
Data Mining II, C.A. Brebbia & N.F.F. Ebecken (Editors)
© 2000 WIT Press, www.witpress.com, ISBN 1-85312-821-X

Scalable parallel algorithms for predictive modelling

P. Christen[1], M. Hegland[1], O. Nielsen[1], S. Roberts[2], I. Altas[3]
[1] *Computer Sciences Laboratory, RSISE,*
Australian National University, Canberra, Australia.
[2] *School of Mathematical Sciences,*
Australian National University, Canberra, Australia.
[3] *School of Information Studies*
Charles Sturt University, Wagga Wagga, Australia.

Abstract

Data Mining applications have to deal with increasingly large data sets and complexity. Only algorithms which scale linearly with data size are feasible. We present parallel regression algorithms which after a few initial scans of the data compute predictive models for data mining and do not require further access to the data. In addition, we describe various ways of dealing with the complexity (high dimensionality) of the data. Three methods are presented for three different ranges of attribute numbers. They use ideas from the finite element method and are based on penalised least squares fits using sparse grids and additive models for intermediate and very high dimensional data. Computational experiments confirm scalability both with respect to data size and number of processors.

1 Introduction

Algorithms applied in data mining have to deal with two major challenges: Large data sets and high dimensions. In recent years, data sets had the size of Gigabytes but Terabyte data collections are being used in business and the first Petabyte collections are appearing in science (Düllmann [5]). It has also been suggested that the size of databases in an average company doubles every 18 months (Bell and Gray [1]) which is akin to the growth of hardware performance according to Moore's law. Consequently,

Data Mining II, C.A. Brebbia & N.F.F. Ebecken (Editors)
© 2000 WIT Press, www.witpress.com, ISBN 1-85312-821-X

data mining algorithms have to be able to scale from smaller to larger data sizes when more data becomes available. The complexity of data is also growing as more attributes tend to be logged in each record. Data mining algorithms must, therefore, also be able to handle high dimensions in order to process such data sets. This combination of large data size with high data complexity poses a formidable challenge for all data mining algorithms and parallel processing is a must in order to get reasonable response times. Moreover, algorithms which do not scale linearly with data size are not feasible. In this paper, we present scalable parallel algorithms for predictive modelling that successfully deal with these issues and are being applied to data mining problems where data sizes consist of about 18 million records with a dimensionality as large as 38. Section 2 presents the basic algorithms, Section 3 gives a definition of full scalability, Section 4 presents three different variants for different ranges of dimensions, and Section 5 describes the implementations.

2 A finite element approach to data mining

All major data mining packages include predictive modelling components. They are often used in combination with clustering, where first clusters are determined and then a predictive model is built to predict the cluster for a record of attributes. Thus, the predictive models lead to an understandable representation of the clusters. As a predictive model describes in some way the average behaviour of a data set, one may use it to find data records for which the values of the response is significantly different to the predicted value. These deviations from the "norm" often have simple natural explanations but, in some cases, may be linked to fraudulent behaviour.

A predictive model is described by a function $y = f(x_1, \ldots, x_d)$ from the set T of attribute vectors of length d into the the response set S. If S is discrete (often binary), the determination of f is the *classification problem* and if S is a set of real numbers, one speaks of *regression*. In the following it will mainly be assumed that all the attributes x_i as well as y are real values and we set $\boldsymbol{x} = (x_1, \ldots, x_d)^T$.

