = \lambda_i(S_T + \beta I)\mathbf{w}_i,$$

with β the regularization parameter. The low-rank and structure of the between-class-scatter matrix can be exploited and the main computational bottleneck is solving the linear system

$$(X^TX + \beta I)\mathbf{w} = X^T\mathbf{b}, \tag{4}$$

with $\mathbf{b}$ determined by the sample labels as detailed in Reference 32.

3 | TWO-LEVEL PRECONDITIONER

The general idea of two-level preconditioning is to precondition a linear system from the solution of a linear system with less variables. This smaller linear system is obtained by projecting the large linear system on a subspace. Geometric Multigrid is used for the solution of elliptic partial differential equations. The discrete problem is solved hierarchically by approximating the original discretized PDE by coarser discretizations of the PDE.[8] The Multigrid principles were extended to Algebraic Multigrid using only the coefficients and sparsity pattern of the matrix instead of the geometry of the problem.[9-11]

3.1 | The idea of two-level iterations

The use of coarse grids for the solution of PDEs is motivated as follows. Many iterative schemes have poor convergence for PDEs. For a class of solvers, for example, Gauss–Seidel iteration, the low frequency and near kernel components of the solution converge very slowly. These solvers are called smoothers because the remaining error of the solution is typically smooth. The coarse grids are used to remove the low frequency components of the error in a more efficient way. When the PDE is discretized on a coarse grid, the low frequency components of the solution on the fine grid and thus the error can be approximated well on to the coarse grid. Generally, a two-level solver consists of the following steps. A small number of iterations is performed on the fine level and a smooth error remains. The residual is mapped on the coarse level and this is called restriction. The restriction operator R maps a (long) vector on the fine grid to a (short) vector on the coarse grid. The linear system on the coarse grid is solved exactly. The coarse solution is mapped back to the fine grid using a prolongation operator P. This uses interpolation to identify all the unknowns on the fine grid and leads to a solution on the fine grid that mainly has high frequency components in the error. These components can then be further reduced by additional smoothing steps.

For symmetric matrices, the restriction operator R is defined as the transpose of the interpolation operator P. Using the Galerkin approach the coarse level is defined by $A_1 = PA_0P^T$. Assuming one step of Richardson's iteration, with ω a nonnegative scalar, as smoother,[33] the resulting iteration matrix is

$$T = (I - \omega A_0)(I - PA_1^{-1}P^TA_0)(I - \omega A_0),$$

where the first factor is called postsmoothing, the third is presmoothing and the middle term is the coarse-grid correction. A more detailed explanation of (algebraic) Multigrid can be found in Reference 11 and the references therein.

3.2 | Data matrices

We want to adopt the idea of approximating error components on the coarse level and prolonging the coarse solution to the fine level. For matrices resulting from PDEs, the effect of aggregating elements is well known. In our case the matrix product of interest is X^TX with X a data matrix and this matrix product defines the covariance matrix.

The eigenvectors of the covariance matrix have a specific physical interpretation. A technique called PCA finds the eigenvectors corresponding to the largest eigenvalues of the covariance matrix. PCA is in essence a projection that maximizes the total variance.[25] PCA actually finds the directions or principal components that projects the data on a subspace maximally preserving the variance of the data. For small sample size problems, the number of features is larger than the number of samples, resulting in a nontrivial nullspace of X^TX. These eigenvectors are actually noise and contain no data information.

A widely used technique in machine learning is clustering. These clustering algorithms find subsets of similar samples or features based on a distance or similarity measure. Ideally, this results in subsets with small variance within one specific subset. Clustered subsets of features can be used to create aggregate features for the coarse level, preserving highest variance in the data set. Since highest variance is preserved in the coarse level, the principal components can be approximated on the coarse level.

Firstly, we propose to define an intuitive averaging restriction operator ($\tilde{P}^T$) based on the clusters and this operator will be improved later. Specifically, a coarse feature $\mathbf{c}_S$ consist of the average of the n_S features $X(:, i)$ in one cluster S and $i \in S$:

$$\mathbf{c}_S = 1/n_S \sum_{i \in S} X(:, i).$$

The resulting restriction operator has the value $1/n_S$ for each feature $i \in S$ in the row corresponding to S. Assuming that the matrix X is ordered such that $X = [X_1, X_2, \dots, X_{F_C}]$ with X_S the features belonging to cluster $S = 1, \dots, F_C$ and F_C the coarse feature dimension, the restriction operator is

$$\tilde{P}^T = \begin{bmatrix} \overbrace{\frac{1}{n_1} \quad \cdots \quad \frac{1}{n_1}}^{n_1} & \overbrace{\quad}^{n_2} & \cdots & \overbrace{\quad}^{n_{F_C}} \\ & \frac{1}{n_2} \quad \cdots \quad \frac{1}{n_2} & & \\ & & \ddots & \\ & & & \frac{1}{n_{F_C}} \quad \cdots \quad \frac{1}{n_{F_C}} \end{bmatrix} \tag{5}$$

Note that the product

$$\tilde{P}^T \tilde{P} = \begin{bmatrix} \frac{n_1}{n_1^2} & & & \\ & \frac{n_1}{n_2^2} & & \\ & & \ddots & \\ & & & \frac{n_{F_C}}{n_{F_C}^2} \end{bmatrix}, \tag{6}$$

and thus $\tilde{P}$ does not define a projection. The averaging interpolation operator can easily be adjusted such that the product $P^T P$ is the identity matrix by defining the coarse features as

$$\mathbf{c}_S = 1/\sqrt{n_S} \sum_{i \in S} X(:, i),$$

resulting in

$$P^T = \begin{bmatrix} \overbrace{\frac{1}{\sqrt{n_1}} \quad \cdots \quad \frac{1}{\sqrt{n_1}}}^{n_1} & \overbrace{\quad}^{n_2} & \cdots & \overbrace{\quad}^{n_{F_C}} \\ & \frac{1}{\sqrt{n_2}} \quad \cdots \quad \frac{1}{\sqrt{n_2}} & & \\ & & \ddots & \\ & & & \frac{1}{\sqrt{n_{F_C}}} \quad \cdots \quad \frac{1}{\sqrt{n_{F_C}}} \end{bmatrix} \tag{7}$$

Using the Galerkin approach[11] the coarse linear system becomes

$$\begin{aligned} P^T(X^T X + \beta I)P &= P^T X^T X P + P^T \beta I P \\ &= X_c^T X_c + \beta \underbrace{P^T P}_{=I}, \end{aligned}$$

with $X_c = XP$ the matrix with coarse features $\mathbf{c}_S$ and the regularization parameter on the coarse level is equal to the regularization parameter on the fine level.

With the coarse level defined, we can create our two-level preconditioner. In AMG the coarse grid correction aims to eliminate the near kernel eigenvectors while the smoothing takes care of the eigenvectors associated with the large

eigenvalues. In our approach, the problem is that both smoothing and coarse correction eliminate the error components of the principal eigenvectors. Therefore, we do not apply classical smoothing and recommend to use CG iterations as presmoother since CG reduces the norm of the error, but the norm of some of the individual components might increase.[34] CG is thus sometimes referred to as a rougher instead of a smoother.

Finally, the coarse size F_C strongly influences the performance of the preconditioner. The spectrum can be divided in three parts: the noise components with small eigenvalues, the principal components represented in the coarse level, and lastly the remaining components with eigenvalues in between the noise and the principal components. The coarse size determines the dimension of the principal components and the remaining eigenvectors with nonzero eigenvalues. Depending on the required tolerance on the solution, these remaining components could cause slow convergence and are only eliminated on the fine level. If this happens, there is a possibility that the residual lies in the nullspace of the restriction operator P^T and the principal components are not present in the residual. Therefore, at least one smoothing iteration is required, since only applying the coarse correction would result in a zero solution and cause the fine level Krylov solver to break down.

3.3 | The ideal case

In the ideal case, features within one cluster are identical. This means that the element values of the eigenvectors corresponding to the features within one cluster are equal. This places a condition on our approach and the clustering algorithms should collect features in clusters such that the element values, corresponding to the features of one cluster, of the eigenvectors with large eigenvalues do not vary significantly.

