RELATIONSHIP BETWEEN CLASSIFIER PERFORMANCE AND DISTRIBUTIONAL COMPLEXITY FOR SMALL SAMPLES

A Thesis

by

SANJU NAIR ATTOOR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2003

Major Subject: Electrical Engineering

RELATIONSHIP BETWEEN CLASSIFIER PERFORMANCE AND DISTRIBUTIONAL COMPLEXITY FOR SMALL SAMPLES

A Thesis

by

SANJU NAIR ATTOOR

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

Edward R Dougherty (Chair of Committee)	
Krishna Narayanan (Member)	Erchin Serpedin (Member)
Jyh-Charn Liu (Member)	Chanan Singh (Head of Department)

August 2003

Major Subject: Electrical Engineering

ABSTRACT

Relationship between Classifier Performance and Distributional Complexity
for Small Samples. (August 2003)
Sanju Nair Attoor, B.E., PSG College of Technology
Chair of Advisory Committee: Dr. Edward R. Dougherty

Given a limited number of samples for classification, several issues arise with respect to design, performance and analysis of classifiers. This is especially so in the case of microarray-based classification. In this paper, we use a complexity measure based mixture model to study classifier performance for small sample problems. The motivation behind such a study is to determine the conditions under which a certain class of classifiers is suitable for classification, subject to the constraint of a limited number of samples being available. Classifier study in terms of the VC dimension of a learning machine is also discussed.

To My parents and My sister

ACKNOWLEDGMENTS

I wish to express my sincere thanks to my advisor, Dr. Edward R. Dougherty, for his invaluable guidance and for being a great inspiration. I am especially thankful to him for all the discussions we had; it has been a great learning experience working with him. This thesis would certainly not have been possible without his encouragement.

I would like to thank Dr. Jyh-Charn Liu, who first introduced me to research during my initial semesters of graduate study and encouraged me to explore different areas of research. I would also like to thank the members of the GSP Lab for the great environment they provided and the invaluable discussions we had.

Lastly, and most importantly, I would like to thank my parents and my sister for their love, encouragment and support.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

CHAPTER I

THE SMALL SAMPLE PROBLEM: INTRODUCTION

A. Introduction

The design, performance and analysis of various classifiers have been long researched. However, the results have typically been based on the assumption of a large number of samples being available. Approximate error probability and error estimates (leave one out and resubstitution estimates) for k-nearest neighbour rules, kernel rules, regular histogram rules, linear discriminant rule and tree classifiers, in the asymptotically large sample case, are well-known [1].

These theoretical estimates auger well for problems that involve large sample sizes. The performance and analysis of classifiers, analytically, based on a small sample size is in most cases, not possible. In a lot of practical problems, especially in genomics [2]-[5], the number of samples available for classifier design is less (often in the order of 20-30 samples). An example is the breast cancer classification problem [6], wherein 22 samples are used to classify a tissue sample as either belonging or not belonging to a breast cancer class. Hence, the need arises to numerically estimate the performance of the classifiers. Classifier performance is usually a function of one or more of the following factors: sample size, data complexity, and operator (classifier) complexity.

Thus, given a data set of a certain sample size and possibly some information on the complexity of the data and a classifier set, it is desired to choose from a set of classifiers, a classifier that best classifies that data and best generalizes the data. Intuitively, one would use the Occam's Razor principle (attributed to the mediaeval

---

This thesis follows the style of *IEEE Transactions on Automatic Control*.

philosopher William of Occam and is otherwise knows as the Principle of Parsimony) which states that "one should not increase, beyond what is necessary, the number of entities required to explain anything". With regard to our problem, this would meaning choosing as simple a classifier as possible given that it performs as good as any other classifier on the data set at hand. For example, if for a certain data set of size 25, the training error for a neural network with 2 hidden neurons and a single hidden layer with sigmoid activation function is 0.09 and that for a perceptron is 0.1, one might be better off choosing the simpler learning machine viz., the perceptron as opposed to the neural network. It is the purpose of this study to discuss this issue of classification for small sample sizes and if in fact the Occam's Razor principle holds.

B. Problem Statement

We now define the problem more formally. Suppose we are given n observations, each observation i being a pair $(\mathbf{x_i}, y_i)$ drawn from a distribution $P(\mathbf{x}, y)$; $\mathbf{x_i}$ being a d-dimensional vector. Let the error for the Bayes classifier $\psi_\bullet$ be $\epsilon_\bullet$. Let ψ_n be the optimal classifier for the sample size n and ϵ_n the corresponding error. Given that the Bayes classifier for the problem cannot be found, the problem is to find the class C of classifiers that has an optimal constrained classifier $\psi_{n,C,\alpha}$ (with an error $\epsilon_{n,C,\alpha}$) that minmises the expected error(or expected risk) for the given observation, α being the set of parameters that defines the classifier.

The expected error of the designed classifier C can be decopmposed as [7]:

$$E(\epsilon_{n,C,\alpha}) = \epsilon_\bullet + \Delta_C + E(\Delta_{n,C,\alpha}) \tag{1.1}$$

where $\Delta_C = \epsilon_C - \epsilon_\bullet$ is the cost of constraint and $\Delta_{n,C,\alpha} = \epsilon_{n,C,\alpha} - \epsilon_C$ is the design error for the constrained classification.

C. Classifier Selection Using VC Dimension

Vapnik in [8], gives a bound on the generalization performance of a learning machine. This bound, is a function of the empirical error $\bar{\epsilon}_{n,C,\alpha}$, the sample size n and a non-negative integer h called the Vapnik-Chervonenkis (VC) dimension. The VC dimension, in general, is defined as the maximum number of training points that can be shattered by a set of functions $\{f(\mathbf{x}, \alpha)\}$.

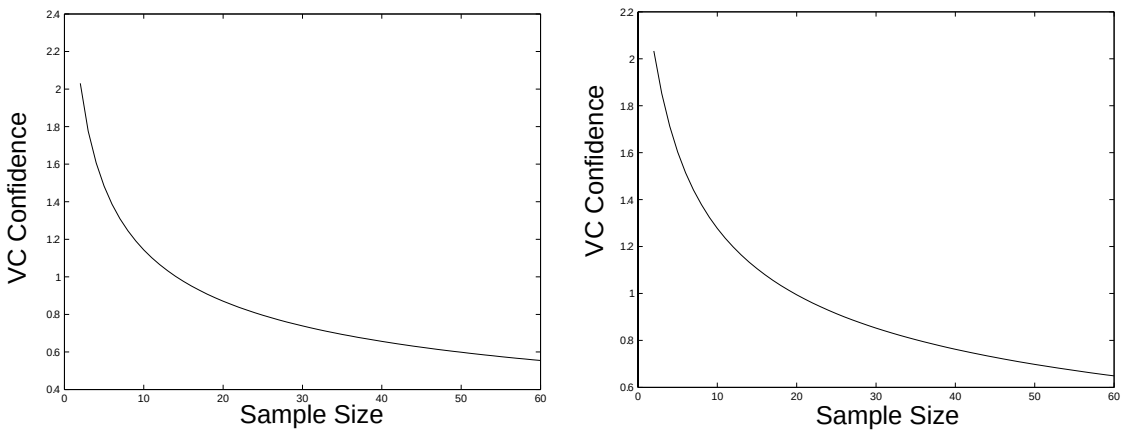
According to Vapnik [8], with probability $1 - \eta$, the following bound holds:

$$E[\epsilon_{n,C,\alpha}] \leq \bar{\epsilon}_{n,C,\alpha} + \sqrt[2]{\frac{(h(log(2n/h) + 1) - log(\eta/4))}{n}} \tag{1.2}$$

The second term in the bound is called the "VC Confidence". It can be seen that the above bound is independent of the distribution $P(\mathbf{x}, y)$. This means that the upper bound holds with a probability $1 - \eta$ regardless of the distribution. The error bound in itself does not convey any useful information when dealing with small sample sizes. The VC confidence is usually almost 1 for sample size of 20 even for a low VC dimension. The VC confidence as a function of sample size for VC dimensions of 3 and 5 are shown in Fig. 1. It can be seen that for $h = 3$ and for a sample size of 20, the VC confidence is as high as 0.8. Even for a sample size of 60, the VC confidence is very high at 0.5.

However, this upper bound may be used to compare classifiers. If h is known, the afore-mentioned upper bound can be readily computed. Now, given a set of classifiers, for a sufficiently small η, we can choose a classifier that gives the lowest upper bound on the expected error. However, except for a few class of classifiers, the VC dimension is not known. The VC dimension for the class of linear classifiers with k parameters is k (i.e., equal to the degree of freedom of the linear classifier).

Some recent papers, Vapnik [9] and Cherkassky [10], have tried to estimate em-



(a) (b)

Fig. 1. Relationship of VC confidence with sample size, η=0.05, a) h=3, b) h=5

pirically what is called the effective VC dimension. According to Vapnik [9], the effective VC dimension, h^* of the set $\{f(\mathbf{x}, \alpha)\}$ is the minimal VC-dimension of this set of functions defined on all subsets $X^* \subset X$ whose measure is almost one. The method to estimate the effective VC dimension has been shown to work for linear classifiers. However, due to certain assumptions made in the derivation of the method, the experimental measurement of the effective VC dimension may not be accurate for other classifiers such as multilayer neural networks. Further discussions on VC dimension and classifier selection are presented in Chapter III.

D. Classifier Selection Based on Distributional Complexity

Choosing a classifier based on the bound on the expected error is one approach for classifier selection provided knowledge of the VC dimension of the classifier is available. Here, we propose an alternate approach based on what is called the "distributional

complexity" is proposed. The distributional complexity is defined as the minimum number of hyperplane decisions parallel to axes, required to obtain an optimal classification. The distributional complexity, unlike the VC dimension, is dependent on the probability distribution $P(\mathbf{x}, y)$ of the data. Classifier selection is now based on estimates of the expected errors of different classifiers for a given distributional complexity. The fact that a classifier ψ_1 has a better performance than all other classifiers ψ_2 for data with distributional complexity χ^* and sample size n, implies that the classifier ψ_1 is the best suited classifier for a data of distribution complexity χ^* and sample size n and thus the distributional complexity χ induces a goodness criterion on the classifier for the sample size n.

Before discussing further the issue of classification for small samples, an overview on classifiers and classification rules used in this work is presented.

E. Classification

Classification involves a classifier ψ, a feature vector $X = (X_1, X_2, \ldots, X_d)$ composed of random variables, and a binary random variable Y to be predicted by $\psi(X)$. The values, 0 or 1, of Y are treated as class labels. The error, $\epsilon[\psi]$, of ψ is the probability, $P(\psi(X) \neq Y)$, that the classification is erroneous. It equals the expected (mean) absolute difference, $E[|Y - \psi(X)|]$, 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 $\mathcal{R}^d$. An optimal classifier, $\psi_\bullet$, is one having minimal error, $\epsilon_\bullet$, among all binary functions on $\mathcal{R}^d$. $\psi_\bullet$ and $\epsilon_\bullet$ 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 $\mathcal{R}^d$.

Supervised classifier design uses a *sample* $S_n = \{(\mathbf{X}^1, Y^1), (\mathbf{X}^2, Y^2), , (\mathbf{X}^n, Y^n)\}$ 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 $\epsilon_\bullet$ is estimated by the error ϵ_n of ψ_n. Because $\epsilon_\bullet$ is minimal, $\epsilon_n \geq \epsilon_\bullet$, and there is a design error (cost of estimation), $\Delta_n = \epsilon_n - \epsilon_\bullet$. 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,

$$E[\Delta_n] = E[\epsilon_n] - \epsilon_\bullet \tag{1.3}$$

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 $(\mathbf{X}, Y)$.

F. Classification Rules

The *nearest-neighbor* (NN) rule is a partition-based rule. For it, $\psi_n(\mathbf{x})$ is the label of the sample point closest to $\mathbf{x}$. The result is a data-dependent partition in which there is a cell of the partition for each sample point, and that cell consists of all points in the space closest to the sample point. The resulting partition is known as a Voronoi partition and the cells are called Voronoi cells. The nearest-neighbor rule is simple, but not consistent. An extension of this rule is the *k-nearest-neighbor* (kNN) rule. For 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 $n \to \infty$.

