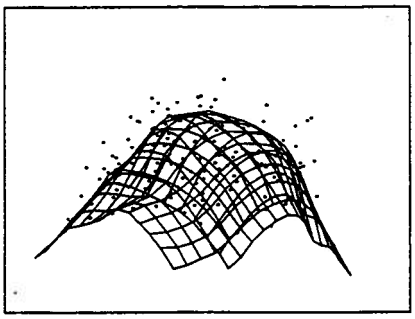
AN OVERVIEW OF PREDICTIVE LEARNING AND FUNCTION APPROXIMATION

Jerome H. Friedman

Technical Report No. 112
September 1994

Laboratory for Computational Statistics



Department of Statistics
Stanford University

AN OVERVIEW OF PREDICTIVE LEARNING AND FUNCTION APPROXIMATION

by
JEROME H. FRIEDMAN
STANFORD LINEAR ACCELERATOR CENTER
and
STANFORD UNIVERSITY

TECHNICAL REPORT NO. 112
SEPTEMBER 1994

DEPARTMENT OF STATISTICS
STANFORD UNIVERSITY
STANFORD, CALIFORNIA

An Overview of Predictive Learning and Function Approximation

JEROME H. FRIEDMAN
Department of Statistics
and
Stanford Linear Accelerator Center
Stanford University

Abstract: Predictive learning has been traditionally studied in applied mathematics (function approximation), statistics (nonparametric regression), and engineering (pattern recognition). Recently the fields of artificial intelligence (machine learning) and connectionism (neural networks) have emerged, increasing interest in this problem, both in terms of wider application and methodological advances. This paper reviews the underlying principles of many of the practical approaches developed in these fields, with the goal of placing them in a common perspective and providing a unifying overview.

1.0. The problem.

The predictive learning problem is remarkably simple to state, if difficult to solve in general. One has a system under study characterized by several (possibly many) simultaneously measurable (observable) quantities, called variables. The variables are divided into two groups. The variables in one group are referred to (respectively) as independent variables (applied mathematics), explanatory/predictor variables (statistics), or input variables (neural networks/machine learning). The variables of the other group also have different names depending on the field of study: dependent variables (applied mathematics), responses (statistics), or output variables (neural networks/machine learning). The goal is to develop a computational relationship between the inputs and the outputs (formula/algorithm) for determining/predicting/estimating values for the output variables given only the values of the input variables.

For example, the system under study might be a manufacturing process. The inputs would be the various parameters that control the process such as chemical concentrations, baking time, precision of various machine tools, etc. The outputs would be measures of quality of the final product(s). The goal here would be to forecast the resulting quality(ies) from knowledge of the input parameter values without having to actually run the process. Realizing this goal could eliminate much experimentation with considerable financial benefit. In another example the "system" might be people potentially sick with one of several diseases or maladies. The inputs would consist of the existence/severity of various symptoms and a collection of medical laboratory measurements. The output(s) would be indicators/severity of the potential diseases.

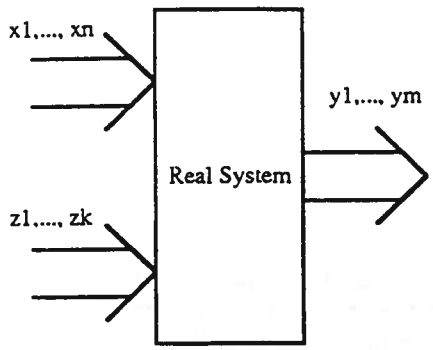


Figure 1. Diagram of the predictive learning problem.

Figure 1 displays a pictorial representation of the problem. The system is represented by the rectangular box, the inputs as lines on the left side of the box and outputs $\{y_1, \cdots, y_K\}$ as lines on the right. The inputs are grouped into two sets. The first $\{x_1, \cdots, x_n\}$ are the inputs whose values are actually measured or observed. The other set $\{z_1, \cdots, z_L\}$ represent other quantities that relate to (affect) the outputs but whose values are neither observed nor controlled. Sometimes this second set of input variables does not exist ($L = 0$). One might be able to identify and measure all possible input variables that relate to the outputs. Often, however, this is not the case. It is unlikely that all possible things that affect the quality of a manufactured product can be measured, such as operator mood, distractions, etc. In medical diagnosis all symptoms are seldom observed and all possible diagnostic tests are not usually made. This means that a simultaneous set of observed input values $\{x_1, \cdots, x_n\}$ does not uniquely specify the output values. There is some uncertainty in the outputs reflecting the lack of knowledge of the unobserved input values.

By drawing the input lines on the left and outputs on the right (Fig. 1) one is tempted to associate a causal relationship between the values of the inputs and outputs. That is, changing the input values causes a change in the values of the outputs. This is often the case, as in the manufacturing process example. Sometimes, however, the reverse is true. Often it is the existence/severity of a disease that causes the existence/severity of the symptoms. Other times neither is true; changes in values of the inputs and outputs are both reflections of changes in other (unobserved) factors. In many problems all three of these mechanisms exist. The point is that causality is not necessary for a derived relationship between the inputs and outputs

to be either accurate, or useful, as in medical diagnosis. The converse is also true; the existence of an accurate input/output relationship need not reflect causality.

1.1. Types of variables.

The input and output variables can each be of two different fundamental types: real or categorical. A real valued variable assumes values over a subset of the real line R^1. Examples are height, weight, voltage, course grade (F–A, mapped to 0–4). Values of these variables have an order relation and a distance defined between all pairs of values. Categorical variables have neither. For a categorical variable, two values are either equal or unequal. Examples are brand names, type of disease, nationality, etc. If a categorical variable assumes only two values then one of its values can be mapped to the real value zero and the other to real value one. It can then be treated as a real valued variable with no loss of generality. This is not the case if a categorical variable takes on more than two values.

When a categorical variable assumes more than two (say K) values it can be converted into K (0/1) real valued variables, one real variable for each categorical value. If the categorical variable assumes its kth value, its corresponding (kth) real valued surrogate is set to one and all of the other real surrogates, corresponding to different categorical values, are set to zero. If the categorical variable can assume only one of its values at a time then there is a linear degeneracy among the real variable surrogates and only $K-1$ of them are needed. This technique of mapping a categorical variable to real valued variables is referred to as "dummy variables" in statistics.

