Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

PERGAMON

Pattern Recognition 37 (2004) 1641–1651

PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/patcog

Classifier performance as a function of distributional complexity

Sanju N. Attoor[a], Edward R. Dougherty[a,b,*]

[a] *Department of Electrical Engineering, Texas A&M University, College Station, TX, USA*
[b] *Department of Pathology, UT MD Anderson Cancer Center, Houston, TX, USA*

Received 18 September 2003; received in revised form 15 October 2003

Abstract

When choosing a classification rule, it is important to take into account the amount of sample data available. This paper examines the performances of classifiers of differing complexities in relation to the complexity of feature-label distributions in the case of small samples. We define the distributional complexity of a feature-label distribution to be the minimal number of hyperplanes necessary to achieve the Bayes classifier if the Bayes classifier is achievable by a finite number of hyperplanes, and infinity otherwise. Our approach is to choose a model and compare classifier efficiencies for various sample sizes and distributional complexities. Simulation results are obtained by generating data based on the model and the distributional complexities. A linear support vector machine (SVM) is considered, along with several nonlinear classifiers. For the most part, we see that there is little improvement when one uses a complex classifier instead of a linear SVM. For higher levels of distributional complexity, the linear classifier degrades, but so do the more complex classifiers owing to insufficient training data. Hence, if one were to obtain a good result with a more complex classifier, it is most likely that the distributional complexity is low and there is no gain over using a linear classifier. Hence, under the model, it is generally impossible to claim that use of the nonlinear classifier is beneficial. In essence, the sample sizes are too small to take advantage of the added complexity. An exception to this observation is the behavior of the three-nearest-neighbor (3NN) classifier in the case of two variables (but not three) when there is very little overlap between the label distributions and the sample size is not too small. With a sample size of 60, the 3NN classifier performs close to the Bayes classifier, even for high levels of distributional complexity. Consequently, if one uses the 3NN classifier with two variables and obtains a low error, then the distributional complexity might be large and, if such is the case, there is a significant gain over using a linear classifier.

Keywords: Classifier design; Classifier dimension; Distributional complexity; Small samples

1. Introduction

When choosing a classification rule, it is important to take into account the amount of sample data available. Whereas it is beneficial that a classification rule be consistent, meaning that the increase in classification error owing to design from sample data goes to zero as the sample size goes to infinity, it is design behavior at small sample sizes that is critical when data are severely limited. For instance, small-sample design has become a paramount issue in functional genomics, where a key desire is to design classifiers based on gene-expression data to discriminate between phenotypes [1,2].

The problem of choosing a classification rule is exacerbated by the difficulty of error estimation with small samples, where one is compelled to train and test a classifier on the same data, with error estimation being accomplished via resubstitution, cross-validation, bootstrap, or some other training-data-based error estimator. Cross-validation seems

* Corresponding author. Tel.: +1-979-862-8154; fax: +1-979-845-6259.
E-mail addresses: edward@ee.tamu.edu, e-dougherty@tamu.edu (E.R. Dougherty).

0031-3203/$30.00 © 2003 Pattern Recognition Society. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2003.10.013

to be the preferred choice of many investigators owing to its approximate global unbiasedness (meaning that on an average the cross-validation error estimate is close to the true error of the designed classifier), but the variance of the cross-validation error estimation is typically very high, which can make it unreliable [3,4]. While resubstitution has much less variance, it is typically low-biased, perhaps extremely so. As for bootstrap estimation, it has reduced variance in comparison to cross-validation, but at an increased computational cost. A consequence of the small-sample error-estimation problem is that, even if a classifier is designed and shows low error based on small-sample error estimation, it may in fact be quite poor—and the situation is worse for complex classifiers [5].

Since design error is typically worse for more complex classifiers, and since error estimation is problematic, prudence seems to dictate the preferability of low-complexity classifiers. The downside, of course, is that low-complexity classifiers have less ability to partition the feature space and therefore are inadequate for classification involving complex feature-label distributions. The purpose of this paper is to examine the performances of classifiers of differing complexities in relation to the complexity of feature-label distributions in the case of small samples. The analysis will be done in such a way as to remove the effects of small-sample error estimation so that we can see if classifier complexity is actually an advantage.

2. Classification

Classification involves a classifier ψ, a feature vector $\mathbf{X} = (X_1, X_2, \ldots, X_d)$ composed of random variables, and a binary random variable Y to be predicted by $\psi(\mathbf{X})$. The values, 0 or 1, of Y are treated as class labels. The error, $\varepsilon[\psi]$, of ψ is the probability, $P(\psi(\mathbf{X}) \neq Y)$, that the classification is erroneous. It equals the expected (mean) absolute difference, $E[|Y - \psi(\mathbf{X})|]$, between the label and the classification. $X_1, X_2, \ldots, X_d$ can be discrete or real-valued. In the latter case, the domain of ψ is d-dimensional Euclidean space $\mathscr{R}^d$. An optimal classifier, $\psi_\bullet$, is the one with minimal error, $\varepsilon_\bullet$, among all binary functions on $\mathscr{R}^d$. $\psi_\bullet$ and $\varepsilon_\bullet$ are called the *Bayes classifier* and *Bayes error*, respectively. Classification accuracy, and thus the error, depends on the probability distribution of the feature-label pair $(\mathbf{X}, Y)$—how well the labels are distributed among the variables being used to discriminate them, and how the variables are distributed in $\mathscr{R}^d$.

Classifier design uses a *sample* $S_n = \{(\mathbf{X}^1, Y^1), (\mathbf{X}^2, Y^2), \ldots, (\mathbf{X}^n, Y^n)\}$ of feature-label pairs and a *classification rule* to construct a classifier ψ_n whose error is hopefully close to the Bayes error. The Bayes error $\varepsilon_\bullet$ is estimated by the error ε_n of ψ_n. Because $\varepsilon_\bullet$ is minimal, $\varepsilon_n \geqslant \varepsilon_\bullet$, and there is a design error (cost of estimation), $\Delta_n = \varepsilon_n - \varepsilon_\bullet$. Since it depends on the sample, ε_n is a random variable, as is Δ_n. Hence, we are concerned with the expected value of Δ_n, $E[\Delta_n] = E[\varepsilon_n] - \varepsilon_\bullet$. Hopefully, $E[\Delta_n]$ gets closer to 0 as the sample size grows. This will depend on the classification rule and the distribution of the feature-label pair $(\mathbf{X}, Y)$.

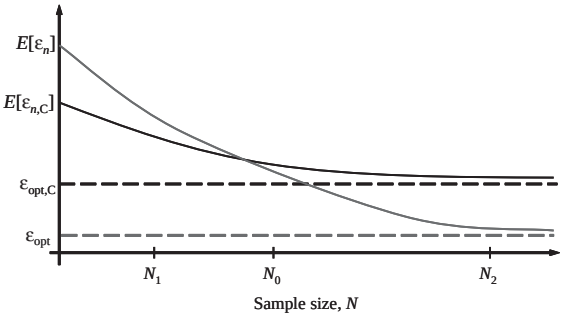

Fig. 1. Relationship between sample size and constraint.