Instead of taking the majority label among a pre-decided number of nearest neighbors as with the NN rule, 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 $\mathbf{x}$. Note that a histogram rule based on the majority between 0 and 1 labels is equivalent to the median of the labels, with the provision that in case of a tie a 0 is given. More generally, a weighted median of binary values is computed by adding up the weights associated with the $0-$ and $1-$labeled points separately, and defining the output to be the larger sum. 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*, and *Epanechnikov* kernels are given by

$$K_h(\mathbf{x}, \mathbf{x}_k) = exp\left[-\left\|\frac{\mathbf{x}-\mathbf{x}_k}{h}\right\|^2\right] \tag{1.4}$$

$$K_h(\mathbf{x}, \mathbf{x}_k) = \begin{cases} \left(1-\left\|\frac{\mathbf{x}-\mathbf{x}_k}{h}\right\|^2\right) & \|\mathbf{x}-\mathbf{x}_k\| \leq h \\ 0 & \|\mathbf{x}-\mathbf{x}_k\| > h \end{cases} \tag{1.5}$$

respectively, 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. Since the Gaussian kernel is never 0, all sample points get some weight. The Epanechnikov kernel is 0 for sample points 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 three kernel rules mentioned are strongly universally consistent if $h \to \infty$ in such a way that $nh^d \to \infty$ as $n \to \infty$.

The cubic histogram rule partitions the data space into cubes of size h. Each cube is assigned a label that takes the maximum vote among the labels of the points that fall in the cube. The histogram rule is given by,

$$\psi_n(\mathbf{x}) = \begin{cases} 0 & if \sum_{i=1}^{n} I_{\{y_i=1\}} I_{\{\mathbf{x}_i \in A_n(\mathbf{x})\}} \leq \sum_{i=1}^{n} I_{\{y_i=0\}} I_{\{\mathbf{x}_i \in A_n(\mathbf{x})\}} \\ 1 & otherwise \end{cases} \tag{1.6}$$

where $A_n(\mathbf{x})$ is the partition that $\mathbf{x}$ falls into. The cubic histogram rule has been shown to be strongly universally consistent.

G. Constrained Classifiers

Finding an optimum classifier is difficult especially given a small sample size and lack of knowledge of the distribution of the data. Instead of finding an optimum classifier, one can restrict the functions from which a classifier must be chosen to a class $\mathcal{C}$. It is then required to find an optimum constrained classifier $\psi_C \in \mathcal{C}$, 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 $\mathcal{C}$ 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 $\mathcal{C}$. This *cost of constraint (approximation)* is $\Delta_C = \epsilon_C - \epsilon_\bullet$. A classification rule yields a classifier $\psi_{n,C} \in \mathcal{C}$ with error $\epsilon_{n,C}$, and $\epsilon_{n,C} \geq \epsilon_C \geq \epsilon_\bullet$. Design error for constrained classification is $\Delta_{n,C} = \epsilon_{n,C} - \epsilon_C$. 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 $\epsilon_{n,C} = \epsilon_\bullet + \Delta_C + \Delta_{n,C}$. The expected error of the designed classifier from $\mathcal{C}$ can be decomposed as

$$E[\epsilon_{n,C}] = \epsilon_\bullet + \Delta_C + E[\Delta_{n,C}] \tag{1.7}$$

The constraint is beneficial if and only if $E[\epsilon_{n,C}] < E[\epsilon_n]$, which means $\Delta_C < E[\Delta_n] - E[\Delta_{n,C}]$. If the cost of constraint is less than the decrease in expected design cost, then the expected error of $\psi_{n,C}$ is less than that of ψ_n. The dilemma: strong constraint

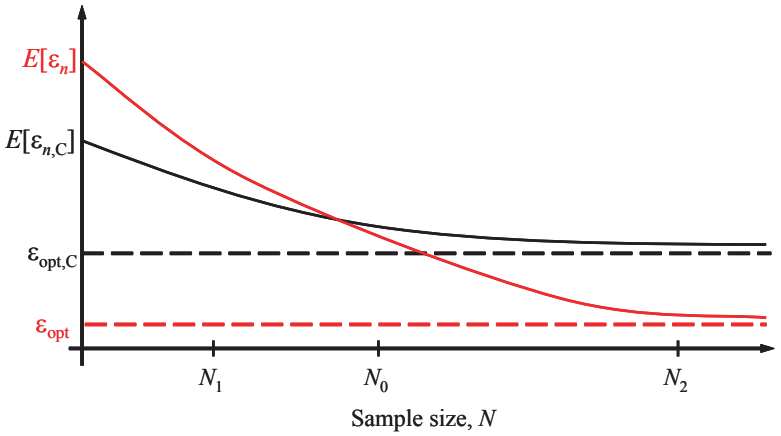


Fig. 2. Relationship between sample size and constraint

reduces $E[\Delta_{n,C}]$ at the cost of increasing ϵ_C.

The relationship between sample size and constraint is shown in Fig. 2.

H. Perceptrons, Support Vector Machines and Neural Networks

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

$$\psi(\mathbf{x}) = T\left[a_0 + \sum_{i=1}^{d} a_i x_i\right] \tag{1.8}$$

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 $a_0, a_1, \ldots, a_d$. An optimal perceptron minimizes the error $\epsilon[\psi] = P(\psi(\mathbf{X}) \neq Y)$. The classical Rosenblatt perceptron algorithm uses an iterative procedure to find a separating hyperplane in the case of a linearly separable

sample.

The *support vector machine (SVM)* provides another 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 SVM has a strong statistical basis and uses *Structural Risk Minimization* rather than the conventional *Empirical Risk Minimization* used by most machine learning algorithms (Rosenblatt perceptron, backpropagation learning). 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 geomterical representation of the Structural Risk Minimzation (SRM) principle [11].

A *(feed-forward) two-layer neural network* has the form

$$\psi(\mathbf{x}) = T\left[c_0 + \sum_{i=1}^{k} c_i \sigma[\psi_i(\mathbf{x})]\right] \tag{1.9}$$

where T thresholds at $\frac{1}{2}$, σ is a sigmoid function, and

$$\psi_i(\mathbf{x}) = \sum_{j=0}^{d} a_{ij} x_j \tag{1.10}$$

where x_0 is the constant 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. As always, greater accuracy of the best filter for a network structure results in greater design cost. Not only does the increase in network complexity result in the need for larger data sets, it also makes estimation of the weights more problematic. Typically, the method of steepest descent on the error surface (as a function of the weights) is used.

I. Error Estimation

A classifier is designed on the training data. Its estimated error is the proportion of errors it makes on the test data. Estimates for the error are either unbiased or asymptotically unbiased and the variance tends to zero as the sample size goes to infinity. A problem arises when the sample size is small. An ideal estimate would be unbiased and would have low variance for a small sample size. Many different estiamtes can be found in the pattern recognition literature. The estimates that we use in this work are the *leave-one-out* and the *resubstitution* estimates.

For the *resubstitution* estimate, $\underline{\epsilon}_n$, we use all sample data to design a classifier ψ_n, and estimate ϵ_n by applying ψ_n to the same data. $\underline{\epsilon}_n$ is then the fraction of errors made by ψ_n. $\underline{\epsilon}_n$ is generally biased low, meaning $E[\underline{\epsilon}_n] \leq E[\epsilon_n]$. For small samples, the bias can be severe.

The *Leave-one-out estimate* is a *cross-validation* estimate. Classifiers are designed from parts of the sample, each is tested on the remaining data, and ϵ_n is estimated by averaging the errors. For *leave-one-out* estimation, 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 $\hat{\epsilon_n}$ is $1/n$ times the number of errors made by the n classifiers. Since the classifiers are designed on sample sizes of $n-1$, $\hat{\epsilon_n}$ actually estimates the error ϵ_{n-1}. It is an unbiased estimator of ϵ_{n-1}, meaning that $E[\hat{\epsilon_n}] = E[\epsilon_{n-1}]$. Unbiasedness is important, but of critical concern is the variance of the estimator for small n. The variance of the *leave-one-out* estimate is severe for small samples.

J. Summary

The problem of small sample classification has been defined. In the subsequent chapters, analysis of classifiers based on two different concepts of complexity measure are discussed viz., the distributional complexity and the VC dimension.

CHAPTER II

CLASSIFIER SELECTION BASED ON DISTRIBUTION COMPLEXITY

A. Introduction

The idea of distributional complexity was introduced in Chapter I. The approach to studying classifiers based on distributional complexity is based on a data model. The higher the complexity measure, the more complex the distribution of the data is and hence the tougher the classification. In the following sections, this approach of studying classifiers based on a complexity measure is discussed in detail.

B. The Distributional Complexity Model

For clarity, we explain the model for two variables (features). The extension to higher dimensions is straightforward. We assume 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_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.

For each pair of distributions, the Bayes classifier can be expressed as a decision tree with zero Bayes error. The complexity of the classifier depends on H. We define the *distributional complexity*, $\chi(H)$, for the matrix H as the minimum number of hyperplanes, used in a decision tree that defines a Bayes classifier. We define the *m-complexity* for the dimension as the expected decision complexity among matrices in $\mathcal{H}_{2,m}$:

$$X_{2,m} = \sum_{H \in \mathcal{H}_{2,m}} \chi(H) P(H) \tag{2.1}$$

where $P(H)$ is the probability of the matrix H. In the uniform case, $P(H) = 1/2^{m^2}$; however, the distribution of H need not be uniform. There would be matrices which are equivalent with respect to rotation and symmetrical with respect to the orthogonal axes. These matrices would then form an "equivalence class". The matrices of an equivalence class would have the same complexity. For instance,

$$\chi \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \chi \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} = \chi \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \chi \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

We may then very well have prior knowledge on the distributions of the equivalence classes. A discussion on equivalence classes follows in a later section. In fact, this prior knowledge may be on the decision complexity itself, in which case it is useful to write the preceding equation as

$$X_{2,m} = \sum_{k} k P(\chi = k) \tag{2.2}$$

Consider now taking a random sample of size n from the joint uniform distribution over the region $R_0 \cup R_1$. This yields a sample of the form

$$S_n = \{(x_{11}, x_{12}, y_1), (x_{21}, x_{22}, y_2), \ldots, (x_{n1}, x_{n2}, y_n)\}$$

where $(x_{k1}, x_{k2}) \in R_0 \cup R_1$ and $y_k \in \{0, 1\}$. Given a set C of classifiers (classification rule C), our object is to design the optimal classifier in C based on the sample, call it ψ_n. The error of ψ_n is an estimate of the Bayes error (which we know to be 0). If we take the expected error of the designed classifier, this gives us a measure $\epsilon_n[C; H]$, which depends on the matrix chosen from $\mathcal{H}_{2,m}$. Assuming we use a consistent classification rule, $\epsilon_n[C; H] \to 0$ as $n \to \infty$, but this is not important for small samples. If we are given a sample size n, and we know H, then classification rule C_1 is better than rule C_2 if $\epsilon_n[C_1; H] \leq \epsilon_n[C_2; H]$. In fact, it is unlikely that we know H, in which case we need to evaluate classification rules relative to the distribution of H. Hence, we consider the expectation of the expected error over $\mathcal{H}_{2,m}$:

$$E^{(n)}_{2,m}[C] = \sum_{H \in \mathcal{H}_{2,m}} \epsilon_n[H; C] P(H) \tag{2.3}$$

Relative to the distribution, classification rule C_1 is better than rule C_2 if $E^{(n)}_{2,m}[C_1] \leq E^{(n)}_{2,m}[C_2]$.

Assuming we judge by expectation, the critical issue regards the closeness of $E^{(n)}_{2,m}[C]$ to 0. Now, if $n > r$, then $E^{(n)}_{2,m}[C] \leq E^{(r)}_{2,m}[C]$. Two issues arise:

1. for a fixed classification rule, can we determine the necessary size of n to obtain a desired precision of error estimation?

2. for fixed sample size, which is the best classification rule among a family of classification rules?

While in the above discussion, balls B_{ij} are assumed to have a uniform density, simu-

lation results are for the case when the balls have a gaussian distribution. Simulations are done for the 2-variable and 3-variable cases. For more variables, the number of computations is huge and is not considered.

