CLASSIFICATION

ULISSES BRAGA-NETO AND EDWARD DOUGHERTY

1. Introduction

Classification plays an important role in genomic signal analysis. For instance, cDNA microarrays can provide expression measurements for thousands of genes at once, and a key goal is to perform classification via different expression patterns. This requires designing a classifier (decision function) that takes a vector of gene expression levels as input, and outputs a class label that predicts the class containing the input vector. Classification can be between different kinds of cancer, different stages of tumor development, or a host of such differences [1–12] (see also the bibliography on microarray-based classification provided as part of the supplementary information to [13]) . Classifiers are designed from a sample of expression vectors. This involves assessing expression levels from RNA obtained from the different tissues with microarrays, determining genes whose expression levels can be used as classifier features (variables), and then applying some rule to design the classifier from the sample microarray data. Expression values have randomness arising from both biological and experimental variability. Design, performance evaluation, and application of features must take this randomness into account. Three critical issues arise. First, given a set of variables, how does one design a classifier from the sample data that provides good classification over the general population? Second, how does one estimate the error of a designed classifier when data are limited? Third, given a large set of potential features, such as the large number of expression levels provided by each microarray, how does one select a set of features as the input to the classifier? Small samples (relative to the number of features) are ubiquitous in genomic signal processing and impact all three issues [14].

2. Classifier Design

Classification involves a *feature vector* $\mathbf{X} = (X_1, X_2, \ldots, X_d)$ on d-dimensional Euclidean space $\mathbb{R}^d$, composed of random variables (*features*), a binary random variable Y, and a *classifier* $\psi : \mathbb{R}^d \to \{0, 1\}$ to serve as a predictor of Y, which means that Y is to be predicted by $\psi(\mathbf{X})$. The values, 0 or 1, of Y are treated as class *labels*. We assume there is a joint *feature-label distribution* F for the pair $(\mathbf{X}, Y)$ that completely characterizes the stochastic classification problem.

The space of all classifiers, which in our case is the space of all binary functions on $\mathbb{R}^d$, will be denoted by $\mathcal{F}$. The error $\varepsilon[\psi]$ of $\psi \in \mathcal{F}$ is the probability that the classification is erroneous, namely, $\varepsilon[\psi] = P(\psi(\mathbf{X}) \neq Y)$. It can be written as

$$\varepsilon[\psi] = E_F[|Y - \psi(\mathbf{X})|], \tag{1}$$

where the expectation is taken relative to the feature-label distribution F (as indicated by the notation E_F). In other words, $\varepsilon[\psi]$ equals the mean absolute difference

between label and classification. Owing to the binary nature of $\psi(\mathbf{X})$ and Y, $\varepsilon[\psi]$ also equals the mean square error between label and classification.

2.1. **Bayes Classifier.** An optimal classifier, ψ_d, is one having minimal error, ε_d, among all $\psi \in \mathcal{F}$, so that it is the minimal mean-absolute-error predictor of Y. The optimal classifier ψ_d is called the *Bayes classifier* and its error ε_d is called the *Bayes error*. The Bayes classifier, and thus the Bayes error, depends on the feature-label distribution of $(\mathbf{X}, Y)$ — how well the labels are distributed among the variables being used to discriminate them, and how the variables are distributed in $\mathbb{R}^d$.

The posterior distributions for $\mathbf{X}$ are defined by $\eta_0(\mathbf{x}) = f_{\mathbf{X},Y}(\mathbf{x}, 0)/f_{\mathbf{X}}(\mathbf{x})$ and $\eta_1(\mathbf{x}) = f_{\mathbf{X},Y}(\mathbf{x}, 1)/f_{\mathbf{X}}(\mathbf{x})$, where $f_{\mathbf{X},Y}(\mathbf{x}, y)$ and $f_{\mathbf{X}}(\mathbf{x})$ are the densities for $(\mathbf{X}, Y)$ and $\mathbf{X}$, respectively. The posteriors $\eta_0(\mathbf{x})$ and $\eta_1(\mathbf{x})$ give the probability that $Y = 0$ or $Y = 1$, respectively, given $\mathbf{X} = \mathbf{x}$. Note that $\eta_0(\mathbf{x}) = 1 - \eta_1(\mathbf{x})$. Note also that, as a function of $\mathbf{X}$, $\eta_0(\mathbf{X})$ and $\eta_1(\mathbf{X})$ are random variables. Furthermore, in this binary-label setting, $\eta_1(\mathbf{x}) = E[Y|\mathbf{x}]$ is the conditional expectation of Y given $\mathbf{x}$. The error of an arbitrary classifier can be expressed as

$$\varepsilon[\psi] = \int\limits_{\{\mathbf{x}|\psi(\mathbf{x})=0\}} \eta_1(\mathbf{x}) f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} + \int\limits_{\{\mathbf{x}|\psi(\mathbf{x})=1\}} \eta_0(\mathbf{x}) f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} \tag{2}$$

It is easy to verify that the right-hand side of Eq. (2) is minimized by

$$\psi_d(\mathbf{x}) = \begin{cases} 1, & \text{if } \eta_1(\mathbf{x}) \geq \eta_0(\mathbf{x}) \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Hence, the Bayes classifier $\psi_d(\mathbf{x})$ is defined to be 1 or 0 according to whether Y is more likely to be 1 or 0 given $\mathbf{x}$ (ties may be broken arbitrarily). For this reason, the Bayes classifier is also known as the *maximum a-posteriori* (MAP) classifier. It follows from Eqs. (2) and (3) that the Bayes error is given by

$$\begin{aligned} \varepsilon_d &= \int\limits_{\{\mathbf{x}|\eta_1(\mathbf{x})<\eta_0(\mathbf{x})\}} \eta_1(\mathbf{x}) f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} + \int\limits_{\{\mathbf{x}|\eta_1(\mathbf{x})\geq\eta_0(\mathbf{x})\}} \eta_0(\mathbf{x}) f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} \\ &= E\left[\min\{\eta_0(\mathbf{X}), \eta_1(\mathbf{X})\}\right] \end{aligned} \tag{4}$$

By Jensen's inequality, it follows from Eq. (4) that $\varepsilon_d \leq \min\{E[\eta_0(\mathbf{X})], E[\eta_1(\mathbf{X})]\}$. Therefore, if either of the posteriors are uniformly small (for example, if one of the classes is much more likely than the other), then the Bayes error is necessarily small.

The problem with the Bayes classifier is that the feature-label distribution is typically unknown, and thus so are the posteriors. Therefore, we must design a classifier from sample data. An obvious approach would be to estimate the posterior distributions from data, but often we do not have sufficient data to obtain good estimates. Moreover, good classifiers can be obtained even when we lack sufficient data for satisfactory density estimation.

2.2. **Classification Rules.** Design of a classifier ψ_n from a random sample $S_n = \{(\mathbf{X}_1, Y_1), (\mathbf{X}_2, Y_2), \ldots, (\mathbf{X}_n, Y_n)\}$ of vector-label pairs drawn from the feature-label distribution requires a classification rule that operates on random samples to yield a classifier. A *classification rule* is a mapping $\Psi_n : [\mathbb{R}^d \times \{0, 1\}]^n \to \mathcal{F}$. Given a sample S_n, we obtain a designed classifier $\psi_n = \Psi_n(S_n) \in \mathcal{F}$, according to the rule Ψ_n. To be fully formal, one might write $\psi_n(S_n; \mathbf{X})$ rather than $\psi_n(\mathbf{X})$; however, we will use the simpler notation, keeping in mind that ψ_n derives from

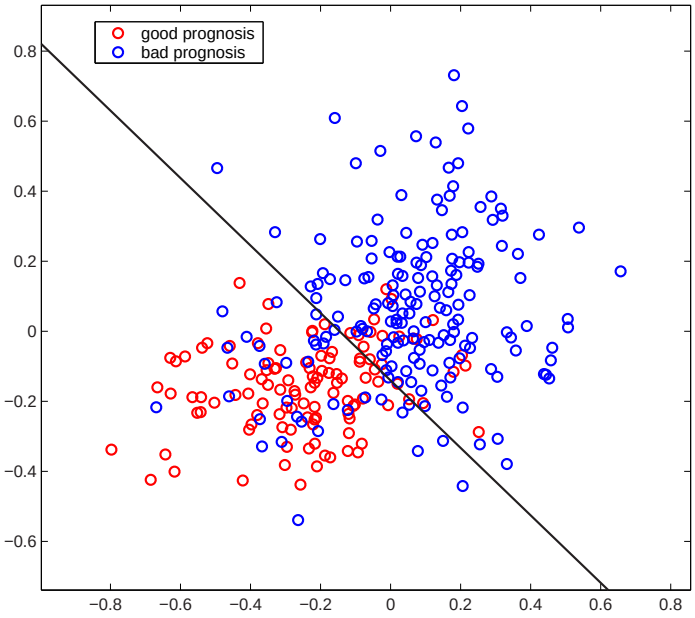

FIGURE 1. Example of a linear designed classifier.

a classification rule applied to a feature-label sample. Note that what is usually called a classification rule is really a sequence of classification rules depending on n. Fig. 1 presents an example of a linear designed classifier, obtained via the LDA classification rule (see Section 2.4.9). The sample data in this example consist of expression values of two top discriminatory genes on a total of 295 microarrays from a cancer classification study [15] (see Section 2.5 for more details about this data set).

The Bayes error ε_d is estimated by the expected error of the designed classifier $\varepsilon_n = \varepsilon[\psi_n]$. There is a *design error*

$$\Delta_n = \varepsilon_n - \varepsilon_d, \tag{5}$$

ε_n and Δ_n being sample-dependent random variables. The expected design error is $E[\Delta_n]$, the expectation being relative to all possible samples. The expected error of ψ_n is decomposed according to

$$E[\varepsilon_n] = \varepsilon_d + E[\Delta_n]. \tag{6}$$

The quantity $E[\varepsilon_n]$, or alternatively $E[\Delta_n]$, measures the global properties of classifications rules, rather than the performance of classifiers designed on individual samples (on the other hand, a classification rule for which $E[\varepsilon_n]$ is small will also tend to produce designed classifiers that display small error).

Asymptotic properties of a classification rule concern large samples (as $n \to \infty$). A rule is said to be *consistent* for a feature-label distribution of $(\mathbf{X}, Y)$ if $\Delta_n \to 0$ in the mean, meaning $E[\Delta_n] \to 0$ as $n \to \infty$. For a consistent rule, the expected design error can be made arbitrarily small for a sufficiently large amount of data. Since the feature-label distribution is unknown a priori, rules for which convergence is independent of the distribution are desirable. A classification rule

is *universally consistent* if $\Delta_n \to 0$ in the mean for any distribution of $(\mathbf{X}, Y)$. Universal consistency is useful for large samples, but has little consequence for small samples.

2.3. **Constrained Classifier Design.** A classification rule can yield a classifier that makes very few, or no, errors on the sample data on which it is designed, but performs poorly on the distribution as a whole, and therefore on new data to which it is applied. This situation is exacerbated by complex classifiers and small samples. If the sample size is dictated by experimental conditions, such as cost or the availability of patient RNA for expression microarrays, then one only has control over classifier complexity. The situation with which we are concerned is typically referred to as *overfitting*. The basic point is that a classification rule should not cut up the space in a manner too complex for the amount of sample data available. This might improve the *apparent* error rate (i.e., the number of errors committed by the classifier using the training data as testing points), but at the same time it will most likely worsen the true error of the classifier for independent future data (also called the *generalization error* in this context). The problem is not necessarily mitigated by applying an error-estimation rule — perhaps more sophisticated than the apparent error rate — to the designed classifier to see if it "actually" performs well, since when there is only a small amount of data available, error-estimation rules are very imprecise (as we will see in Section 4), and the imprecision tends to be worse for complex classification rules. Hence, a low error estimate is not sufficient to overcome our expectation of a large expected error when using a complex classifier with a small data set. Depending on the amount of data available, we need to consider constrained classification rules.

Constraining classifier design means restricting the functions from which a classifier can be chosen to a class $\mathcal{C} \subseteq \mathcal{F}$. This leads to trying to find an optimal *constrained classifier*, $\psi_{\mathcal{C}} \in \mathcal{C}$, having error $\varepsilon_{\mathcal{C}}$. Constraining the classifier can reduce the expected error, but at the cost of increasing the error of the best possible classifier. Since optimization in $\mathcal{C}$ is over a subclass of classifiers, the error $\varepsilon_{\mathcal{C}}$ of $\psi_{\mathcal{C}}$ will typically exceed the Bayes error, unless the Bayes classifier happens to be in $\mathcal{C}$. This cost of constraint (approximation) is

$$\Delta_{\mathcal{C}} = \varepsilon_{\mathcal{C}} - \varepsilon_d. \tag{7}$$

A classification rule yields a classifier $\psi_{n,\mathcal{C}} \in \mathcal{C}$, with error $\varepsilon_{n,\mathcal{C}}$, and $\varepsilon_{n,\mathcal{C}} \geq \varepsilon_{\mathcal{C}} \geq \varepsilon_d$. Design error for constrained classification is

$$\Delta_{n,\mathcal{C}} = \varepsilon_{n,\mathcal{C}} - \varepsilon_{\mathcal{C}}. \tag{8}$$

For small samples, this can be substantially less than Δ_n, depending on $\mathcal{C}$ and the classification rule. The error of the designed constrained classifier is decomposed as

$$\varepsilon_{n,\mathcal{C}} = \varepsilon_d + \Delta_{\mathcal{C}} + \Delta_{n,\mathcal{C}}. \tag{9}$$

Therefore, the expected error of the designed classifier from $\mathcal{C}$ can be decomposed as

$$E[\varepsilon_{n,\mathcal{C}}] = \varepsilon_d + \Delta_{\mathcal{C}} + E[\Delta_{n,\mathcal{C}}]. \tag{10}$$

The constraint is beneficial if and only if $E[\varepsilon_{n,\mathcal{C}}] < E[\varepsilon_n]$, i.e., if

$$\Delta_{\mathcal{C}} < E[\Delta_n] - E[\Delta_{n,\mathcal{C}}]. \tag{11}$$

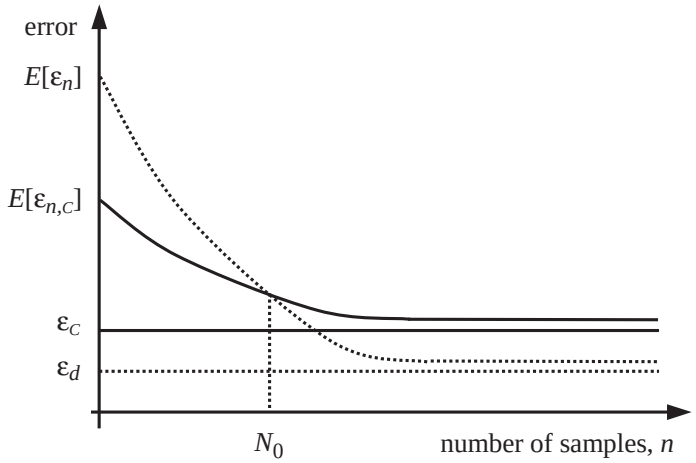

FIGURE 2. Errors of unconstrained and constrained classifiers.