Definition 1 (Ideal dataset for clustering). The ideal dataset for clustering $X = [X_1, X_2, \dots, X_{F_C}]$ consists of sets of features $X_i \in \mathbb{R}^{N \times n_i}$ consisting of n_i duplicate features $\mathbf{x}_i$ for $i = 1 \dots F_C$.

Theorem 1 (Ideal clustering case). *Let X satisfy Definition 1, the F_C eigenvectors with nonzero eigenvalues of X^TX are constant within the clusters. The prolongation operator P defined in (7) defines a projection PP^T for these eigenvectors and the eigenvectors of X^TX with nonzero eigenvalues have a one to one correspondence with the eigenvectors of P^TX^TXP.*

Proof. Given $X = [X_1, X_2, \dots, X_{F_C}]$ the ideal dataset with X_i consisting of n_i duplicate features $\mathbf{x}_i$ for $i = 1 \dots F_C$. The matrix product X^TX has rank F_C and has the following block structure

$$X^TX = \begin{bmatrix} \mathbf{x}_1^T\mathbf{x}_1 \mathbb{1}^{n_1 \times n_1} & \mathbf{x}_1^T\mathbf{x}_2 \mathbb{1}^{n_1 \times n_2} & \dots & \mathbf{x}_1^T\mathbf{x}_{F_C} \mathbb{1}^{n_1 \times n_{F_C}} \\ \mathbf{x}_2^T\mathbf{x}_1 \mathbb{1}^{n_2 \times n_1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \mathbf{x}_{F_C}^T\mathbf{x}_1 \mathbb{1}^{n_{F_C} \times n_1} & \dots & & \mathbf{x}_{F_C}^T\mathbf{x}_{F_C} \mathbb{1}^{n_{F_C} \times n_{F_C}} \end{bmatrix}. \tag{8}$$

The matrix product X^TX can be written as

$$X^TX = PKP^T, \tag{9}$$

with

$$K = \begin{bmatrix} \frac{\mathbf{x}_1^T\mathbf{x}_1}{n_1} & \frac{\mathbf{x}_1^T\mathbf{x}_2}{\sqrt{n_1}\sqrt{n_2}} & \dots & \frac{\mathbf{x}_1^T\mathbf{x}_{F_C}}{\sqrt{n_1}\sqrt{n_{F_C}}} \\ \frac{\mathbf{x}_2^T\mathbf{x}_1}{\sqrt{n_2}\sqrt{n_1}} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{\mathbf{x}_{F_C}^T\mathbf{x}_1}{\sqrt{n_{F_C}}\sqrt{n_1}} & \dots & & \frac{\mathbf{x}_{F_C}^T\mathbf{x}_{F_C}}{n_{F_C}} \end{bmatrix}. \tag{10}$$

Using the eigendecomposition of $K = \hat{V}\Lambda\hat{V}^T$, we have

$$X^TX = P\hat{V}\Lambda\hat{V}^TP^T. \tag{11}$$

By the structure of P, the element values of the eigenvectors with nonzero eigenvalues $P\hat{\mathbf{v}}_i = \mathbf{v}_i$ are the same within each cluster. Given $P\hat{\mathbf{v}}_i = \mathbf{v}_i$, we have that $PP^T\mathbf{v}_i = P\hat{\mathbf{v}}_i = \mathbf{v}_i$ and the interpolation operator (7) defines a projection for the eigenvectors with constant values within the clusters. Using the Rayleigh–Ritz method, the eigenpairs $(\lambda_i, \hat{\mathbf{v}}_i)$ of P^TX^TXP can be used to find the original eigenpairs $(\lambda_i, P\hat{\mathbf{v}}_i = \mathbf{v}_i)$. ■

3.4 | Clustering

For matrices originating from grids, the coarser levels are defined by aggregating grid elements based on different criteria. Recall that for our applications, the matrix consists of samples with features. Similarity between features can be used to cluster these features. Typically a predefined distance measure d is used to define similarity, widely used distance measures are Euclidean distance, Jaccard distance and the cosine distance. All features within one cluster are then similar to each other and can be represented by the average feature. In this section we look at three clustering algorithms and one sub-sampling technique to create a coarser level.

3.4.1 | Leader–follower clustering

Leader–follower clustering is given in Algorithm 1.[35] Starting from an initial set of leading clusters, each sample is added to an existing cluster if the sample is similar within the required tolerance to the cluster leader. This cluster leader can then optionally be updated. If a sample is not similar enough to any existing leader, it forms a new cluster.

Algorithm 1. Leader–follower clustering

Initialize: given an initial set of clusters $C = \{\mathbf{c}_s\}$ and a distance measure d
for each sample $\mathbf{x}_i$ in X **do**
  index $= \arg\min_{s|\mathbf{c}_s \in C} d(\mathbf{x}_i, \mathbf{c}_s)$
  if $d(\mathbf{x}_i, \mathbf{c}_{\text{index}}) <$ Tolerance **then**
    Add $\mathbf{x}_i$ as follower to the cluster leader $\mathbf{c}_{\text{index}}$ and update number of followers $n_{\text{index}} = n_{\text{index}} + 1$
    Optionally update the cluster leader: $\mathbf{c}_{\text{index}} = \mathbf{c}_{\text{index}} + \frac{\mathbf{x}_i - \mathbf{c}_{\text{index}}}{n_{\text{index}}}$
  else
    Add $\mathbf{x}_i$ as cluster leader: $C = C + \{\mathbf{x}_i\}$
  end if
end for

By varying the tolerance more or less clusters are found by Leader–Follower clustering. Small tolerance gives more clusters while high tolerance produces a smaller number of clusters. For our experiments we do not update the leader and preserve sparsity of the leaders if a new sample is added to his followers.

3.4.2 | K-means++

Another option to perform clustering on the features is K-means++. The algorithm consists of two phases.[36] During the setup phase, K-means++ finds initial K prototypes based on the distance between features. These prototypes can be considered as the leaders for their cluster. The first prototype is chosen at random with the features uniformly distributed. The next initial prototype is randomly chosen with a probability weighted by the distance squared of each data point to the existing prototype.

The K-means++ algorithm then proceeds to the next phase by assigning each point to a cluster and then updating the prototypes according to its assigned cluster points. Once the prototypes are updated, the data points are reassigned. When the assignments no longer change, the algorithm has converged.

3.4.3 | Quadratic Rényi-entropy-based subsampling

Defining the coarser level can be done by maximizing the entropy of the coarser level or subset. Rényi-entropy[37,38] based subsampling for large-scale data starts from a working set and a training set.[39] The entropy of the working set is the decisive parameter and the elements of the working set define the coarse level. Two random data points are chosen from both sets and the Rényi-entropy is calculated for both sets. The chosen points from both sets are swapped and the Rényi-entropy is recalculated for both sets. If there is an increase in entropy for the working set, the data points are switched else the original sets are maintained.

The quadratic Rényi-entropy for a subset of size F_C is given by

$$S_{R2}^{F_C} = -\log\left(\frac{1}{F_C^2|D|^2}\sum_{k=1}^{F_C}\sum_{l=1}^{F_C}\kappa\left(\frac{\mathbf{x}_k - \mathbf{x}_l}{D\sqrt{2}}\right)\right),$$

where κ is a kernel function and D the diagonal of the bandwidth values of the kernel in each dimension. Note that for each pair of data points under investigation the entropy can be updated based on both data points and there is no need to recalculate the entropy completely. These chosen features act then as the prototypes for the clusters and the remaining features are assigned to the cluster of the closest prototype.

3.4.4 | Graph aggregation

Clustering of graphs has been used to find communities in networks.[40-43] A graph $G = (V, E)$ consists of a set of nodes V and edges E connecting two nodes. A symmetric matrix A can be represented by its undirected adjacency graph G. For data matrices X, computing the product X^TX will result in a symmetric matrix. The adjacency graph for this matrix-product results in a very dense graph, since each feature is highly probable to be connected to a large number of features. Constructing a k-nearest neighbor graph, however, results in a sparse graph connecting each feature to its k-closest features. Creating a k-nearest neighbor graph can be computationally intensive but an approximate graph can be computed efficiently.[44] Using only the innerproducts between the nearest neighbors results in a sparse system matrix approximating the original symmetric and positive definite system matrix ($X^TX + \beta I$). Graph matching has been used for AMG.[45-47] We consider the half-approximate matching as detailed in BootCMatch.[47] This is one of the more recent and generally applicable AMG methods available. To apply this aggregation technique to our application, the distance between two features is used as edge weight. The detailed greedy half approximate matching algorithm[47] is adjusted to consider the minimum distance as the dominant edge.

