Comparative Analysis of Krylov Iterative Methods in Support Vector Machines

Matthew Freed, Joseph Collins, and Jeonghwa Lee
Department of Computer Science, Shippensburg University, Shippensburg, PA, U.S.A

Abstract— *Data mining and classification is a growing and important field in bioinformatics. Machine learning algorithms such as support vector machines can be used with genetic information to predict disease susceptibility. In particular, single nucleotide polymorphisms have been analyzed to classify an individual into "sick" or "healthy" categories for a specific genetic disorder. The most computationally intensive part of the support vector machine algorithm involves solving a quadratic programming problem through the use of an iterative solver. This research examines various iterative solving methods that are utilized within support vector machines. In such a solver, the solution of the problem is obtained through successively converging on an optimal result. These solvers are analyzed based on efficiency and the accuracy of the classification.*

Keywords: Data mining, gene classification, Krylov iterative methods, support vector machine

1. Introduction

With the development of the deoxyribonucleic acid (DNA) microarray technique, it has become possible to gather genetic information at lower costs [2]. The greater amount of information available has led to an effort to apply data mining and classification techniques to this information [11]. The end goal is to develop an algorithm that, when given genetic information as input, can predict an individual's susceptibility to disease.

Single nucleotide polymorphisms (SNPs) show much promise in this search. SNPs are single base changes of one nucleotide in a strand of DNA. Sets of SNPs present in a single block of DNA can be gathered together in a genotype [4]. One method that has been used to analyze genetic information is the support vector machine (SVM). This classification algorithm treats each SNP genotype as a feature vector. The SVM builds a model after reading in sets of genotypes from individuals in the "healthy" and "sick" categories [6].

In building the model, the SVM constructs a hyperplane that best separates the data points into the two categories. To do this, it solves a quadratic programming problem [6]. When dealing with genetic information, the dimensionality of the data can be very large. There may be hundreds or thousands of SNPs in each feature vector. The resulting quadratic programming (QP) problem can be computationally intensive, and not feasibly solvable with a direct solver [12]. To overcome this, iterative solvers can be used. Iterative solvers approach a solution over many iterations to provide an approximation. In many cases, this approximation is good enough for practical purposes [12].

This paper is organized as follows: Section 2 gives a concise introduction to the implementation of the SVM. In Section 3, a selection of Krylov iterative methods are discussed in detail. Section 4 presents the numerical results of our experiments. Concluding remarks are made in Section 5.

2. Support Vector Machines

The SVM, first introduced by Vladimir Vapnik in 1992, has been established as a powerful algorithmic approach to the problem of classification, which belongs to the larger context known as supervised learning [12]. Within this supervised learning problem of classification, one is given a set of training data consisting of n individual points,

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathbb{R}^p,\, y_i \in \{-1, 1\}\}_{i=1}^{n}, \qquad (1)$$

where y_i may be the value of ± 1, which indicates the class for which $\mathbf{x}_i$ belongs—either in $(+1)$ or out (-1) of the set that one wishes to learn to recognize [3]. Each $\mathbf{x}_i$ from Eq. (1) is a real vector in p-dimensions that describes the data point. The initial goal of the SVM is to locate the maximum margin hyperplane that divides the points described by $y_i = 1$ from those as $y_i = -1$. A hyperplane may be represented as the set of points $\mathbf{x}$ which satisfies the following decision rule:

$$f(x) \equiv \mathbf{w} \cdot \mathbf{x} - b = 0, \qquad (2)$$

where $\mathbf{w}$ is a normal vector perpendicular to the hyperplane, and all training points with $y_i = 1$ lie on one side of the hyperplane, while all the training points with $y_i = -1$ lie on the other side [12]. SVMs aim to choose $\mathbf{w}$ (a normal vector to the hyperplane) and b (some offset) to maximize the distance between the parallel hyperplanes such that they are as far apart as possible while still separating the data, hence establishing $f(\mathbf{x})$ as the decision rule. Using Eq. (2), these parallel hyperplanes may be described as follows:

$$\mathbf{w} \cdot \mathbf{x} - b = 1, \qquad (3)$$

and

$$\mathbf{w} \cdot \mathbf{x} - b = -1. \tag{4}$$

For training data that are linearly separable—that is, two sets of points in p-dimensions that may be separated by a hyperplane—one may select the two hyperplanes of the margin in such a way that there are no points between them and then try to maximize their distance. As one increases the size of the margin, one must prevent data points from falling into it. To ensure that this does not occur, one must utilize the following constraints on Eq. (3) – (4):

