# LAGO: A Computationally Efficient Approach for Statistical Detection

Mu Zhu, Wanhua Su and Hugh A. Chipman*

July 18, 2005

## Abstract

We study a general class of statistical detection problems where the underlying objective is to detect items belonging to a rare class from a very large database. We propose a computationally efficient method to achieve this goal. Our method consists of two steps. In the first step, we estimate the density function of the rare class alone with an adaptive bandwidth kernel density estimator. The adaptive choice of the bandwidth is inspired by the ancient Chinese board game known today as Go. In the second step, we adjust this density locally depending on the density of the background class nearby. We show that the amount of adjustment needed in the second step is approximately equal to the adaptive bandwidth from the first step, which gives us additional computational savings. We name the resulting method LAGO for "locally adjusted Go-kernel density estimator." We then apply LAGO to a real drug discovery data set and compare its performance with a number of existing and popular methods.

**Key Words**: Adaptive bandwidth kernel density estimator; Average precision; Drug discovery; Nearest neighbor; Order statistics; Radial basis function (RBF) network; Support vector machine (SVM).

*Mu Zhu is Assistant Professor and Wanhua Su is Doctoral Student, Department of Statistics and Actuarial Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada. Hugh A. Chipman is Associate Professor, Department of Mathematics and Statistics, Acadia University, Wolfville, NS B4P 2R6, Canada.

## 1 The Statistical Detection Problem

Suppose we have a large collection of items, $\mathcal{C}$, of which only a fraction $\pi$ ($\pi \ll 1$) is relevant to us. We are interested in computational tools to help us identify and single out these items. The reason an item is considered relevant depends on the context of a specific problem. For example, for fraud detection (e.g., Bolton and Hand 2002) the relevant items are individual transactions that are fraudulent; for drug discovery the relevant items are chemical compounds that are active against a specific biological target (such as the HIV virus); and so on.

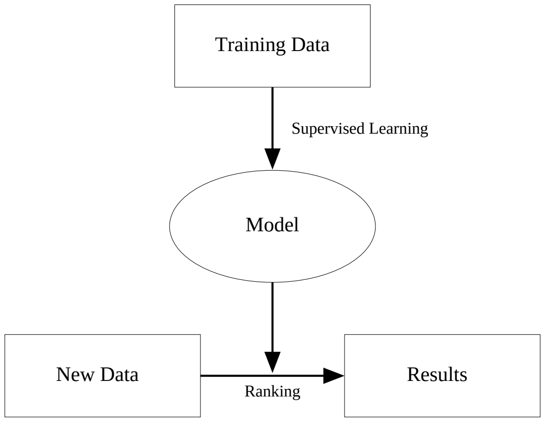

Figure 1: *Illustration of the typical modelling and prediction process.*

Typically, we have a training data set $\{(y_i, \mathbf{x}_i)\}_{i=1}^{n}$, where $\mathbf{x}_i \in \mathbb{R}^d$ is a vector of predictors, $y_i = 1$ if observation $i$ is relevant and $y_i = 0$ otherwise. Supervised learning methods (e.g., classification trees, neural networks) are used to build a predictive model using the training data. The model is then used to screen a large number of new cases; it often produces a relevance score or an estimated probability for each of these new cases. The top-ranked cases can then be passed onto further stages of investigation. A summary of this process is provided in Figure 1.

If it is decided that the top 50 cases should be investigated further, we shall say that these 50 cases are the ones "detected" by the model or the algorithm, although, strictly speaking, the algorithm really does not detect these cases *per se*; it merely ranks them as being more likely than others to be what we want.

The data structure and the types of supervised learning methods often used here are similar to those encountered in a standard two-class classification problem. However, the underlying objective is very different. In particular, automatic classification is of little interest. This is because further investigation of the top-ranked candidates is almost always necessary. For example, in the drug discovery application, one never starts to manufacture a drug without further testing a compound; in the fraud detection application, one seldom terminates a credit card account without confirming the suspected fraud. Therefore, we are most interested in producing an effective *ranking* of all the candidates so that any further investigation (often very expensive) is least likely to be carried out in vain.

## 1.1 Performance Evaluation

Bolton and Hand (2002) emphasized that, because relevant cases are quite rare for these detection problems, "simple misclassification rate cannot be used as a performance measure" for evaluating different methods. For example, if only 0.1% of the transactions are fraudulent, then a model that simply classifies everything as non-fraudulent will have a misclassification rate of only 0.001, but it is clearly not a useful model.

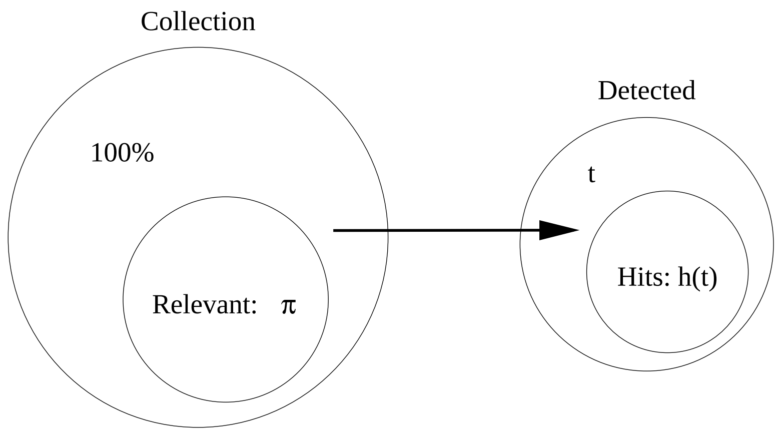

Figure 2: *Illustration of a typical detection operation. A small fraction $\pi$ of the entire collection $\mathcal{C}$ is relevant. An algorithm detects a fraction $t$ from $\mathcal{C}$ and $h(t)$ is later confirmed to be relevant.*

Suppose an algorithm detects (in the sense made explicit above) a fraction $t \in [0, 1]$ from $\mathcal{C}$ and $h(t) \in [0, t]$ is later confirmed to be relevant; these are often called "hits" (as opposed to "misses"). Figure 2 provides a schematic illustration; Table 1 is the corresponding confusion matrix.

Table 1: *The Confusion Matrix.*

| | Relevant | Irrelevant | Total |
|---|---|---|---|
| Detected | $h(t)$ | $t - h(t)$ | $t$ |
| Undetected | $\pi - h(t)$ | $(1 - t) - (\pi - h(t))$ | $1 - t$ |
| Total | $\pi$ | $1 - \pi$ | 1 |

The function $h(t)$ is called a "hit curve" (e.g., Wang 2005, Chapter 2). Figure 3 shows some typical hit curves for a hypothetical case where only 5% of all cases are of interest. The dotted curve on the top, $h_P(t)$, is an ideal curve produced by a perfect algorithm; every item detected is an actual hit until all potential hits (5% in total) are exhausted. The diagonal dotted curve, $h_R(t)$, is the expected hits under random selection. The solid (blue) and dashed (red) curves, $h_A(t)$ and $h_B(t)$, are those of two typical detection algorithms.

The hit curve $h(t)$ tells us a lot about the performance of a model or an algorithm. For example, if $h_A(t) > h_B(t)$ for every $t$, then algorithm A is unambiguously superior to algorithm B. If an algorithm consistently produces hit curves that rise up very quickly as $t$ increases, then it can often be regarded as a strong algorithm. In particular, a perfect algorithm would have a hit curve that

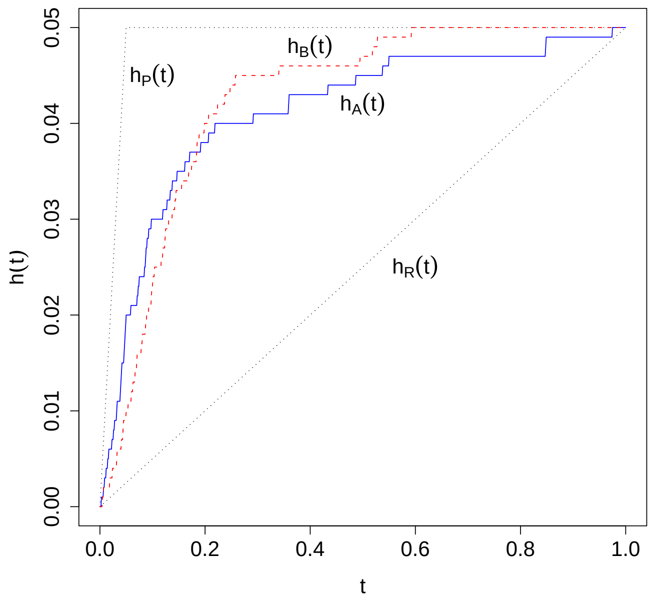

Figure 3: *Illustration of some hit curves for a situation with 5% interesting cases. The curve $h_P(t)$ is an ideal curve produced by a perfect algorithm; $h_R(t)$ corresponds to the case of random detection; $h_A(t)$ and $h_B(t)$ are hit curves produced by typical detection algorithms.*

rises with a maximal slope of one until $t = \pi$, i.e., everything detected is a hit until all possible hits are exhausted; afterwards the curve necessarily stays flat (see the curve $h_P(t)$ in Figure 3).

