The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

COST EFFECTIVE MACHINE LEARNING APPROACHES FOR LINEAR SOLVER SELECTION

A Thesis in

Computer Science and Engineering

by

Brice A. Toth

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2009

The thesis of Brice A. Toth was reviewed and approved* by the following:

Padma Raghavan
Professor of Computer Science and Engineering
Thesis Adviser

Sanjukta Bhowmick
Assistant Professor of Computer Science and Engineering

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

Numerical simulations are important in many areas of science and engineering. These simulations often involve the solution of large, sparse systems of linear equations. The solution time of these large, sparse linear systems dominates the execution time of the simulation. The choice of the linear system solution methods, in association with the characteristics of a given problem, directly influence the execution time. Selecting an effective solution method to match problem parameters is a key factor in reducing execution time and improving the performance and accuracy of the solution. The vast number of solution methods and the variability of the performance of solvers based on characteristics of a given problem make it difficult to determine the optimal method. Recent investigation in this domain have used machine learning techniques to choose an optimal solver. However, the computation of problem characteristics required by the learning algorithms are often time and memory intensive. For example, calculating problem characteristics, such as eigenvalues, can cause those techniques to take as long or more than running the solver.

In this thesis, we present a method which utilizes low cost machine learning classifiers to select a solver for a given sparse system. We present a method for reducing the feature space of model characteristics, henceforth features. We use machine learning algorithms to produce classifiers and then compare their predictive accuracy and time for construction. We observe that these classifiers are less computationally expensive than the solver computation time. We show that reducing our large set of features to a

smaller, cost-effective set saves computation time and can still be used to train machine learning algorithms to predict solvability of a given linear system within the same range of accuracy.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank Dr. Padma Raghavan for accepting me as an advisee. Dr. Raghavan's understanding of my situation as a non-traditional graduate student was much appreciated.

I am indebted to Dr. Sanjukta Bhowmick for her excellent tutelage. Dr. Bhowmick tirelessly read drafts and offered suggestions.

I would also like to thank Gabriel Burnett, Brandin Claar, and David Hadka for their counsel.

I would like to thank my wife, Heather, for her unending support throughout the creative process.

Most importantly, I would like to thank my parents George and Eleanor. Without their guidance and encouragement I would not be the person I am today.

Preface

The research described in this thesis is the culmination of several years of graduate courses and research in conjunction with a full-time software engineering position. As job responsibilities expanded, it became increasingly difficult to find sufficient time to apply to this project. The work documented herein, however, was both stimulating and personally rewarding making this accomplishment well worth the while.

Chapter 1

Introduction

The solution of sparse linear systems is a fundamental problem in many science and engineering applications, such as modeling the acoustic properties of the engine of an underwater vehicle [21], simulating the controls of a nuclear reactor core [16], representing atmospheric conditions to predict weather [25], and computing the effects of stress from wind or weight on a structure [7]. There exist many types of linear solvers, including direct [12], iterative [30], and multigrid [27] methods. Often preconditioners are used to improve iterative methods. The number of parametric options and the variability of performance for different problem characteristics for linear systems makes selecting an appropriate solver a challenging task. Computational characteristics of the linear systems are not known apriori. Therefore, it is an extremely difficult problem to determine the best solution method for a given linear system.

Because selecting a solver can be an arduous undertaking, machine learning has been employed in an effort to automate the process [15]. Execution time of a solver is the amount of time that it takes to converge to a solution. The number of linear system characteristics, or features, and cost of computing them affects the execution time of the machine learning algorithms. Characteristics in the feature set include matrix properties such as norms, trace, bandwidth, condition number, structural values, and other spectral

properties of a model. Depending on the features used, machine learning can be as costly as the execution time of a solver or worse.

In this thesis, we present a method for reliably selecting an effective solver for a given sparse system using low cost machine learning classifiers. We present a method for reducing the feature space of model characteristics and finding low cost features. We use different machine learning algorithms to produce a classifier, and compare their predictive accuracy and time for construction. We show that reducing our large set of characteristics to a smaller, cost-effective set saves computation time and can still be used to train machine learning algorithms to efficiently predict solvability of a given linear system.

We have used several open source software packages for our experiments. These packages include the Portable, Extensible Toolkit for Scientific Computation (PETSc) [3] which implements a suite of iterative linear solvers and preconditioners, AnaMod [13] which is part of the SALSA [10] package for functions to compute the complete feature set of each model, and Weka [32] for a suite of machine learning algorithms. We obtained matrices from the University of Florida Sparse Marix Collection [9] to form linear systems of the form, $Ax = b$.

This thesis is organized as follows. In Chapter 2, we discuss background information about iterative linear solvers and preconditioners as well as machine learning and feature reduction techniques. In Chapter 3, we present our method for producing low cost, high accuracy classifiers. In Chapter 4, we present the results of our methods. In Chapter 5, we discuss our concluding remarks and present potential ideas for further research.

Chapter 2

Review of Background Information

In this chapter, we include brief reviews of direct, iterative, and Krylov solvers as well as preconditioners. We then discuss feature selection and machine learning methods.

2.1 Sparse Linear Systems

Linear systems are represented in the form $Ax = b$, where A is an $m \times n$ coefficient matrix, b is a known vector of length m, and x is an unknown vector of length n. Solving such a system for a solution of x can have no solution, a unique solution, or infinite solutions. In this thesis, we will consider only non-singular matrices which have a unique solution.

A sparse matrix has few nonzero elements, compared to its number of rows. In order to reduce the storage space, zero elements are not stored. Compressed Row Storage [2] (CRS) is one general format method for storing sparse matrices that puts nonzeros of a matrix in contiguous memory locations. This is accomplished by creating three vectors, one to store floating point values and two to store column indices and row locations. CRS is illustrated in the example below.

$$A = \begin{bmatrix} 13 & 0 & 0 & 0 & 5 \\ 0 & 10 & 15 & -1 & 0 \\ 0 & 0 & 3 & 1 & 0 \\ 2 & 0 & 0 & 4 & 0 \\ 0 & 7 & 0 & 1 & -2 \end{bmatrix}$$

$$values = \begin{Bmatrix} 13 & 5 & 10 & 15 & -1 & 3 & 1 & 2 & 4 & 7 & 1 & -2 \end{Bmatrix}$$

$$columnindices = \begin{Bmatrix} 1 & 5 & 2 & 3 & 4 & 3 & 4 & 1 & 4 & 2 & 4 & 5 \end{Bmatrix}$$

$$rowpointer = \begin{Bmatrix} 1 & 3 & 6 & 8 & 10 & 13 \end{Bmatrix}$$

2.1.1 Direct Methods

Utilizing unique data structures to store nonzero elements of sparse matrices saves storage space and increases efficiency of solution methods. Direct methods often use an LU Factorization [23, 19] to compute the solution. A is represented as a product of two triangular matrices, a lower triangular matrix L and an upper triangular matrix U, so that $A = LU$. Our linear system represented by $Ax = b$ becomes $LUx = b$ and can be solved in two stages. Stage one solves $Lz = b$ for z and stage two solves $Ux = z$ for x. Several methods exist for determining L and U, including Doolittle's factorization, Crout's factorization, and Cholesky's factorization [23]. These methods are all variations of Gaussian elimination.

While direct methods produce an exact solution if one exists, they can still be costly in terms of storage and computation. LU Factorization overwrites the storage of A with U as the upper triangle and diagonal and L as the lower triangle. This step can

be computationally expensive depending on the values of the matrix. Factorization can create a problem called fill-in where a zero value can change to a non-zero as a result of matrix operations as shown in Saad [30]. Finally, two computationally expensive steps must be taken to arrive at the solution, namely $O(n^3)$ floating-point multiplications for the factorization and $O(n^2)$ multiplications for solving the right-hand-side vector [19].

2.1.2 Iterative Methods

Iterative methods start with an initial estimate and iterate to achieve a desired accuracy with each consecutive step. Unlike direct methods, iterations can stop when the residual of error is small. Often, the value of the residual, given by $\|b - Ax\|$, is used to determine a point to stop iterating. It is possible iterative methods may not converge to a solution, however storage considerations and speed of convergence make them attractive.

The Jacobi method is an iterative method where A is split into the form $M - N$, with M being the diagonal and N being $-(L + U)$. The iteration is given in Equation 2.1 [29]. This method only uses new values after the complete set of values has been computed, because of this, the method converges slowly.

$$x^{s+1} = M^{-1}\left(Nx^s + b\right) \tag{2.1}$$

The Gauss-Seidel method builds on Jacobi by setting M to $L + D$ and N to $-U$. Further, Gauss-Seidel uses new values as they are computed, increasing the convergence rate over Jacobi.

Another method is the Successive Overrelaxation method (SOR). A weighted average between the previous iteration and the current Gauss-Seidel iterate are used. This weighted average is represented by ω and is used to accelerate the convergence rate of the iterations [4]. SOR is shown in Equation 2.2 [29].

$$x^{s+1} = (D + \omega L)^{-1} \left((1 - \omega) D - \omega U \right) x^{s} + M^{-1} b \tag{2.2}$$

2.1.2.1 Krylov Methods

Krylov methods are based on orthogonal projections onto Krylov subspaces. Given two vectors, v_1 and v_2, we can say they are orthogonal to each other if $v_1^T v_2 = 0$. Krylov subspaces take the following form

$$K_m (A, v) \equiv span \left\{ v, Av, A^2 v, \ldots, A^{m-1} v \right\} \tag{2.3}$$

Given our linear system $Ax = b$, we approximate a solution x_m when $x_m \in K_m$ where $b - Ax_m \perp L_m$, if L_m is another subspace of dimension m. Different Krylov methods depend on different choices for the subspace L_m. In the simplest case, a Krylov method would result in $A^{-1} b \approx q_{m-1} (A) b$ where q_{m-1} is a polynomial approximation.

The Conjugate Gradient (CG) [4, 30, 19] method is for solving symmetric positive definite linear systems based on optimization. The algorithm takes advantage of the symmetry of A. The method works by using search direction vectors to update iterates and residuals. The parameter α is selected to minimize the search function. Iterates are updated by a multiple of the search vector so that each iteration is defined as $x^i =$

$x^{i-1} + \alpha_i p^i$. Residuals are updated similarly as $r^i = r^{i-1} - \alpha q^i$, where $q^i = Ap^i$. In turn, this parameter is used to update the iterate solution and compute the new residual. Each iteration calculates two inner products to satisfy orthogonality conditions. This algorithm can be seen in Appendix A, Algorithm (2) [4].

Since CG is not usable with nonsymmetric systems, BCG was developed using two mutually orthogonal sequences to rectify nonsymmetry with the unfortunate side effect of not providing a minimization. The Conjugate Gradient Squared (CGS) [4, 30] method is based on the BiConjugate Gradient (BCG) [4, 30] method. CGS was further developed to improve upon the performance of BCG. This idea is accmomplished by applying the polynomial twice in order to produce a smaller residual vector, $r^{(i)}$, and is shown in Appendix A, Algorithm (3).

Due to the squaring in the CGS algorithm, rounding errors can be potentially large [30]. The Stabilized Version of the BiConjugate Gradient method (BICGSTAB) [4] is designed to rectify this problem. Where CGS multiplies the polynomial used to calculate the residual during each iteration twice, BICGSTAB tries to stabilize this factor instead using a recurrence relation to produce the residual with a goal of stabilizing the convergence behavior. This is shown in Appendix A, Algorithm (4).

MINRES [4] is a method that can be used with symmetric indefinite systems. The method seeks to minimize the residual (hence the name) in the 2-norm using an orthonormal transformation relative to the Krylov subspace. This is shown in Appendix A, Algorithm (5).

An extension of MINRES which applies to unsymmetric systems, the Generlized Minimal Residual (GMRES) [4, 30] method generates orthogonal vectors and retains all

previously computed vectors due to the lack of symmetry, with restarts controlling use of storage space. After m iterations, accumulated data is cleared and the intermediate result is used as the initial data point for the next m iterations. However, poorly chosen restarts can cause stagnation due to the possibility that the number of iterations required to converge may not be met. Of note is that a residual norm $\left\|b = Ax^{(i)}\right\|$ can be calculated prior to formation of iterates which saves computations until the residual is of worthy size. This comes from the iterates being constructed as $x^{(i)} = x^{(0)} + y_1 v^{(1)} + \cdots + y_i v^{(i)}$ where y_k is chosen to minimize the residual. This is shown in Appendix A, Algorithm (6).