In many applications, the response variable y is known to depend in a smooth way on the values of the attributes so it is natural to compute f as a least squares approximation to the data with an additional smoothness constraint imposed. In this paper, we state the problem formally as follows: Given n data points $(\boldsymbol{x}^{(i)}, y^{(i)}), i = 1, \ldots, n$ where $\boldsymbol{x}^{(i)} \in \mathbf{R}^d$ and $y^{(i)} \in \mathbf{R}$ we wish to minimise the functional

$$J_\alpha(f) = \|f(\boldsymbol{x}) - y\|_2^2 + \alpha \int_{\mathbf{R}^d} |\mathcal{L}f(\boldsymbol{x})|^2 \, d\boldsymbol{x} \tag{1}$$

where $\mathcal{L}$ is a differential operator which maps real functions of d variables into real vector-valued functions of d variables. Examples include the gradient, Laplacian, Hessian etc. The smoothing parameter α controls the

trade-off between smoothness and fit: In the limit $\alpha \to 0$ the function f becomes an interpolant. If α is large, f becomes very smooth but may not reflect the data very well. Techniques for choosing α appropriately is beyond the scope of this paper and we refer to Wahba [12] for details on this matter.

For computational purposes f is often restricted further to a particular functional space, for example the space of piecewise multilinear functions, and different methods arise from these choices. In this paper, we describe and compare three different choices that we have implemented as tools and used for predictive modelling. These are:

- **TPSFEM**: Piecewise multilinear finite elements
- **HISURF**: Hierarchical finite elements (Bi-orthogonal wavelets)
- **ADDFIT**: Additive models

All of these tools approximate the minimiser in Equation (1) but they differ in how well they approximate f and more importantly in their algorithmic complexities. Roughly speaking, TPSFEM gives the most accurate approximation at the highest computational cost whereas ADDFIT has the lowest cost but the coarsest approximation. HISURF sits somewhere in between these two extremes and provides good approximations at a reasonable cost. Details about these methods are given in Section 4. All three methods follow the same general computational procedure as outlined below.

Using a Galerkin projection of Equation (1) onto the chosen functional space we obtain the $m \times m$ linear system

$$\left(\boldsymbol{M}^T\boldsymbol{M} + \alpha\boldsymbol{L}\right)\boldsymbol{f} = \boldsymbol{M}^T\boldsymbol{y} \tag{2}$$

where the vector $\boldsymbol{f}$ consists of approximate function values of f. The matrix $\boldsymbol{M}$ interpolates values in $\boldsymbol{f}$ to the n values that approximate the data points $\boldsymbol{y}$ and $\boldsymbol{L}$ is a discretisation of the differential operator $\mathcal{L}$. The interpolation matrix $\boldsymbol{M}$ has n rows and m columns, where m denotes the degrees of freedom in $\boldsymbol{f}$. Normally, m is chosen such that $m << n$. Computing $\boldsymbol{f}$ then consists of two steps:

1. **Assembly**: The $m \times m$ matrix $\boldsymbol{M}^T\boldsymbol{M}$ and the $m \times 1$ vector $\boldsymbol{M}^T\boldsymbol{y}$ are assembled. This step requires access to all n data points and it can be organised such that the computational work is linear in n. If $m < n$ this is a reduction operation on the original data.
2. **Solving**: This step assembles the $m \times m$ penalty matrix and solves the entire system for $\boldsymbol{f}$. This step does not involve the n data points directly and the computational work depends only on m, typically as $O(m^3)$.

Note that for large n step 1 will dominate whereas for large m step 2 will dominate.

Data Mining II, C.A. Brebbia & N.F.F. Ebecken (Editors)
© 2000 WIT Press, www.witpress.com, ISBN 1-85312-821-X

3 Full scalability and parallel reduction algorithms

An algorithm is called *scalable with respect to the data size* n if it has a linear complexity such that the processing time $T(n)$ is of the form

$$T(n) = \theta n + \beta. \tag{3}$$

If the data mining algorithm reads all the data, its complexity cannot be less than $O(n)$. Scalability is essential in order to be able to analyse large and growing data sets.