Table 1 also makes it clear that the hit curve $h(t)$ is related to the well-known receiver operating characteristic (ROC) curve (e.g., Cantor and Kattan 2000) but the two are not the same. While both have $h(t)$ on their vertical axes, the horizontal axis for the ROC curve is $t - h(t)$ instead of $t$. In medicine, the area under the ROC curve is often used to measure the effectiveness of diagnostic tests.

### 1.2 Average Precision

While the hit curve provides an effective visual performance assessment, modelling algorithms also require quantitative performance measures. Often an algorithm will have a number of tuning or regularization parameters that must be chosen empirically with a semi-automated procedure such as cross-validation. For example, for the $K$ nearest-neighbor algorithm we must select the appropriate number of neighbors, $K$. This is done by calculating a cross-validated version of the performance criterion for a number of different $K$s and selecting the one that gives the best cross-validated performance. For this purpose, we clearly prefer a numeric performance measure, not an entire curve!

In this article, we shall use a numeric performance measure that is most widely used in the information retrieval community known as the *average precision* (AP; see, e.g., Deerwester *et al.* 1990; Dumais 1991; Peng *et al.* 2003). Appendix A gives more details on how the average precision is defined and calculated. As far as we are aware of, the precise connection between the AP and the area under the ROC curve is not known. In fact, trying to establish this connection is part of our ongoing research program. For most readers whose primary interests are in the modelling aspect of our work, it suffices to know that the AP is a numeric summary of the hit curve $h(t)$, and that if method A has a higher AP than method B then method A can be regarded as the better method.

## 2 A General Paradigm for Efficient Modelling

We first propose a general paradigm that is efficient for constructing computational models for the statistical detection problem. Recall that we are most interested in the ranking of all potential candidates. Given a vector of predictors $\mathbf{x}$, the posterior probability is arguably a good ranking function, i.e., items with a high probability of being relevant should be ranked first. By Bayes' Theorem, the posterior probability can be expressed as

$$g(\mathbf{x}) \equiv P(y = 1|\mathbf{x}) = \frac{\pi_1 p_1(\mathbf{x})}{\pi_1 p_1(\mathbf{x}) + \pi_0 p_0(\mathbf{x})}. \tag{1}$$

Here $\pi_0$ and $\pi_1$ are prior probabilities; $p_0$ and $p_1$ are conditional density functions of $\mathbf{x}$ given $y = 0$ and $y = 1$, respectively. In order to rank items from a new data set $\{\mathbf{x}_i; i = 1, 2, ..., N\}$, it is clear that a very accurate estimate of $g(\mathbf{x}_i)$ is not necessary as long as the estimated function ranks these observations in the correct order. As far as ranking is concerned, all monotonic transformations of $g$ are clearly equivalent. Therefore it suffices to concentrate on the ratio function

$$f(\mathbf{x}) = \frac{p_1(\mathbf{x})}{p_0(\mathbf{x})} \tag{2}$$

since the function $g$ is of the form

$$g(\mathbf{x}) = \frac{af(\mathbf{x})}{af(\mathbf{x}) + 1}$$

for some constant $a$ not depending on $\mathbf{x}$, which is clearly a monotonic transformation of $f$.

A key feature for the statistical detection problems is that most observations are from the background class (class 0; $C_0$) and only a small number of observations are from the class of interest (class 1; $C_1$). This important feature allows us to make the following assumptions:

A1. For all practical purposes the density function $p_1(\mathbf{x})$ can be assumed to have bounded local support, possibly over a number of disconnected regions, $\mathcal{S}_\gamma \subset \mathbb{R}^d$, $\gamma = 1, 2, ..., \Gamma$, in which case the support of $p_1$ can be written as

$$\mathcal{S} = \bigcup_{\gamma=1}^{\Gamma} \mathcal{S}_\gamma \subset \mathbb{R}^d.$$

A2. For every observation $\mathbf{x}_i \in C_1$, there are at least a certain number of observations, say $m$, from $C_0$ in its immediate local neighborhood; moreover, the density function $p_0(\mathbf{x})$ in that neighborhood can be assumed to be relatively flat in comparison with $p_1(\mathbf{x})$. We shall be more specific about what we mean by the "immediate local neighborhood" in Section 3.2.

Regions outside $\mathcal{S}$ are clearly not of interest since $f(\mathbf{x})$ would be zero there and can, therefore, be completely ignored; see equation (2). Assumptions A1 and A2 above then imply that, in order to estimate the function $f$, we can simply estimate $p_1$ and adjust it locally according to the density $p_0$ nearby. Notice that this strategy would give us a significant computational advantage since the number of observations belonging to class 1 is typically very moderate even for very large data sets. We shall exploit this general principle below to construct a computationally efficient method.

## 3 LAGO: Univariate Model Construction

To apply the general principles outlined in the previous section, we first assume the predictor $x \in \mathbb{R}$ is just a scalar. We will generalize this to $\mathbb{R}^d$ for $d > 1$ in Section 4.

### 3.1 Step 1: Estimating $p_1$

The first step is to estimate $p_1$. To do so, we use an adaptive bandwidth kernel estimator:

$$\hat{p}_1(x) = \frac{1}{n_1} \sum_{y_i=1} \mathcal{K}\left(x; x_i, r_i\right), \tag{3}$$

where $n_1$ is the number of observations belonging to class 1 and $\mathcal{K}(x; x_i, r_i)$ denotes a kernel function centered at $x_i$ with bandwidth $r_i$. For each $x_i \in C_1$, we choose $r_i$ adaptively to be the average distance between $x_i$ and its $K$-nearest neighbors from $C_0$, i.e.,

$$r_i = \frac{1}{K} \sum_{w_j \in N(x_i, K)} |x_i - w_j| \tag{4}$$

where the notation $N(x_i, K)$ is used to refer to the set that contains the $K$-nearest class-0 neighbors of $x_i$. The number $K$ here is a tuning parameter that must be selected empirically (e.g., via cross-validation).

The original inspiration of this particular bandwidth choice comes from the ancient Chinese game of Go (Figure 4). In this game, two players take turns to place black or white stones onto a $19 \times 19$ board and try to conquer as many territories as possible. At any given stage during the game, any serious player must evaluate the relative strength of his or her position on the board as well as the strategic value of a number of possible moves. Ignoring the various rules of the game which are irrelevant for this article, the basic principles behind such evaluation can be roughly summarized as follows: any given stone on the board exerts a certain amount of influence over its immediate neighborhood region; the amount of influence is inversely related to how close the opponent's stones are lying around it.

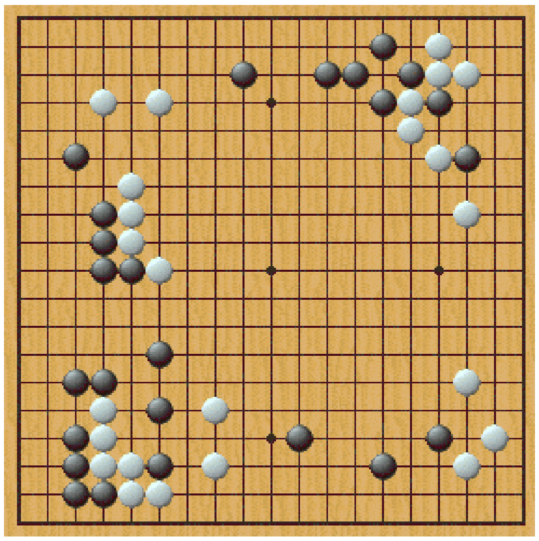

Figure 4: *The ancient Chinese game of Go is a game in which each player tries to claim as many territories as possible on the board. Image taken from http://go.arad.ro/Introducere.html.*

For kernel density estimation, the situation is analogous: each observation $x_i \in C_1$ exerts a certain amount of influence over its immediate neighborhood region; the extent of this influence is controlled by the bandwidth parameter. Following the basic evaluation principle from the game of Go, we allow each $x_i$ to have a different bandwidth parameter $r_i$ depending on the distance between $x_i$ and its neighboring observations from $C_0$. We shall refer to this special kernel density estimator as the "Go-kernel density estimator."