Finding an optimum classifier is essentially impossible, especially given a small sample size and lack of knowledge of the feature-label distribution. Instead of finding an optimum classifier, one can restrict the functions from which a classifier must be chosen to a class C. It is then required to find an optimum constrained classifier $\psi_C \in C$, having error ε_C. Constraining the classifier can reduce the expected design error, but at the cost of increasing the error of the best possible classifier. Since optimization in C is over a subclass of classifiers, the error, ε_C, of ψ_C will typically exceed the Bayes error, unless the Bayes classifier happens to be in C. This *cost of constraint* (*approximation*) is $\Delta_C = \varepsilon_C - \varepsilon_\bullet$. A classification rule yields a classifier $\psi_{n,C} \in C$ with error $\varepsilon_{n,C}$, and $\varepsilon_{n,C} \geqslant \varepsilon_C \geqslant \varepsilon_\bullet$. Design error for constrained classification is $\Delta_{n,C} = \varepsilon_{n,C} - \varepsilon_C$. For small samples, this can be substantially less than Δ_n, depending on C and the rule. The error of the designed constrained classifier is decomposed as $\varepsilon_{n,C} = \varepsilon_\bullet + \Delta_C + \Delta_{n,C}$. The expected error of the designed classifier from C can be decomposed as

$$E[\varepsilon_{n,C}] = \varepsilon_\bullet + \Delta_C + E[\Delta_{n,C}]. \tag{1}$$

The constraint is beneficial if and only if $E[\varepsilon_{n,C}] < E[\varepsilon_n]$, which means $\Delta_C < E[\Delta_n] - E[\Delta_{n,C}]$. The dilemma: strong constraint reduces $E[\Delta_{n,C}]$ at the cost of increasing Δ_C.

The relationship between sample size and constraint can be shown graphically. Consider a consistent rule, constraint, and distribution for which $E[\varepsilon_{n+1}] \leqslant E[\varepsilon_n]$ and $E[\varepsilon_{n+1,C}] \leqslant E[\varepsilon_{n,C}]$. Then Fig. 1 illustrates the design problem. The axes correspond to sample size and error. The horizontal dashed lines represent $\varepsilon_\bullet$ and ε_C; the decreasing solid lines represent $E[\varepsilon_n]$ and $E[\varepsilon_{n,C}]$. If n is sufficiently large, then $E[\varepsilon_n] \leqslant E[\varepsilon_{n,C}]$; however, if n is sufficiently small, then $E[\varepsilon_n] \geqslant E[\varepsilon_{n,C}]$. 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 lines.

Quantitatively, the dilemma can be appreciated by considering the VC dimension of a classification rule [5]. The VC dimension provides a measure of the degree of fineness to which a classifier designed from the rule can partition the observation space. The empirical-error rule chooses the classifier in C that makes the least number of errors on the sample data. For this rule,

$$E[\Delta_{n,C}] \leqslant 8\sqrt{\frac{V_C \log n + 4}{2n}}, \tag{2}$$

where V_C is the VC dimension of C [3,5]. It is clear from the bound that n must greatly exceed the VC dimension for the bound to be small. The VC dimension of a perceptron is $d + 1$. For a neural network with an even number k of neurons, the VC dimension has the lower bound dk [6]. To appreciate the implications, suppose $d = k = 10$. Setting $V_C = 100$ and $n = 5000$ yields a bound exceeding 1, which says nothing. Admittedly, the bound is the worst-case because there are no distributional assumptions. The situation may not be nearly so bad. Still, one must proceed with care.

3. Distributional complexity

Our intuitive notion of distributional complexity within the context of pattern classification has to do with the separability of the conditional label distributions, $f(\mathbf{x}|0)$ and $f(\mathbf{x}|1)$. If they are linearly separable, then the optimal linear classifier has zero error, which means that the joint feature-label distribution, $f(\mathbf{x}, y)$, has minimal complexity. Linear separability extends, in a weakened form, to the Bayes classifier being linear—for instance, in the case of $f(\mathbf{x}|0)$ and $f(\mathbf{x}|1)$ being Gaussian with equal variances. Focusing on two-dimensional classifiers, a more complex situation arises when $f(\mathbf{x}|0)$ is concentrated in the first and third quadrants, and $f(\mathbf{x}|1)$ is concentrated in the second and fourth quadrants. Then the optimal classifier is a binary decision tree determined by $x_1 = 0$ and $x_2 = 0$. Here, we require two straight lines for perfect classification, whereas in the linearly separable case we require only a single straight line. The situation is really no different if we were to rotate the axes because then, although a binary decision tree would fail to give the Bayes classifier, a neural network could provide the two straight lines, which are now just a rotation of the axes. The entire matter extends in a weakened form to the case where the Bayes classifier is determined by two straight lines, say for $f(\mathbf{x}|0)$ and $f(\mathbf{x}|1)$ each being mixtures of two Gaussians with variances equal and the Gaussians for each being at opposite corners of a square.

The issue confronting us is the choice of a classification rule. In the case when $f(\mathbf{x}|0)$ and $f(\mathbf{x}|1)$ are linearly separable, one could use a neural network, but would that be wise? Since much less data are needed to well train a linear classifier, it is likely that a linear classification rule, such as a linear support vector machine (SVM), would be expected to outperform a neural network. In the case when $f(\mathbf{x}|0)$ is concentrated in the first and third quadrants, and $f(\mathbf{x}|1)$ is concentrated in the second and fourth quadrants, it is clear that for sufficient training data a neural network will outperform a linear classification rule, but even if it outperforms a linear classification rule for a small sample, will its true error on the distribution still reflect the fact that the Bayes error is zero?

Based on the preceding heuristics, we define the *distributional complexity*, χ, of a feature-label distribution to be the minimal number of hyperplanes necessary to achieve the Bayes classifier, if the Bayes classifier is achievable by a finite number of hyperplanes, and we define it to be infinity otherwise. Notice that, according to this definition, the distributional complexity is rotationally invariant relative to the feature-label distribution. For example, in the two cases so far discussed in this section, $\chi = 1$ and $\chi = 2$, respectively.

Before examining classifier design relative to distributional complexity, a couple of points need to be clarified. First, high distributional complexity does not imply that a low-complexity classifier cannot perform well. Put more rigorously, given $\varepsilon > 0$, there exists a feature-label distribution with $\chi = \infty$ such that the optimal linear classifier has error less than ε.