The Chebyshev Iteration [4] is used for solving nonsymmetric problems, and the algorithm itself is very similar to CG. This method avoids computations of inner products, which eliminates a performance bottleneck, however it requires spectral knowledge of the coefficient matrix A such that an ellipse around the spectrum can be defined. Scalars c and d are selected to define a family of ellipses that enclose the spectrum of A and to reduce the rate of convergence. This is shown in Appendix A, Algorithm (7).

The Richardson Method [23] uses an identity matrix I to formulate the iteration $x^{(k)} = (I - A)x^{(k-1)} + b = x^{(k-1)} + r^{(k-1)}$. A solution is reached if $\|I - A\| < 1$ for a subordinate matrix norm. This is shown in Appendix A, Algorithm (8).

The Transpose-Free Quasi-Minimal Residual Method (TFQMR) [30] is based on the CGS algorithm, where the initialization is the same but calculating x_j is done in two half-steps because the change between iterations is two matrix-by-vector multiplications. For convenience, we double all subscripts of the CGS algorithm when discussing TFQMR to avoid 1/2 indices. Thus, breaking the approximate solution into the half-steps gives

us $x_{2j+1} = x_{2j} + \alpha_{2j} u_{2j}$ and $x_{2j+2} = x_{2j+1} + \alpha_{2j} q_{2j}$. This is shown in Appendix A, Algorithm (9).

2.1.3 Preconditioners

Iterative methods are susceptible to slow convergence rates. Preconditioning is a sort of transformation that can be used on a linear system to give a new linear system that is meant to be easier to solve via an iterative method than the original system. A successful preconditioner would find a matrix M that is a related approximation of A or A^{-1} and converges to a solution faster than with A. Preconditioning adds cost at startup and per iteration, however, a successful preconditioner for a problem will pay off with faster convergence. A preconditioner that works well with one type of problem will not necessarily be useful for other problem types as properties of systems vary. For example a simple scaling may work well for one system yet make another system harder to solve. [30, 4] Previously mentioned iterative methods such as Jacobi, Gauss-Seidel, and SOR can be used as preconditioners.

Incomplete Factorization (ILU) [4, 30] generally computes upper and lower sparse triangular matrices L and U with the intent that the residual equals $LU - A$ while satisfying certain constraints. Often the constraints force matrix positions outside of a prescribed set to be equal to zero during the factorization. In a specific case, ILU(0), this prescribed set of positions is all non-zero values in the matrix. Original matrix elements would be considered level 0 and each additional level would be considered caused by elements of the previous level. A formal definition can be seen in Equation 2.4. We used ILU(0), ILU(1), ILU(2), and ILU(3) as preconditioners for our experiments.

$$\text{for each} \quad k,i,j > : \quad a_{i,j} \leftarrow \begin{cases} a_{i,j} - a_{i,k} a_{k,k}^{-1} a_{k,j} & \text{if} \quad (i,j) \in S \\ a_{i,j} & \text{otherwise} \end{cases} \tag{2.4}$$

Incomplete Cholesky Factorization (ICC) [19] can be classified as a sparse approximation of a Cholesky Factorization [17]. For a symmetric and positive definite system, the LU Factorization is determined to be $U = L^T$ or $A = LL^T$. ICC is instead a sparse lower triangular matrix K, which results in $A = KK^T$.

2.2 Machine Learning

Machine learning is a subset of artificial intelligence that focuses on the design of algorithms that increase performance over time, based on data. These algorithms make sense of a large set of data and try to determine patterns within it. Through machine learning we gain knowledge about the structure of our data and gain an understanding of the contents to explain what has been learned and serve as the basis for further predictions. We will use the terms machine learning and data mining interchangeably.

Within the field of machine learning, there exist unsupervised and supervised learning techniques. Unsupervised learning seeks to determine how data is organized where the learner is only given unlabeled examples. Supervised learning predicts the value of the function for any valid input object after having been primed with a number of training examples. Training examples consist of pairs of input objects and desired outputs. We use supervised learning in our experiments.

We use cross-validation to estimate how accurately a predictive model will perform in practice. Cross-validation [32] partitions the data k times. Data is randomly divided into k groups. Each group is separated out and the learning method trained on the remaining groups. An error rate is computed for the separate group. This continues until each group is trained (k times total) on each subset. The error estimates are then averaged to conclude an overall error rate and a final classifier is produced as a combination of each result.

IB1 and IBk [32] are lazy learners. Lazy learners store all training instances first and then perform classification, as opposed to methods that attempt to classify training instances as they are read. IB1 and IBk are also instance-based learners [28]. Instance-based learners are comprised of the following: (1) some distance metric, (2) how many neighbors are considered, (3) an optional weighting function, and (4) how to fit the test case with local points. IB1 stores all the training data and then a distance function determines which member of the training set is closest to an unknown, predicting the class of the test. IBk uses a k-nearest-neighbor [8] algorithm where the number of neighbors is specified as k. If predictions occur for more than one neighbor, they can be weighted by distance from the test case. Figure 2.1 illustrates classification differences for IB1 and IBk, where $k = 15$.

The Naive Bayes (NB) [32] method is based on Bayes' theorem, which relates the conditional probability as $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$. The method "naively" assumes independence of features, such that there is no relationship between the presence or absence of any feature with any other feature. Calculating probabilities from frequency of occurence avoids a situation where the probability is 0, which would cancel out all

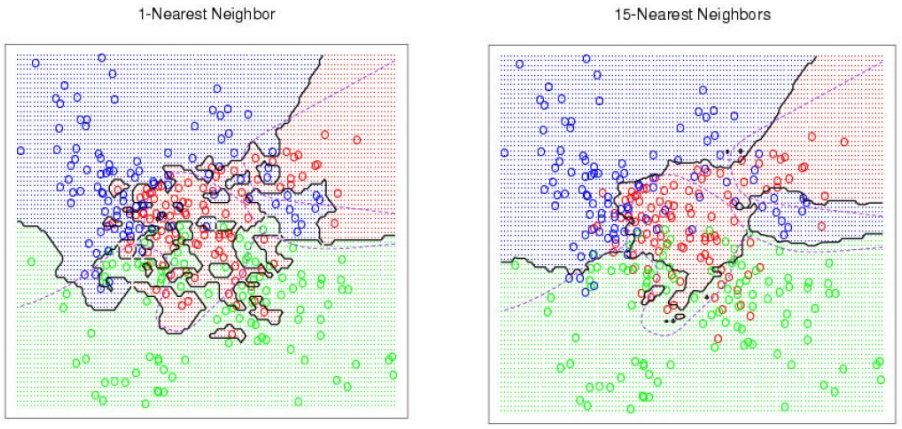

Fig. 2.1. IB1, IBk [31]

the others. Figure 2.2 shows an example where objects are classified as green or red on the left and on the right shows a white object added which will be colored red given the probability calculations for the system.

Even though such assumptions of independence may not reflect the real distribution, NB often works much better than expected in complex real-world situations as shown in Zhang [33]. NB is robust enough to ignore deficiencies in the naive probability model. Distributions can be independently estimated as a one dimensional distribution, so that data sets do not have to scale exponentially with the number of features. NB needs only a small amount of training data to estimate the means and variances of variables for classification and because of the assumed independence, only variances for each class need to be computed, not the entire covariance matrix.

Decision trees [32] are another machine learning technique. Building a decision tree is accomplished recursively by selecting a feature to place at the root and then

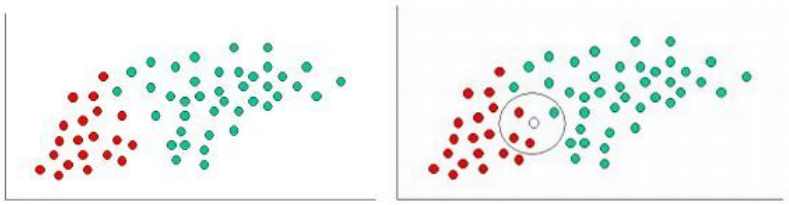

Fig. 2.2. Naive Bayes, Initial(l) and Pointed Added(r) [20]

making one branch for each possible value. Features are reduced to subsets and the process is repeated recursively. The larger the decision tree, the better the classification.

Decision Stump [32] is a decision tree that builds a one-level tree. It deals with missing values by treating them separately and extending an additional branch from the stump. The example shown in Figure 2.3 shows a one-level tree with possible outcomes if a variable $x_3 = 1$.

Alternating Decision Tree (ADT) [32] is a decision tree where the tree is grown by adding nodes to it through each iteration. In this method, decision nodes, indicating a conditional predicate, are referred to as splitter nodes and value nodes are referred to as prediction nodes. Prediction nodes are leaves if no splitter has been added to them. A splitter and two prediction nodes get added through each cycle unless nodes are the same, in which case they get merged. Figure 2.4 shows an ADT example where the splitter nodes are car properties and the prediction nodes are values related to those properties.

Support Vector Machine (SVM) [32] is a set of supervised learning methods which optimizes the maximum separation between two sets of data. SVM builds two sets of

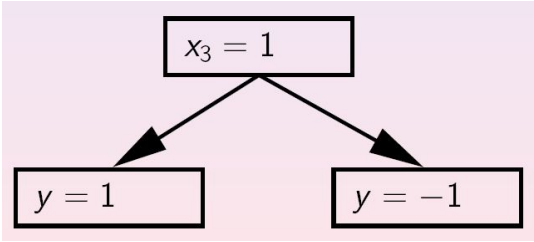

Fig. 2.3. Decision Stump, Example [5]

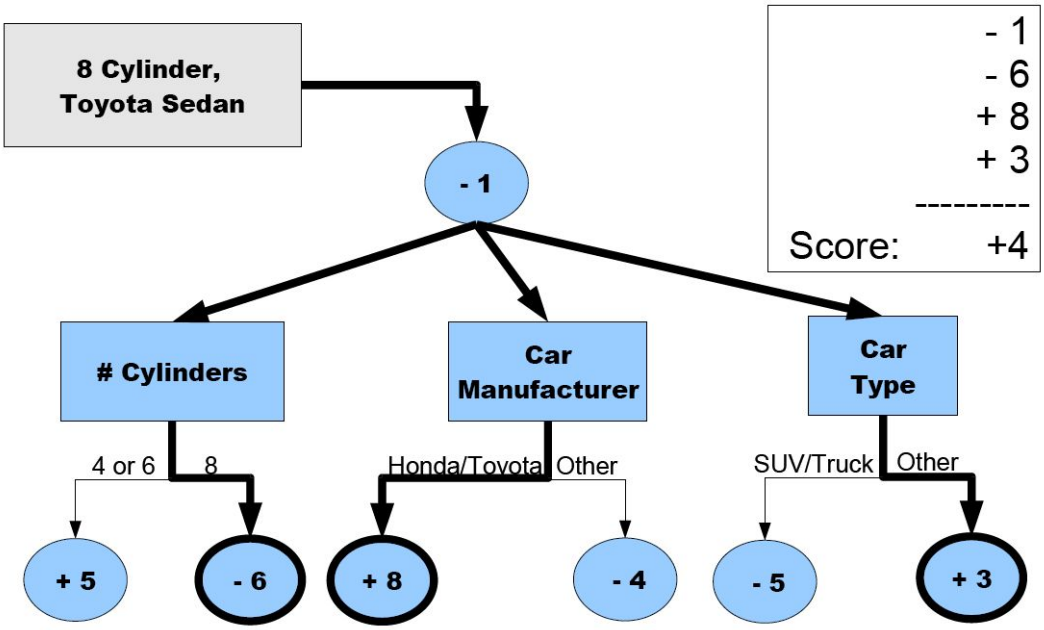

Fig. 2.4. Alternating Decision Tree, Car Example [1]

vectors in an n-dimensional space with a hyperplane maximizing the margin between the two sets. Extra nonlinear terms are included in the function yielding higher-order decision boundaries. Any hyperplane can be written as the set of points x that satisfy $w \cdot x - b = 0$. The vector w is perpendicular to the hyperplane. $\frac{b}{\|w\|}$ determines the offset of the hyperplane from the origin along w. We then select w, b to minimize $\|w\|$ according to $c_i(w \cdot x_i - b)$ for $i = 1, \cdots, n$, where c_i is either 1 or -1, indicating the class to which point x_i belongs, and each x_i is a p-dimensional vector. Figure 2.5 illustrates a linear hyperplane with support vectors circled.