C. Data Generation - 2 Variable Case

The next important step is to generate the data given the model and the complexity. For the 2 variable case, all $2^9 - 2 = 510$ (leaving out the trivial cases) possible configurations are considered. For each configuration, the complexity is manually computed. Configurations are then grouped according to complexity. The algorithm for data generation is as follows:

1. List all 510 configurations and compute the complexity for each configuration (this is done manually).
2. Group the configurations according to complexity.
3. For each configuration, do the following n times:
   - Pick a Ball B_{ij} at random.
   - Generate a random vector (of length 2) $\mathbf{x}_i$ following a bivariate gaussian distribution in B_{ij}. The components of the random vector are iid.
   - Assign a label 0 or 1 to y_i depending on whether the label for B_{ij} is 0 or 1 (from the configuration).
4. Do the above step L times to average over the randomizations of the input vector.
5. Repeat the above two steps for all the configurations.
6. Group the data according to complexity.

7. For data set of sample size n, in each complexity group, compute the leave-one-out and the resubstitution estimates.

D. Data Generation - 3 Variable Case

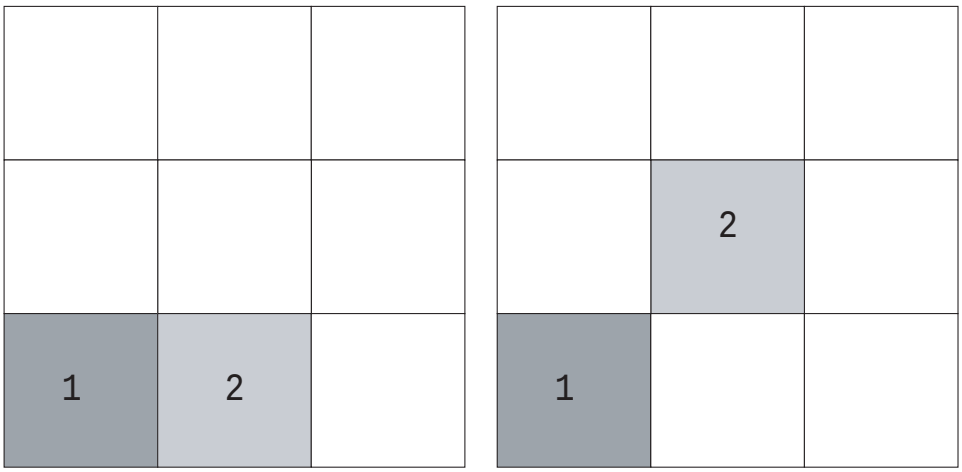
For the 3 variable case, the number of possible configurations is enormously huge (2^{27}). In such a case, we resort to sampling a configuration for a given distributional complexity. After a configuration is chosen, data generation is done in the same way as for the 2 variable case.

The sampling alogithm is based on the converse of the definition of the *distributional complexity*. Consider a given distributional complexity, $\chi[H]$. According to the converse of the definition, a tree exists with $\chi[H]$ nodes. In fact, many such trees exist. It is required to sample a tree from the possible tree constructions possible 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 terminlogies 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 3-dimensional space. An example for a trace in 2-dimensions would be a set of decisions $[(x > 2/3), (y > 1/3)]$ and would result in a partition (shaded region) as shown in Figure 3 (a).

2. Connected partition: a partition will not consist of subpartitions that are not adjacent to each other. For example, in Figure 3(a), the shaded regions form a connected partition as subpartitions 1 and 2 are adjacent to each other. In Figure 3(b), the shaded regions do no form a connected partition as partitions



(a) (b)

Fig. 3. Example of (a) connected and (b) non-connected partition

1 and 2 are not 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 by 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 $\chi[H]$.

It is required that the partition formed by a trace is connected while the partitions formed by different traces are non-connected. If partitions from two traces are adjacent, this would mean that at least one decision is common to both the partitions and there is a redundant count in the number of hyperplanes found in the tree. This would mean an overestimation of the actual complexity $\chi[H]$. Avoiding this overestimation of the complexity $\chi[H]$ is achieved using a rule-based approach.

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).

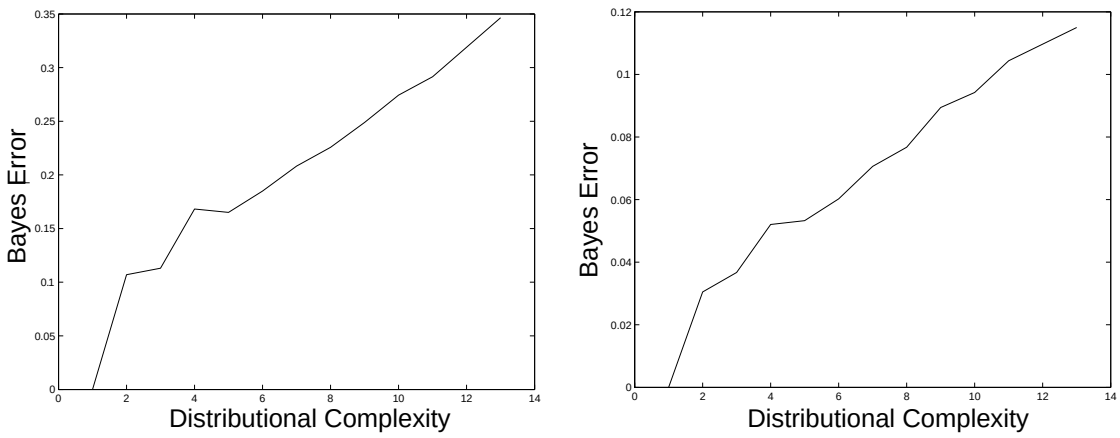
8. A set of 27-bit binary patterns are generated for each complexity. Data is generated as in the 2 variable case using the binary patterns to assign the labels.

E. Relationship between Bayes Error and Distributional Complexity

For the model considered, it is important that the Bayes error has a monotonous non-decreasing relationship with the Distributional Complexity. A straight forward theoretical estimation of the Bayes error for a given value of distributional comlexity is not possible without enumerating all the possible configurations. For example, in the 2 variable case, some of the possible configurations for a distributional complexity of 3 would be:

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Let e_g be the error due to overlap between two adjacent gaussians. It is assumed that the overlap of diaganolly placed gaussians is significantly lesser than that between two adjacent gaussians. In such an event, the Bayes error for the three cases would be $6e_g$, $3e_g$ and $5e_g$ respectively. Given the probabilities p_k^χ of the events of error ke_g^χ possible for a distributional complexity χ, the Bayes error ϵ_χ^* is



(a) (b)

Fig. 4. Bayes error for the 2 variable case a)sigma=1/6, b) sigma=1/10

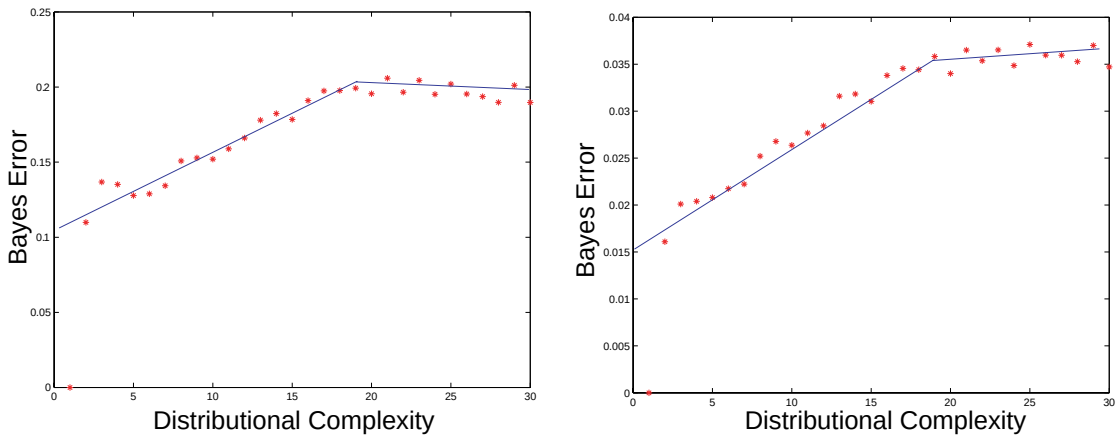
$$\epsilon^*_\chi = \sum_k k * e^\chi_g * p^\chi_k \tag{2.4}$$

However, finding the probabilities analytically is a difficult problem. For the 2 variable case, since all the possible configurations are considered, a numerical simulation is done to find the Bayes error for a given complexity. The relationship of the Bayes error with complexity is shown in Fig. 4.

For the 3 variable case, sufficient sampling of configurations is done so as to obtain a consistent estimate of the Bayes error. The relationship of the Bayes error with complexity for the 3 variable case is shown in Fig. 5.

F. Experimental Results

Figures 6 to 11 show how the error estimate (leave-one-out) varies with the complexity. In almost all the cases, the kNN (k=3) outperforms all the classifiers. The linear SVM



(a) (b)

Fig. 5. Bayes error for the 3 variable case a)sigma=1/6, b) sigma=1/12

works very well for complexity 1 (linear case). The regular histogram is highly biased and 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 support vector machine. There have been papers in literature which use a neural network for small classification problem and claim to achieve better results. The results here show that one would be better off using a linear support vector machine instead of a neural network for small samples due to the better generalization and lower VC dimension of a linear classifier.

Figures 12 to 17 show the results for the 3-dimensional case.

The estimated error for the 3-variable case is lesser than that for the 2-variable case this would mean that adding a variable helps in classification: as to how many variables are necessary for good optimal classification performance is another research area.