3.5 | Least-squares interpolation

We assume that the coarse level is able to approximate the principal components. For AMG, it is possible to define the interpolation operator P to exactly interpolate certain prototype vectors in a least-squares manner.[15,48] Suppose we have the K first principal components $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_K\}$ and $V \in \mathbb{R}^{F\times K}$. Least squares interpolation minimizes the error squared, made when interpolating these principal components, for each feature i and thus defines each row $P(i, :)$

$$P(i,:) = \arg\min_{p_{ij}, j\in C_i}\sum_{k=1}^{K}\eta_k\left(\mathbf{v}_k(i) - \sum_{j\in C_i}p_{ij}\mathbf{v}_k(j)\right)^2, \tag{12}$$

with C_i the coarse features used to define the value of fine feature i and weights $\eta_k = \langle (X^TX + \beta I)\mathbf{v}_k, \mathbf{v}_k\rangle$ chosen to reflect the energy norm. For the given aggregation techniques in Section 3.4, we have that $|C_i| = 1$, since each feature is assigned to one cluster or aggregate. This can be extended if multiple clusters are used to interpolate the value of one fine feature. For example, the k-closest leaders to feature i of Leader–Follower clustering can be chosen as the coarse features used in the interpolation process. The coarse features C_i for Leader–Follower clustering are the leaders and for Rényi-entropy subsampling the features in the working set. For K-means++, there is no natural definition for the coarse feature and one of the features of one cluster is randomly chosen as the representing coarse feature.

In Bootstrap AMG[15] the prototypes vectors approximate the eigenvectors with small eigenvalues. Bootstrap AMG refines the prototypes used for interpolation by calculating the eigenvectors on the coarsest level. In contrast, we are interested in the principal eigenvectors. In practice, only a few prototype vectors are required and these can be simply calculated using Arnoldi iterations[49] on the finest level.

3.6 | Multilevel preconditioning

In AMG, coarsening is applied hierarchically until the size is sufficiently small and exact solvers can be applied. The multilevel concept has proven its merits in various applications, see for example Reference 50. The clustering algorithms are used hierarchically to create several levels of coarser features. For K-means and Rényi-entropy subsampling the size of the coarse level is chosen. For the leader follower algorithm, the threshold can be varied. The performance of the clusterings algorithms is determined by the correlation of the features. The smaller the dimension of the coarse level, the larger the variance within clusters.

Since we propose the use of a nonstationary smoother, flexible preconditioning is required. Flexible variants for Generalized minimal residual method (GMRES)[51] and CG are available.[52] These flexible iterative solvers allow for the coarser levels to be solved inexactly and hence even recursive multilevel Krylov-based cycles are possible.[53] In theory, it is possible to create a multilevel preconditioner. In practice, however, it is more efficient to only use the coarsest level and skip the intermediate levels as experimentally shown in Section 4.

4 | NUMERICAL EXPERIMENTS

We have applied our preconditioner for variants of the linear system

$$(X^T X + \beta I)\mathbf{x} = X^T \mathbf{b},$$

with $\mathbf{b}$ a normal distributed random vector using different feature matrices X. Table 1 provides the characteristics of the used data matrices X in our experiments. For the larger datasets with more than 10,000 features, the coarse size F_C was chosen in the range [3000, 7500], since calculating the Cholesky decomposition for this dimension range is feasible. Leader–Follower and Graph aggregation do not allow to set the size of coarse level up front. For Leader–Follower, a large threshold is chosen and decreased until the coarse size is in the required range. For a large threshold, a Leader–Follower clustering is computed faster than when using a small threshold. Graph aggregation was recursively applied until the coarse level was small enough. For the smaller datasets, the coarse level was chosen such that the number of features was reduced by a factor two or more. The experiments were implemented in C++[1] and compiled with gcc 9.3.0 and OPENMP 4.0 with compile option -O3 using a machine with an Intel(R) Core(TM) i7-6560U (2.20GHz) processor with an L3 cache memory of 4096 KB and 16 GB of DRAM where three out of four cores were used.

Table 2 shows the execution time for regularized least-squares linear system (LS) (1) with fixed tolerance ϵ and β for CG, diagonally preconditioned CG (DCG), and flexible CG using a two-level preconditioner (TLFCG) with two iterations of CG as presmoother. The coarse level was created using Leader–Follower clustering with Euclidean distance and the adjusted average interpolation in (7). The Euclidean distance was used for all given experiments. Table 2 additionally provides the number of iterations and the speed-up with respect to CG for DCG and TLFCG. Besides the execution time for solving (1), Table 2 details the execution time of Bayesian regression (BR) with noise injection using 600 samples for CG and TLFCG. For BR the coarse level was solved using the SVD while the linear system uses the Cholesky Decomposition.

Most of the datasets in Table 1 are underdetermined or undersampled. Regularization is added and by setting $\beta > 0$, noise components with large norms are penalized. In machine learning, the regularization parameter is often used as a measure against overfitting and several values are tried using cross-validation. Furthermore, by increasing β the condition number decreases and CG converges faster. The values of β and ϵ in Table 2 are chosen from a set for which CG converges slow but within a reasonable number of iterations. More results for varying β and ϵ are given in Section 4.2. Note that if β is chosen very large, CG will converge fast and no preconditioner is required. For BR, the value of β is determined automatically within the sampling process and the reported tolerance ϵ is used.

[1] https://scm.cs.kuleuven.be/scm/git/multilevel_macau

TABLE 1 The characteristics of the data matrices X used in the experiments

Name	Source	Value type	Number of rows	Number of columns	Number of nonzeros	Fillin(%)
Trec10	54	Double	106	478	8612	17.00
CNAE	55	Double	1080	856	7233	0.78
micromass	55	Double	360	1300	48,713	10.41
ovariancancer	56	Double	216	4000	864,000	100.00
gisette	55	Double	6000	5000	29,729,997	99.10
DrivFace	55	Double	606	6400	3,878,400	100.00
air04	54	Double	823	8904	72,965	1.00
arcene	55	Double	100	10,000	540,941	54.09
SNP1	57	Short int	10,000	10,000	57,641,064	57.64
tmc2007	66	Double	21,519	30,438	2,283,179	0.35
nlpdata	56	Double	31,572	34,023	2,277,757	0.21
rcv1_multi	58	Double	15,564	47,236	1,028,284	0.14
SNP2	57	Short int	10,000	50,000	282,212,533	56.44
news20	59	Double	15,935	62,061	1,272,569	0.13
MSI mouse	60	double	14,640	80,339	329,606,165	28.02
SNP3	57	short int	10,000	100,000	570,261,477	57.03
E2006	61	double	16,087	150,360	19,971,015	0.83
ChEMBL	62	binary	167,668	291,714	12,246,376	0.03

TABLE 2 The number of iterations and execution time for solving regularized least squares linear systems using Conjugate Gradient (CG), diagonally preconditioned CG (DCG), and Flexible conjugate gradient method (FCG) accelerated with our two-level preconditioner (TLFCG) for different datasets. Additionally, the execution time for Bayesian regression (BR) using 600 samples is reported for CG and TLFCG. The reported speed-up (SU) is with respect to CG for solving the linear system. For TLFCG, the speed-up for BR (SU-BR) is also given. TLFCG achieves speed-up for most of the data sets and solves the linear system with the smallest number of iterations