Second, distributional complexity is rotationally invariant, but classifier performance need not be. This means that we need to be careful how we measure classifier performance relative to distributional complexity. Given a model feature-label distribution $f(\mathbf{x}, y)$, one can consider all rotations $f_\theta(\mathbf{x}, y)$ of $f(\mathbf{x}, y)$. Now suppose we wish to evaluate the performance of a classification rule relative to sample size n and model complexity. For the particular model $f_\theta(\mathbf{x}, y)$, one has choices on how to proceed. One could measure the performance as the average performance over all θ. Or one could choose θ so as to minimize (maximize) χ_θ and measure the performance for the minimizing (maximizing) value of θ. That the matter is irrelevant for rotationally invariant classification rules such as linear SVM and k-nearest-neighbor (KNN) classification (two popular rules) lessens the consequences of our decision.

Thus far, distributional complexity has been defined for a particular distribution; however, if we take distributional complexity as the basic quantity, then we may wish to consider a set of distributions having the same complexity and average the classifier error over these distributions. The purpose of this approach is to evaluate classifiers based on their abilities with regard to distributional complexity, not a particular distribution possessing a distributional complexity of interest. In this vein, suppose, for $\chi = 1, 2, \ldots, K$, we have sets of feature-label distributions, say $\{f_{11}, f_{12}, \ldots, f_{1r_1}\}, \{f_{21}, f_{22}, \ldots, f_{2r_2}\}, \ldots, \{f_{K1}, f_{K2}, \ldots, f_{Kr_K}\}$, where $\chi(f_{kj}) = k$ for all (k, j). Given a classification rule (family of classifiers) Ψ and sample size n, we can define the *χ-error* of Ψ relative to the model M composed

of the sets of feature-label distributions as

$$\omega_{\Psi}[M;n,\chi]=\sum_{j=1}^{r_\chi}\varepsilon[\psi_n;f_{\chi j}]P(f_{\chi j}|\chi), \tag{3}$$

where $\varepsilon[\psi_n;f_{\chi j}]$ is the expected error for a classifier $\psi\in\Psi$ designed from a random sample of size n drawn from $f_{\chi j}$, and $P(f_{\chi j}|\chi)$ is the probability of the distribution $f_{\chi j}$ given distributional complexity χ. The latter probability is important because, whereas there are r_χ distributions in the model possessing distributional complexity χ, they need not be equally likely in the model. Selection of a classification rule can be based on comparison of efficiency. Rule Ψ^1 is better than rule Ψ^2 for complexity χ at sample size n relative to the model M if $\omega_{\Psi^1}[M;n,\chi]\leqslant\omega_{\Psi^2}[M;n,\chi]$. In the model M we denote the distributions corresponding to a given complexity χ by $M^\chi=\{f_{\chi 1},f_{\chi 2},\ldots,f_{\chi r_\chi}\}$, and we call M^χ a *complexity class*.

Unlike the VC dimension, the distributional complexity is dependent on the feature-label distribution of the data. In one sense this is a drawback because it requires an estimate of χ from the data or prior knowledge of χ. The advantage is that the distribution-free quality of the VC dimension makes it too conservative and insufficiently fine as a measure of performance. Since, in practice, good estimation of χ from small samples is impossible, to use distributional complexity for classifier selection one can proceed heuristically by assuming a value of χ and a model over which to evaluate classifier efficiency. Our purpose here is to choose a model and compare classifier efficiencies for various samples sizes and distributional complexities.

4. Model for distributional complexity

For clarity, we begin by explaining the model for two variables (features). Extension to higher dimensions is straightforward. Begin by assuming a square grid of $m\times m$ points (m^N for N features),

$$G_{2,m}=\begin{pmatrix} z_{11} & z_{12} & \cdots & z_{1m}\\ z_{21} & z_{22} & \cdots & z_{2m}\\ \vdots & \vdots & \vdots & \vdots\\ z_{m1} & z_{m2} & \cdots & z_{mm}\end{pmatrix},$$

where the vertical and horizontal distances between adjacent points is 2δ ($G_{N,m}$ for N features). A ball, B_{ij}, of radius δ is centered at z_{ij}. Consider the class $H_{2,m}$ of all $m\times m$ binary $(0,1)$ matrices. For each matrix H, let $H(i,j)$ denote the binary value at the (i,j) position, let H_0 and H_1 be the sets of indices corresponding to 0- and 1-valued entries, respectively, and let U_0 and U_1 be the uniform distributions over the regions $R_0=\cup\{B_{ij}:(i,j)\in H_0\}$ and $R_1=\cup\{B_{ij}:(i,j)\in H_1\}$, respectively. This results in the model $M_{2,m}$. For each pair of distributions, the Bayes classifier can be expressed as a decision tree with zero Bayes error. The distributional complexity, $\chi(H)$, of the classifier depends on H and is the minimum number of hyperplanes used in a decision tree that defines a Bayes classifier. Since the labels 0 and 1 are interchangeable, we identify H and its complement and include only one of them in the model.

The model $M_{2,m}$ is composed of families $M_{2,m}^\chi$ of distributions of various complexities. For instance,

$$M_{2,3}^1=\left\{\begin{pmatrix}1&1&1\\0&0&0\\0&0&0\end{pmatrix},\begin{pmatrix}0&0&0\\0&0&0\\1&1&1\end{pmatrix},\begin{pmatrix}1&0&0\\1&0&0\\1&0&0\end{pmatrix},\begin{pmatrix}0&0&1\\0&0&1\\0&0&1\end{pmatrix}\right\}.$$

We note that the elements in $M_{2,3}^1$ are all rotations of a single element. If we wish to consider them individually in the computation of efficiency, we can give them all the same probability; if we wish to consider them as a single probability structure, equivalent under rotation, we can assign three of them probability zero. In this case rotational equivalence suffices, but in other families we have symmetries with respect to both rotation and the orthogonal axes. For instance,

$$\left\{\begin{pmatrix}1&1&1\\0&0&0\\0&0&1\end{pmatrix},\begin{pmatrix}1&0&0\\0&0&0\\1&1&1\end{pmatrix},\begin{pmatrix}1&1&1\\0&0&0\\1&0&0\end{pmatrix},\begin{pmatrix}1&0&0\\1&0&0\\1&0&1\end{pmatrix}\right\}\subset M_{2,3}^3.$$

But we must not suppose that a complexity class consists only of symmetries. For instance,

$$\left\{\begin{pmatrix}1&1&0\\0&0&0\\1&1&1\end{pmatrix},\begin{pmatrix}0&1&0\\0&0&0\\0&0&0\end{pmatrix},\begin{pmatrix}1&1&1\\1&0&0\\1&1&1\end{pmatrix}\right\}\subset M_{2,3}^3. \tag{4}$$

While we have explained the model for uniform distributions over the balls B_{ij}, in which case the Bayes error is zero, we will actually apply the model with Gaussians centered at each point z_{ij}. Not only is this in accord with usual Gaussian mixture models but it also makes the Bayes error greater than zero, which is more realistic. Simulations are done for the 2-variable and 3-variable cases. For more variables, the number of computations is huge and is not considered. Anyway, since our main interest is with small samples, using more than 3 variables is usually not prudent.