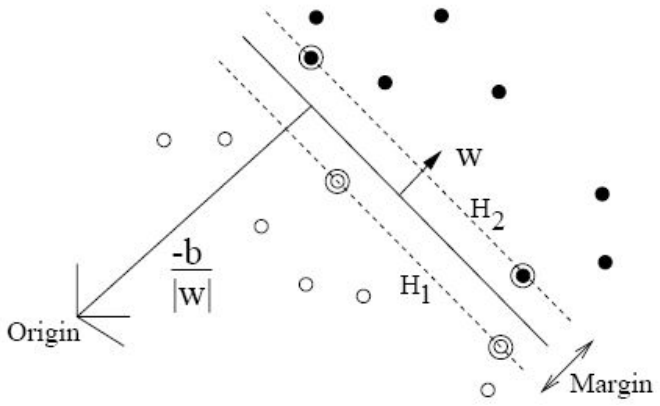

Fig. 2.5. Support Vector Machine, Example Hyperplane [6]

In our experiments, we found NB, IB1, IBk, ADT, and DS to compute classifiers in comparable amounts of time. However, SVM was significantly slower in all cases.

2.3 Feature Selection

Feature Selection [26] is a process for selecting a subset of features from a larger set of original features based on criteria not classification. The goal is to remove redundant, noisy, or irrelevant data [22] to then improve classification time and prediction accuracy. An optimal feature set is often measured by some evaluation criteria. The typical process for feature selection includes the following four steps: (1) subset generation, (2) evaluation, (3) stopping criteria, and (4) validation. Algorithms for feature selection can be broken into three categories: filter methods, wrapper methods, and hybrid methods.

Filter methods [24] apply a simple filter independently across a data set, which ignores the effects of the selected subset on the overall system. An example would be a variable ranking scheme where features are sorted according to some rank and then broken into subsets.

Wrapper methods [24] incorporate a search method to inspect all possible features and appraise each subset by fitting the subset to a model. The first step in building a wrapper is to define the search method, which can include exhaustive, best first, greedy forward selection, greedy backward elimination, genetic algorithms, etc. The accuracy of the classifier is then predicted, based upon a model that depends on the selected classifier, to guide the search and halt it.

Because they do not take into account the classification algorithm, it is possible that filter methods may select a feature that has a negative impact on classification. Wrapper methods can be computationally expensive depending on the data set. Taking

these caveats into consideration, hybrid methods [26], also know as embedded methods [18], are a combination of filters and wrappers where selection is part of the training process of the classifier. By combining techniques, an optimal filter can be predetermined and used as part of the model.

Chapter 3

On Classification Cost Reduction

This chapter describes our method for reducing the cost of building a high-accuracy classifier.

3.1 Problem Summary

We seek to produce high-accuracy, low-cost machine learning classifiers for linear solver selection. The time to generate classifiers is tied to the number of features involved, more features means higher cost. So, we reduce the cost of machine learning by reducing our feature set.

3.2 Feature Computation

The computation of features of linear systems vary greatly in cost. Features such as trace, diagonal average, and diagonal variance can be computed in $O(n)$ time. Matrix norms, structural properties, including diagonal dominance and left or right bandwidth, and sparsity data, including number of nonzero elements in the matrix and maximum number of nonzero elements per row can be computed in $O(nnz)$ time given an n by n matrix with nnz nonzeroes and are computed to their exact values [17]. Spectral properties such as the condition number, eigenvalues, and singular values can be more

expensive to compute than solving the linear system and often have to be approximated. For mathematical descriptions of features and their complexity see Table C.3.

Increasing the number of features does not necessarily contribute to a corresponding gain in the accuracy of the classifier. We show that by judiciously selecting features with a high amount of information to provide to the classification algorithms, we can maintain high-accuracy.

3.3 Low Cost Feature Selection

The size of a feature set impacts the computational cost of generating and using classifiers. Feature set size can be reduced by removing redundant features and those containing little information. Redundant features can occur because of certain linear system properties. For example, in a symmetric matrix, left-bandwidth and right-bandwidth would be redundant due to the structure of the matrix. These features need only be computed once to discover the redundancy and not needed again. Features with low information are ones that do not vary and hence do not contribute significantly to the resulting classifier because they do not provide the classifier with a representative diversity. Our method to reduce the cost of building machine learning classifiers for linear solver selection is done in two stages.

We begin by reducing the number of features in the set based on variance. Each feature has a distribution of values across the entire set. The variance of the values of each feature is a good metric of the statistical dispersion. We conclude that if the values of a feature are constant for all values in the set, then no extra information is available from that property and it can be removed. Similarly, if several features show exactly the

same variance, it is likely that a single feature from that set can be used and the others eliminated.

We complete our technique by ordering the remaining feature set. The number of features relates directly to the dimension of values generated by the classifier. Increasing the number of features results in potentially more accurate classification results because the classification algorithms have a higher dimension of values. Increasing the number of features, however, also increases the amount of time taken to build the classifier. Our solution is to construct a classifier by selecting few features based on increasing order of their computational complexity after the set has already been reduced by removing redundant and unprofitable features. This results in a high-accuracy, low-cost classifier.

Formally, we take the entire set of features, $F = f_1, f_2, \ldots, f_n$ and produce a smaller set $\hat{F}$ by eliminating low-information and redundant features from the set using statistical measures such as the variance, $\phi(f_i)$, and the mean, $\mu(f_i)$, of the individual features. This step is our filter method in the feature selection process as it is applied independently across the full set of features. The cardinality of the feature set $\hat{F}$ is further reduced to the final set F^* by ordering the remaining features in increasing order of their time for computation, $\psi(f_i)$, and adding these ordered features one by one to an initially empty set $\bar{F}$ until the accuracy of the classifier generated using $\bar{F}$, given by $K(\bar{F})$, is greater than a specified accuracy, A. This step is our wrapper method as we further eliminate features and organize the remaining features to get our desired accuracy. The final set of features in $\bar{F}$ gives the set F^*. An algorithm for this method is shown below:

Algorithm 1 Feature Reduction and Ordering Algorithm

Compute $\phi(f_i)$ and $\mu(f_i)$ for all $f_i \in F$.
Eliminate f_i if $\phi(f_i) \leq \alpha$.
for all $\tilde{F} \subset F$, where $\phi(f^i) = \phi(f^j)$ and $\mu(f^i) = \mu(f^j)$ **do**
  for all pairs $f_i, f_j \in \tilde{F}$ **do**
    Eliminate all but one feature f_k from $\tilde{F}$, such that $\psi(f_k) = min(\psi(f_i))$, for all $f_i \in \tilde{F}$. Use tie breaking if necessary.
  end for
end for
After completion of elimination we have the reduced set $\hat{F}$.
Order $\hat{F}$ such that if $\psi(\hat{f}_i) \leq \psi(\hat{f}_j)$, then $i < j$where $\hat{f}_i, \hat{f}_j \in \hat{F}$.
Set $\bar{F}$ to an empty set and $i = 0$.
while $K(\bar{F}) \leq A$ **do**
  $\bar{F} = \hat{f}_i \cup \bar{F}$.
  $i = i + 1$.
end while
Set $F^* = \bar{F}$

Chapter 4

Experimental Results

In this chapter, we present a detailed description of the methods used to produce our results, including the software packages utilized, the test suites of matrices, and our steps taken to arrive at our results. We then describe the outcomes of our method.

4.1 Software Utilized

PETSc is a package of data structures and methods for solving partial differential equations in various scientific applications. PETSc provides an abstract interface wherein users can easily call libraries of methods to be used in modeling scientific applications such as preconditioners, nonlinear solvers, and linear solvers [3]. PETSc is freely available for many platforms.

Anamod [13], which is part of the SALSA [10] package, computes matrix features such as norms, trace, bandwidth, condition number, structural values, and other spectral properties. A listing of important features along with a mathematical description and time complexity can be seen in Table C.3. The complete list of features can be seen in [14].

Weka is a suite of machine learning algorithms that contain tools for the following tasks: data pre-processing, classification, regression, clustering, association rules, and visualization. Weka is open source and written in Java so that it is not platform dependent and easily extendible.

4.2 Matrices

We have used two test suites, each of 50 matrices, shown in Tables C.1 and C.2, chosen from the University of Florida Sparse Matrix Collection [9] to give a diverse range of features, one which is symmetric matrices and the other which is unsymmetric matrices. We found, during the course of initial experimentation, a few matrices with extremely large condition numbers or with sparse patterns, would result in excessive addition of zeroes for some preconditioners and cause test programs to never reach completion due to their size and structure and removed these matrices. The linear systems were formed by taking $Ax = 0$, where A is the matrix from the test suite.

4.2.1 Matrix Features

In Table C.3 we compiled a listing of features computed by Anamod with descriptions and time complexity needed to compute them.

4.3 Linear Solvers

4.3.1 Symmetric Matrices

Using PETSc, we selected a set of ten Krylov methods and seven preconditioners, as well as one with no preconditioner, making a total of 80 solvers. The following

preconditioners were chosen for the tests: SOR, JACOBI, ICC, ILU with fill levels 0 through 3. The selected Krylov methods were: CG, CGS, BICGSTAB, MINRES, GMRES with restart values of 5, 30, and 60, Chebyshev, Richardson, and TFQMR. The total number of entries in our resulting set of symmetric matrices and solvers was 3969. A since a small number of entries was discarded due to the simulation failing to terminate.

Table D.1 shows the fastest and slowest solver, the number of solvers that failed to terminate, the median time to solve in seconds, and the mean time to solve in seconds for each matrix. The fastest and slowest solvers vary from matrix to matrix.

4.3.2 Unsymmetric Matrices

For the unsymmetric set of matrices, we had to eliminate the following Krylov methods: CG, Chebyshev, MINRES because they only work with symmetric matrices. This left us with 56 solvers for this set. The 2800 entries in our resulting set of unsymmetric matrices and solvers were further reduced to 1800 due to several simulations failing to terminate and the structure of the matrices conflicting in certain cases with the preconditioners.

Table D.2 shows the fastest and slowest solver, the number of solvers that failed to terminate, the median time to solve in seconds, and the mean time to solve in seconds for each matrix. The fastest and slowest solvers vary from matrix to matrix.

4.4 Machine Learning

For data pre-processing, we used filters from Weka, NumericToNominal and Discretize, on our data set. NumericToNominal was necessary because our data was in the form of numeric attributes and the classification algorithms required those values be converted to nominal values. Attributes that are nominal have values that are distinct symbols from a prespecified, finite set where the values serve as labels. Discretize then took those nominal values and organized them in equivalent bins. The following classification methods were chosen: NaiveBayesSimple, SVM, ADTree, DecisionStump, IB1, and IBk.

4.5 Analysis

In this section we describe the results of using our method to build low-cost, high-accuracy classifiers.

4.5.1 Feature Set Reduction

We calculate the variance of the features in each set and then eliminate all features whose variance is less than 10^2.

4.5.1.1 Symmetric Matrix Data Set

For the symmetric set, a graph of feature variance can be seen in Figure 4.1. We see that 20 features fall below the 10^2 threshold and can be removed. We observe that the range of accuracy is higher when the classifiers are generated from features in the high variance range. The maximum accuracy of a classifier generated using SVM with

one feature from the zero variance group is at most 63.69% accuracy whereas a classifier generated from the high variance group can result in as high as 80.67% accuracy. A comparison of accuracy for the zero and high variance groups for each set can be seen in Table 4.1.

