Chapter 9

CLASSIFYING MICROARRAY DATA USING SUPPORT VECTOR MACHINES

Sayan Mukherjee

PostDoctoral Fellow: MIT/Whitehead Institute for Genome Research and Center for Biological and Computational Learning at MIT,

e-mail: sayan@mit.edu

1. INTRODUCTION

Over the last few years the routine use of DNA microarrays has made possible the creation of large data sets of molecular information characterizing complex biological systems. Molecular classification approaches based on machine learning algorithms applied to DNA microarray data have been shown to have statistical and clinical relevance for a variety of tumor types: Leukemia (Golub et al., 1999), Lymphoma (Shipp et al., 2001), Brain cancer (Pomeroy et al., 2002), Lung cancer (Bhattacharjee et al., 2001) and the classification of multiple primary tumors (Ramaswamy et al., 2001).

One particular machine learning algorithm, *Support Vector Machines* (SVMs), has shown promise in a variety of biological classification tasks, including gene expression microarrays (Brown et al., 2000, Mukherjee et al., 1999). SVMs are powerful classification systems based on regularization techniques with excellent performance in many practical classification problems (Vapnik, 1998, Evgeniou et al., 2000).

This chapter serves as an introduction to the use of SVMs in analyzing DNA microarray data. An informal theoretical motivation of SVMs both from a geometric and algorithmic perspective followed by an application to leukemia classification is described in Section 2. The problem of gene selection is described in Section 3. Section 4 states some results on a variety of cancer morphology and treatment outcome problems. Multiclass classification is described in Section 5. Section 6 lists software sources and

rules of thumb that a user may find helpful. The chapter concludes with a brief discussion of SVMs with respect to some other algorithms.

2. SUPPORT VECTOR MACHINES

In binary microarray classification problems, we are given l experiments $\{(\boldsymbol{x}_1, y_1),\ldots, (\boldsymbol{x}_l, y_l)\}$. This is called the *training set*, where $\boldsymbol{x}_i$ is a vector corresponding to the expression measurements of the i^{th} experiment or sample (this vector has n components, each component is the expression measurement for a gene or EST) and y_i is a binary class label, which will be ± 1. We want to estimate a multivariate function from the training set that will accurately label a new sample, that is $f(\boldsymbol{x}_{new}) = y_{new}$. This problem of learning a classification boundary given positive and negative examples is a particular case of the problem of approximating a multivariate function from sparse data. The problem of approximating a function from sparse data is *ill-posed*. Regularization is a classical approach to making the problem *well-posed* (Tikhonov and Arsenin, 1977).

A problem is well-posed if it has a solution, the solution is unique, and the solution is stable with respect to perturbations of the data (small perturbations of the data result in small perturbations of the solution). This last property is very important in the domain of analyzing microarray data where the number of variables, genes and ESTs measured, is much larger than the number of samples. In statistics when the number of variables is much larger that the number of samples one is said to be facing the "curse of dimensionality" and the function estimated may very accurately fit the samples in the training set but be very inaccurate in assigning the label of a new sample, this is referred to as over-fitting. The imposition of stability on the solution mitigates the above problem by making sure that the function is smooth so new samples similar to those in the training set will be labeled similarly, this is often referred to as the *blessing of smoothness*.

2.1 Mathematical background of SVMs

We start with a geometric intuition of SVMs and then give a more general mathematical formulation.

2.1.1 A geometrical interpretation

A geometric interpretation of the SVM illustrates how this idea of smoothness or stability gives rise to a geometric quantity called the margin which is a measure of how well separated the two classes can be. We start by assuming that the classification function is linear

$$f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} = \sum_{i=1}^{n} w_i x_i \qquad (9.1)$$

where x_i and w_i are the i^{th} elements of the vectors $\boldsymbol{x}$ and $\boldsymbol{w}$, respectively. The operation $\boldsymbol{w} \cdot \boldsymbol{x}$ is called a *dot product*. The label of a new point $\boldsymbol{x}_{new}$ is the sign of the above function, $y_{new} = \text{sign}\,[\, f(\boldsymbol{x}_{new}) \,]$. The classification boundary, all values of $\boldsymbol{x}$ for which $f(x) = 0$, is a *hyperplane*[1] defined by its normal vector $\boldsymbol{w}$ (see Figure 9.1).

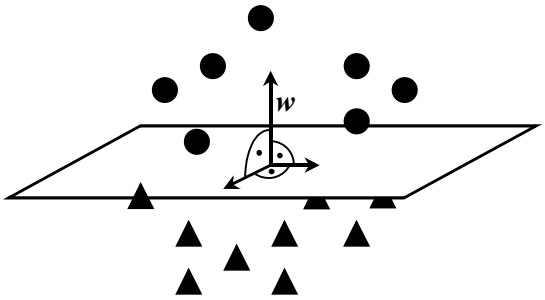


Figure 9.1. The hyperplane separating two classes. The circles and the triangles designate the members of the two classes. The normal vector to the hyperplane is the vector $\boldsymbol{w}$.

Assume we have points from two classes that can be separated by a hyperplane and $\boldsymbol{x}$ is the closest data point to the hyperplane, define $\boldsymbol{x}_0$ to be the closest point on the hyperplane to $\boldsymbol{x}$. This is the closest point to $\boldsymbol{x}$ that satisfies $\boldsymbol{w} \cdot \boldsymbol{x} = 0$ (see Figure 9.2). We then have the following two equations:

$$\boldsymbol{w} \cdot \boldsymbol{x} = k \quad \text{for some } k, \text{ and}$$
$$\boldsymbol{w} \cdot \boldsymbol{x} = 0.$$

Subtracting these two equations, we obtain $\boldsymbol{w} \cdot (\boldsymbol{x} - \boldsymbol{x}_0) = k$.
Dividing by the norm of $\boldsymbol{w}$ (the norm of $\boldsymbol{w}$ is the length of the vector $\boldsymbol{w}$), we obtain:

$$\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \cdot (\boldsymbol{x} - \boldsymbol{x}_0) = \frac{k}{\|\boldsymbol{w}\|}$$