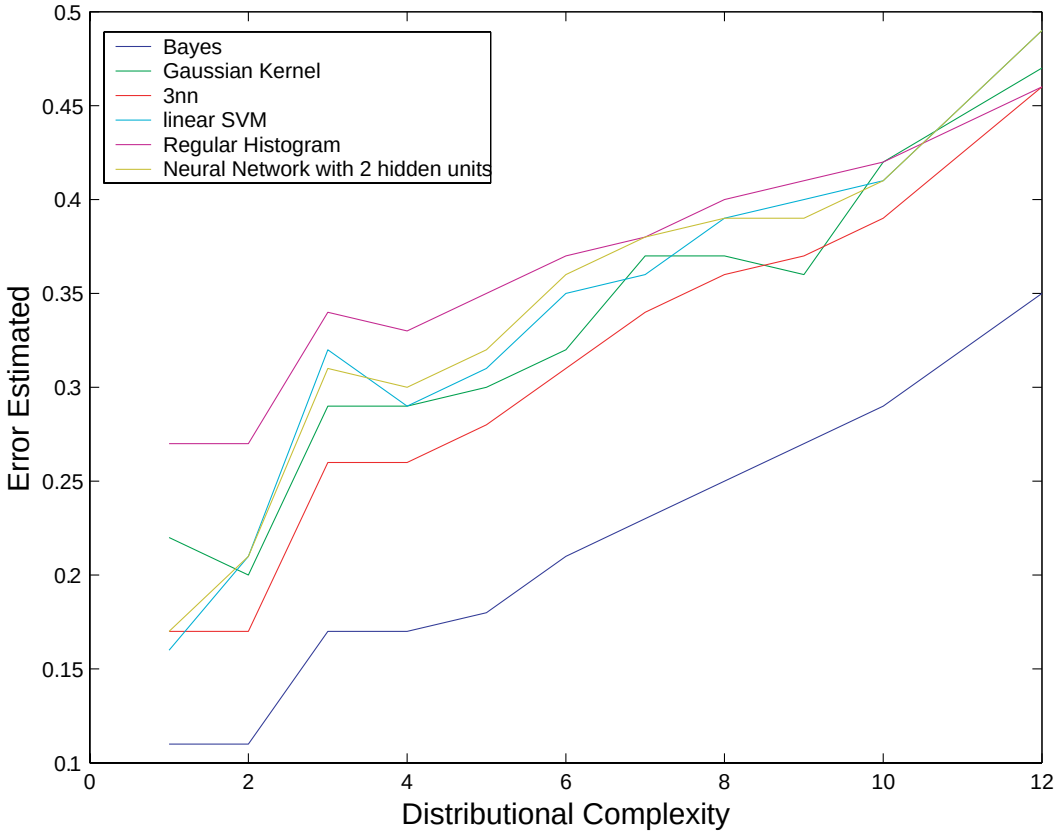


Fig. 6. Error estimate vs complexity N=20, sigma=1/6

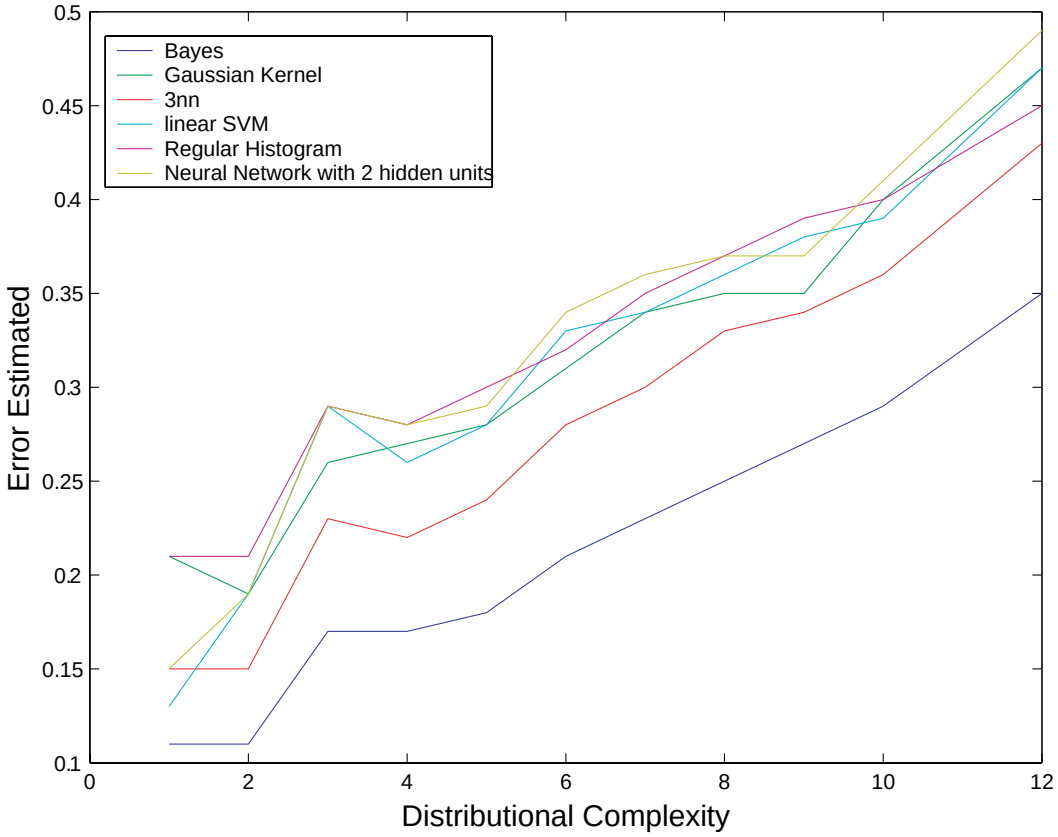


Fig. 7. Error estimate vs complexity N=40, sigma=1/6

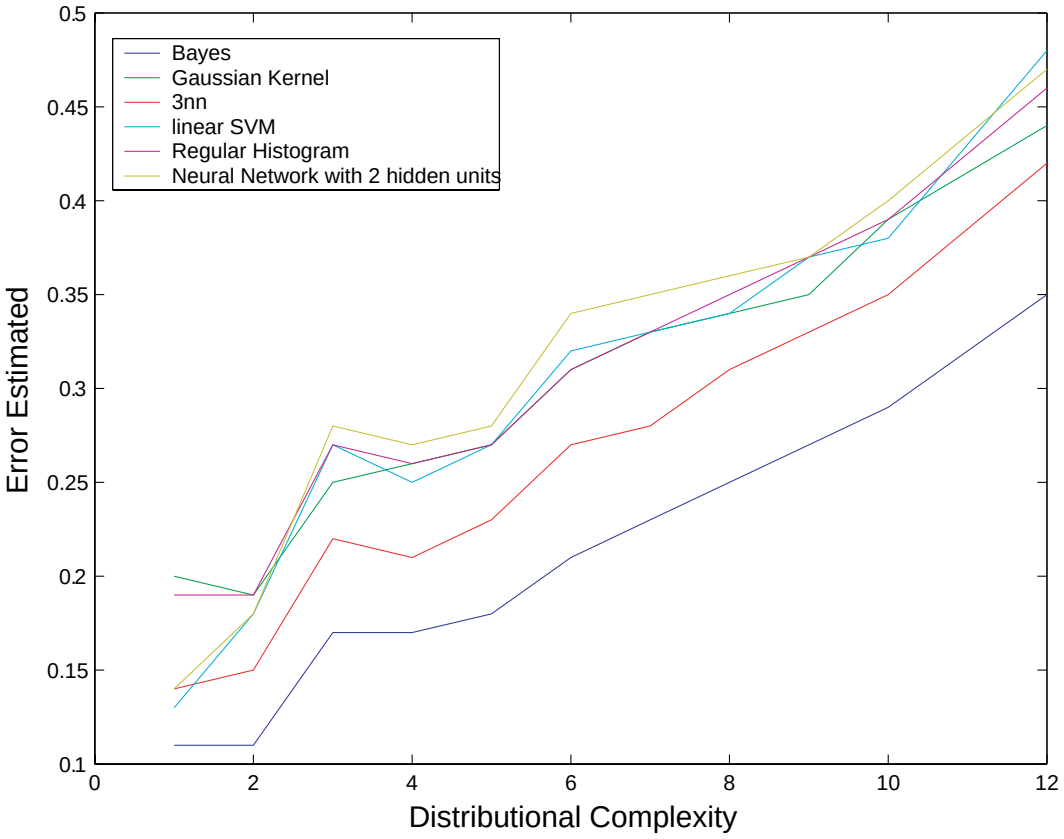


Fig. 8. Error estimate vs complexity N=60, sigma=1/6

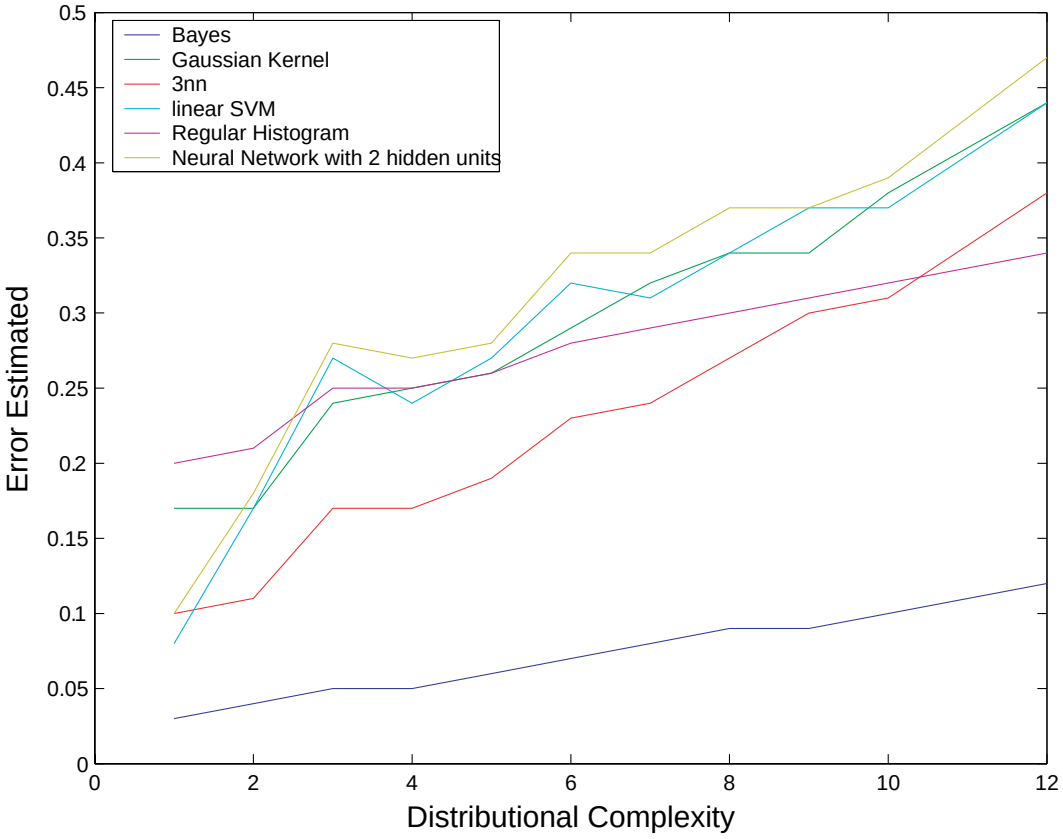


Fig. 9. Error estimate vs complexity N=20, sigma=1/10

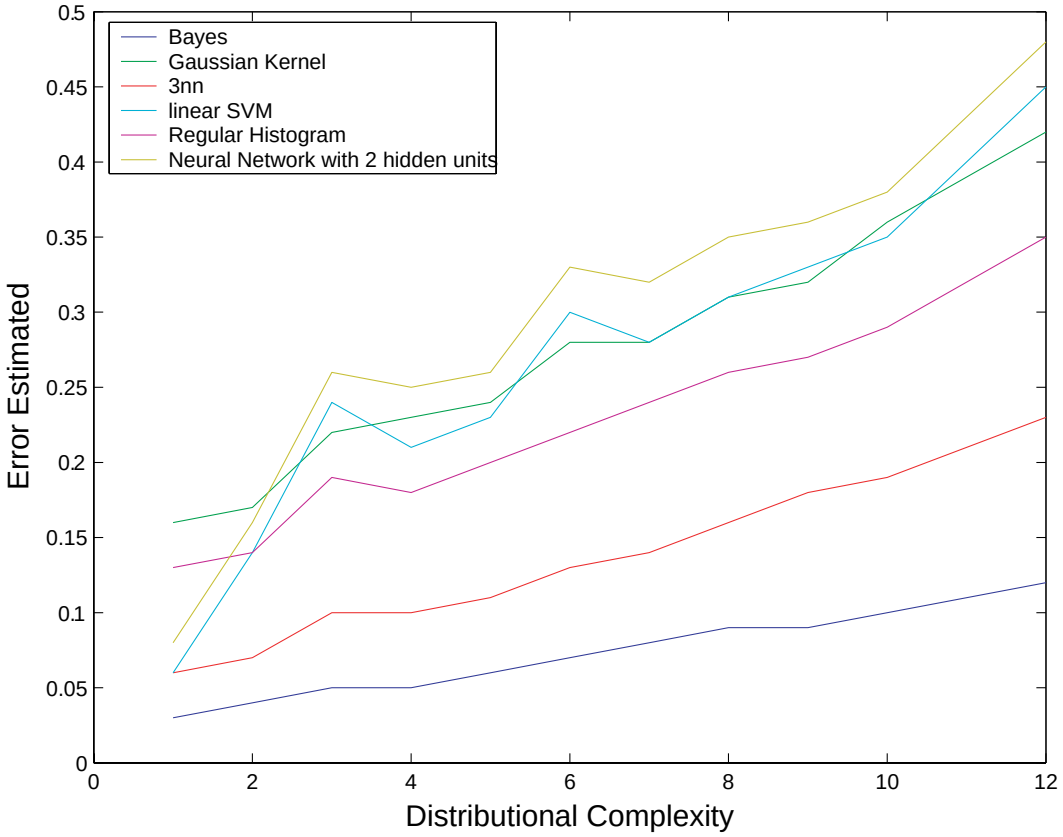


Fig. 10. Error estimate vs complexity N=40, sigma=1/10

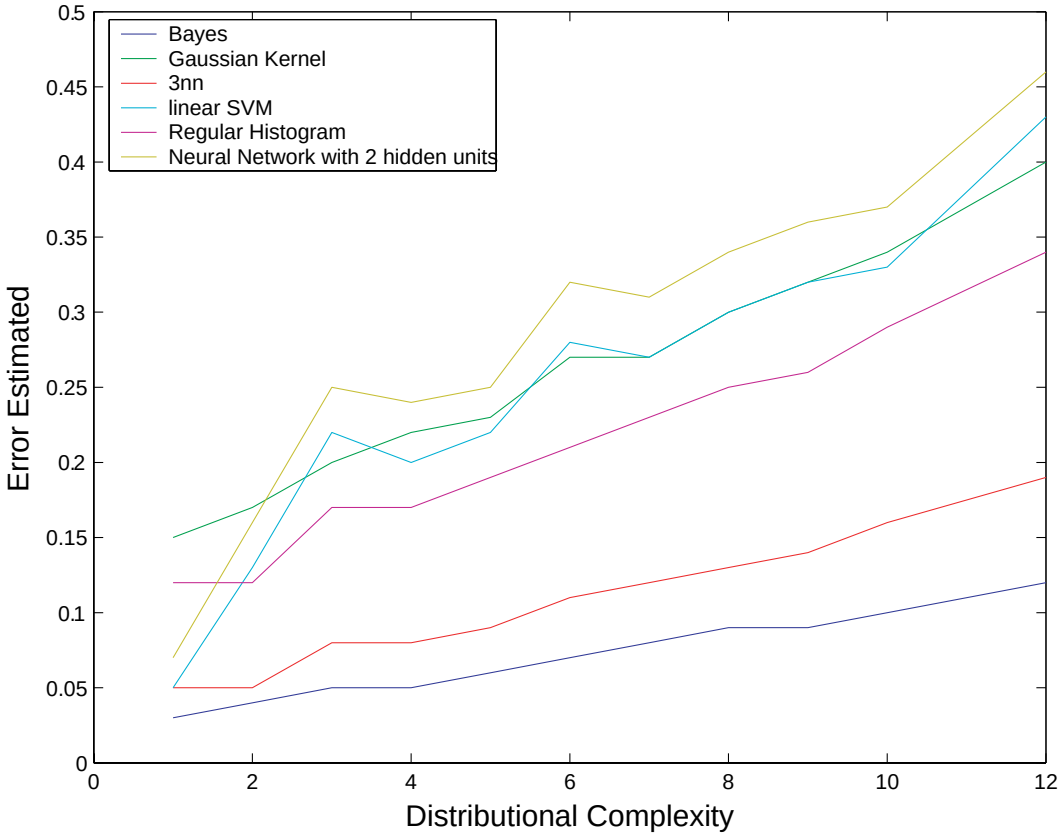


Fig. 11. Error estimate vs complexity N=60, sigma=1/10

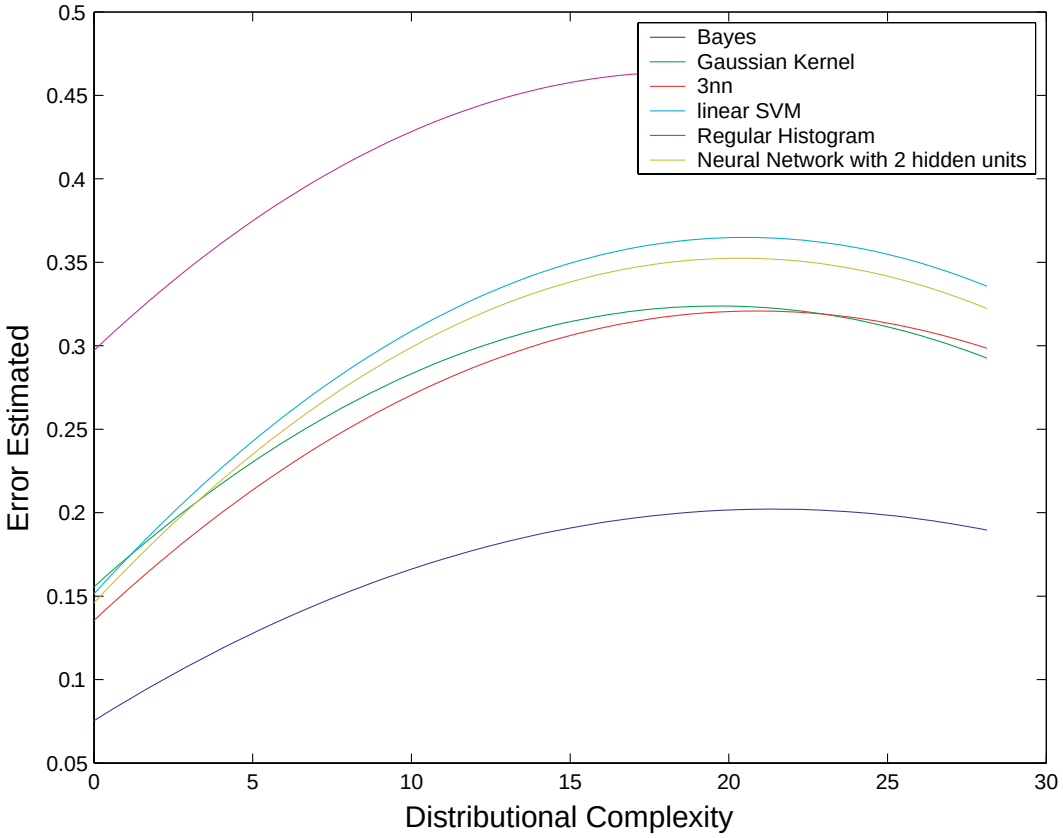


Fig. 12. Error estimate vs complexity N=20, sigma=1/6 (regressed)

