Towards Low-Cost, High-Accuracy Classifiers for Linear Solver Selection^{*}

Sanjukta Bhowmick, Brice Toth, and Padma Raghavan

Department of Computer Science and Engineering, The Pennsylvania State University,
University Park, PA 16802
bhowmick@cse.psu.edu,
btoth@cse.psu.edu,
raghavan@cse.psu.edu

Abstract. The time to solve linear systems depends to a large extent on the choice of the solution method and the properties of the coefficient matrix. Although there are several linear solution methods, in most cases it is impossible to predict apriori which linear solver would be best suited for a given linear system. Recent investigations on selecting linear solvers for a given system have explored the use of classification techniques based on the linear system parameters for solver selection. In this paper, we present a method to develop low-cost high-accuracy classifiers. We show that the cost for constructing a classifier can be significantly reduced by focusing on the computational complexity of each feature. In particular, we filter out low information linear system parameters and then order the remaining parameters to decrease the total computation cost for classification at a prescribed accuracy. Our results indicate that the speedup factor of the time to compute the feature set using our ordering can be as high as 262. The accuracy and computation time of the feature set generated using our method is comparable to a near-optimal one, thus demonstrating the effectiveness of our technique.

1 Introduction

The solution of sparse linear systems comprises the most time-consuming portion of many scientific simulations. There exist many classes of linear solvers, e.g., direct [7], iterative [2], and multigrid [18] methods. Preconditioning techniques are used to improve the convergence of iterative methods, leading to an even greater number of solver-preconditioner combinations (henceforth called solvers). Earlier research [10, 19] indicates that no linear solution method is consistently the best across different linear systems. Even with a single application, the “best” solution method can vary across time-steps and nonlinear iterations thereby making solver selection a very challenging problem. However, using appropriate solvers can significantly reduce the simulation time of the application [4, 10, 19].

Recent research concerns investigating the application of machine learning techniques [3, 6, 13, 14, 22] to predict efficient solvers. Classifiers are trained using a sample

^{*} This work was supported by grants CCF-0830679 and OCI-0720749 of the National Science Foundation.

dataset of linear systems and associated solvers. Each entry in the sample dataset consists of (i) the set of linear system properties (feature set), (ii) a solver and (iii) the solution criteria. In its simplest version, the machine learning algorithm builds a binary classifier. Given a new linear system (not in the database), a solver and a solution criteria (represented in the database), the classifier predicts whether or not the solver solves the linear system according to the given solution criteria.

Earlier work in this area [13, 14, 22] focuses primarily on generating high-accuracy classifiers and the problem of feature selection is not addressed. Results from [11] demonstrate that filtering out zero-variance features can reduce classifier construction time. These results indicate the need for further investigation on the choice of features and their impact on classifier accuracy and time for construction.

In this paper, we present a method for generating low-cost, high-accuracy classifiers. Section 2 contains a brief overview of classification methods. Sections 3 and 4 contain our main contributions including an ordering scheme to select features and an empirical evaluation of its effectiveness. Section 5 contains brief concluding remarks and future research directions.

2 Background: Supervised Learning and Feature Selection

Supervised learning involves designing a classification function based on a set of already classified data. Often *k-fold cross-validation* is used to improve the accuracy of the classifiers. The database is divided into k subsets. At each instance of learning, $k-1$ subsets are used as a *training set*, and the remaining subset is used as a *testing set*. The training set is used to build the classifier and the testing set is used to verify the accuracy of the classifier. This process is repeated k times, each time with a different subset as the testing set. The results are combined to produce the final classifier whose accuracy is measured by the percentage of correctly predicted entries in the testing sets [12, 21].

We consider an entry to be represented by a feature set $f_1, f_2, \dots, f_n$. The objective of a binary classifier is to determine whether an unknown entry E belongs to group $C = 0$ or $C = 1$. Some common supervised learning techniques are given below,

K Nearest Neighbor (NN(k)): In this method the class of entry E is determined according to the class of the majority of its k nearest neighbors in the feature space. The NN(k) algorithm is sensitive to the value of k , usually the higher the value, the better the classification.