[1] A *hyperplane* is the extension of the two or three dimensional concepts of lines and planes to higher dimensions. Note, that in an n-dimensional space, a hyperplane is an $n-1$ dimensional object in the same way that a plane is a two dimensional object in a three dimensional space. This is why the *normal* or *perpendicular* to the hyperplane is always a vector.

where $\|\boldsymbol{w}\| = \sqrt{\sum_{i=1}^{n} w_i^2}$. Noting that $\boldsymbol{w} / \|\boldsymbol{w}\|$ is a unit vector (a vector of length 1), and the vector $\boldsymbol{x} - \boldsymbol{x}_0$ is parallel to $\boldsymbol{w}$, we conclude that

$$\|\boldsymbol{x} - \boldsymbol{x}_0\| = \frac{|k|}{\|\boldsymbol{w}\|}.$$

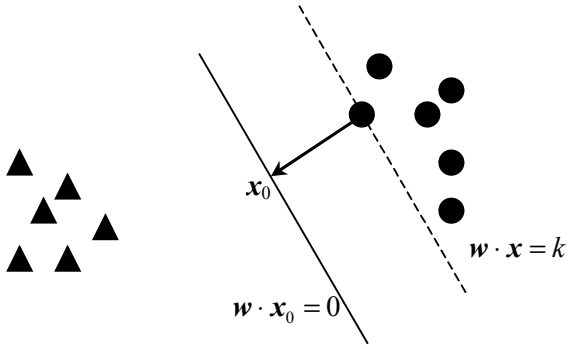


Figure 9.2. The black line is the hyperplane separating the triangles from the circles defined by its normal vector $\boldsymbol{w}$. The circle on the dashed line is the point $\boldsymbol{x}$ closest to the hyperplane, and $\boldsymbol{x}_0$ the closest point to $\boldsymbol{x}$ *o*n the hyperplane.

Our objective is to maximize the distance between the hyperplane and the closest point, with the constraint that the points from the two classes fall on opposite sides of the hyperplane. The following optimization problem satisfies the objective:

$$\max_{\boldsymbol{w}} \min_{x_i} \frac{y_i(\boldsymbol{w} \cdot x_i)}{\|\boldsymbol{w}\|} \quad \text{subject to} \quad y_i(\boldsymbol{w} \cdot \boldsymbol{x}) > 0 \text{ for all } \boldsymbol{x}_i \tag{9.2}$$

Note that $y(\boldsymbol{w} \cdot \boldsymbol{x}) = |k|$ when the point $\boldsymbol{x}$ is the circle closest to the hyperplane in Figure 9.2.

For technical reasons, the optimization problem stated above is not easy to solve. One difficulty is that if we find a solution $\boldsymbol{w}$, then $c\,\boldsymbol{w}$ for any positive constant c is also a solution. This is because we have not fixed a scale or unit to the problem. So without any loss of generality, we will require that for the point $\boldsymbol{x}_i$ closest to the hyperplane $k = 1$. This fixes a scale and unit to the problem and results in a guarantee that $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i) \geq 1$ for all $\boldsymbol{x}_i$. All other points are measured with respect to the closest point, which is distance 1 from the optimal hyperplane. Therefore, we may equivalently solve the problem

$$\max_{\boldsymbol{w}} \min_{\boldsymbol{x}_i} \frac{y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i)}{\|\boldsymbol{w}\|} \quad \text{subject to} \quad y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i) \geq 1 \tag{9.3}$$

An equivalent, but simpler problem (Vapnik, 1998) is

$$\min_{\boldsymbol{w}} \tfrac{1}{2}\|\boldsymbol{w}\|^2 \quad \text{subject to} \quad y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i) \geq 1 \tag{9.4}$$

Note that so far, we have considered only hyperplanes that pass through the origin. In many applications, this restriction is unnecessary, and the standard separable (i.e. the hyperplane can separate the two classes) SVM problem is written as

$$\min_{\boldsymbol{w},b} \tfrac{1}{2}\|\boldsymbol{w}\|^2 \quad \text{subject to} \quad y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 \tag{9.5}$$

where b is a free threshold parameter that translates the optimal hyperplane relative to the origin. The distance from the hyperplane to the closest points of the two classes is called the margin and is $1/\|\boldsymbol{w}\|^2$. SVMs find the hyperplane that maximize the margin. Figure 9.3 illustrates the advantage of a *large margin*.

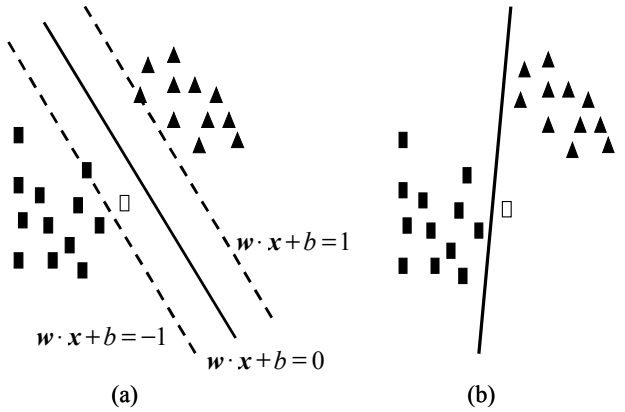


Figure 9.3.(a) The maximum margin hyperplane separating two classes. The solid black line is the hyperplane ($\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$). The two dashed lines are those for the points in the two classes closest to the hyperplane ($\boldsymbol{w} \cdot \boldsymbol{x} + b = \pm 1$). A new point, the blank rectangle, is classified correctly in (a). Note, the larger the margin the greater the deviation allowed or margin for error. (b) A non-maximum margin hyperplane separating the two classes. Note, that the same new point is now classified incorrectly. There is less margin for error.