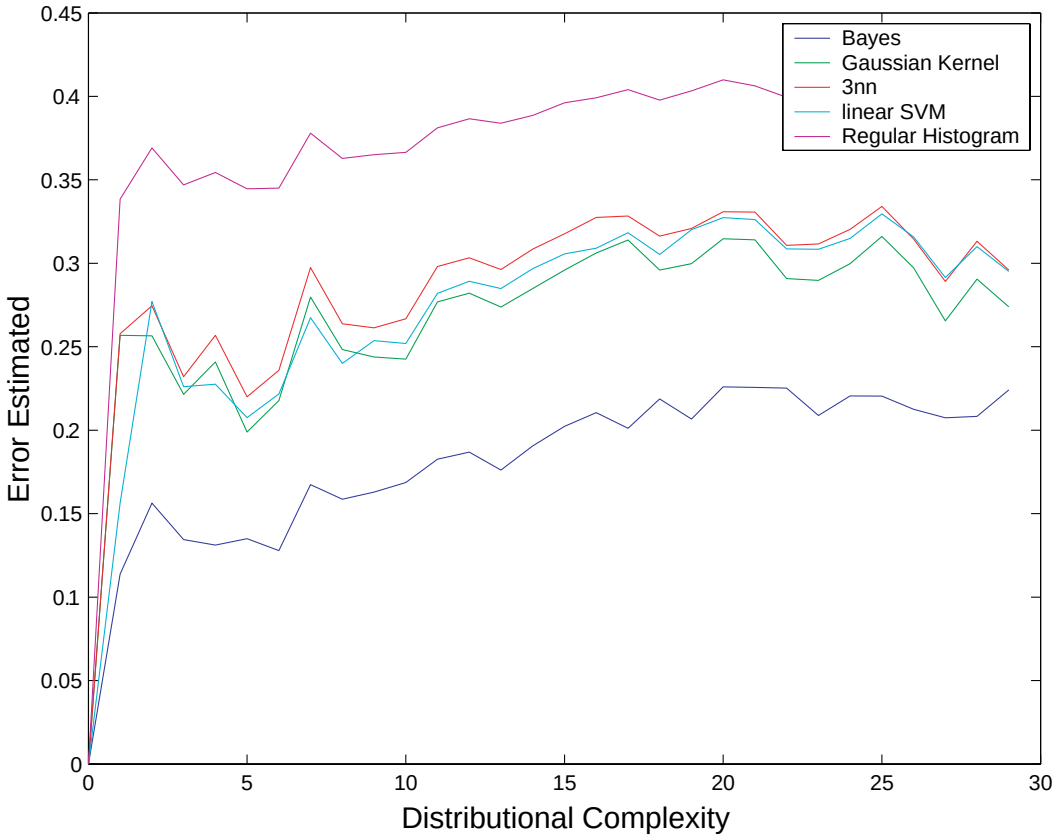


Fig. 13. Error estimate vs complexity N=40, sigma=1/6

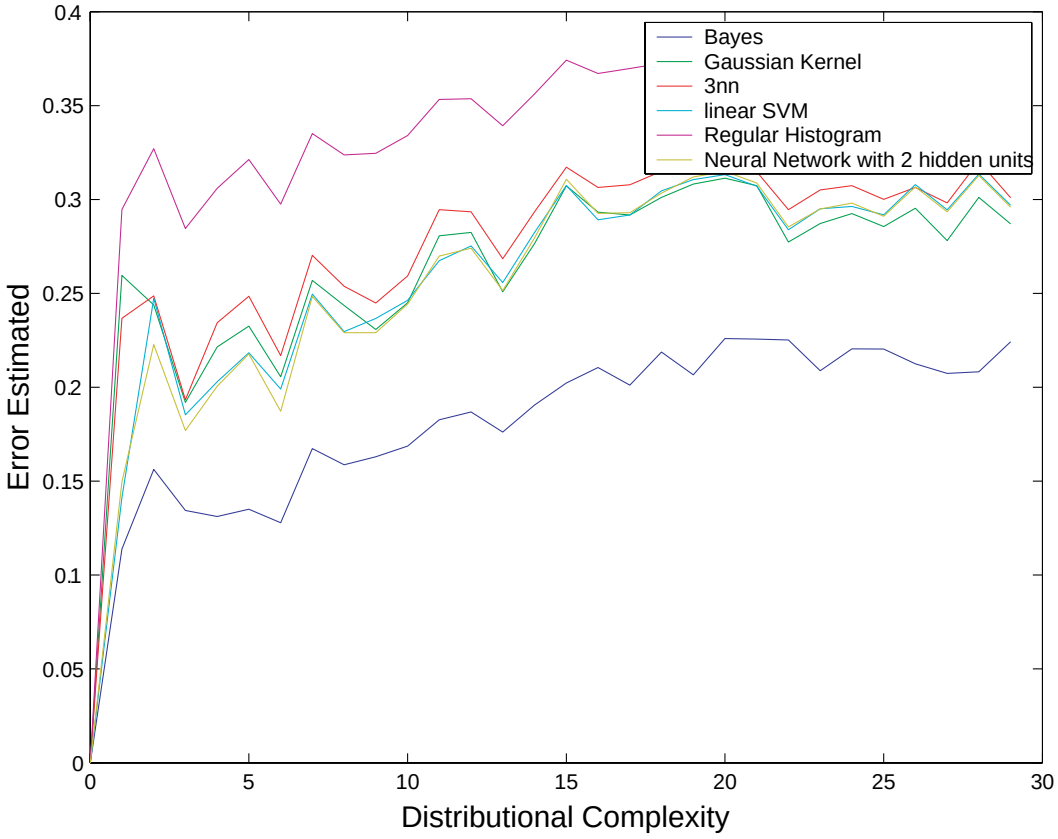


Fig. 14. Error estimate vs complexity N=60, sigma=1/6

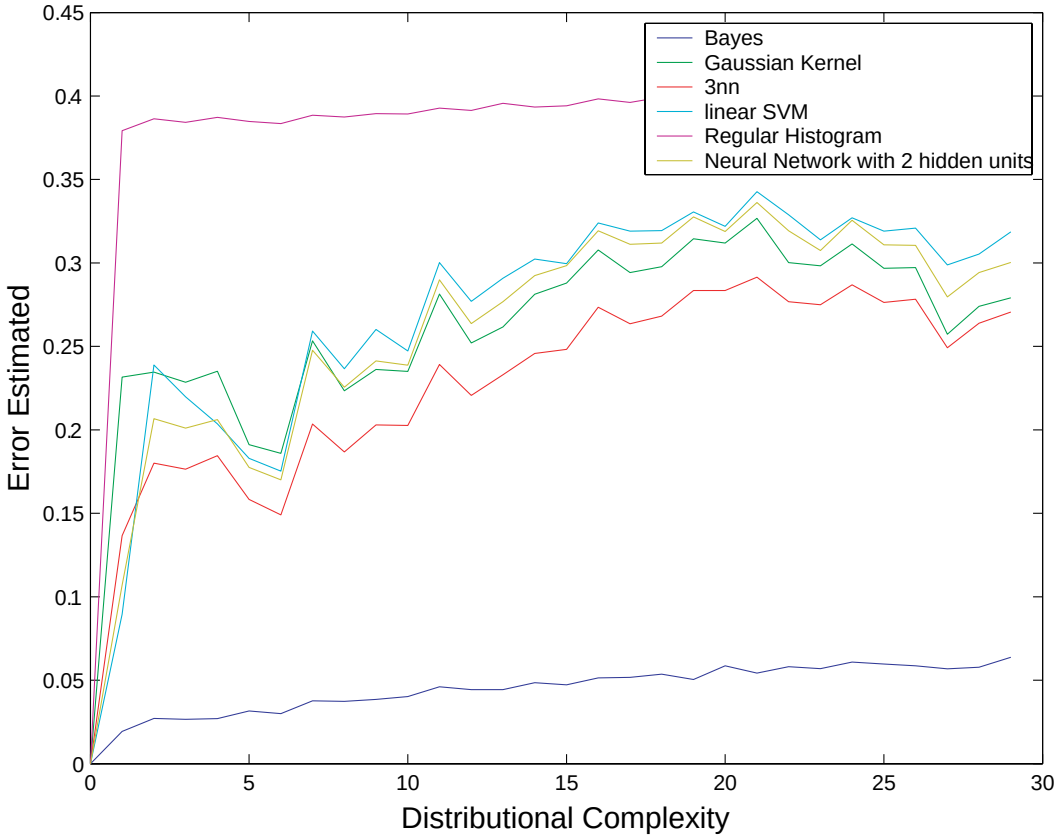


Fig. 15. Error estimate vs complexity N=20, sigma=1/12

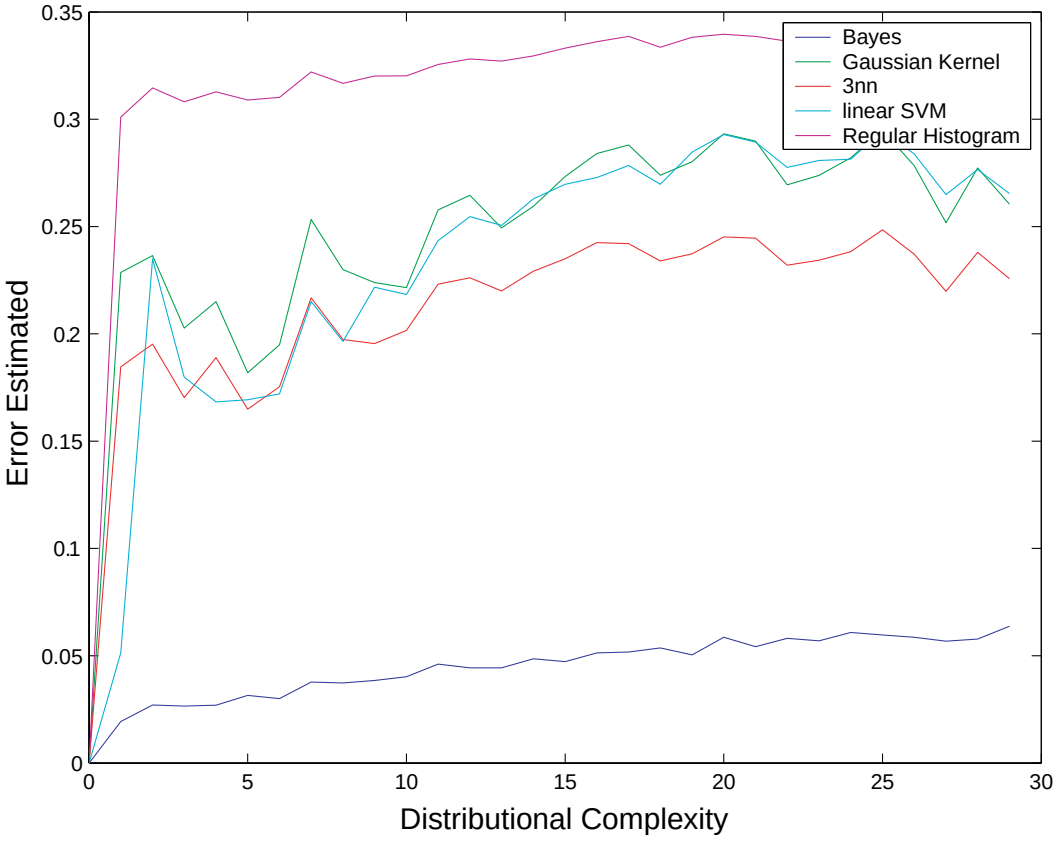


Fig. 16. Error estimate vs complexity N=40, sigma=1/12

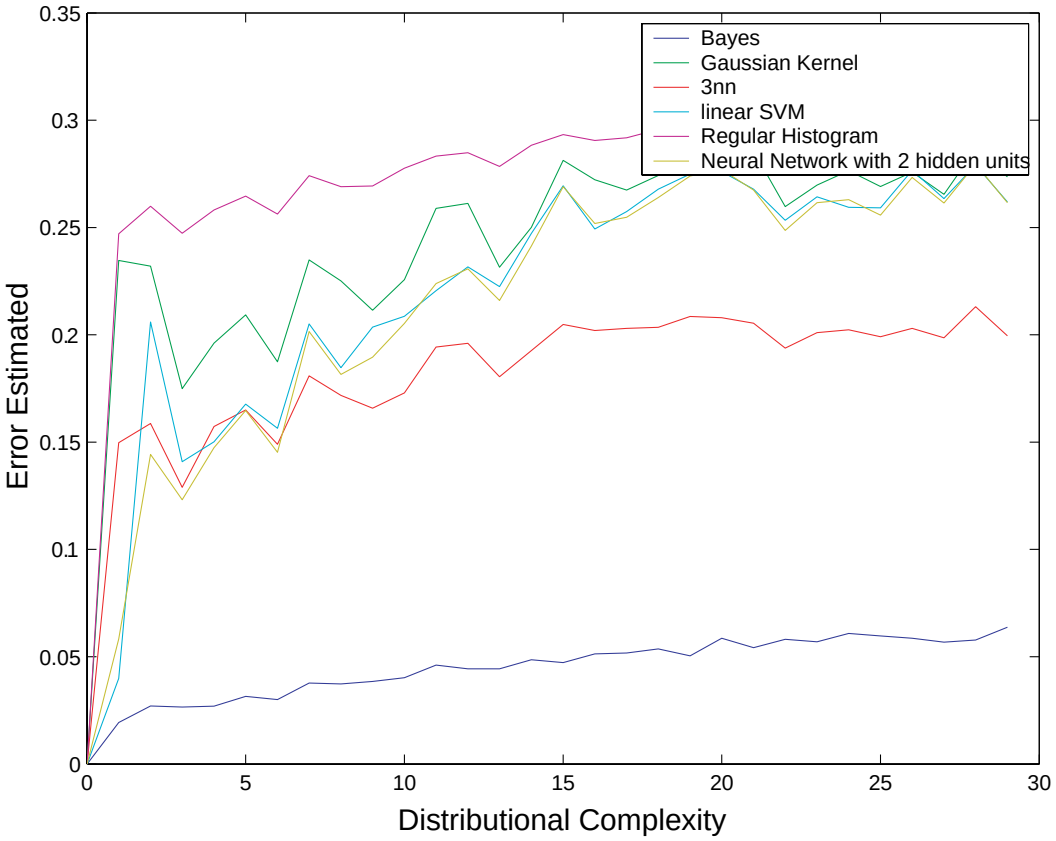


Fig. 17. Error estimate vs complexity N=60, sigma=1/12

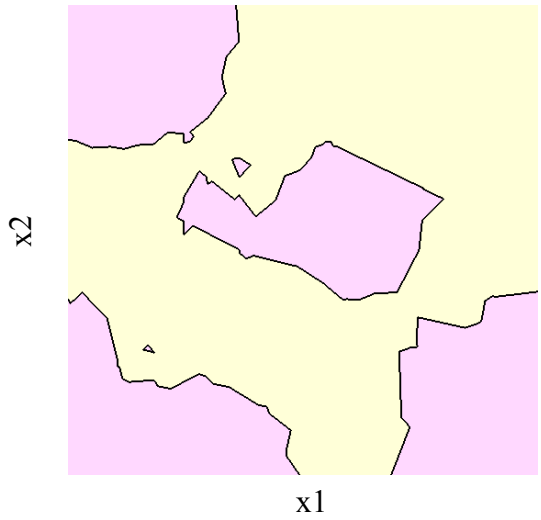


Fig. 18. Decision boundary plot for kNN

G. Classifier Decision Boundaries and Complexity

The classifier decision boundaries for some of the classifiers for a configuration with complexity 10 (2-dimensional case) are shown in figures 18 and 19. The actual data distribution of complexity 10 is shown in Figure 20. 18 shows how a 3NN classifier splits up the space. The decision boundary can be seen to relatively more complex than for the neural network or the gaussian kernel. However, the kNN does a very good job in approximating the partition. The neural network, on the other hand, approximates the partition very badly. All the classifiers were designed on a sample size of 60.

H. Summary

It has been seen that the kNN(k=3) outperforms almost all classifiers for all complexities except in the linear case. This justifies the use of 3NN classifiers for the small

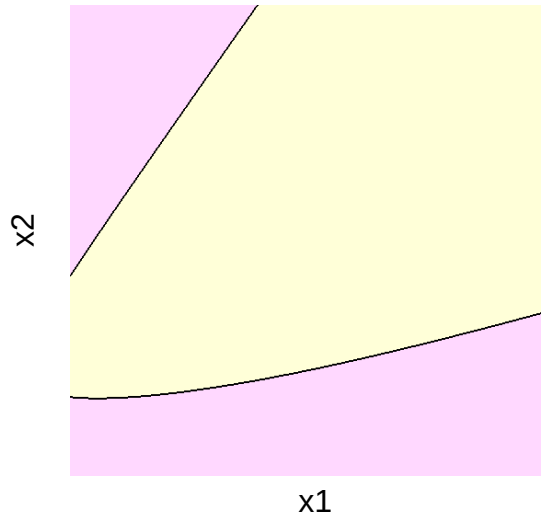


Fig. 19. Decision boundary plot for neural network with 3 hidden units

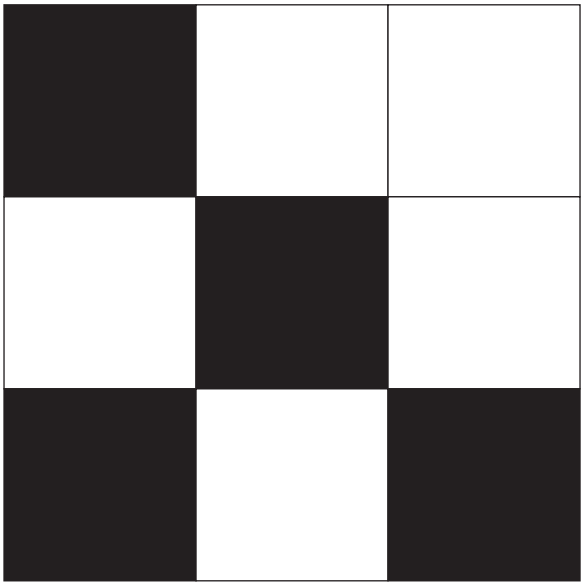


Fig. 20. Actual configuration of complexity 10

sample case. Also, in practical problems, it has been found that 3NN performs better than most other classifiers but there has been no study to confirm this. The above simulations confirm the choice of 3NN for small samples.

CHAPTER III

CLASSIFIER SELECTION AND VC DIMENSION: CONCLUSION

A. Introduction

As outlined in chapter 1, one of the methods of classifier selection is based on VC dimension. More specifically, it involves computing the VC confidence component of the upper bound on the expected error and selecting a classifier that minimises this VC confidence. This upper bound, which holds with a probability $1 - \eta$, is given as [8]:

$$\epsilon_{n,C} \leq \epsilon_{n,C,\alpha} + \sqrt[2]{\frac{(h(log(2n/h) + 1) - log(\eta/4))}{n}} \tag{3.1}$$

The difficulty with such a method is the estimation of the VC dimension.

B. Expected Error Estimate and VC Dimension

Instead of setting out to find the VC dimension of a learning machine, it might be easier to find the VC dimension at which a learning machine would perform the best. It is well-known that the VC dimension of a linear combination of l fixed basis functions is $l + 1$ [12]. Thus, it is possible to find a learning machine defined as linear combination of basis functions, of a certain VC dimension h^*. Based on this idea, it is possible to estimate the expected risk for various VC dimensions and find the VC dimension at which the estimated expected risk is minimum. This VC dimension would then be optimal for the sample size considered.