## 3.2 Step 2: Local Adjustment of $p_1$

The next step is to adjust the density estimate $\hat{p}_1$ locally according the density of class 0 nearby. Here we view (3) as a mixture and adjust each mixture component (centered at $x_i$) accordingly. Based on (2), we should estimate $p_0$ locally around every $x_i \in C_1$, say $p_0(x; x_i)$, and divide it into

$\mathcal{K}(x; x_i, r_i)$. Assumption A2 implies that we can simply estimate $p_0(x; x_i)$ locally as a constant, say $c_i$. This means our estimate of $f$ should be of the form:

$$\hat{f}(x) = \frac{1}{n_1} \sum_{y_i=1} \frac{\mathcal{K}(x; x_i, r_i)}{c_i} \tag{5}$$

for appropriately estimated constants $c_i$, $i = 1, 2, ..., n_1$. Below, we present an argument that makes the estimation of $c_i$ unnecessary in practice, which gives us an additional computational advantage.

Intuitively, if $p_0(x; x_i) \approx c_i$ is roughly a constant nearby, then $c_i$ should on the average be inversely proportional to the adaptive bandwidth $r_i$ given in (4), i.e., if the density of class 0 nearby is low, we will expect the average distance between $x_i$ and its $K$-nearest neighbors from $C_0$ to be large, and *vice versa.*

Theorem 1 below (see Appendix B for a proof) makes this intuition more precise in an idealized situation where, instead of saying $p_0(x; x_i) \approx c_i$, we shall explicitly assume that, for every $x_i \in C_1$, there exist i.i.d. observations $w_1, w_2, ..., w_m$ from $C_0$ that can be taken to be uniformly distributed on the interval $[x_i - 1/2c_i, x_i + 1/2c_i]$. Such an idealization does not make a significant difference in practice but will allow us to provide a proof that is accessible to a much wider audience.

**Theorem 1** *Let $x_0$ be a fixed observation from class 1. Suppose $w_1, w_2, ..., w_m$ are i.i.d. observations from class 0 that are uniformly distributed around $x_0$, say on the interval $[x_0 - 1/2c_0, x_0 + 1/2c_0]$. If $r_0$ is the average distance between $x_0$ and its $K$ nearest neighbors from class 0 ($K < m$), then we have*

$$E(r_0) = \frac{K+1}{4(m+1)c_0}. \qquad \blacksquare$$

For the idealized situation, the statement in Theorem 1 is exact. In practice, we can assume (see A2) there are at least $m$ observations from $C_0$ distributed *approximately* uniformly around every $x_i \in C_1$. For $K < m$, then, Theorem 1 above allows us to conclude that $r_i$ will be *approximately* proportional to $1/c_i$. Therefore, having already computed $r_i$ in the previous step, we can avoid having to estimate $c_i$ separately for each $x_i$ and simply express our estimate of $f$ as

$$\hat{f}(x) = \frac{1}{n_1} \sum_{y_i=1} r_i \; \mathcal{K}\left(x; x_i, r_i\right). \tag{6}$$

We shall use the acronym LAGO to refer to the resulting estimator $\hat{f}$ as the "locally adjusted Go-kernel density estimator."

## 3.3 Discussions

It is important to realize that, while A1 and A2 are sufficient conditions for LAGO to work, they are by no means necessary conditions. Approximating functions locally as a constant is not uncommon. For example, in regression trees (CART; Breiman *et al.* 1984), one approximates the regression function $f(\mathbf{x})$ with a piecewise constant surface even if one knows *a priori* that the function is not truly piecewise constant. Therefore, assumption A2 merely provides us with the motivation for estimating $p_0$ locally as a constant in Step 2 (Section 3.2), but one may proceed with such an approximation regardless of whether one believes in A2 or not.

## 4 LAGO: The General Multivariate Model

We now extend the methodology presented above to $\mathbb{R}^d$ for $d > 1$. For multivariate predictor $\mathbf{x} \in \mathbb{R}^d$, we simply apply the naïve Bayes principle (see, e.g., Hastie *et al.* 2001, Section 6.6.3), i.e., we model each dimension independently and multiply the marginal models together:

1. For every training observation in class 1, $\mathbf{x}_i \in C_1$, find its $K$-nearest class-0 neighbors in $\mathbb{R}^d$ and compute a specific bandwidth vector $\mathbf{r}_i = (r_{i1}, r_{i2}, ..., r_{id})^T$, where $r_{ij}$ is the average distance between $\mathbf{x}_i$ and its $K$-nearest class-0 neighbors in the $j$th dimension.
2. For every new observation $\mathbf{x} = (x_1, x_2, ..., x_d)^T$ where a prediction is required, score and rank $\mathbf{x}$ according to:

$$f(\mathbf{x}) = \frac{1}{n_1} \sum_{y_i=1} \left\{ \prod_{j=1}^{d} r_{ij} \; \mathcal{K}\left(x_j; x_{ij}, r_{ij}\right) \right\}. \tag{7}$$

Generally speaking, using the naïve Bayes principle is the same as using ellipsoidal (e.g., Gaussian) or rectangular (e.g., uniform) kernel functions whose principal axes are aligned with the coordinate system. If initial exploratory analysis should indicate the existence of fairly strong correlations among the predictors, making the naïve Bayes principle inappropriate, one could transform the data with methods such as principal component analysis and apply LAGO in the transformed space, as is often done in density estimation (see, e.g., Silverman 1986, p. 77–78).

### 4.1 Choice of Kernel

In practice we must choose a kernel function $\mathcal{K}(u)$. Common choices include the uniform kernel, the triangular kernel and the Gaussian kernel (see Figure 5). Our experience has indicated that if the uniform kernel is used, many observations can be tied in terms of their rank score (7); this does *not* produce an effective ranking. Significant improvements can, therefore, be obtained by choosing $\mathcal{K}(u)$ to be Gaussian or triangular. In addition, there is usually no significant difference between the Gaussian and the triangular kernel (see Section 6 below), making the triangular kernel the most attractive choice due to its compact local support.

### 4.2 A Radial Basis Function (RBF) Network

The LAGO estimator can be viewed as a radial basis function (RBF) network. An RBF network (e.g., Hastie *et al.* 2001, Section 6.7) is a model for functional estimation that estimates an arbitrary function using a number of kernel functions as basis functions. In general, an RBF network has the form:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \beta_i \; \mathcal{K}(\mathbf{x}; \boldsymbol{\mu}_i, \mathbf{r}_i), \tag{8}$$

where $\mathcal{K}(\mathbf{x}; \boldsymbol{\mu}, \mathbf{r})$ is a kernel function centered at location $\boldsymbol{\mu}$ with radius (or bandwidth) vector $\mathbf{r} = (r_1, r_2, ..., r_d)^T$. Clearly, in order to construct an RBF network, one must make specific decisions

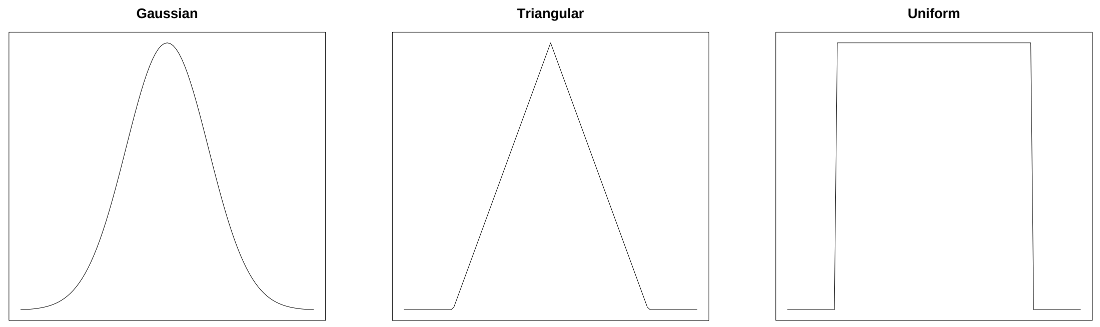

Figure 5: *Some commonly used kernel functions: Gaussian (left), triangular (middle), and uniform (right).*

about the centers $\boldsymbol{\mu}_i$ and the radii $\mathbf{r}_i$ for $i = 1, 2, ..., n$ or specify an algorithm to pick them. For $d$-dimensional problems ($d > 1$), it is not uncommon to specify the kernel function $\mathcal{K}$ as a product of $d$ univariate kernel functions as in (7) (e.g., Hastie *et al.* 2001).