			CG			DCG			TLFCG					
Name	β	ϵ	Iter.	LS(s)	BR(s)	Iter.	LS(s)	SU	Iter.	LS(s)	SU	BR(s)	SU-BR	F_C
Trec10	1E-8	1E-8	302	3.40E-3	1.131	127	1.86E-3	1.83	**1**	**1.2E-4**	**28.19**	**4.17E-1**	**2.71**	207
CNAE	1E-8	1E-8	785	1.21E-2	**3.43**	313	**6.15E-03**	**1.96**	**107**	2.25E-2	0.54	4.83	0.71	375
micromass	1	1E-6	1399	6.86E-2	1.29E1	229	1.02E-2	6.74	**1**	**5.48E-4**	**125.18**	**8.21E-1**	**15.69**	483
ovariancancer	1E-8	1E-8	160	1.51E-1	3.41E1	54	5.85E-2	2.58	**1**	**5.28E-3**	**28.6**	**1.41E1**	**2.42**	435
gisette	1E-4	1E-4	12,603	4.45E+2	1.81E5	13,119	4.72E+2	0.94	**300**	**5.70E+1**	**7.81**	**1.21E2**	**1495.87**	2465
DrivFace	1E-8	1E-8	577	3.53	4.13E2	604	2.87	0.95	**1**	**3.0E-2**	**90.77**	**9.26E1**	**4.46**	2028
air04	1E-8	1E-8	440	3.58E-2	**1.68E1**	400	4.35E-2	0.82	**2**	**2.44E-2**	**1.47**	8.57E1	0.20	3045
arcene	1E-4	1E-8	55	2.96E-2	8.85	51	3.27E-2	0.91	**1**	**1.73E-2**	**1.71**	**6.36**	**1.39**	3274
SNP1	1E-2	1E-3	11061	6.89E2	3.64E3	14393	8.98E2	0.76	**97**	**3.19E1**	**21.60**	**8.56E2**	**4.25**	3216
tmc2007	1E-4	1E-4	3875	1.93E+1	4.17E3	506	**2.61**	**7.41**	**167**	8.76	2.20	**1.92E3**	**2.17**	4728
nlpdata	1E-4	1E-3	98,093	3.31E+2	1.56E5	8805	3.32E+1	9.97	**227**	**1.40E+1**	**23.60**	**6.75E2**	**231.11**	5793
rcv1_multi	1E-4	1E-3	1523	2.42	**1.43E3**	1030	**1.96**	**1.24**	**201**	5.96	0.41	3.55E3	0.40	3944
SNP2	1E-2	1E-3	18,371	6.11E+3	3.91E4	24,581	7.14E+3	0.86	**53**	**8.58E+1**	**71.19**	**6.69E3**	**5.85**	5582
news20	1E-6	1E-6	21,830	6.59E+1	**1.288E4**	21,268	7.09E+1	0.93	**707**	**4.88E+1**	**1.35**	1.38E4	0.93	6151
MSI mouse	1E-6	1E-6	675	3.44E2	1.12E6	108	**5.65E1**	**6.09**	**30**	8.00E1	4.31	**5.84E5**	**1.91**	5269
SNP3	1E-2	1E-3	11,512	7.45E+3	6.84E4	10,358	6.45E+3	1.16	**23**	**1.21E+2**	**61.61**	**1.52E4**	**4.5**	4893
E2006	1E-6	1E-6	3943	1.74E+2	2.52E2	5529	2.48E2	0.70	**203**	**5.97E1**	**2.91**	**2.38E2**	**1.06**	7036
ChEMBL	1E-3	1E-3	11790	2.43E2	5.79E4	11535	2.66E2	0.91	**712**	**1.06E2**	**2.29**	**4.14E4**	**1.40**	5581

As can be seen from Table 2, applying our two-level preconditioner results in speed-up with respect to CG for most of the data sets. Diagonally preconditioned CG can reduce the number of iterations but the gain is often small or there is no gain at all. TLFCG requires less iterations to reach the required tolerance for all cases but each iteration is more expensive. The actual speed-up depends on the quality of clusters. The goal is to achieve clusters for which the correlation within a cluster is high but correlation between clusters is small which is data dependent. Note that the gisette dataset is nearly dense with more samples than features and other techniques to solve the linear system are better suited than CG.

Table 3 compares the Gibbs sampling approach for BR with BR, available in MATLAB.[56] The average Pearson correlation (ρ) and root-mean-square-error (RMSE) for the test set using fivefold cross-validation are provided. The Gibbs sampler uses 600 samples with 100 samples discarded as burn-in. For the smallest datasets, the sampler approach is slower but for the larger datasets the sampler approach is faster and thus scales better due to the use of iterative methods. Both approaches achieve similar predictive performances.

The setup time for TLFCG is detailed in Table 4 for Leader–Follower clustering (LF), K-means++ (KM), Rényi-entropy-based subsampling (RE) and graph aggregation (GA) using a five-nearest neighbor graph. These reported

TABLE 3 The average execution time (Bayesian regression, BR), Pearson correlation (ρ) and root-mean-square-error (RMSE) for fivefold cross validation for standard Bayesian regression in MATLAB and the TLFCG Gibbs sampling approach. The Gibbs sampling approach is faster for larger datasets, requires less memory and achieves similar predictive accuracy

	MATLAB Bayesian regression			TLFCG Gibbs sampling		
Data	ρ	**RMSE**	**BR(s)**	ρ	**RMSE**	**BR(s)**
Trec10	**0.909**	9.18	**3.51E-1**	0.761	**7.97**	4.17E-1
CNAE	0.967	1.24	**2.29**	0.967	**1.02**	4.83
micromass	Failed due to ill-conditioned data			**0.993**	**9.85e+05**	**8.21E-1**
ovariancancer	**0.995**	1.22	5.65E1	0.988	**0.98**	**1.41E1**
Gisette	**0.998**	**1.43**	3.78E2	0.986	1.70	**1.21E2**
DrivFace	**0.971**	**2.85**	1.98E2	0.965	3.27	**9.26E1**
air04	0.857	6.28	4.15E2	**0.918**	**4.06**	**8.57E1**
arcene	0.965	**1.07E3**	8.10E2	**0.969**	1.22E3	**6.36**
SNP1	Out of Memory			**0.994**	**28.81**	**8.56E2**
nlpdata	Out of Memory			**0.992**	**4.92E1**	**6.75E2**

TABLE 4 The setup time of our two-level preconditioner. The reported times are the sum of the clustering time and the time to create the coarse solver. The singular value decomposition (SVD) is only reported in combination with Leader–follower (LF) clustering. For K-means++ (KM), Rényi-entropy subsampling and graph aggregation (GA), the setup time in combination with the Cholesky decomposition is reported for most data sets. The setup time is related to the number of nonzeros of X as given in Table 1

	LF			KM		RE		GA	
Name	F_C	**Cholesky(s)**	**SVD(s)**	F_C	**Cholesky(s)**	F_C	**Cholesky(s)**	F_C	**Cholesky(s)**
Trec10	207	1.532E-2	2.57E-2	200	1.71E-2	200	1.4E-2	262	4.71E-3
CNAE	375	1.52E-2	6.32E-2	400	2.75E-2	400	1.51E-1	528	5.21E-2
micromass	483	9.01E-2	1.01E-1	500	1.97E-1	500	4.01E-1	745	9.09E-2
ovariancancer	435	1.08	8.34E-1	400	8.29	400	1.42	510	1.62
gisette	2465	8.58E1	1.23E2	3000	1.01E3	3000	2.72E2	3583	8.4E1
DrivFace	2028	8.17	4.31	2000	9.00E1	2000	2.17E1	3441	1.11E1
air04	3045	4.26E-1	3.11	3000	4.76	3000	2.28	5069	1.14
arcene	3274	1.90	1.78	3000	2.67E1	3000	8.05	5572	2.8
SNP1	3216	1.56E2	2.02E2	5000	1.89E3	5000	2.1E3	5564	2.73E3
tmc2007	4728	6.03	1.58E3	5000	2.77E2	5000	3.87E1		
nlpdata	5793	7.11	1.58E3	6000	3.02E2	6000	2.99E1	5413	2.20E1
rcv1_multi	3944	2.73	5.79E2	4000	1.79E2	4000	7.63		
SNP2	5582	1.04E3	1.11E3	5000	7.37E3	5000	3.86E3	4720	5.31E3
news20	6151	6.09	2.05E2	6000	3.79E2	6000	2.14E1	7059	7.06E1
E2006	7036	5.59E1	2.60E3	7000	8.80E3	7000	3.34E2	6955	4.01E2
ChEMBL	5581	2.28E1	1.58E3						

times are in combination with a Cholesky decomposition on the coarsest level. For LF, the setup time using a SVD is additionally given. Computing the Cholesky decomposition or SVD was done using LAPACK.[63] For SVD, only the right singular vectors of X_C are required to solve Equation (1) on the coarse level. As can be seen from Table 4 the setup time is directly related to the number of nonzeros (see Table 1).