Consider the simple 1-dimensional case. Consider the set of basis functions $\mathbf{x} = (1, x, x^2, x^3, \ldots, x^{(H-1)})$, where x is a one-dimensional variable. The VC dimension of a linear combination of the basis functions is H; the linear combination is in fact, a polynomial classifier. The polynomial classifier will be used to classify a set of

training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$.

The training set is generated as follows. For each realization of the random variable x, the corresponding label is found by mapping the 1-dimensional data into a K-dimensional space and constructing a hyperplane in the K-dimensional space. The labels are assigned depending on which side of the hyperplane the transformed K-dimensional vector falls onto. Thus, a training set (x_i, y_i) of size n is obtained. Different realizations of the training set are obtained by using a random hyperplane generated for each set.

Using the fact that a polynomial classifier of degree $k-1$ has a VC dimension of k [12], polynomial classifiers, each of degrees $1, 2, \ldots, H-1, \ldots, H+40$ (H+40 is not a fixed choice) are designed, on the training set. A test set of size at least $5n$ according to the same rule used in generating the training set, is generated. The design and test process is repeated for different training data sets and the error estimates are averaged.

The results for sample sizes of 20, 40 and 60 are shown in Figures 21, 22 and 23 respectively. The results were seen to be almost independent of the choice of K. As can be expected, the estimated expected error or the test error decreases with increasing VC dimension at first and then increases, while the training error goes to zero as the VC dimension increases. It may be inferred that classifiers with VC dimesions of $5, 7$ and 10 would give better generalization results for sample sizes of 20, 40 and 60 respectively.

The results for the 2-dimensional case are shown in figures 21, 22 and 23. The basis functions for the 2-D case are of the form $x_1^i * x_2^j$ for values of i and j from 0 to M (M is any constant > 0).