If the cost of constraint is less than the decrease in expected design error, then the expected error of $\psi_{n,\mathcal{C}}$ is less than that of ψ_n. The dilemma: strong constraint reduces $E[\Delta_{n,\mathcal{C}}]$ at the cost of increasing $\varepsilon_{\mathcal{C}}$.

The matter can be graphically illustrated. For two classification rules to be shortly introduced, the discrete-data plug-in rule and the cubic histogram rule with fixed cube size, $E[\Delta_n]$ is non-increasing, meaning that $E[\Delta_{n+1}] \leq E[\Delta_n]$. This means that the expected design error never increases as sample sizes increase, and it holds for any feature-label distribution. Such classification rules are called *smart*. They fit our intuition about increasing sample sizes. Now consider a consistent rule, constraint, and distribution for which $E[\Delta_{n+1}] \leq E[\Delta_n]$ and $E[\Delta_{n+1,\mathcal{C}}] \leq E[\Delta_{n,\mathcal{C}}]$. Fig. 2 illustrates the design problem. If n is sufficiently large, then $E[\varepsilon_n] < E[\varepsilon_{n,\mathcal{C}}]$; however, if n is sufficiently small, then $E[\varepsilon_n] > E[\varepsilon_{n,\mathcal{C}}]$. The point N_0 at which the decreasing lines cross is the cut-off: for $n > N_0$, the constraint is detrimental; for $n < N_0$, it is beneficial. When $n < N_0$, the advantage of the constraint is the difference between the decreasing solid and dashed lines.

A fundamental theorem provides bounds for $E[\Delta_{n,\mathcal{C}}]$ [16]. The *empirical-error rule* chooses the classifier in $\mathcal{C}$ that makes the least number of errors on the sample data. For this (intuitive) rule, $E[\Delta_{n,\mathcal{C}}]$ satisfies the bound

$$E[\Delta_{n,\mathcal{C}}] \leq 8\sqrt{\frac{V_{\mathcal{C}} \log n + 4}{2n}} \tag{12}$$

where $V_{\mathcal{C}}$ is the *VC* (*Vapnik-Chervonenkis*) *dimension* of $\mathcal{C}$. Details of the VC dimension are outside the scope of this paper. Nonetheless, it is clear from Eq. (12) that n must greatly exceed $V_{\mathcal{C}}$ for the bound to be small.

To illustrate the problematic nature of complex (high-VC-dimension) classifiers, we apply the preceding bound to two classifier classes to be introduced in the next section. The VC dimension of a linear classifier is $d + 1$, where d is the number of variables, whereas the VC dimension of a neural network with an even number k of neurons has the lower bound $V_{\mathcal{C}} \geq dk$ [17]. If k is odd, then $V_{\mathcal{C}} \geq d(k-1)$. Thus, if one wants to use a large number of neurons to obtain a classifier that can very finely fit the data, the VC dimension can greatly exceed that of a linear

classifier. To appreciate the implications, suppose $d = k = 10$. Then the VC dimension of a neural network is bounded below by 100. Setting $V_{\mathcal{C}} = 100$ and $n = 5000$ in Eq. (12) yields a bound exceeding 1, which says nothing. Not only is the inequality in Eq. (12) a bound, it is worst-case because there are no distributional assumptions. The situation may not be nearly so bad. Still, one must proceed with care, especially in the absence of distributional knowledge. Increasing complexity is often counterproductive unless there is a large sample available. Otherwise, one could easily end up with a very bad classifier whose error estimate is very small!

2.4. **Specific Classification Rules.** In this section of the chapter we discuss some commonly employed classification rules, beginning with a rule that is employed in different manners to produce related rules.

2.4.1. *Plug-in Rule.* Considering the Bayes classifier defined by Eq. (3), let $\eta_{1,n}(\mathbf{x})$ be an estimate of $\eta_1(\mathbf{x})$ based on a sample S_n, and let $\eta_{0,n}(\mathbf{x}) = 1 - \eta_{1,n}(\mathbf{x})$. A reasonable classification rule is to define $\psi_n(\mathbf{x})$ according to Eq. (3) with $\eta_{0,n}(\mathbf{x})$ and $\eta_{1,n}(\mathbf{x})$ in place of $\eta_0(\mathbf{x})$ and $\eta_1(\mathbf{x})$, respectively. For this *plug-in rule*,

$$\Delta_n = \int\limits_{\{\mathbf{x}:\psi_n(\mathbf{x})\neq\psi_d(\mathbf{x})} |\eta_{1,n}(\mathbf{x}) - \eta_{0,n}(\mathbf{x})|\, f_{\mathbf{X}}(\mathbf{x})\, d\mathbf{x} \tag{13}$$

A sufficient condition for the plug-in rule to be consistent is given by

$$\lim_{n\to\infty} \int\limits_{\mathbb{R}^d} |\eta_1(\mathbf{x}) - \eta_{1,n}(\mathbf{x})|^{\frac{1}{2}}\, d\mathbf{x} \;=\; 0 \tag{14}$$

2.4.2. *Histogram Rule.* Suppose that $\mathbb{R}^d$ is partitioned into cubes of equal side length r_n. For each point $\mathbf{x} \in \mathbb{R}^d$, the *histogram rule* defines $\psi_n(\mathbf{x})$ to be 0 or 1 according to which is the majority among the labels for points in the cube containing $\mathbf{x}$. If the cubes are defined so that $r_n \to 0$ and $nr_n^d \to \infty$ as $n \to \infty$, then the rule is universally consistent [18].

2.4.3. *Multinomial Discrimination.* The situation in which only a finite number of observed patterns are possible, say $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_m$, is referred to as *multinomial discrimination*. An important example is the so-called *fundamental rule* [19], which assigns at each pattern $\mathbf{z}_i$ the label with the maximum relative frequency among all sample points corresponding to $\mathbf{z}_i$. It can be checked easily that this is the plug-in version of Eq. (3) — for this reason, the fundamental rule is also called the *discrete-data plug-in rule*. The fundamental rule corresponds to a special case of the histogram rule, when the partition used is such that each cell contains exactly one of the possible patterns. For a zero Bayes error and equiprobable patterns, we have that $E[\varepsilon_n] \geq (1 - 1/m)^n$, which shows clearly the effect of using too small a sample. Indeed, if $n \leq m/2$, then the inequality yields $E[\varepsilon_n] \geq 0.5$, which shows that the fundamental rule is useless in this case. In the other direction (for large samples), it is shown in [19] that the fundamental rule is universally consistent and $E[\varepsilon_n] \leq \varepsilon_d + 1.075\sqrt{m/n}$. Multinomial discrimination plays a key role in gene prediction for quantized expression values, in particular, binarized gene expressions in which a gene is qualitatively labeled as ON (1) or OFF (0) [20–22]. In this situation, if there are r binary gene values used to predict a target gene value, then $m = 2^r$ and prediction reduces to multinomial discrimination. Extension to the case of any finite expression quantization is straightforward. This kind of

quantization occurs with discrete gene regulatory networks, in particular, Boolean networks [23, 24]. In a related vein, it has been shown that binarized (ON, OFF) expression values can be used to obtain good classification [25] and clustering [26].

2.4.4. *K-nearest-neighbor Rule.* For the basic *nearest-neighbor rule* (NN), ψ_n is defined for each $\mathbf{x} \in \mathbb{R}^d$ by letting $\psi_n(\mathbf{x})$ take the label of the sample point closest to $\mathbf{x}$. For the NN rule, no matter the feature-label distribution of $(\mathbf{X}, Y)$, $\varepsilon_d \leq \lim_{n\to\infty} E[\varepsilon_n] \leq 2\varepsilon_d$ [27]. It follows that $\lim_{n\to\infty} E[\Delta_n] \leq \varepsilon_d$. Hence, the asymptotic expected design error is small if the Bayes error is small; however, this result does not give consistency. More generally, for the *k-nearest-neighbor rule*, with k odd, the k points closest to $\mathbf{x}$ are selected and $\psi_n(\mathbf{x})$ is defined to be 0 or 1 according to which is the majority among the labels of these points. If $k = 1$, this gives the NN rule. The limit of $E[\varepsilon_n]$ as $n \to \infty$ can be expressed analytically and various upper bounds exist. In particular, $\lim_{n\to\infty} E[\Delta_n] \leq (ke)^{-1/2}$. This does not give consistency, but it does show that the design error gets arbitrarily small for sufficiently large k as $n \to \infty$. The kNN rule is universally consistent if $k \to \infty$ and $k/n \to 0$ as $n \to \infty$ [28].

2.4.5. *Kernel Rules.* The *moving-window rule* takes the majority label among all sample points within a specified distance of $\mathbf{x}$. The rule can be "smoothed" by giving weights to different sample points: the weights associated with the 0- and 1-labeled sample points are added up separately, and the output is defined to be the label with the larger sum. A *kernel rule* is constructed by defining a weighting kernel based on the distance of a sample point from $\mathbf{x}$. The *Gaussian kernel* is defined by $K_h(\mathbf{x}) = e^{-\|\mathbf{x}/h\|^2}$, whereas the *Epanechnikov kernel* is given by $K_h(\mathbf{x}) = 1-||\mathbf{x}/h||^2$ if $||\mathbf{x}|| \leq h$ and $K_h(\mathbf{x}) = 0$ if $||\mathbf{x}|| > h$. If $\mathbf{x}$ is the point at which the classifier is being defined, then the weight at a sample point $\mathbf{x}_k$ is $K_h(\mathbf{x} - \mathbf{x}_k)$. Since the Gaussian kernel is never 0, all sample points get some weight. The Epanechnikov kernel is 0 for sample points at a distance more than h from $\mathbf{x}$, so that, like the moving-window rule, only sample points within a certain radius contribute to the definition of $\psi_n(\mathbf{x})$. The moving-window rule is a special case of a kernel rule with the weights being 1 within a specified radius. The kernel rules we have given are universally consistent [19].

2.4.6. *Linear Classifiers.* For classification rules determined by parametric representation, the classifier is postulated to have a functional form $\psi(x_1, x_2, \ldots, x_d; a_0, a_1, \ldots, a_r)$, where the parameters $a_0, a_1, \ldots, a_r$ are to be determined by some estimation procedure based on the sample data. For parametric representation, we assume the labels to be -1 and 1. The most basic functional form involves a linear combination of the co-ordinates of the observations. A binary function is obtained by thresholding. A *linear classifier*, or *perceptron*, has the form

$$\psi(x) = \mathrm{T}\left[a_0 + \sum_{i=1}^{d} a_i x_i\right] \tag{15}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ and T thresholds at 0 and yields -1 or 1. A linear classifier divides the space into two half-spaces determined by the hyperplane defined by the parameters $a_0, a_1, \ldots, a_d$. The hyperplane is determined by the equation formed from setting the linear combination equal to 0. Using the dot product, $\mathbf{a} \cdot \mathbf{x}$, which is equal to the sum in the preceding equation absent the constant term a_0,

the hyperplane is defined by $\mathbf{a} \cdot \mathbf{x} = -a_0$. Numerous design procedures have been proposed to avoid the computational requirement of full optimization for linear classifiers. Each finds parameters that hopefully define a linear classifier whose error is close to optimal. Often, analysis of the design procedure depends on whether the sample data is *linearly separable*, meaning there exists a hyperplane such that points with label -1 lie on one side of the hyperplane and the points with label 1 lie on the other side. There are many design algorithms for linear classification, each meant to achieve some advantage relative to other methods.

