A Simple and Efficient Algorithm for Gene Selection using Sparse Logistic Regression

S. K. Shevade*†and S. S. Keerthi‡

November 29, 2002

Motivation:

This paper gives a new and efficient algorithm for the sparse logistic regression problem. The proposed algorithm is based on the Gauss-Seidel method and is asymptotically convergent. It is simple and extremely easy to implement; it neither uses any sophisticated mathematical programming software nor needs any matrix operations. It can be applied to a variety of real-world problems like identifying marker genes and building a classifier in the context of cancer diagnosis using microarray data.

Results: The gene selection method suggested in this paper is demonstrated on four real-world datasets and found to be consistent with the literature or biological knowledge of these sets.

Availability: The implementation of this algorithm is available at the site: `http://guppy.mpe.nus.edu.sg/~mpessk/SparseLOGREG.shtml`

Contact: `mpessk@guppy.mpe.nus.edu.sg`

1 Introduction

Logistic regression is a powerful discriminative method. It also has a direct probabilistic interpretation built into its model. One of the advantages of logistic regression is that it provides the user with explicit probablities of classification apart from the class label information. Moreover, it can be

*Dept of Computer Science and Automation, Indian Institute of Science, Bangalore - 560 012, India.

†This work was done when the author was visiting Genome Institute of Singapore, National University of Singapore, Singapore - 117 528.

‡Control Division, Dept. of Mechanical Engineering, National University of Singapore, Singapore-117 576.

easily extended to the multi-category classification problem. In this paper the problem of logistic regression is addressed with the application of gene expression data particularly in mind.

The development of microarray technology has created a wealth of gene expression data. Typically, these datasets involve thousands of genes (features) while the number of tissue samples is in the range 10–100. The objective is to design a classifier which separates the tissue samples into predefined classes (e.g. tumour and normal). Different machine learning techniques like Parzen windows, Fisher's linear discriminant, and decision trees have been applied to solve this classification problem (Brown et al., 2000). Since the data dimension is very large, Support Vector Machines (SVMs) have been found to be very useful for this classification problem (Brown et al., 2000; Furey et al., 2000). Apart from the classification task, it is also important to remove the irrelevant genes from the data so as to simplify the inference. One of the aims of the microarray data experiments is to identify a small subset of informative genes, called marker genes, which discriminate between the tumour and the normal tissues, or between different kinds of tumour tissues. Identification of marker genes from a small set of samples becomes a difficult task and there is a need to develop an efficient classification algorithm for such an application where the number of features is much larger than the number of samples. Furey et al. (2000) used Fisher score to perform feature selection prior to training. This method was compared against feature selection using radius-margin bound for SVMs by Weston et al. (2001). Guyon et al. (2002) introduced an algorithm, called Recursive Feature Elimination (RFE), where features are successively eliminated during training of a sequence of SVM classifiers. Li et al. (2002) introduced two Bayesian classification algorithms which also incorporate automatic feature selection. In their work, the Bayesian technique of *automatic relevance determination* (ARD) was used to perform feature selection.

The main aim of this paper is to develop an efficient algorithm for the sparse logistic regression problem. The proposed algorithm is very much in the spirit of the Gauss-Seidel method (Bertsekas and Tsitsiklis, 1989) for solving unconstrained optimization problems. The algorithm is simple, *extremely easy to implement*, and can be used for feature selection as well as classifier design, especially for the microarray dataset. It *does not need any sophisticated LP or QP solver*, nor does it involve any matrix operations which is a drawback of some of the previously suggested methods. Our algorithm also has the advantage of not using any extra matrix storage and therefore it can be easily used in cases where the number of features used is very high.

This paper is organized as follows. In section 2 we describe the problem formulation and the optimality conditions for the problem. The details of the proposed algorithm are presented in section 3. Section 4 discusses the feature selection and classifier design procedure. Computational experiments are reported in section 5. Section 6 concludes the paper. The pseudocode for the proposed algorithm is given in the appendix of this paper.

2 Problem Formulation and Optimality

In this paper we focus on the two category classification problem. The ideas discussed here can be easily extended to the multi-category problem and those details will be addressed in a future paper. Let $\{(\tilde{x}_i, y_i)\}_{i=1}^m$ denote the training set, where $\tilde{x}_i$ is the i-th input pattern, $\tilde{x}_i \in R^N$ and y_i is the corresponding target value; $y_i = 1$ means $\tilde{x}_i$ is in class 1 and $y_i = -1$ means $\tilde{x}_i$ is in class 2. Typically for the microarray data, $N \gg m$. Therefore, it is appropriate to consider using a linear classifier in the input space directly without transforming it into a higher dimensional feature space. For conveniently taking into account the bias term in the regression function let us define $x^T = (1, \tilde{x}^T)$. Let us specify the functional form of the regression function $f(x)$ as a linear model,

$$f(x_i) = \sum_{j=0}^{N} \alpha_j x_{ij} \tag{2.1}$$

where α denotes the weight vector. We address the first element (corresponding to the bias term) of the vectors x and α as the 'zeroth element'. To avoid notational clutter, throughout this paper we use the index i to denote the elements of the training set ($i = 1, \ldots, m$) and the index j to represent the elements of x and α ($j = 0, \ldots, N$, unless specified explicitly).