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq +1 \qquad \text{when } \mathbf{y}_i = +1 \tag{5}$$

and

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1 \qquad \text{when } \mathbf{y}_i = -1. \tag{6}$$

Eq. (5) – (6) represent parallel hyperplanes that separate the data, which—together—are referred to as the fat plane [12]. Eq (5) – (6) may be rewritten as

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad \text{for all } 1 \leq i \leq n. \tag{7}$$

Using geometry, the perpendicular distance between these parallel hyperplanes (twice the margin) is

$$2 \times \text{margin} = 2(\mathbf{w} \cdot \mathbf{w})^{-\frac{1}{2}}. \tag{8}$$

Utilizing Eq. (7) – (8), one may construct the fattest possible fat plane, known as the maximum margin SVM [12], by solving a particular problem in quadratic programming:

$$\text{minimize:} \quad \tfrac{1}{2}\mathbf{w} \cdot \mathbf{w} \tag{9a}$$

$$\text{subject to:} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \quad i = 1, \ldots, m. \tag{9b}$$

When arriving to the solution of Eq. (9), some of the training data points will lie on the extreme boundaries of the fat plane, denoted the support vectors [12]. The Krylov iterative methods for solving this quadratic programming problem are discussed in the following section.

3. Krylov Iterative Methods

Given a square system of n linear equations with a vector of unknowns $\mathbf{x}$, we may construct the following matrix equation:

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{10}$$

where the components of Eq. (10) may be represented as

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}. \tag{11}$$

From Eq. (11), $\mathbf{A}$ may then be decomposed into a diagonal component $\mathbf{D}$ and strictly lower and upper triangular components $\mathbf{L}$ and $\mathbf{U}$:

$$\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}, \tag{12}$$

where the components of Eq. (12) may be represented as

$$\mathbf{D} = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}, \quad \mathbf{L} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}, \tag{13a}$$

$$\mathbf{U} = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{13b}$$

The system of linear equations from Eq. (10) – (13) may be rewritten as

$$(\mathbf{D} + \omega\mathbf{L})\mathbf{x} = \omega\mathbf{b} - [\omega\mathbf{U} + (\omega - 1)\mathbf{D}]\mathbf{x} \tag{14}$$

for a constant $\omega > 1$ [1]. Various iterative methods exist to solve the expression of Eq. (14).

3.1 Successive Overrelaxation Method

The successive overrelaxtion (SOR) method is derived by extrapolating the Gauss-Seidel method [8]. This extrapolation takes the form of a weighted average between the previous iterate and the computed Gauss-Seidel iterate successively for each component:

$$x_i{}^k = \omega\bar{x}_i^k + (1 - \omega)x_i{}^{k-1}, \tag{15}$$

where $\bar{x}$ represents a Gauss-Seidel iterate and ω is the extrapolation factor [1].

In matrix terms, the SOR algorithm may be written as follows:

$$\mathbf{x}^k = (\mathbf{D} - \omega\mathbf{L})^{-1}[\omega\mathbf{U} + (1 - \omega)\mathbf{D}]\mathbf{x}^{k-1} + \omega(\mathbf{D} - \omega\mathbf{L})^{-1}\mathbf{b}, \tag{16}$$

where the matrices $\mathbf{D}$, $\mathbf{L}$ and $\mathbf{U}$ represent the diagonal, strictly lower-triangular and strictly upper-triangular parts of $\mathbf{A}$ from Eq. (13), respectively [1].

The underlying success behind SOR is to choose a value for ω that accelerates the rate of convergence. When $\omega = 1$, the SOR method simplifies to the Gauss-Seidel method [1], yet it will fail to converge if $\omega \notin \{0, 2\}$ [8]. Generally speaking, it is impossible to choose the most desirable value for ω in advance, thus it is common to utilize the following heuristic estimate:

$$\omega = 2 - O(h), \tag{17}$$

where h is the mesh spacing of the discretization of the underlying physical domain [1].

3.2 Quasi-Minimal Residual Method

Iterative methods often exhibit irregular convergence behaviors. A related algorithm, known as the quasi-minimal residual (QMR) method [5], attempts to overcome this problem. The underlying idea behind this algorithm is to

solve the reduced tridiagonal system in a least squares sense. QMR also uses look-ahead techniques to avoid breakdowns in its Lanczos process, which makes it more robust than SOR [1].