Table 4.1. Maximum Accuracy for One Feature, SVM

Set	0 Variance Group	High Variance Group
Symmetric	63.69%	80.67%
Unsymmetric	36.73%	73.29%

After filtering out the low variance features, we identify redundant features like $(row-variability, col-variability)$, $(left-bandwidth, right-bandwidth)$, $(trace, trace-abs)$, redundant norms and spectral values. By filtering out all but one feature from each of these groups, we can further reduce our feature set. Due to the nature of the symmetric set, such as left-bandwidth and right-bandwidth being identical, there were more occurences of redundant features.

4.5.1.2 Unsymmetric Matrix Data Set

For the unsymmetric set, a graph of feature variance can be seen in Figure 4.2. We see that 8 features fall below this threshold and can be removed. As in the symmetric set, we observe that the range of accuracy is higher when the classifiers are generated from features in the high variance range. The maximum accuracy of a classifier generated

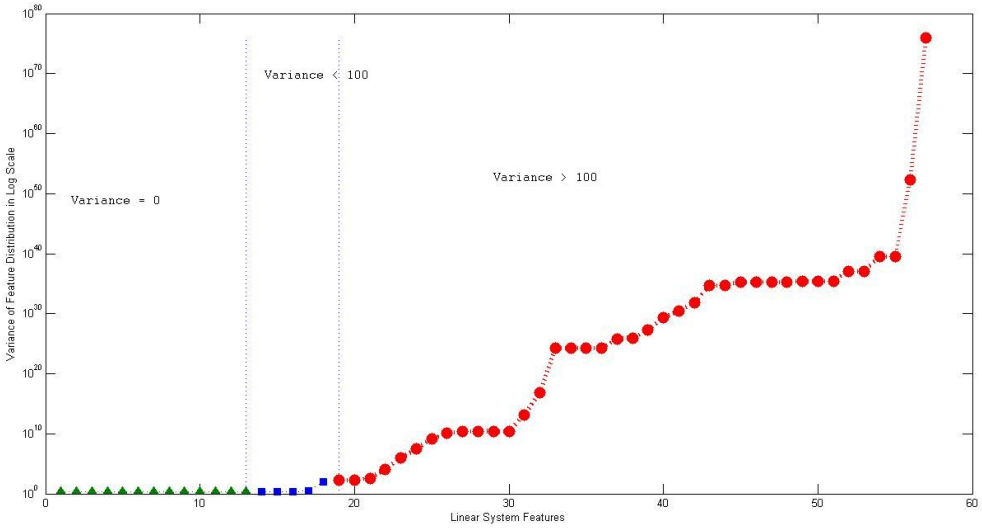

Fig. 4.1. Log Scale Variance, Symmetric

using SVM with one feature from the zero variance group is at most 36.73% from the zero variance group and as high as 73.29% from the high variance group. Due to the structure of the unsymmetric matrices, there are fewer redundant features. Here, all but one feature can be removed for redundant norms and spectral values.

4.5.2 Feature Set Ordering

4.5.2.1 Symmetric Matrix Data Set

We see in Figure 4.3 a comparison of the change in accuracy as the number of features supplied to the classifier is increased for the symmetric set. The last data point in the graph shows the accuracy of a classifier using the entire high variance set. We reach high accuracy in only a few steps for most classifiers and note that the accuracy does not continue increasing significantly regardless of how many more features are added. This shows us that we can maintain a high-accuracy result with fewer features.

We then compare the accuracy as features are added to each classifier for different orderings, *increasing*, *decreasing*, and random. Figure 4.4 compares orderings for the symmetric set. *Increasing order* is represented by blue, *decreasing order* is represented by red, and random order is represented by green. Section A shows results for Naive Bayes, section B shows results for SVM, section C shows results for IB1, section D shows results for IBk, section E shows results for Alternating Decision Tree, and section F shows results for Decision Stump. We observe that *increasing order* performs better for Naive Bayes and IB1, is roughly even for SVM, IBk, and ADT, and performs slightly worse for Decision Stump.

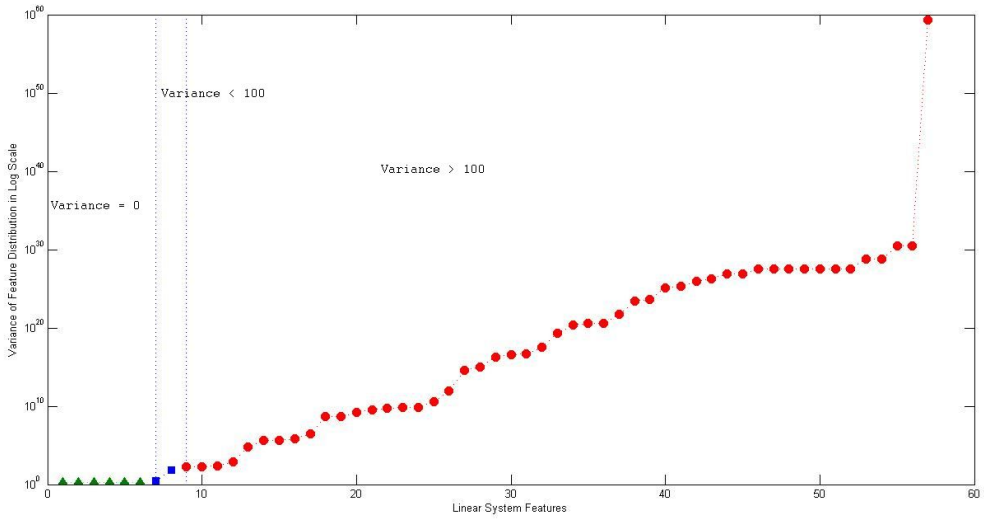

Fig. 4.2. Log Scale Variance, Unsymmetric

Anamod does not give the time for calculating each individual feature, so we obtain the timing results by computing the feature values in MATLAB. We compute the features from a sample of the original matrix set and project the results because MATLAB was unable to compute values for all the matrices due to their large size. Table 4.2 compares the time taken to compute the features we used with each classifier. The values indicate that the increasing order achieves an average speedup of 2468.33 (and a maximum of 3904.11) compared to the decreasing order, where speedup is defined as slower time to compute divided by faster time to compute. Note that in case of ADT and DS, though all 9 features were used in the increasing order method, the feature computation time is comparable to that of the decreasing order.

4.5.2.2 Unsymmetric Matrix Data Set

Figure 4.5 shows a comparison of the change in accuracy as the number of features supplied to the classifier is increased for the unsymmetric set. As in the symmetric set, the last data point in the graph shows the accuracy of a classifier using the entire high variance set. Again, we reach high accuracy in only a few steps for most classifiers and note that the accuracy does not continue increasing significantly regardless of how many more features are added. This shows us that we can maintain a high-accuracy result with fewer features for the unsymmetric set as well.

As in the symmetric set, we compare the accuracy as features are added to each classifier for different orderings, *increasing*, *decreasing*, and random. This is shown in Figure 4.6 with the same colorings and labels as the symmetric set. We observe that *increasing order* performs better for Naive Bayes, IBk, ADT, and Decision Stump, while

Table 4.2. Comparison of time to compute feature sets with increasing and decreasing orderings for the symmetric set

Method	Feature Set	Computing Time (in secs)	Feature Set	Computing Time (in secs)	Speedup
	Decreasing Order		Increasing Order		
NB	(condition number left-bandwidth)	30.90	(trace)	.0085	3635.29
NN(1)	(condition number left-bandwidth)	30.90	(trace, diagonal-variance)	.0085	3635.29
NN(10)	(condition number left-bandwidth)	30.90	(trace)	.0085	3635.29
SVM	(condition number left-bandwidth nnzeros, diagonal-dominance)	66.37	(trace, diagonal-) variance, diagonal-average, nrows)	0.017	3904.11
ADT	(condition number)	0.56	All 9	66.40	.0084
DS	(condition number)	0.56	All 9	66.40	.0084
Average Speedup					2468.33
Geometric Mean of Speedup					48.64

it is roughly even for SVM and IB1. These results show that *increasing order* maintains a high level of accuracy.

As for the symmetric set, we obtain timing results from MATLAB. Table 4.3 compares the time taken to compute the features we used with each classifier for the unsymmetric set. The values indicate that the increasing order achieves an average speedup of 2064 (and a maximum of 5784) compared to the decreasing order.

4.5.2.3 Summary

Based on our observations, we conclude that for most machine learning algorithms, our *increasing order* strategy for features achieves near-optimal classifier accuracy with low computation time. *Increasing order* proves to reach the highest accuracy or a comparable high accuracy with few features as opposed to other ordering that may reach higher accuracy but with additional features. Also, we see that Naive Bayes is able to obtain similar accuracies as SVM, while doing so in much less time.

Table 4.3. Comparison of time to compute feature sets with increasing and decreasing orderings for the unsymmetric set.

Method	Feature Set	Computing Time (in secs)	Feature Set	Computing Time (in secs)	Speedup
	Decreasing Order		Increasing Order		
NB	First 7	80.97	(trace)	.014	5783.6
IB1	First 7	80.97	(trace, diagonal-variance, diagonal-average, nrows,norm1, diagonal-dominance)	43.49	1.86
IBk	First 7	80.97	(trace, diagonal-variance, diagonal-average)	.0456	1775.66
SVM	First 7	80.97	(trace, diagonal-) variance, diagonal-average, nrows)	.048	1686.88
ADT	All 9	80.997	trace, diagonal-variance, diagonal-average	.0456	177.63
DS	All 9	80.997	trace, diagonal-variance	.0274	2956.09
Average Speedup					2063.62
Geometric Mean of Speedup					506.68