LAGO can be viewed as a highly specialized RBF network. In particular, we have specifically chosen to put basis functions at and only at all the class-1 observations in the training data, each with a rather specific radius vector defined separately in each dimension by equation (4). Once the centers and the radii are specified, the coefficients $\beta_i$ ($i = 1, 2, ..., n$) are often estimated empirically. In constructing the LAGO model, we have, instead, specified these coefficients implicitly in the model as well, i.e.,

$$\beta_i = \frac{1}{n_1} \prod_{j=1}^{d} r_{ij}.$$

## 4.3 A More General Parameterization

To the extent that LAGO can be viewed as an RBF network, it is possible to consider a more direct and slightly more general way of parameterizing such a model. Start with the adaptive bandwidth kernel density estimator for $p_1$ described in Section 3.1. Figure 6 illustrates the effect of the density $p_0$ on the ranking function $f$ that is of interest. In this simplified and hypothetical illustration, the density $p_1$ has two components. If the background class has a flat density function $p_0$, then the ranking function $f$ is clearly equivalent to $p_1$. Now suppose the background class has density function $p_0'$ instead. Figure 6 shows that this will in general have two effects on the ranking function $f$. For any given kernel component of $f$, if $p_0'$ is relatively low nearby, we should stretch its bandwidth (or radius) and lift its height. On the other hand, if $p_0'$ is relatively high nearby, we should dampen its bandwidth (or radius) and lower its height. We shall call the effect on the bandwidth (or radius) the $\alpha$-effect and the one on the height, the $\beta$-effect.

We can then parameterize these two effects explicitly. To construct an RBF network, suppose each component is a kernel function belonging to a location-scale family, i.e.,

$$\frac{1}{r_i} \mathcal{K} \left( \frac{x - x_i}{r_i} \right).$$

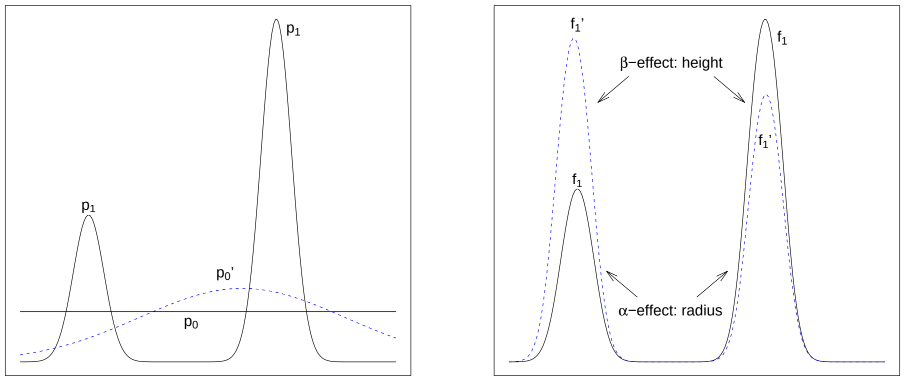

Figure 6: *Illustration. Left: Density functions $p_0$ and $p_1$. Right: The ratio function $f(x)$.*

If $r_i$ is chosen adaptively as in (4) and hence can be taken on average to be inversely proportional to the density $p_0$ in a local neighborhood around $x_i$ (Section 3.2), we can then explicitly parameterize the $\alpha$- and $\beta$-effects as follows:

$$r_i^{\beta'} \frac{1}{\alpha r_i} \mathcal{K}\left(\frac{x - x_i}{\alpha r_i}\right) \quad \propto \quad r_i^{\beta' - 1} \mathcal{K}\left(\frac{x - x_i}{\alpha r_i}\right) \quad \equiv \quad r_i^{\beta} \mathcal{K}\left(\frac{x - x_i}{\alpha r_i}\right),$$

where $\alpha$ and $\beta$ are tuning parameters to be selected empirically.

We have in effect argued in Section 3.2 that the parameter $\beta$ should, in theory, be set to 0 (or $\beta' = 1$). The parameter $\alpha$, on the other hand, can be beneficial in practice. Theorem 1 makes it clear that $r_i$ will generally increase with $K$. For relatively large $K$, we have found that we can usually obtain a model with very similar performance by setting $\alpha > 1$ and using a much smaller $K$; this is also quite intuitive. Therefore, by retaining the parameter $\alpha$ in the model, we can restrict ourselves to a much narrower range when selecting $K$ by cross-validation, which is computationally attractive. Hence, the final LAGO model that we will fit is of the form:

$$f(\mathbf{x}) = \frac{1}{n_1} \sum_{y_i = 1} \left\{ \prod_{j=1}^{d} r_{ij} \ \mathcal{K}\left(x_j; x_{ij}, \alpha r_{ij}\right) \right\}, \tag{9}$$

with altogether two tuning parameters, $K$ and $\alpha$.

## 5 Support Vector Machines (SVMs)

The support vector machine (e.g., Cristianini and Shawe-Taylor 2000) has, in recent years, become a very popular and successful supervised learning technique. Schölkopf *et al.* (1997) argued that the SVM can be viewed as an automatic way of constructing an RBF network. In this section, we give a very brief and highly condensed overview of what the SVM is, how it can be used for the statistical detection problem, and why it can be viewed as an RBF network and hence becomes directly relevant for us.

## 5.1 Overview of SVM

A hyperplane in $\mathbb{R}^d$ is characterized by

$$f(\mathbf{x}) = \boldsymbol{\beta}^T\mathbf{x} + \beta_0 = 0.$$

Given $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$ (two classes), a hyperplane is called a separating hyperplane if there exists $c > 0$ such that

$$y_i(\boldsymbol{\beta}^T\mathbf{x}_i + \beta_0) \geq c \quad \forall i. \tag{10}$$

Clearly, a hyperplane can be reparameterized by scaling, e.g.,

$$\boldsymbol{\beta}^T\mathbf{x} + \beta_0 = 0 \quad \text{is the same as} \quad s(\boldsymbol{\beta}^T\mathbf{x} + \beta_0) = 0$$

for any scalar $s$. A separating hyperplane satisfying the condition

$$y_i(\boldsymbol{\beta}^T\mathbf{x}_i + \beta_0) \geq 1 \quad \forall i, \tag{11}$$

i.e., scaled so that $c = 1$ in (10), is sometimes called a *canonical* separating hyperplane (Cristianini and Shawe-Taylor 2000).

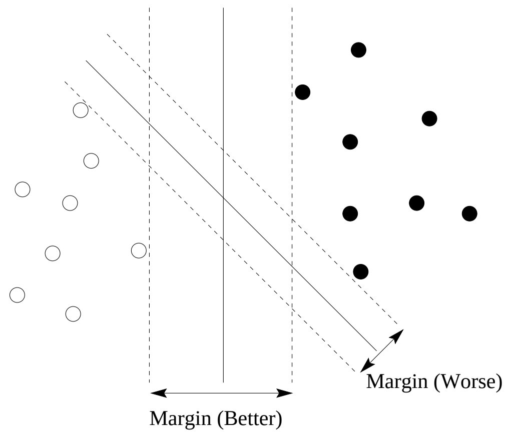

Figure 7: *Two separating hyperplanes, one with a larger margin than the other.*

As illustrated in Figure 7, there exist an infinite number of separating hyperplanes if two classes are perfectly separable. Given a training data set, any separating hyperplane can be characterized by its *margin*, a geometric concept illustrated in Figure 7. Strictly speaking, the margin is equal to $2\min\{y_i d_i\}$ where $d_i$ is the signed distance between observation $\mathbf{x}_i$ and the hyperplane; in particular, $d_i$ is equal to (see, e.g., Cristianini and Shawe-Taylor 2000; Hastie *et al.* 2001, Section 4.5)

$$d_i = \frac{1}{\|\boldsymbol{\beta}\|}(\boldsymbol{\beta}^T\mathbf{x}_i + \beta_0). \tag{12}$$

It is then easy to see from (11) that a canonical separating hyperplane has margin equal to $2\min\{y_i d_i\} = 2/\|\boldsymbol{\beta}\|$.

The support vector machine is based on the notion that the "best" canonical separating hyperplane to separate two classes is the one with the largest margin. Therefore, we are interested in solving the following convex optimization problem:

$$\min \quad \frac{1}{2}\|\boldsymbol{\beta}\|^2 + \gamma \sum_{i=1}^{n} \xi_i \tag{13}$$

$$\text{subject to} \quad \xi_i \geq 0 \quad \text{and} \quad y_i(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0) \geq 1 - \xi_i \quad \forall i. \tag{14}$$

Notice that in general we must introduce the extra variables $\xi_i$ to relax the separability condition (10) so that some observations are allowed to cross over to the other side of the hyperplane because we can't assume the two classes are always perfectly separable. The term $\gamma \sum \xi_i$ acts as a penalty to control the degree of such relaxation.