In parallel computing, scalability means that good parallel speedup is obtained for large enough problem sizes. We assume that all processors have access to the disk (possibly local). Let $T(n,p)$ be the time required for p processors to analyse n records of data. The *parallel efficiency* is defined as

$$E(n,p) = \frac{T(n,1)}{p\,T(n,p)}.$$

We say that the algorithm is *scalable with respect to the number of processors* p if there is a constant $E_\infty > 0$ independent of the number of processors such that

$$\lim_{n\to\infty} E(n,p) = E_\infty > 0, \quad p = 1,2,3,\ldots.$$

Ideally, $E_\infty \approx 1$. Parallel data mining algorithms require both types of scalability and we suggest to call this *full scalability*. One gets

Lemma 1. *A parallel algorithm is* fully scalable *if and only if*

$$T(n,p) = \frac{\gamma\, n}{p} + \beta(p) \tag{4}$$

where p is the number of processors and n is the data size.

Proof. The "constants" in Equation (3) may depend on the number of processors. Thus, an algorithm satisfying Equation (4) is scalable with respect to data size and an algorithm which is scalable with respect to data size satisfies

$$T(n,p) = \theta(p)\, n + \beta(p).$$

For such an algorithm the limit of the parallel efficiency is

$$E_\infty = \lim_{n\to\infty} E(n,p) = \frac{\theta(1)}{p\,\theta(p)}.$$

Thus one has parallel scalability if and only if

$$\theta(p) = \frac{\theta(1)}{pE_\infty} = \frac{\gamma}{p}.$$

□

Data Mining II, C.A. Brebbia & N.F.F. Ebecken (Editors)
© 2000 WIT Press, www.witpress.com, ISBN 1-85312-821-X

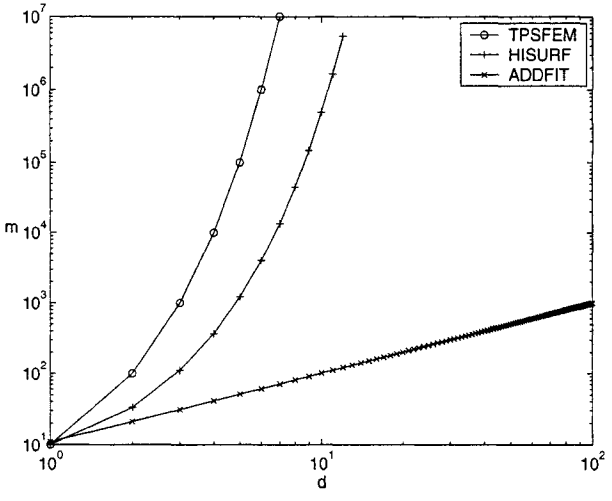

Figure 1: Dimension m as a function of the number of attributes d.

For predictive modelling both the determination of the model as well as the evaluation on a large data set needs to be fully scalable. The evaluation on the full data set is, for example, required in order to detect "unusual" records.

Fully scalable algorithms may be built around sampling of the data set, aggregation of tables into small (or at least manageable) tables, or binning. These algorithms all determine the predictive model in two steps: First, the data is scanned one to three times and reduced to a manageable size. Second, the model is determined from the reduced data (Weiss and Indurkhya [13]). We suggest calling algorithms of this type *reduction algorithms*. The size of the reduced data set needs to be independent of the data size but typically will have an influence on the approximation error or bias. Reduction algorithms with a fully scalable reduction step are fully scalable. Note that the second term $\beta(p)$ is the time needed to process the reduced data. If this stage is also processed in parallel, $\beta(p)$ decreases with increasing p.

4 Dealing with high dimensions

The three methods TPSFEM, HISURF and ADDFIT presented in Section 2 differ mainly in their approximation properties and the size of the models generated, see Figure 1. In Figure 2, the result of the application of these three techniques to aeromagnetic data available from Australian Society of Exploration Geophysicists (`http://www.aseg.org.au/`) is shown. All three techniques are fully scalable as defined in Section 3. MATLAB prototypes of the codes are available from the authors on request.