2.4.7. *Support Vector Machines.* The *support vector machine* (*SVM*) provides a method for designing linear classifiers [29]. Fig. 3 shows a linearly-separable data set and three hyperplanes (lines). The outer lines pass through points in the sample data, and the third, called the *maximal-margin hyperplane* (*MMH*) is equidistant between the outer lines. It has the property that the distance from it to the nearest -1-labeled sample point is equal to the distance from it to the nearest 1-labeled sample point. The sample points closest to it are called *support vectors* (the circled sample points in Fig. 3). The distance from the MMH to any support vector is called the *margin*. The matter is formalized by recognizing that differently labeled sets are separable by the hyperplane $\mathbf{u} \cdot \mathbf{x} = c$, where $\mathbf{u}$ is a unit vector and c is a constant, if $\mathbf{u} \cdot \mathbf{x}_k > c$ for $y_k = 1$ and $\mathbf{u} \cdot \mathbf{x}_k < c$ for $y_k = -1$. For any unit vector $\mathbf{u}$, the margin is given by

$$\rho(u) = \frac{1}{2}\left(\min_{\{\mathbf{x}^k:\, y^k=1\}} \mathbf{u} \cdot \mathbf{x}_k \; - \max_{\{\mathbf{x}^k:\, y^k=-1\}} \mathbf{u} \cdot \mathbf{x}_k \right) \tag{16}$$

The MMH, which is unique, can be found by solving the following quadratic optimization problem:

$$\begin{aligned} &\min \, ||\mathbf{v}||, \text{ subject to} \\ &\mathbf{v} \cdot \mathbf{x}_k + b \geq 1, \text{ if } y_k = 1 \\ &\mathbf{v} \cdot \mathbf{x}_k + b \leq -1, \text{ if } y_k = -1 \end{aligned} \tag{17}$$

If $\mathbf{v}_0$ satisfies this optimization problem, then the vector defining the MMH and the margin are given by $\mathbf{u}_0 = \mathbf{v}_0 / ||\, \mathbf{v}_0||$ and $\rho(\mathbf{u}_0) = ||\mathbf{v}_0||^{-1}$, respectively.

If the sample is not linearly separable, then one has two choices: find a reasonable linear classifier or find a nonlinear classifier. In the first case, the preceding method can be modified by making appropriate changes to the optimization problem (17); in the second case, one can map the sample points into a higher dimensional space where they are linearly separable, find a hyperplane in that space, and then map back into the original space (we refer the reader to [29] for the details).

2.4.8. *Quadratic Discriminant Analysis.* Let R_k denote the region in $\mathbb{R}^d$ where the Bayes classifier has the value k, for $k = 0,1$. According to Eq. (3), $\mathbf{x} \in R_k$ if $\eta_k(\mathbf{x}) > \eta_j(\mathbf{x})$, for $j \neq k$ (ties in the posteriors being broken arbitrarily). Since $\eta_k(\mathbf{x}) = f_{\mathbf{X}|Y}(\mathbf{x}|k) f_Y(k) / f_{\mathbf{X}}(\mathbf{x})$, upon taking the logarithm and discarding the common term $f_{\mathbf{X}}(\mathbf{x})$, this is equivalent to $\mathbf{x} \in R_k$ if $d_k(\mathbf{x}) > d_j(\mathbf{x})$, where the *discriminant* $d_k(\mathbf{x})$ is defined by

$$d_k(\mathbf{x}) = \log f_{\mathbf{X}|Y}(\mathbf{x}|k) + \log f_Y(k) \tag{18}$$

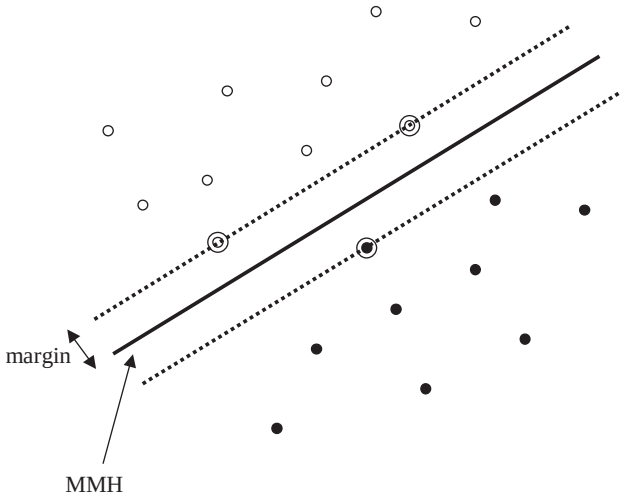

FIGURE 3. Maximal-margin hyperplane for linearly-separable data. The support vectors are the circled sample points.

If the conditional densities $f_{\mathbf{X}|Y}(\mathbf{x}|0)$ and $f_{\mathbf{X}|Y}(\mathbf{x}|1)$ are normally distributed, then

$$f_{\mathbf{X}|Y}(\mathbf{x}|k) = \frac{1}{\sqrt{(2\pi)^n \det[\mathbf{K}_k]}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mathbf{u}_k)'\mathbf{K}_k^{-1}(\mathbf{x}-\mathbf{u}_k)\right] \tag{19}$$

where $\mathbf{K}_k$ and $\mathbf{u}_k$ are the covariance matrix and mean vector for class k, respectively. Dropping the constant terms and multiplying by the factor 2 (which has no effect on classification), the discriminant becomes

$$d_k(\mathbf{x}) = -(\mathbf{x}-\mathbf{u}_k)'\mathbf{K}_k^{-1}(\mathbf{x}-\mathbf{u}_k) - \log(\det[\mathbf{K}_k]) + 2\log f_Y(k) \tag{20}$$

Hence, the discriminant is quadratic in $\mathbf{x}$. The first term in (20) is known as the *Mahalanobis distance* between $\mathbf{x}$ and $\mathbf{u}_k$. A simple calculation shows that the optimal decision boundary $d_1(\mathbf{x}) - d_0(\mathbf{x}) = 0$ is given by

$$\begin{aligned} &\mathbf{x}'\left(\mathbf{K}_0^{-1} - \mathbf{K}_1^{-1}\right)\mathbf{x} \;-\; 2\left(\mathbf{u}_0'\mathbf{K}_0^{-1} - \mathbf{u}_1'\mathbf{K}_1^{-1}\right)\mathbf{x} + \\ &+\; \mathbf{u}_0'\mathbf{K}_0^{-1}\mathbf{u}_0 - \mathbf{u}_1'\mathbf{K}_1^{-1}\mathbf{u}_1 + \log\left(\frac{\det[\mathbf{K}_0]}{\det[\mathbf{K}_1]}\right) + 2\log\left(\frac{f_Y(1)}{f_Y(0)}\right) = 0. \end{aligned} \tag{21}$$

This is an equation in the form $\mathbf{x}'\mathrm{A}\mathbf{x} - \mathbf{b}'\mathbf{x} + c = 0$. In 2-dimensional space, such an equation produces *conical-section* decision curves, whereas in 3-dimensional spaces, it produces decision surfaces known as *quadrics*. Plugging sample-based estimates for the covariance matrices, mean vectors, and priors into (21), leads to a classification rule known as *quadratic discriminant analysis* (*QDA*). Depending on the estimated coefficients in (21), decision boundaries ranging from paraboloids to spheres can be produced by QDA.

2.4.9. *Linear Discriminant Analysis.* If both conditional densities possess the same covariance matrix $\mathbf{K}$, then the quadratic term and the first logarithmic term vanish

in (21), yielding

$$(\mathbf{u}_1 - \mathbf{u}_0)'\mathbf{K}^{-1}\mathbf{x} - \frac{1}{2}\left(\mathbf{u}_1'\mathbf{K}^{-1}\mathbf{u}_1 - \mathbf{u}_0'\mathbf{K}^{-1}\mathbf{u}_0\right) + \log\left(\frac{f_Y(1)}{f_Y(0)}\right) = 0. \tag{22}$$

This is an equation in the form $\mathbf{a}\mathbf{x}' + m = 0$. Such equations produces decision surfaces that are hyperplanes in d-dimensional space. Plugging into (22) sample-based estimates for the covariance matrix, mean vectors, and priors, leads to a classification rule known as *linear discriminant analysis* (*LDA*). In practice, the usual maximum-likelihood estimates are employed for the mean vectors, whereas the estimate of the covariance matrix is often given by the *pooled covariance matrix*:

$$\widehat{\mathbf{K}} = \frac{1}{2}(\widehat{\mathbf{K}}_0 + \widehat{\mathbf{K}}_1), \tag{23}$$

where $\widehat{\mathbf{K}}_k$ is the usual maximum-likelihood estimate of the covariance matrix of class k (note that, in general, $\widehat{\mathbf{K}}_0 \neq \widehat{\mathbf{K}}_1$). In addition, especially in the case of small sample sizes, it is common practice to assume equally-likely classes, so that term $\log(f_Y(1)/f_Y(0))$ is zero. This avoids the use of unreliable estimates of the priors derived from limited data.

2.4.10. *Nearest-Mean Classifier.* If, besides a common covariance matrix and equally-likely classes, one assumes uncorrelated conditional distributions, with covariance matrix $\mathbf{K} = \sigma^2\mathbf{I}$, then Eq. (22) reduces to

$$(\mathbf{u}_1 - \mathbf{u}_0)'\mathbf{x} - \frac{1}{2}\left(||\mathbf{u}_1||^2 - ||\mathbf{u}_0||^2\right) = 0. \tag{24}$$

The optimal hyperplane in this case is perpendicular to the line joining the means and passes through the midpoint of that line. Therefore, a sample point is assigned to class k if its distance to the mean vector $\mathbf{u}_k$ is minimal. This also follows from the fact that the discriminant function in (20) can be written in this case simply as $d_k(\mathbf{x}) = -||\mathbf{x}-\mathbf{u}_k||$. Substituting sample-based mean estimates for the mean vectors in (24) leads to the *nearest-mean classifier* (*NMC*). This classification rule has the advantage of avoiding the estimation (and inversion) of the covariance matrices, so it can be effective in extreme small-sample scenarios.

Equations (21), (22) and (24), for the QDA, LDA, and NMC rules, respectively, were derived under the Gaussian assumption, but in practice can perform well so long as the underlying class conditional densities are approximately Gaussian — and one can obtain good estimates of the relevant covariance matrices. Owing to the greater number of parameters to be estimated for QDA as opposed to LDA and NMC, one can proceed with smaller samples for LDA than with QDA, and in extreme small-sample cases, NMC may be the most effective choice, due to its avoiding the estimation of the covariance matrices. Of course, if the assumption of equal and/or uncorrelated covariance matrices does not hold, then the LDA and NMC rules will have asymptotic expected error biased away from the Bayes error. Therefore, in large-sample scenarios, QDA is preferable. However, LDA has been reported to be more robust relative to the underlying Gaussian assumption than QDA [30]. In our experience, LDA has proved to be a very robust classification rule (see Section 2.5), which is effective for a wide range of sample sizes.

2.4.11. *Neural Networks.* A (feed-forward) two-layer *neural network* has the form

$$\psi(\mathbf{x}) = \mathrm{T}\left[c_0 + \sum_{i=1}^{k} c_i \sigma[\psi_i(\mathbf{x})]\right] \tag{25}$$

where T thresholds at 0, σ is a *sigmoid function* (i.e., a nondecreasing function with limits -1 and $+1$ at $-\infty$ and ∞, respectively), and

$$\psi_i(\mathbf{x}) = a_{i0} + \sum_{j=1}^{d} a_{ij} x_j \tag{26}$$

Each operator in the sum of Eq. (25) is called a *neuron*. These form the hidden layer. We consider neural networks with the threshold sigmoid: $\sigma(x) = -1$ if $x \leq 0$ and $\sigma(x) = 1$ if $x > 0$. If $k \to \infty$ such that $(k \log n)/n \to 0$ as $n \to \infty$, then, as a class, neural networks are universally consistent [31], but one should beware of the increasing number of neurons required.

A key point here is that any function whose absolute value possesses finite integral can be approximated arbitrarily closely by a sufficiently complex neural network. While this is theoretically important, there are limitations to its practical usefulness. Not only does one not know the function, in this case the Bayes classifier, whose approximation is desired, but even were we to know the function and how to find the necessary coefficients, a close approximation can require an extremely large number of model parameters. Given the neural-network structure, the task is to estimate the optimal weights. As the number of model parameters grows, use of the model for classifier design becomes increasingly intractable owing to the increasing amount of data required for estimation of the model parameters. Since the number of hidden units must be kept relatively small, thereby requiring significant constraint, when data are limited, there is no assurance that the optimal neural network of the prescribed form closely approximates the Bayes classifier. Model estimation is typically done by some iterative procedure, with advantages and disadvantages being associated with different methods [32].

2.4.12. *Classification Trees.* The histogram rule partitions the space without reference to the actual data. One can instead partition the space based on the data, either with our without reference to the labels. Tree classifiers are a common way of performing data-dependent partitioning. Since any tree can be transformed into a binary tree, we need only consider binary classification trees. A tree is constructed recursively based on some criteria. If S represents the set of all data, then it is partitioned according to some rule into $S = S_1 \cup S_2$. There are four possibilities: (i) S_1 is partitioned into $S_1 = S_{11} \cup S_{12}$ and S_2 is partitioned into $S_2 = S_{21} \cup S_{22}$; (ii) S_1 is partitioned into $S_1 = S_{11} \cup S_{12}$ and partitioning of S_2 is terminated; (iii) S_2 is partitioned into $S_2 = S_{21} \cup S_{22}$ and partitioning of S_1 is terminated; and (iv) partitioning of both S_1 and S_2 is terminated. In the last case, the partition is complete; in any of the others, it proceeds recursively until all branches end in termination, at which point the leaves on the tree represent the partition of the space. On each cell (subset) in the final partition, the designed classifier is defined to be 0 or 1, according to which is the majority among the labels of the points in the cell.