The dummy variable technique (trick) can always be used to deal with categorical variables. For categorical inputs it is not always the best way to do so. Methods differ on how well they can accommodate (extract information from) categorical inputs when they exist. (They don't occur for many problems.) Categorical outputs, however, occur often and represent an important class of problems referred to as pattern recognition in engineering, and classification/discriminant analysis in statistics. They are also the main focus of machine learning in artificial intelligence. Medical disease diagnosis is an example. For a categorical output variable the dummy variable trick is nearly always used. A single categorical output variable is converted into its real valued surrogates and the problem becomes a multiple (real-valued) output problem. This is discussed in more detail in Section 6.0. The point here is that it is sufficient to consider only real valued outputs in the supervised learning problem for the present. Translation of real-valued solutions back to the categorical output problem are deferred to Section 6.0.

1.2. Statistical model.

The underlying mathematical model associated with the problem (Fig. 1) is

$$y_k = g_k(x_1, \cdots, x_n, z_1, \cdots, z_L), \quad k = 1, K. \tag{1}$$

Here y_k is the kth output value and g_k is a (real) single valued deterministic function of all possible (observed and unobserved) inputs that relate to changing values of y_k. To reflect the uncertainty associated with the nonobserved inputs $\{z_1, \cdots, z_L\}$, this relation (1) is replaced by a statistical model

$$y_k = f_k(x_1, \cdots, x_n) + \varepsilon_k, \quad k = 1, K. \tag{2}$$

Here f_k is a (single-valued deterministic) function of the observed inputs $\{x_1, \cdots, x_n\}$ only, and an additional random (stochastic) component ε_k is added to reflect the fact that simultaneous specification of a set of (observed) input values $\{x_1, \cdots, x_n\}$ does not uniquely specify an output value (unless ε_k is a constant). A specific set of input values specifies a distribution of (random) y_k values, characterized by the distribution of the random variable ε_k.

The fundamental models (1) (2) are represented by separate relationships for each output whose generic form is given by suppressing the k index. This suggests that they can be treated as separate problems without regard to the commonality of their input variable sets. This can be done and sometimes it represents the best way to proceed. Strategies have been proposed, however, that attempt to exploit possible associations among the output values to improve accuracy. Discussion of these strategies and the conditions under which they may (or may not) yield improvement over treating each output independently is deferred to Section 5.0. As noted there, the issue is not yet settled and is still an open topic for research. Until then we will treat the multiple output problem as separate single output problems. In many (most) problems, interest usually focuses on a single output.

In the following we denote

$$x = \{x_1, \cdots, x_n\} \tag{3}$$

as a simultaneous set of input values. Our statistical model for each output (2) becomes

$$y = f(x) + \varepsilon, \tag{4}$$

with $f(x)$ being a single valued deterministic function of an n-dimensional argument, and ε a random variable which is presumed to follow some probabilistic law (probability distribution), $\varepsilon \sim F_\varepsilon(\varepsilon)$. That is, $F_\varepsilon(\varepsilon')$ gives the probability of observing a value of $\varepsilon \leq \varepsilon'$. Using statistical notation we will denote with the symbol $E[\cdot]$ the average of a quantity over this distribution, i.e.

$$E_\varepsilon[h] = \int_{-\infty}^{\infty} h(\varepsilon') dF_\varepsilon(\varepsilon').$$

The model (2) is ambiguous since one can add a constant value to ε and subtract the same (constant) value from $f(x)$ and leave all values of the

output y unchanged. This (definitional) ambiguity is usually resolved by the additional definition

$$E(\epsilon) = 0 \tag{5}$$

for all values of $\boldsymbol{x}$. In this case one has

$$f(\boldsymbol{x}) = E_{\epsilon}[y \mid \boldsymbol{x}]. \tag{6}$$

That is $f(\boldsymbol{x})$ is defined as the (unique) average output value y, for the specified set of input values $\boldsymbol{x}$.

1.3. Supervised learning.

The goal of any learning approach is to obtain a useful approximation $\hat{f}(\boldsymbol{x})$, to the true ("target") function $f(\boldsymbol{x})$ (4) (6) that underlies the predictive relationship between the inputs and output. There are many ways to do this. One could try to derive the fundamental equations corresponding to the physical or chemical laws that control the system. Experts could be consulted as to their knowledge of the system. Supervised learning attempts to learn the input/output relationship by example through a "teacher". This process is depicted in Figure 2. Here one observes the system under study, simultaneously measuring both the (observed) inputs and the corresponding output values. This is done repeatedly, collecting a "training" sample of N simultaneous sets of input/output values

$$\{y_i, x_{i1}, \cdots, x_{in}\}_1^N = \{y_i, \boldsymbol{x}_i\}_1^N. \tag{7}$$

The (observed) input values to the system are also input to an artificial system (learning algorithm – usually a computer program) that also produces outputs $\{\hat{f}(\boldsymbol{x}_i)\}_1^N$ in response to the inputs $\{\boldsymbol{x}_i\}_1^N$. The artificial system has the property that it can modify (under constraints) the input/output relationship $\hat{f}(\boldsymbol{x})$ that it produces in response to differences $\{y_i - \hat{f}(\boldsymbol{x}_i)\}_1^N$ (errors) between the artificial and real system outputs, as provided by a "teacher". This modification process is called learning (by example). Upon completion of the learning process the hope is that the artificial and real outputs will be close enough to be useful for all sets of (simultaneous) values of inputs likely to be encountered.