The sparser the data, the smaller the clustering time. The difference in setup time between Cholesky and SVD depends on the number of rows of X_C. If the number of rows is higher than the number of columns, calculating the SVD will result in larger execution time. The optimal value of β on the coarse level is the same as the value of β on the finest level. This means that when using the same coarse matrix X_C, the Cholesky decomposition has to be recalculated if β varies while the SVD can be reused. Note that the execution time of computing the Cholesky composition can be insignificant with respect to the time solving the linear system. Computing, for example, only the Cholesky decomposition for a coarse level with 5581 features for the ChEMBL data takes abou t 1 s while solving the linear system takes about 4 min.

4.1 | Designing the two-level preconditioner

There are different ways to create the two-level preconditioner. Firstly, there are different clustering algorithms. We have detailed LF, KM, RE, and GA. These are basic but generally applicable clustering algorithms. Next, there are different ways to design the interpolation operator P and subsequently create the coarse level from a clustering. We defined using the plain average (A), the adjusted square root average (SA) as detailed in (7) and the least-squares interpolation used in bootstrap (LS). Figure 2 shows the relative residual in function of the numbers of iterations for different clustering algorithms and different interpolation operators for four data sets using the same values for β and ϵ as given in Table 2. The dimension of the coarse levels are given in Table 4. The coarse sizes for KM and RE were chosen close to the values of LF and GA. The coarse level in these experiments was solved using a Cholesky Decomposition.

As can be seen in the left column of Figure 2, the efficiency of the aggregation technique is data dependent. LF performs well for these four datasets. Note that using KM or LF results in significant reduction of number of iterations with respect to CG. For RE subsampling, the radial basis function (RBF) kernel $\mathcal{k}(x,y) = \exp\left(-\frac{||x-y||^2}{2\sigma^2}\right)$ with $\sigma = 0.6$ was used. The choice of kernel can be further optimized depending on the data. Since a kernel is required, RE is less optimal as a black-box solver. Graph aggregation using graph matching performs well for the SNP2 data set, but less optimal for the other three data sets.

The right column of Figure 2 details the different interpolations methods. The least-squares interpolation and the adjusted average (7) have similar performance for these data sets with the least-squares interpolation slightly better for 3 data sets. Note that both these interpolation operators reduce the number of iterations significantly. Taking the average performs badly for two datasets, as seen in Figure 2(f), (d).

After designing the coarse level, we propose to use CG as rougher. Figure 3 shows the absolute residual in function the iteration for applying two iterations of CG before the coarse-correction (Pre-CG), before and after the coarse correction (PrePost-CG) or after the coarse correction (Post-CG). Figure 3 additionally shows the result when applying GMRES or Richardson iteration instead of CG. As can be seen from Figure 3(a), (c), applying CG before the coarse correction results in the faster and more stable convergence. Using CG as a rougher additionally has a small advantage over GMRES and Richardson iteration as shown in Figure 3(b), (d).

The chosen aggregation method results in a coarser level. This aggregation method can be recursively applied resulting in a hierarchy of levels. In theory a multilevel preconditioner can be created. Table 5, however, shows that in practice it is more efficient to only use the coarsest level. The number of fine level iterations is reduced by using the intermediate levels, but the overall execution time is larger since more time is spent on the intermediate levels. The two-level preconditioner can be seen as a subspace preconditioner combined with CG as a rougher.

4.2 | Influence of the regularization parameter

The regularization parameter β plays an important role in the performance of an iterative solver. Figure 4 shows the speed-up in execution time of two-level preconditioned FCG with respect to CG for different values of β and different values of ϵ. There are two phenomenons visible in the figure. Firstly, the larger the value of the regularization parameter β, the faster CG will converge and generally the smaller the speed-up we can achieve. The actual convergence depends on the eigenspectrum and is thus data dependent. Note that the eigenspectrum is influenced by the value of β.

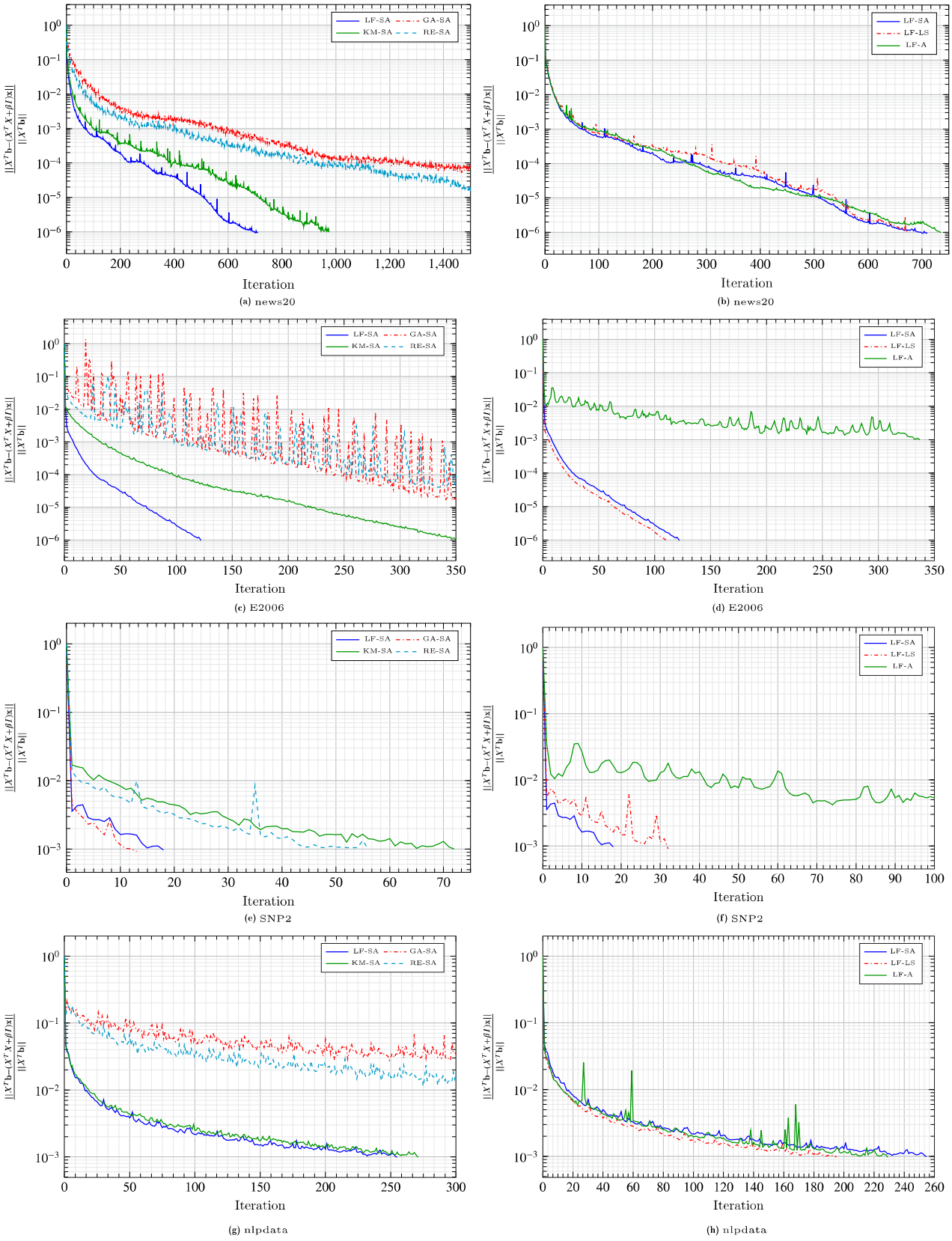


FIGURE 2 The relative residual in function of the number of iterations for different clustering algorithms (left) and different interpolation operators (right) for news20, E2006, SNP2 and nlpdata. The results left depict the performance of Leader–follower clustering (LF), K-means++ (KM), Rényi-entropy subsampling and graph aggregation (GA). The subfigures on the right show the different interpolation operators: adjusted square root average (SA), average (A) and least squares interpolation (LS). LF in combination with SA or LS interpolation works well. (A) news20; (B) news20; (C) E2006; (D) E2006; (E) SNP2; (F) SNP2; (G) nlpdata; (H) nlpdata