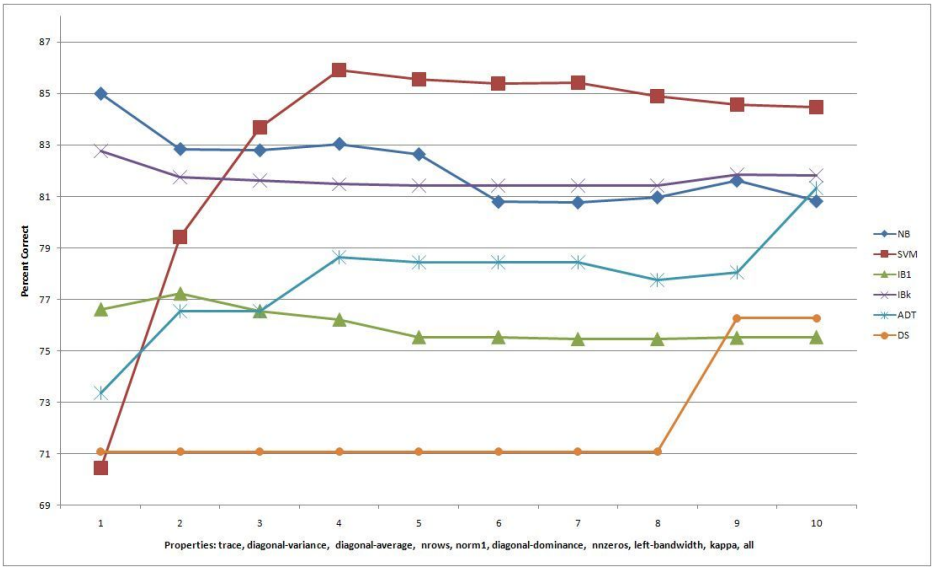

Fig. 4.3. Accuracy of Classifiers, Symmetric

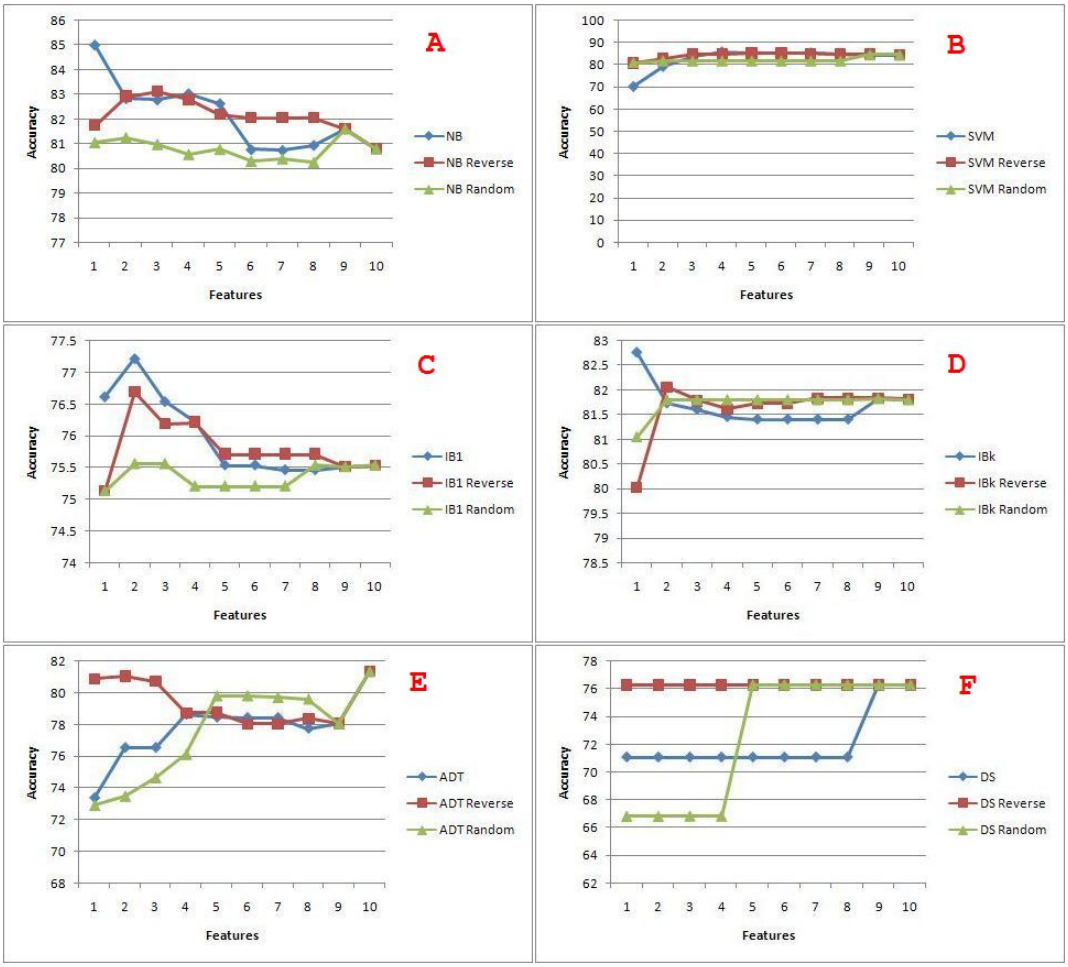

Fig. 4.4. Accuracy of Classifiers, Symmetric, Comparison of Ordering

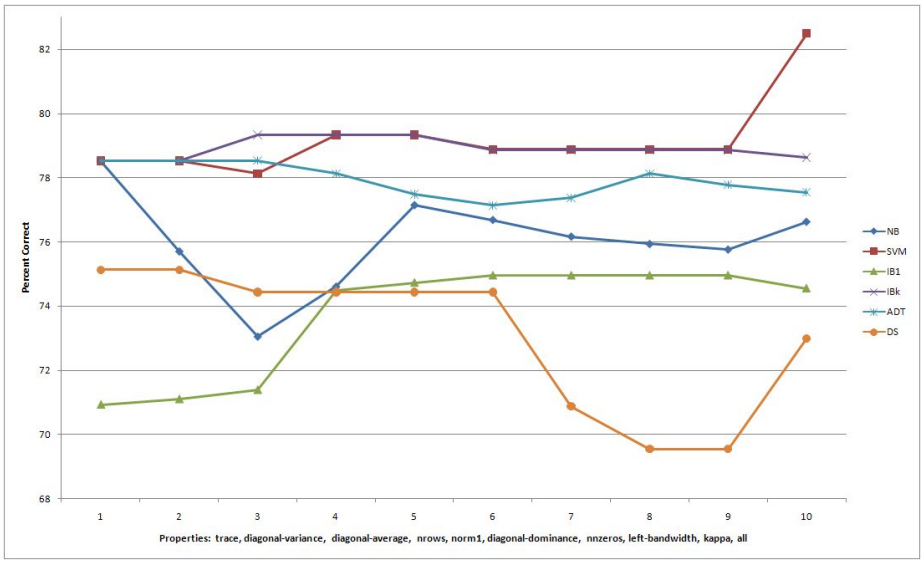

Fig. 4.5. Accuracy of Classifiers, Unsymmetric

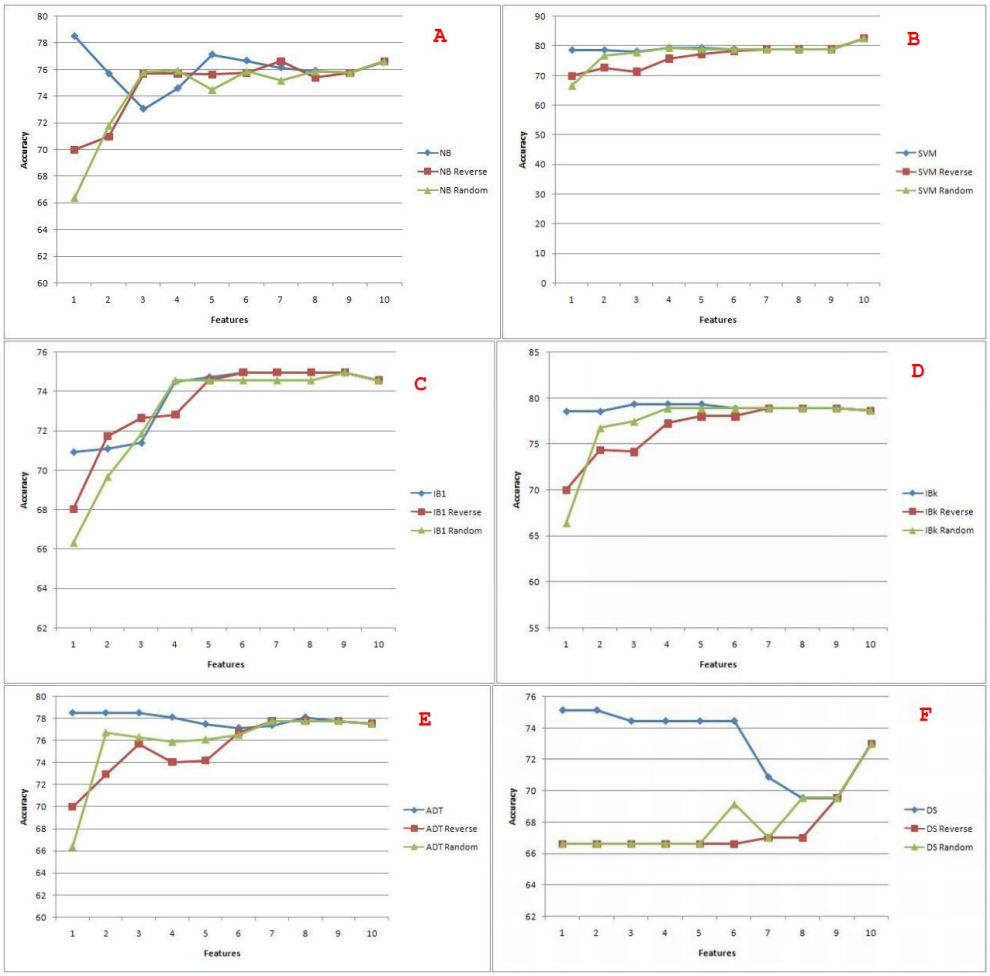

Fig. 4.6. Accuracy of Classifiers, Unsymmetric, Comparison of Ordering

Chapter 5

Conclusions

In this chapter we present conclusions about our method of low-cost, high-accuracy feature selection and ideas for future work.

We have designed and implemented a technique for producing low-cost, high-accuracy classifiers based on ordering features in increasing order of their computational complexity. Based on two datasets and the machine learning method used, efficient feature selection can be achieved.

We present near-optimal feature sets to demonstrate that our method achieves low-cost and high-accuracy. Our results can be extended to other datasets and are not limited to linear system sets. As part of our future research, we plan to study the cost and accuracy of our method, which uses variance, against a method where we use Fisher's Criterion [11] instead of variance. We also plan to study the use of genetic algorithms for feature selection. Additionally, we would like to further automate our system by using Weka's Java API.

Appendix A

Algorithms

Algorithm 2 Conjugate Gradient Method

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$.
for $i = 1, 2, \ldots$ **do**
  solve $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = {r^{(i-1)}}^T z^{(i-1)}$
  if $i = 1$ **then**
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  end if
  $q^{(i)} = Ap^{(i)}$
  $\alpha_i = \rho_{i-1}/{\rho^{(i)}}^T q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence; continue if necessary
end for

Algorithm 3 Conjugate Gradient Squared Method

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$.
Choose $\tilde{r}$ (for example, $\tilde{r} = r^{(0)}$)
for $i = 1, 2, \ldots$ **do**
  $\rho_{i-1} = \tilde{r}^T r^{i-1}$
  if $\rho_{i-1} = 0$ **method fails then**
  end if
  if $i = 1$ **then**
    $u^{(1)} = r^{(0)}$
    $p^{(1)} = u^{(1)}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $u^{(i)} = r^{(i-1)} + \beta_{i-1} q^{(i-1)}$
    $p^{(i)} = u^{(i)} + \beta_{i-1}(q^{(i-1)} + \beta_{i-1} p^{(i-1)})$
  end if
  solve $M\hat{p} = p^{(i)}$
  $\hat{v} = A\hat{p}$
  $\alpha_i = \rho_{i-1}/\tilde{r}^T \hat{v}$
  $q^{(i)} = u^{(i)} - \alpha_i \hat{v}$
  solve $M\hat{u} = u^{(i)} + q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i \hat{u}$
  $\hat{q} = A\hat{u}$
  $r^{(i)} = r^{(i-1)} - \alpha_i \hat{q}$
  check convergence; continue if necessary
end for

Algorithm 4 Stabilized Version of the BiConjugate Gradient Method

Compute $r_0 = b - Ax_0$, r_0^* arbitrary
$p_0 = r_0$
for $j = 0, 1, \ldots,$ until convergence **do**
  $\alpha_j = (r_j, r_0^*)/(Ap_j, r_0^*)$
  $s_j = r_j - \alpha_j Ap_j$
  $\omega_j = (As_j, s_j)/(As_j, s_j)$
  $x_{j+1} = x_j + \alpha_j p_j + \omega_j s_j$
  $r_{j+1} = s_j - \omega_j As_j$
  $\beta_j = \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \times \frac{\alpha_j}{\omega_j}$
  $p_{j+1} = r_{j+1} + \beta_j(p_j - \omega_j Ap_j)$
end for

Algorithm 5 Minimal Residual Iteration

Compute $r = b - Ax$ and $p = Ar$
for until convergence **do**
  $\alpha \leftarrow (p, r)/(p, p)$
  $x \leftarrow x + \alpha r$
  $r \leftarrow r - \alpha p$
  Compute $p = Ar$
end for

Algorithm 6 Generalized Minimal Residual Method

Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$, and $v_1 = r_0/\beta$
for $j = 1, 2, ..., m$ **do**
  Compute $w_j = Av_j$
  for $i = 1, \cdots, j$ **do**
    $h_{ij} = (w_j, v_i)$
    $w_j = w_j - h_{ij} v_i$
  end for
  $h_{j+1,j} = \left\| w_j \right\|_2$ If $h_{j+1,j} = 0$ set $m = j$ and go to define Hessenberg
  $v_{j+1} = w_j / h_{j+1,j}$
end for
Define $(m+1) \times m$ Hessenberg matrix $\overline{H}_m = \left\{ h_{ij} \right\}_{1 \le i \le m+1, 1 \le j \le m}$
Compute y_m, the minimizer of $\left\| \beta e_1 - \overline{H}_m y \right\|_2$, and $x_m = x_0 + V_m y_m$