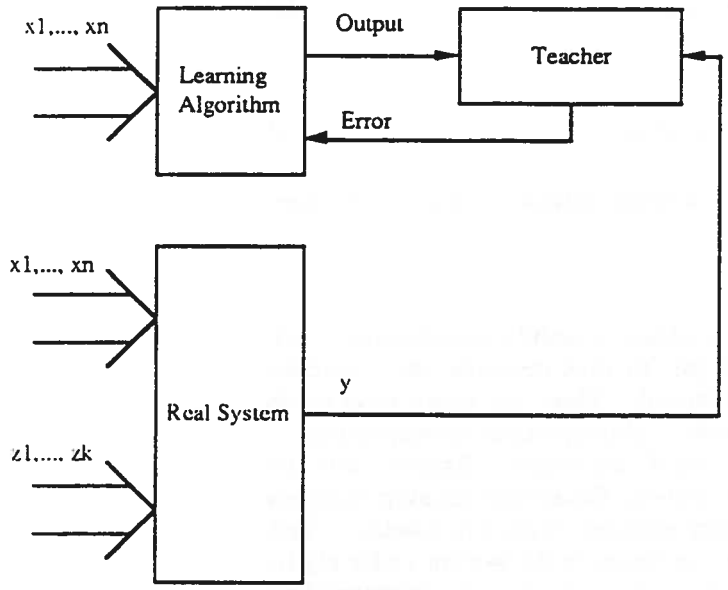


Figure 2. Diagram of supervised learning approach.

1.4. Function approximation/estimation.

The learning paradigm of the previous section has been the motivation for research into the supervised learning problem in the fields of machine learning (analogies to human reasoning) and neural networks (biological analogies to the brain). The approach taken in applied mathematics and statistics has been from the perspective of function approximation/estimation. In this view a simultaneous set of input values (3) is regarded as a point in an n-dimensional Euclidean space ($\boldsymbol{x} \in R^n$) and the target function (4) (6) as a function defined on that space. It is thus a surface (manifold) in the $(n+1)$-dimensional joint input/output space. The training sample (7) represents a point cloud in R^{n+1} related to the surface $[\boldsymbol{x}, f(\boldsymbol{x})] \in R^{n+1}$ by $\{\boldsymbol{x}_i, f(\boldsymbol{x}_i) + \varepsilon_i\}_1^N$ (4). The goal is to obtain a useful approximation to $f(\boldsymbol{x})$ for all $\boldsymbol{x}$ in some region of R^n, given its value (possibly contaminated with noise, $\varepsilon \neq 0$) only at the set of points represented by the training sample. Although somewhat less glamorous than the learning paradigm, treating supervised learning from the point of view of function approximation allows the geometrical concepts of Euclidean spaces and mathematical concepts of probabilistic inference to be applied to the problem. This will be the approach

taken here.

Function approximation is often divided into two subjects depending on the existence of an error term ε (4). If it is everywhere equal to zero the problem is referred to as interpolation. In this case there are no unobserved inputs (1), $\{z_i \cdots\} = \text{null}$, and the observed inputs $\boldsymbol{x} = \{x_1 \cdots x_n\}$ are the only ones that relate to changing output y values. In this case specifying a simultaneous set of inputs $\boldsymbol{x}$, uniquely specifies an output value. The interpolations problem is then to approximate true target function $f(\boldsymbol{x})$ everywhere within a region of the input space, given only its value at a finite number of points within the region (training sample). This is the problem treated in applied mathematics (multivariable function approximation).

When unobserved inputs do exist, the error term (4) is not generally zero and the output y becomes a random variable. Specifying a set of (observed) input values $\boldsymbol{x}$, specifies a distribution of output y-values whose mean is the target function $f(\boldsymbol{x})$ (6). This is the problem usually studied in the statistical literature and is referred to as nonparametric (flexible) multivariate regression. One goal is the same as in the interpolation problem; approximate the target function everywhere within a region of the input space given a training sample. The difference is that here the problem is more difficult since the target function is only approximately known at the training (input) points owing to the error term (4). Another (less ambitious) goal often addressed in the statistical literature is to estimate the output values only at the training sample points, given error contaminated values at those points. This is theoretically more tractable but seldom useful in practice.

There are two distinct practical reasons for application of supervised learning/function approximation: prediction and interpretation. In prediction it is expected that in the future new observations will be encountered for which only the input values are known, and the goal is to predict (estimate) a likely output value for each such case. The function estimate $\hat{f}(\boldsymbol{x})$ obtained from the training data through the learning algorithm is to be used for this purpose. In this case the primary goal is accuracy; one would like the estimates $\hat{f}$ to be close to the (unobserved) output y over this future prediction sample. Let $\Delta(y, \hat{f})$ be some measure of distance (error) between the two quantities. Common examples are

$$\Delta[y, \hat{f}] = |y - \hat{f}|, \quad \text{or} \tag{8a}$$

$$\Delta[y, \hat{f}] = (y - \hat{f})^2, \tag{8b}$$

the later being the most popular because its minimization leads to the simplest algorithms. A reasonable measure of (lack of) performance would then be the global prediction error

$$\Delta_P = \frac{1}{N_P} \sum_{i=1}^{N_P} \Delta[y_i, \hat{f}(\boldsymbol{x}_i)]$$

where the sum is over the N_P (future) prediction samples. Generally the precise locations of the prediction sample are not known in advance but can be assumed to be a random sample from some probability density $p(\boldsymbol{x})$, whose values give the relative probabilities of encountering a new observation to be predicted at $\boldsymbol{x}$. The global error then becomes

$$\Delta_P = \int E_\varepsilon \Delta[f(\boldsymbol{x}) + \varepsilon, \hat{f}(\boldsymbol{x})] p(\boldsymbol{x}) d\boldsymbol{x}$$

where ε is the noise term (4) and E_ε is the average over its distribution $F_\varepsilon(\varepsilon)$. In particular if (8b) is chosen, then the mean-squared prediction error is given by

$$\begin{aligned} mspe &= \int E_\varepsilon [f(\boldsymbol{x}) + \varepsilon - \hat{f}(\boldsymbol{x})]^2 p(\boldsymbol{x}) d\boldsymbol{x} \\ &= \int [f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})]^2 p(\boldsymbol{x}) d\boldsymbol{x} + \int \sigma^2(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x} \end{aligned} \tag{9}$$

where $\sigma^2(\boldsymbol{x}) = E_\varepsilon(\varepsilon^2|\boldsymbol{x})$ is the variance of the noise at $\boldsymbol{x}$. The first term (9) is known simply as the mean-squared-error

$$mse = \int [f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})]^2 p(\boldsymbol{x}) d\boldsymbol{x} \tag{10}$$

and it completely captures the dependence of the prediction error on $\hat{f}(\boldsymbol{x})$. Minimizinig (9) and (10) with respect to $\hat{f}(\boldsymbol{x})$ gives the same result. In particular, these results show that the target function $f(\boldsymbol{x})$ (4) (6) is the best (mean-squared-error) predictor of future outputs and accurate approximation and future prediction are equivalent goals.