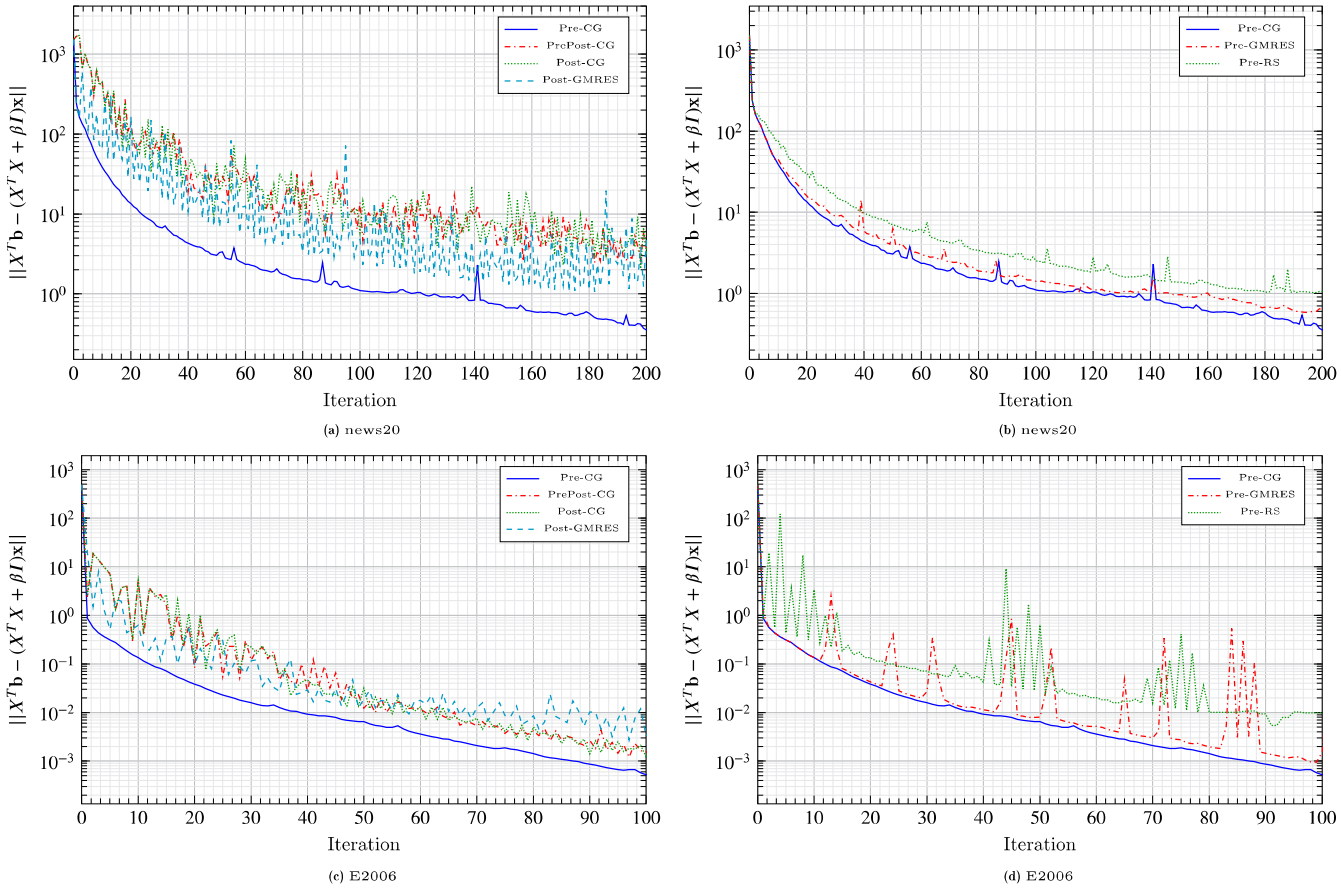


FIGURE 3 The residual in function of the number of iterations for different smoothing options for news20 and E2006. The used smoothers are Conjugate Gradient, GMRES and Richardson-iteration (RS). The smoothers were applied before the coarse correction (Pre) , after the coarse correction (Post) and before and after the coarse correction (PrePost). (A) news20; (B) news20; (C) E2006; (D) E2006

TABLE 5 The number of fine level iterations and executions time for recursively applying FCG with different tolerances ϵ_{sub} on each level and directly projecting to the coarsest level (Two-level) for news20 and E2006. The required tolerance ϵ was set to $1E-3$ and $\beta = 1E-6$. The sizes of the different levels for news20 are [62, 061, 27, 543, 16, 315, 7101] and for E2006 [150, 360, 42, 337, 27, 865, 20, 629, 8463]. The coarsest levels are used in combination with the finest level if not all available levels are exploited in the preconditioner

nb	$\epsilon_{\text{sub}} = 1e-1$		$\epsilon_{\text{sub}} = 1e-3$		**Two-level**	
levels	**Iter.**	**Exec. (s)**	**Iter.**	**Exec. (s)**	**Iter.**	**Exec. (s)**
3	67	11.48	13	137.69	55	4.44
4	19	48.57	2	77.11	54	4.19
nb	$\epsilon_{\text{sub}} = 1e-1$		$\epsilon_{\text{sub}} = 1e-3$		**Two-level**	
levels	**Iter.**	**Exec. (s)**	**Iter.**	**Exec. (s)**	**Iter.**	**Exec. (s)**
3	2	1.09	1	7.32	3	0.83
4	2	1.28	1	61.23	2	1.06
5	1	45.21	1	365.64	3	0.82

Figure 4 additionally shows that the speed-up depends on the required tolerance ϵ. As can be seen in previous Figures 2 and 3 the residual of our two-level preconditioner initially decays fast but does slow down. As a result the speed-up for the smaller tolerances can be less than for the larger tolerance. In practice however applications with required relative errors smaller than 10^{-6} are seldom.

