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

S. K. Shevade
Dept. of Computer Science and Automation
Indian Institute of Science
Bangalore - 560012
India

shirish@csa.iisc.ernet.in

S. S. Keerthi
Dept. of Mechanical Engineering
National University of Singapore
Singapore - 117576
Republic of Singapore

mpessk@nus.edu.sg

1 Introduction

In this article we present experimental results using our algorithm on some more publicly available datasets. The pseudocode for the proposed gene selection algorithm is also given. Different variables and equations mentioned in this document are described in the main paper.

2 Some more results

In this section we report the performance of the proposed gene selection algorithm on leukaemia and ovarian cancer datasets.

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

Each of these datasets were pre-processed using the procedure described in the main paper. The results of the gene selection algorithm on these datasets are reported in Figures 1-2 and Tables 1-2. The average test error for these datasets is shown in Table 3.

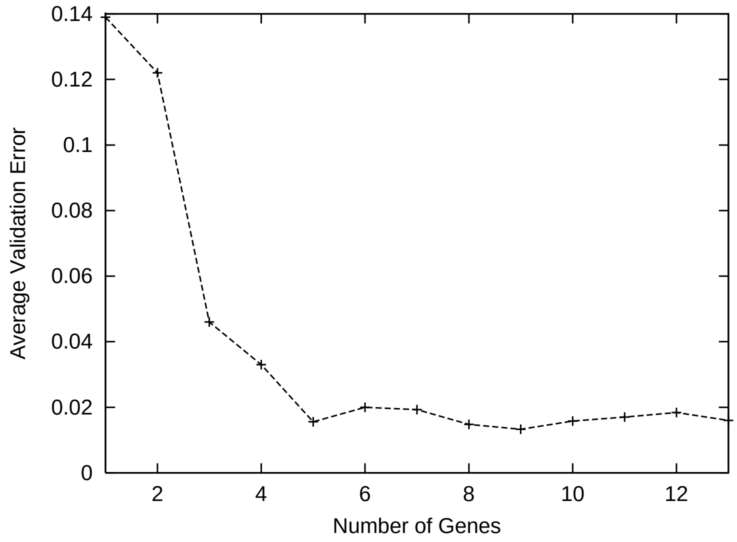

Figure 1: Variation of validation error on the leukaemia dataset, shown as the number of genes used in the feature set. The average validation error reached the minimum when nine genes were used. The list of these nine genes is given in Table 1.

Table 1: 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

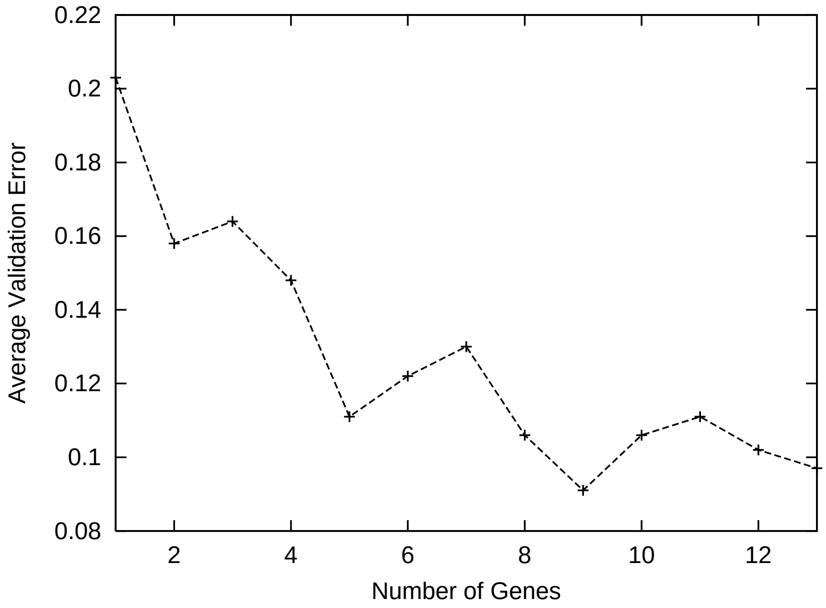

Figure 2: Variation of validation error on the ovarian cancer dataset, shown as the number of genes used in the feature set. The average validation error reached the minimum when nine genes, listed in Table 2, were used.

Table 2: 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

Table 3: Average test error for different datasets

Dataset	Average Test Error
Leukaemia	0.069
Ovarian Cancer	0.167

References

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.

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 genes overexpressed in ovarian carcinomas. *Gene, 238*, 375–385.

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, defined in the main paper . 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.

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

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);
   };
}
```