3.3 Biconjugate Gradient Method

The biconjugate gradient (BiCG) method [9] is commonly used in solving systems of linear equations. It is a generalized form of the conjugate gradient method, in that it can be applied to matrices that are non-symmetric. To formulate the biconjugate gradient method as an iterative method, it is necessary to use a metric at each iteration to determine if the approximation vector $\mathbf{x}$ is closer to the solution $\mathbf{x}_*$. It has been shown that this solution is also the minimizer for the quadratic function [13]:

$$f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x^T A x} - \mathbf{x^T b}, \quad \mathbf{x} \in \mathbf{R}^n. \tag{18}$$

Therefore, as Eq. (18) is smaller than the previous iteration, the value of $\mathbf{x}$ is closer to the solution. Starting with an initial guess $\mathbf{x}_0$, the gradient of the function will be $\mathbf{Ax}_0 - \mathbf{b}$. If $\mathbf{x}_0$ is assumed to start at 0, the first basis vector p_1 will equal $\mathbf{b}$. Each of the other vectors in the basis is conjugate to the gradient of the function.

The error at each iteration is measured as the residual. This residual is defined as

$$r_k = b - Ax_k. \tag{19}$$

In calculating the basis vectors at each step, this residual from Eq. (19) is taken into account in order to move the approximation towards the solution. The value of x therefore is updated during each iteration to

$$x = x + \alpha \times p, \tag{20}$$

where α is based on the residual divided by $A \times p$.

The resulting algorithm involves two matrix vector products, including one transpose product. This is the major computational cost of the biconjugate gradient method [10].

3.4 Biconjugate Gradient Stabilized Method

The biconjugate gradient stabilized (BiCGSTAB) method [14] is a variation of the biconjugate gradient and conjugate gradient squared (CGS) methods. BiCGSTAB makes several improvements, most importantly in that it stabilizes the algorithm. CGS relies on the squaring of the residual, which can result in rounding errors that affect the approximation in greater amounts at each iteration. As a result, the convergence pattern may be irregular. BiCGSTAB smooths this convergence by updating the way the approximation x is determined at each step:

$$x = x + \alpha \times p + \omega \times s. \tag{21}$$

Here, ω from Eq. (21) is a scaling factor that allows the distance that the approximation changes to vary. Larger steps may be taken during iterations, which assists in speeding up convergence. The stabilizer s is what allows for a smoother convergence. It is based on the residual and the matrix:

$$s = r - \alpha A \times p. \tag{22}$$

Eq. (22) results in avoiding the irregular convergence that is associated with BiCG. Further, there is no transpose involved in this algorithm, which is often desirable for solving certain matrices [13].

4. Numerical Results

The dataset used with the SVM was taken from publicly available genetic information. It is derived from the 616 kilobase region on Chromosome 5q31 [4]. Within this region may contain the genetic variation that is responsible for Crohn's disease [3]. The data contains a total of 103 genotyped single nucleotide polymorphisms for each of the 387 genotypes. Of this, 144 of them are case and 243 are control.

The SVM package chosen was the Mangasarian-Musicant variation due to its brevity [12]. Each of the genotypes was treated as a feature vector and read as input to the SVM. The kernel function increased the dimensionality of the dataset using a linear kernel. Half of the genotypes were used as the training set. During the solving of the SVM, each of the four different iterative solvers was used to solve the quadratic programming portion. The other half of the genotypes was then classified using the training data. Table 1 shows the resulting data that was collected.

For each of the solvers, we measured the efficiency of the solver as well as the accuracy of the classification resulting from the solution. The number of iterations each solver took to converge as well as the time it took can be considered a measure of its efficiency. For the classification, a simple accuracy measurement consisting of the percent of genotypes correctly classified. The sensitivity and specificity of the data was also taken. The sensitivity is the proportion of individuals who have the disease and are correctly identified as such. The specificity is the proportion of individuals who do not have the disease and are correctly identified.

All of the solvers except QMR were able to achieve convergence within the maximum number of iterations. QMR terminated after 43 iterations as a result of a breakdown in the gamma variable. However, it still was able to produce a viable classification. The QMR algorithm is not as robust as some of the other methods, and the algorithm failed on this particular matrix.