A wide variety of classification trees whose leaves are rectangles in $\mathbb{R}^d$ can be obtained by perpendicular splits. At each stage of growing the tree, a decision to

split a rectangle R is made according to a coordinate decision of the form $x_i^j \leq \alpha$, where $\mathbf{x}^j = (x_1^j, x_2^j, ..., x_d^j)$ is a sample point in $\mathbb{R}^d$. Also at each stage, there are two collections of rectangles, R_0 and R_1, determined by majority vote of the labels, so that $R \in \mathrm{R}_1$ if and only if the majority of labels for points in R have value 1. The 0 and 1 decision regions are determined by the unions of rectangles in R_0 and R_1, respectively. A final classification tree, and therefore the designed classifier, depends on the splitting criterion, choice of α, and a stopping criterion. Two desirable attributes of a stopping criterion are that the leaf nodes (final rectangles) be small in number so that the complexity of the classifier be not too great for the amount of data (thus avoiding overfitting), and that the labels in each final rectangle be not evenly split, thereby increasing the likelihood that the majority label accurately reflects the distribution in the rectangle. A rectangle is said to be *pure* relative to a particular sample if all labels corresponding to points in the rectangle possess the same label.

One popular method of splitting, which goes under the name *Classification and Regression Trees* (*CART*), is based on the notion of an *impurity function*. For any rectangle R, let $N_0(R)$ and $N_1(R)$ be the numbers of 0-labeled and 1-labeled points, respectively, in R, and let $N(R) = N_0(R) + N_1(R)$ be the total number of points in R. The *impurity* of R is defined by

$$\kappa(R) = \xi(p_R, 1 - p_R) \tag{27}$$

where $p_R = N_0(R)/N(R)$ is the proportion of 0 labels in R, and where $\xi(p, 1-p)$ is a nonnegative function satisfying the following conditions: (1) $\xi(0.5, 0.5) \geq \xi(p, 1-p)$ for any $p \in [0, 1]$; (2) $\xi(0, 1) = \xi(1, 0) = 0$; and (3) as a function of p, $\xi(p, 1-p)$ increases for $p \in [0, 0.5]$ and decreases for $p \in [0.5, 1]$. Several observations follow from the definition of ξ: (1) $\kappa(R)$ is maximum when the proportions of 0-labeled and 1-labeled points in R are equal (corresponding to maximum impurity); (2) $\kappa(R) = 0$ if R is pure; and (3) $\kappa(R)$ increases for greater impurity.

We mention three possible choices for ξ:

(1) $\xi_e(p, 1-p) = -p \log p - (1-p) \log(1-p)$ (*entropy impurity*)
(2) $\xi_g(p, 1-p) = p(1-p)$ (*Gini impurity*)
(3) $\xi_m(p, 1-p) = \min(p, 1-p)$ (*misclassification impurity*)

The origins of these three impurities lie in the definition of $\kappa(R)$: $\xi_e(p, 1-p)$ provides an entropy estimate, $\xi_g(p, 1-p)$ provides a variance estimate for a binomial distribution, and $\xi_m(p, 1-p)$ provides a misclassification-rate estimate.

A splitting regimen is determined by the manner in which a split will cause an overall decrease in impurity. Let i be a coordinate, α be a real number, R be a rectangle to be split along the i-th coordinate, $R^i_{\alpha,-}$ be the sub-rectangle resulting from the ith coordinate being less than or equal to α, and $R^i_{\alpha,+}$ be the sub-rectangle resulting from the ith coordinate being greater than α. Define the *impurity decrement* by

$$\Delta_i(R, \alpha) = \kappa(R) - \frac{N(R^i_{\alpha,-})}{N(R)} \kappa(R^i_{\alpha,-}) - \frac{N(R^i_{\alpha,+})}{N(R)} \kappa(R^i_{\alpha,+}) \tag{28}$$

A good split will result in impurity reductions in the sub-rectangles. In computing $\Delta_i(R, \alpha)$, the new impurities are weighted by the proportions of points going into the sub-rectangles. CART proceeds iteratively by splitting a rectangle at $\hat{\alpha}$ on the ith coordinate if $\Delta_i(R, \alpha)$ is maximized for $\alpha = \hat{\alpha}$. There are two possible

splitting strategies: (i) the coordinate i is given and $\Delta_i(R,\, \alpha)$ is maximized over all α and R; (ii) the coordinate is not given and $\Delta_i(R,\, \alpha)$ is maximized over all i, α, and R. Various stopping strategies are possible — for instance, stopping when maximization of $\Delta_i(R,\, \alpha)$ yields a value below a preset threshold, or when there are fewer than a specified number of sample points assigned to the node. One may also continue to grow the tree until all leaves are pure and then prune.

2.5. **Classification Performance.** In this section, we present classification results obtained with real patient data. Our purpose is to compare the performance of several of the classification rules described in the previous sections, in terms of the expected classification error, for different sample sizes and number of variables (dimensionality).

The data used in the experiments come from a microarray-based classification study [15] that analyzed a large number of microarrays, prepared with RNA from breast tumor samples from each of 295 patients (see Fig. 1 for a plot of the expression values of two genes in these data). Using a previously established 70-gene prognosis profile [33], a prognosis signature based on gene-expression was proposed in [15], which correlated well with patient survival data and other existing clinical measures. Of the $N = 295$ microarrays, $N_0 = 115$ belong to the "good-prognosis" class, whereas the remaining $N_1 = 180$ belong to the "poor-prognosis" class.

Our experiments were set up in the following way. We used log-ratio gene expression values associated with the top genes found in [33]. We consider four basic cases, corresponding to $d = 2, 3, 4, 5$ genes. In each case, we searched the best combination, in terms of estimated Bayes error, of d genes among the top 10 genes, with the purpose of not considering situations where there is too much confusion between the classes, which makes the expected errors excessively large. The Bayes error was computed by using (4) in conjunction with a Gaussian-kernel density estimation method, for which the kernel variance is automatically selected by a pseudolikelihood-based technique [34].

In each case, 1000 observations S_n of size ranging from $n = 20$ to $n = 120$, in steps of 5, were drawn independently from the pool of 295 microarrays. Sampling was stratified in the sense that the proportion of each class in the random sample was fixed to N_0/N for the first class and N_1/N for the second class. A classifier was designed for each sample S_n, using one of the classification rules described previously, and its classification error was approximated by means of a holdout estimator (see Section 4.1), whereby the $295 - n$ sample points not drawn are used as an independent test set (this is a good approximation to the true error, given the large test sample). The errors for the 1000 independent sample sets were averaged to provide a Monte-Carlo estimate of the expected error for the classification rule.

Fig. 4 displays four plots, one for each dimensionality considered. We have considered seven classification rules: linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), nearest-mean classification (NMC), 1-nearest neighbor (1NN), 3-nearest neighbor (3NN), CART with a stopping rule that ends splitting when there are six or fewer sample points in a node, and a neural network (NNET) with 4 nodes in the hidden layer.

Confirming observations we have made previously, we can see that LDA performs quite well, and so does 3NN. We see that QDA does a very good job for larger sample sizes, but its performance degrades quickly for smaller sample sizes. NMC does a very credible job, given its simplicity, and it can actually do quite well for

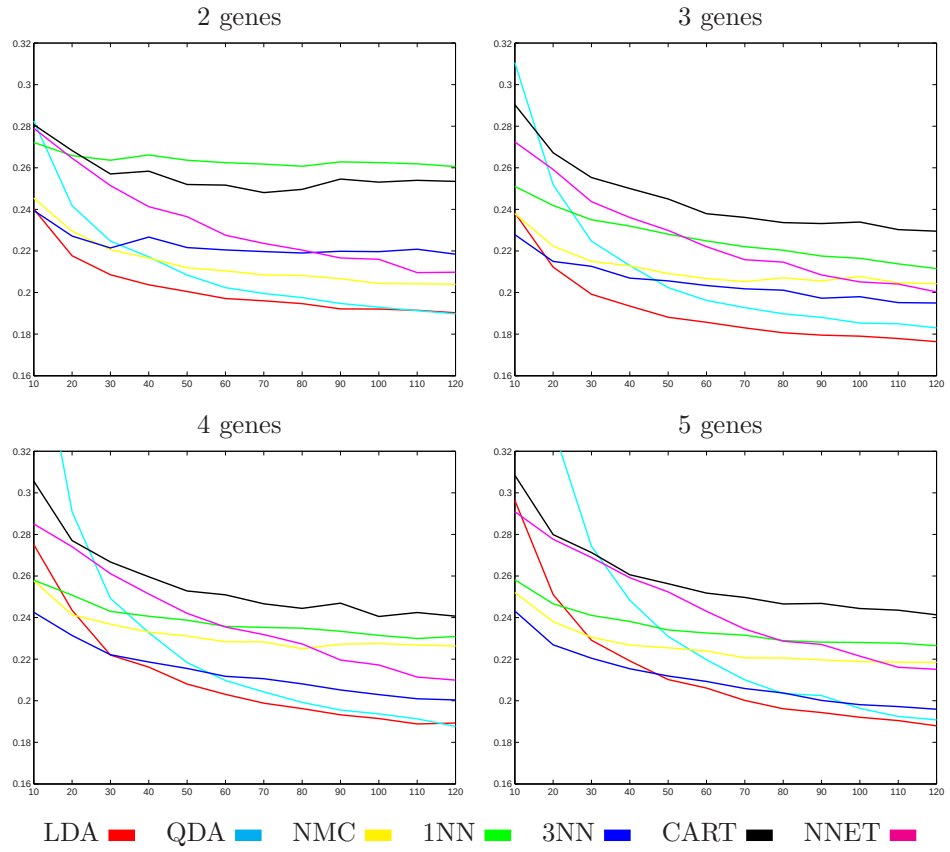

FIGURE 4. Expected error versus sample size for several classification rules and number of genes.

very small sample sizes, as compared to the other classification rules. The neural network performed well for 2 variables, but its performance quickly degrades as the number of genes increases, which can be explained by the high complexity of this classification rules, which leads to overfitting. CART and 1NN do not perform well with this data set, due to severe overfitting (even with the regularizing stopping criterion used for CART).

3. Regularization

Thus far we have taken the perspective that a collection of features is given, sample data is obtained, and a classifier based on the features is designed from the data via a classification rule. The feature set and sample data are taken as given, and the designer selects a classification rule. In this section we consider alterations to this paradigm in order to deal with the small-sample problem, more specifically, a sample that is small relative to the number of features and classifier complexity. These methods fall under the general category of *regularization*.

3.1. **Regularized Discriminant Analysis.** The small-sample problem for quadratic discriminant analysis (QDA) can be appreciated by considering the spectral

decompositions of the covariance matrices,

$$\mathbf{K}_k = \sum_{j=1}^{d} \lambda_{kj} \mathbf{v}_{kj} \mathbf{v}'_{kj} \tag{29}$$

where λ_{k1}, λ_{k2},..., λ_{kd} are the eigenvalues of $\mathbf{K}_k$ in decreasing order and $\mathbf{v}_{kj}$ is the eigenvector corresponding to λ_{kj}. Then it can be shown that the quadratic discriminant of Eq. (20) takes the form

$$d_k(\mathbf{x}) \;=\; -\sum_{j=1}^{d} \frac{[\mathbf{v}_{kj}(\mathbf{x}-\mathbf{u}_k)]^2}{\lambda_{kj}} \;-\; \sum_{j=1}^{d} \log \lambda_{kj} \;+\; 2\log f_Y(k) \tag{30}$$

The discriminant is strongly influenced by the smallest eigenvalues. This creates a difficulty because the large eigenvalues of the sample covariance matrix are biased high and the small eigenvalues are biased low — and this phenomenon is accentuated for small samples.

Relative to QDA, a simple method of regularization is to apply LDA even though the covariance matrices are not equal. This means estimating a single covariance matrix by pooling the data. This reduces the number of parameters to be estimated and increases the sample size relative to the smaller set of parameters. Generally, regularization reduces variance at the cost of bias, and the goal is substantial variance reduction with negligible bias.

A softer approach than strictly going from QDA to LDA is to shrink the individual covariance estimates in the direction of the pooled estimate. This can be accomplished by introducing a parameter α between 0 and 1 and using the estimates

$$\hat{\mathbf{K}}_k(\alpha) = \frac{n_k(1-\alpha)\hat{\mathbf{K}}_k + n\alpha\hat{\mathbf{K}}}{n_k(1-\alpha)+n\alpha} \tag{31}$$

where n_k is the number of points corresponding to $Y = k$, $\hat{\mathbf{K}}_k$ is the sample covariance matrix for class k, and $\hat{\mathbf{K}}$ is the pooled estimate of the covariance matrix. QDA results from $\alpha = 0$ and LDA from $\alpha = 1$, with different amounts of shrinkage occurring for $0 < \alpha < 1$ [35]. While reducing variance, one must be prudent in choosing α, especially when the covariance matrices are very different.

To get more regularization while not overly increasing bias, one can shrink the regularized sample covariance matrix $\hat{\mathbf{K}}_k(\alpha)$ towards the identity multiplied by the average eigenvalue of $\hat{\mathbf{K}}_k(\alpha)$. This has the effect of decreasing large eigenvalues and increasing small eigenvalues, thereby offsetting the biasing effect seen in Eq. (30) [36]. Thus, we consider the estimate

$$\hat{\mathbf{K}}_k(\alpha,\beta) = (1-\beta)\hat{\mathbf{K}}_k(\alpha) + \frac{\beta}{n}\mathrm{tr}[\hat{\mathbf{K}}_k(\alpha)]\mathbf{I} \tag{32}$$

where $\mathrm{tr}[\hat{\mathbf{K}}_k(\alpha)]$ is the trace of $\hat{\mathbf{K}}_k(\alpha)$, $\mathbf{I}$ is the identity, and $0 \le \beta \le 1$. To apply this *regularized discriminant analysis* using $\hat{\mathbf{K}}_k(\alpha,\beta)$ requires selecting two model parameters. Model selection is critical to advantageous regularization, and typically is problematic; nevertheless, simulation results for Gaussian conditional distributions indicate significant benefit of regularization for various covariance models, and very little increase in error even in models where it does not appear to help.

