Parallel Learning Algorithm for Large-Scale Regression with Additive Models

Valeriy Khakhutskyy[1]** and Markus Hegland[2]**

[1] Institute for Advanced Study, Technische Universität Munchen, Lichtenbergstrasse 2a, D-85748 Garching, Germany,
khakhutv@in.tum.de

[2] Centre for Mathematics and its Applications, Mathematical Sciences Institute Australian National University, Canberra, ACT 0200, Australia.

Abstract. We present a novel parallel algorithm for training additive regression models. The approach relates to a number of other methods including the backfitting algorithm and alternating direction method of multipliers. However, we extend the scope of possible applications to include any ANOVA-type decomposition or an ensemble of random subspace projection models, and we show how the algorithm can be parallelised for distributed systems.
The experimental results illustrate the convergence and scaling properties of the algorithm on real and synthetic data.

Keywords: backfitting, ADMM, additive models, parallelisation, regression, BiCGStab, ensemble learning

1 Introduction

Machine learning applications continuously grow in size and dimensionality as with the growing interest in data science and increasing computational power more effort is spent on data gathering and analysis. At the same time, the asymptotics of the underlying function approximation algorithms remain unchanged.

With respect to the number of data entries, the linear basis expansion models, i.e. splines or sparse grids [1], show linear complexity of computation and storage. This is already optimal for a regression problem without any additional assumptions. Hence, further research attempts to minimise the complexity constants with better implementations, parallelisation, or sub-sampling heuristics [2–4].

With respect to dimensionality, the complexity of most approximation algorithms suffers from the "curse of dimensionality": a term coined by Richard Bellman in 1961, which denotes that the computation and storage complexity for a fixed approximation accuracy depends exponentially on the dimensionality

** With the support of the Technische Universität München Institute for Advanced Study, funded by the German Excellence Initiative (and the European Union Seventh Framework Programme under grant agreement n 291763).

[5]. The results from information-based complexity suggest that the curse of dimensionality can be avoided only if a problem possesses a special structure, i.e. smoothness or separability, that can be exploited by an algorithm [6, 7].

The methods that exploit this special structure include ANOVA decomposition, additive models [8], sparse grids, and random forests. Additive models are well established in statistics and thoroughly studied in the literature [9, 10]. Similar concepts are popular in the machine learning community. For example, recent developments apply new optimisation methods [11] and parallelisation paradigms [12]. Ensembles of random subspace projection models were successfully used for classification and regression [13]. Moreover, estimation of additive models is an integral part of generalised additive models – a more powerful but also a more computationally expensive representation concept [9].

In this paper, we discuss an approach for large-scale regression based on additive models and a parallel BiCGStab algorithm for optimisation. The method combines flexibility of choosing from a large number of suitable parametric and non-parametric models and an optimisation algorithm with fast convergence and a potential for efficient parallelisation both in data size and dimensionality. It is used for training additive models, but this restriction can be relaxed to include ensembles of subspace projection models. We extend the fitting method with data partitioning using kd-trees and orthogonalisation, which improves data locality for additive models and has a potential to adapt to manifolds.

The remainder of this paper is organised as follows: Section 2 introduces the necessary theoretical background. In Section 3 we suggest a Krylov-space method for solution of normal equations and show how the problem structure can be exploited for efficiency and parallelisation. We illustrate convergence and scalability of the new method using benchmark problems in Section 4. Finally, we conclude with a discussion of the results and provide an overview of future work in Section 5.

2 Theoretical Background

We consider a dataset of the form $(\mathbf{t}^{(1)}, y^{(1)}), \ldots, (\mathbf{t}^{(N)}, y^{(N)})$ with input variables $\mathbf{t}^{(i)}$ and target variables $y^{(i)}$. Ridge regression is often used to find an approximation of the input-target mapping f in a function space V:

$$\min_{f \in V} \tfrac{1}{2} \sum_{i=1}^{N} (f(\mathbf{t}^{(i)}) - y^{(i)})^2 + \tfrac{1}{2}\lambda \|Df\|_2^2, \tag{1}$$

with a positive regularisation parameter λ and some smoothness operator D.