For diagnostic purposes, it is important to have a classifier which uses as few features (genes) as possible. Since we are interested in finding a small subset of genes which enable us to do good classification it is appropriate to use a sparse model of the regression function, that is, in the final function $f(x)$ in (2.1), only a small number of α's corresponding to relevant features will have nonzero values. Therefore, we solve the sparse logistic regression problem, which can be formulated as the following optimization problem:

$$\begin{array}{ll} \min_\alpha & \rho = \sum_i g(-y_i f(x_i)) \\ s.t. & \sum_{j=1}^N |\alpha_j| \leq t \end{array} \tag{2.2}$$

where $t \geq 0$ is a parameter that is tuned using techniques such as cross-validation and the function g is given by:

$$g(\xi) = \log(1 + e^{\xi}) \tag{2.3}$$

which is the negative log-likelihood function associated with the probabilistic model

$$\text{Prob}(y|x) = \frac{1}{1 + e^{-y \cdot f(x)}} \tag{2.4}$$

Once the α's are determined by solving (2.2), the class of the test sample, $\bar{x}$, is $+1$ if $f(\bar{x}) > 0$ and -1 otherwise.

The formulation (2.2) was first suggested by Tibshirani (1996) as an extension of the "LASSO" (Least Absolute Shrinkage and Selection Operator) method for the linear regression problem. The LASSO estimator for the regression problem is obtained by solving the following optimization problem:

$$\min_{\alpha} \quad \frac{1}{2} \sum_i (y_i - f(x_i))^2 \quad s.t. \quad \sum_{j=1}^{N} |\alpha_j| \leq t \tag{2.5}$$

In (Tibshirani, 1996), two iterative algorithms were suggested for the solution of (2.5). One of these algorithms treats (2.5) as a problem in $N+1$ variables and 2^N constraints. In each iteration it solves a linear least squares problem subject to a subset of constraints E. Once a solution is found in an iteration, a constraint violated by the current solution is added to the set E and a new feasible point is found. The linear least squares problem is then resolved subject to the new set of constraints E. This procedure is repeated until the optimality conditions are satisfied. Note that, during each iteration one constraint is added to the set E and there exist only a finite (2^N) number of constraints. Therefore, this algorithm will converge in a finite number of steps. Use of active set method was also suggested for the solution of (2.5) where every time the set E contains only the set of equality constraints satisfied by the current point. The second algorithm, suggested in (Tibshirani, 1996), states (2.5) as a problem in $2N + 1$ variables and $2N + 1$ constraints, which can be solved using any quadratic programming solver.

In (Osborne, Presnell, and Turlach, 2000), the problem (2.5) was treated as a convex programming problem and an efficient algorithm for computing the linear model coefficients was given using duality theory. This algorithm was stated only for the cases where the objective function is a quadratic loss function. These ideas, however, can be extended to problems with non-quadratic objective functions, for example (2.2), by optimizing the quadratic approximation of the cost function in (2.2) at any given point and repeating this procedure until convergence. Recently, Roth (2002) extended the ideas given in (Osborne et al., 2000) to a more general class of cost functions including the one given in (2.2). The proof of convergence of this algorithm to a general class of loss functions is also given therein. In this method, the problem (2.2) is transformed to a L_1 constrained least squares problem which can then be solved using iteratively re-weighted least squares (IRLS) method on a transformed set of variables. To avoid solving large dimensional constrained least squares problems, especially for the microarray data, Roth (2002) uses a wrapper approach where a maximum violating variable is added to a small variable set, X, and the constrained least squares problem is solved with respect to the variables in the set X only. This procedure is repeated till the optimality conditions are satisfied.

All of the above methods work well and are very efficient; but they rely on mathematical programming solvers that require detailed matrix operations. The main contribution of this paper is to utilize the special structure of (2.6) and devise a simple algorithm which is extremely easy to implement; it neither uses any mathematical programming package nor needs any matrix operations. The simplicity of our algorithm is evident from the pseudocode given in the appendix of this paper.

Using optimality conditions it can be shown that there exists a $\gamma > 0$ for which, (2.2) is equivalent to the following unconstrained optimization problem:

$$\min_{\alpha} \quad W = \gamma \sum_{j=1}^{N} |\alpha_j| + \sum_{i} g(-y_i f(x_i)) \tag{2.6}$$

This means that the family of classifiers obtained by varying t in (2.2) and the family obtained by varying γ in (2.6) are the same. The addition of the penalty term, $\sum_j |\alpha_j|$, to the original objective function can be seen as putting a Laplacian prior over the vector α. The above formulation thus promotes the choice of a sparse model, where the final α's are either large or zero. In this paper, we shall concentrate on the model in (2.6).

We solve (2.6) directly without converting it into its dual. The problem (2.6) can be solved using an appropriate unconstrained nonlinear programming technique. We choose to basically use the Gauss-Seidel method which uses coordinate-wise descent approach, mainly because the method is extremely easy to implement while also being very efficient. In this method, one variable is optimized at a time keeping the other variables fixed, and the process is repeated till the optimality conditions are satisfied. Asymptotic convergence of this method for a more general version of the problem has been proved in (Bertsekas and Tsitsiklis, 1989; Chapter 3, Proposition 4.1). It should be noted that strict convexity of g plays an important role in this proof.