Another reason for applying supervised learning is interpretation. The goal here is to use the structural form of the approximating function $\hat{f}(\boldsymbol{x})$ to try to gain insight and understanding concerning the mechanism that produced the data. In this context no future data is necessarily envisioned. The approximation is intended to serve primarily as a descriptive statistic for illuminating the properties of the input/output relationship. Some properties of interest might include identification of those input variables that are most relevant to (associated with) the variation in the output, the nature of the dependence of the output on the most relevant inputs, and how that changes with changing values of still other input values. Such information can be quite useful for understanding how the system works and perhaps how to improve it. Also, nearly all scientific research is based on understanding from data rather than (the narrow goal of) future prediction.

Accuracy (10) has some importance to this application since its not very useful to interpret an approximation that bears little resemblance to the true input/output relationship $f(\boldsymbol{x})$. However it is not the only goal. The informal criterion (to be optimized) is the amount of (correct) information learned about the system. Human engineering considerations are involved

in determining the best ways to summarize and present the approximating equation/algorithm in the best form (graphical and tabular) for human interpretation. As will be seen below, the major successful methods for supervised learning differ greatly in the extent to which they can be adapted to this interpretational goal. These differences tend to be much larger than their differences in prediction accuracy, and only a few provide highly interpretable output.

2.0 Difficulty of the problem.

Although the problem addressed by supervised learning is straight forward to state it has (so far) been difficult to provide a completely general (useful) solution. This difficulty, coupled with the broad range of important applications, has motivated much research activity in this area producing a large number of diverse learning algorithms. In this section we address reasons for the difficulty and in following sections many of the proposed solutions are presented, compared and contrasted.

All the disparate difficulties stem from one common origin – the finite size of the training sample. If the training sample were infinite in size (and one had an infinitely fast computer) one could directly compute the target function from (6) at any point $\boldsymbol{x}$ (or more precisely its average value over an infinitesimal volume centered at $\boldsymbol{x}$, in the limit that the volume approaches zero). The problem is well posed and the solution (6) unique. For example, (6) uniquely minimizes (9) and (10).

In the case of a training sample (7) of size N, the solution to the analog of (9)

$$\hat{f}(\boldsymbol{x}) = \underset{g(\boldsymbol{x})}{\operatorname{argmin}} \sum_{i=1}^{N} [y_i - g(\boldsymbol{x}_i)]^2 \tag{11}$$

is not unique. There are a large (infinite) number of functions that can interpolate the data points yielding the minimum value (zero) for the criterion (11). In the case of no noise, $\varepsilon = 0$ everywhere (4), one of these functions will be the target $f(\boldsymbol{x})$. In the noisy case, $\varepsilon \neq 0$, none of them will be $f(\boldsymbol{x})$, or perhaps even close to it. Thus (11) represents an ill-posed problem. In both the noiseless and noisy cases solving (11) does not solve the problem for finite training samples.

In order to obtain useful results for finite N, one must restrict the eligible solutions to (11) to a smaller set then all possible functions. The nature of this restriction is decided by the user based on considerations outside the data. This is usually done by a choice of learning method. The restrictions imposed (explicitly or implicitly) by many popular learning algorithms is the topic of Sections 4.0–4.4 It is clear, however, that this restrictive approach does not really remove the ambiguity caused by the multiplicity of possible solutions. Imposing a particular restriction may result in a unique solution to (11), but there are an infinite number of possible restrictions, each of which

can lead to a different (unique) solution to (11); the multiplicity has simply been transferred to a different mechanism (constraint choice).

All learning methods impose some type of smoothness constraint on solutions to (11). This does not completely remove the ambiguity since there are a wide variety of possible ways to define and impose smoothness. However all (reasonable) definitions require some type of regular behavior in small neighborhoods of the input space. That is, for all input points $\boldsymbol{x}$ that lie close enough to each other, $\hat{f}(\boldsymbol{x})$ exhibits some special structure (e.g. nearly constant, linear, or close to a polynomial of low degree). The strength of the constraint is controlled by the size of the neighborhood; choosing a larger neighborhood imposes a stronger constraint. Imposing a stronger constraint in turn makes solutions to (11) more sensitive to the particular type of constraint chosen, thereby increasing the variability of solutions to differing smoothness definitions.

Minimal neighborhood size is dictated by the training sample size or, more precisely, by the density with which the input space is sampled. Clearly, if a particular neighborhood contains no training data then all information concerning the target function $f(\boldsymbol{x})$ in that neighborhood must be inferred from sample points outside the neighborhood through the smoothness constraint alone. The more densely the input space is sampled the smaller such regions become thereby placing less reliance on the constraint to approximate the target function. This in turn makes the approximation less sensitive to the particular type of constraint chosen. In the limit of infinite training sample size ($N \rightarrow \infty$) all regions of the input space are sampled with infinite density, the size of the constraint neighborhoods can become arbitrarily small (thereby effectively removing the constraints), and the solution to (11) is uniquely defined by (6), which is, in fact, the true target function.