In practice, data sets are often not linearly separable. To deal with this situation, we add slack variables that allow us to violate our original distance constraints. The problem becomes now:

$$\min_{\boldsymbol{w},b,\xi} \quad \tfrac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_i \xi_i \quad \text{subject to} \quad y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) \geq 1 - \xi_i \tag{9.6}$$

where $\xi_i \geq 0$ for all i. This new formulation trades off the two goals of finding a hyperplane with large margin (minimizing $\|\boldsymbol{w}\|$), and finding a hyperplane that separates the data well (minimizing the ξ_i). The parameter C controls this trade-off. This formulation is called the soft margin SVM. The parameter C controls this trade-off. It is no longer simple to interpret the final solution of the SVM problem geometrically. Figure 9.4 illustrates the *soft margin* SVM.

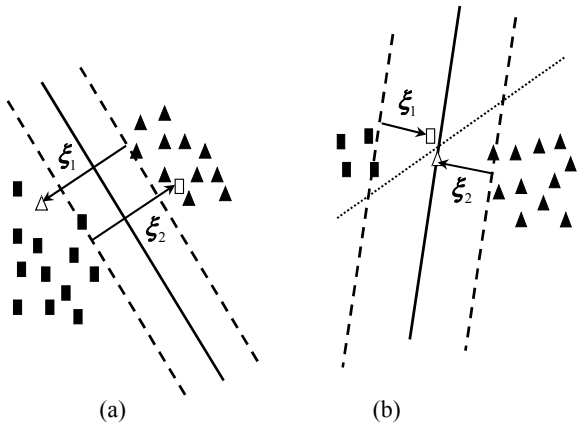


Figure 9.4.(a) The data points are not linearly separable. The solid black line is the SVM solution. The white triangle and the white rectangle are misclassified. The slack variables designate the distance of these points from the dashed lines for the corresponding classes. (b) The classes are separable. The dotted line is the solution when the tradeoff parameter C is very large (e.g., infinite), and this gives us the *maximum margin classifier* for the separable case. If the tradeoff parameter is small, then one allows errors (given by the two slack variables), but one gets a much larger margin.

SVMs can also be used to construct nonlinear separating surfaces. The basic idea here is to nonlinearly map the data to a feature space of high or possibly infinite dimensions, $\boldsymbol{x} \rightarrow \phi(\boldsymbol{x})$. We then apply the linear SVM algorithm in this feature space. A linear separating hyperplane in the feature space corresponds to a nonlinear surface in the original space. We can now rewrite Equation 9.6 using the data points mapped into the feature space, and we obtain Equation 9.7.

$$\min_{\boldsymbol{w},b,\xi} \tfrac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_i \xi_i \tag{9.7}$$

$\xi \geq 0$ for all i, where the vector $\boldsymbol{w}$ has the same dimensionality as the feature space and can be thought of as the normal of a hyperplane in the feature space. The solution to the above optimization problem has the form

$$f(\boldsymbol{x}) = \boldsymbol{w} \cdot \phi(\boldsymbol{x}) + b = \sum_{i=1}^{l} c_i \phi(x_i) \cdot \phi(\boldsymbol{x}) + b \tag{9.8}$$

since the normal to the hyperplane can be written as a linear combination of the training points in the feature space,

$$\boldsymbol{w} = \sum_{i=1}^{l} c_i \phi(x_i) .$$

For both the optimization problem (Equation 9.7) and the solution (Equation 9.8), the dot product of two points in the feature spaces needs to be computed. This dot product can be computed without explicitly mapping the points into feature space by a *kernel function*, which can be defined as the dot product for two points in the feature space:

$$K(x_i, x_j) \equiv \phi(x_i) \cdot \phi(x_j) \tag{9.9}$$

So our solution to the optimization problem has now the form:

$$f(\boldsymbol{x}) = \sum_{i=1}^{l} c_i K(x_i, x_j) + b \tag{9.10}$$

Most of the coefficients c_i will be zero; only the coefficients of the points closest to the maximum margin hyperplane in the feature space will have nonzero coefficients. These points are called the *support vectors*. Figure 9.5 illustrates a nonlinear decision boundary and the idea of support vectors.

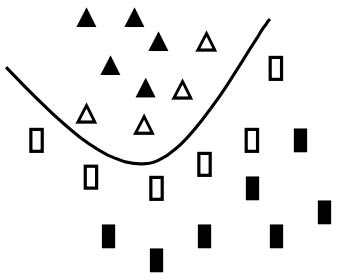


Figure 9.5 The curve is the nonlinear decision boundary given by the SVM. The white triangles and the white rectangles are the support vectors. Only these points contribute in defining the nonlinear decision boundary.

The following example illustrates the connection between the mapping into a feature space and the kernel function. Assume that we measure the expression levels of two genes, TrkC and SonicHedghog (SH). For each sample, we have the expression vector $\boldsymbol{x} = (\boldsymbol{x}_{SH}, \boldsymbol{x}_{TrkC})$. We use the following mapping $\phi(\boldsymbol{x})$:

$$\phi : \boldsymbol{x} \to \left\{ \boldsymbol{x}_{SH}^2, \boldsymbol{x}_{TrkC}^2, \sqrt{2}\boldsymbol{x}_{SH}\boldsymbol{x}_{TrkC}, \boldsymbol{x}_{SH}, \boldsymbol{x}_{TrkC}, 1 \right\}$$

If we have two samples $\boldsymbol{x}$ and $\boldsymbol{z}$, then we obtain:

$$\begin{aligned} K(\boldsymbol{x}, \boldsymbol{z}) \equiv \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{z}) &= (\boldsymbol{x} \cdot \boldsymbol{z} + 1)^2 \\ &= \boldsymbol{x}_{SH}^2 \boldsymbol{z}_{SH}^2 + \boldsymbol{x}_{TrkC}^2 \boldsymbol{z}_{TrkC}^2 + 2\boldsymbol{x}_{SH}\boldsymbol{x}_{TrkC}\boldsymbol{z}_{SH}\boldsymbol{z}_{TrkC} + \\ &\quad \boldsymbol{x}_{SH}\boldsymbol{z}_{SH} + \boldsymbol{x}_{TrkC}\boldsymbol{z}_{TrkC} + 1 \end{aligned}$$

which is called a *second order polynomial kernel*. Note, that this kernel uses information about both expression levels of individual genes and also expression levels of pairs of genes. This can be interpreted as a model that incorporates co-regulation information. Assume that we measure the expression levels of 7,000 genes. The feature space that the second order polynomial would map into would have approximately 50 million elements, so it is advantageous that one does not have to explicitly construct this map.