We now derive the first order optimality conditions for (2.6). Since (2.6) is a convex programming problem these conditions are both necessary and sufficient for optimality. Let us define

$$\begin{aligned} \xi_i &= -y_i f(x_i) \\ F_j &= \textstyle\sum_i \frac{e^{\xi_i}}{1+e^{\xi_i}} y_i x_{ij} \end{aligned} \tag{2.7}$$

The first order optimality conditions for the problem (2.6) can be easily derived from geometry: (i) since W is differentiable with respect to α_0, $\frac{\partial W}{\partial \alpha_0} = 0$; (ii) if $j > 0$ and $\alpha_j \neq 0$, then, since W is differentiable with respect to α_j at such a α_j, we have $\frac{\partial W}{\partial \alpha_j} = 0$; and (iii) if $j > 0$ and $\alpha_j = 0$, then, since W is only directionally differentiable with respect to α_j at $\alpha_j = 0$, we require the right side derivative of W with respect to α_j to be non-negative and the left side derivative to be non-positive. These conditions can be rewritten as the following algebraic conditions.

$$\begin{array}{rcll} F_j & = & 0 & \text{if } j = 0 \\ F_j & = & \gamma & \text{if } \alpha_j > 0, j > 0 \\ F_j & = & -\gamma & \text{if } \alpha_j < 0, j > 0 \\ -\gamma \leq & F_j & \leq \gamma & \text{if } \alpha_j = 0, j > 0 \end{array}$$

Thus, if we define

$$\begin{aligned} viol_j &= |F_j| && \text{if } j = 0 \\ &= |\gamma - F_j| && \text{if } \alpha_j > 0, j > 0 \\ &= |\gamma + F_j| && \text{if } \alpha_j < 0, j > 0 \\ &= \max(F_j - \gamma, -\gamma - F_j, 0) && \text{if } \alpha_j = 0, j > 0 \end{aligned} \tag{2.8}$$

then the first order optimality conditions can be compactly written as

$$viol_j = 0 \quad \forall\, j \tag{2.9}$$

Since, in asymptotically convergent procedures it is hard to achieve exact optimality in finite time, it is usual to stop when the optimality conditions are satisfied up to some tolerance, τ. For our purpose, we will take it that any algorithm used to solve (2.6) will terminate successfully when

$$viol_j \leq \tau \ \ \forall \ \ j. \tag{2.10}$$

This can be used as a stopping condition for (2.6). Throughout, we will refer to optimality as optimality with tolerance.

In the next section, we describe the actual details of our algorithm for solving the problem (2.6).

3 A Simple Algorithm for Sparse Logistic Regression

The algorithmic ideas discussed here are based on the Gauss-Seidel method used for solving unconstrained optimization problem. In solving (2.6) using Gauss-Seidel method, one variable α_j which violates the optimality conditions is chosen and the optimization subproblem is solved with respect to this variable α_j alone, keeping the other α's fixed. This procedure is repeated as long as there exists a variable which violates the optimality conditions. The method terminates when the optimality conditions (2.9) are satisfied. Note that the objective function in (2.6) is strictly convex and it will strictly decrease at every step when the optimization subproblem is solved with respect to one variable. Repeated application of this procedure for the problem (2.6) will make sure that the algorithm will converge asymptotically (Bertsekas and Tsitsiklis, 1989) to the solution. Of course, if the stopping condition (2.10) is used then the algorithm will terminate in finite time.

For a given α, let us define the following sets: $I_z = \{j : \alpha_j = 0, j > 0\}$; and $I_{nz} = \{0\} \cup \{j : \alpha_j \neq 0, j > 0\}$. Also, let $I = I_z \cup I_{nz}$. The key to efficiently solving (2.6) using Gauss-Seidel method is the selection of the variable α_j in each iteration with respect to which the objective function is optimized. At optimality, it is expected that the resulting model is sparse, that is, there will be only a few weights α_j with nonzero values. It is important that the algorithm spends most of its time adjusting the nonzero α's and making the subset I_{nz} self-consistent as far as optimality of the variables in this subset is concerned. Therefore, it is appropriate to find the maximum violating variable, say α_v in the set I_z and then solve (2.6) using the variables in the set $I_{nz} \cup \{v\}$ only, until the optimality conditions for these variables are satisfied. This procedure can be repeated till no violator remains in the set I_z, at which point the algorithm can be terminated. This procedure can be best explained using Algorithm 1.

Algorithm 1

Input Training Examples
  Initialize α's to 0
  while Optimality Violator exists in I_z
    Find the maximum violator, v, in I_z
    repeat
      Optimize W w.r.t. α_v
      Find the maximum violator, v, in I_{nz}
    until No violator exists in I_{nz}
  end while
Output A set of α's for the function in (2.1)

Algorithm 1 can be thought of as a two loop approach. The type I loop runs over the variables in I_z to choose the maximum violator, while the type II loop iterates over the variables in I_{nz} for selecting the maximum violator. Once the maximum violating variable in a given set is chosen, some nonlinear optimization technique needs to be used to solve the unconstrained optimization problem (2.6) with respect to a single variable. Note that the objective function is convex. A combination of the bisection method and Newton method was used in our algorithm. In this method, two points L and H for which the derivative of the objective function has opposite signs are chosen. This ensures that the root always lies in a bracketed interval, $[L, H]$. The Newton method is tried first, and a check is made in the algorithm to make sure the iteration obtains a solution in this interval of interest. If the Newton method takes a step outside the interval we do not accept this next point, and instead resort to the bisection method. In the bisection method, the next point is chosen to be the midpoint of the given interval. Depending upon the sign of the derivative of the objective function at this point the new interval is decided. Now, the abovementioned procedure starting from the trying of the Newton step is repeated. Since it is always ensured that the root lies in the interval of interest, the method is guaranteed to reach the solution.

It is important to note that the objective function, W, has different right hand and left hand derivatives with respect to α_j at $\alpha_j = 0$. Therefore, if the nonzero variable attains a value of 0 when our algorithm is executed it is necessary to see whether further progress in the objective function can be made by altering the same variable (but now in the opposite direction). This will make sure that after successful termination of the type II loop the optimality conditions for the variables in I_{nz} are satisfied.