5. Simulation

Simulation results are obtained by generating data based on the model and the distributional complexities. For the 2-variable case, all $2^8 - 2 = 254$ (leaving out the trivial cases and complementations) possible configurations are considered and the complexities computed for the complexity classes. Given a matrix H from the complexity class $M^1_{2,3}$, a sample of size n is generated by randomly selecting n points $z_{i_1 j_1}, z_{i_2 j_2}, \ldots, z_{i_n j_n}$ (with repetition allowed), randomly drawing a point $x_{i_l j_l}$ from each Gaussian at each point $z_{i_l j_l}, l = 1, 2, \ldots, n$, and assigning the label $H(i, j)$ to $x_{i_l j_l}$. A classifier from the class Ψ is designed based on the sample, and its error is computed based on the model. This is repeated L times to get an estimate of the expected error, $\varepsilon[\psi_n; H]$. The entire procedure is repeated for all matrices with $\chi = 1$ and $\omega_\Psi[M; n, 1]$ is obtained from Eq. (3) (assuming equal probabilities for the matrices). We then turn to $\chi = 2$, and so on.

For the 3-variable case, the number of configurations is huge ($2^{26} - 2$), which precludes using all distributions in $M^\chi_{3,3}$ to obtain $\omega_\Psi[M; n, \chi]$. Thus, we resort to sampling distributions from $M^\chi_{3,3}$. Once a distribution is chosen, data generation is done in the same way as for the 2-variable case. The sampling method is explained in Appendix B.

The Bayes error should have a monotonically nondecreasing relationship with the distributional complexity. A straightforward theoretical estimation of the Bayes error for a given value of χ requires enumerating all the possible configurations in each complexity class. For example, in the 2-variable case, consider the three configurations of distributional complexity given in Eq. (4) and let e_g be the error due to overlap between two adjacent Gaussians. If it is assumed that the overlap of diagonally placed Gaussians is significantly less than that between two adjacent Gaussians, then the Bayes error for the three cases would be approximately $6e_g$, $3e_g$, and $5e_g$, respectively. Given these considerations, we can then compute the *Bayes χ-error* for the model analogously to the χ-error by

$$\omega[M^\chi_{d,m}] = \sum_{j=1}^{r_\chi} \varepsilon_{BAY}[H_{\chi j}] P(H_{\chi j}|\chi), \tag{5}$$

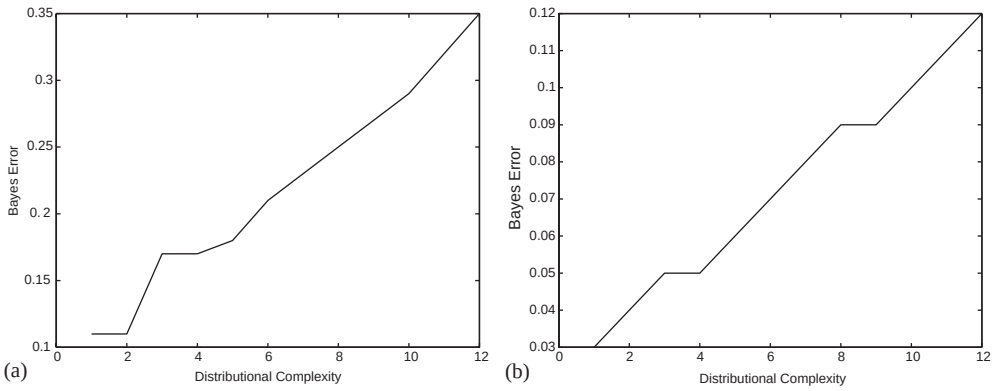

Fig. 2. Bayes error for the 2-variable case: (a) $\sigma = 1/6$ and (b) $\sigma = 1/10$.

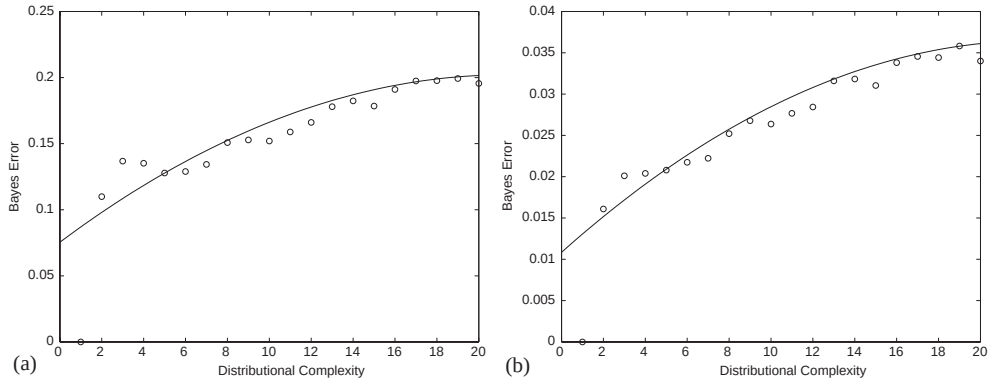

Fig. 3. Bayes error for the 3-variable case: (a) $\sigma = 1/6$ and (b) $\sigma = 1/12$.