4.1 TPSFEM (Hegland et. al. [8, 3])

The *Thin Plate Spline Finite Element* approach uses a nonconforming finite element approximation to the thin plate splines introduced in Duchon [4] for one to three dimensions. In more than three dimensions, a term of the form $\int f_{x_1,\ldots,x_d}(x)^2 dx$ is added as the usual smoothing term (which is the norm of second derivatives) that does not yield continuous smooths for higher than three dimensions. It turns out that the "curse of dimensionality" is a major obstacle to handling higher than 3 to 5 dimensional spaces as the dimension of the finite element space grows exponentially in the number of variables d, see Figure 1.

4.2 HISURF

This tool uses a hierarchical piecewise linear bi-orthogonal wavelet basis and tensor products thereof to approximate f. It can be shown (Nielsen et. al. [9]) that many elements of this particular basis can be deactivated without sacrificing the essential approximation power. The error is increased by a factor $\log_2(s)^{(d-1)/2}$ by this procedure where s denotes the number of grid points in each dimension. In return, the dimension of the truncated system is of the order $m \approx (\log s)^{d-1} s$; which is a significant reduction compared to TPSFEM, especially for large s.

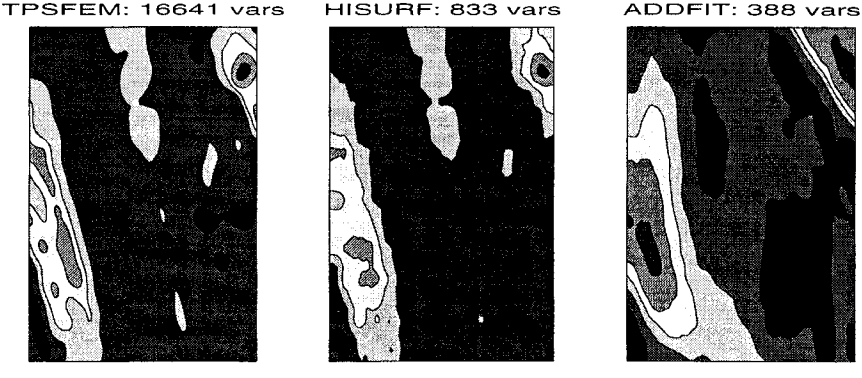

Figure 2: Modelling 2D aeromagnetic data with 735,700 records.

4.3 ADDFIT

We have adopted (generalised) additive models (Hastie and Tibshirani [7]) to deal with very high dimensional data. Figure 1 clearly shows its superiority with respect to complexity. However, this comes at a cost as the approximation power of additive models is much poorer as can be seen for the magnetisation example in Figure 2. Both advantages and disadvantages

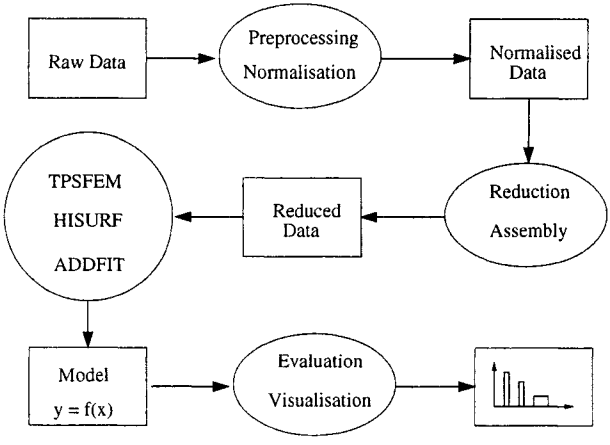

Figure 3: Steps of our data mining algorithm

are due to the simple form of additive models:

$$f(\boldsymbol{x}) = f_0 + \sum_{s=1}^{d} f_s(x_s).$$

All the functions f_s are piecewise linear and so the dimension of the finite element space is $pd + 1$. In Figure 2, the combinations $z_1 = x_1 + x_2$ and $z_2 = x_1 - x_2$ were used as variables and by adding other linear combinations one may be able to get better approximations. While the finer details are lost in this simple representation, one can still see the main features (like extrema) of the field.