Note from (2.7) and (2.8) that for checking the optimality conditions for each variable it is necessary to calculate ξ_i for all the examples. Since this calculation is done repeatedly, the efficient implementation of the algorithm requires that ξ is stored in the memory for all the examples. After a single variable, say α_j, is updated it becomes necessary to update $\xi_i \; \forall \; i$. This can be done by using the following simple equation:

$$\xi_i^{new} = \xi_i^{old} + (\alpha_j^{old} - \alpha_j^{new}) y_i x_{ij} \tag{3.1}$$

Let us now comment on the speed of our algorithm for (2.6). For a given value of γ and a dataset containing 72 samples with 7129 features per sample, it took about 20 seconds to solve (2.6) on a SUN UltraSparc III CPU running on 750 MHz machine. Moreover, once a solution of (2.6) is found for a given value of γ, we can use that solution to devise a "good" starting point for solving (2.6) at γ_{new}. This idea is very useful for improving efficiency when γ needs to be tuned using cross-validation; see the section below.

4 Feature Selection and Classifier Design

Before we discuss the procedure for feature selection it is important to note that the solution of (2.6) is useful for the tasks of feature selection as well as classifier design. Since (2.6) automatically introduces sparseness into the model, the nonzero α's in the final solution of (2.6) help us in deciding the relevant features for classification. To identify the relevant features, we proceed as follows.

To design a classifier and identify the relevant features using (2.6), it is important to find the appropriate value of γ in (2.6). This can be done using techniques like k-fold cross-validation. For this purpose, the entire set of training samples is divided into k segments. For a given value of γ, the problem (2.6) is solved k times, each time using a version of the training set in which one of the segments is omitted. Each of these k classifiers is then tested on the samples from the segment which was omitted during training, and the "validation error" is found by averaging the test set results over all the k classifiers. This procedure is repeated for different values of γ and the optimal value $\gamma^\star$ is chosen to be the one which gives minimum validation error. The final classifier is then obtained by solving (2.6) using all the training samples and setting γ to $\gamma^\star$.

For a given value of γ, each of the k classifiers uses a set of genes (corresponding to nonzero α's) for classification. We assign a count of one to each gene if it is selected by a classifier at $\gamma^\star$. Since there are k different classifiers designed at $\gamma^\star$, a gene can get a maximum count of k meaning that it is used in all the k classifiers. To avoid any bias arising from a particular k-fold split of the training data[1], the above procedure is repeated 100 times, each time splitting the data differently into k folds and doing the cross-validation. The counts assigned to a gene for 100 different experiments are then added to give a *relevance count* for every gene. Thus a gene can have a maximum relevance count of $100k$ since k-fold cross-validation was repeated 100 times.

One of the aims of using the microarray data is to develop a classifier using as few genes as possible. It is therefore important to rank the genes using some method and then select some of these genes in a systematic way so as to design a classifier which generalizes well on the unseen data. As the relevance count of a gene reflects its importance in some sense, it is appropriate to rank the genes based on the relevance count. Having decided the order of importance of the genes, the next step is to choose the "right" set of genes to design a classifier. For this purpose, the following systematic method can be used.

Let S denote the set of features being analysed. To examine the performance of these features, the k-fold cross-validation experiment (on (2.6)) is repeated 100 times using only the features in the set S. To avoid any bias towards a particular value of γ, the validation error at every value of γ is averaged over those 100 experiments. The minimum of these validation error values over different choices of γ then gives the average validation error for the set S. Ideally, one would like to identify the set of few features for which the average validation error is minimum. For this purpose, the top ranked genes (ranked according to relevance count) are added to the set S one-by-one, and the average validation error is calculated. The procedure is repeated as long as adding the next top ranked gene to the set S does not reduce the average validation error significantly. This final set S of genes is then used along with all the training examples to design a classifier for diagnostic purposes.

[1] This issue is particularly important for microarray data since the number of examples is usually small.

Note that this final set of genes is derived from all the training samples available. Typically for the microarray dataset, the training set size is very small and there are no test samples available. Therefore, one needs to use the entire training data for classifier design. The average validation error mentioned earlier uses the entire training set for gene selection and so it is not an "unbiased estimate". To examine how good the feature selection procedure is, it becomes necessary to give an unbiased estimate of this procedure. This estimate can be obtained using techniques of external cross-validation suggested in (Ambroise and McLachlan, 2002). For this purpose, the training data is split into P different segments ($P = 10$ is a good choice). The gene selection procedure described above is executed P times, each time using the training samples derived by omitting one of the P segments and testing the final classifier (obtained with the reduced set of genes) on the samples from the segment which was omitted during training. This procedure gives an unbiased estimate of the performance of the gene selection method described earlier. We will call this estimate as the average test error. As one might expect, the average test error is usually higher than the average validation error.

5 Numerical Experiments

In this section we report the test performance of the proposed gene selection algorithm on some publicly available datasets: colon cancer, leukaemia, ovarian cancer, and breast cancer. As the problem formulation (2.6) and the one used in (Roth, 2002) are the same[2], the results of different algorithms for the same formulation are expected to be close.

For the colon cancer dataset (Alon et al., 1999) the task is to distinguish tumour from normal tissue using microarray data with 2000 features per sample. The original data consists for 22 normal and 40 cancer tissues. The ovarian cancer dataset (Schummer et al., 1999) consisted of 30 examples obtained from from ovarian tumours and 24 normal examples. Each of these examples uses 1536 features.