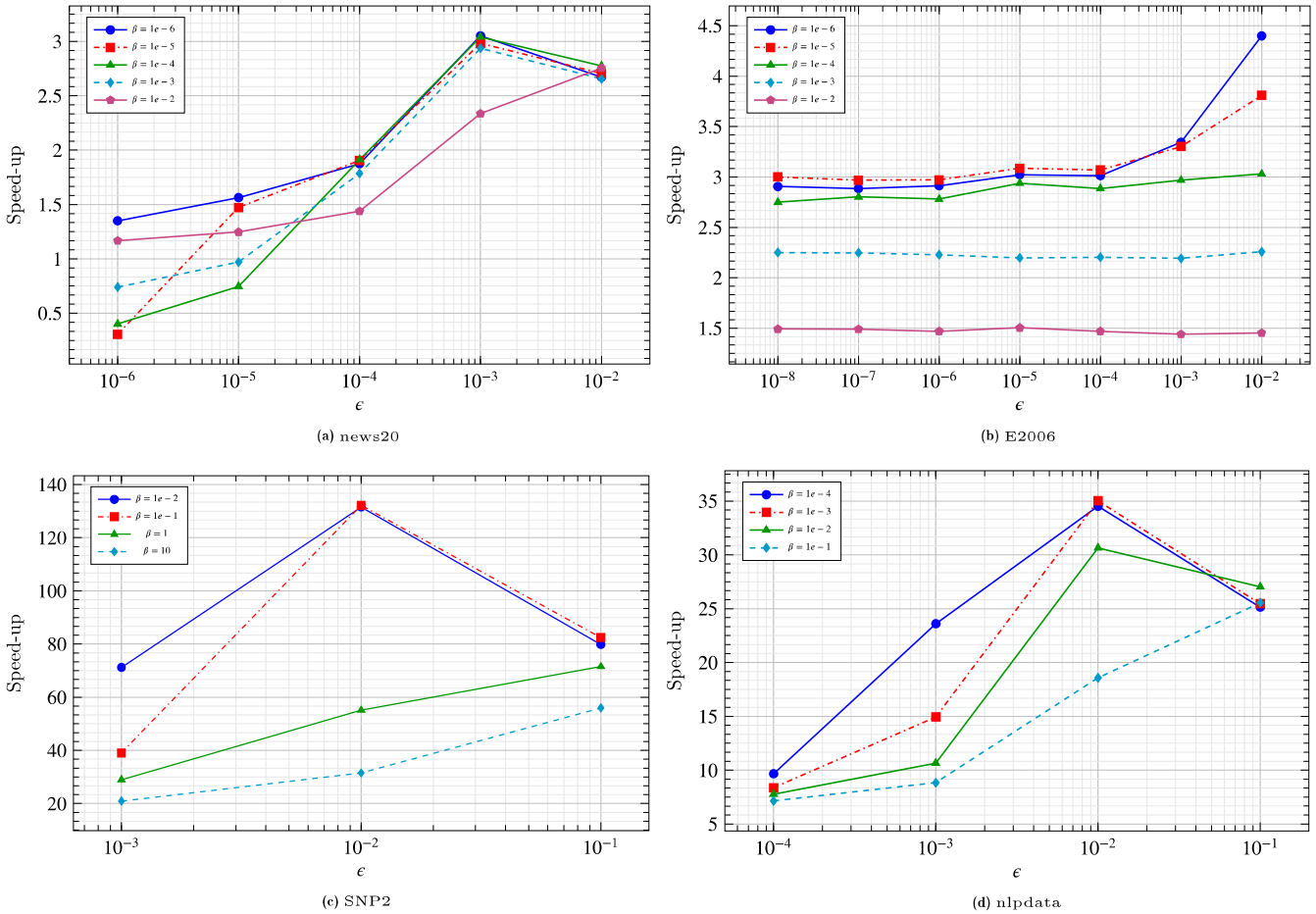


FIGURE 4 The speed-up of our two-level preconditioner using two iterations of conjugate gradient (CG) as presmoother and Leader–Follower clustering with adjusted square root average interpolation with respect to CG for solving a regularized least squares linear system in function of the required tolerance ϵ for different regularization parameters β for news20 (A), E2006 (B), SNP2 (C), and nlpdata (D)

5 | CONCLUSION

The two-level preconditioner described here finds its application in machine learning. We have shown that it is possible to approximate the principal eigenvectors on a coarser level by means of clustering the columns of the feature matrix. This coarser level is used in a two-level preconditioner to accelerate Flexible CG. The efficiency of our preconditioner is data dependent and clustering algorithms perform differently for distinct data sets. Using general clustering algorithms such as K-means++ and Leader-Follower, we have shown that the number of iterations reduces dramatically for several data sets. The setup time of the clustering algorithms can be significantly but can be reused when multiple system solves are required. The setup time can also be further reduced by only approximately calculating the distance to other features by using locality-sensitive-hashing.[64,65] We showed results for clustering in the column space (features) of X. It is also possible to additionally cluster in the row space (samples), reducing the execution time of SVD on the coarsest level.

Using the two-level preconditioner to accelerate FCG, speed-up was achieved for almost all datasets. The efficiency of the preconditioner was shown to be data dependent. Even modest relative speed-up can result in large absolute gain. Even with modest speed-up, the number of iterations is significantly reduced resulting in a more robust iterative solver. Using more than two-levels results in even less iterations but does increase the execution time.

ACKNOWLEDGEMENTS

The authors would like to thank the reviewers for their comments and suggestions. This study does not have any conflicts to disclose. This work was supported by Research Foundation - Flanders (FWO, No. G079016N). We would additionally like to thank David Vanavermaete for generating the SNP datasets.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available in the repository multilevel_macau at https://scm.cs.kuleuven.be/scm/git/multilevel_macau. These data were derived from the following resources available in the public domain, see References 54-62,66

ORCID

Joris Tavernier https://orcid.org/0000-0002-3308-1332

REFERENCES

1. Demmel J. Applied numerical linear algebra. Philadelphia, PA: Society for Industrial and Applied Mathematics; 1997.
2. Trefethen LN, Bau D III. Numerical linear algebra. Vol 50. Philadelphia, PA: Siam; 1997.
3. van der Sluis A. Condition numbers and equilibration of matrices. Numer Math. 1969;14(1):14–23. https://doi.org/10.1007/BF02165096.
4. Kershaw DS. The incomplete Cholesky–conjugate gradient method for the iterative solution of systems of linear equations. J Comput Phys. 1978;26(1):43–65. http://www.sciencedirect.com/science/article/pii/0021999178900980.
5. Lin CJ, Moré JJ. Incomplete Cholesky factorizations with limited memory. SIAM J Sci Comput. 1999;21(1):24–45. https://doi.org/10.1137/S1064827597327334.
6. Saad Y. ILUT: a dual threshold incomplete LU factorization. Numer Linear Algebra Appl. 1994;1(4):387–402. https://doi.org/10.1002/nla.1680010405.
7. Benzi M, Meyer CD, Tůma M. A Sparse approximate inverse preconditioner for the conjugate gradient method. SIAM J Sci Comput. 1996;17(5):1135–49. https://doi.org/10.1137/S1064827594271421.
8. Briggs W, Henson V, McCormick S. A multigrid tutorial. 2nd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics; 2000.
9. Ruge JW, Stüben K. Algebraic multigrid. Multigrid methods. Philadelphia, PA: Society for Industrial and Applied Mathematics; 1987.p. 73–130. https://doi.org/10.1137/1.9781611971057.
10. Stüben K. A review of algebraic multigrid. In: Sloan D, Süli E, Vandewalle S, editors. Partial differential equations volume 7 of numerical analysis 2000 Amsterdam. Netherlands: Elsevier; 2001. p. 281–309.
11. Vassilevski PS. Multilevel block factorization preconditioners: matrix-based analysis and algorithms for solving finite element equations. New York, NY: Springer-Verlag; 2008.
12. Brezina M, Falgout R, MacLachlan S, Manteuffel T, McCormick S, Ruge J. Adaptive smoothed aggregation (αSA) multigrid. SIAM Rev. 2005;47(2):317–46. https://doi.org/10.1137/050626272.
13. Brezina M, Falgout R, MacLachlan S, Manteuffel T, McCormick S, Ruge J. Adaptive algebraic multigrid. SIAM J Sci Comput. 2006;27(4):1261–86. https://doi.org/10.1137/040614402.
14. MacLachlan S, Manteuffel T, McCormick S. Adaptive reduction-based AMG. Numer Linear Algebra Appl. 2006;13(8):599–620. https://doi.org/10.1002/nla.486.
15. Brandt A, Brannick J, Kahl K, Livshits I. Bootstrap amg. SIAM J Sci Comput. 2011;33(2):612–32. https://doi.org/10.1137/090752973.
16. D'Ambra P, Vassilevski PS. Improving solve time of aggregation-based adaptive AMG. Numer Linear Algebra Appl. 2019;26(6):e2269. https://doi.org/10.1002/nla.2269.
17. Reichel L, Shyshkov A. Cascadic multilevel methods for ill-posed problems. J Comput Appl Math. 2010;233(5):1314–25. Special Issue Dedicated to William B. Gragg on the Occasion of His 70th Birthday. http://www.sciencedirect.com/science/article/pii/S0377042709002192.
18. Jacobsen M, Hansen PC, Saunders MA. Subspace preconditioned LSQR for discrete Ill-posed problems. BIT Numer Math. 2003;43(5):975–89. https://doi.org/10.1023/B:BITN.0000014547.88978.05.
19. Paige CC, Saunders MA. LSQR: an algorithm for sparse linear equations and sparse least squares. ACM Trans Math Softw. 1982;8(1):43–71.
20. Robbins H, Monro S. A Stochastic approximation method. Ann Math Stat. 1951;22(3):400–7. http://www.jstor.org/stable/2236626.
21. Bottou L. Large-scale machine learning with stochastic gradient descent. In: Lechevallier Y, Saporta G, editors. Proceedings of COMPSTAT '2010. Heidelberg, Germany: Physica-Verlag HD; 2010. p. 177–86.
22. Johnson R, Zhang T. Accelerating stochastic gradient descent using predictive variance reduction. In: CJC B, Bottou L, Welling M, Ghahramani Z, Weinberger KQ, editors. Advances in neural information processing systems 26. Volume 315. New York, NY: Curran Associates, Inc.; 2013. –323 http://papers.nips.cc/paper/4937-accelerating-stochastic-gradient-desce%nt-using-predictive-variance-reduction.pdf.
23. Zouzias A, Freris NM. Randomized extended kaczmarz for solving least squares. SIAM J Matrix Anal Appl. 2013;34(2):773–93. https://doi.org/10.1137/120889897.
24. Gower RM, Richtárik P. Randomized iterative methods for linear systems. SIAM J Matrix Anal Appl. 2015;36(4):1660–90. https://doi.org/10.1137/15M1025487.
25. Bishop CM. Pattern recognition and machine learning. New York, NY: Springer-Verlag; 2006.
26. Gelman A, Carlin JB, Stern HS, Dunson DB, Vehtari A, Rubin DB. Bayesian data analysis. Boca Raton, FL: Chapman & Hall/CRC Press; 2013.
27. Simm J, Arany A, Zakeri P, et al. Macau: scalable Bayesian factorization with high-dimensional side information using MCMC. Paper presented at: Proceedings of the 2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP). Tokyo, Japan; 2017. p. 1–6.