As mentioned above, (1) suffers from the curse of dimensionality so that only the problems with a moderate number of input dimensions can be handled. Approaches that do not face the curse are based on decomposition of the space V into a sum of simpler function spaces

$$V = V_1 + \ldots + V_n. \tag{2}$$

For example, in the context of ANOVA, the decomposition of a function f with a d-dimensional input has the form

$$f(t) = f_0 + \sum_{j=1}^{d} f_j(t_j) + \sum_{1 \le i < j \le d} f_{i,j}(t_i, t_j) + \sum_{1 \le i < j < k \le d} f_{i,j,k}(t_i, t_j, t_k) + \dots ,$$

where t_j stands for the j-th component of the data point $\mathbf{t}$.

More general, the decomposition (2) may be not exact. For example, if we limit the number of terms V_j and restrict them to specific low dimensional function spaces, we obtain an ensemble of subspace projection models. Hereafter we consider only the 1-dimensional terms f_j, but extension to other function spaces is straightforward.

While it is not assumed that the V_j are linearly independent, we assume that the smoothness operator is consistent with the decomposition of V, such that the optimisation problem assumes the form

$$\min_{f_0 \in \mathbb{R}, f_1 \in V_1, \dots, f_d \in V_d} \tfrac{1}{2} \sum_{i=1}^{N} (f_0 + \sum_{j=1}^{d} f_j(t_j^{(i)}) - y^{(i)})^2 + \sum_{j=1}^{d} \tfrac{\lambda_j}{2} \|D_j f_j\|_2^2. \qquad (3)$$

Many models for representation of f_j, for example, linear basis expansion models, can be cast in terms of linear algebra. Let vector $\mathbf{x}$ denote parameters of $f(\mathbf{t})$ with respect to some generating system, e.g. coefficients of a polynomial model. Furthermore, let $\mathbf{Ax}$ be a vector of function values $f(\mathbf{t}^{(i)})$ and $\mathbf{y}$ – the vector of target values $y^{(i)}$. Taking into account the residual $\mathbf{r}$, we obtain

$$\underbrace{[\mathbf{A}_1 \dots \mathbf{A}_d]}_{\mathbf{A}} \underbrace{\left[\mathbf{x}_1^T \dots \mathbf{x}_d^T\right]^T}_{\mathbf{x}} = \mathbf{y} - \mathbf{r} \qquad (4)$$

with $\mathbf{x}_j \in \mathbb{R}^{m_j}$, $\mathbf{A}_j \in \mathbb{R}^{N \times m_j}$, $\mathbf{y}, \mathbf{r} \in \mathbb{R}^N$, $\mathbf{x} \in \mathbb{R}^m, \mathbf{A} \in \mathbb{R}^{N \times m}$, and $m := m_1 + \dots + m_d$. The problem (3) can now be written as

$$\min_{\mathbf{x}} \tfrac{1}{2} \|\mathbf{Ax} - \mathbf{y}\|_2^2 + \tfrac{\lambda}{2} \mathbf{x}^T \mathbf{D} \mathbf{x} \qquad (5)$$

with $\mathbf{D}$ a block diagonal matrix, which can be partitioned in a structure compatible with that of $\mathbf{x}$. Often, $\mathbf{D}$ is just an identity matrix. In order to minimise (5), one needs to solve the normal equations

$$(\mathbf{A}^T \mathbf{A} + \lambda \mathbf{D}) \mathbf{x} = \mathbf{A}^T \mathbf{y}. \qquad (6)$$