2.1. Curse-of-Dimensionality.

The discussion of the previous section implies that there are two ways to obtain an accurate estimate of the target function $f(\boldsymbol{x})$ (4) (6). One way would be to place a very restrictive set of constraints on the approximation $\hat{f}(\boldsymbol{x})$ defining a small set (class) of eligible solutions to (11). This will be effective to the extent that the target function $f(\boldsymbol{x})$ is a member of this class or sufficiently close to one of its members. A good choice of constraints would in turn require knowledge (outside the data) concerning the properties of the target function. In absence of such knowledge one must appeal to the second alternative for obtaining a good approximation: a large enough training sample to densely pack the input space. Although this is often feasible for input variable spaces of low dimension (few input variables), it is not possible for high dimensional spaces, even with very large training samples. This fact is often referred to as the "curse-of-dimensionality" (Bellman, 1961).

There are many manifestations of this curse, all of which tend to render geometrical intuition gained from low dimensional settings inapplicable to

higher dimensional problems. As above, let n be the dimensionality of the input space ($x \in R^n$) and N be the training sample size. One problem is that sampling density is proportional to $N^{\frac{1}{n}}$. Thus, if $N_1 = 100$ represents a dense sample for a single input problem ($x \in R^1$), then $N_{10} = N_1^{10} = 10^{20}$ is the sample size required for the same sampling density with ten inputs ($x \in R^{10}$). Thus, in high dimensions all (feasible) training samples very sparsely populate the input space.

This sparse sampling property complicates geometric intuition in a number of ways. One is that in a high dimensional space interpoint distances between sample points are all large and approximately equal. This is because the volume of a sphere in an n-dimensional space is proportional to its radius to the nth power. Therefore neighborhoods that contain even only a few sample points have large radii. Consider a random set of training points sampled from a uniform distribution in the n-dimensional unit hypercube, $x \sim U^n[0,1]$. The average (expected) edge length of a hypercubical neighborhood containing a fraction p of the training points (volume) is $e_n(p) = p^{\frac{1}{n}}$. In ten dimensions, $e_{10}(.01) = 0.63$ and $e_{10}(0.1) = 0.80$. The edge length corresponding to the entire training sample is 1.0, so that to capture 1% or 10% respectively of the data points one must include over 63% or 80% (respectively) of the range of each input variable. Such neighborhoods are not very "local".

Another consequence of the sparsity of sampling in high dimensions is that all sample points are close to an edge of the sample. Again consider $x \sim U^n[0,1]$. In a sample of size N the expected L_∞ distance to any data points is

$$d(n,N) = \frac{1}{2}\left(\frac{1}{N}\right)^{\frac{1}{n}}.$$

for $n = 10$, $d(10,1000) \simeq 0.5$ and $d(10,10000) \simeq 0.4$. The maximum possible distance to any edge is 0.5 (the point located at the very center in each dimension). Thus, almost every point will be closer to an edge or boundary of the sample than to its closet sample point. Note that using L_∞ distance represents the most favorable choice since an L_∞ ball (hypercube) has the largest volume for a given radius. The situation would be much worse for say L_2 (Euclidean) distance. The reason that this represents a problem is that prediction (estimation) is much more difficult near the edges of the training sample. One must extrapolate (using the smoothness constraints) from neighboring sample points rather than interpolating between them.

This edge effect problem is not peculiar to uniform sampling in bounded (hypercubical) regions. Consider a training and prediction sample drawn from a (standard) normal distribution (unbounded), in the input space $x \sim N(0, I_n)$ where I_n is the n-dimensional identity matrix. The distance squared from the origin to any point follows a χ^2_n-distribution on n degrees of freedom. The expected distance is then $\sqrt{n - 1/2}$ and its standard deviation is $1/\sqrt{2}$. Consider a unit vector $a = x/|x|$ in the direction defined by a prediction

point $\boldsymbol{x}$ and the origin, and project $\boldsymbol{x}$ and the training data $\{\boldsymbol{x}_i\}_1^n$ onto this direction. That is the projected points are

$$z = \boldsymbol{a}^T\boldsymbol{x}, \quad \{z_i = \boldsymbol{a}^T\boldsymbol{x}_i\}_1^N.$$

From the χ_n^2 distribution the expected location of the prediction point in this projection is $\sqrt{n-1/2}$ with standard deviation of $1/\sqrt{2}$. However, since the training sample is unrelated to the direction $\boldsymbol{a} = \boldsymbol{x}/|\boldsymbol{x}|$, the projected training data points will follow a standard normal distribution $z_i \sim N(0,1)$. For $n = 10$, the expected location of a test point $\boldsymbol{x}$ is 3.1 standard deviations from the center of the training data (origin) in its own projection $\boldsymbol{a} = \boldsymbol{x}/|\boldsymbol{x}|$. For $n = 20$ it is 4.4 standard deviations away. Thus, (nearly) every prediction point sees itself lying at the edge of the training sample with all of the training points clumped to one side near the origin. This effect is clearly aggravated by increasing the dimension of the input space.

The above discussion indicates that the curse-of-dimensionality represents a formidable obstacle to accurate function approximation in high dimensions (many inputs). There are also arguments to the contrary; the curse-of-dimensionality is a fiction that does not really exist. One such argument is based on Kolmogorov's solution to Hilbert's 13th problem (Kolmogorov, 1957). Kolmogorov's theorem states than any continuous function of n-variables can be completely specified by a function of a single argument,

$$f(x_1, \cdots, x_n) = \sum_{j=1}^{2n+1} g_f \left[\sum_{i=1}^{n} \lambda_i \, \varphi_j(x_i) \right]. \tag{12}$$

The constants $\{\lambda_i\}_1^n$ are universal in the sense that their values do not depend on $f(\boldsymbol{x})$. The functions $\{\varphi_j(u)\}_1^{2n+1}$ are functions of a single argument and also universal in the same sense; they do not depend on $f(\boldsymbol{x})$. The single argument (continuous) function $g_f(z)$ does depend on $f(\boldsymbol{x})$ and thus completely specifies it through (12). Therefore, there exists a continuous function of one variable $g_f(z)$ that completely characterizes any continuous function $f(\boldsymbol{x})$ of n variables. From this one can draw the conclusion that, fundamentally, functions of more than one variable do not exist; expressing them in that way is simply a matter of choice of representation. Thus, the curse-of-dimensionality is also simply a matter of representation and not a fundamental aspect of the problem.