Alternating Decision Trees (ADT): In the decision tree algorithm, each node in the tree represents a particular feature. At each point of traversing the tree, a classification decision is made based on the value of that feature in the unclassified entry. A decision tree with one root is known as a Decision Stump (DS). Alternating decision trees represent a weighted and more generalized form of decision trees containing predictors as well as decision nodes. The combined score of the visited predictor nodes and the final leaves determines the classification of the entry. The larger the decision tree, more accurate the classification.

Naive Bayes (NB): This method uses Bayes' theorem to classify the input vector. A naive bayes classifier assumes that the features defining the input vector are independent

of each other. The classification can be expressed as; $NB(E) = \frac{p(C=0)}{p(C=1)} \prod_i^n \frac{p(f_i|C=0)}{p(f_i|C=1)}$; where $p(f_i|C = c)$ is the probability that f_i occurs given that classification is c .

Support Vector Machines (SVM): This method is based on creating a separating hyperplane that maximizes the distance between the two classes in the feature space. The support vector machine classifier can be formulated as; select w, b to minimize $\|w\|$, such that $c_i(w \cdot x_i - b) \geq 1$, for $1 \leq i \leq n$, where w is the vector perpendicular to the separating hyperplane.

Feature selection is a preprocessing step in machine learning for eliminating redundant, noisy and irrelevant data [15], thereby reducing classification time and improving prediction accuracy. Steps for feature selection consist of generating a subset of the features, evaluating its effectiveness in improving classification, and once the final subset is determined, validating it across different data sets [17].

Feature selection methods can be broadly categorized as (i) the *filter model*, where a simple evaluation technique or filter is used to identify the feature subset (ii) the *wrapper model*, where the feature subset model is based on evaluation over a data mining algorithm [16] and (iii) the *hybrid model* which uses both an independent measure (filter) as well as a datamining algorithm for feature selection.

3 Towards Low Cost Feature Selection

The evaluation of linear system features represents a significant cost in generating the classifiers. The cost of computing a feature varies from being proportional to the size of the matrix to being even more expensive than solving the linear system. Properties related to matrix sparsity, structure and the norms typically exhibit computational complexity of $O(n)$ or $O(nnz)$, for a n by n matrix with nnz nonzeros and can be calculated to their exact values. However, spectral properties such as the singular values, eigenvalues or the condition number can be even more expensive to calculate than solving the linear system itself and can be approximated at best.

The cardinality of the feature set contributes to the cost of constructing and using the classifier. Low cardinality sets can be obtained by eliminating low information or redundant features. Low information features are those that do not contribute to the predictive ability of the classifier. Redundant features are those which, based on certain matrix properties, might have the same value and need not be computed multiple times. Since many linear systems are generated and solved as part of a larger simulation process, the properties of the matrix may not be known beforehand. Additionally, scientists using the classifier may lack the requisite expertise to eliminate the redundant features.

Earlier research in feature selection concerns eliminating features of zero variance since they do not contribute to the classifier prediction and identifying redundant features based on loading vectors [11]. This technique of eliminating redundant features, however, can be as expensive as generating an SVM classifier. In this paper, we have extended the definition of low-information features to those whose variance is below a certain threshold, α , not necessarily 0 (as in [11]). We also avoid identifying redundant features based on loading vectors and instead propose inexpensive techniques based on the variance and mean of the feature distribution. Additionally we propose a feature

ordering scheme based on the computational complexity of feature calculation, towards constructing a high-accuracy classifier at low cost.

Our two step method is specified below. In the first step, we eliminate low-information features based on statistics of feature distribution, and in the second step we order the remaining features in increasing order of computational complexity to obtain a low-cost feature set of low cardinality.

1. Feature Set Reduction: The predictive power of a feature depends on the distribution of its values across the dataset. Consider two extreme cases of distribution of values. When the values of a feature remain constant across all entries in the dataset, no extra information can be obtained from that property. At the other end, if the number of unique values of a feature is equal to the number of entries, we can base our predictions on just that one property.