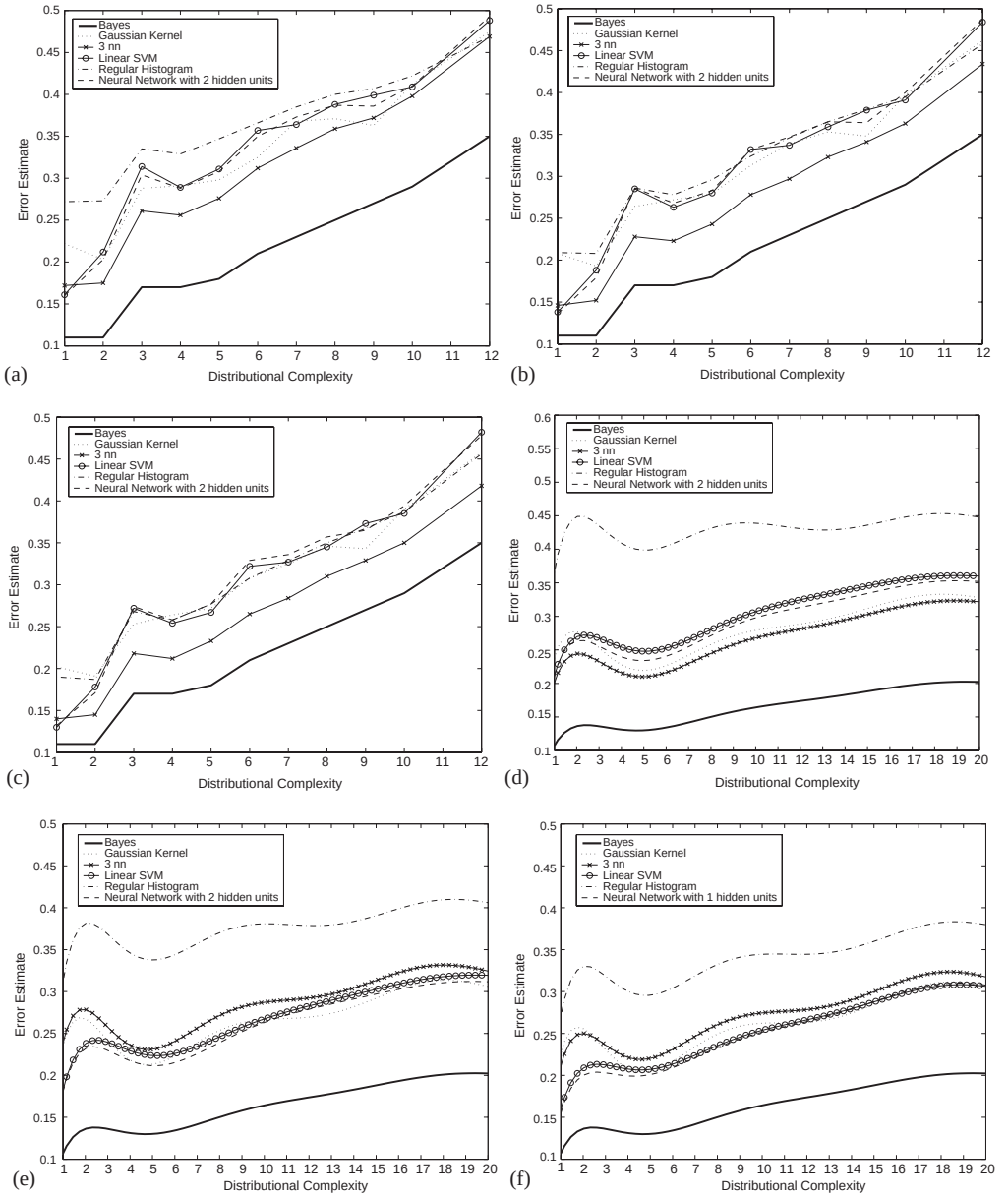

Fig. 4. Error estimate vs. complexity for $\sigma = 1/6$, (a) $n = 20$, 2 variables, (b) $n = 40$, 2 variables, (c) $n = 60$, 2 variables, (d) $n = 20$, 3 variables, (e) $n = 60$, 3 variables, (f) $n = 60$, 3 variables.

where the Bayes error, $\varepsilon_{BAY}[H_{\chi j}]$, corresponding to the matrix $H_{\chi j}$ in the χ complexity class can be approximated by an expression of the form ke_g.

This has been done for the 2-variable case using all possible configurations and numerical methods to find the Bayes error for each configuration. The relationship between complexity and Bayes error (between χ and $\omega[M^{\chi}_{2,3}]$) for two values of σ is shown in Fig. 2 for $\chi = 1$ to $\chi = 12$. Distributional complexities in excess of 12 are of very little practical value. The relationship is monotonic. Note that the error is unchanged between $\chi = 3$ and $\chi = 4$ for both values of σ and that there is very little change in $\omega[M^{\chi}_{2,3}]$ between $\chi = 3$ and $\chi = 5$ for $\sigma = 1/6$. All errors have been estimated using leave-one-out cross-validation with 500 replications to ensure precise estimation.

For the 3-variable case, sampling of configurations is done to estimate the Bayes errors, and therefore $\omega[M^{\chi}_{3,3}]$. Hence, the values of $\omega[M^{\chi}_{3,3}]$ are approximate. The relationship

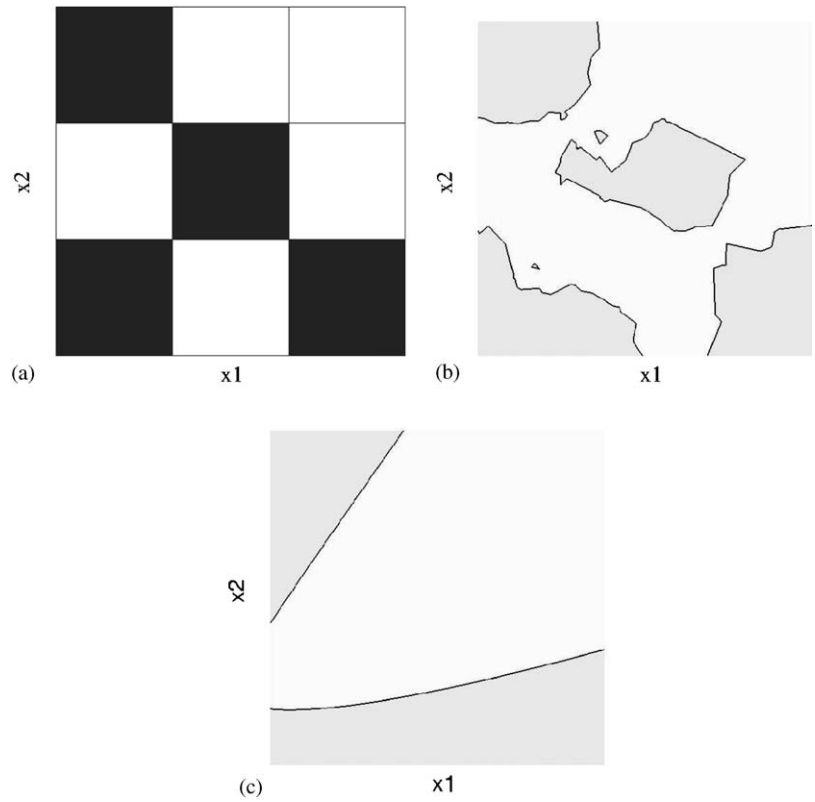

Fig. 5. (a) Actual configuration of complexity 10. (b) Decision boundary plot for 3NN. (c) Decision boundary plot for neural network with 2 hidden units.

between χ and $\omega[M^{\chi}_{3,3}]$ is shown in Fig. 3 for two values of σ. In the case of $\sigma = 1/6$, $\omega[M^{\chi}_{3,3}]$ is a bit higher for $\chi = 3$ and $\chi = 4$ than for $\chi = 5$ and $\chi = 6$. This appears to correspond to the flat spot between $\chi = 3$ and $\chi = 5$ for the 2-variable case. The slight lack of monotonicity here may be due to sampling. There may even be a dip in the true values of $\omega[M^{\chi}_{3,3}]$; nevertheless, the dip is slight and the relationship between χ and $\omega[M^{\chi}_{3,3}]$ is close to being monotonic.