5 Architecture and implementation

As discussed before, we would like to achieve full scalability as defined in Lemma 1. To obtain scalability with respect to data size, our algorithms are designed to access the data from secondary storage in one single pass. Once the data is read and the matrix and vector are assembled the predictive algorithms work on the reduced representations only.

Achieving a scalable parallel algorithm is mainly based on the requirement that the amount of communication must not depend on the number of data records n, as communication costs would dominate otherwise. The only steps that deal with the original data set (and thus depend on n) are the preprocessing (which has to be done once only for a data set) and the assembly of the matrices needed by the three methods TPSFEM, HISURF and ADDFIT. We describe these steps here in more detail.

Assume that we have a d dimensional data set (i.e. d attributes). In a first preprocessing step, continuous variables are normalised (i.e. they are

© 2000 WIT Press, www.witpress.com, ISBN 1-85312-821-X

transformed to values between 0.0 and 1.0) and categorical variables are enumerated (i.e. categories are numbered starting from 1). This normalisation process can be parallelised almost ideally as each processor has to process d/p attributes. Unfortunately, this may lead to load imbalance, as some processors may have to process one more attribute than others. Moreover, the time to process a categorical variable depends heavily on the number of categories in an attribute, as each data record has to be checked against all already existing categories.

For an example data set consisting of $n = 17,898,598$ data records each with $d = 38$ attributes we measured the following times on a Sun Enterprise 4000 SMP machine with ten 167 MHz Ultra-Sparc processors, 4.75 GBytes of main memory and an attached 256 GBytes RAID disk array: The first processor finished preprocessing and normalisation of four attributes (with at most 26 categories) after 70 minutes, while it took 91 minutes to finish for the last processor also with four attributes but with $2,711,766$ categories in one attribute. Since the number of categories is often unknown prior to preprocessing, it is hard to achieve good load balancing. For continuous – as well as categorical variables with only a small number of categories – good load balancing can be achieved, as the time in such a case is bounded by file access.

Once the data files have been normalised they are available as binary files on disc. The next step, which also requires access to the data, is the assembly of the matrix and vector as needed by the three methods. For each method a different matrix has to be assembled, but the structure of the assembly process remains the same. Basically, each data record adds some values into the matrix and vector at specific places.

As the assembly of each data record is independent of all others, it can be parallelised such that each processor assembles a fraction n/p of all data records into a local matrix and vector. To do so, each processor opens all data files and seeks the position in the files where its partition begins. At the end of the assembly, all local matrices and vectors have to be collected and summed to form the global matrix and vector. The dimension m of this matrix and vector only depends on the number of categories for categorical attributes and the number of grid points for continuous attributes, respectively. Most importantly, it is independent of the number of data records n. Load balancing in this step is easily achieved, because each processor reads and processes the same number of data records.

Today, most parallel computers are time-shared among several users, so the global load of a processor can dynamically change over time. To overcome this, we implemented the assembly step in a master-slave fashion, which automatically results in a good balanced load on all processors. The master sends the partition and number of data records to assemble to a slave, which assembles them and returns a message back to the master after it is finished. The master then sends another message to this slave. This is repeated until no more data records are left to assemble.

Table 1: Timing results for matrix and vector assembly.

Number of processes	1	2	4	6	8	10
Time for assembly [sec]	7484	3855	2018	1345	1046	983
Speedup	–	1.94	3.70	5.56	7.16	7.61
Efficiency	–	0.97	0.93	0.93	0.89	0.76

Table 1 shows preliminary results for the assembly of $n = 17,898,598$ data records into a dense matrix of dimension $m = 752$ on the 10 processor Sun Enterprise 4000. During the measurements we had to share the machine with other users, and so no optimal speedup results could be achieved.

After the assembly of the matrix and right-hand side vector this linear system can be solved depending on the matrix dimension by a sequential or parallel linear system solver like ScaLAPACK (Blackford et. al. [2]) or Aztec (Tuminaro et. al. [10]).