Thus the variance across the values of an individual feature provides a good estimate of the statistical dispersion. If a set of feature vectors exhibit *exactly* the same variance and the same mean, it is extremely likely that any one feature from the set can be expressed as a linear function of the others. We can thus select any *one* of these features and eliminate the other redundant feature vectors.

2. Feature Set Ordering: The cardinality of the feature set represents the potential dimensionality of the data space. As we increase the number of features, we project the data points into higher dimensions which can potentially lead to better classification results. However, a large number of features also increases the time to construct the classifier. It is therefore desirable to construct a classifier without having to include all features. Towards this goal we propose selecting features in increasing order of their computational complexity to generate a high-accuracy, low-cost classifier.

Our methods operates on the original set of features, $F = f_1, f_2, \dots, f_n$, to 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. The cardinality of the feature set $\hat{F}$ is further reduced to the final set F^* . This is achieved by ordering the remaining features in increasing order of their computational complexity, $\psi(f_i)$, 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 . The final set of features in $\bar{F}$ gives the set F^* . The individual steps are given below;

1. Compute $\phi(f_i)$ and $\mu(f_i)$ for all $f_i \in F$.
2. Eliminate f_i if $\phi(f_i) \leq \alpha$.
3. For all $\tilde{F} \subset F$, where $\phi(f^i) = \phi(f^j)$ and $\mu(f^i) = \mu(f^j)$; for all pairs $f_i, f_j \in \tilde{F}$,
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.
4. After completion of Step 3 we have the reduced set $\hat{F}$.
5. Order $\hat{F}$ such that if $\psi(\hat{f}_i) \leq \psi(\hat{f}_j)$, then $i < j$ where $\hat{f}_i, \hat{f}_j \in \hat{F}$.
6. Set $\bar{F}$ to an empty set and $i = 0$.
7. While $K(\bar{F}) \leq A$
 $\bar{F} = \hat{f}_i \cup \bar{F}$.
 $i = i + 1$.
8. Set $F^* = \bar{F}$

4 Empirical Results

In this section we empirically evaluate the effectiveness of our feature selection scheme for classifier construction. Our experiments are based on a collection of 50 symmetric matrices¹, obtained from the University of Florida Sparse Matrix Collection [5]. We use a suite of linear solvers from the Portable Extensible Toolkit for Scientific Computing (PETSc) [1]. We select a combination of ten Krylov iterators and seven preconditioners and one instance where no preconditioner is used, making a total of 80 linear solvers. The Krylov methods used are: BiConjugate Gradient Squared (BCGS), Conjugate Gradient (CG), Conjugate Gradient Squared (CGS), Chebychev, Generalized Minimal Residual (GMRES) with restart values of 5, 30, and 60, Minimal Residual (MINRES), Richardson, and Transpose-Free Quasi-Minimal Residual (TFQMR).

Each Krylov method is used with the set of the following preconditioners: Incomplete Cholesky factorization (ICC), with level of fill 0, Incomplete LU Factorization (ILU) with levels of fill 0, 1, 2, and 3, Jacobi, and Successive Over Relaxation (SOR). More details about the Krylov iterators and the preconditioners can be found in [1, 20]. The total number of entries in the dataset of matrices and associated solvers is 3969 (a small percentage of entries were discarded where the simulation failed to terminate).

We use Anamod [9], developed as part of the SALSA [6] package, to calculate the properties of the matrices. Each matrix is associated with 57 features, reflecting the sparsity, variations within the matrix structure, the norm, and the spectral properties. Table 1 enumerates some of the important features, the complete list of features can be found in [8]. The complexity of calculating the features is given by $O(n)$, $O(nnz)$ and $O(ls)$, i.e., proportional to the time taken to solve the linear system.