6. Experimental results

We provide detailed experimental analysis for both high and low values of σ, namely $\sigma = 1/6$ and $\sigma = 1/12$. For $\sigma = 1/6$, the Bayes error is in the range of 0.15 for modest levels of distributional complexity, those associated with "doable" problems ($\chi \leqslant 5$ for 2 variables and $\chi \leqslant 10$ for 3 variables). For $\sigma = 1/12$, the Bayes error is less than 0.05 for modest levels of complexity.

Figs. 4(a)–(c), show how the error varies with the distributional complexity for $n = 20, 40, 60$ in the 2-variable case for $\sigma = 1/6$. As expected, performance improves for increasing n. In almost all the cases, the 3NN classifier outperforms the others. The linear SVM works very well for complexity 1 (linear case). The regular histogram has the worst performance of all. A significant observation is that a neural network with two hidden units has almost the same performance as a linear classifier—the linear SVM. If one obtains a low error estimate with the neural net, then one is likely to obtain a similar error with the linear SVM. Hence, it is impossible in this model to claim that the use of the neural net is beneficial. In essence, the sample sizes are too small to take advantage of the added complexity of the neural net. More generally, beyond complexity 2, no classifier performs well, even though the Bayes error is only slightly in excess of 0.1. The sample size is just too small for $\chi > 2$.

An indication of the superior performance of the 3-nearest-neighbour (3NN) classifier as opposed to the neural net can be seen in the kinds of decision boundaries they produce. Fig. 5(a) shows a configuration for $\chi = 10$ and Fig. 5(b) and (c) shows corresponding decision boundaries for the 3NN classifier and neural net, respectively, for $n = 60$. The 3NN classifier does a much better job in approximating the partition.

A final point regarding the 2-variable case is that all of the tested classifiers perform more poorly for $\chi = 3$ than for $\chi = 4$, except for the Gaussian kernel. This indicates that, although $\omega[M^{3}_{2,3}] = \omega[M^{4}_{2,3}]$, the design cost is higher for $\chi = 3$

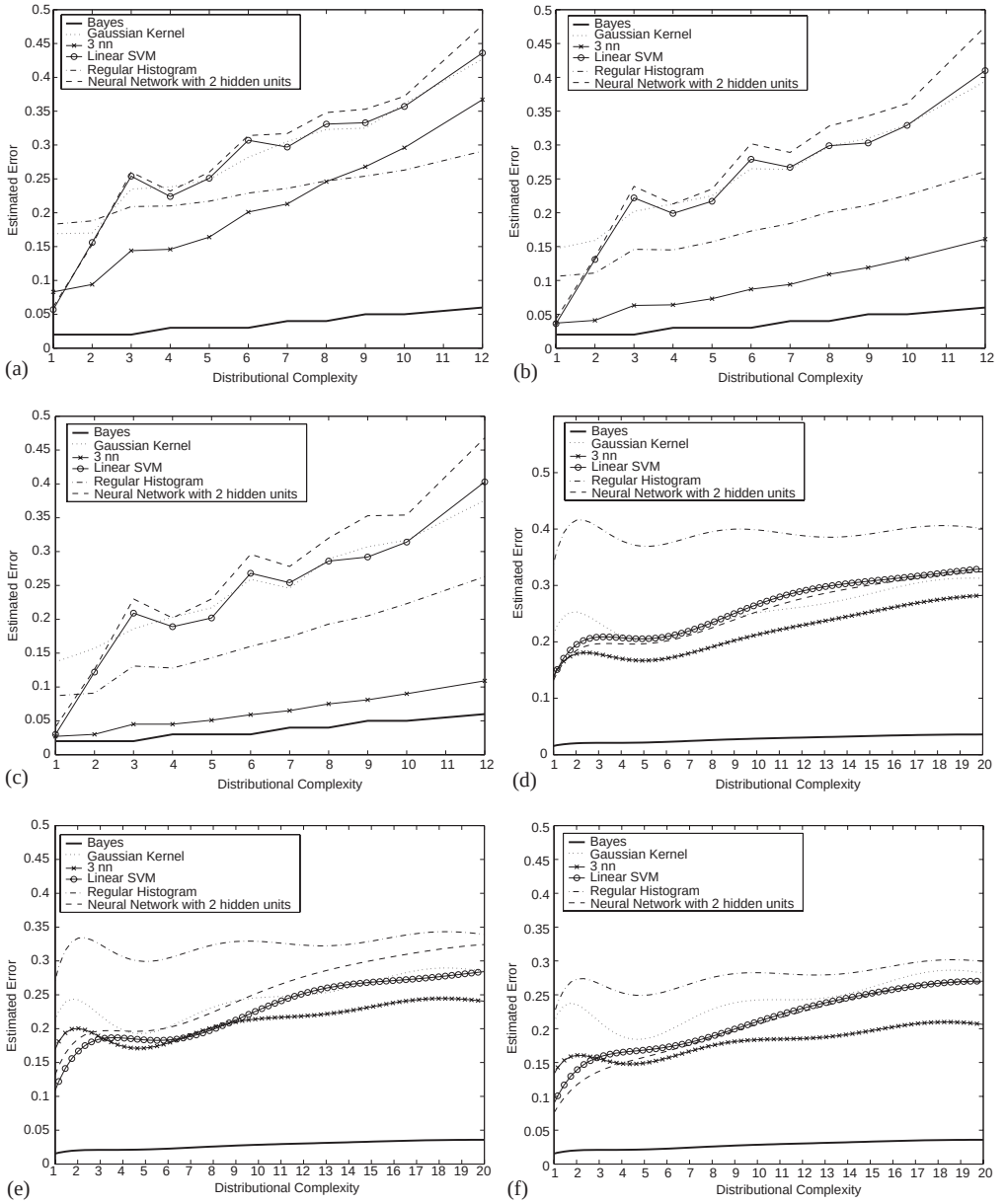

Fig. 6. Error estimate vs. complexity for $\sigma = 1/12$: (a) $n = 20$, 2 variables; (b) $n = 40$, 2 variables; (c) $n = 60$, 2 variables; (d) $n = 20$, 3 variables; (e) $n = 60$, 3 variables; (f) $n = 60$, 3 variables.

than for $\chi = 4$ for all classifiers except the Gaussian kernel. There is nothing strange about this because the complexity-3 configurations may simply be more difficult for the design algorithms than the complexity-4 configurations.

Fig. 4(d)–(f) shows the results for the 3-variable case for $\sigma = 1/6$ and $n = 20, 40, 60$, where a spline regression has been used to construct the curves. Similar observations to those made in the 2-variable case can be made, the key difference being that configurations are sampled for each value of χ, as opposed to the 2-variable case where all configurations are considered.