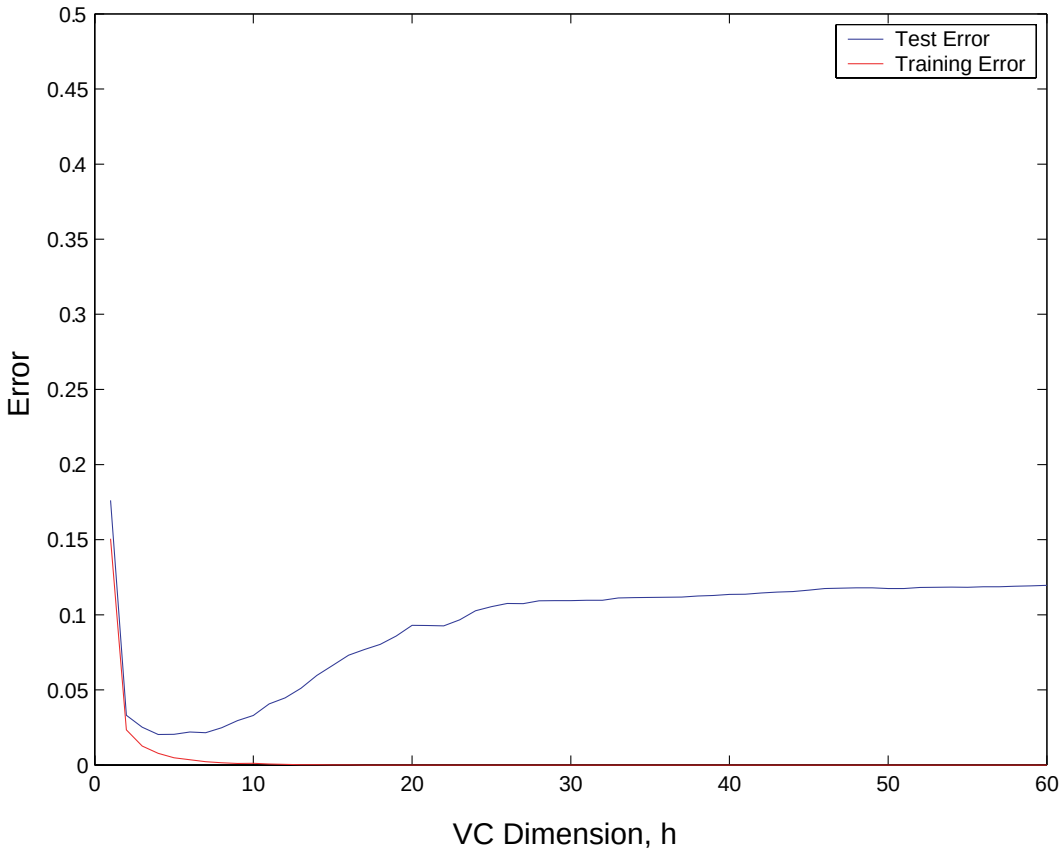


Fig. 21. Error estimate vs VC dimension N=20, d=1

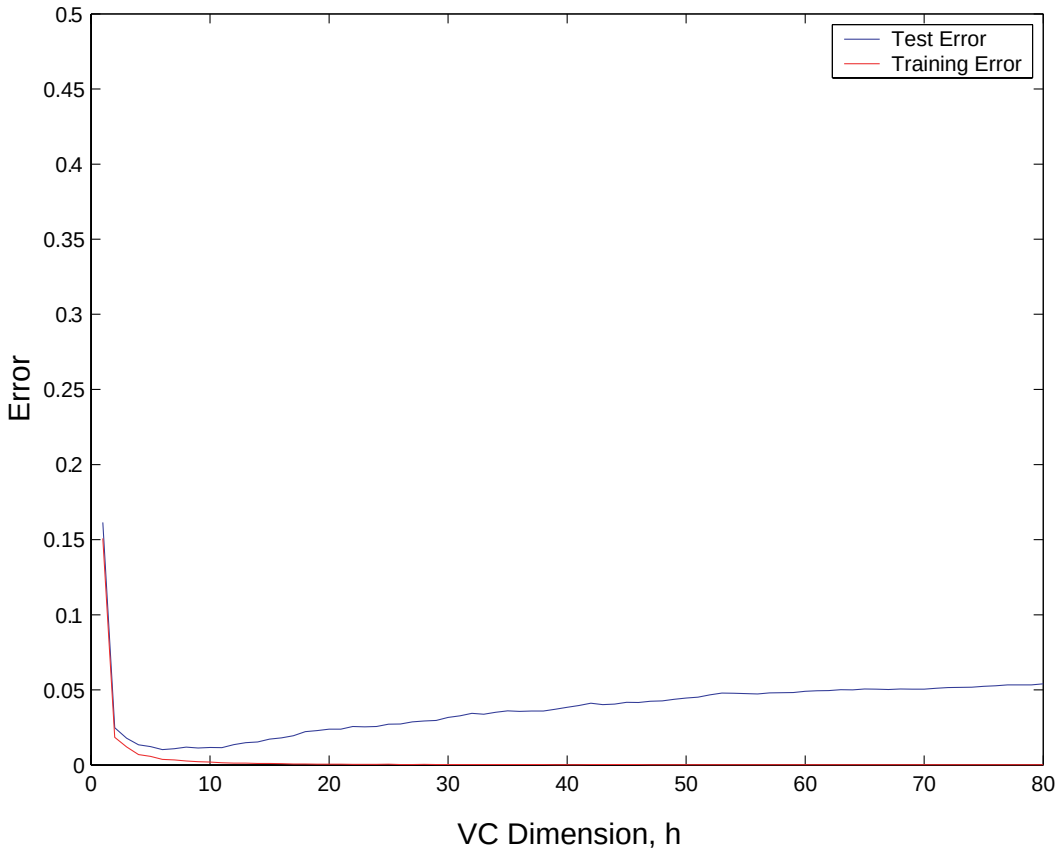


Fig. 22. Error estimate vs VC dimension N=40, d=1

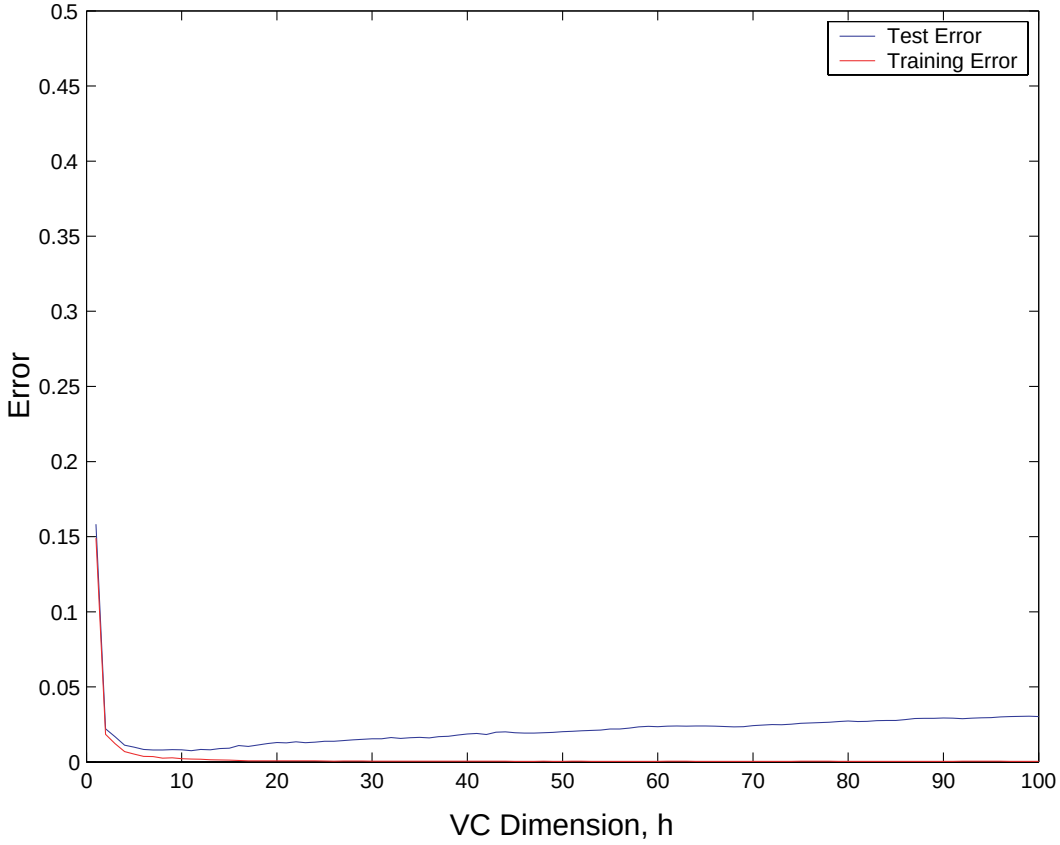


Fig. 23. Error estimate vs VC dimension N=60, d=1

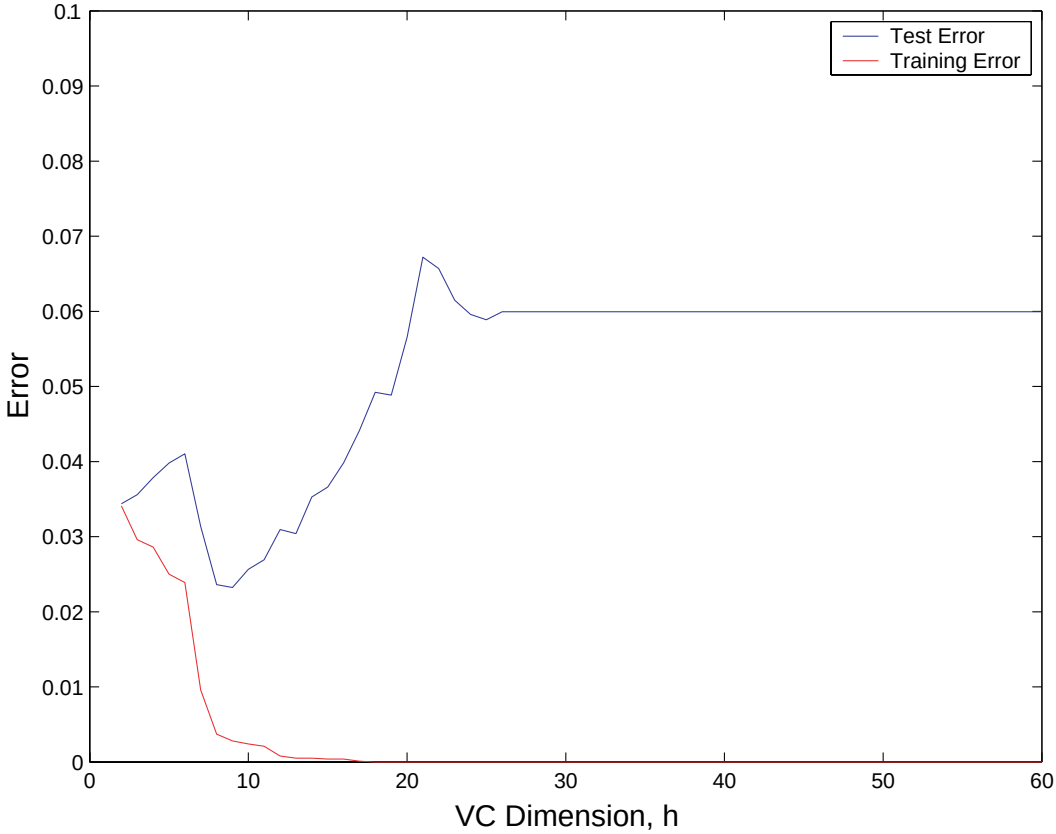


Fig. 24. Error estimate vs VC dimension N=20, d=2

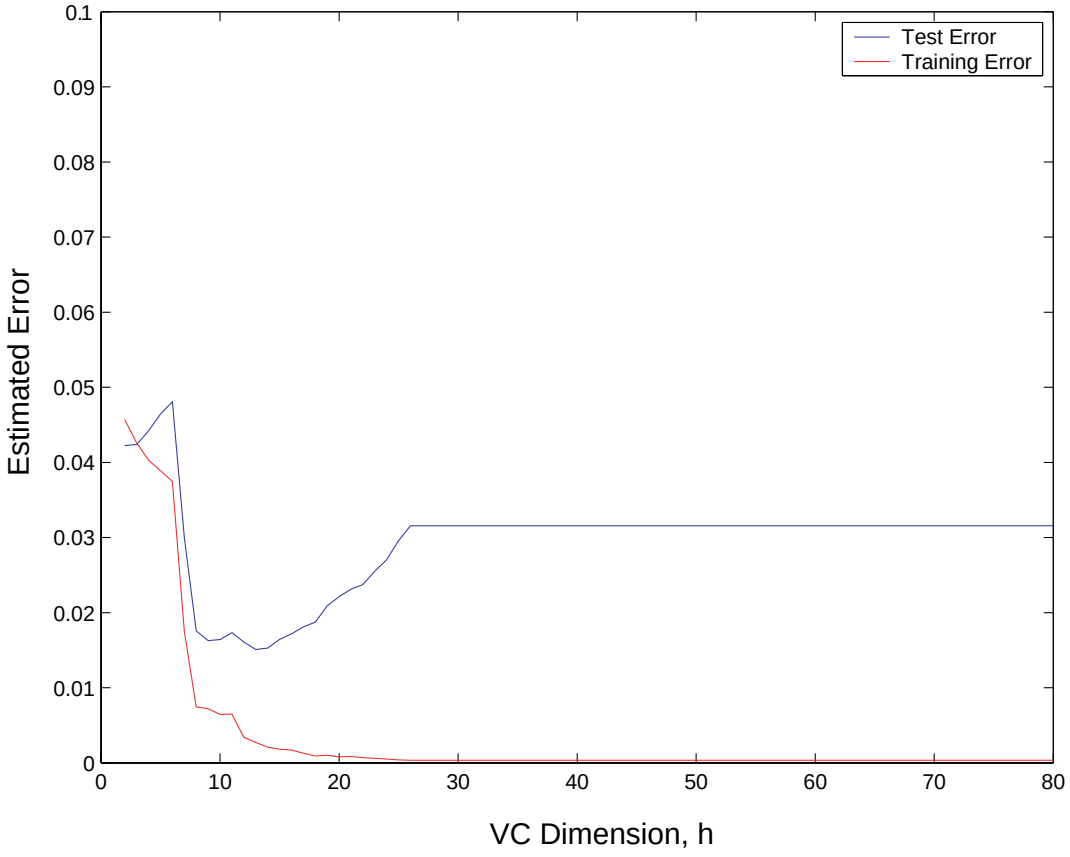


Fig. 25. Error estimate vs VC dimension N=40, d=2

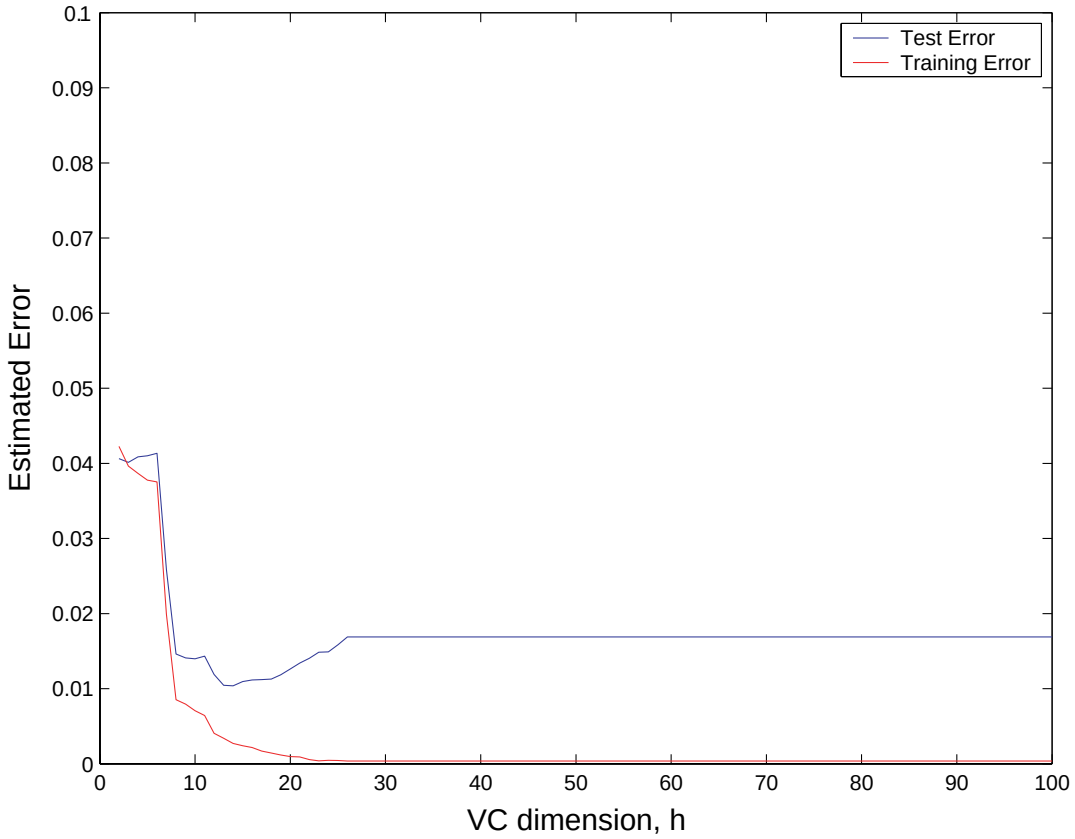


Fig. 26. Error estimate vs VC dimension N=60, d=2

C. Conclusion

It may be possible to estimate the optimal VC dimension for a given sample size. It has to be seen as to how a polynomial classifier (or any other classifier for the matter) for the estimated optimal VC dimension performs for the sample size considered. Without performing more experiments in this regard, it is difficult to conclude that an optimal VC dimension for a given sample size would work in practice.

REFERENCES

[1] L. Devroye, L. Gyorfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*. New York: Springer-Verlag, 1996.

[2] J. L. De Risi, V. R. Iyer, and P. O. Brown, "Exploring the metabolic and genetic control of gene expression on a genomic scale," *Science*, vol. 278, pp. 680-666, 1997.

[3] D. J. Duggan, M. L. Bittner, Y. Chen, P. S. Meltzer, and J. M. Trent, "Expression profiling using cDNA microarrays," *Nature Genetics*, vol. 21, pp. 10-14, 1999.

[4] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander, "Molecular classification of cancer: class discovery and class prediction by gene expression monitoring," *Science*, vol. 286, pp. 531-537, 1999.

[5] M. Schena, D. Shalon, R. W. Davis, and P. O. Brown, "Quantitative monitoring of gene expression patterns with a complementary DNA microarray," *Science*, vol. 270, pp. 467-470, 1995.

[6] I. Hedenfalk, D. Duggan, Y. Chen, M. Radmacher, M. Bittner, et. al., "Gene-Expression Profiles in Hereditary Breast Cancer," *New England Journal of Medicine*, vol. 344, pp. 539-548, 2001.

[7] E.R. Dougherty, "Small sample issues for microarray-based classification," *Comparative and Functional Genomics*, vol. 2, pp. 28-34, 2001.

[8] V. Vapnik, *Statistical Learning Theory*. New York: John Wiley, 1998.

[9] V. Vapnik, E. Levin, and Yann Le Cun, "Measuring the VC-Dimension of a Learning Machine," *Neural Computation*, vol. 6:5, pp. 851-876, 1994.