3.2. **Noise Injection.** Rather than regularizing the estimated covariance matrix, one can regularize the data itself by *noise injection*. This can be done by "spreading" the sample data, by means of synthetic data generated about each sample point, thereby creating a large synthetic sample from which to design the classifier while at the same time making the designed classifier less dependent on the specific points in the small data set. For instance, one may place a circular Gaussian distribution at each sample point, randomly generate points from each distribution, and then apply a classification rule. Such a Monte-Carlo approach has been examined relative to LDA [37]. A spherical distribution need not be employed. Indeed, it has been demonstrated that it can be advantageous to base noise injection at a sample point based on the nearest neighbors of the point [37]. This kind of noise injection is not limited to any particular classification rule; however, it can be posed analytically in terms of matrix operations for linear classification, and this is critical to situations in which a large number of feature sets must be examined, in particular, microarray-based classification, where a vast number of potential feature sets are involved [9].

Noise injection can take a different form in which the sample data points themselves are perturbed by additive noise instead of new synthetic points being generated. This approach has been used in designing neural networks (of which linear classifiers are a special case), in particular, where owing to a small sample, the same data points are used repeatedly [38].

3.3. **Error Regularization.** Rather than considering a single class from which to choose a classifier, one can consider a sequence of classes, $\mathcal{C}_1, \mathcal{C}_2, \ldots$, find the best classifier in each class according to the data, and then choose among these according to which class is of appropriate complexity for the sample size. For instance, one might assume a nested sequence, $\mathcal{C}_1 \subset \mathcal{C}_2 \subset \ldots$ The idea is to define a new measurement that takes into account both the error estimate of a designed classifier and the complexity of the class from which it has been chosen — the more complex the class, the larger the penalty. In this vein, we define a new *penalized error* that is a sum of the estimated error and a complexity penalty $\rho(n)$,

$$\tilde{\varepsilon}_n[\psi] = \hat{\varepsilon}_n[\psi] + \rho(n). \tag{33}$$

Relative to the constraint sequence $\{\mathrm{C}^j\}$, *structural risk minimization* proceeds by selecting the classifier in each class that minimizes the empirical error over the sample, and then choosing among these the one possessing minimal penalized error, where in each case the penalty is relative to the class containing the classifier [39, 40].

Minimum-description-length complexity regularization replaces error minimization by minimization of a sum of entropies, one relative to encoding the error and the other relative to encoding the classifier description, in an effort to balance increased error and increased model complexity [41, 42]. The MDL approach has been employed for microarray-based prediction [22, 43].

3.4. **Feature Selection.** The feature-selection problem is to select a subset of k features from a set of n features that provides an optimal classifier with minimum error among all optimal classifiers for subsets of a given size. For instance, for the large number of expression measurements on a cDNA microarray, it is necessary to find a small subset with which to classify. The inherent combinatorial nature of the

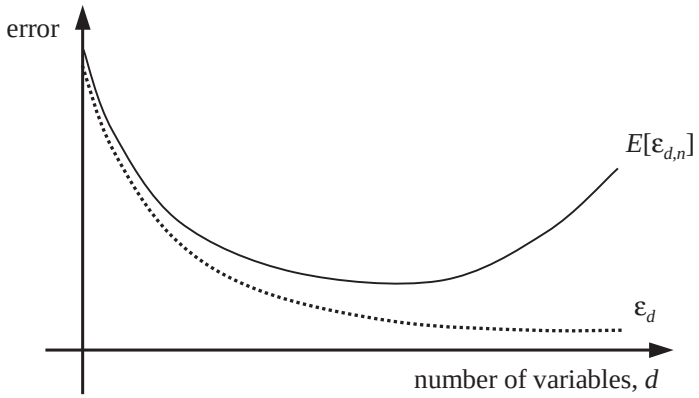

FIGURE 5. Bayes error and expected error versus number of features.

problem is readily seen from the fact that all k-element subsets must be checked to assure selection of the optimal k-element feature set [44].

An issue concerning error estimation is monotonicity of the error measure. The Bayes error is monotone: if A and B are feature sets for which $A \subset B$, then $\varepsilon_B \leq \varepsilon_A$, where ε_A and ε_B are the Bayes errors corresponding to A and B, respectively. However, if $\varepsilon_{A,n}$ and $\varepsilon_{B,n}$ are the corresponding errors resulting from designed classifiers on a sample of size n, then it cannot be asserted that $E[\varepsilon_{B,n}]$ does not exceed $E[\varepsilon_{A,n}]$. Indeed, it is typical to observe a "peaking phenomenon" for fixed sample size, whereby the expected error decreases at first and then increases, for increasingly large feature sets. Thus, monotonicity does not apply for the expected error. This is illustrated in Fig. 5, where the Bayes error ε_d and the expected error $E[\varepsilon_{d,n}]$ of the designed filter are plotted against the number of variables d. We can see that ε_d decreases, whereas $E[\varepsilon_{d,n}]$ decreases and then increases. We remark that the peaking phenomenon is referred to by some authors as the *Hughes phenomenon* [45, 46]. Note that, were $E[\varepsilon_{d,n}]$ known, then we could conclude that ε_d is no worse than $E[\varepsilon_{d,n}]$; however, we have only estimates of the error $\varepsilon_{d,n}$, which for small samples can be above or below ε_d.

A full exhaustive search can be mitigated by using a branch and bound feature-selection algorithm that takes advantage of the monotonicity property of the Bayes error [47]. If $A \subset B$ and C is a feature set for which $\varepsilon_C \geq \varepsilon_A$, then $\varepsilon_C \geq \varepsilon_B$. In principle, this approach yields an optimal solution; however, it suffers from two problems. First, worst-case performance can be exponentially complex, thereby making its use less attractive for very large feature sets; and second, estimation of the Bayes error must be used and therefore monotonicity is lost, a problem that is exacerbated by small samples. As is generally true with feature-selection methods, other criteria besides the Bayes error can be used to select features, monotonic criteria being necessary for strict application of the branch and bound algorithm. Even with the loss of monotonicity, the branch-and-bound approach may still provide good results [48].

When considering a large collection of features, the branch-and-bound technique is not sufficiently computationally efficient and suboptimal approaches need to be

considered. The most obvious approach is to consider each feature by itself and choose the k features that perform individually the best. While easy, this method is subject to choosing a feature set with a large number of redundant features, thereby obtaining a feature set that is much worse than the optimal. Moreover, features that perform poorly individually may do well in combination with other features [9].

Perhaps the most common approach to suboptimal feature selection is *sequential selection*, either forward or backward, and their variants. Forward selection begins with a small set of features, perhaps one, and iteratively builds the feature set. Backward selection starts with a large set and iteratively reduces it. Owing to simpler calculations, forward selection is generally faster, and we will restrict our attention to it, noting that analogous comments apply to backwards selection. Here again, monotonicity issues and the "peaking" phenomenon arise: adjoining variables stepwise to the feature vector decreases the Bayes error but can increase errors of the designed filters.

Being more precise relative to forward selection, if A is the feature set at a particular stage of the algorithm and Q is the full set of potential features, then all sets of the form $A \cup \{X\}$ are considered, with $X \in Q - A$, and the feature X is adjoined to A if it is the one for which the error $\varepsilon[A \cup \{X\}]$ is minimal. An obvious problem is that once a feature is in the growing feature set, it cannot be removed. This problem can be handled by adjoining two or more features by considering sets of the form $A \cup B$, where B is a subset of $Q - A$ possessing b features, and then deleting features by considering sets of the form $A \cup B_0 - C$, where B_0 has been chosen on the adjoining part of the iteration and C is a subset of $A \cup B_0$ possessing $c < b$ features. At each stage of the iteration the feature set is grown by $b - c$ features. While growing by adjoin-delete iterations is superior to just adjoining, there is still inflexibility owing to the *a priori* choices of b and c. Flexibility can be added to sequential forward selection by considering *sequential forward floating selection* (*SFFS*), where the number of features to be adjoined and deleted is not fixed, but is allowed to "float" [49].

When there is a large number of potential random variables for classification, feature selection is problematic and the best method to use depends on the circumstances. Evaluation of methods is generally comparative and based on simulations [50].

3.5. **Feature Extraction.** Rather than reduce dimensionality by selecting from the original features, one might take the approach of *feature extraction*, where a transform is applied to the original features to map them into a lower dimensional space. Since the new features involve a transform of the original features, the original features remain (although some may be eliminated by compression) and are still part of the classification process. A disadvantage of feature extraction is that the new features lack the physical meaning of the original features – for instance, gene-expression levels. A potential advantage of feature extraction is that, given the same number of reduced features, the transform features may provide better classification than selected individual features. Perhaps the most common form of feature extraction is *principal component analysis* (PCA).

Consider the (random) observation vector $\mathbf{X} = (X_1, X_2, \ldots, X_n)$, where the observations have been normalized to have zero means. Since the covariance matrix

$\mathbf{K}$ is symmetric, if $\lambda_1$and $\lambda_2$are distinct eigenvalues, then their respective eigenvectors will be orthogonal and the desired orthonormal eigenvectors can be found by dividing each by its own magnitude. On the other hand, if an eigenvalue has repeated eigenvectors, then these will be linearly independent and an algebraically equivalent set can be found by the Gram-Schmidt orthogonalization procedure.

According to the Karhunen-Loeve theorem, if the vectors $\mathbf{u}_1$, $\mathbf{u}_2$,..., $\mathbf{u}_n$ are the orthonormalized eigenvectors of $\mathbf{K}$ corresponding to the eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$, then

$$\mathbf{X} = \sum_{i=1}^{n} Z_i \mathbf{u}_i \tag{34}$$

where Z_1, Z_2,..., Z_n are uncorrelated and given by $Z_i = \mathbf{X} \cdot \bar{\mathbf{u}}_i$. The values Z_1, Z_2,..., Z_n are called the *principal components* for $\mathbf{X}$. For $m < n$, data compression is achieved by approximating $\mathbf{X}$ by

$$\mathbf{X}_m = \sum_{i=1}^{m} Z_i \mathbf{u}_i \tag{35}$$

The *mean-square error* between $\mathbf{X}$ and $\mathbf{X}_m$ is given by

$$\mathrm{E}[\mathbf{X}, \mathbf{X}_m] = \sum_{k=1}^{n} E[|X_k - X_{m,k}|^2] \tag{36}$$

where the components of $\mathbf{X}_m$ are $X_{m,1}$, $X_{m,2}$,..., $X_{m,n}$. It can be shown that

$$\mathrm{E}[\mathbf{X}, \mathbf{X}_m] = \sum_{k=m+1}^{n} \lambda_k \tag{37}$$

Since the eigenvalues are decreasing with increasing k, the error is minimized by keeping the first m terms. To apply PCA for the purposes of feature extraction, $Z_1, Z_2, \ldots, Z_m$ are employed.

4. Error Estimation

Error estimation is key aspect of classification, as it impacts both classifier design and variable selection. Recall that the performance measure of a designed classifier is the "true" error ε_n, whereas the performance measure of a classification rule (for fixed sample size n) is the expected error $E[\varepsilon_n]$. However, both these quantities can only be computed exactly if one knows the feature-label distribution for the classification problem. Since in practice such knowledge is rarely, if ever, at hand, one needs to estimate the true error from the available sample data.

An error estimator $\hat{\varepsilon}_n$ may be a deterministic function of the sample data S_n, in which case it is a *non-randomized error estimator*. Such an error estimator is random only through the random sample. Among popular non-randomized error estimators, we have resubstitution, leave-one-out, and fixed-fold cross-validation. By contrast, *randomized error estimators* have "internal" random factors that affect their outcome. Popular randomized error estimators include random-fold cross-validation and all bootstrap error estimators (all aforementioned error estimators will be discussed in detail below).

A key feature that often dictates the performance of an error estimator is its variance, especially in small-sample settings. The *internal variance* of an error estimator is the variance due only to its internal random factors, $V_{\mathrm{int}} = \mathrm{Var}(\hat{\varepsilon}_n | S_n)$.

This variance is zero for non-randomized error estimators. The full variance $\mathrm{Var}(\hat{\varepsilon}_n)$ of the error estimator is the one we are really concerned about, since it takes into account the uncertainty introduced by the random sample data. Using the well-known conditional-variance formula, $\mathrm{Var}(X) = E[\mathrm{Var}(X|Y)] + \mathrm{Var}(E[X|Y])$ [51], one can break down $\mathrm{Var}(\hat{\varepsilon}_n)$ as:

$$\mathrm{Var}(\hat{\varepsilon}_n) = E[V_{\mathrm{int}}] + \mathrm{Var}(E[\hat{\varepsilon}_n|S_n]). \tag{38}$$

The second term on the right-hand side is the one that includes the variability due to the random sample. Note that, for non-randomized $\hat{\varepsilon}_n$, we have $V_{\mathrm{int}} = 0$ and $E[\hat{\varepsilon}_n|S_n] = \hat{\varepsilon}_n$. For randomized error estimators, the first term on the right-hand side has to be made small through intensive computation, in order to achieve small overall estimator variance. This is one of the reasons why randomized error estimators are typically very inefficient computationally, as we will see below.