The following two kernels, the polynomial and Gaussian kernel, are commonly used:

$$K(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x} \cdot \boldsymbol{z} + 1)^p \text{ and } K(\boldsymbol{x}, \boldsymbol{z}) = \exp(-\|\boldsymbol{x} - \boldsymbol{z}\| / 2\sigma^2).$$

2.1.2 A theoretical motivation for SVMs and Regularization

SVMs can also be formulated as an algorithm that finds a function f that minimizes the following functional[2] (note that this functional is basically Equation 9.7, rewritten in a more general way):

$$\min_f \quad \frac{1}{l}\sum_{i=1}^{l}\left(1 - y_i f\left(x_i\right)\right)_+ + \frac{1}{C}\|f\|_K^2 \tag{9.11}$$

where the first term, the *hinge loss function*, is used to measure the error between our estimate $f(x_i)$ and y_i, and $(a)_+ = \min(a, 0)$. The expression $\|f\|_K^2$ is a measure of smoothness, and the margin is $1 / \|f\|_K^2$. The variable l is the number of training examples, and $1 / C$ is the regularization parameter that trades off between smoothness and errors. The first term ensures that the estimated function has a small error on the training samples and the second term ensures that this function is also smooth. This functional is a particular

[2] A *functional* is a function that maps other functions to real numbers.

instance of the *Tikhonov regularization principle* (Tikhonov and Arsenin, 1977).

Given a data set $S = \{(\boldsymbol{x}_1, y_1),\ldots, (\boldsymbol{x}_l, y_l)\}$, the SVM algorithm takes this training set and finds a function f_S. The error of this function on the training set is called the *empirical error*, and we can measure it using Equation 9.12:

$$I_{emp}[f_S] = \frac{1}{l}\sum_{i=1}^{l}\left(1 - y_i f(x_i)\right)_+ \tag{9.12}$$

However, what we really care about is how accurate we will be given a new data sample, which is $(1 - \boldsymbol{y}_{new} f(\boldsymbol{x}_{new}))_+$. In general, we want to weight this error by the probability of drawing the sample $(\boldsymbol{x}_{new}, \boldsymbol{y}_{new})$, and average this over all possible data samples. This weighted measure is called the *expected* or *generalization error*:

$$I_{exp}[f_S] = \int \left(1 - yf(\boldsymbol{x})\right) p(y \mid \boldsymbol{x}) d\boldsymbol{x} dy \tag{9.13}$$

where $p(y \mid \boldsymbol{x})$ is the unknown conditional probability function for $\boldsymbol{x}$ to belong to class y.

For algorithms that implement Tikhonov regularization, one can say with high probability (Bousquet and Elisseeff, 2002, Mukherjee et al., 2002):

$$\left|I_{emp}[f_s] - I_{exp}[f_s]\right| \le \Phi\left(l, \|f_S\|\right) \tag{9.14}$$

where the function Φ decreases as $\|f\|$ decreases (or margin increases) and l increases. This tells us that if our error rate on the training set is low and the margin is large, then the error rate on average for a new sample will also be low. This is a theoretical motivation for using regularization algorithms such as SVMs. Note, that for the number of samples typically seen in microarray expression problems, plugging in values of l and f_S into Φ will not yield numbers small enough to serve as practical error bars.

2.2 An application of SVMs

One of the first cancer classification studies was discriminating acute myeloid leukemia (AML) from acute lymphoblastic leukemia (ALL) (Golub et al., 1999). In this problem, a total of 38 training samples belong to the two classes, 27 ALL cases vs. 11 AML cases. The accuracy of the trained classifier was assessed using 35 test samples. The expression levels of 7,129 genes and ESTs were given for each sample. A linear SVM trained on this data accurately classified 34 of 35 test samples (see Figure 9.6.)

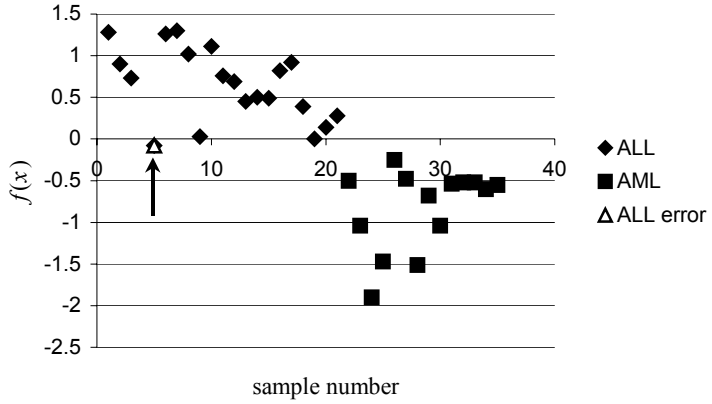


Figure 9.6 The signed distance, $f(\boldsymbol{x})$, from the optimal separating hyperplane for the test samples. The diamonds are the correctly labeled ALL samples. The squares indicate the correctly labeled AML samples. The triangle marks the misclassified ALL case (see arrow).

From Figure 9.6 an intuitive argument can be formulated: the larger the absolute value of the signed distance, $|f(\boldsymbol{x})|$, the more confident we can be in the classification. There exist approaches to convert the real-valued $f(\boldsymbol{x})$ into confidence values $p(y = \pm 1 \mid \boldsymbol{x})$ (Mukherjee et al., 1999, Platt, 1999). Thus, we can classify samples that have a confidence value larger than a particular threshold, whereas the classification of samples with confidence values below this threshold will be rejected. We applied this methodology in (Mukherjee et al., 1999) to the above problem, with a positive threshold of $|f(\boldsymbol{x})| = 0.1$ and a negative threshold of $|f(\boldsymbol{x})| = -0.2$.