Fig. 6 provides analogous curves to the those in Fig. 4, except now $\sigma = 1/12$. Most of the same commentary applies; however, there is a salient exception in the 2-variable case. For it, when $n = 60$ (and to some extent when $n = 40$), the 3NN classifier performs close to the Bayes classifier, even for high levels of distributional complexity. This means that, with 2 variables and little overlap between the

label-conditioned distributions, the 3NN classifier is well trained with quite small samples even for complex distributions.

7. Conclusion

Rather than take a distribution-free approach to measure the efficacy of a classification rule, we have taken a model-based approach and compared classifier performances in relation to distributional complexity. Our interest is with small samples, and hence it is reasonable to consider classifiers based on no more than 3 variables. For the most part, we have seen that there is little improvement when one uses a complex classifier instead of a linear SVM. In the case of $\chi = 1$, the problem is linear and naturally the linear SVM performs the best. Others perform well for $\chi = 1$, but not as well, so that one is better off with a linear classifier. For higher levels of distributional complexity, the linear classifier degrades, but so do the more complex classifiers owing to insufficient training data. Hence, if one were to obtain a good result with a more complex classifier, it is most likely that the distributional complexity is low (likely, $\chi = 1$) and there is no gain over using a linear classifier. Of the nonlinear classifiers examined in the small-sample setting, the 3NN classifier usually outperforms the others for 2 variables, but even its performance is typically poor for $\chi > 2$. A strong exception to this observation is when σ is sufficiently small so that there is little overlap between the conditional distributions and the sample size is not too small. Hence, if one uses the 3NN classifier with two variables and obtains a low error when $n \geqslant 40$, then the distributional complexity might be large and, if such is the case, there is a significant gain over using a linear classifier. This exception does not extend to 3 variables, in which training with $n = 60$ is not sufficient to provide improvement over a linear classifier even for a small increase in distributional complexity.

Appendix A. Classification rules

For the kNN rule, 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 the chosen points. A slight adjustment is made if k is even. The kNN rule is universally consistent if $k \to \infty$ in such a way that $k/n \to 0$ as $n \to \infty$ [7].

The *moving-window rule* pre-sets a distance and takes the majority label among all sample points within that distance of $\mathbf{x}$. The moving-window rule can be "smoothed" by giving more weight to sample points closer to $\mathbf{x}$. A kernel rule is constructed by defining a weighting kernel based on the distance of a sample point from $\mathbf{x}$, in conjunction with a smoothing (scaling) factor. The *Gaussian* kernel is given by $K_h(\mathbf{x}, \mathbf{x}_k) = \exp[-\|(\mathbf{x} - \mathbf{x}_k)/h\|^2]$, where $\| \bullet \|$ denotes Euclidean distance, $\mathbf{x}$ is the point at which the classifier is being defined, and $\mathbf{x}_k$ is a sample point. The moving-window rule is a special case of a kernel rule with the weights being 1 within a specified radius. The moving-window and Gaussian kernel rules are strongly universally consistent if $h \to \infty$ in such a way that $nh^d \to \infty$ as $n \to \infty$ [8].

With the *cubic histogram rule*, the space is partitioned into cubes of side length r_n. For each point $\mathbf{x}$, $\psi_n(\mathbf{x})$ is defined 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 [9].

A classical way of constructing classifiers is to use 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. A *perceptron* has the form $\psi(\mathbf{x}) = T[a_0 + \sum_{i=1}^{d} a_i x_i]$, where $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ and T thresholds at 0 and yields -1 or 1. A perceptron divides the space into two half-spaces determined by the hyperplane defined by the parameters $a_0, a_1, \ldots, a_d$. The SVM provides a method for designing perceptrons. The SVM aims to maximize margin (the shortest distance between the closest vector and the hyperplane that separates the classes with zero error) thus resulting in a *maximal-margin hyperplane* (MMH). The training of a SVM involves the solution of a quadratic programming problem. Solution via the normal quadratic approach is slower. An approximate but fast version of the SVM training algorithm is used in this work. The training is based on a geometrical representation of the structural risk minimization (SRM) principle [10].

A *two-layer neural network* has the form $\psi(\mathbf{x}) = T[c_0 + \sum_{i=1}^{k} c_i \sigma[\psi_i(x)]]$, where T thresholds at $\frac{1}{2}$, σ is a sigmoid function, and $\psi_i(\mathbf{x}) = \sum_{j=0}^{d} a_{ij} x_j$, where $x_0 = 1$. Increasing the complexity of the neural network by placing more functions in the hidden layer provides increasing approximation to the Bayes classifier, and this approximation can be obtained to any desired degree. The increase in network complexity results in the need for larger data sets, thereby making estimation of the weights problematic. Typically, steepest descent on the error surface (as a function of the weights) is used. A k-node neural network that minimizes the empirical error is universally consistent if $k \to \infty$ such that $(k \log n)/n \to 0$ as $n \to \infty$ [11].

Appendix B. Data generation in the 3-variable case

For the 3-variable case, the number of possible configurations is enormously huge (2^{27}). In such a case, we resort to sampling a configuration for a given distributional complexity. After a configuration is chosen, data generation is done in the same way as for the 2-variable case.

The sampling algorithm is based on the definition of the *distributional complexity*. Consider a given distributional

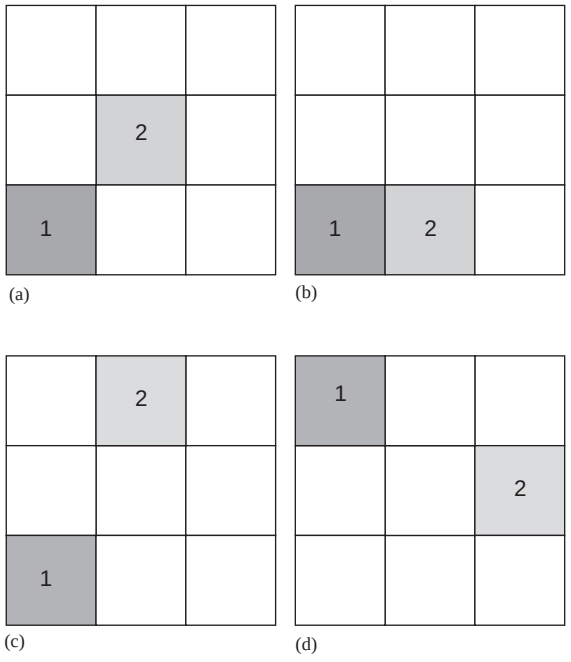

Fig. 7. Example of (a) connected, (b) nonconnected partition, (c) valid partition with complexity 5 and (d) another valid partition with complexity 5.

complexity, $\chi[H]$. According to the converse of the definition, for a given distributional complexity, $\chi[H]$, a configuration exists; the Bayes classifier which is a tree classifier with $\chi[H]$ hyperplanes. In fact, it is possible that more than one configuration exists. It is required to sample a configuration from the possible tree constructions for $\chi[H]$. In essence, one has to construct a tree with $\chi[H]$ nodes, each node (excluding the root) being a decision hyperplane parallel to the axis. However, there should not be any redundant decision hyperplanes in the tree that would lead to overestimation of the given complexity $\chi[H]$.