If we substitute (4) into (6) we obtain the system

$$\begin{bmatrix} \mathbf{A}_1^T \mathbf{A}_1 + \lambda \mathbf{D}_1 & \mathbf{A}_1^T \mathbf{A}_2 & \cdots & \mathbf{A}_1^T \mathbf{A}_d \\ \mathbf{A}_2^T \mathbf{A}_1 & \mathbf{A}_2^T \mathbf{A}_2 + \lambda \mathbf{D}_2 & \cdots & \mathbf{A}_2^T \mathbf{A}_d \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_d^T \mathbf{A}_1 & \mathbf{A}_d^T \mathbf{A}_2 & \cdots & \mathbf{A}_d^T \mathbf{A}_d + \lambda \mathbf{D}_d \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_d \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1^T \mathbf{y} \\ \mathbf{A}_2^T \mathbf{y} \\ \vdots \\ \mathbf{A}_d^T \mathbf{y} \end{bmatrix}. \qquad (7)$$

We are particularly interested in problems where the solution of (7) is used to determine the predicted values $\hat{\mathbf{f}} = \mathbf{Ax} = \mathbf{y} - \mathbf{r}$, which in view of (4) can be written as $\hat{\mathbf{f}} := \mathbf{f}_1 + \dots + \mathbf{f}_d,$ with $\mathbf{f}_j = \mathbf{A}_j \mathbf{x}_j$, $j = 1, \dots, d$. If we multiply every

row block j of (7) by $\mathbf{A}_j(\mathbf{A}_j^T\mathbf{A}_j + \lambda\mathbf{D}_j)^{-1}$ from the left and introduce

$$\mathbf{S}_j := \mathbf{A}_j(\mathbf{A}_j^T\mathbf{A}_j + \lambda\mathbf{D}_j)^{-1}\mathbf{A}_j^T \quad j = 1, \ldots, d, \tag{8}$$

we obtain equations of the form

$$\underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{S}_1 & \mathbf{S}_1 & \ldots & \mathbf{S}_1 \\ \mathbf{S}_2 & \mathbf{I} & \mathbf{S}_2 & \ldots & \mathbf{S}_2 \\ \mathbf{S}_3 & \mathbf{S}_3 & \mathbf{I} & \ldots & \mathbf{S}_3 \\ \vdots & \vdots & & \ddots & \vdots \\ \mathbf{S}_d & \mathbf{S}_d & \mathbf{S}_d & \ldots & \mathbf{I} \end{bmatrix}}_{\mathbf{S}} \underbrace{\begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{f}_3 \\ \vdots \\ \mathbf{f}_d \end{bmatrix}}_{\mathbf{f}} = \begin{bmatrix} \mathbf{S}_1\mathbf{y} \\ \mathbf{S}_2\mathbf{y} \\ \mathbf{S}_3\mathbf{y} \\ \vdots \\ \mathbf{S}_d\mathbf{y} \end{bmatrix}. \tag{9}$$

Following the tradition from statistics, we call $\mathbf{S}_j$ a *smoothing matrix*. The normal equation (9) is the one we are interested in solving instead of (7). In Section 4 we motivate this preference.

A popular choice for solving (5) on distributed systems is alternating direction method of multipliers (ADMM) [9]. In fact, one can show that if the penalisation parameter is equal to $1/\lambda$, the ADMM update steps would correspond to a sequence of a Jacobi-iteration for solution of (7) followed by an update of the auxiliary variable $\bar{\mathbf{z}}$. This motivates the comparison of the two methods in Section 4. We have to note, however, that this relationship does not transfer if the penalty or regularisation term is not quadratic.

3 Fitting Methods

Traditionally, problem (9) is solved using the *backfitting algorithm* [10] in sequence of a blocked Gauss-Seidel iterations:

$$\textbf{for } j = 1 \text{ to } d \textbf{ do: } \mathbf{f}_j = \mathbf{S}_j \left(\mathbf{y} - \sum_{k \neq j} \mathbf{f}_k \right).$$

Buja et al. have shown that the convergence speed of the backfitting algorithm heavily depends on the magnitude and distribution of the smoothers' eigenvalues which are significantly smaller than 1. The error terms corresponding to the eigenvectors with eigenvalues near 1 (e.g. constant, linear and low-frequency) would not be eliminated at all by the algorithm [10].

For regression models, however, it is not unusual that a number of large eigenvalues are clustered around 1.0. Moreover, a typical distribution of the eigenvalues of the matrix $\mathbf{S}$ contains a single large eigenvalue close to d, followed by a cluster of small eigenvalues.