A simple rule of thumb value to use as threshold is $|f(\boldsymbol{x})| = 1$. Function values greater than this threshold are considered *high confidence*. This value is in general too large to use for rejections.

Polynomial or Gaussian kernels did not increase the accuracy of the classifier (Mukherjee et al., 1999). However, when "important" genes were removed, the polynomial classifier did improve performance. This suggests that correlation information between genes can be helpful. Therefore, we removed 10 to 1,000 of the most "informative" genes from the test and training sets according to the signal-to-noise (S2N) criterion (see Section 3.1). We then applied linear and polynomial SVMs on this data set and reported the error rates (Mukherjee et al., 1999). Although the differences are not statistically significant, it is suggestive that until the removal of 300 genes, the polynomial kernel improves the performance, which suggests that modeling correlations between genes helps in the classification task.

3. GENE SELECTION

It is important to know which genes are most relevant to the binary classification task and select these genes for a variety of reasons: removing noisy or irrelevant genes might improve the performance of the classifier, a candidate list of important genes can be used to further understand the biology of the disease and design further experiments, and a clinical device recording on the order of tens of genes is much more economical and practical than one requiring thousands of genes.

The gene selection problem is an example of what is called *feature selection* in machine learning (see Chapter 6 of this volume). In the context of classification, feature selection methods fall into two categories *filter methods* and *wrapper methods*. Filter methods select features according to criteria that are independent of those criteria that the classifier optimizes. On the other hand, wrapper methods use the same or similar criteria as the classifier. We will discuss two feature selection approaches: *signal-to-noise* (S2N, also known as *P-metric*) (Golub et al., 1999, Slonim et al., 2000;), and *recursive feature elimination* (RFE) (Guyon et al., 2002). The first approach is a filter method, and the second approach is a wrapper method.

3.1 Signal-to-noise (S2N)

For each gene j, we compute the following statistic:

$$S(j) = \frac{\mu_+(j) - \mu_-(j)}{\sigma_+(j) + \sigma_-(j)}, \tag{9.15}$$

where $\mu_+(j)$ and $\mu_-(j)$ are the means of the classes +1 and –1 for the j^{th} gene. Similarly, $\sigma_+(j)$ and $\sigma_-(j)$ are the standard deviations for the two classes for the j^{th} gene. Genes that give the most positive values are most correlated with class +1, and genes that give the most negative values are most correlated with class –1. One selects the most positive $m/2$ genes and the most negative $m/2$ genes, and then uses this reduced dataset for classification. The question of estimating m is addressed in Section 3.3.

3.2 Recursive Feature Elimination (RFE)

The method recursively removes features based upon the absolute magnitude of the hyperplane elements. We first outline the approach for linear SVMs. Given microarray data with n genes per sample, the SVM outputs the normal to the hyperplane, $\boldsymbol{w}$, which is a vector with n components, each corresponding to the expression of a particular gene. Loosely speaking, assuming that the expression values of each gene have similar ranges, the

absolute magnitude of each element in $\boldsymbol{w}$ determines its importance in classifying a sample, since the following equation holds:

$$f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b = \sum_{i=1}^{n} w_i x_i + b$$

The idea behind RFE is to eliminate elements of $\boldsymbol{w}$ that have small magnitude, since they do not contribute much in the classification function. The SVM is trained with all genes; then we compute the following statistic for each gene:

$$S(j) = |w_j| \tag{9.16}$$

Where w_j the value of the j^{th} element of $\boldsymbol{w}$. We then sort S from largest to smallest value and we remove the genes corresponding to the indices that fall in the bottom 10% of the sorted list S. The SVM is retrained on this smaller gene expression set, and the procedure is repeated until a desired number of genes, m, is obtained. When a nonlinear SVM is used, the idea is to remove those features that affect the margin the least, since maximizing the margin is the objective of the SVM (Papageorgiou et al., 1998). The nonlinear SVM has a solution of the following form:

$$f(\boldsymbol{x}) = \sum_{i=1}^{l} c_i K(\boldsymbol{x}, x_i) + b \tag{9.17}$$

Let M denote the margin. Then we obtain Equation 9.18:

$$\frac{1}{M} = \sum_{p,r=1}^{l} c_p c_r K(x_p, x_r) \tag{9.18}$$

So for each gene j, we compute to which extent the margin changes using the following statistic:

$$S(j) = \left| \frac{\partial(1/M)}{\partial x_j} \right| \tag{9.19}$$

where x_j is the j^{th} element of a vector of expression values $\boldsymbol{x}$. We then sort S from the largest to the smallest value, and we remove the genes corresponding to the indices that fall in the bottom 10% of the sorted list S. The SVM is retrained and the procedure is repeated just as in the linear case.

3.3 How Many Genes To Use?

A basic question that arises for all feature selection algorithms is how many genes the classifier should use. One approach to answer this question is using hypothesis and permutation testing (Golub et al., 1999). The null hypothesis is that the S2N or RFE statistic for each gene computed on the training set comes from the same distribution as that for a *random data set*. A random data set is the training set with its labels randomly permuted.

In detail, the permutation test procedure for the S2N or RFE statistic is as follows:

(1) Generate the statistic for all genes using the actual class label and sort the genes accordingly.
(2) Generate 100 or more random permutations of the class labels. For each case of randomized class labels, generate the statistics for all genes and sort the genes accordingly.
(3) Build a histogram from the randomly permuted statistics using various numbers of genes. We call this number *k*. For each value of *k*, determine different percentiles (1%, 5%, 50% etc.) of the corresponding histogram.
(4) Compare the actual signal-to-noise scores with the different significance levels obtained for the histograms of permuted class labels for each value of *k* (see Figure 9.7 for an illustration).