[10] Shao, V. Cherkassky, and W. Li, "Measuring the VC-dimension using optimized experimental design," *Neural Computation*, Cambridge, MA: MIT Press, vol. 12:8, pp. 1969-1986, 2000.

[11] 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.

[12] V. Cherkassky, and F. Mulier, *Learning from Data: Concepts, Theory and Methods*. New York: Wiley Interscience, 1998.

[13] E.R. Dougherty, and S.N. Attoor, "Design Issues and Commparison of Methods for Microarray Based Classification," in *Computational and Statistical Approaches to Genomics*, W. Zhang, and I. Shmulevich, Eds., Kluwer Publications, Boston, 2002.

[14] D. S. Eduardo, "VC dimension of neural networks," *Neural Networks and Machine Learning*, Berlin: Springer-Verlag, pp. 69-95, 1998.

VITA

SANJU NAIR ATTOOR

Address: 40/1, 'SREYAS', NRG Street, Coimbatore, India - 641038

Education: B.E., ECE, PSG College of Technology, India

M.S., Electrical Engineering, Texas A&M University

Publications:

1. S. N. Attoor, J. C. Liu, L. Rilett, and S. Gupta, " Non-Linear Analysis of Traffic Flow," *The 4th International IEEE conference on Intelligent Transportation Systems*, Oakland CA, Aug. 25-29, 2001, pp. 681-685.

2. M. Yeary, N. Kehtarnavaz, S. Attoor, M. Haji, and D. Horton, "A DSP-based implementation of a WCDMA reverse link design ," *Broadband Communications for the Internet Era Symposium digest, 2001 IEEE Emerging Technologies Symposium on*, 10-11 Sept. 2001, pp. 138-141.

3. G. Pok, S. Liu, and S. N. Attoor, "Selective Removal of Impulse Noise Based on Homogeneity Level Information," *IEEE Transactions on Image Processing*, March 2002.

4. E.R. Dougherty, and S.N. Attoor, "Design Issues and Commparison of Methods for Microarray Based Classification," in *Computational and Statistical Approaches to Genomics*, W. Zhang, and I. Shmulevich, Eds., Kluwer Publications, 2002.

5. T. Hsing, S. Attoor, and E. Dougherty, "Relation between permutation-test p values and classifier error estimates," *Machine Learning*, vol. 52:1, pp. 11-30, July 2003.

6. M. Leyk, D. V. Nguyen, S. N. Attoor, E. R. Dougherty, N. D. Turner, L. K. Bancroft, R. S. Chapkin, J. R. Lupton, and R. J. Carroll, "Summarizing FLARE Assay Images in Colon Carcinogenesis" (in preparation).

7. S. Attoor, E. R. Dougherty, Y. Chen, M. L. Bittner, and J. M. Trent, "Which is Better for cDNA-Microarray-Based Classification: Ratios or Direct Intensities" (in preparation).

8. S. Attoor, and E. R. Dougherty, "Classifier performance as a function of Distributional Complexity" (in preparation).