Although the premise of the preceding paragraph is clearly correct, the conclusion is not. The reason is that representing $f(\boldsymbol{x})$ by $g_f(z)$ does not change its complexity. Even very mild functions of n variables result in very "wild" corresponding functions $g_f(z)$ (12). Thus, the density of training sample points $\{z_i\} \in R^1$ must be much greater than $\{\boldsymbol{x}_i\} \in R^n$, leading to the requirement of comparable training sample sizes. Hilbert conjectured that a result like (12) was not true; "bad" (high dimensional) functions cannot be represented in a simple way by "good" (low dimensional) functions. His

intuition was correct. Quoting G. G. Lorentz (1986) "Kolmogorov's theorem shows only that the number of variables n is not a satisfactory characteristic of badness." The fundamental issue is the complexity of $f(\boldsymbol{x})$, and common functions of many variables tend (have greater opportunity) to be more complex that those of a lower dimensional argument.

Another argument against the fundamentality of the curse-of-dimensionality is that is can be overcome in many situations. Consider the case (most) often studied in statistics. The model is given by (4) with the target function linear in its arguments.

$$f(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_j x_j. \tag{13}$$

The errors (ε) are taken to be homoscedastic [variance of ε is constant (σ^2) independent of $\boldsymbol{x}$], and one restricts the approximation to also be linear

$$\hat{f}(\boldsymbol{x}) = \sum_{j=1}^{n} \hat{a}_j x_j, \tag{14}$$

with the coefficients $\{\hat{a}_j\}_1^n$ estimated by least squares

$$\{\hat{a}_j\}_1^n = \underset{\{a_j\}_1^n}{\operatorname{argmin}} \sum_{i=1}^{N} \left[y_i - \sum_{j=1}^{n} a_j x_{ij} \right]^2. \tag{15}$$

It is easily shown that in this case

$$E_\varepsilon[f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})]^2 = n\sigma^2/N \tag{16}$$

which increases only linearly with the number of input variables n, not exponentially as the curse-of-dimensionality would suggest.

More generally, suppose

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} \alpha_m B_m(\boldsymbol{x}) \tag{17}$$

where $\{B_m(\boldsymbol{x})\}_1^M$ are fixed prespecified functions of n variables, and we restrict the approximation to

$$\hat{f}(\boldsymbol{x}) = \sum_{m=1}^{M} \hat{a}_m B_m(\boldsymbol{x}), \tag{18}$$

again using least squares to estimate the coefficients

$$\{\hat{a}_m\}_1^M = \operatorname{argmin} \sum_{i=1}^{N} \left[y_i - \sum_{m=1}^{M} a_m B_m(\boldsymbol{x}_i) \right]^2.$$

In this case

$$E_{\epsilon}[f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})]^2 = M\sigma^2/N \tag{19}$$

which is independent of the input dimensionality n.

In both cases (16) (19) the curse-of-dimensionality was overcome by the fact that the target function $f(\mathbf{x})$ (13) (17) was a member of a restricted class of functions and we chose an approximation method (14) (18) especially appropriate for that class. Either we knew the class in advance or were very lucky. In the context of the discussion in Section 2.0, we have restricted the eligible solutions to (11) to be a small subset of all possible functions, and the target function happened to a be member of that subset

As another illustration that complexity rather than dimensionality is the fundamental issue consider two target functions

$$f_n(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_j x_j,$$

$$f_1(x_1) = \sum_{j=1}^{n} \beta_j g_j(x_1).$$

The first $f_n(\boldsymbol{x})$ is a simple (linear) function of n variables and the second $f_1(x_1)$ is a more complicated function of a single variable ($\{g_j(x_1)\}_1^n$ nonlinear). If we choose the most appropriate respective approximators

$$\hat{f}_n(\boldsymbol{x}) = \sum_{j=1}^{n} \hat{a}_j x_j \quad \text{and} \quad \hat{f}_1(x_1) = \sum_{j=1}^{n} \hat{b}_j g_j(x_1)$$

then the approximation difficulty is the same in both cases (16) (19) even though the number of input variables for the first can be much larger than the second.

The basic reason for the curse-of-dimensionality is that functions of a high dimensional argument have the potential to be much more complex that lower dimensional ones and those complications are harder to discern. For a problem with a single input one can make a scatter plot of $\{y_i, x_i\}_1^N$ and graphically view the (approximate) relationship. For two-inputs one can use advanced data visualization techniques to view the resulting three dimensional point cloud. Problems involving many inputs result in correspondingly higher dimensional point clouds that are difficult to visualize.

Since it is impossible to densely populate high dimensional spaces with feasible training samples the only way to overcome the curse-of-dimensionality and be successful is to incorporate knowledge from outside the training data (assumptions) concerning the target function $f(\boldsymbol{x})$. As discussed below, choosing a particular approximation method either explicitly or implicitly

does this. The degree of success depends on how closely the actual target function (in any particular situation) matches the assumptions associated with the chosen method. This was illustrated (in very simple settings) with the above examples. The various available approximation methods differ in the particular nature of the "knowledge" they impose, the strength of that imposition, and their robustness to violations of those assumptions. No method dominates all others over all situations (potential target functions). Also, as noted in Section 1.4, methods differ widely on the interpretability of the approximations they produce.

3.0 Penalization.

One way to restrict solutions to (11) is to add a penalty to the criterion to be optimized

$$\hat{f}(\boldsymbol{x}) = \underset{g(\boldsymbol{x})}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} [y_i - g(\boldsymbol{x}_i)]^2 + \lambda \phi[g(\boldsymbol{x})] \right\}. \tag{20}$$

Here $g(\boldsymbol{x})$ ranges over all possible functions and $\phi[g(\boldsymbol{x})]$ is (non-negative) number (functional) associated with each one. The quantity $\lambda > 0$ ("regularization parameter") is inserted so that the strength of the penalty can be conveniently adjusted separately from its form $\phi[g(\boldsymbol{x})]$. For example, restricting $f(\boldsymbol{x})$ to be member of a particular class of functions [as in (14) or (18)] can be accomplished by setting $\phi[g(\boldsymbol{x})] = \infty$ for all $g(\boldsymbol{x})$ not in that class. Additionally, if some functions within the class were to be favored more than others, they would be assigned relatively smaller values of the penalty.