At this point we suggest to use a BiCGStab-based Krylov method [14] for the solution of (9). Not only is it better suited for system matrices with clustered eigenvalues, it would eliminate the error components that are problematic for backfitting. We refer our reader to the original paper by van der Vorst [14] for a review of the original algorithm, and adopt its notation in this section.

A number of improvements for BiCGStab were suggested in the literature that would reduce the communication on distributed systems [15].

We derived a BiCGStab algorithm for fitting additive models that can be parallelised in the number of models and data entries. We can improve the scalability of the original algorithm by exploiting the special structure of the system matrix $\mathbf{S}$: the j-th block of the result of matrix-vector multiplication can be calculated as

$$\mathbf{v}_j = \mathbf{p}_j + \mathbf{S}_j(\sum_{i \neq j} \mathbf{p}_j) = \mathbf{p}_j + \mathbf{S}_j(\sum_{i=1}^{d} \mathbf{p}_i - \mathbf{p}_j).$$

Similar to the ideas in [15], we further reduce the communication overhead by postponing the calculation of the scalar products until the aggregation of the results of matrix-vector multiplications is necessary.

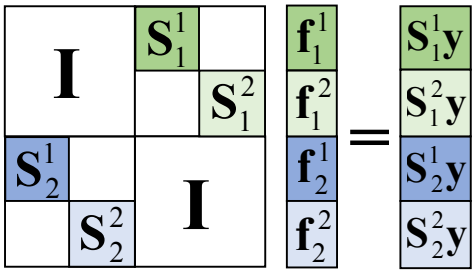
Algorithm 1 presents the new fitting procedure. The vectors with the subscript "Σ", e.g. $\mathbf{t}_\Sigma$, represent the sum of individual vectors, e.g. $\sum_i \mathbf{t}_i$. The function `Allreduce` is known from MPI programming and is similar to the reduce operation in the Map-Reduce framework. It denotes an application of an operation (in our case SUM) component-wise to the first argument across all processes and stores the results in the second argument. The scalar products in the second argument of the `Allreduce` functions in Lines 10 and 20 stand for variables containing the corresponding scalar products. In these calls a vector and scalar products are joined into a single memory segment to reduce communication.

Depending on the application requirements, two alternatives of data parallelism can be suggested: On the one hand, a simple SIMD parallelisation of the linear algebra operations can be performed using an appropriate BLAS implementation (MKL, OpenBLAS, etc) on shared memory systems. On the other hand, we suggest to partition data using kd-trees and to decouple individual problems as illustrated on Fig. 1. While we may loose smoothness of the solution across the partition borders, the decomposition completed with orthogonalisation is known to adapt to the internal data manifolds [16] and improves the data alignment important for additive models.

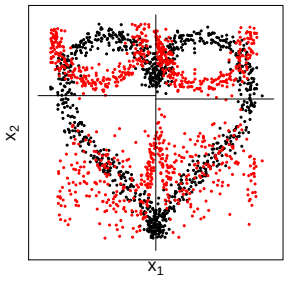
In this case of domain decomposition, communication in Lines 10 and 20 can be performed in three steps as illustrated on Fig. 2 to minimise the amount of sent and received data. In the first step, we calculate the sum of large vectors $[\mathbf{s}_j^T\mathbf{r}_j^0, \mathbf{t}_j^T\mathbf{s}_j, \mathbf{t}_j^T\mathbf{t}_j, \mathbf{t}_j^T\mathbf{r}_j^0, \mathbf{t}_j]$ and $[\mathbf{v}_j^T\mathbf{r}_j^0, \mathbf{v}_j]$ among the processors that share the same data partition (Fig. 2a). At this stage the size of the communicated vectors is basically the number of points in the partition. In the second step, only the scalar product results $[\mathbf{s}_j^T\mathbf{r}_j^0, \mathbf{t}_j^T\mathbf{s}_j, \mathbf{t}_j^T\mathbf{t}_j, \mathbf{t}_j^T\mathbf{r}_j^0]$ and $[\mathbf{v}_j^T\mathbf{r}_j^0]$ are reduced between the root processes of individual partitions (Fig. 2b). While the communication between partitions is more expensive in distributed clusters, we need to communicate only a small constant number of values, which is done very efficiently in the common MPI implementations. In the last step, we broadcast the updated values from the roots to all processes in the partitions (Fig. 2c).