The optimization problem above has a quadratic objective function and linear inequality constraints, making it a standard quadratic programming problem. We will omit the details here (see Cristianini and Shawe-Taylor 2000) and simply state what the solution looks like. The solution for $\boldsymbol{\beta}$ is characterized by

$$\hat{\boldsymbol{\beta}} = \sum_{i \in SV} \hat{\alpha}_i y_i \mathbf{x}_i,$$

where $\hat{\alpha}_i \geq 0$ $(i = 1, 2, ..., n)$ are solutions to the dual optimization problem and SV, the set of "support vectors" with $\hat{\alpha}_i > 0$ strictly positive. This means the resulting hyperplane can be written as

$$\hat{f}(\mathbf{x}) = \hat{\boldsymbol{\beta}}^T \mathbf{x} + \hat{\beta}_0 = \sum_{i \in SV} \hat{\alpha}_i y_i \mathbf{x}_i^T \mathbf{x} + \hat{\beta}_0 = 0.$$

Up to this point it appears that the SVM is a strictly linear classifier. However, it is easy to replace the inner product $\mathbf{x}_i^T \mathbf{x}$ with a kernel function $\mathcal{K}(\mathbf{x}; \mathbf{x}_i)$ to get a nonlinear decision boundary:

$$\hat{f}(\mathbf{x}) = \sum_{i \in SV} \hat{\alpha}_i y_i \mathcal{K}(\mathbf{x}; \mathbf{x}_i) + \hat{\beta}_0 = 0. \tag{15}$$

The boundary is linear in the space of $h(\mathbf{x})$ where $h(\cdot)$ is such that $\mathcal{K}(\mathbf{u}; \mathbf{v}) = \langle h(\mathbf{u}), h(\mathbf{v}) \rangle$ is the inner product in the space of $h(\mathbf{x})$. Note that the mapping $h(\cdot)$ is not explicitly required.

To fit the SVM, two tuning parameters are required: the penalty parameter $\gamma$ and a bandwidth parameter $\sigma$ for the kernel $\mathcal{K}(\mathbf{x}; \mathbf{x}_i)$. The software we shall use is the function `svm` from a library called `e1071` in `R` (R Development Core Team 2004).

## 5.2 SVM for Detection Problems

To classify a new observation $\mathbf{x}$, all that is needed is whether $\hat{f}(\mathbf{x})$ is positive or negative. For detection problems, however, we must assign every new observation a rank score. A natural choice is to use the signed distance between $\mathbf{x}$ and the estimated decision boundary

$$d(\mathbf{x}) = \frac{1}{\|\hat{\boldsymbol{\beta}}\|}(\hat{\boldsymbol{\beta}}^T \mathbf{x} + \hat{\beta}_0).$$

For ranking purposes, all monotonic transformations of $d(\mathbf{x})$ are equivalent so it suffices to use just the inner product

$$\hat{\boldsymbol{\beta}}^T \mathbf{x} = \sum_{i \in SV} \hat{\alpha}_i y_i \mathbf{x}_i^T \mathbf{x}.$$

If the inner product $\mathbf{x}_i^T \mathbf{x}$ is replaced with a kernel function $\mathcal{K}(\mathbf{x}; \mathbf{x}_i)$, the final ranking function then becomes

$$F_{svm}(\mathbf{x}) = \sum_{i \in SV} \hat{\alpha}_i y_i \mathcal{K}(\mathbf{x}; \mathbf{x}_i). \tag{16}$$

## 5.3 SVM as an RBF network

Notice the similarity between (16) and (8). The ranking function produced by the SVM is a linear combination of a number of kernel functions centered at all the support points $\mathbf{x}_i \in SV$. Hence SVM can be viewed as an automatic way of constructing an RBF network since the SVs and the coefficients $\hat{\alpha}_i$ are automatically determined by the algorithm. Since LAGO can be viewed as a specially constructed RBF network (Section 4.2), an automatically constructed RBF network provides a good benchmark for us. In addition, it is also instructive to compare the centers chosen by the automatic SVM algorithm (the support vectors) with the ones we choose (all the class-1 observations); see Section 6.5.

## 5.4 SVM with Asymmetric Class Weights (ASVM)

It is also possible to weigh observations from the two classes differently in SVM, something that could potentially benefit these highly asymmetric detection problems. Suppose $w_0$ and $w_1$ are weights for $C_0$ and $C_1$, respectively. The convex optimization problem (13) becomes

$$\min \quad \frac{1}{2}\|\boldsymbol{\beta}\|^2 + \gamma_1 \sum_{y_i=1} \xi_i + \gamma_0 \sum_{y_i=0} \xi_i, \tag{17}$$

where $\gamma_1 = \gamma w_1$, $\gamma_0 = \gamma w_0$, and the constraints remain the same as (14). Solving (17) is the same as solving (13), except the penalty term involving $\gamma$ is split into two separate terms, one with $\gamma_1$ for $y_i = 1$ and another with $\gamma_0$ for $y_i = 0$. We shall call this the asymmetric support vector machine (ASVM), which can be fitted using the same function `svm` in `R` but with four tuning parameters — $\sigma$, $\gamma$, $w_0$ and $w_1$. Without loss of generality, we can fix $w_0 = 1$ and tune the remaining three parameters, $\sigma$, $\gamma$ and $w_1$, simultaneously.

# 6 An Application: Drug Discovery

In this section, we illustrate the use of LAGO with a real drug discovery data set. First, we give some background on the drug discovery problem.

## 6.1 Drug Discovery

Recent technological advances in high-throughput screening (HTS) have allowed chemists to screen large numbers of chemical compounds against specific biological targets such as the HIV virus. As a result of HTS, each compound can be classified as being active ($y = 1$) or inactive ($y = 0$) against the biological target under investigation. Here being active usually means that the compound is able to inhibit a certain protein and hence protect the cells against the target such as a virus. Active compounds are then passed onto further stages of drug development.

A pharmaceutical company usually has a large chemical library — a collection of different chemical compounds. In order to reduce cost, we can screen only part of the chemical library and use the results to build a computational model to screen the remaining compounds and identify the ones that would have tested active. In fact, these models can even be used to virtually screen compounds not yet in the company's chemical library. In order to build such a computational model, we need to have a number of predictors to characterize each compound, e.g., by its molecular structure. These are often computed with various algorithms from computational chemistry. A commonly used set of predictors is called BCUT numbers (Burden 1989; Lam *et al.* 2002); here we omit the details of exactly what the BCUT numbers are as these details are not directly relevant to us.

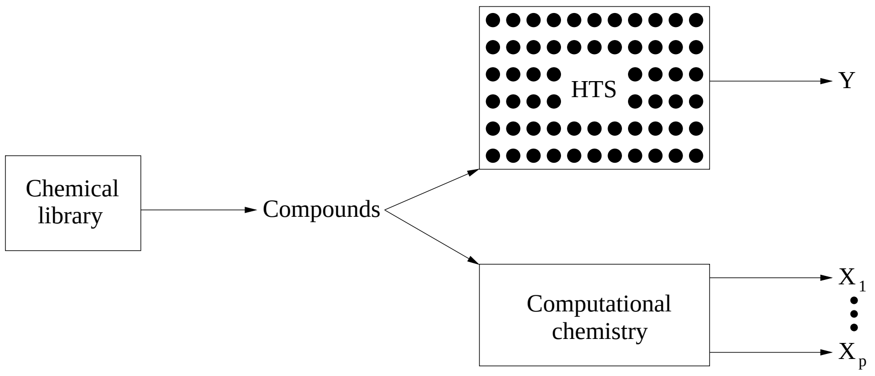

Figure 8: *Illustration of the high throughput screening process.*

Figure 8 summarizes the process described above. For any given biological target, active compounds are almost always quite rare. Therefore, building a computational model in order to identify these active compounds for drug discovery is exactly a statistical detection problem. Figure 8 also illustrates an important feature of statistical detection problems: the inexpensive availability of predictor variables for observations that have not yet had $y$ measured.

## 6.2 Data

The data set we use is the AIDS antiviral database from the National Cancer Institute (NCI). The October 1999 release of the NCI data, available at `http://dtp.nci.nih.gov/docs/aids/aids_data.html` were used in this analysis. The publicly available data set identifies each compound by name and provides the response value $y$. There are 29,812 chemical compounds, of which only 608 are active against the HIV virus. Each compound is described by $d = 6$ BCUT numbers, which are generated and kindly provided to us by computational chemists at GlaxoSmithKline, Inc. The BCUT numbers can be thought of as mapping compounds into a six-dimensional space in which activity occurs in local regions. Again, for details of what these BCUT numbers are, refer to Burden (1989).

## 6.3 Methods

We apply LAGO with three different kernel functions, uniform, triangular and Gaussian, and use the following algorithms as benchmarks: the SVM and ASVM for reasons discussed above (Section 5) and the K-nearest neighbor classifier (KNN). For both SVM and ASVM, the Gaussian kernel is used. KNN is chosen here as a benchmark for two reasons. First, the way the adaptive bandwidths are computed in LAGO has a similar flavor to the KNN algorithm. Secondly, an earlier study (Wang 2005) using the same data set has already concluded that KNN is one of the best methods amongst a number of techniques compared including trees, neural networks, MARS (Friedman 1991), generalized additive models and logistic regression. In that study, four experiments were conducted to evaluate the performance of different methods. In each experiment, the data set was randomly split using stratified sampling into a training set and a test set, each with $n = 14,906$ compounds, of which 304 are active compounds. Performance was then assessed by the AP on the test set. Over the four replications, a paired t-test concluded that KNN significantly outperformed all other methods at a 5% significance level.