3.1 Bayesian formulation.

Choice of a penalty (20) is equivalent to supplying external (outside the data) information concerning the target function $f(\boldsymbol{x})$. This is most easily seen in a Bayesian formulation. Bayes theorem states that the probability of a model (approximation) given the training data is proportional to the probability of seeing the data given the model, times the a priori probability of the model. In our notation this becomes

$$\Pr\left[g(\boldsymbol{x})|\{y_i, \boldsymbol{x}_i\}_1^N\right] \sim \Pr\left[\{y_i, \boldsymbol{x}_i\}_1^N | g(\boldsymbol{x})\right] \cdot \Pr\left[g(\boldsymbol{x})\right]. \tag{21}$$

A natural choice for $\hat{f}(\boldsymbol{x})$ would be the function that is most probable given the training data

$$\hat{f}(\boldsymbol{x}) = \underset{g(\boldsymbol{x})}{\operatorname{argmax}} \Pr\left[g(\boldsymbol{x})|\{y_i, \boldsymbol{x}_i\}_1^N\right]. \tag{22}$$

This is known as the maximum aposteriori probability (MAP) estimate. The first factor on the right side of (21) is called the likelihood of the data (given the model $g(\boldsymbol{x})$) and is thus a model for the error mechanism $\epsilon(4)$. Suppose

that the errors have a Gaussian distribution with constant variance $\sigma^2, \varepsilon \sim N(0, \sigma^2)$. Then from (4) the likelihood becomes

$$\Pr\left[\{y_i, \boldsymbol{x}_i\}_1^N | g(\boldsymbol{x})\right] = \Pr\left[\{\boldsymbol{x}_i\}_1^N\right] \prod_{i=1}^{N} \frac{1}{2\pi\sigma} e^{-\varepsilon_i^2/2\sigma^2}$$

with $\varepsilon_i = y_i - g(\boldsymbol{x}_i)$. Substituting this into (21), taking the logarithm of both sides, and discarding terms that do not involve $g(\boldsymbol{x})$ gives an equivalent to (22)

$$\hat{f}(\boldsymbol{x}) = \underset{g(\boldsymbol{x})}{\operatorname{argmin}} \left\{ \frac{1}{\sigma^2} \sum_{i=1}^{N} [y_i - g(\boldsymbol{x}_i)]^2 - 2 \log \Pr\left[g(\boldsymbol{x})\right] \right\}. \tag{23}$$

The solution to (23) is the same as that for (20) provided one makes the association

$$\lambda\phi[g(\boldsymbol{x})] = -2\sigma^2 \log \Pr\left[g(\boldsymbol{x})\right]. \tag{24}$$

Thus, in this setting the penalty in (20) has a direct interpretation involving the a priori probability ("prior") of $g(\boldsymbol{x})$ occurring, which (by definition) can only be determined from information outside the training data.

As the error variance σ^2 increases, (23) shows that the influence of the data on the MAP solution decreases relative to that of the prior probability. In the limit $\sigma^2 \to \infty$, there is no information in the training data and the a priori knowledge incorporated into the prior completely determines the solution (perhaps not uniquely). As the error variance σ^2 decreases the training data receives increasing weight relative to the prior $Pr[g(\boldsymbol{x})]$ in determining the MAP solution. In the limit $\sigma^2 \to 0$ (interpolation problem) only functions that interpolate the training data points are eligible solutions (data term zero). The MAP procedure then chooses the particular one among them with the highest a priori probability as determined by the chosen prior $\Pr[g(\boldsymbol{x})]$, unless all functions $g(\boldsymbol{x})$ that interpolate the data have zero probability (infinite penalty) under that prior. In this case the MAP solution may not be defined (for the interpolation problem) since there is no allowed (nonzero prior probability) function that could have (exactly) produced the observed training sample. This implies that the assumed prior cannot be (completely) correct (reflect reality). However, even in this situation it may be possible to express $\Pr[g(\boldsymbol{x})]$ as a limiting case of a more inclusive prior (permitting functions that interpolate the data) and adjust the limiting rates as that prior approaches $\Pr[g(\boldsymbol{x})]$ so that as $\sigma^2 \to 0$ the corresponding limit of the right hand side of (23) exists. In any case (20) is always well defined and $\lambda\phi[g(\boldsymbol{x})]$ retains the interpretation as representing (user supplied) external information.

One should not interpret the above discussion to imply that in the noiseless case ($\sigma^2 = 0$) an approximation that interpolates the training data is necessarily superior to one that does not. This point is illustrated in Figure 3. The target function is represented by $f(\boldsymbol{x})$, $h(\boldsymbol{x})$ represents a solution

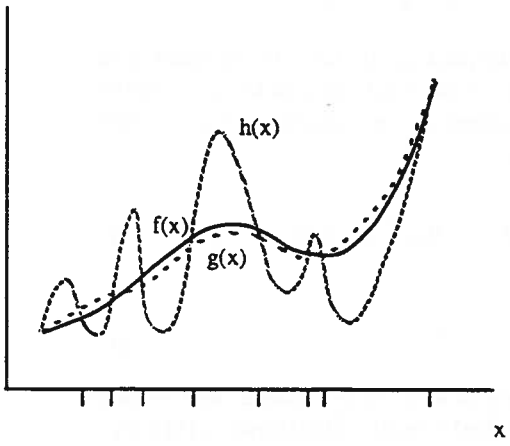


Figure 3. Illustration that interpolators are not necessarily the best approximators.

from (20) obtained by a penalty that was finite (non zero prior probability) for some functions that interpolate the training data points, and $g(x)$ is a solution using another penalty (prior) that precludes data interpolation. As illustrated, the second solution $g(x)$ is superior by almost any reasonable error measure (e.g. *mse* (10)). In this case neither penalty corresponded to a prior (24) that correctly reflected the (relative) probability of realizing the actual target function $f(x)$. The second penalty corresponded to a prior probability that, while giving zero chance to the true target, gave high probability to functions close to it (small *mse* (10)).