Our database of matrix features and solvers is classified into two groups. If for a linear system-solver pair the solution converges in 1500 iterations, the entry is labeled as group 1, otherwise it is labeled as group 0. In our dataset, out of the 3969 entries, 37.2% converged according to the solution criteria, which required the residual norm to be at least 10^{-3} . We constructed classifiers based on the learning techniques described in Section 2. We used the Weka [21] toolkit to implement these methods on a 10-fold cross validation of the dataset.

4.1 Analysis of Results

Features Set Reduction: We calculate the variance of the features in the dataset and eliminate all those whose variance is lower than 10^2 . The graph of the feature variance is given in Figure 1. Out of the 57 properties, 20 have variance below this threshold. We also observe that the range of accuracy is higher when the classifiers are generated from features in the high variance range. For example, the maximum accuracy of a classifier generated using SVM with one feature from the zero variance group is at

¹ The matrix IDs are: msc00726, msc01050, msc01440, nasa2146, nasa2910, bcscstk15, msc04515, crystk01, bcscstk16, s2rmq4m1, s3rmq4m1, s3rmt3m1, nd3k, msc10848, t2dah_a, bcscstk18, vibrobox, crystm02, dis120, bodyy4, bodyy5, bodyy6, t3dl_a, bcscstk36, msc23052, smt, wathen100, fdm2, jnlbrng1, torsion1, minsurfo, c-62, c-66, nasasrb, qa8fk, qa8fm, finan512, s3dkq4m2, s3dktk3m2, m_t1, x104, shipsec8, ship_003, cfd2, augustus5, engine, pwtk, lin, af_shell4, and augustus7.

Table 1. Partial set of matrix 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)
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	$\sqrt{TrAsq(AA^T - A^T A)}$	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	$\max_j \sum_{i=1}^n (a(i, j)), \text{ if } a(i, j) = a(j, i)$	O(nnz)
symmetry-anorm	$\max_j \sum_{i=1}^n (a(i, j)), \text{ if } a(i, j) \neq a(j, i)$	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)
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	$\max(i - j), \text{ for all } a(i, j) \neq 0$	O(nnz)
right-bandwidth	$\max(j - i), \text{ for all } a(i, j) \neq 0$	O(nnz)
row-variability	$\max_i \log_{10} \frac{\max_j a(i, j) }{\min_j a(i, j) }$	O(nnz)
col-variability	$\max_j \log_{10} \frac{\max_i a(i, j) }{\min_i a(i, j) }$	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)
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)

most 63.69%. A similar experiment with a classifier generated from one feature from above the threshold range can yield as much as 80.67% accuracy.

We observe that, among the remaining 37 features there are some groups of redundant features such as (*row-variability*, *col-variability*), (*left-bandwidth*, *right-bandwidth*), (*n-rows*, *diag-zerostart*), (*norm1*, *normInf*, *symmetry-snorm*), (*normF*, *symmetry-fsnorm*), (*trace*, *trace-abs*), etc. By eliminating all but one feature from these groups we can further reduce the number of features to 21. Compared to the elements in the feature sets

Fig. 1. Range of variance of different matrix parameters listed in Table 1

used in similar experiments, such as 66 in Xu et. al. [22], 57 in Bhowmick et. al. [3], 33 in Holloway et. al. [13] and 22 in Fuentes [11], the cardinality of the set is lower.

Feature Set Ordering: We order the features according to their computational complexity (as given in Table 1). We provide the empirical results on the classifier accuracy for two sets of orderings. The first ordering, henceforth called the *increasing order*, is based on our strategy of adding features one by one in increasing order of computational cost. The *decreasing* order denotes the case when the features are added in reverse order, i.e., the most computationally expensive feature first and so forth.

Figure 2 compares the change in accuracy as the number of features associated with the classifier is increased. The top two sub-figures represent classifiers created using NB, SVM and NN(k) with $k=1$ and $k=10$. The lower two sub-figures represent classifiers created using ADT and DS. The left hand sub-figures represent the accuracy as features are added in increasing order and the right hand sub-figures denote the accuracy as the features are added in decreasing order. The dotted lines in the figures mark the boundary between the different groups of computational complexity, i.e., $O(n)$, $O(nnz)$ and $O(ls)$. The last value in the graphs denotes the accuracy achieved by a classifier generated using all the 21 features. The point of the highest accuracy obtained is marked by an ellipse.