Before further discussing the algorithm, some terminologies used are explained:

(1) *Trace*: a set of nodes starting from the root and ending in a leaf. A trace creates a partition in the three-dimensional space. An example for a trace in two-dimensions would be a set of decisions $[(x > 2/3), (y > 1/3)]$ and would result in a partition (shaded region) as shown in Fig. 7(a).
(2) *Connected partition*: a partition will not consist of subpartitions that are not adjacent to each other. For example, in Fig. 7(a), the shaded regions do not form a connected partition as partitions 1 and 2 are not adjacent to each other. In Fig. 7(b), the shaded regions form a connected partition as subpartitions 1 and 2 are adjacent to each other.

Some other observations need to be made.

(1) Along each dimension/variable, there are 4 decision hyperplanes viz., $[(h < 1/3), (h > 1/3), (h < 2/3), (h > 2/3)]$.
(2) Along each dimension, there can only be 6 hyperplane combinations viz., $[\phi, (h < 1/3), (h > 1/3), (h < 2/3), (h > 2/3), (h > 1/3, h < 2/3)]$. ϕ is the empty set. For the 3-variable case, this would mean $6^3 - 1 = 215$ (excluding the no hyperplane combination) possible hyperplane combinations.
(3) Each hyperplane combination is a connected partition.
(4) A tree (and hence the equivalent matrix H) is a union of traces. The number of nodes, excluding the root, in the tree is the required complexity $\chi[H]$.

Before we present the sampling algorithm, let us look at an illustration of the sampling process. We use the 2-variable case for clarity of explanation. Suppose the given distributional complexity is 5 and it is required to sample a configuration with this complexity. There can be $6^2 - 1 = 35$ valid hyperplane combinations for the 2-variable case (observation 2). We have to chose a combination of these 35 hyperplanes that gives a distributional complexity 5. Some of the combinations are partitions with regions of complexities (2,3), (1,4), (1,2,2) and (1,1,3). If we decide to have only two regions (traces), we have to choose between regions with complexity combination (2,3) or (1,4). Let us consider the partition with regions of complexities 2 and 3. One possible partitioning would be as shown in Fig. 7(b). The actual complexity of this configuration is 3 and not 5 as regions 1 and 2 have a hyperplane in common and hence form a connected partition. The partition in Fig. 7(c) forms a nonconnected partition with regions of complexities 2 and 3. This is a valid partition that can be sampled. Another valid partition that can be sampled is shown in Fig. 7(d).

The algorithm for sampling a configuration for a given distributional complexity is given below:

(1) Construct a set of valid hyperplane combinations possible for a trace.
(2) For the desired complexity, pick a random number B for the number of branches (starting from the root) that the decision tree will have. This is equivalent to specifying the number of traces that the tree will have.
(3) For each branch (trace) i, generate a random complexity C_i. The complexity C_B for the last branch is the residual complexity measure after the complexities for the previous branches have been chosen.
(4) Pick a random hyperplane combination S_i with C_i hyperplanes from among the possible valid combinations list constructed earlier.
(5) The picking of the hyperplane combination S_i invalidates the availability of other hyperplane combinations

for choice by later branches. The hyperplane combinations that are invalidated are chosen based on rules. An example of such a rule is that all partitions that are contained within the partition chosen due to S_i be invalidated.

(6) The above steps are repeated for all branches. In case, a certain tree cannot be constructed due to lack of available hyperplane combinations after processing a few branches, the construction of the tree starts all over again from step 2. The partitions have a label 1. All other regions have a label 0.

(7) The 27-bit binary pattern for the labels is obtained from the decision tree by feeding the centroid of each of the 27 balls to the decision tree and checking the outcome (0 or 1).

(8) A set of 27-bit binary patterns are generated for each complexity. Data are generated as in the 2-variable case using the binary patterns to assign the labels.

References

[1] E.R. Dougherty, Small sample issues for microarray-based classification, Comp. Funct. Genom. 2 (2001) 28–34.

[2] E.R. Dougherty, S.N. Attoor, Design issues and comparison of methods for microarray based classification, in: W. Zhang, I. Shmulevich (Eds.), Computational and Statistical Approaches to Genomics, Kluwer Publications, Boston, 2002.

[3] L. Devroye, L. Gyorfi, G. Lugosi, A Probabilistic Theory of Pattern Recognition, Springer, New York, 1996.

[4] U.M. Braga-Neto, E.R. Dougherty, Is cross-validation valid for small-sample microarray classification, Bioinformatics 20 (2004).

[5] V. Vapnik, Statistical Learning Theory, Wiley, New York, 1998.

[6] E. Baum, On the capabilities of multilayer perceptrons, Complexity 4 (1988) 193–215.

[7] C. Stone, Consistent nonparametric regression, Ann. Stat. 5 (1977) 595–645.

[8] L. Devroye, A. Kryzak, An equivalence theorem for L_1 convergence of the Kernel regression estimate, Stat. Plann. Infer. 23 (1989) 71–82.

[9] L. Gordon, R. Olshen, Asymptotically efficient solutions to the classification problem, Ann. Stat. 6 (1978) 525–533.

[10] D. Roobaert, Direct SVM: a fast and simple support vector machine perceptron, Proceeding of IEEE, International Workshop on Neural Networks for Signal Processing, Sydney, Australia, December 2000, pp. 356–365.

[11] A. Farago, G. Lugosi, Strong universal consistency of neural network classifiers, IEEE Trans. Inf. Theory 39 (1993) 1146–1151.

About the Author—SANJU N. ATTOOR obtained his B.E. in Electronics and Communication Engineering from PSG College of Technology, India and his M.S. in Electrical Engineering from Texas A&M University, in 2003. His areas of interest include pattern recognition and bioinformatics.

About the Author—EDWARD DOUGHERTY is a professor in the Department of Electrical Engineering at Texas A&M University in College Station. He holds a Ph.D. in Mathematics from Rutgers University and an M.S. in Computer Science from Stevens Institute of Technology. He is author of 11 books and editor of four others. He has published more than one hundred journal papers, is an SPIE fellow, is currently Chair of the SIAM Activity Group on Imaging Science, and has served as editor of the Journal of Electronic Imaging for 6 years. Prof. Dougherty has contributed extensively to the statistical design of nonlinear operators for image processing. His current research is focused in genomic signal processing, with the central goal being to model genomic regulatory mechanisms. He is head of the Genomic Signal Processing Laboratory at Texas A&M University.