Here, we also conduct four experiments, using the same four splits of the data set. The four experiments will also be referred to as "Split 1", ..., "Split 4" in the text below. In each experiment, all tuning parameters are selected using 5-fold cross-validation on the training set whereas all performance measures are computed on the test set. For KNN, there is only one tuning parameter $K$. For SVM, there are two tuning parameters, $\gamma$ and $\sigma$ (see Section 5.1); for ASVM, there is an additional tuning parameter for the class weights (see Section 5.4). For LAGO, there are two tuning parameters, $K$ and $\alpha$ (see Section 4.3). The actual tuning parameters selected for LAGO are given in Table 2.

Table 2: *Tuning Parameters Selected for LAGO Using Different Kernels.*

| | Uniform | | Triangular | | Gaussian | |
|---|---|---|---|---|---|---|
| | $K$ | $\alpha$ | $K$ | $\alpha$ | $K$ | $\alpha$ |
| Split 1 | 7 | 1 | 4 | 3 | 3 | 3 |
| Split 2 | 7 | 1 | 2 | 5 | 2 | 4 |
| Split 3 | 7 | 1 | 4 | 3 | 5 | 2 |
| Split 4 | 7 | 1 | 4 | 3 | 4 | 2 |

**Remark.** In deciding to use 50% of the data for training, we attempt to represent a large-scale drug discovery problem with a smaller, public-domain data set. We seek to balance two factors: (1) the NCI data set consists of nearly 30,000 compounds while real pharmaceutical libraries can contain a million compounds; (2) as such, a training set selected from a real pharmaceutical library would often contain far fewer than 50% of all compounds — in fact, they might well contain roughly the same number of compounds as our training set (approximately 15,000). Thus, the size of our training set matches real problems but our test set is smaller than usual. However, despite this smaller test set, results in Section 6.4 are still statistically significant. In addition, the proportion of active compounds in our data set (about 2%) is also typical of real problems.

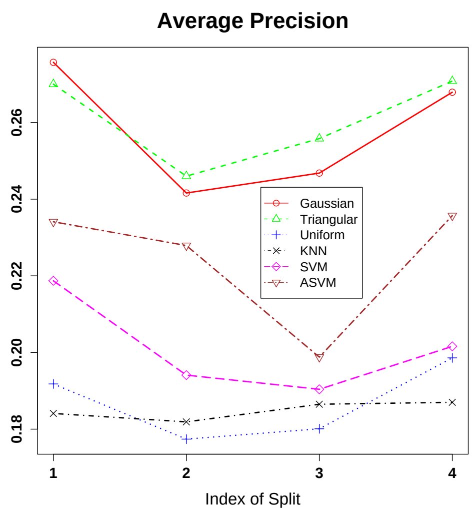

Figure 9: *The average precision of all algorithms evaluated on the test data. The terms "Gaussian," "Triangular" and "Uniform" refer to our LAGO model using the corresponding kernel functions.*

## 6.4 Results

Figure 9 plots the average precision and Figure 10 shows the hit curves when different methods are applied to different test data. Because we measure performance four times using slightly different training and test data, it is also possible to perform an ANOVA comparison of these different methods. Let $\mu_K, \mu_S, \mu_A, \mu_U, \mu_T$ and $\mu_G$ denote the average performance (measured by average precision) of KNN, SVM, ASVM and LAGO using the uniform, the triangular and the Gaussian kernels, respectively; we construct five (non-orthogonal) contrasts (see Table 3). The ANOVA result is given in Table 4. The main conclusion, at either a 1% or a 5% significance level, is the following:

$$\text{(Triangle LAGO} \sim \text{Gaussian LAGO)} \succ \text{ASVM} \succ \text{SVM} \succ \text{(Uniform LAGO} \sim \text{KNN)},$$

where "$\sim$" means "not significantly different from" and "$\succ$" means "significantly better than."

Notice that, although the hit curve for ASVM can sometimes be higher than that of LAGO, e.g., for $n > 300$ in Splits 2 and 4, the hit curve for LAGO is always higher initially. This means the top-ranked compounds identified by LAGO are more likely to be active than the top-ranked compounds identified by ASVM. LAGO has consistently higher average precision because this measure puts greater weight on early detection.

Table 3: *Estimated Contrasts.*

| Contrast | Expression | Estimate |
|---|---|---|
| Cntr1 | $\mu_T - \mu_G$ | 0.0027 |
| Cntr2 | $\mu_G - \mu_A$ | 0.0339 |
| Cntr3 | $\mu_A - \mu_S$ | 0.0230 |
| Cntr4 | $\mu_S - (\mu_K + \mu_U)/2$ | 0.0157 |
| Cntr5 | $\mu_U - \mu_K$ | 0.0014 |

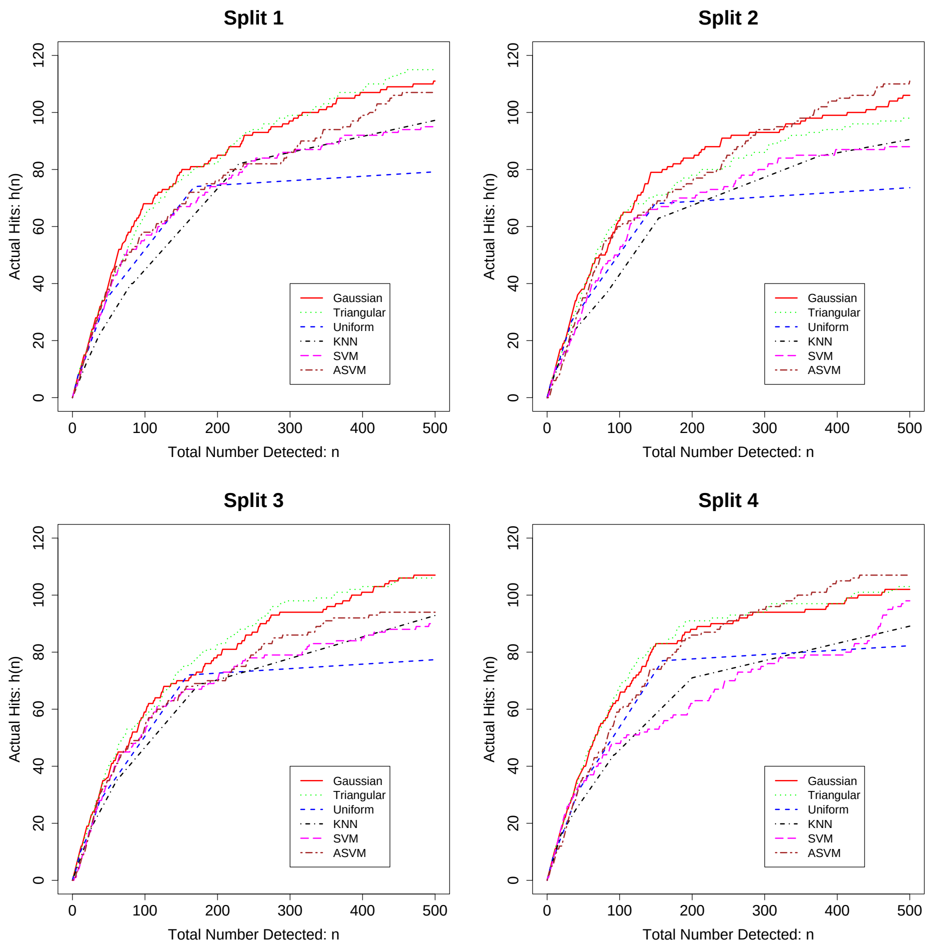

Figure 10: *The hit curve on test data. Only the initial part of the curves (up to $n = 500$) are shown. The terms "Gaussian," "Triangular" and "Uniform" refer to our LAGO model using the corresponding kernel functions.*

Table 4: *ANOVA Analysis of Differences Among Methods.*

| Source | SS ($\times 10^{-4}$) | df | MS ($\times 10^{-4}$) | $F_0$ | P-Value |
|---|---|---|---|---|---|
| Methods | 233.504 | 5 | 46.701 | 64.307 | <0.0001 |
| *Cntr1* | *0.140* | *1* | *0.140* | *0.193* | *0.6664* |
| *Cntr2* | *22.916* | *1* | *22.916* | *31.556* | *<0.0001* |
| *Cntr3* | *10.534* | *1* | *10.534* | *14.505* | *0.0017* |
| *Cntr4* | *6.531* | *1* | *6.531* | *8.994* | *0.0090* |
| *Cntr5* | *0.036* | *1* | *0.036* | *0.050* | *0.8258* |
| Splits | 18.877 | 3 | 6.292 | 8.664 | 0.0014 |
| Error | 10.893 | 15 | 0.726 | | |
| Total | 263.274 | 23 | | | |

## 6.5 Comparison with SVM and ASVM

As mentioned in Section 5, since both LAGO and SVM can be viewed as RBF networks (constructed using entirely different approaches), it is instructive to examine in more detail the structure of these two RBF network models. Recall the main feature of our RBF network LAGO is that we choose to put basis functions at and only at all the class-1 observations in the training set, which is very efficient since class 1 is the rare class. Recall also that the SVM will put basis functions at all the support points.