4.1. **Holdout Estimation.** We now proceed to discuss specific error estimation techniques. If there is an abundance of sample data, then it can be split into *training data* and *test data*. A classifier is designed on the training data, and its estimated error is the proportion of errors it makes on the test data. We denote this test-data error estimate by $\hat{\varepsilon}_{n,m}$, where m is the number of sample pairs in the test data. Since the test data is random and independent from the training data, this is a randomized error estimator. It is unbiased in the sense that, given the training data S_n, $E[\hat{\varepsilon}_{n,m}|S_n] = \varepsilon_n$, and thus $E[\hat{\varepsilon}_{n,m}] = E[\varepsilon_n]$. The internal variance of this estimator can be bounded as follows [19]:

$$V_{\mathrm{int}} = E[(\hat{\varepsilon}_{n,m} - \varepsilon_n)^2|S_n] \leq \frac{1}{4m} \tag{39}$$

which tends to zero as $m \to \infty$. Moreover, by using (38), we get that the full variance of the holdout estimator is simply

$$\mathrm{Var}(\hat{\varepsilon}_{n,m}) = E[V_{\mathrm{int}}] + \mathrm{Var}[\varepsilon_n]. \tag{40}$$

Thus, provided that m is large, so that V_{int} is small (this is guaranteed by (39) for large enough m), the variance of the holdout estimator is approximately equal to the variance of the true error itself, which is typically small, provided n is not too small.

The problem with using both training and test data is that, in practice, one often does not usually have available a large enough data set to be able to make both n and m large enough. For example, in order to get the standard-deviation bound in Eq. (39) down to an acceptable level, say 0.05, it would be necessary to use 100 test samples. On the other hand, data sets that contain fewer than 100 overall samples are quite common. Therefore, for a large class of practical problems where samples are at a premium, holdout error estimation is effectively ruled out. In such cases, one must use the same data for training and testing.

4.2. **Resubstitution.** One approach is to use all sample data to design a classifier ψ_n, and estimate ε_n by applying ψ_n to the same data. The *resubstitution* estimate, $\hat{\varepsilon}_n^R$, is the fraction of errors made by ψ_n on the training data:

$$\hat{\varepsilon}_n^R = \frac{1}{n}\sum_{i=1}^{n} |Y_i - \psi_n(\mathbf{X}_i)| \tag{41}$$

This is none other than the apparent error rate mentioned in Section 2.3. Resubstitution is usually biased low, meaning $E[\hat{\varepsilon}_n^R] \leq E[\varepsilon_n]$ — but not always. For fixed-partition histogram rules, meaning those that are independent of the sample size and the data, the resubstitution error estimate is biased low and its variance is bounded in terms of the sample size by $\mathrm{Var}[\hat{\varepsilon}_n^R] \leq 1/n$. For small samples, the bias can be severe. It typically improves for large samples. Indeed, for fixed-partition histogram rules, $E[\hat{\varepsilon}_n^R]$ is monotonically increasing. The mean-square error for resubstitution error estimation for a fixed-partition histogram rule having at most q cells possesses the bound [19]:

$$E[|\hat{\varepsilon}_n^R - \varepsilon_n|^2] \leq \frac{6q}{n}. \tag{42}$$

4.3. **Cross-validation.** With cross-validation, classifiers are designed from parts of the sample, each is tested on the remaining data, and ε_n is estimated by averaging the errors. In *k-fold cross-validation*, S_n is partitioned into k folds $S_{(i)}$, for $i= 1, 2,\ldots, k$, where for simplicity we assume that k divides n. Each fold is left out of the design process and used as a test set, and the estimate is the overall proportion of errors committed on all folds:

$$\hat{\varepsilon}_{n,k}^{CV} = \frac{1}{n}\sum_{i=1}^{k}\sum_{j=1}^{n/k} |Y_j^{(i)} - \psi_{n,i}(\mathbf{X}_j^{(i)})| \tag{43}$$

where $\psi_{n,i}$ is designed on $S_n - S_{(i)}$ and $(\mathbf{X}_j^{(i)}, Y_j^{(i)})$ is a sample point in $S_{(i)}$. Picking the folds randomly leads to *random-fold cross-validation*. On the other hand, pre-selecting which parts of the sample go into each fold leads to *fixed-fold cross-validation*, a non-randomized error estimator. The process may be repeated, where several cross-validated estimates are computed, using different partitions of the data into folds, and the results averaged. In *stratified cross-validation*, the classes are represented in each fold in the same proportion as in the original data. A k-fold cross-validation estimator is unbiased as an estimator of $E[\varepsilon_{n-n/k}]$, i.e., $E[\hat{\varepsilon}_{n,k}^{CV}] = E[\varepsilon_{n-n/k}]$.

A *leave-one-out* estimator is an n-fold cross-validated estimator. A single observation is left out, n classifiers are designed from sample subsets formed by leaving out one sample pair, each is applied to the left-out pair, and the estimator $\hat{\varepsilon}_n^{CV}$ is $1/n$ times the number of errors made by the n classifiers (where for notational ease we write $\hat{\varepsilon}_n^{CV}$ instead of $\hat{\varepsilon}_{n,n}^{CV}$). Note that both fixed-$n$-fold and random-$n$-fold cross-validated estimators coincide with the same non-randomized leave-one-out estimator. The estimator $\hat{\varepsilon}_n^{CV}$ is unbiased as an estimator of $E[\varepsilon_{n-1}]$, i.e., $E[\hat{\varepsilon}_n^{CV}] = E[\varepsilon_{n-1}]$. A key concern is the variance of the estimator for small n [13]. Performance depends on the classification rule. The mean-square error for leave-one-out error estimation for a fixed-partition histogram rule possesses the bound [19]:

$$E[|\hat{\varepsilon}_n^{CV} - \varepsilon_n|^2] \leq \frac{1 + 6e^{-1}}{n} + \frac{6}{\sqrt{\pi(n-1)}} \tag{44}$$

Comparing Eqs. (42) and (44), we can see that $\sqrt{n-1}$ for leave-one-out estimation as opposed to n in the denominator for resubstitution shows greater variance for leave-one-out, for fixed-partition histogram rules.

To appreciate the difficulties inherent in the leave-one-out bounds, we will simplify them in a way that makes them more favorable to precise estimation. The performance of $\hat{\varepsilon}_n^{CV}$ guaranteed by Eq. (44) becomes better if we lower the bound. A lower bound than the one in Eq. (44) is $1.8/\sqrt{n-1}$. Even for this better standard-deviation bound, the numbers one gets for $n = 50$ and 100 still exceed 0.5 and 0.435, respectively. So the bound is essentially useless for small samples.

4.4. **Bootstrap.** The bootstrap methodology is a general resampling strategy that can be applied to error estimation [52]. It is based on the notion of an *empirical distribution* F^*, which serves as a replacement to the original unknown feature-label distribution F. The empirical distribution is discrete, putting mass $\frac{1}{n}$ on each of the n available data points. A *bootstrap sample* S_n^* from F^* consists of n equally-likely draws with replacement from the original sample S_n. Hence, some points may appear multiple times, whereas others may not appear at all. The probability that a given point will not appear in S_n^* is $(1-1/n)^n \approx e^{-1}$. Therefore, a bootstrap sample of size n contains on average $(1-e^{-1})n \approx (0.632)n$ of the original points. The basic bootstrap error estimator is the *bootstrap zero estimator* [53], which is defined by

$$\hat{\varepsilon}_n^{BZ} = E_{F^*}[|Y - \psi_n^*(\mathbf{X})| : (\mathbf{X}, Y) \in S_n - S_n^*] \tag{45}$$

where S_n is fixed. The classifier ψ_n^* is designed on the bootstrap sample and tested on the points that are left out. In practice, the expectation is approximated by a sample mean based on B independent replicates, $S_{n,b}^*$, $b = 1, 2, \ldots, B$:

$$\hat{\varepsilon}_n^{BZ} = \frac{\sum_{b=1}^{B}\sum_{i=1}^{n} |Y_i - \psi_{n.b}^*(\mathbf{X}_i)| I_{(\mathbf{X}_i, Y_i) \in S_n - S_{n,b}^*}}{\sum_{b=1}^{B}\sum_{i=1}^{n} I_{(\mathbf{X}_i, Y_i) \in S_n - S_{n,b}^*}} \tag{46}$$

The bootstrap zero estimator is clearly a randomized error estimator. In order to keep the internal variance low, and thus achieve a small overall variance, a large enough number B of bootstrap samples must be employed. In the literature, B between 25 and 200 has been recommended. In addition, a variance-reducing technique is often employed, called balanced bootstrap resampling [54], according to which each sample point is made to appear exactly B times in the computation.

The bootstrap zero estimator tends to be a high-biased estimator of $E[\varepsilon_n]$, since the number of points available for design is on average only $0.632n$. The *.632 bootstrap estimator* tries to correct this bias by doing a weighted average of the zero and resubstitution estimators [53]:

$$\hat{\varepsilon}_n^{B632} = (1 - 0.632)\,\hat{\varepsilon}_n^{R} + 0.632\,\hat{\varepsilon}_n^{BZ} \tag{47}$$

On the other hand, the *bias-corrected bootstrap* estimator tries to correct for resubstitution bias. It is defined by

$$\hat{\varepsilon}_n^{BBC} = \hat{\varepsilon}_n^{R} + \frac{1}{B}\sum_{b=1}^{B}\sum_{i=1}^{n}\left(\frac{1}{n} - P_{i,b}^*\right)|Y_i - \psi_{n,b}^*(\mathbf{X}_i)| \tag{48}$$

where $P_{i,b}^*$ is the proportion of times that $(\mathbf{X}_i, Y_i)$ appears in the bootstrap sample $S_{n,b}^*$. This estimator adds to the resubstitution estimator the bootstrap estimate of its bias.

4.5. **Bolstering.** A quick calculation reveals that the resubstitution estimator is given by

$$\hat{\varepsilon}_n^R = E_{F^*}[|Y - \psi_n(\mathbf{X})|], \tag{49}$$

where F^* is the empirical feature-label distribution. Relative to F^*, no distinction is made between points near or far from the decision boundary. If one spreads the probability mass at each point of the empirical distribution, then variation is reduced because points near the decision boundary will have more mass on the other side than will points far from the decision boundary. Hence, more confidence is attributed to points far from the decision boundary than to points near it.

To take advantage of this observation, consider a probability density function $f_i^\diamond$, for $i = 1, 2, \ldots, n$, called a *bolstering kernel*, and define the *bolstered empirical distribution* $F^\diamond$, with probability density function given by:

$$f^\diamond(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i^\diamond(\mathbf{x} - \mathbf{x}_i) \tag{50}$$

The *bolstered resubstitution* estimator [55] is obtained by replacing F^* by $F^\diamond$ in Eq. (49):

$$\hat{\varepsilon}_n^{\diamond R} = E_{F^\diamond}[|Y - \psi_n(\mathbf{X})|]. \tag{51}$$

Bolstering may actually be applied to any error-counting estimation procedure; for example, one can define a *bolstered leave-one-out* estimator [55]. However, in what follows, we focus for the most part on the bolstered resubstitution case.

The following is a computational expression, equivalent to (51), for the bolstered resubstitution estimator:

$$\hat{\varepsilon}_n^{\diamond R} = \frac{1}{n}\sum_{i=1}^{n}\left(I_{y_i=0}\int_{A_1} f_i^\diamond(x - x_i)\,dx \;+\; I_{y_i=1}\int_{A_0} f_i^\diamond(x - x_i)\,dx\right). \tag{52}$$

The integrals are the error contributions made by the data points, according to whether $y_i = 0$ or $y_i = 1$. The bolstered resubstitution estimate is equal to the sum of all error contributions divided by the number of points. If the classifier is linear, then the decision boundary is a hyperplane and it is usually possible to find analytical expressions for the integrals, for instance, for Gaussian bolstering; otherwise, Monte-Carlo integration can be employed, and experience shows that very few Monte-Carlo samples are necessary. See Fig. 6 for an illustration, where the classifier is linear and the bolstering kernels are uniform circular distributions (note that the bolstering kernels need not have the same variance). The samples in this example correspond to a subset of the cancer data used in Section 2.5 (the linear classifier is Fig. 6 was obtained via LDA).

A key issue is choosing the amount of bolstering, i.e., the kernel variances. Since the purpose of bolstering is to improve error estimation in the small-sample setting, we do no want to use bolstering kernels that require complicated inferences. Hence, we consider zero-mean, spherical bolstering kernels with covariance matrices of the form $\sigma_i\mathbf{I}$. The choice of the parameters $\sigma_1, \sigma_2, \ldots, \sigma_n$ determines the variance and bias properties of the corresponding bolstered estimator. If $\sigma_1 = \sigma_2 = \ldots = \sigma_n = 0$, then there is no bolstering and the bolstered estimator reduces to the original estimator. As a general rule, larger σ_i lead to lower-variance estimators, but after a certain point this advantage is offset by increasing bias.

FIGURE 6. Bolstered resubstitution for linear classification, assuming uniform circular bolstering kernels. The bolstered resubstitution error is the sum of all contributions (shaded areas) divided by the number of points.

We wish to select σ_1, σ_2,..., σ_n to make the bolstered resubstitution estimator nearly unbiased. One can think of $(\mathbf{X}, Y)$ in Eq. (1) as a random test point. Given $Y = k$, this test point is at a "mean distance" δ_k from the data points belonging to class k, for $k = 1, 2$. Resubstitution tends to be optimistically biased because the test points in (41) are all at distance zero from the training data. Since bolstered resubstitution spreads the test points, the task is to find the amount of spreading that makes the test points to be as close as possible to the true mean distance to the training data points.