For the leukaemia dataset (Golub et al., 1999) the task is to distinguish acute myeloid leukaemia (AML) from acute lymphoblastic leukaemia (ALL). The dataset consists of 47 and 25 samples of type ALL and AML respec-

[2]The algorithm for the solution of (2.6) and the actual method of selecting relevant genes used by Roth (2002), however, are quite different from our method described in section 4.

tively, with 7129 features per sample. These datasets are also available at `http://lara.enm.bris.ac.uk/colin`.

We also evaluated our algorithm on the breast cancer dataset (West et al., 2001). The original dataset consisted of 49 tumour samples. The number of genes representing each sample is 7129. The aim is to classify these tumour samples into estrogen receptor-positive (ER+) and estrogen receptor-negative (ER-). West et al. (2001) observed that in five tumour sample cases, the classification results using immunohistochemistry and protein immunoblotting assay conflicted. Threfore, we decided to exclude these five samples from our training set. The dataset is available at `http://mgm.duke.edu/genome/dna_micro/work/`.

Each of these datasets was pre-processed using the following procedure. The microarray dataset was arranged as matrix with m rows and N columns. Each row of this matrix was standardized to have mean zero and unit variance. Finally, each column of the consequent matrix was standardized to have mean zero and unit variance. This transformed data was then used for all the experiments.

The results of the gene selection algorithm on these different datasets are reported in Figures 1-4 and Tables 1-4. As discussed in section 4, the relevance count of every gene was obtained by repeating the k-fold cross-validation procedure 100 times, where k was set to 3. Tables 1-4 denote the important genes used in the classifier design for each of the datasets alongwith their relevance counts.

For the colon cancer dataset, the top three ranked genes are same as those obtained in (Li et al., 2002), although the gene selection procedure adopted there was different than the one we used. For the ovarian cancer dataset, the top ranked gene (No. 1526) was same while the remaining features listed in Table 3 were different from those reported in (Li et al., 2002). The main reason for this is that the classification hypothesis need not be unique as the samples in gene expression data typically lie in a high dimensional space. Moreover, if the two genes are co-regulated then it is possible that one of them might get selected during the gene selection process.

Some comments about the genes selected for the breast cancer dataset (shown in Table 4) are worthy of mention. Cathepsins have been shown to be good markers in cancer (Kos and Schweiger, 2002). Recently, it has been shown that YB-1, also called Nuclease-sensitive element-binding protein 1 (NSEP1), is elevated in breast cancer; see the following site: `http://www.wfubmc.edu/pathology/faculty/berquin.htm`. Moreover, microtubule associated protein is the cell cycle control gene, and therefore is

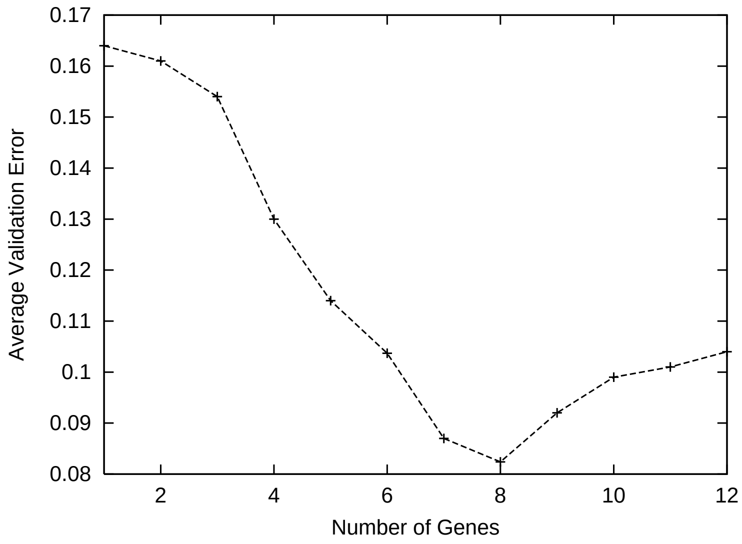

Figure 1: Variation of validation error on the colon cancer dataset, shown as the number of genes used in the feature set. The average validation error reached the minimum when eight genes, listed in Table 1, were used.

Table 1: Selected top eight genes with their relevance counts for the colon cancer dataset

Sr. No.	Gene Annotation	Relevance Count
1	H.sapiens mRNA for GCAP-II/uroguanylin precursor	229
2	COLLAGEN ALPHA 2(XI) CHAIN (Homosapiens)	165
3	MYOSIN HEAVY CHAIN, NONMUSCLE (Gallus gallus)	151
4	PLACENTAL FOLATE TRANSPORTER (Homo sapiens)	145
5	ATP SYNTHASE COUPLING FACTOR 6, MITOCHONDRIAL PRECURSOR (HUMAN)	124
6	GELSOLIN PRECURSOR, PLASMA (HUMAN)	117
7	Human mRNA for ORF, complete cds.	107
8	60S RIBOSOMAL PROTEIN L24 (Arabidopsis thaliana)	100