Table 5 lists the number of training observations from both classes that are chosen as support vectors by SVM and ASVM. In all cases, almost every observation from class 1 (active compounds) is chosen as a support vector. This suggests using all class-1 observations in the training data is a good idea. On the other hand, both SVM and ASVM also use a fairly large number of observations from class 0 (inactive compounds) as support vectors. Given that the overall performance of SVM and ASVM is inferior to that of LAGO, these extra support vectors seem to be quite wasteful.

We must also stress here that fitting the ASVM is an *extremely* expensive computational exercise. Over 26 days of CPU time were needed to produce the results for all four splits. This was made feasible by a powerful parallel computing cluster with 38 nodes.

Table 5: *Number of Support Vectors from Classes $C_0$ and $C_1$.*

| | SVM | | ASVM | |
|---|---|---|---|---|
| | $C_0$ | $C_1$ | $C_0$ | $C_1$ |
| Split 1 | 11531 | 294 | 5927 | 291 |
| Split 2 | 11419 | 303 | 3472 | 284 |
| Split 3 | 11556 | 293 | 11706 | 290 |
| Split 4 | 1863 | 293 | 6755 | 281 |
| Total Possible | 14602 | 304 | 14602 | 304 |

## 6.6 Empirical Evidence for $\beta = 0$

Finally, we present some empirical evidence to support the theoretical claim that the extra parameter $\beta$ in the more general parameterization of LAGO (Section 4.3) should be set to 0. To do so, we treat $\beta$ as an extra tuning parameter. With $K$ and $\alpha$, we now have altogether three tuning parameters, which makes cross-validation slightly more difficult and tedious. To simplify the computation, we conduct an experiment on the NCI AIDS antiviral data by fixing the parameter $K$ and using cross-validation to select $\alpha$ and $\beta$ simultaneously. We use the Gaussian kernel and simply fix $K = 5$, which is a reasonable choice based on results from Table 2. Figure 11 shows that cross validation selects $\beta = 0$, completely in agreement with our theoretical choice!

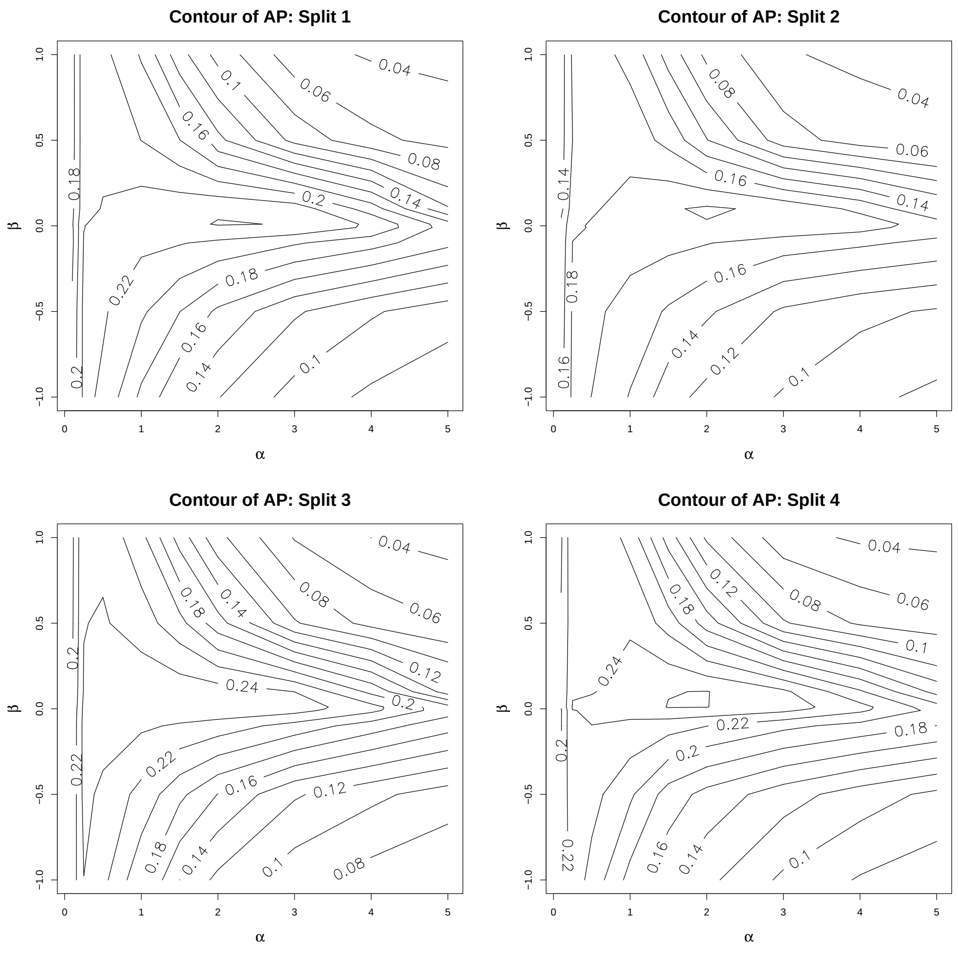

Figure 11: *Choosing $\alpha$ and $\beta$ (while fixing $K = 5$) using 5-fold cross-validation on the training data. The contours are calculated on a $9 \times 9$ grid, for $\alpha$ = 0.1, 0.25, 0.5, 1, 1.5, 2, 3, 4, 5 and $\beta$ = -1, -0.5, -0.1, -0.01, 0, 0.01, 0.1, 0.5, 1.*

## 7 Summary

We now summarize the main contributions of this article. First of all, we outlined a general paradigm for constructing efficient computational models for the statistical detection problem. This paradigm can be summarized as follows: *Estimate the density for the rare but important class alone and adjust locally depending on the density of the background class nearby.* The two steps are: estimation and adjustment. Significant computational saving can be achieved because we need only model the rare class.

Secondly, based on the general paradigm, we developed the new LAGO model. A powerful feature of LAGO is that the two steps (estimation and adjustment) are computationally combined into one. In particular, we estimate $p_1$ with an adaptive bandwidth kernel density estimator and can show that the adjustment needed in the second step is proportional to the adaptive bandwidth already calculated in the first step. Hence, we obtain additional computational saving because the adjustment step is automatic and requires no extra computation.

Finally, we showed that LAGO could be viewed as an RBF network. Based on this point of view, we presented a more general parameterization for LAGO and illustrated its usefulness with a real data set from drug discovery. We showed that LAGO was competitive and, when the Gaussian or the triangular kernel was used, it outperformed both SVM and KNN.

## A The Average Precision

In this appendix, we give details about how the *average precision* is defined and calculated. Referring to Figure 2, let $h(t)$ be the hit curve; let

$$r(t) = \frac{h(t)}{\pi} \quad \text{and} \quad p(t) = \frac{h(t)}{t}. \tag{18}$$

The *average precision* is defined as

$$\text{AP} = \int_0^1 p(r)dr = \int_0^1 p(t)dr(t) = \int_0^1 \frac{\pi r(t)dr(t)}{t}. \tag{19}$$

It is most instructive to examine a few concrete examples.

**Example 1 (Random Selection).** Suppose $h(t) = \pi t$, i.e., the proportion of relevant items among those detected so far stays constant at $\pi$. This is the case of random selection. In this case, $r(t) = h(t)/\pi = t$. Therefore

$$\text{AP(Random)} = \int_0^1 \frac{\pi r(t)dr(t)}{t} = \int_0^1 \frac{\pi t}{t}dt = \pi. \qquad \blacksquare$$

**Example 2 (Perfect Detection).** Suppose

$$h(t) = \begin{cases} t, & t \in [0, \pi]; \\ \pi, & t \in (\pi, 1]. \end{cases}$$

That is, everything detected is relevant until all relevant items are exhausted at $t = \pi$. This is the case of *ideal* detection; one can't possibly do any better than this. This implies

$$r(t) = \begin{cases} \frac{t}{\pi}, & t \in [0, \pi]; \\ 1, & t \in (\pi, 1]. \end{cases}$$

Therefore,

$$\text{AP(Perfect)} = \int_0^1 \frac{\pi r(t) dr(t)}{t} = \int_0^\pi \left( \frac{\pi \times \frac{t}{\pi} \times \frac{1}{\pi}}{t} \right) dt + \int_\pi^1 \left( \frac{\pi \times 1 \times 0}{t} \right) dt = 1. \qquad \blacksquare$$

**Example 3 (Practical Calculation).** In practice, $h(t)$ takes value only at a finite number of points $t_i = i/n$, $i = 1, 2, ..., n$. Hence, the integral (19) is replaced with a finite sum

$$\int_0^1 p(t) dr(t) = \sum_{i=1}^n p(t_i) \Delta r(t_i) \tag{20}$$

where $\Delta r(t_i) = r(t_i) - r(t_{i-1})$. We illustrate this with a concrete example. Table 6 summarizes the performance of two hypothetical methods, A and B.