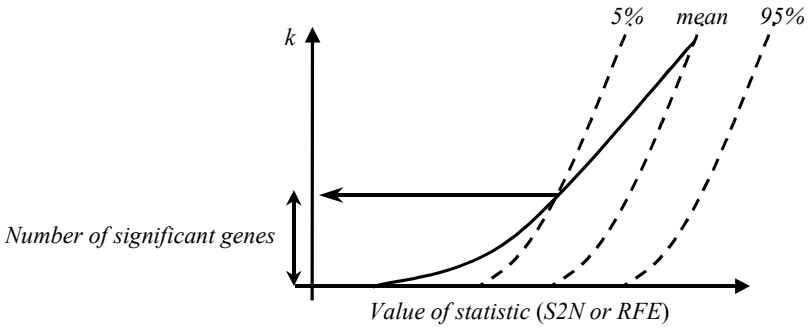


Figure 9.7. The solid curve is the S2N or RFE statistic rank ordered computed on the training set. The three dashed lines are the 5th, 50th, and 95th percentiles of the same rank ordered statistic as computed from the random data. The number of statistical genes is designated as the value of *k*, where the solid curve crosses the 5th percentile curve.

4. ERROR RATES FOR MORPHOLOGY AND TREATMENT OUTCOME PREDICTION

We examined the error rate for SVMs and two other algorithms, *Weighted Voting Average* (WVA), and *k-nearest neighbors* (*k*NN) on seven binary

cancer classification problems. The problems are as follows: discriminating acute myeloid leukemia (AML) from acute lymphoblastic leukemia (ALL), and discriminating B-cells from T-cells for acute lymphoblastic leukemia (Golub et al., 1999), discriminating follicular (FSC) lymphoma from diffuse large cell lymphoma (DLCL) and discriminating high risk from low risk lymphoma patients (Shipp et al., 2001), discriminating glioblastomas (GL) from meduloblastomas (MD), and discriminating high risk from low risk patients with medulloblastoma (Pomeroy et al., 2002). See Table 9.1 for number of samples in each class for the data sets.

Error rates for all data sets except for AML vs. ALL were measured using leave-one-out cross validation[3]. For AML vs. ALL, the test/train split described in (Golub et al., 1999) was used. S2N was used for feature selection for WVA and *k*NN algorithms. The SVM used the radius-margin ratio as a gene selection methodology. The errors for both outcome prediction problems were much larger than those for the morphology prediction problems. The errors are reported in Table 9.2.

The number of genes used in each classification task for each algorithm was determined using cross-validation. In general, SVMs required more genes in the classification tasks.

Table 9.1. The number of samples in the various data sets.

Data set	*# of Samples*	*Class* –1	*Class* +1
AML vs. ALL (*train*)	38	27 ALL	11 AML
AML vs. ALL (*test*)	35	21 ALL	14 AML
B-cell vs. T-cell	23	15 B-cell	8 T-cell
FSC vs. DLCL	77	19 FSC	58 DLCL
GL vs. MD	41	14 GL	27 MD
Lymphoma outcome	58	20 Low risk	38 High risk
Medullo outcome	50	38 Low risk	12 High risk

[3] See Chapter 7 of this volume for more details on *leave-one-out cross-validation*.

Table 9.2. Absolute number of errors for the various data sets.

		Errors			
Data set	*Method*	*Total*	*Class* 1	*Class* –1	# *of genes used*
AML vs. ALL	WVA	2	1	1	50
	*k*NN	3	1	2	10
	SVM	0	0	0	40
B-cell vs. T-cell	WVA	0	0	0	9
	*k*NN	0	0	0	10
	SVM	0	0	0	10
FSC vs. DLCL	WVA	6	1	5	30
	*k*NN	3	1	2	200
	SVM	4	2	2	250
GL vs. MD	WVA	1	1	0	3
	*k*NN	0	0	0	5
	SVM	1	1	0	100
Lymphoma outcome	WVA	15	5	10	12
	KNN	15	8	7	15
	SVM	13	3	10	100
Medullo outcome	WVA	13	6	7	6
	*k*NN	10	6	4	5
	SVM	7	6	1	50

5. MULTICLASS CLASSIFICATION

Ramaswamy et al. investigated whether the diagnosis of multiple adult malignancies could be achieved purely by molecular classification, using DNA microarray gene expression profiles (Ramaswamy et al., 2001). In total, 218 tumor samples, spanning 14 common tumor types, and 90 normal tissue samples were subjected to oligonucleotide microarray gene expression analysis. These tumor types/localizations are: breast (BR), prostate (PR), lung (LU), colorectal (CO), lymphoma (L), bladder (BL), melanoma (ME), uterus (UT), leukemia (LE), renal (RE), pancreas (PA), ovary (OV), mesothelioma (MS), and central nervous system (CNS). The expression levels of 16,063 genes and ESTs were used to train and evaluate the accuracy of a multiclass classifier based on the SVM algorithm. The overall classification accuracy was 78%, far exceeding the accuracy of random classification (9%). Table 9.3 shows the number of training and test samples per tumor class.

Table 9.3. The number of samples in the training set (train) and the test set (test).

	BR	*PR*	*LU*	*CO*	*L*	*BL*	*ME*	*UT*	*LE*	*RE*	*PA*	*OV*	*MS*	*CNS*
train	8	8	8	8	16	8	8	8	24	8	8	8	8	16
test	3	2	3	5	6	3	2	2	6	3	3	3	3	4

Multiple class prediction is intrinsically more difficult than binary prediction because the classification algorithm has to learn to construct a greater number of separation boundaries or relations. In binary classification, an algorithm can "carve out" the appropriate decision boundary for only one of the classes; the other class is simply the complement. In multiclass classification problems, each class has to be defined explicitly. A multiclass problem can be decomposed into a set of binary problems, and then combined to make a final multiclass prediction.

The basic idea behind combining binary classifiers is to decompose the multiclass problem into a set of easier and more accessible binary problems. The main advantage in this divide-and-conquer strategy is that any binary classification algorithm can be used. Besides choosing a decomposition scheme and a classifier for the binary decompositions, one also needs to devise a strategy for combining the binary classifiers and providing a final prediction. The problem of combining binary classifiers has been studied in the computer science literature (Hastie and Tibshirani, 1998, Allwein et al., 2000, Guruswami and Sahai, 1999) from a theoretical and empirical perspective. However, the literature is inconclusive, and the best method for combining binary classifiers for any particular problem is open.