We observe that a set of only 9 features is sufficient to create a classifier whose accuracy is comparable the accuracy of the classifier constructed using all features. This list of features in increasing order of complexity is: trace, diagonal-variance, diagonal-average, nrow, norm1, diagonal-dominance, nnzeros, left-bandwidth, and the condition number.

We observe that for machine learning methods SVM, NB, and NN(k) the best accuracy achieved by the increasing order is comparable or even higher than the best accuracy achieved using the decreasing order. The number of features required to obtain the classifier with the best accuracy is almost equal across the increasing and decreasing orders. Since the increasing order is based on selecting features of low computational complexity, it is evident that our method will generate classifiers of lower costs.

However, in the case of the decision tree based methods (ADT and DS) the classifier does not achieve high accuracy until the condition number is added to the feature set.

Fig. 2. Comparison of accuracy of different classifiers. Top sub-figures: Classifiers generated using NB, SVM and NN(k). Bottom sub-figures: Classifiers generated using ADT and DS. Left figures: Features added in increasing order of computational complexity. Right figures: Features added in decreasing order of computational complexity.

Table 2. Comparison of time to compute feature sets with increasing and decreasing orderings

Method	Feature Set	Computing Time (in secs)	Feature Set	Computing Time (in secs)	Speedup
	Decreasing Order		Increasing Order		
NB	(condition number left-bandwidth)	113.08	(trace)	.43	262.9
NN(1)	(condition number left-bandwidth)	113.08	(trace, diagonal-variance)	.89	127.05
NN(10)	(condition number left-bandwidth)	113.08	(trace)	.43	262.9
SVM	(condition number left-bandwidth nnzeros, diagonal-dominance)	113.95	(trace, diagonal-variance, diagonal-average, nnrows)	1.13	100.8
ADT	(condition number)	112.09	All 9	115.21	.97
DS	(condition number)	112.09	All 9	115.21	.97
Average Speedup					125.93
Geometric Mean of Speedup					30.67

Therefore decreasing order achieves the highest accuracy classifier with fewer features than the increasing order. We conjecture that this happens because the classification process depends on highly localized decisions.

Table 2 compares the time taken to compute the features for the highest accuracy classifiers (with at most the listed 9 features). Since Anamod does not give the time for calculating each individual feature, we obtain the timing results by computing the requisite feature values in MATLAB. We compute the features from a sample of the original matrix set and project the results. The values indicate that the increasing order achieves an average speedup of 125 (and a maximum of 262) compared to the decreasing order. 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.

We used a trial-and-error process to estimate the near-optimal feature set that generates classifiers with the highest accuracy. The highest accuracy feature sets for SVM, NB, NN(1), and NN(10) were identical with the highest accuracy ones obtained using our increasing order strategy. The near-optimal feature set for ADT and DS coincided with that obtained by the decreasing order strategy. Based on these observations, we conclude that for most machine learning methods our strategy for feature ordering achieves near-optimal classifier accuracy with low feature computation time.

5 Conclusions and Future Work

We have designed a technique for generating low-cost, high-accuracy classifiers based on ordering features in increasing order of their computational complexity. Based on the dataset and the machine learning method used, efficient feature selection can speed up the cost of building classifiers on average by a factor of 125. Most of our feature sets are identical to near-optimal ones demonstrating that our strategy achieves both low-cost and high-accuracy. Though we used our feature reduction technique on a linear system dataset, these results can easily be extended to other classification problems as well. As part of our future research we plan to study the effect of our algorithms over a variety of machine learning algorithms and datasets.

Acknowledgments

This work is a continuation of earlier research of one of the authors in application of machine learning to solver selection. We wish to thank, Victor Eijkhout, Erika Fuentes, Yoav Freund and David Keyes, the collaborators of the earlier project.