Algorithm 7 Chebyshev Iteration

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$.
$d = (\lambda_{max} + \lambda_{min})/2$,$c = (\lambda_{max} - \lambda_{min})/2$
for $i = 1$, 2, ... **do**
  solve $Mz^{(i-1)} = r^{(i)}$
  if $i = 1$ **then**
    $p^{(1)} = z^{(0)}$
    $\alpha_1 = 2/d$
  else
    $\beta_{i-1} = \alpha_{i-1}(c/2)^2$
    $\alpha_i = 1/(d - \beta_{i-1}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1}p^{(i-1)}$
  end if
  $x^{(i)} = x^{(i-1)+\alpha_i p^{(i)}}$
  $r^{(i)} = b - Ax^{(i)} (= r^{(i-1)} - \alpha_i A p^{(i)})$
  check convergence; continue if necessary
end for

Algorithm 8 Richardson Method

input $n, (a_{i,j}), (b_i), (x_i), M$
for $k = 1$ to M **do**
  for $i = 1$ to n **do**
    $r_i \leftarrow b_i - \sum_{j=1}^{n} a_{ij} x_j$
  end for
  for $i = 1$ to n **do**
    $x_i \leftarrow x_i + r_i$
  end for
end for
output $k, (x_i), (r_i)$

Algorithm 9 Transpose-Free Quasi-Minimal Residual Method

Compute $w_0 = u_0 = r_0 = b - Ax_0$, $v_0 - Au_0$, $d_0 = 0$
$\tau_0 = \|r_0\|_2$, $\theta_0 = \eta_0 = 0$
Choose r_0^* such that $\rho_0 \equiv (r_0^0, r_0) \neq 0$
for $m = 0$, 1, ..., until convergence **do**
  if m is even **then**
    $\alpha_{m+1} = \alpha_m = \rho_m/(v_m, r_0^*)$
    $u_{m+1} = u_m - \alpha_m v_m$
  end if
  $w_{m+1} = w_m - \alpha_m A u_m$
  $d_{m+1} = u_m + (\theta_m^*/\alpha_m)\eta_m d_m$
  $\theta_{m+1} = \|w_{m+1}\|_m / \tau_m$, $c_{m+1} = (1 + \theta_{m+1}^2)^{-1/2}$
  $\tau_{m+1} = \tau_m \theta_{m+1} c_{m+1}$, $\eta_{m+1} = c_m^2 \alpha_m$
  $x_{m+1} = x_m + \eta_{m+1} d_{m+1}$
  if m is odd **then**
    $p_{m+1} = (w_{m+1}, r_0^*)$,$\beta_{m-1} = \rho_{m+1}/\rho_{m-1}$
    $u_{m+1} = w_{m+1} + \beta_{m-1} u_m$
    $v_{m+1} = Au_{m+1} + \beta_{m-1}(Au_m + \beta_{m-1} v_{m-1})$
  end if
end for

Appendix B

Scripts

Listing B.1. Perl script to parse Anamod output

```
1  #!/usr/bin/perl -w
2  #use strict;
3
4  #path to files and result file name
5  $dir = "C:\\thesis\\SparseMatrices\\unsymmetric\\anamod_output\\";
6  $out = "features.csv";
7
8  open(MYOUT, "> $out") or die "cannot open $out : $!";
9  opendir(OUT, $dir) or die "cannot opendir $dir: $!";
10
11 my $counter = 0;
12 my $first_file_columns;
13
14 while(defined($file = readdir(OUT))) {
15     if ($file =~ m/.out$/) {
16         print "$file\n" if -T "$dir/$file";
```

```
17      open(CURRENT, "< $file") or die "cannot open $file : $!";
18          my @col;
19          my @val;
20          while (my $line = <CURRENT>) {
21              if ( $line =~ /<(\S+)>.*<(\S+)>/ ) {
22                  push @col, $1;
23                  push @val, $2;
24              }
25              elsif( $line =~ /<(\S+)>/ ) {
26                  push @col, $1;
27                  push @val, "0.0";
28              }
29          }
30          #close FH;
31          my $columns = join ",", @col;
32          if ($counter == 0) {
33              $first_file_columns = $columns;
34              print MYOUT $first_file_columns,"\n";
35          }
36          die "$file apparently does not have equivalent data"
37              if ($columns ne $first_file_columns);
38          print MYOUT join(",", @val),"\n";
```

```
39              $counter++;
40          close(CURRENT);
41      }
42  }
43
44  warn "$counter file(s) slurped";
45
46  closedir(OUT);
47
48  close(MYOUT);
```

Listing B.2. Perl script to parse PETSc output

```
1  #!/usr/bin/perl -w
2  #use strict;
3  use Time::localtime;
4
5  $dir = "C:\\thesis\\SparseMatrices\\unsymmetric\\petsc_output\\";
6  $out = "parsed.log";
7  $tm = localtime;
8
9  open(MYOUT, "> $out") or die "cannot open $out : $!";
10 print MYOUT "Timestamp: ";
```

```
11 print MYOUT $tm->mon+1;
12 print MYOUT "-";
13 print MYOUT $tm->mday;
14 print MYOUT "-";
15 print MYOUT $tm->year+1900;
16 print MYOUT "\n\n";
17
18 print MYOUT "KSP\tMAT\tPC_TYPE\tMAT_SIZE\tNONZERO\tNUM_ITER\tRES_NORM\tSOL_T
19 opendir(OUT, $dir) or die "cannot opendir $dir: $!";
20
21 while(defined($file = readdir(OUT))) {
22    if ($file =~ m/.out$/) {
23       print "$file\n" if -T "$dir/$file";
24       open(CURRENT, "< $file") or die "cannot open $file : $!";
25       $findunder = index($file,"_");
26       $length = length($file);
27       $ksp = substr($file,0,$findunder);
28       print MYOUT $ksp;
29       print MYOUT "\t";
30       $findunderX = index($file,"_",$findunder+1);
31       $mat = substr($file,$findunderX+1,-4);
32       print MYOUT $mat;
```

```
33          print MYOUT "\t";
34          $undercalc = $findunderX - $findunder;
35          $pc_type = substr($file,$findunder+1,$undercalc-1);
36          print MYOUT $pc_type;
37          print MYOUT "\t";
38
39          $count = 0;
40          $gotsize = 0;
41          $gotnonzeros = 0;
42
43          while (<CURRENT>) {
44              $line = $_;
45              #last if $. == 24;
46              if ($line=~/^PC Object/) {
47                  $count = $count +1;
48              }
49              if (($count == 1)&&($line=~/rows=/)&&($gotsize==0)) {
50                  $line=~/rows=(.+),/;
51                  $gotsize = 1;
52                  print MYOUT $1;
53                  print MYOUT "\t";
54              }
```

```
55          if ($gotsize == 1) {
56             if (($line=~/nonzeros=/)&&($gotnonzeros==0)) {
57                $gotnonzeros = 1;
58                $line=~/nonzeros=(.+),/;
59                print MYOUT $1;
60                print MYOUT "\t";
61             }
62             if ($line=~/Number of iterations =/) {
63                $line=~/iterations = (.+)\s/;
64                print MYOUT $1;
65                print MYOUT "\t";
66             }
67             if ($line=~/Residual norm/) {
68                $line=~/norm (.+)\s/;
69                print MYOUT $1;
70                print MYOUT "\t";
71             }
72          }
73          if ($line=~/ 3:          KSPSolve:/) {
74             $sol_t_avg = substr($line,20,11);
75             print MYOUT $sol_t_avg;
76             print MYOUT "\t";
```

```
77                $sol_flop_avg = substr($line,40,11);
78                print MYOUT $sol_flop_avg;
79                print MYOUT "\t";
80            }
81            if ($line=~/PCSetUp/) {
82                $pc_time_max = substr($line,28,11);
83                print MYOUT $pc_time_max;
84                print MYOUT "\t";
85                $pc_flops_max = substr($line,44,8);
86                print MYOUT $pc_flops_max;
87            }
88        }
89
90        $count = 0;
91        $gotsize = 0;
92        $gotnonzeros = 0;
93
94        #print MYOUT $line;
95        print MYOUT "\n";
96
97        close(CURRENT);
98    }
```

```
99  }
100
101 closedir(OUT);
102
103 close(MYOUT);
```

Listing B.3. Perl script to merge Anamod and PETSc output

```
1  #!/usr/bin/perl -w
2  #use strict;
3
4  #$dir = "D:\\thesis\\SparseMatrices\\unsymmetric\\anamod_petsc\\";
5  $out = "merged_unsymmetric.csv";
6  $properties = "features_withname.csv";
7  $results = "parsed_cut.csv";
8
9  open(MYOUT, "> $out") or die "cannot open $out : $!";
10 open(PROP, $properties) or die "cannot open $properties : $!";
11 open(RES, $results) or die "cannot open $results : $!";
12
13 my %data;
14 while(<PROP>) {
15     my ($name, @vals) = split(/,/);
```

```
16      $data{$name} = \@vals;
17      #print "$name\n";
18  }
19  print "size of data (properties) hash:  " . keys(%data) . "\n";
20
21  while(<RES>) {
22      my @resval = split(/,/);
23      chomp(@resval);
24      my $stemp = "mat" . $resval[1] . "_" . $resval[11] . $resval[0];
25      $resval[0] = $stemp;
26      #print "$stemp\n";
27      if (exists($data{$stemp})) {
28          #print @resval;
29          my @atemp = @{$data{$stemp}};
30          #print @atemp;
31          push (@resval,@atemp);
32          #print "@resval\n";
33          print MYOUT join(",",@resval);
34      } else {
35          #Cut out _0 and _10 and _11 so, some will not exist
36          print "Key does not exist!\n";
37      }
```

```
38 }
39
40 close(MYOUT);
41 close(PROP);
42 close(RES);
```

Appendix C

Matrices and Features

Table C.1: Set of Symmetric Matrices

Matrix	**Rank**	**Non-zeroes**	**Description**
msc00726	726	34518	Symmetric Test Matrix
msc01050	1050	26198	Symmetric Test Matrix
msc01440	1440	44998	Symmetric Test Matrix
nasa2146	2146	72250	Structure from NASA Langley 2146 Degrees of Freedom
nasa2910	2910	174296	Structure from NASA Langley 2910 Degrees of Freedom
bcsstk15	3948	117816	Module of an Offshore Platform
msc04515	4515	97707	Symmetric Test Matrix
crystk01	4875	315891	FEM Crystal Free Vibration Stiffness Matrix
bcsstk16	4884	290378	Corp of Engineers Dam
s2rmq4m1	5489	263351	FEM Cylindrical Sheel 30X30
s3rmq4m1	5489	262943	FEM Cylindrical Shell 30X30

Continued on Next Page...

Table C.1 – Continued

Matrix	**Rank**	**Non-zeroes**	**Description**
s3rmt3m1	5489	217669	FEM Cylindrical Shell 30X30
nd3k	9000	3279690	ND Problem Set
msc10848	10848	1229776	Symmetric Test Matrix
t2dah_a	11445	176117	Micropyros Thruster
bcsstk18	11948	149090	Structural Stiffness Matrix
vibrobox	12328	301700	Vibroacoustic problem (flexible box, structure only)
crystm02	13965	322905	FEM Crystal Free Vibration Mass Matrix
dis120	14400	184204	Symmetric Matrix from Visible Surface Computation
bodyy4	17546	121550	From NASA, collected by Alex Pothen
bodyy5	18589	128853	From NASA, collected by Alex Pothen
bodyy6	19366	76787	From NASA, collected by Alex Pothen
t3dl_a	20360	509866	Micropyros Thruster
bcsstk36	23052	1143140	Stiffness Matrix, Automobile Shock Absorber Assembly
msc23052	23052	1142686	Symmetric Test Matrix
smt	25710	3753184	Surface Mounted Transistor
wathen100	30401	471601	Matrix from Andy Wathen, Oxford Univ.

Continued on Next Page...