The mean distance δ_k can be approximated by the sample-based estimate

$$\hat{\delta}_k = \frac{\sum_{i=1}^{n} \min_{j \neq i} \{||\mathbf{x}_i - \mathbf{x}_j||\} I_{y_i = k}}{\sum_{i=1}^{n} I_{y_i = k}} \tag{53}$$

This estimate is the mean minimum distance between points belonging to class $Y = k$.

Rather than estimating a separate bolstering kernel standard deviation for each sample point, we propose to compute two distinct standard deviations τ_1 and τ_2, one for each class, based on the mean-distance estimates δ_1 and δ_2 (this limits the complexity of the estimation problem, which is advantageous in small-sample settings). Thus, we let $\sigma_i = \tau_k$, for $y_i = k$. To arrive at estimates for τ_1 and τ_2, let D be the random variable giving the distance to the origin of a randomly selected point from a unit-variance bolstering kernel, and let F_D be the probability distribution function for D. In the case of a bolstering kernel of standard deviation τ_k, all distances get multiplied by τ_k, so if D' is the distance random variable for this more general case, then $F_{D'}(x) = F_D(x/\tau_k)$. For the class $Y = k$, the value of τ_k is to be chosen so that the median distance of a random test point to the origin of the bolstering kernel of standard deviation τ_k is equal to the mean-distance $\hat{\delta}_k$, the result being that half of the test points will be farther from the origin than $\hat{\delta}_k$,

and the other half will be closer. A little reflection shows that τ_k is the solution of the equation $F_{D'}^{-1}(0.5) = \tau_k F_D^{-1}(0.5) = \hat{\delta}_k$, so that the final estimated standard deviations for the bolstering kernels are given by

$$\sigma_i = \frac{\hat{\delta}_k}{F_D^{-1}(0.5)}, \quad \text{for } y_i = k. \tag{54}$$

Note that, as the number of samples in the sample increases, $\hat{\delta}_k$ decreases, and therefore so does σ_i. This is the expected behavior in this case, since plain resubstitution tends to be less biased as the number of samples increases. Note also that the denominator $F_D^{-1}(0.5)$ may be viewed as a constant dimensionality correction factor (being a function of the number of dimensions through D), which can be precomputed and stored off-line.

We mention that, when resubstitution is heavily low-biased, it may not be a good idea to spread incorrectly classified data points, as that increases optimism of the error estimator (low bias). Bias is reduced by letting $\sigma_i = 0$ (no bolstering) for incorrectly classified points. The resulting estimator is called the *semi-bolstered resubstitution* estimator [55].

4.6. **Error Estimation Performance.** We now illustrate the small-sample performance of several of the error estimators discussed in the previous subsections by means of simulation experiments based on synthetic data (see also [13, 55]). We consider resubstitution (resub), leave-one-out (loo), stratified 10-fold cross-validation with 10 repetitions (cv10r), the balanced .632 bootstrap (b632), Gaussian bolstered resubstitution (bresub), Gaussian semi-bolstered resubstitution (sresub) and Gaussian bolstered leave-one-out (bloo). The number of bootstrap samples is $B = 100$, which makes the number of designed classifiers be the same as for cv10r. We employ three classification rules (in order of complexity: LDA, 3NN, and CART. For LDA, the bolstered estimators are computed using analytical formulas developed in [55]; for 3NN and CART, Monte-Carlo sampling is used — we have found that only $M = 10$ Monte-Carlo samples per bolstering kernel is adequate, and increasing M to a larger value reduces the variance of the estimators only slightly.

The experiments assess the empirical distribution of $\varepsilon_n - \hat{\varepsilon}_n$ for each error estimator $\hat{\varepsilon}_n$. This distribution measures the difference between the true error and the estimated error of the designed classifier. Deviation distributions are from 1000 independent data sets drawn from several models. The synthetic model for LDA consists of spherical Gaussian class-conditional densities with means located at $(\delta, \delta, \ldots, \delta)$ and $(-\delta, -\delta, \ldots, -\delta)$, where $\delta > 0$ is a separation parameter that controls the Bayes error. The synthetic model for 3NN and CART corresponds to class-conditional densities given by a mixture of spherical Gaussians, with means at opposing vertices of a hypercube centered at the origin and side 2δ. For instance, for $d = 5$, the class-conditional density for class 0 has means at $(\delta, \delta, \delta, \delta, \delta)$ and $(-\delta, -\delta, -\delta, \delta, -\delta)$, and the class-conditional density for class 1 has means at $(\delta, -\delta, \delta, -\delta, \delta)$ and $(-\delta, \delta, -\delta, \delta, -\delta)$. In all cases, there are equal *a priori* class probabilities.

Table 4.6 lists the parameters for the twelve experiments considered in this study, corresponding to choices among the three classification rules, for various separations δ between the class means, low or moderate dimensionality d, and equal or distinct standard deviations σ_1 and σ_2 for the class-conditional densities. The parameters were chosen so as to give Bayes error about 0.1 in half of the cases and about 0.2 in

the other half. These are difficult models, with considerable overlapping between the classes (even in the cases where the Bayes error is 0.1) owing to large discrepancy in variance between the classes, not to actual separation between the means.

Experiment	Rule	d	δ	σ_1	σ_2	Bayes error
1	LDA	2	0.59	1.00	1.00	0.202
2	LDA	2	0.59	1.00	4.00	0.103
3	LDA	5	0.37	1.00	1.00	0.204
4	LDA	5	0.37	1.00	2.16	0.103
5	3NN	2	1.20	1.00	1.00	0.204
6	3NN	2	1.20	1.00	5.20	0.103
7	3NN	5	0.77	1.00	1.00	0.204
8	3NN	5	0.77	1.00	2.35	0.105
9	CART	2	1.20	1.00	1.00	0.204
10	CART	2	1.20	1.00	5.20	0.103
11	CART	5	0.77	1.00	1.00	0.204
12	CART	5	0.77	1.00	2.35	0.105

TABLE 1. Twelve experiments used in the simulation study

Plots of beta-density fits of the empirical deviation distribution for sample size $n = 20$ are displayed in Figs. 7–9 (see [55] and its companion website for the complete results of this simulation study). Note that the narrower and taller the distribution, the smaller the variance of the deviation, whereas the closer its mean is to the vertical axis, the more unbiased the error estimator is. We can see that resubstitution, leave-one-out, and even 10-fold cross-validation are generally outperformed by the bootstrap and bolstered estimators. Bolstered resubstitution is very competitive with the bootstrap, in some cases outperforming it. For LDA, the best estimator overall is bolstered resubstitution. For 3NN and CART, which are classifiers known to overfit in small-sample settings, the situation is not so clear. For 3NN, we can see that bolstered resubstitution fails in correcting the bias of resubstitution for $d = 5$, despite having small variance (note that it is still the best overall estimator for 3NN in Experiment 5). For CART, the bootstrap estimator is affected by the extreme low-biasedness of resubstitution. In this case, bolstered resubstitution performs much better than in the 3NN case, but the best overall estimator is the semi-bolstered resubstitution. The bolstered leave-one-out is generally much more variable than the bolstered resubstitution estimators, but it displays much less bias.

Computation time can be critical. We have found that resubstitution is by far the fastest estimator, with its bolstered versions coming just behind. Leave-one-out, and its bolstered version, are fast for a small number of samples, but performance quickly degrades with an increasing number of samples. The 10-fold cross-validation and the bootstrap estimator are the slowest estimators. Bolstered resubstitution can be hundreds of times faster than the bootstrap estimator (see [55] for a listing of timings).

We close this subsection with a comment on error estimation and the measurement of feature-set performance. Given a large number of potential feature sets,

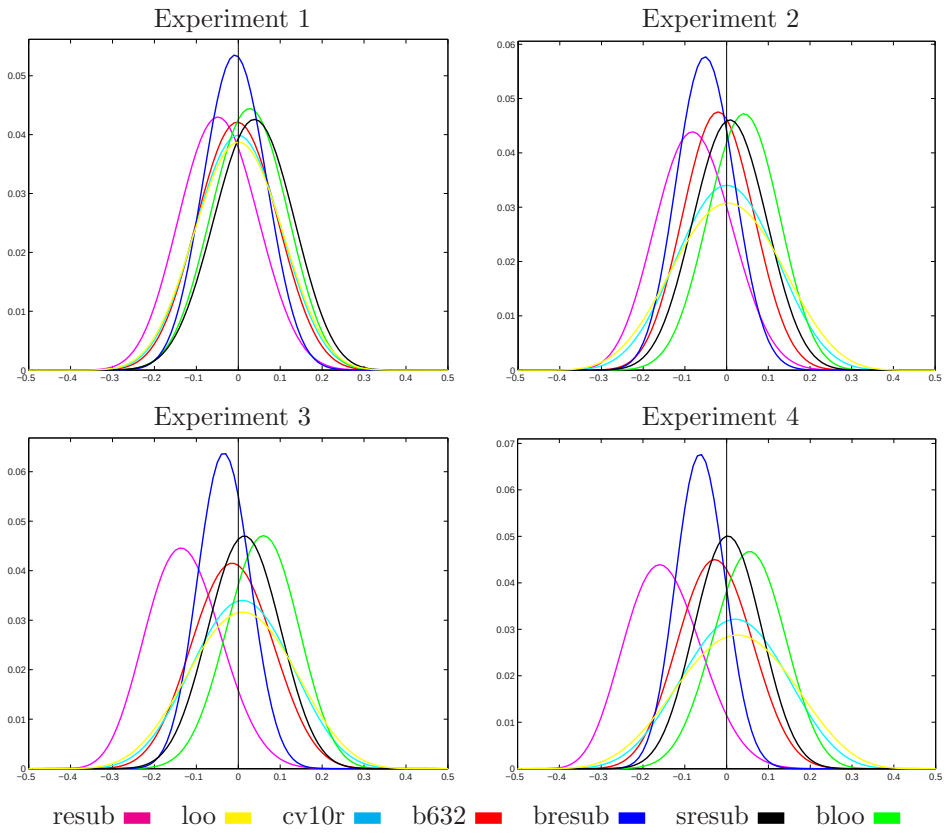

FIGURE 7. Beta fits to empirical deviation distribution, for LDA and $n = 20$.

one may wish to rank them according to the performances of their optimal classifiers, which in practice means the performances of their designed classifiers. This requires estimating the true errors of the various feature-set classifiers. It is not uncommon for this ranking to be done by cross-validation. Resubstitution is usually rejected for this task on account of its typical bias, even though there can be enormous computational savings, especially when one is considering thousands of feature sets. However, there is evidence that, for top-performing feature sets (the only ones in which we are actually interested), the ranking accuracies of resubstitution and cross-validation, in comparison to the true ranking, are essentially equivalent [56].

REFERENCES

[1] A. Ben-Dor, L. Bruhn, N. Friedman, I. Nachman, M. Schummer, Z. Yakhini, Tissue classification with gene expression profiles, Computational Biology 7 (2000) 559–583.

[2] M. Bittner, P. Meltzer, Y. Chen, Y. Jiang, E. Seftor, M. Hendrix, M. Radmacher, R. Simon, Z. Yakhini, A. Ben-Dor, N. Sampas, E. Dougherty, E. Wang, F. Marincola, C. Gooden, J. Lueders, A. Glatfelter, P. Pollock, J. Carpten, E. Gillanders, D. Leja, K. Dietrich,

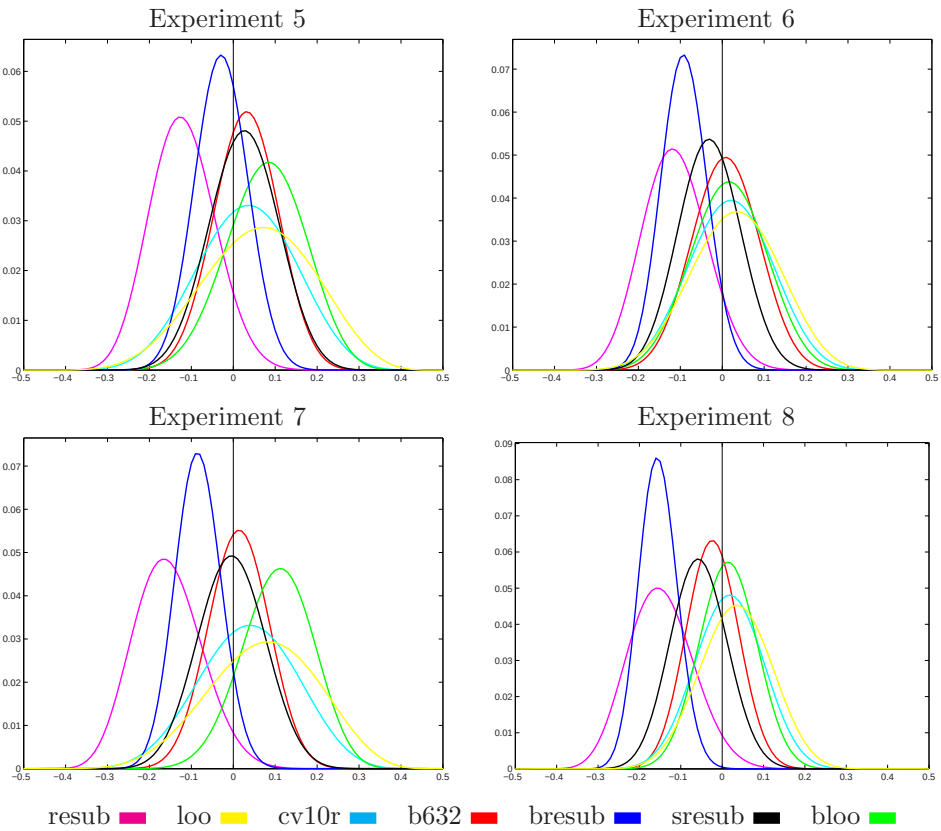

FIGURE 8. Beta fits to empirical deviation distribution, for 3NN and $n = 20$.