Standard modern approaches for combining binary classifiers can be stated in terms of what is called *output coding* (Dietterich and Bakiri, 1991). The basic idea behind output coding is the following: given k classifiers trained on various partitions of the classes, a new example is mapped into an output vector. Each element in the output vector is the output from one of the k classifiers, and a *codebook* is then used to map from this vector to the class label (see Figure 9.7). For example, given three classes, the first classifier may be trained to discriminate classes 1 and 2 from 3, the second classifier is trained to discriminate classes 2 and 3 from 1, and the third classifier is trained to discriminate classes 1 and 3 from 2.

Two common examples of output coding are the *one-versus-all* (OVA) and *all-pairs* (AP) approaches. In the OVA approach, given k classes, k independent classifiers are constructed where the i^{th} classifier is trained to separate samples belonging to class i from all others. The codebook is a diagonal matrix, and the final prediction is based on the classifier that produces the strongest confidence:

$$class = \arg\max_{i=1..k} f_i \qquad (9.20)^4$$

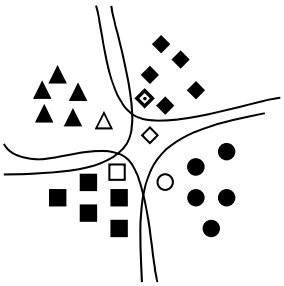
where f_i is the signed confidence measure of the i^{th} classifier (see Figure 9.8). In the all-pairs approach, $\frac{1}{2}\, k(k-1)$ classifiers are constructed, with each classifier trained to discriminate between a class pair i and j. This can be

[4] The procedure arg max$f(x)$ simply selects the value or argument of x that maximizes $f(x)$.

thought of as a $k \times k$ matrix, where the ij^{th} entry corresponds to a classifier that discriminates between classes i and j. The codebook, in this case, is used to simply sum the entries of each row and select the row for which this sum is maximal:

$$class = \arg\max_{i=1..k} \left[\sum_{j=1}^{k} f_{ij} \right] \tag{9.21}$$

where f_{ij} is the signed confidence measure for the ij^{th} classifier.



class	*Classifier* $C(\blacktriangle)$	$C(\blacklozenge)$	$C(\bullet)$	$C(\blacksquare)$	
▲	+1	0	0	0	
◆	0	+1	0	0	
●	0	0	+1	0	
■	0	0	0	+1	
test △	**0.80**	0.20	0.01	0.20	▲
◇	0.50	**0.60**	0.30	0.10	◆
○	0.10	0.10	**0.90**	0.20	●
□	0.30	0.10	0.40	**0.80**	■
◈	0.60	**0.70**	0.10	0.10	◆

predicted

(a) (b)

Figure 9.8. OVA classification. (a) Four binary classifiers are trained. The curves designate the non-linear decision boundaries. The first classifier discriminates between the ▲-class and all other classes; the second classifier discriminates between the ◆-class and all other classes; the third classifier discriminates between the ●-class and all other classes, and the fourth classifier discriminates between the ■-class and all other classes. The following five cases are the test cases: △, ◇, ○, □, and ◈. (b) The codebook for OVA classification is represented in the upper part of the table. The numbers in the table are the ideal outputs of the classifiers for the cases of the four classes. For example, the classifier $C(\blacktriangle)$, which has learnt to discriminate ▲-cases from all other cases, ideally outputs +1 for ▲-cases and 0 otherwise. The lower part of the table shows the outputs of the classifiers for the test cases. The classifier that outputs the highest number determines the class. For example, $C(\bullet)$ yields 0.90 for the case ○; consequently, ○ is classified as member of class ●.

Intuitively, there is a tradeoff between the OVA and AP approaches. The discrimination surfaces that need to be learned in the all-pairs approach are, in general, more natural and, theoretically, should be more accurate. However, with fewer training examples, the empirical surface constructed may be less precise. The actual performance of each of these schemes, or others such as random codebooks, in combination with different classification algorithms is problem dependent. The OVA approach gave the best results on this dataset, and these results are reported in Table 9.4. The train/test split in the table is the same as that in (Ramaswamy et al., 2001). The error rate on the training set was measured using leave-one-out cross validation. A sample classification was considered *high confidence* if $\max f_i \geq 1$, and low confidence otherwise.

Table 9.4. Error rates for the multiclass data set.

				Confidence			
				High		*Low*	
Data set	*Validation Method*	*Samples*	*Total accuracy*	*Fraction*	*Accuracy*	*Fraction*	*Accuracy*
Train	cross-validation	144	78%	80%	90%	20%	28%
Test	train/test	54	78%	78%	83%	22%	58%

6. SOFTWARE SOURCES AND RULES OF THUMB

The SVM experiments described in this chapter were performed using a modified version of the SvmFu package, which can be downloaded from http://www.ai.mit.edu/projects/cbcl/). Another available SVM package is SVMTorch (http://www.idiap.ch/learning/SVMTorch.html). SvmFu has some advantages in that it has both multiclass and leave-one-out cross-validation built in. SVMTorch is the only software of the above that does SVM regression as well as classification.

The following are some rules of thumb when using SVMs:

(1) *Normalizing your data*: in general it is a good idea to rescale your data such that all kernel values fall between –100 and 100; a simple way to do this is by normalizing all entries of the microarray such that they fall between $-10(n)^{½}$ and $+10(n)^{½}$, where n is the number of expression values per sample;

(2) *Choosing the regularization parameter C*: given the above normalization, the regularization parameter usually does not have much effect, so set it somewhere between 1–100;

(3) *Choosing the kernel*: for microarray applications, a linear kernel is usually sufficient; you can use polynomial kernels if you want to examine correlations between genes, but it will in general not greatly improve the performance; if the linear kernel does not give good performance, it is worth trying the Gaussian kernel;

(4) *Choosing the variance of the Gaussian kernel*: set σ such that the average distance between two training points

$$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp(||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2 / 2\sigma^2) = 1/ l,$$

where l is the number of samples.