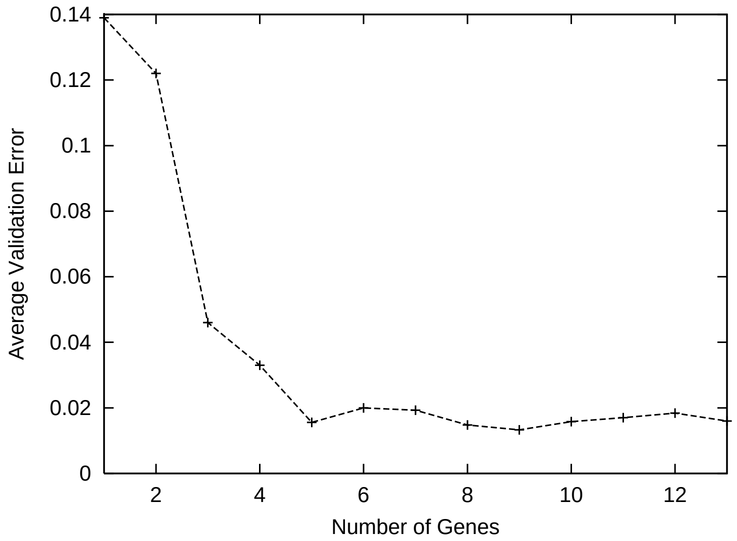

Figure 2: Variation of validation error on the leukaemia dataset, shown as the number of genes used in the feature set. The average validation error reached the minimum when nine genes were used. The list of these nine genes is given in Table 2.

Table 2: Selected top nine genes with their relevance counts for the leukaemia dataset

Sr. No.	Gene Number	Relevance Count
1	1779	267
2	4951	244
3	4847	241
4	3847	190
5	6184	182
6	1834	175
7	6539	174
8	1882	155
9	6169	144

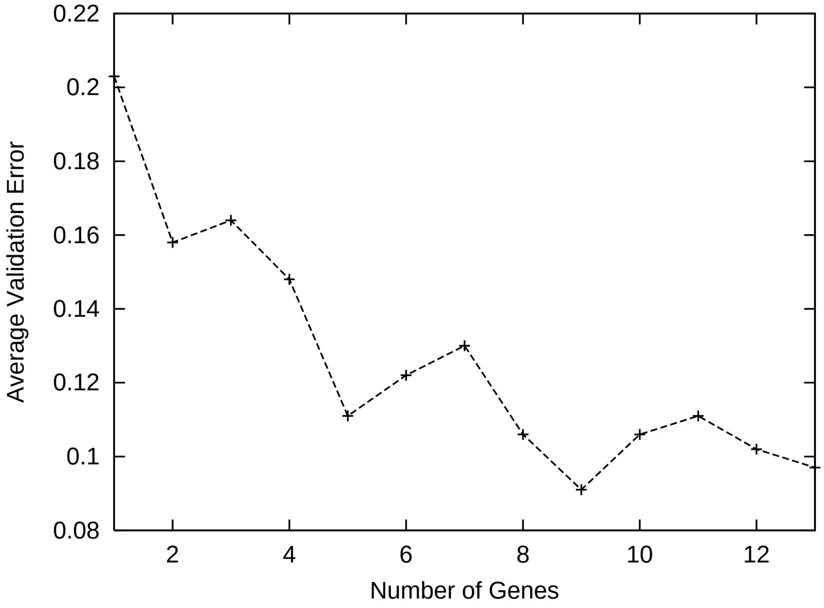

Figure 3: Variation of validation error on the ovarian cancer dataset, shown as the number of genes used in the feature set. The average validation error reached the minimum when nine genes, listed in Table 3, were used.

Table 3: Selected top nine genes with their relevance counts for the ovarian cancer dataset

Sr. No.	Gene Number	Relevance Count
1	1526	283
2	305	234
3	323	209
4	337	200
5	1387	196
6	389	191
7	300	189
8	396	186
9	467	183

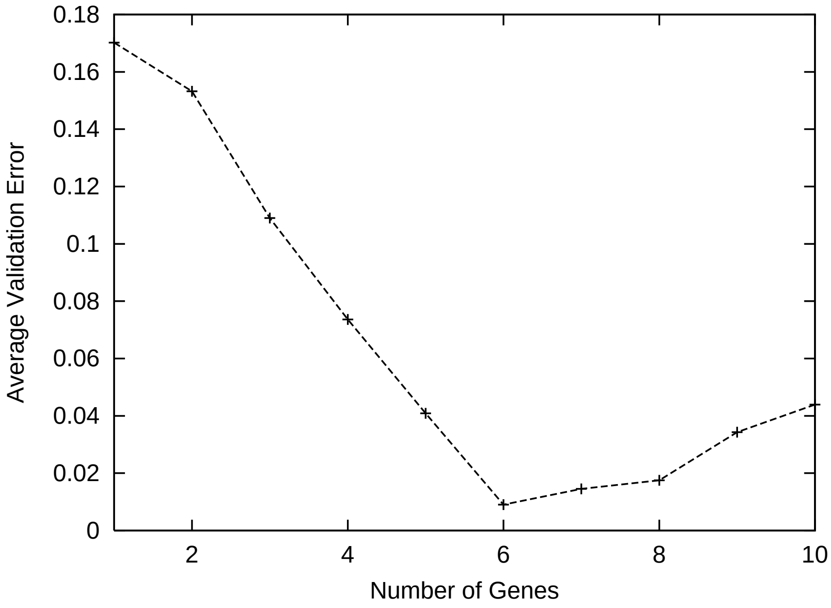

Figure 4: Variation of average validation error on the breast cancer dataset, shown as the number of genes used in the feature set. The average validation error reached the minimum when six genes were used. These genes are listed in Table 4.