C. Beaudry, M. Berens, D. Alberts, V. Sondak, N. Hayward, J. Trent, Molecular classification of cutaneous malignant melanoma by gene expression profiling, Nature 406 (2000) 536–540.

[3] G. Callagy, E. Cattaneo, Y. Daigo, L. Happerfield, L. Bobrow, P. Pharoah, C. Caldas, Molecular classification of breast carcinomas using tissue microarrays, Diagn. Mol. Pathol. 12 (1) (2003) 27–34.

[4] L. Dyrskjot, T. Thykjaer, M. Kruhoffer, , J. Jensen, N. Marcussen, S. Hamilton-Dutoit, H. Wolf, T. Orntoft, Identifying distinct classes of bladder carcinoma using microarrays, Nature Genetics 33 (1) (2003) 90–96.

[5] T. Furey, N. Cristianini, N. Duffy, D. Bednarski, M. Schummer, D. Haussler, Support vector machine classification and validation of cancer tissue samples using microarray expression data, Bioinformatics 16 (10) (2000) 906–914.

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

[7] I. Hedenfalk, D. Dugan, Y. Chen, M. Radmacher, M. Bittner, R. Simon, P. Meltzer, B. Gusterson, M. Esteller, M. Raffeld, Z. Yakhini, A. Ben-Dor, E. R. Dougherty, J. Kononen, L. Bubendorf, W. Fehrle, S. Pittaluga, S. Gruvberger, N. Loman, O. Johannsson, H. Olsson, B. Wilfond, G. Sauter, O.-P. Kallioniemi, A. Borg, J. Trent, Gene-expression profiles in hereditary breast cancer, The New England Journal of Medicine 344 (8) (2001) 539–548.

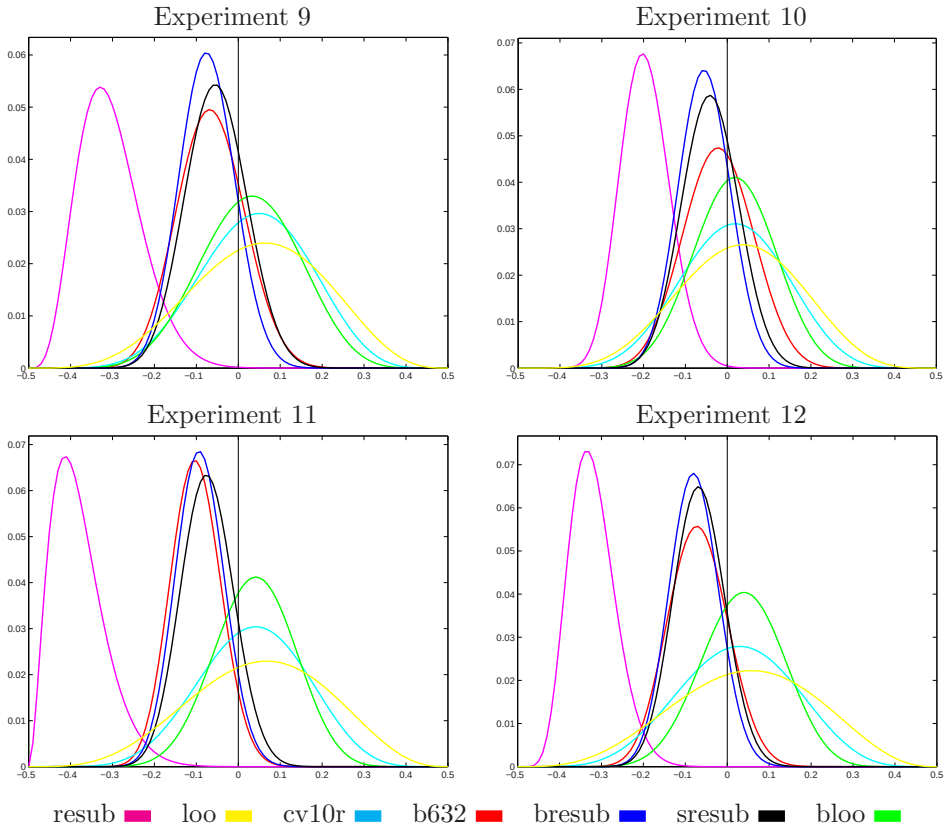

FIGURE 9. Beta fits to empirical deviation distribution, for CART and $n = 20$.

[8] J. Khan, J. Wei, M. Ringner, L. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. Antonescu, C. Peterson, P. Meltzer, Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks, Nature Medicine 7 (2001) 673–679.

[9] S. Kim, E. Dougherty, I. Shmulevich, K. Hess, S. Hamilton, J. Trent, G. Fuller, W. Zhang, Identification of combination gene sets for glioma classification, Molecular Cancer Therapy 1 (2002) 1229–1236.

[10] T. Kobayashi, M. Yamaguchi, S. Kim, J. Morikawa, S. Ogawa, S. Ueno, E. Suh, E. Dougherty, I. Shmulevich, H. Shiku, W. Zhang, Microarray reveals differences in both tumors and vascular specific gene expression in de novo cd5+ and cd5- diffuse large b-cell lymphomas, Cancer Research 63 (2003) 60–66.

[11] A. Levenson, I. Kliakhandler, K. Svoboda, K. Pease, S. Kaiser, r. Ward, J.E, V. Jordan, Molecular classification of selective oestrogen receptor modulators on the basis of gene expression profiles of breast cancer cells expressing oestrogen receptor alpha, Br. J. Cancer 87 (4) (2002) 449–456.

[12] A. Szabo, K. Boucher, W. Carroll, L. Klebanov, A. Tsodikov, A. Yakovlev, Variable selection and pattern recognition with gene expression data generated by the microarray technology, Mathematical Biosciences 176 (1) (2002) 71–98.

[13] U. Braga-Neto, E. Dougherty, Is cross-validation valid for microarray classification?, Bioinformatics 20 (3) (2004) 374–380.

[14] E. Dougherty, Small sample issues for microarray-based classification, Comparative and Functional Genomics 2 (2001) 28–34.
[15] M. van de Vijver, Y. He, L. van't Veer, H. Dai, A. Hart, D. Voskuil, G. Schreiber, J. Peterse, C. Roberts, M. Marton, M. Parrish, D. Astma, A. Witteveen, A. Glas, L. Delahaye, T. van der Velde, H. Bartelink, S. Rodenhuis, E. Rutgers, S. Friend, R. Bernards, A gene-expression signature as a predictor of survival in breast cancer, The New England Journal of Medicine 347 (25) (2002) 1999–2009.
[16] V. Vapnik, A. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities, Probability and its Applications 16 (1971) 264–280.
[17] E. Baum, On the capabilities of multilayer perceptrons, Complexity 4 (1988) 193–215.
[18] L. Gordon, R. Olshen, Asymptotically efficient solutions to the classification problem, Annals of Statistics 6 (1978) 525–533.
[19] L. Devroye, L. Gyorfi, G. Lugosi, A Probabilistic Theory of Pattern Recognition, Springer, New York, 1996.
[20] E. Dougherty, M. Bittner, Y. Chen, S. Kim, K. Sivakumar, J. Barrera, P. Meltzer, J. Trent, Nonlinear filters in genomic control, in: Proceedings of the IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing, Antalya, Turkey, 1999.
[21] S. Kim, E. Dougherty, M. Bittner, Y. Chen, K. Sivakumar, P. Meltzer, J. Trent, A general framework for the analysis of multivariate gene interaction via expression arrays, Biomedical Optics 5 (4) (2000) 411–424.
[22] I. Tabus, J. Astola, On the use of mdl principle in gene expression prediction, Journal of Applied Signal Processing 2001 (4) (2001) 297–303.
[23] S. Kauffman, The Origins of Order: Self-Organization and Selection in Evolution, Oxford University Press, 1993.
[24] I. Schmulevich, E. Dougherty, W. Zhang, From Boolean to probabilistic Boolean networks as models of genetic regulatory networks, Proceedings of the IEEE 90 (2002) 1778–1792.
[25] X. Zhou, X. Wang, E. Dougherty, Binarization of microarray data based on a mixture model, Molecular Cancer Therapeutics 2 (7) (2003) 679–684.
[26] I. Shmulevich, W. Zhang, Binary analysis and optimization-based normalization of gene expression data, Bioinformatics 18 (4) (2002) 555–565.
[27] T. Cover, P. Hart, Nearest-neighbor pattern classification, IEEE Trans. on Information Theory 13 (1967) 21–27.
[28] C. Stone, Consistent nonparametric regression, Annals of Statistics 5 (1977) 595–645.
[29] V. Vapnik, Statistical Learning Theory, Wiley, New York, 1998.
[30] P. Wald, R. Kronmal, Discriminant functions when covariances are unequal and sample sizes are moderate, Biometrics 33 (1977) 479–484.
[31] A. Farago, G. Lugosi, Strong universal consistency of neural network classifiers, IEEE Trans. on Information Theory 39 (1993) 1146–1151.
[32] C. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, New York, 1995.
[33] L. van't Veer, H. Dai, M. van de Vijver, Y. He, A. Hart, M. Mao, H. Peterse, K. van der Kooy, M. Marton, A. Witteveen, G. Schreiber, R. Kerkhoven, C. Roberts, P. Linsley, R. Bernards, S. Friend, Gene expression profiling predicts clinical outcome of breast cancer, Nature 415 (2002) 530–536.
[34] A. Izenman, Recent developments in nonparametric density estimation, Journal of the American Statistical Association 86 (413) (1991) 205–224.
[35] D. Titterington, Common structure of smoothing techniques in statistics, International Statistical Review 53 (1985) 141–170.
[36] J. Friedman, Regularized discriminant analysis, Journal of the American Statistical Association 84 (405) (1989) 165–175.
[37] M. Skurichina, R. Duin, S. Raudys, K-nearest neighbours noise injection in multilayer perceptron training, IEEE Trans. on Neural Networks 11 (2) (2000) 504–511.
[38] J. Sietsma, R. Dow, Neural network pruning — why and how, in: Proc. IEEE International Conference on Neural Networks I, 1988, pp. 325–333.
[39] V. Vapnik, A. Chervonenkis, Theory of Pattern Recognition, Nauka, Moscow, 1974.
[40] V. Vapnik, Estimation of Dependencies Based on Empirical Data, Springer-Verlag, New York, 1982.

[41] A. Kolmogorov, Three approaches to the quantitative definition of information, Probl. Pederach. Inform. 1 (1965) 3–11.
[42] J. Rissanen, Stochastic complexity and modeling, Annals of Statistics 14 (1986) 1080–1100.
[43] I. Tabus, J. Rissanen, J. Astola, Classification and feature gene selection using the normalized maximum likelihood model for discrete regression, Signal Processing 83 (4) (2003) 713–727, special issue on Genomic Signal Processing.
[44] T. Cover, J. van Campenhout, On the possible orderings in the measurement selection problem, IEEE Trans. on Systems, Man, and Cybernetics 7 (1977) 657–661.
[45] A. Jain, B. Chandrasekaran, Dimensionality and sample size considerations in pattern recognition practice, in: P. Krishnaiah, L. Kanal (Eds.), Classification, Pattern Recognition and Reduction of Dimensionality, Vol. 2 of Handbook of Statistics, North-Holland, Amsterdam, 1982, Ch. 39, pp. 835–856.
[46] G. Hughes, On the mean accuracy of statistical pattern recognizers, IEEE Trans. on Information Theory 14 (1968) 55–63.
[47] P. Narendra, K. Fukunaga, A branch and bound algorithm for feature subset selection, IEEE Trans. on Computers 26 (9) (1977) 917–922.
[48] Y. Hamamoto, S. Uchimura, Y. Matsunra, T. Kanaoka, S. Tomita, Evaluation of the branch and bound algorithm for feature selection, Pattern Recognition Letters 11 (1990) 453–456.
[49] P. Pudil, J. Novovicova, J. Kittler, Floating search methods in feature selection, Pattern Recognition Letters 15 (1994) 1119–1125.
[50] A. Jain, D. Zongker, Feature selection: Evaluation, application, and small sample performance, IEEE Trans. on Pattern Analysis and Machine Intelligence 19 (2) (1997) 153–158.
[51] S. Ross, A first course in probability, 4th Edition, Macmillan, New York, 1994.
[52] B. Efron, The Jacknife, The Bootstrap, and Other Resampling Plans, SIAM NSF-CBMS, 1982, monograph #38.
[53] B. Efron, Estimating the error rate of a prediction rule: Improvement on cross-validation, Journal of the American Statistical Association 78 (382) (1983) 316–331.
[54] M. Chernick, Bootstrap Methods: A Practitioner's Guide, John Wiley & Sons, New York, NY, 1999.
[55] U. Braga-Neto, E. Dougherty, Bolstered error estimation, Pattern Recognition, 37 (6) (2004) 1267–1281.
[56] U. Braga-Neto, R. Hashimoto, E. Dougherty, D. Nguyen, R. Carroll, Is cross-validation better than resubstitution for ranking genes?, Bioinformatics 20 (2) (2004) 253–258.

(U. Braga-Neto) Section of Clinical Cancer Genetics, UT MD Anderson Cancer Center, Houston, TX, and Department of Electrical Engineering, Texas A&M University, College Station, TX, ubraga@mdanderson.org

(E. Dougherty) Department of Pathology, UT MD Anderson Cancer Center, Houston, TX, and Department of Electrical Engineering, Texas A&M University, College Station, TX, edward@ee.tamu.edu