Algorithm 1 Parallel BiCGStab Algorithm: code for processor $j, 0 \leq j \leq n-1$

Input: $\mathbf{S}_j$ smoothing matrix, $\mathbf{y}$ target vector
Output: $\mathbf{f}_j$ predictions of the function $f_j(x)$ at the data points
$\mathbf{r}_j^0 = S_j\mathbf{y}; \mathbf{r}_j = \mathbf{r}_j^0$
`Allreduce`$([\mathbf{r}_j, \mathbf{r}_j^T\mathbf{r}_j], [\mathbf{r}_\Sigma, \rho^{\text{new}}], \text{SUM})$
$\alpha = \omega = 1$; $\rho^{\text{old}} = \beta = \rho^{\text{new}}$
$\mathbf{f}_j = \mathbf{t}_j = \mathbf{v}_j = \mathbf{v}_\Sigma = \mathbf{p}_j = \mathbf{p}_\Sigma = \mathbf{s}_j = \mathbf{s}_\Sigma = \mathbf{0}$
while not converged **do**
  if iteration > 0 **then**
    $\rho^{\text{old}} = \rho^{\text{new}}$
    `Allreduce`$([\mathbf{s}_j^T\mathbf{r}_j^0, \mathbf{t}_j^T\mathbf{s}_j, \mathbf{t}_j^T\mathbf{t}_j, \mathbf{t}_j^T\mathbf{r}_j^0, \mathbf{t}_j], [\mathbf{s}^T\mathbf{r}^0, \mathbf{t}^T\mathbf{s}, \mathbf{t}^T\mathbf{t}, \mathbf{t}^T\mathbf{r}^0, \mathbf{t}_\Sigma], \text{SUM})$
    $\omega = \frac{\mathbf{t}^T\mathbf{s}}{\mathbf{t}^T\mathbf{t}}; \rho^{\text{new}} = \mathbf{s}^T\mathbf{r}^0 - \omega\mathbf{t}^T\mathbf{r}^0; \beta = \frac{\rho^{\text{new}}}{\rho^{\text{old}}} \cdot \frac{\alpha}{\omega}$
    $\rho^{\text{old}} = \rho^{\text{new}}$
    $\mathbf{r}_j = \mathbf{s}_j - \omega\mathbf{t}_j; \mathbf{r}_\Sigma = \mathbf{s}_\Sigma - \omega\mathbf{t}_\Sigma$
    $\mathbf{f}_j = \mathbf{f}_j + \omega\mathbf{s}_j$
    check convergence on $\mathbf{r}$
  end if
  $\mathbf{p}_j = \beta(\mathbf{p}_j - \omega\mathbf{v}_j) + \mathbf{r}_j$
  $\mathbf{p}_\Sigma = \beta(\mathbf{p}_\Sigma - \omega\mathbf{v}_\Sigma) + \mathbf{r}_\Sigma$
  $\mathbf{v}_j = \mathbf{p}_j + S_j(\mathbf{p}_\Sigma - \mathbf{p}_j)$
  `Allreduce`$([\mathbf{v}_j^T\mathbf{r}_j^0, \mathbf{v}_j], [\mathbf{v}^T\mathbf{r}^0, \mathbf{v}_\Sigma], \text{SUM})$
  $\alpha = \rho^{\text{old}}/\mathbf{v}^T\mathbf{r}^0$
  $\mathbf{s}_j = \mathbf{r}_j - \alpha\mathbf{v}_j; \mathbf{s}_\Sigma = \mathbf{r}_\Sigma - \alpha\mathbf{v}_\Sigma$
  $\mathbf{f}_j = \mathbf{f}_j + \alpha\mathbf{p}_j$
  check convergence on $\mathbf{s}$
  $\mathbf{t}_j = \mathbf{s}_j + S_j(\mathbf{s}_\Sigma - \mathbf{s}_j)$
end while