Table 4: Selected top six genes with their relevance counts for the breast cancer dataset

Sr. No.	Gene Annotation	Relevance Count
1	Human splicing factor SRp40-3 mRNA, complete cds	171
2	H.sapiens mRNA for cathepsin C	151
3	Human mRNA for KIAA0182 gene, partial cds	133
4	Y box binding protein-1 (YB-1) mRNA	130
5	Human transforming growth factor-beta 3 (TGF-beta3) mRNA, complete cds	127
6	Human microtubule-associated protein tau mRNA, complete cds	115

Table 5: Average test error for different datasets

Dataset	Average Test Error
Leukaemia	0.069
Colon Cancer	0.177
Ovarian Cancer	0.167
Breast Cancer	0.181

associated with cancer. TGF-β has been shown to be a critical cytokine in the development and progression of many epithelial malignancies, including breast cancer (Arrick and Derynck, 1996).

The generalization behaviour of our algorithm was examined using the technique of external cross-validation described at the end of section 4. The average test error for each of the datasets is reported in Table 5.

6 Conclusion

We have presented a simple and efficient training algorithm for feature selection and classification using sparse logistic regression. The algorithm is useful, especially for problems where the number of features is much higher than the number of training set examples. The algorithm, when applied to gene microarray datasets, gives results comparable with other methods. It can be easily extended to the case where the aim is to identify the genes which discriminate between different kinds of tumour tissues (multi-category classification problem). The proposed algorithm can also be used for solving other problem formulations where strictly convex cost functions are used, for example, the LASSO problem formulation where quadratic loss function is used. In fact, in the case where the quadratic loss function is used, the one dimensional optimization problem becomes very simple since its solution can be reached in one Newton step. We are currently investigating the extension of the proposed algorithm to sparse versions of kernelized formulations like LS-SVM (Suykens and Vandewalle, 1999) and kLOGREG (Roth, 2001).

Acknowledgements

The authors thank Phil Long for his valuable comments and suggestions on the draft of this paper.

References

Alon, U., Barkai, N., Notterman, D., Gish, K., Ybarra, S., Mack, D., and Levine, A. (1999). Broad patterns of gene expression revealed by clustering analysis of tumour and normal colon cancer tissues probed by oligonucleotide arrays. *Cell Biology*, *96*, 6745–6750.

Ambroise, C., and McLachlan, G. J. (2002). Selection bias in gene extraction on the basis of microarray gene expression data. *Proceedings of the National Academy of Sciences, USA*, *99*, 668–674.

Arrick, B. A., and Derynck, R. (1996). The biological role of transforming growth factor beta in cancer development. In Waxman, J. (Ed.), *Molecular Endocrinology of Cancer*, pp. 51–78. Cambridge University Press, New York, USA.

Bertsekas, D. P., and Tsitsiklis, J. N. (1989). *Parallel and distributed computation: numerical methods.* Prentice Hall, Englewood Cliffs, NJ, USA.

Brown, M., Grundy, W., Lin, D., Cristianini, N., Sugnet, C., Furey, T., Ares, Jr., M., and Haussler, D. (2000). Knowledge based analysis of microarray gene expression data using support vector machines. *Proceedings of the National Academy of Sciences, USA*, *97*, 262–267.

Furey, T., Cristianini, N., Duffy, N., Bednarski, D., Schummer, M., and Haussler, D. (2000). Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, *16*, 906–914.

Golub, T., Slonim, D., Tamaya, P., Huard, C., Gaasenbeek, M., Mesirov, J., Coller, H., Loh, M., Downing, J., Caligiuri, M., Bloomfield, C., and Lander, E. (1999). Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, *286*, 531–537.

Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, *46*, 389–422.

Kos, J., and Schweiger, A. (2002). Cathepsins and cystatins in extracellular fluids - useful biological markers in cancer. *Radiol. Oncol.*, *36*, 176–179.

Li, Y., Campbell, C., and Tipping, M. (2002). Bayesian automatic relevance determination algorithms for classifying gene expression data. Available at *http://lara.enm.bris.ac.uk/cig/pubs/2002/bio3.ps.*

Osborne, M., Presnell, B., and Turlach, B. (2000). On the LASSO and its dual. *Journal of Computational and Graphical Statistics*, *9*, 319–337.

Roth, V. (2001). Probabilistic discriminative kernel classifiers for multiclass problems. In Radig, B., and Florczyk, S. (Eds.), *Pattern Recognition–DAGM'01*, pp. 246–253. Springer. Available at *http://www-dbv.informatik.uni-bonn.de/pdf/roth.dagm01.pdf.*

Roth, V. (2002). The generalized LASSO: a wrapper approach to gene selection for microarray data. Tech. rep. IAI-TR-2002-8, University of Bonn, Computer Science III, Bonn, Germany.

Schummer, M., Ng, W., Bumgarner, R., Nelson, P., Schummer, B., Bednarski, D., Hassell, L., Baldwin, R., Karlan, B., and Hood, L. (1999). Comparative hybridization of an array of 21,500 ovarian cdnas for the discovery of genes overexpressed in ovarian carcinomas. *Gene*, *238*, 375–385.

Suykens, J., and Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural Processing Letters*, *9*(3), 293–300.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, *58*(1), 267–288.

West, M., Blanchette, C., Dressman, H., Huang, E., Ishida, S., Spang, R., Zuzan, H., Olson, Jr., J. A., Marks, J. R., and Nevins, J. R. (2001). Predicting the clinical status of human breast cancer by using gene expression profiles. *Proceedings of the National Academy of Sciences*, *98*, 11462–11467.

Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., and Vapnik, V. (2001). Feature selection for SVMs. In *Advances in Neural Information Processing Systems 13*, pp. 668–674.