28. Salakhutdinov R, Mnih A. Bayesian probabilistic matrix factorization using Markov Chain Monte Carlo. Proceedings of the 25th International Conference on Machine Learning. ICML '08. New York, NY: ACM; 2008. p. 880–887. 10.1145/1390156.1390267
29. Hosmer DW Jr, Lemeshow S, Sturdivant RX. Applied logistic regression. Vol 398. Boca Raton, FL: John Wiley & Sons; 2013.
30. Vapnik VN. An overview of statistical learning theory. IEEE Trans Neural Netw. 1999;10(5):988–99.
31. Lin CJ, Weng RC, Keerthi SS. Trust region Newton method for logistic regression. J Mach Learn Res. 2008;9(Apr):627–50.
32. Cai D, He X, Han J. SRDA: an efficient algorithm for large-scale discriminant analysis. IEEE Trans Knowl Data Eng. 2008;20(1):1–12.
33. Saad Y. Iterative methods for sparse linear systems. 2nd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics; 2003. https://doi.org/10.1137/1.9780898718003.
34. Douglas C, Douglas J Jr. A unified convergence theory for abstract multigrid or multilevel algorithms serial and parallel. SIAM J Numer Anal. 1993;30(1):136–58.
35. Hartigan JA. Clustering algorithms. 99th New York. NY: John Wiley & Sons, Inc; 1975.
36. Arthur D, Vassilvitskii S. k-means++: the advantages of careful seeding. Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms. Philadelphia, PA: Society for Industrial and Applied Mathematics; 2007. p. 1027–1035.
37. Girolami M. Orthogonal series density estimation and the kernel eigenvalue problem. Neural Comput. 2002;14(3):669–88. https://doi.org/10.1162/089976602317250942.
38. Rényi A. On measures of entropy and information. Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability. vol. 1, Contributions to the Theory of Statistics. Berkeley; 1961. p. 547–561.
39. Brabanter KD, Brabanter JD, Suykens JAK, Moor BD. Optimized fixed-size kernel models for large data sets. Comput Stat Data Anal. 2010;54(6):1484–504. http://www.sciencedirect.com/science/article/pii/S0167947310000393.
40. Schaeffer SE. Graph clustering. Comput Sci Rev. 2007;1(1):27–64.
41. Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E. Fast unfolding of communities in large networks. J Stat Mech Theory Exper. 2008;2008(10):P10008.
42. Lancichinetti A, Fortunato S. Community detection algorithms: a comparative analysis. Phys Rev E. 2009;80(5):056117.
43. D'Ambra P, Cutillo L, Vassilevski PS. Bootstrap AMG for spectral clustering. Comput Math Methods. 2019;1(2):e1020. https://doi.org/10.1002/cmm4.1020.
44. Muja M, Lowe DG. Scalable nearest neighbor algorithms for high dimensional data. IEEE Trans Pattern Anal Mach Intell. 2014;36(11):2227–40.
45. Brannick J, Chen Y, Zikatanov L. An algebraic multilevel method for anisotropic elliptic equations based on subgraph matching. Numer Linear Algebra Appl. 2012;19(2):279–95.
46. Brannick J, Chen Y, Kraus J, Zikatanov L. Algebraic multilevel preconditioners for the graph Laplacian based on matching in graphs. SIAM J Numer Anal. 2013;51(3):1805–27.
47. D'ambra P, Filippone S, Vassilevski PS. BootCMatch: a software package for bootstrap AMG based on graph weighted matching. ACM Trans Math Softw. 2018;44(4):39. https://doi.org/10.1145/3190647.
48. Manteuffel T, McCormick S, Park M, Ruge J. Operator-based interpolation for bootstrap algebraic multigrid. Numer Linear Algebra Appl. 2010;17(2-3):519–37.
49. Arnoldi WE. The principle of minimized iterations in the solution of the matrix eigenvalue problem. Q Appl Math. 1951;9(1):17–29.
50. Brandt A. Multiscale scientific computation: review 2001. In: Barth TJ, Chan T, Haimes R, editors. Multiscale and multiresolution methods. Berlin, Heidelberg/Germany: Springer; 2002. p. 3–95.
51. Saad Y. A Flexible inner-outer preconditioned GMRES algorithm. SIAM J Sci Comput. 1993;14(2):461–9. https://doi.org/10.1137/0914028.
52. Notay Y. Flexible conjugate gradients. SIAM J Sci Comput. 2000;22(4):1444–60. https://doi.org/10.1137/S1064827599362314.
53. Notay Y, Vassilevski PS. Recursive Krylov-based multigrid cycles. Numer Linear Algebra Appl. 2008;15(5):473–87. https://doi.org/10.1002/nla.542.
54. Davis TA, Hu Y. The university of Florida sparse matrix collection. ACM Trans Math Softw. 2011;38(1):1.
55. Lichman M. UCI machine learning repository. Irvine, CA: University of California, School of Information and Computer Sciences; 2013. http://archive.ics.uci.edu/ml.
56. MATLAB. Statistics and machine learning toolbox. Natick, MA: The MathWorks; 2018.
57. Hickey JM, Gorjanc G, Hearne S, Huang BE. AlphaMPSim: flexible simulation of multi-parent crosses. Bioinformatics. 2014;30(18):2686–8.
58. Lewis DD, Yang Y, Rose TG, Li F. RCV1: a new benchmark collection for text categorization research. J Mach Learn Res. 2004;5:361–97. http://dl.acm.org/citation.cfm?id=1005332.1005345.
59. Keerthi SS, DeCoste D. A modified finite Newton method for fast solution of large scale linear SVMs. J Mach Learn Res. 2005;6(Mar):341–61.
60. Rübel O, Greiner A, Cholia S, Louie K, Bethel EW, Northen TR. OpenMSI: a high-performance web-based platform for mass spectrometry imaging. Anal Chem. 2013;85(21):10354–61. https://doi.org/10.1021/ac402540a.
61. Kogan S, Levin D, Routledge BR, Sagi JS, Smith NA. Predicting risk from financial reports with regression. Proceedings of the 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics Human Language Technologies. Boulder, Colorado: Association for Computational Linguistics; 2009. p. 272–280. https://www.aclweb.org/anthology/N09-1031.
62. Bento AP, Gaulton A, Hersey A, et al. The ChEMBL bioactivity database: an update. Nucleic Acids Res. 2014;42(D1):D1083–90.
63. Anderson E, Bai Z, Bischof C, et al. LAPACK users' guide. 3rd. Philadelphia, PA: Society for Industrial and Applied Mathematics; 1999.

64. Gionis A, Indyk P, Motwani R. Similarity search in high dimensions via hashing. Paper presented at: Proceedings of the 25th International Conference on Very Large Data Bases. VLDB '99. San Francisco, CA: Morgan Kaufmann Publishers Inc.; 1999. p. 518–529. http://dl.acm.org/citation.cfm?id=645925.671516.
65. Andoni A, Razenshteyn I. Optimal datadependent hashing for approximate near neighbors. Proceedings of the 47th Annual ACM Symposium on Theory of Computing. STOC '15. New York, NY: ACM; 2015. p. 793–801. https://doi.org/10.1145/2746539.2746553
66. SIAM Text Mining Competition. 2007. https://www.csie.ntu.edu.tw/\LY1\textbackslash~cjlin/libsvmtools/datasets/multilabel.html\LY1\textbackslash#siam-competition2007

How to cite this article: Tavernier J, Simm J, Meerbergen K, Moreau Y. Two-level preconditioning for Ridge Regression. *Numer Linear Algebra Appl*. 2021;e2371. https://doi.org/10.1002/nla.2371