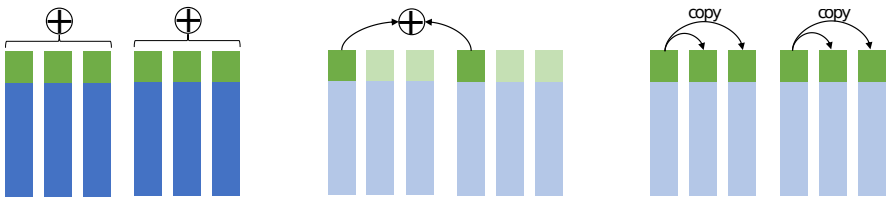


(a) Task and data decomposition of Equation (9).



(b) Data partitioning and transformation with kd-trees.

Fig. 1: Parallelisation of the fitting algorithm using task and data decomposition with kd-trees. On the left: Transformation of Equation (9). The coloured blocks are computed simultaneously. On the right: Kd-tree based partitioning (solid lines) of data with a low-dimensional manifold (black dots) and its orthogonalisation in individual partitions (red dots).



(a) Step 1. (b) Step 2. (c) Step 3.

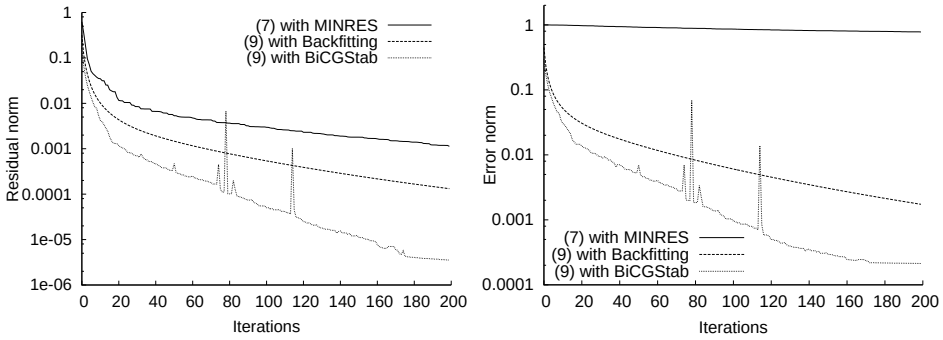
Fig. 2: Three-steps communication model for Lines 10 and 20 of Algorithm 1 with data partitioning.

4 Results

In this section we show results that highlight the properties of the presented fitting algorithm. We begin by comparing the convergence of different problem formulations and fitting methods. Then we discuss the impact of data partitioning and normalisation as well as show strong scaling results of the distributed parallel implementation of Algorithm 1.

To motivate the use of the smoothing matrix formulation and normal equations of the form (9) instead of (7), we begin by illustrating the convergence speed of the same problem in these two formulations.

The synthetic dataset used in this experiment was generated from a linear model with random coefficients and a small additive noise term. The dataset has 100 dimensions, whereas only 10 of them are informative.



(a) Residual (b) Error

Fig. 3: Comparison of problem formulations (7) and (9) for minimisation of residual and error of a synthetic 100-dimensional dataset from model. Fitting of 1,000 data points was performed using regression cubic splines with $\lambda = 10^{-7}$ and dof=10.

Figure 3 illustrates the relative residual and prediction error norms. One can see the superiority of the problem formulation (9) both by using the classical backfitting algorithm with Gauss-Seidel iterations and by using the BiCGStab

method. While the residual in the formulation (7) decreases, this does not considerably affect the error of the resulting additive model, which is our main goal.

We observed that the convergence of the backfitting algorithm is usually more stable but much slower. As BiCGStab does not directly minimise the error norm, the spikes as seen on Fig. 3 are not uncommon, although the algorithm would usually continue to converge after a spike.