Appendix: Pseudocode for the Algorithm

Notes for the pseudocode given below:

1. The index i runs over the number of examples while the index j runs over the number of features plus one.

2. The α_j are stored in `alpha` array and ξ_i are stored in `xi` array.

3. The sets `I_z` and `I_nz` denote the sets I_z and I_{nz} respectively, defind in section 3. Note that $I = I_z \cup I_{nz}$.

4. The function `MAX` returns the maximum value of its arguments.

5. The second derivative of W with respect to the variable α_j (denoted by `Delta_j` in the pseudocode) at a nonzero value of α_j $(j \neq 0)$ or at any value of α_j $(j = 0)$ is given by

$$\Delta_j = \sum_i \frac{e^{\xi_i}}{1+e^{\xi_i}} {x_{ij}}^2. \qquad (A.1)$$

6. Although the objective function in (2.6) is not continuously differentiable with respect to α_j, $j \neq 0$, we define a quantity called $slope_j$ for every variable. At the points where the function is differentiable, $slope_j$ denotes the derivative of the objective function with respect to α_j. At the point of non-differentiability, we define $slope_j$ to be: the right side derivative if it is negative; the left side derivative if it is positive, and 0 otherwise. This slope of the objective function with respect to each variable can be calculated using the following equations.

$$\begin{aligned} slope_j &= -F_j && \text{if } j = 0 \\ &= \gamma - F_j && \text{if } \alpha_j > 0 \text{ or } (\alpha_j = 0 \text{ and } \gamma - F_j < 0) \\ &= -\gamma - F_j && \text{if } \alpha_j < 0 \text{ or } (\alpha_j = 0 \text{ and } -\gamma - F_j > 0) \\ &= 0 && \text{otherwise} \end{aligned} \qquad (A.2)$$

Note that the calculation of $slope_j$ and $viol_j$ (in (2.8)) requires the knowledge of F_j, which should be calculated after the ξ_i are available for the current values of α_j.

7. τ in (2.10) is referred to as `tol` in the pseudocode; 10^{-6} is a suitable value for it. For the finite termination of the procedure that optimizes W with respect to α_j, we ensured that $|slope_j| \leq .1 * tol \ \forall \ j$.

8. At the beginning of a step when α_j is optimized, the interval $[L, H]$ that brackets the minimizer is set as follows.

   Let κ denote the maximum floating point number represented on a computer. Also, let s_{j0} denote the slope (defined in the previous item) of the objective function with respect to α_j at $\alpha_j = 0$.

   Case 1: $j = 0$.

   $$L = -\kappa, \quad H = \kappa \tag{A.3}$$

   For the cases 2 through 7 given below, $j > 0$.

   Case 2: $\alpha_j > 0, slope_j > 0, s_{j0} \geq 0$

   $$\alpha_j = 0, \quad L = -\kappa, \quad H = 0 \tag{A.4}$$

   Case 3: $\alpha_j > 0, slope_j > 0, s_{j0} < 0$

   $$L = 0, \quad H = \alpha_j \tag{A.5}$$

   Case 4: $\alpha_j \geq 0, slope_j < 0$

   $$L = \alpha_j, \quad H = \kappa \tag{A.6}$$

   Case 5: $\alpha_j < 0, slope_j < 0, s_{j0} \leq 0$

   $$\alpha_j = 0, \quad L = 0, \quad H = \kappa \tag{A.7}$$

   Case 6: $\alpha_j < 0, slope_j < 0, s_{j0} > 0$

   $$L = \alpha_j, \quad H = 0 \tag{A.8}$$

   Case 7: $\alpha_j \leq 0, slope_j > 0$

   $$L = -\kappa, \quad H = \alpha_j \tag{A.9}$$

   Note that for the cases 2 and 5 where α_j is reset to zero, it becomes necessary to recompute $\xi_i \ \forall \, i$ and set $slope_j$ to s_{j0}.

```
procedure InitializeVariables()
{
   initialize alpha[j], xi[i] to 0 for all i and j;
   Store the index 0 in I_nz and the remaining indices in I_z;
   return;
}
procedure FindMaxViolator(X)
{
   % Finds the maximum violating example from the set X
   MaxViol = tol; v = -1;
   loop j over X
       Compute viol_j;
       if (viol_j > MaxViol) {
           MaxViol = viol_j; v = j;
       }
   return(v);
}
procedure OptimizeForAlpha(j)
{
   Compute slope_j, L, and H using (A.2) through (A.9)
   while (|slope_j| > .1*tol) {
       Compute Delta_j; alpha_old = alpha[j];
       alpha[j] = alpha_old - slope_j/Delta_j;
       if (alpha[j] <= L || alpha[j] >= H)
            alpha[j] = (L+H)/2;
       Update xi_i for all i in I using (3.1) and compute slope_j using (A.2).
       if (slope_j > .1*tol)
           H = alpha[j];
       else if (slope_j < -.1*tol)
           L = alpha[j];
   }
   if (j > 0) {
      if  (|alpha[j]| > 0)
          Store the index j in I_nz;
      else
          Store the index j in I_z;
   }
   return;
}
```

```
procedure main()
{
   % If the solution is available at some value of gamma=gamma1 and solution
   % is to be found at gamma=gamma2 then retain the old values of alpha
   % and other variables and set flag to 1. Otherwise set flag to 0
   if (flag == 0)
       InitializeVariables();
   id = FindMaxViolator(I_z);
   if (id < 0)
       id = FindMaxViolator(I_nz);
   while (id >= 0) {
       do {
           OptimizeForAlpha(id);
           id = FindMaxViolator(I_nz);
       } while (id >= 0);
       id = FindMaxViolator(I_z);
   };
}
```