Table C.1 – Continued

Matrix	**Rank**	**Non-zeroes**	**Description**
fdm2	32010	159466	Discrete Kohn-Sham Equation
jnlbrng1	40000	199200	Quadratic Journal Bearing Problem
torsion1	40000	197608	Optimization Problem
minsurfo	40806	203622	Minimum Surface Problem
c-62	41731	559341	Nonlinear Optimization
c-66	49989	499007	Nonlinear Optimization
nasasrb	54870	54870	Structure of Shuttle Rocket Booster
qa8fk	66127	1660579	3D Acoustic FE Stiffness Matrix
qa8fm	66127	1660579	3D Acoustic FE Mass Matrix
finan512	74752	596992	Portfolio Optimization, 512 Scenarios
s3dkq4m2	90449	4427725	FEM Cylindrical Shell 150X100
s3dkt3m2	90449	3686223	FEM Cylindrical Shell 150X100
m_t1	97578	9753570	Tubular Joint
x104	108384	8713602	SESAM Test Example
shipsec8	114919	3303553	Ship Section, Detail from Production Run
ship_003	121728	3777036	Ship Structure
cfd2	123440	3087898	Symmetric Pressure Matrix
augustus5	134133	1155912	Diffusion equation from 3D mesh
engine	143571	4706073	Linear Tetrahedral Elements

Continued on Next Page...

Table C.1 – Continued

Matrix	**Rank**	**Non-zeroes**	**Description**
pwtk	217918	11524432	Pressurized Wind Tunnel Stiffness Matrix
Lin	256000	1766400	Large Sparse Eigenvalue Problem
af_shell4	504855	17562051	Sheet Metal Forming Simulation
augustus7	1060864	9313876	Diffusion equation from 3D mesh

Table C.2: Set of Unsymmetric Matrices

Matrix	**Rank**	**Non-zeroes**	**Description**
mbeacxc	496	49920	Unsymmetric Matrix U.S. Economy 1972
raefsky6	3402	130371	Slosh Tank Model
heart1	3557	1385317	Quasi-static FEM of a heart
tols4000	4000	8784	Tolosa Matrix
Chebyshev3	4101	36879	Integration matrix, Chebyshev method, 4th order semilinear initial BVP
rdist1	4134	94408	Chemical process separation
viscoplastic1	4326	61166	FEM discretization of a viscoplastic collision problem
cavity21	4562	131735	Driven Cavity 20 x 20, Reynolds number: 500

Continued on Next Page...

Table C.2 – Continued

Matrix	**Rank**	**Non-zeroes**	**Description**
cavity25	4562	131735	Driven Cavity 20 x 20, Reynolds number: 900
cavity26	4562	138040	Driven Cavity 20 x 20, Reynolds number: 1000
gemat11	4929	33108	Unsymmetric Matrix from G.E. OPF Program
gemat12	4929	33044	Unsymmetric Matrix from G.E. OPF Program
rw5151	5151	20199	Markov Chain Modeling, Random Walk ($M = 100$)
g7jac020	5850	42568	Jacobian from CEPII's 'G7marmotte' OLG model, oldstack 020
g7jac020sc	5850	42568	Jacobian from CEPII's 'G7marmotte' OLG model, oldstack 020 (scaled)
utm5940	5940	83842	Uedge Test Matrix
raefsky	6316	167178	Landing Hydrofoil Airplane FSE Model
cell1	7055	30082	Telecom Italia Mobile, GSM cell phone problem

Continued on Next Page...

Table C.2 – Continued

Matrix	**Rank**	**Non-zeroes**	**Description**
Pd	8081	13036	Matrix problem X=A/B with many sparse right-hand-sides
cryg10000	10000	49699	Crystal Growth Eigenmodes
fd15	11532	44206	Single-material crack problem, finite diff log grid
sinc15	11532	551184	Single-material crack problem, sinc basis log grid
poisson3Da	13514	352762	Comsol, Inc. www.femlab.com : 3D Poisson problem
airfoil_2d	14214	259688	Unstructured 2D mesh (airfoil)
lhr14	14270	305750	Light hydrocarbon recovery
lhr14c	14270	307858	Light hydrocarbon recovery, corrected version of LHR14
epb1	14734	95053	Plate-fin heat exchanger (simple case)
fd18	16428	63406	Single-material crack problem, finite diff log grid
sinc18	16428	948696	Single-material crack problem, sinc basis log grid

Continued on Next Page...

Table C.2 – Continued

Matrix	**Rank**	**Non-zeroes**	**Description**
lhr17c	17576	381975	Light hydrocarbon recovery, corrected version of LHR17
ns3Da	20414	1679599	Comsol, Inc. www.femlab.com : 3D Navier Stokes
rim	22560	1014951	FEM, fluid mechanics problem
Zd_Jac3	22835	1915726	Chemical process simulation
Zd_Jac6	22835	1711557	Chemical process simulation
af23560	23560	460598	NACA 0012 Airfoil $M = 0.8$, Eigenvalue Calculation
epb2	25228	175027	Plate-fin heat exchanger w/flow redistrib
viscoplastic2	32769	381326	FEM discretization of a viscoplastic collision problem
Zhao2	33861	166453	Electromagnetic matrix from A. C. Cangellaris and Li Zhao
lhr34c	35152	764014	Light hydrocarbon recovery, corrected version of LHR34
chem_master1	40401	201201	chemical master eqn, $aij*h = probofi- > j$ transition in time h (Markov model)

Continued on Next Page...

Table C.2 – Continued

Matrix	**Rank**	**Non-zeroes**	**Description**
jan99jac120	41374	229385	Jacobian from Bank of Canada 'jan99' model, oldstack 120
jan99jac120sc	41374	229385	Jacobian from Bank of Canada 'jan99' model, oldstack 120, with scaling
g7jac180sc	53370	641290	Jacobian from CEPII's 'G7marmotte' OLG model, oldstack 180 (scaled)
g7jac180	53370	641290	Jacobian from CEPII's 'G7marmotte' OLG model, oldstack 180
poisson3Db	85623	2374949	Comsol, Inc. www.femlab.com : 3D Poisson problem
lung2	109460	492564	Coupled temp,water vapour transport in lung
Baumann	112211	748331	chemical master eqn, $aij*h = probofi- > j$ transition in time h (Markov model)
stomach	213360	3021648	3D electro-physical model of a duodenum
language	399130	1216334	finite-state machine for natural language processing
largebasis	440020	5240084	Large basis matrix from Q. Li, Univ. Wisconsin

Continued on Next Page...

Table C.2 – Continued

Matrix	**Rank**	**Non-zeroes**	**Description**
Hamrle3	1447360	5514242	circuit simulation matrix

Table C.3: Partial set of features for matrix A generated using Anamod

Feature Name	**Description**	**Complexity**
trace	$\sum_{i=1}^{n}(a(i,i))$	O(n)
trace-abs	$\sum_{i=1}^{n}(abs(a(i,i)))$	O(n)
n-dummy-rows	number of rows with only one element	O(n)
dummy-rows-kind	For all dummy rows,D, 0 if $D(i,i) = 1$;	
	1 if $D(i,i) \neq 1$ and $D(i,i) \neq 0$; 2 if $D(i,i) = 0$	O(n)
diag-zerostart	$\min(i)$, $\forall_{j>i} a(j,j) = 0$	O(n)
diag-definite	1 if $a(i,i) > 0 \forall_i$, 0 otherwise	O(n)
diagonal-average	$1/n \sum_{i}^{n}(abs(a(i,i))$	O(n)
diagonal-variance	variance of the diagonal average	O(n)

Continued on Next Page...

Table C.3 – Continued

Feature Name	**Description**	**Complexity**
diagonal-sign	-10 all zero, -2 all negative, -1 nonpositive, 0 indefinite, 1 nonnegative, 2 all positive	O(n)
nrows	number of rows in the matrix	O(n)
trace-asquared (TrAsq(A))	$\sum_{i=1}^{n}\sum_{j=1}^{n}(a(i,j)*a(j,i))$	O(nnz)
commutator-normF	$\sqrt{TrAsq(AA^T - A^TA)}$	O(nnz)
norm1	$max_i \sum_{j=1}^{n}(a(i,j))$	O(nnz)
normInf	$max_j \sum_{i=1}^{n}(a(i,j))$	O(nnz)
normF	$\sqrt{TrAsq(A)}$	O(nnz)
diagonal-dominance	$min_i(a(i,i) - \sum_{j=1}^{n}(a(i,j)))$	O(nnz)
symmetry-snorm	$max_j \sum_{i=1}^{n}(a(i,j))$, if $a(i,j) = a(j,i)$	O(nnz)
symmetry-anorm	$max_j \sum_{i=1}^{n}(a(i,j))$, if $a(i,j) \neq a(j,i)$	O(nnz)
symmetry-fsnorm	$\sqrt{TrAsq(\hat{A})}$, where $\hat{A}$ is the symmetric part of A	O(nnz)
symmetry-fanorm	$\sqrt{TrAsq(\hat{A})}$, where $\hat{A}$ is the anti-symmetric part of A	O(nnz)

Continued on Next Page...

Table C.3 – Continued

Feature Name	**Description**	**Complexity**
nnzeros	number of nonzero elements in the matrix	O(nnz)
max-nnzeros-per-row	maximum number of nonzeros per row	O(nnz)
min-nnzeros-per-row	minimum number of nonzeros per row	O(nnz)
left-bandwidth	$\max(i-j)$, for all $a(i,j) \neq 0$	O(nnz)
right-bandwidth	$\max(j-i)$, for all $a(i,j) \neq 0$	O(nnz)
row-variability	$\max_i \log_{10} \frac{\max_j \lvert a(i,j)\rvert}{\min_j \lvert a(i,j)\rvert}$	O(nnz)
col-variability	$\max_j \log_{10} \frac{\max_i \lvert a(i,j)\rvert}{\min_i \lvert a(i,j)\rvert}$	O(nnz)
n-ritz-values	number of Ritz values	O(ls)
kappa	estimated condition number	O(ls)
positive-fraction	fraction of computed eigenvalues that has positive real part	O(ls)
sigma-max	maximum singular value	O(ls)
sigma-min	minimum singular value	O(ls)
lambda-max-by-magnitude-real	real part of maximum eigenvalue by its magnitude	O(ls)

Continued on Next Page...

Table C.3 – Continued

Feature Name	**Description**	**Complexity**
lambda-max-by-magnitude-img	imaginary part of maximum eigenvalue by its magnitude	O(ls)
lambda-min-by-magnitude-real	real part of minimum eigenvalue by its magnitude	O(ls)
lambda-min-by-magnitude-img	imaginary part of minimum eigenvalue by its magnitude	O(ls)

Appendix D

Solver Performance

Table D.1: Performance of Symmetric Matrices

Matrix	Fastest Solver	Slowest Solver	Failed	Median	Mean
af_shell4	Richardson, None	GMRES60, ILU0	0	2591.1	2925.024
augustus5	CGS, None	GMRES60, ILU2	0	300.24	389.7124
augustus7	CGS, None	GMRES60, ILU1	3	2914.7	3897.105
bccstk16	Richardson, None	Chebychev, ILU3	0	0.42625	6.010322
bcsstk15	Richardson, None	Richardson, ILU3	0	5.6992	13.36755
bcsstk18	Richardson, None	GMRES5, ILU2	0	4.9282	12.5808
bcsstk36	Richardson, None	BCGS, ILU3	0	213.65	236.4891
bodyy4	Richardson, None	Chebychev, ILU3	0	2.4911	10.16956
bodyy5	CG, ILU1	GMRES5, Jacobi	8	1.2306	4.286745
bodyy6	Richardson, None	GMRES60, None	1	2.31975	9.805185
c-62	MINRES, Jacobi	Richardson, SOR	0	80.596	358.7854
c-66	CG, Jacobi	Richardson, SOR	0	94.2745	802.099
cfd2	Richardson, ICC	GMRES60, ILU2	1	593.565	730
crystk01	CGS, None	TFQMR, ILU3	0	42.7835	60.13575

Continued on Next Page...