The comparison between ADMM and BiCGStab is presented on Fig. 4. We use the Sloan Digital Sky Survey dataset in Data Release 5 (SDSS) [17] to predict the photometric redshift of galaxies based on 6 cosmological parameters [2]. Both fitting methods are comparable, although the BiCGStab-based method exhibits a faster convergence at the beginning. A parallel implementation of the backfitting algorithm with red-black Gauss-Seidel iterations would not converge.

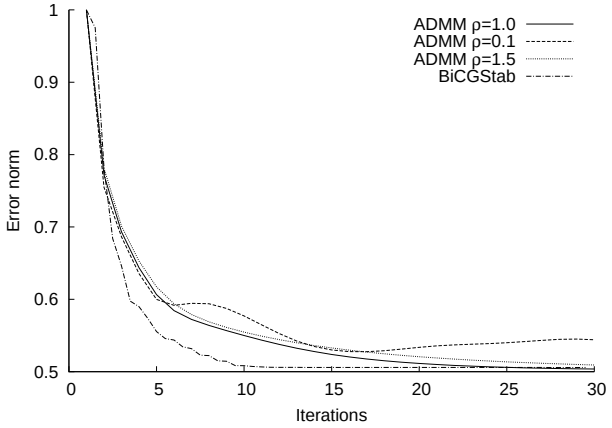


Fig. 4: Training error using ADMM and BiCGStab for regression on SDSS with 60,000 entries and random subspace projection ensemble with five 3-dimensional sparse grids with level 3 and $\lambda = 10^{-6}$.

As an example of a high-dimensional prediction problem we consider the year prediction task on the million song dataset [18] with 90 features and 463,715 examples in the training dataset. We use the first 88 features which is more convenient to split across a different number of processors. We use linear models as smoothing operators $\mathbf{S}_j$, however the extension to other linear basis expansion models is straightforwards.

As we described in the previous section, partitioning the dataset into subdomains and normalisation of individual partitions can improve the performance and accelerate the convergence of additive models. Figure 5 compares the relative residual norm and the mean prediction error for the same model with and without the orthogonalisation step. With orthogonalisation the algorithm converges after the first step.

Figure 6a illustrates the strong scaling of the algorithm for execution of 45 iterations using between 2 and 256 processors. There is a trade-off between the reduction of computation time and the increase of communication overhead (Fig. 6b). For up to 8 processors every process receives all examples, starting from 16 processors we introduce data partitioning, which is responsible for a jump in communication overhead. An application of more computationally intensive basis

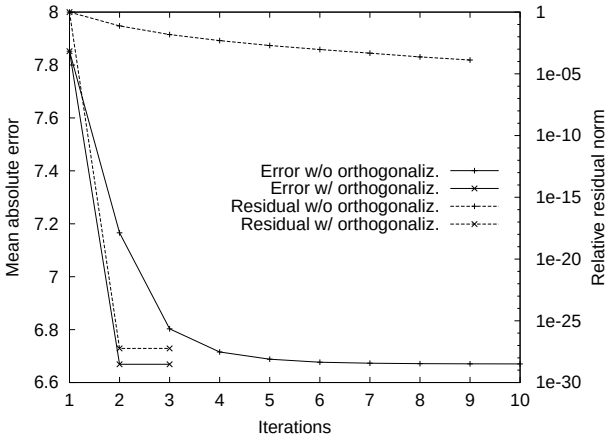


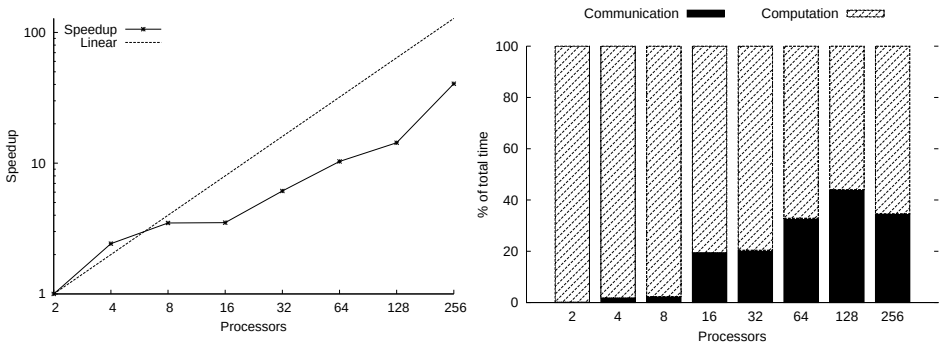
Fig. 5: Comparison of the relative residual norm and the mean prediction error for the same model with and without the orthogonalisation step on the million song dataset.

functions or smoothing kernels would improve the computation/communication ratio.