The current implementation of our methods is done in a modular way. For computing intensive processes we utilise parallel codes written in C and using MPI (Gropp et. al. [6]) for communication as described above. With the portability of MPI we can easily run our codes on workstation clusters, distributed as well as shared memory machines.

Of course, data mining not only consists of building models (see Figure 3). Preprocessing and cleaning of the data have to be done first, and results should be presented graphically to the user. We use the scripting language *Python* (van Rossum [11]) to glue all our modules together and give a consistent interface to the user. As our parallel programs mainly take inputs from file and write outputs to file they can be called from Python by simple system calls. Computationally intensive preprocessing is done by our parallel code as described above. More complex preprocessing and data aggregation can easily be programmed in Python. Another way is using a database (e.g. *MySQL*) and importing its interface into Python for easier access. All in all, we get a powerful, shell-like interface to mine huge and high-dimensional data sets with our predictive modelling methods.

6 Conclusion and outlook

We have presented a three-fold approach to predictive modelling for very large data-sets. They are all based on approximate smoothing splines using finite element techniques. While all the techniques are equally scalable, both with respect to the data size and the number of processors used, they differ in their approximation power and range of dimensions for which they are applicable. A first technique, called TPSFEM, is based on piecewise multilinear elements and is most suited to low-dimensional problems. For the intermediate range of dimensions we suggest HISURF which uses a

sparse-grid type approach based on bi-orthogonal wavelets. Finally, for very high-dimensions we have developed ADDFIT implementing additive models. In future work we plan to add support for data types other than continuous and categorical variables. We hope to include in a first instance support for sets, time series and graphs. We plan also to explore how the reduction techniques could be used to address data mining problems other than prediction.

Acknowledgements

This research was supported by the Advanced Computational Systems CRC and by the Swiss and Danish National Science Foundations.

References

[1] G. Bell and J.N. Gray, *The revolution yet to happen*, Beyond Calculation (P.J. Denning and R.M. Metcalfe, eds.), Springer Verlag, 1997, pp. 5–32.

[2] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley, *Scalapack user's guide*, SIAM, 1997.

[3] P. Christen, I. Altas, M. Hegland, S. Roberts, K. Burrage, and R. Sidje, *A parallel finite element surface fitting algorithm for data mining*, CTAC-99, 1999.

[4] J. Duchon, *Splines minimizing rotation-invariant semi-norms in Sobolev spaces*, Constructive theory of functions of several variables (Proc. Conf., Math. Res. Inst., Oberwolfach, 1976), Springer, Berlin, 1977, pp. 85–100. Lecture Notes in Math., Vol. 571.

[5] D. Düllmann, *Petabyte databases*, SIGMOD '99, ACM, 1999.

[6] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable parallel programming with the message-passing interface*, MIT Press, Cambridge, MA, USA, 1994.

[7] T.J. Hastie and R.J. Tibshirani, *Generalized additive models*, Monographs on statistics and applied probability, vol. 43, Chapman and Hall, 1990.

[8] M. Hegland, S. Roberts, and I. Altas, *Finite element thin plate splines for data mining applications*, Mathematical Methods for Curves and Surfaces, 1998, pp. 245–253.

[9] O.M. Nielsen, M. Hegland, and Z. Shen, *High dimensional smoothing based on multiresolution analysis*, In preparation.

[10] R.S. Tuminaro, M. Heroux, S.A. Hutchinson, and J.N. Shadid, *Official Aztec user's guide: Version 2.1*, Massive Parallel Computing Research Laboratory, Sandia National Laboratory, December, 1999.

[11] G. van Rossum, *Python tutorial*, Centrum voor Wiskunde en Informatica (CWI), 1995.

[12] G. Wahba, *Spline models for observational data*, CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 59, SIAM, 1990.

[13] S.M. Weiss and N. Indurkhya, *Predictive data mining, a practical guide*, Morgan Kaufmann Publishers, 1998.