References

1. Balay, S., Buschelman, K., Gropp, W., Kaushik, D., Knepley, M., McInnes, L., Smith, B.F., Zhang, H.: PETSc Users Manual, Technical Report ANL-95/11 - Revision 2.2.1, Argonne National Laboratory (2004), <http://www.mcs.anl.gov/petsc>
2. Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., van der Vorst, H.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM (1994)
3. Bhowmick, S., Eijkhout, V., Freund, Y., Fuentes, E., Keyes, D.: Application of Machine Learning to the Selection of Sparse Linear Solvers, Technical Report (2007)

4. Bhowmick, S., Raghavan, P., McInnes, L., Norris, B.: Faster PDE-Based Simulations Using Robust Composite Linear Solvers. *Future Generation Computer Systems* 20, 373–386 (2004)
5. Davis T.: University of Florida Sparse Matrix Collection, NA Digest, 97(23) (1997), <http://www.cise.ufl.edu/research/sparse/matrices>
6. Dongarra, J., Eijkhout, V.: Self Adapting Numerical Algorithms for Next Generation Applications. *International Journal of High Performance Computing Applications* 17(2), 125–132 (2003)
7. Duff, I.S., Erisman, A.M., Reid, J.K.: *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford (1986)
8. Eijkhout V. and Fuentes E., Anamod Online Documentation, <http://www.tacc.utexas.edu/eijkhout/doc/anamod/html/>
9. Eijkhout, V., Fuentes, E.: A Proposed Standard for Numerical Metadata. *ACM Trans. Math. Software* (submitted)
10. Ern, A., Giovangigli, V., Keyes, D.E., Smooke, M.D.: Towards Polyalgorithmic Linear System Solvers For Nonlinear Elliptic Problems. *SIAM J. Sci. Comput.* 15(3), 681–703 (1994)
11. Fuentes, E.: *Statistical and Machine Learning Techniques Applied to Algorithm Selection for Solving Sparse Linear Systems*, Doctoral Dissertation, University of Tennessee (2007)
12. Hastie, T., Tibshirani, R., Friedman, J.H.: *The Elements of Statistical Learning*. Springer, Heidelberg (2001)
13. Holloway, A., Chen, T.-Y.: Neural Networks for Predicting the Behavior of Preconditioned Iterative Solvers. In: Gervasi, O., Gavrilova, M.L. (eds.) *ICCSA 2007, Part I. LNCS*, vol. 4705, pp. 302–309. Springer, Heidelberg (2007)
14. Kuefler, E., Chen, T.-Y.: On Using Reinforcement Learning to Solve Sparse Linear Systems. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) *ICCS 2008, Part I. LNCS*, vol. 5101, pp. 955–964. Springer, Heidelberg (2008)
15. John, G.H., Kohavi, R., Pfleger, K.: Irrelevant Features and the Subset Selection Problem. In: *International Conference on Machine Learning*, pp. 121–129 (1994)
16. Kohavi, R., John, G.H.: Wrappers for Feature Subset Selection. *Artificial Intelligence* 97, 273–324 (1997)
17. Liu, H., Lei, Y.: Toward Integrating Feature Selection Algorithms for Classification and Clustering. *IEEE Transactions on Knowledge and Data Engineering* 17(4), 491–502 (2005)
18. McCormick, S.F. (ed.): *Multigrid Methods: Theory, Applications, and Supercomputing*. M. Dekker, New York (1988)
19. Nachtigal, N.M., Reddy, S.C., Trefethen, L.N.: How Fast Are Nonsymmetric Matrix Iterations? *SIAM J. Matrix Anal. Appl.* 13(3), 778–795 (1992)
20. Saad, Y.: *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company (1995)
21. Witten, I.H., Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco (2005)
22. Xu, S., Zhang, J.: A Data Mining Approach to Matrix Preconditioning Problem. In: *Proceedings of the Eighth Workshop on Mining Scientific and Engineering Datasets (MSD 2005)* (2005)