Table D.1 – Continued

Matrix	Fastest Solver	Slowest Solver	Failed	Median	Mean
crystm02	CG, ILU0	Chebychev, ILU3	0	6.30135	28.88639
dis120	MINRES, ILU	GMRES60, SOR	1	2.1921	14.49787
engine	Richardson, None	GMRES5, ILU2	0	717.145	847.9936
fdm2	Richardson, None	GMRES60, Jacobi	0	14.685	24.53284
finan512	Richardson, None	Chebychev, ILU2	0	1.76055	11.95854
jnlbrng1	Richardson, None	Chebychev, ILU3	0	2.1702	9.29861
Lin	Richardson, None	GMRES60, ILU3	0	591.57	822.1142
m_t1	Richardson, None	CGS, ILU2	1	1316.6	2015.831
minsurfo	Richardson, None	TFQMR, None	0	1.15175	3.844826
msc00726	Richardson, None	Chebychev, ILU3	0	0.61061	2.793687
msc01050	Richardson, None	TFQMR, ILU3	0	4.5358	5.167892
msc01440	Richardson, None	GMRES5, ILU2	0	0.64974	1.610075
msc04515	CG, ICC	GMRES60, SOR	0	0.36866	3.977201
msc10848	Richardson, None	TFQMR, ILU0	0	83.538	146.7675
msc23052	Richardson, None	BCGS, ILU1	0	130.145	177.368
nasa2146	Richardson, None	Richardson, SOR	0	0.181275	1.356385
nasa2910	Richardson, None	TFQMR, ICC	0	4.2959	15.35672
nasasrb	Richardson, None	BCGS, ILU0	0	330.71	390.5288
nd3k	CGS, None	Richardson, ILU3	0	425.15	763.6123
pwtk	Richardson, None	BCGS, ILU0	1	1707.85	1932.057

Continued on Next Page...

Table D.1 – Continued

Matrix	Fastest Solver	Slowest Solver	Failed	Median	Mean
qa8fk	CGS, None	Richardson, ILU3	0	90.316	450.8415
qa8fm	Richardson, Jacobi	Chebychev, ILU2	0	3.0629	23.62002
s2rmq4m1	Richardson, None	GMRES5, ILU3	0	22.074	31.14555
s3dkq4m2	Richardson, None	BCGS, ILU0	0	742.68	832.2134
s3dkt3m2	Richardson, None	TFQMR, ILU3	0	1064.5	1022.834
s3rmq4m1	Richardson, None	GMRES5, ILU3	0	11.175	20.75854
s3rmt3m1	Richardson, None	GMRES30, ICC	0	13.354	20.92178
ship_003	Richardson, None	GMRES5, ILU3	24	1218.35	1435.381
shipsec8	Richardson, None	BCGS, ILU0	0	569.81	846.7974
smt	Richardson, None	BCGS, ILU1	1	405.49	538.7713
t2dah_a	MINRES, Jacobi	TFQMR, ILU3	0	49.3475	54.88617
t3dl_a	MINRES, Jacobi	GMRES5, ILU3	0	84.3625	168.9461
torsion1	GMRES5, None	Chebychev, ILU3	0	1.28645	7.134025
vibrobox	Richardson, None	TFQMR, ILU0	20	61.251	62.75769
wathen100	Richardson, None	Chebychev, ILU3	0	1.3121	15.96707
x104	Richardson, None	BCGS, ILU0	0	1011	1605.345

Table D.2: Performance of Unsymmetric Matrices

Matrix	**Fastest Solver**	**Slowest Solver**	**Failed**	**Median**	**Mean**
af23560	Richardson, None	Richardson, SOR	0	8.535	167.5954
airfoil_2d	Richardson, None	TFQMR, ICC	0	5.595	21.18619
Baumann	Richardson, None	Richardson, SOR	0	456	656.3483
cavity21	CGS, Jacobi	Richardson, ILU1	21	1.5	10.09195
cavity25	Richardson, None	Richardson, ILU1	21	2.04	61.44935
cavity26	Richardson, None	Richardson, SOR	14	12.6	14.01055
cell1	Richardson, None	BCGS, ILU3	0	15.65	19.56483
Chebyshev3	Richardson, None	Richardson, SOR	7	7.64	37.04531
chem_master1	Richardson, None	GMRES60, Jacobi	0	17.5	32.95705
cryg10000	Richardson, None	Richardson, SOR	0	24.35	25.62086
epb1	Richardson, ICC	GMRES60, None	0	5.23	8.561382
epb2	Richardson, ICC	GMRES60, None	0	2.255	54.53222
fd15	Richardson, None	Richardson, SOR	35	16.7	13.01909
fd18	Richardson, None	Richardson, SOR	35	23.4	135.1945
g7jac020	Richardson, None	GMRES60, Jacobi	42	10.5	116.4952
g7jac020sc	Richardson, Jacobi	GMRES60, None	42	10.85	7.15785
g7jac180	Richardson, None	GMRES60, Jacobi	42	168	6.0466
g7jac180sc	Richardson, Jacobi	GMRES60, Jacobi	42	169.5	4.562857
gemat11	Richardson, None	GMRES60, Jacobi	42	9.585	93.08357

Continued on Next Page...

Table D.2 – Continued

Matrix	Fastest Solver	Slowest Solver	Failed	Median	Mean
gemat12	Richardson, None	GMRES60, Jacobi	42	7.37	384.3709
heart1	Richardson, None	TFQMR, ILU0	7	6.83	71.70275
jan99jac120	Richardson, None	GMRES60, Jacobi	42	89.75	14.36143
jan99jac120sc	Richardson, ICC	GMRES60, ICC	35	114	126.8614
language	CGS, None	GMRES5, Jacobi	7	18	539.9158
largebasis	CGS, None	GMRES60, Jacobi	42	1830	401.3576
lhr14	GMRES5, Jacobi	TFQMR, Jacobi	42	22.639	41.81871
lhr14c	Richardson, Jacobi	Richardson, SOR	38	1.1865	588.8332
lhr17c	Richardson, Jacobi	Richardson, SOR	38	1.735	1.039444
lhr34c	Richardson, Jacobi	Richardson, SOR	38	3.895	104.9939
lung2	Richardson, None	Richardson, SOR	0	1.095	284.6977
mbeacxc	Richardson, Jacobi	Richardson, SOR	35	1.04	1728.094
ns3Da	Richardson, Jacobi	Richardson, SOR	14	200.5	2.948024
Pd	CGS, None	GMRES30, Jacobi	7	0.474	20.20347
poisson3Da	Richardson, ICC	GMRES30, ICC	0	14.1	61.31937
poisson3Db	BCGS, ILU0	BCGS, ILU2	53	122	41
raefsky5	Richardson, None	TFQMR, ICC	0	0.129	7.969214
raefsky6	Richardson, None	TFQMR, ICC	0	0.123	137.6808
rdist1	Richardson, None	Richardson, SOR	35	1.47	1154.686
rim	Richardson, None	Richardson, SOR	14	274	202.3635

Continued on Next Page...

Table D.2 – Continued

Matrix	**Fastest Solver**	**Slowest Solver**	**Failed**	**Median**	**Mean**
rw5151	CGS, SOR	Richardson, SOR	35	1.5	662.8062
sinc15	Richardson, None	Richardson, SOR	35	70.7	16.51905
sinc18	Richardson, None	Richardson, SOR	35	153	673.8143
stomach	Richardson, ICC	Richardson, SOR	0	21.15	884.8409
tols4000	Richardson, None	Richardson, SOR	35	2.43	23.10952
utm5940	Richardson, ICC	Richardson, SOR	0	6.585	28.83815
viscoplastic1	Richardson, Jacobi	Richardson, SOR	0	3.505	191.1783
viscoplastic2	Richardson, Jacobi	Richardson, SOR	21	80.3	1440.389
Zd_Jac3	Richardson, None	Richardson, SOR	35	33.6	28.77619
Zd_Jac6	Richardson, None	Richardson, SOR	35	158	150.9333
Zhao2	Richardson, Jacobi	Richardson, SOR	0	83.4	142.9386

References

[1] Aaron Arvey. Introduction to Boosting and Alternating Decision Trees. `http://jboost.sourceforge.net/doc.html`.

[2] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, 2000.

[3] Satish Balay, Kris Buschelman, Victor Eijkhout, William Gropp, Dinesh Kaushik, Matt Knepley, Lois Curfman McInnes, Barry Smith, and Hong Zhang. PETSc Users Manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2003.

[4] Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June M. Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, 1994.

[5] Samy Bengio. Statistical Machine Learning from Data, 2006. IDIAP Research Institute and EPFL.

[6] C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[7] Kyung K. Choi and Nam H. Kim. *Structural Sensitivity Analysis and Optimization: Linear systems*. Springer, 2005.

[8] Belur V. Dasarathy. *Nearest Neighbor: Pattern Classification Techniques.* IEEE Computer Society, 1990.

[9] Timothy A. Davis. The University of Florida Sparse Matrix Collection, 2007.

[10] J. Dongarra and V. Eijkhout. Self adapting numerical algorithm for next generation applications. *International Journal of High Performance Computing Applications*, 17(2):125–132, 2003.

[11] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification, Second Edition.* Wiley-Interscience, 2000.

[12] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford, 1986.

[13] V. Eijkhout and E. Fuentes. A Proposed Standard for Numerical Metadata. *submitted to ACM Trans. Math. Software.*, 2006.

[14] V. Eijkhout and E. Fuentes. *Anamod Online Documentation*, 2007. `http://www.tacc.utexas.edu/~eijkhout/doc/anamod/html/`.

[15] Erika L. Fuentes. *Statistical and Machine Learning Techniques Applied to Algorithm Selection for Solving Sparse Linear Systems.* PhD thesis, University of Tennessee, 2007.

[16] V.Y. Glizer. Observability of Singularly Perturbed Linear Time-Dependent Differential Systems with Small Delay. *Journal of Dynamical and Control Systems*, pages 329–363, July 2004.

[17] Gene H. Golub and Charles F. Van Loan. *Matrix Computations, Third Edition*. The Johns Hopkins University Press, 1996.

[18] Isabelle Guyon and Andre Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning 3*, 2003.

[19] Michael T. Heath. *Scientific Computing, An Introductory Survey, Second Edition*. McGraw-Hill Higher Education, 2002.

[20] Thomas Hill and Paul Lewicki. *Electronic Statistics Textbook*. Statsoft, Inc., 2007.

[21] RN Hota and ML Munjal. A New Hybrid Approach for the Thermo-Acoustic Modelling of Engine Exhaust Systems. *International Journal of Acoustics and Vibration*, pages 129–138, 2004.

[22] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. *International Conference on Machine Learning*, pages 121–129, 1994.

[23] David Kinkaid and Ward Cheney. *Numerical Analysis: Mathematics of Scientific Computing, Third Edition*. Thompson Learning, Inc., 2002.

[24] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, pages 97(1–2):273–324, 1997.

[25] Eric J. Kostelich. Data Assimilation Issues - Issues in Weather and Climate. `http://math.la.asu.edu/~eric/msri/ejk\_msri1.pdf`.

[26] Huan Liu and Lei Yu. Toward Integrating Feature Selection Algorithms for Classification and Clustering. *IEEE Transactions on Knowledge and Data Engineering*, 2005.

[27] S.F. McCormick. *Multigrid methods: theory, applications, and supercomputing.* M. Dekker, New York, 1988.

[28] Andrew W. Moore. Instance-based Learning, 2001. `http://www.cs.cmu.edu/~awm/tutorials/`.

[29] Paul Plassmann. CSE 557, Concurrent Matrix Computation, February 2004.

[30] Yousef Saad. *Iterative Methods for Sparse Linear Systems, Second Edition.* SIAM, 2003.

[31] Tsai. Machine Learning and Bioinformatics. `http://dspc11.cs.ccu.edu.tw/ml93/lecture7-instance-based-knn.pdf`.

[32] Ian H. Witten and Eibe Frank. *Data Mining, Practical Machine Learning Tools and Techniques, Second Edition.* Morgan Kaufmann Publishers, 2005.

[33] Harry Zhang. The Optimality of Naive Bayes. *Proceedings of the 17th International FLAIRS conference*, 2004. `http://www.cs.unb.ca/profs/hzhang/publications/FLAIRS04ZhangH.pdf`.