In this case, we have

$$\text{AP(A)} = \sum_{i=1}^{10} p(t_i) \Delta r(t_i) = \left( \frac{1}{1} + \frac{2}{2} + \frac{3}{4} \right) \times \frac{1}{3} \approx 0.92$$

and

$$\text{AP(B)} = \sum_{i=1}^{10} p(t_i) \Delta r(t_i) = \left( \frac{1}{1} + \frac{2}{4} + \frac{3}{8} \right) \times \frac{1}{3} \approx 0.63,$$

which agrees with our intuition that algorithm A performs better here because it detects the three hits earlier on. $\blacksquare$

## B Proof of Theorem 1

In this appendix, we prove Theorem 1. Suppose $w_1, w_2, ..., w_m$ are i.i.d. observations uniformly distributed on the interval $[x_0 - 1/2c_0, x_0 + 1/2c_0]$. Let $z_j = |w_j - x_0|$ be the distance between $w_j$ and $x_0$, we first claim that

$$\text{E}(z_{(k)}) = \frac{k}{2(m+1)c_0} \tag{21}$$

where $z_{(k)}$ is the $k$th order statistic of $z_j, j = 1, 2, ..., m$. We can then write the average distance between $x_0$ and its K nearest neighbors from $w_j, j = 1, 2, ..., m$ as

$$r_0 = \frac{1}{K} \sum_{k=1}^K z_{(k)}. \tag{22}$$

Table 6: *Two Algorithms for a Toy Data Set with 3 Relevant Cases.*

| | Algorithm A | | | Algorithm B | | |
|---|---|---|---|---|---|---|
| Item ($i$) | Hit | $p(t_i)$ | $\Delta r(t_i)$ | Hit | $p(t_i)$ | $\Delta r(t_i)$ |
| 1 | 1 | $\frac{1}{1}$ | $\frac{1}{3}$ | 1 | $\frac{1}{1}$ | $\frac{1}{3}$ |
| 2 | 1 | $\frac{2}{2}$ | $\frac{1}{3}$ | 0 | $\frac{1}{2}$ | 0 |
| 3 | 0 | $\frac{2}{3}$ | 0 | 0 | $\frac{1}{3}$ | 0 |
| 4 | 1 | $\frac{3}{4}$ | $\frac{1}{3}$ | 1 | $\frac{2}{4}$ | $\frac{1}{3}$ |
| 5 | 0 | $\frac{3}{5}$ | 0 | 0 | $\frac{2}{5}$ | 0 |
| 6 | 0 | $\frac{3}{6}$ | 0 | 0 | $\frac{2}{6}$ | 0 |
| 7 | 0 | $\frac{3}{7}$ | 0 | 0 | $\frac{2}{7}$ | 0 |
| 8 | 0 | $\frac{3}{8}$ | 0 | 1 | $\frac{3}{8}$ | $\frac{1}{3}$ |
| 9 | 0 | $\frac{3}{9}$ | 0 | 0 | $\frac{3}{9}$ | 0 |
| 10 | 0 | $\frac{3}{10}$ | 0 | 0 | $\frac{3}{10}$ | 0 |

Equation (21) above then implies

$$\mathrm{E}(r_0) = \frac{1}{K}\sum_{k=1}^{K}\mathrm{E}(z_{(k)}) = \frac{1}{K}\sum_{k=1}^{K}\frac{k}{2(m+1)c_0} = \frac{K+1}{4(m+1)c_0}, \tag{23}$$

which completes the proof.

In order to prove equation (21), first note that we can assume $x_0 = 0$ without loss of generality. If $w_1, w_2, ..., w_m$ are uniformly distributed on $[-1/2c_0, 1/2c_0]$, then it is easy to show that the distribution for $z_j = |w_j|$, $p(z)$, is uniform on the interval $[0, 1/2c_0]$, i.e., $p(z) = 2c_0$ if $0 \leq z \leq 1/2c_0$.

Now if $z_{(k)}$ is the $k$th order statistic of $z_1, z_2, ..., z_m$, then using standard results on order statistics (e.g., Ross 2000, p. 58, Example 2.37) we can obtain the density for $z_{(k)}$:

$$f_{z_{(k)}}(z) = \frac{m!}{(k-1)!(m-k)!}(2c_0)(2c_0 z)^{k-1}(1-2c_0 z)^{m-k}. \tag{24}$$

Using (24) we can calculate $\mathrm{E}(z_{(k)})$ as follows:

$$\mathrm{E}(z_{(k)}) = \int_0^{1/2c_0} z f_{z_{(k)}}(z)dz = \int_0^{1/2c_0}\frac{m!}{(k-1)!(m-k)!}(2c_0 z)^k(1-2c_0 z)^{m-k}dz.$$

By letting $y = 2c_0 z$, this integral becomes

$$\begin{aligned}\int_0^1 \frac{m!}{(k-1)!(m-k)!}(y)^k(1-y)^{m-k}\frac{1}{2c_0}\,dy &= \int_0^1\left(\frac{k}{m+1}\right)\frac{(m+1)!}{k!(m-k)!}(y)^k(1-y)^{m-k}\frac{1}{2c_0}\,dy \\ &= \frac{k}{2(m+1)c_0},\end{aligned}$$

where the last step is due to the fact that

$$\int_0^1 \frac{(m+1)!}{k!(m-k)!}(y)^k(1-y)^{m-k}dy = 1$$

because the integrand above can be recognized as a Beta$(k+1, m-k+1)$ density function.

## Acknowledgment

This work is partially supported by the Natural Science and Engineering Research Council (NSERC) of Canada as well as the Mathematics of Information Technology and Complex Systems (MITACS) network. Infrastructure provided by the Canada Foundation for Innovation and operated by the Acadia Centre for Mathematical Modelling and Computation was used in this research. The authors would like to thank S. Young for providing the NCI AIDS Antiviral Data Set, Duane Currie for assistance with parallel computing, as well as William J. Welch, R. Wayne Oldford, Jerry F. Lawless and Mary E. Thompson for various suggestions and comments. The first author would especially like to thank Professor William Welch for giving him the most wonderful opportunity to spend time in the statistics department at the University of British Columbia while working on this article.

## References

Bolton, R. J. and Hand, D. J. (2002). Statistical fraud detection: A review. *Statistical Science*, **17**(3), 235–255.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth.

Burden, F. R. (1989). Molecular identification number for substructure searches. *Journal of Chemical Information and Computer Sciences*, **29**, 225–227.

Cantor, S. B. and Kattan, M. W. (2000). Determining the area under the ROC curve for a binary diagnostic test. *Medical Decision Making*, **20**(4), 468–470.

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

Deerwester, S., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the Society for Information Science*, **41**(6), 391–407.

Dumais, S. T. (1991). Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments and Computers*, **23**(2), 229–236.

Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, **19**, 1–67.

Hastie, T. J., Tibshirani, R. J., and Friedman, J. H. (2001). *The Elements of Statistical Learning: Data-Mining, Inference and Prediction*. Springer-Verlag.

Lam, R. L. H., Welch, W. J., and Young, S. S. (2002). Uniform coverage designs for molecule selection. *Technometrics*, **44**(2), 99–109.

Peng, F., Schuurmans, D., and Wang, S. (2003). Augmenting naive bayes classifiers with statistical language models. *Information Retrieval*, **7**(3), 317–345.

R Development Core Team (2004). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0; URL http://www.R-project.org.

Ross, S. M. (2000). *Introduction to Probability Models*. Academic Press, 7th edition.

Schölkopf, B., Sung, K. K., Burges, C. J. C., Girosi, F., Niyogi, P., Poggio, T., and Vapnik, V. (1997). Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE Transactions on Signal Processing*, **45**(11), 2758–2765.

Silverman, B. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman and Hall.

Wang, Y. (2005). *Statistical Methods for High Throughput Screening Drug Discovery Data*. Ph.D. dissertation, University of Waterloo, Department of Statistics and Actuarial Science, Waterloo, Ontario, Canada.