Out of the four solvers, BiCG was the most accurate, correctly placing 64.1% of the SNPs into the proper category. The classification of BiCGSTAB was similar with a 62.1% accuracy. SOR and QMR both resulted in classifications with 60.2% accuracies. The accuracies of each of the solvers were relatively similar. Further, the results were comparable to the same dataset used with other SVM packages. In particular,

Table 1: Classification results of the QP solvers

Solver	Iterations	Solve Time (sec)	Accuracy (%)	Specificity (%)	Sensitivity (%)
SOR	60	0.0500	60.2	51.9	68.6
QMR	43	0.0050	60.2	51.9	68.6
BiCG	149	0.0020	64.1	61.5	66.7
BiCGSTAB	63	0.0012	62.1	53.8	70.5

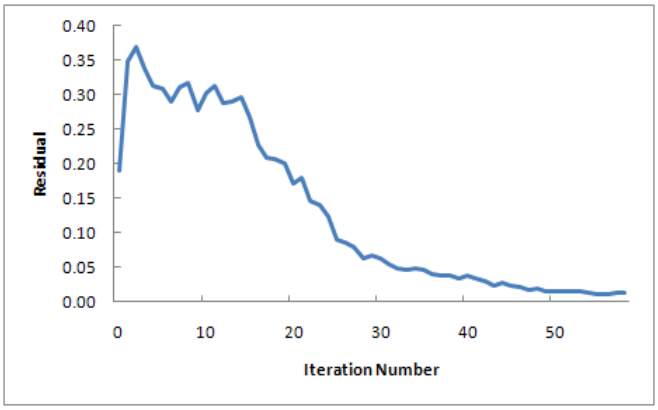

Fig. 1. Convergence history of the SOR solver

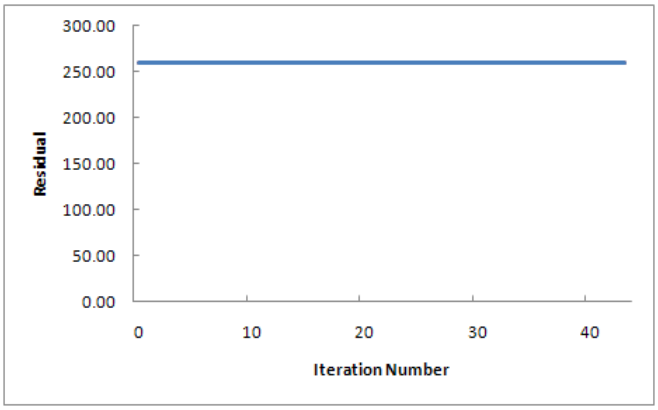

Fig. 2. Convergence history of the QMR solver

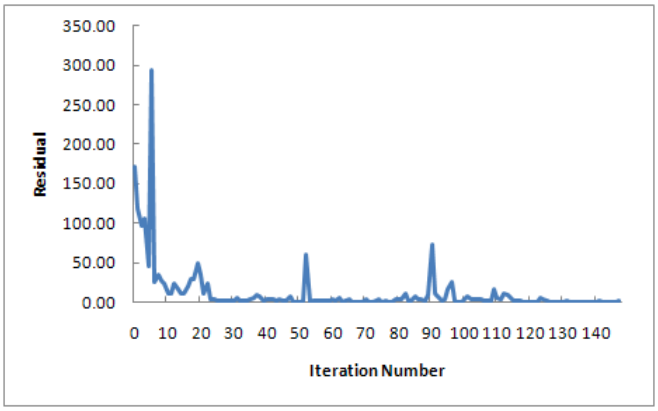

Fig. 3. Convergence history of the BiCG solver

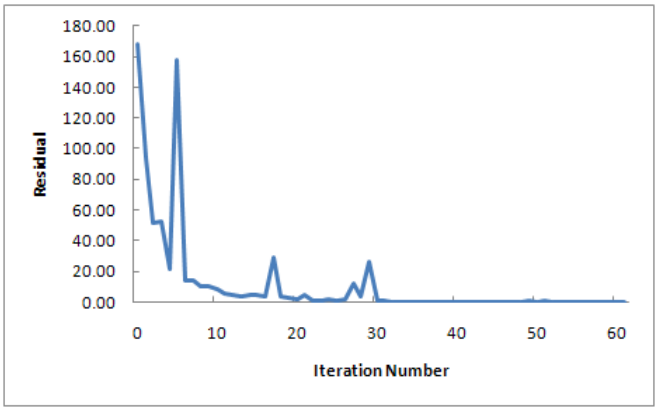

Fig. 4. Convergence history of the BiCGSTAB solver

the commonly used SVM-Light package resulted in 62% accuracy [7].