(5) *Multiclass problems*: for a multiclass problem, use the OVA and AP decompositions; in general, more complicated coding systems do not help. When there are very few (5-10) training samples per class, OVA will in general give the best results.

7. DISCUSSION

The theoretical advantage of SVMs is that the idea of margin or stability mitigates the problem of overfitting the training data. This is crucial in the DNA microarray domain, which is characterized by thousands of genes/variables and very few samples. Unlike most other algorithms (for example, *k*NN or WVA), the SVM performs very well even without feature selection, using all 7,000-16,000 expression values. The multiclass example illustrates that the SVM is especially helpful when there are very few training samples. Other algorithms such as *k*NN or WVA are made stable by removing the noisy genes and reducing the number of features to 10-100. One can loosely conclude that the *curse of dimensionality* is overcome in SVMs using the *blessing of smoothness*, and simply by reducing the dimensionality in the other algorithms. A practical result of this is that for some classification problems, the SVM will tend to use more genes to make a classification that *k*NN or WVA.

ACKNOWLEDGEMENTS

I would like to acknowledge my colleagues at Cancer Genomics Group at the Whitehead/MIT Center for Genome Research and at the Center for Biological and Computational Learning at MIT. The author is supported by the Office of Science (BER), US Department of Energy, Grant No. DE-FG02-01ER63185.

REFERENCES

Allwein E.L., Schapire R.E., Singer Y. (2000). Reducing multiclass to binary: a unifying approach for margin classifiers. Journal of Machine Learning Research 1:113-141.

Bhattacharjee A., Richards W.G, Staunton J., Li C., Monti S., Vasa P., Ladd C., Beheshti J., Bueno R., Gillette M., Loda M., Weber G., Mark E.F., Lander E.S., Wong W., Johnson B.E., Golub T.R., Sugarbaker D.J., Meyerson M. (2001). Classification of human lung carcinomas by mRNA expression profiling reveals distinct adenocarcinoma subclasses. Proc. Natl. Acad. Sci. USA 98:13790–13795.

Bousquet O. and Elisseeff A. (2002). Stability and Generalization. Journal of Machine Learning Research 2, 499-526.

Brown M.P.S., Grundy W.N., Lin D., Cristianini N., Sugnet C., Furey T.S., Ares M., Jr., Haussler D. (2000). Knowledge-based analysis of microarray gene expression data using support vector machines. Proc. Natl. Acad. Sci. USA 97(1):262-267.

Dietterich T.G. and Bakiri G. (1991). Error-correcting output codes: A general method for improving multiclass inductive learning programs. Proc. of the Ninth National Conference on Artificial Intelligence, AAAI Press, 572-577.

Evgeniou T., Pontil M., Poggio T. (2000). Regularization networks and support vector machines. Advances in Computational Mathematics 13:1-50.

Golub T.R., Slonim D.K., Tamayo P., Huard C., Gaasenbeek M., Mesirov J.P., Coller H., Loh M.L., Downing J.R., Caligiuri M.A., Bloomfield C.D., Lander E.S. (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. Science 286(5439):531-7.

Guruswami V. and Sahai A. 1999. Multiclass learning, boosting and error-correcting codes. Proc. of the Twelfth Annual Conference on Computational Learning Theory, ACM Press, 145-155.

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

Hastie T.J. and Tibshirani R.J. (1998). Classification by pairwise coupling. In Jordan M.I., Kearnsa M.J., Solla S.A., eds., Advances in Neural Information Processing Systems, volume 10, MIT Press.

Mukherjee S., Rifkin R., Poggio T. (2002). Regression and Classification with Regularization. Proc. of Nonlinear Estimation and Classification, MSRI, Berkeley, Springer-Verlag.

Mukherjee S., Tamayo P., Slonim D., Verri A., Golub T., Mesirov J.P., Poggio T. (2000). Support vector machine classification of microarray data. Technical Report, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Papageorgiou C., Evgeniou T., Poggio T. (1998). A trainable pedestrian detection system. Intelligent Vehicles, pp. 241-246.

Platt J.C. 1999. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood methods. Advances in Large Margin Classifiers, MIT Press.

Pomeroy S.L., Tamayo P., Gaasenbeek M., Sturla L.M., Angelo M., McLaughlin M.E., Kim J.Y., Goumnerova L.C., Black P.M., Lau C., Allen J.C., Zagzag D., Olson J., Curran T., Wetmore C., Biegel J.A., Poggio T., Mukherjee S., Rifkin R., Califano A., Stolovitzky G., Louis D.N., Mesirov J.P., Lander E.S., and Golub T.R. (2002). Prediction of central nervous system embryonal tumour outcome based on gene expression. Nature, 415(24):436-442. (and supplementary information).

Ramaswamy S., Tamayo P., Rifkin R., Mukherjee S., Yeang C.H., Angelo M. Ladd C., Reich M., Latulippe E., Mesirov J.P., Poggio T., Gerald W., Loda M., Lander E.S., Golub. T.R.: Multiclass cancer diagnosis using tumor gene expression signatures, Proc. Natl. Acad. Sci. USA. 98(26):15149-15154.

Shipp MA, Ross KN, Tamayo P, Weng AP, Kutok JL, Aguiar RC, Gaasenbeek M, Angelo M, Reich M, Pinkus GS, Ray TS, Koval MA, Last KW, Norton A, Lister TA, Mesirov J, Neuberg DS, Lander ES, Aster JC, Golub TR. (2002). Diffuse large B-cell lymphoma outcome prediction by gene expression profiling and supervised machine learning. Nat Med 8(1):68-74.

Slonim D., Tamayo P., Mesirov J., Golub T., Lander E. (2000). Class prediction and discovery using gene expression data. In Proc. of the 4th Annual International Conference on Computational Molecular Biology (RECOMB), Universal Academy Press, pp. 263-272, Tokyo, Japan.

Tikhonov A.N. and Arsenin V.Y. (1977). Solutions of Ill-posed Problems. W.H. Winston, Washington D.C.

Vapnik V.N. (1998). Statistical Learning Theory. John Wiley & Sons, New York.