5 Conclusion

We presented a new approach for fitting additive models using BiCGStab algorithm that can be used for large-scale regression problems. While the convergence of the BiCGStab method cannot be proved theoretically, it usually works well in practice. It converges fast and can be efficiently parallelised for distributed computer architecture.

The method relates to ADMM for distributed optimisation and shows comparable convergence speed. Our future work will include the development of preconditioning methods to stabilise and accelerate the convergence.



(a) Strong scaling

(b) Computation/communication

Fig. 6: Scaling properties of Algorithm 1. On the left: strong scaling results for the million song dataset with 88 features and 463,715 examples. On the right: the computation-to-communication relationship for different number of processors.

References

1. J. Garcke, M. Griebel, and M. Thess, "Data mining with sparse grids," *Computing*, vol. 67, no. 3, pp. 225–253, 2001.
2. D. Pflüger, *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. München: Verlag Dr. Hut, Aug. 2010.
3. A. Heinecke and D. Pflüger, "Emerging architectures enable to boost massively parallel data mining using adaptive sparse grids," *International Journal of Parallel Programming*, pp. 1–43, July 2012.
4. W. Xu and W. Xu, "Towards optimal one pass large scale learning with averaged stochastic gradient descent," *CoRR*, vol. abs/1107.2490, 2011.
5. R. Bellman, *Adaptive Control Processes: A Guided Tour*. 'Rand Corporation. Research studies, Princeton University Press, 1961.
6. E. Novak and H. Woźniakowski, "Approximation of infinitely differentiable multivariate functions is intractable," *Journal of Complexity*, vol. 25, pp. 398–404, Aug. 2009.
7. M. Hegland and G. W. Wasilkowski, "On tractability of approximation in special function spaces," *J. Complex.*, vol. 29, pp. 76–91, Feb. 2013.
8. C. J. Stone, "The dimensionality reduction principle for generalized additive models," *The Annals of Statistics*, vol. 14, pp. 590–606, June 1986.
9. T. Hastie and R. Tibshirani, *Generalized Additive Models*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability, Taylor & Francis, 1990.
10. A. Buja, T. Hastie, and R. Tibshirani, "Linear smoothers and additive models," *The Annals of Statistics*, vol. 17, pp. 453–510, June 1989.
11. E. Chu, A. Keshavarz, and S. Boyd, "A distributed algorithm for fitting generalized additive models," *Optimization and Engineering*, vol. 14, pp. 213–224, Mar. 2013.
12. D. Hsu, N. Karampatziakis, J. Langford, and A. Smola, *Parallel online learning*, ch. 14. Cambridge University Press, 2011.
13. T. K. Ho, "The random subspace method for constructing decision forests," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, pp. 832–844, Aug 1998.
14. H. van der Vorst, "Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 13, no. 2, pp. 631–644, 1992.
15. L. Yang and R. P. Brent, "The improved bicgstab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures," in *Algorithms and Architectures for Parallel Processing, 2002. Proceedings. Fifth International Conference on*, pp. 324–328, Oct 2002.
16. C. Guangliang and M. Maggioni, "Multiscale geometric wavelets for the analysis of point clouds," in *Information Sciences and Systems (CISS), 2010 44th Annual Conference on*, pp. 1–6, March 2010.
17. J. K. Adelman-McCarthy et al., "The fifth data release of the sloan digital sky survey," *The Astrophysical Journal Supplement Series*, vol. 172, no. 2, p. 634, 2007.
18. T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.