The convergence histories of the solvers can be seen in Figures 1–4. The failure of the QMR method can be seen as no change in the residual. SOR converges slower than BiCG and BiCGSTAB. These two solvers quickly reached a low residual value, and then slowly converged within the tolerance. The stabilizing effect of the BiCGSTAB algorithm over the BiCG can clearly be seen. The convergence history of BiCG is erratic, with many increases in the residual. The BiCGSTAB smooths this, resulting in a much more stable convergence.

In terms of time, BiCGSTAB was clearly the most efficient solver, converging in 0.0012s. Of note is that not all times correlated with the number of iterations. For example, SOR iterated a similar number of times as BiCGSTAB. However, each iteration took a greater amount of time, and as a result took longer to converge. The BiCG took the greatest number of iterations. The time per iteration was smaller than QMR and SOR, and as a result converged in less time.

5. Conclusion

Support vector machines can be applied confidently to the problem of classification of genetic data. As more and more genetic information becomes available, classification algorithms such as the SVM can be used to make useful

models based on the data. With the addition of sophisticated iterative methods, an accurate solution can be achieved in less time.

The numerical results demonstrate the efficiency of various iterative solvers. As can be seen with the failure of QMR, certain methods may not be applicable with certain matrices. The BiCGSTAB provided a model with high classification accuracy. It is also the most efficient of the methods examined in terms of time. Overall, the results suggest that BiCGSTAB is a robust algorithm that is a good choice for solving large quadratic programming problems. This experiment may assist researchers in selecting an iterative method when dealing with data mining using genetic information.

References

[1] Barrett, R., Berry, M., Chan, T. F,; 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, 2nd ed.* Society for Industrial and Applied Mathematics, Philadelphia, PA, (1994).

[2] Caron, L., Bell, J., *Association Study Designs for Complex Diseases*, Nature Reviews: Genetics, 91–98, (2001).

[3] Cortest, C. & Vapnik, V., *Support-Vector Networks*, Machine Learning, Vol. 20, No. 3, 273–297, (1995).

[4] Daly, M., Rioux, J., Schaffner, S., Hudson, T. & Lander, E., *High Resolution Haplotype Structure in the Human Genome*, Nature Genetics, Vol. 29, 229–232, (2001).

[5] Freund, R. & Nachtigal, N., *QMR: A Quasi-Minimal Residual Method for Non-Hermitian Linear Systems*, Numer. Math., Vol. 60, 315-339, (1991).

[6] Guyon, I., Weston, J., Barnhill, S., *Gene Selection for Cancer Classification using Support Vector Machines*, Machine Learning, 389–422, (2002).

[7] Joachims, T. *Making Large Scale SVM Learning Practical. Advances in Kernel Methods – Support Vector Learning*, MIT (1999).

[8] Kahan, W., *Gauss-Seidel Methods of Solving Large Systems of Linear Equations*, Ph.D. Thesis, Toronto, Canada, University of Toronto, (1958).

[9] Lanczos, C., *Solution of Systems of Linear Equations by Minimzed Iterations*, J. Research National Bureau Standards, Vol. 49, 33–53, (1952).

[10] Lee, J., Zhang, J. & Lu, C., *Performance of Preconditioned Krylov Iterative Methods for Solving Hybrid Integral Equations in Electromagnetics*, Journal of Applied Computational Electromagnetics Society, Vol. 18, No. 4, 54–61, (2003).

[11] Mao, W., Lee, J., *A Combinatorial Analysis of Genetic Data for Crohn's Disease*, Journal of Biomedical Science and Engineering, 52–58, (2008).

[12] Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P., *Numerical Recipes: The Art of Scientific Computing, 3rd ed.*, Cambridge University Press, Cambridge, (2007).

[13] Saad, Y., *Iterative Methods for Sparse Linear Systems, 2nd ed.*, Society for Industrial and Applied Mathematics, (2003).

[14] van der Vorst, H., *Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems*, SIAM J. Sci. Stat. Comput., Vol. 13, 631–644, (1992).