3.2 Parametric penalties.

Many of the well known learning (function approximation) methods restrict their solutions to a particular functional form, that is, $\hat{f}(x)$ (20) is a member of a class of functions characterized by a set of parameters $\{\theta_1 \cdots \theta_p\}$

$$\hat{f}(x) \in \{h(x|\theta_1, \cdots, \theta_p)\} \tag{25}$$

(Many examples are provided in the following sections.) This implies that

$$\hat{f}(x) = h(x|\hat{\theta}_1, \cdots \hat{\theta}_p)$$

for some particular set of parameter values $\{\hat{\theta}_1 \cdots \hat{\theta}_p\}$. This can be accomplished by setting $\varphi[g(x)]$ to be infinitely large for all functions not of that

particular parametric family

$$g(x) \notin \{h(x|\theta_1, \cdots \theta_p)\} \Rightarrow \varphi[g(x)] = \infty.$$

This is equivalent to restricting the minimization in (20) to be only over functions in that family (finite penalty). Since each function in the parametric family is uniquely specified by a simultaneous set of parameter values, (20) becomes a parameter estimation problem

$$\{\hat{\theta}_1, \cdots, \hat{\theta}_p\} = \underset{\theta_1, \cdots, \theta_p}{\text{argmin}}\{\sum_{i=1}^{N}[y_i - h(x_i|\theta_1, \cdots \theta_p)]^2 + \lambda\eta(\theta_1, \cdots, \theta_p)\} \quad (26)$$

with

$$\hat{f}(x) = h(x|\hat{\theta}_1, \cdots, \hat{\theta}_p). \quad (27)$$

Because of the one-to-one mapping between sets of parameter values and functions in the parametric family, the (finite) penalty functional $\varphi[h(x|\theta_1, \cdots, \theta_p)]$ becomes a (simple corresponding) function of the parameters $\eta(\theta_1, \cdots, \theta_p)$ as indicated in (27).

If all functions $h(x|\theta_1, \cdots \theta_p)$ are taken to have equal a priori probability then the (least restrictive) parameter penalty function $\eta(\theta_1, \cdots \theta_p)$ would be a constant independent of the parameter values. If one has a prior preference for certain functions among this class then this would translate into a non-constant penalty with smaller values for (those parameter values corresponding to) the more (a priori) highly favored functions. Two such penalties that are very popular are

$$\eta_r(\theta_1, \cdots, \theta_p) = \sum_{j=1}^{p} \theta_j^2 \quad \text{(ridge)} \quad (28)$$

and

$$\eta_s(\theta_1, \cdots, \theta_p) = \sum_{j=1}^{p} I(\theta_j \neq 0) \quad \text{(subset selection)}. \quad (29)$$

The ridge penalty (28) favors approximations corresponding to small values of the parameters and causes a preference for solutions with many small values over those with a wide variation of parameter values. It corresponds to a Gaussian prior probability distribution on the parameters centered at zero (for all parameters) with covariance matrix proportional to the identity matrix. The strength parameter λ (26) characterizes the (common) variance under this prior for each parameter.

The subset selection penalty (29) simply counts the number of non zero parameter values. That is

$$I(\theta_j \neq 0) = \begin{cases} 1 & \text{if } \theta_j \neq 0 \\ 0 & \text{if } \theta_j = 0 \end{cases} \quad (30)$$

It clearly favors approximations with a large number of the parameters set to zero, but does not specify which ones. The strength parameter λ (26) in this case can be viewed as a charge or cost for each parameter that is allowed to enter (non zero value). This penalty (29) (30) can take on only $p+1$ distinct (integer) values from zero (all parameters equal to zero) to λp (all non zero). For each distinct value λk ($k = 1, p$) minimizing (26) is equivalent to minimizing the data term

$$\hat{f}_k(\boldsymbol{x}) = \underset{\#\{\theta_j \neq 0\}_1^p = k}{\operatorname{argmin}} \sum_{i=1}^{N} [y_i - h(\boldsymbol{x}_i|\theta_1, \cdots, \theta_p)]^2 \tag{31}$$

under the constraint that the number of non zero parameter values $\#\{\theta_j \neq 0\}_1^p$ is equal to k. It (31) thus selects the best subset of k (out of p) nonzero parameters (and their optimal values). For a given value of $\lambda \geq 0$ one of these k subset solutions will minimize (26). Thus, as the value of λ is increased from zero (all parameters potentially nonzero) to infinity (all parameters equal to zero), each of the k subset solutions (31) in turn becomes the solution to (26) and there are there therefore only $p+1$ distinct solutions. Thus one could equivalently regard k as the regularization parameter.

If both $\eta(\theta_1, \cdots, \theta_p)$ and $h(\boldsymbol{x}|\theta_1, \cdots, \theta_p)$ are continuous functions of the parameters, and the parameters themselves take on continuous real values, then the solution to (26) can be obtained by numerical optimization techniques. The ridge penalty (28) is such a continuous penalty function. However, the subset selection penalty is not a continuous function of its arguments owing to the discontinuous indicator function (30). Therefore, combinatorial optimization is required to obtain a solution to (26) (29) (30). Basically this reduces to minimizing (31) over all possible $\binom{p}{k}$ subsets of size k, for $k = 1, p$. This is generally too expensive computationally. For some especially simple parameterizations, e.g. linear

$$h(\boldsymbol{x}|\theta_1, \cdots, \theta_p) = \sum_{k=1}^{p} \theta_k B_k(\boldsymbol{x}) \tag{32}$$

"branch and bound" techniques can be derived (Furnival and Wilson, 1974) to greatly spreed up the combinatorial optimization making it feasible for $p \leq 30$. Otherwise, fast (combinatorial) techniques for obtaining approximate solutions are used based on greedy (stepwise) strategies or random search (genetic) algorithms (Holland, 1975).

Another approach is to approximate the discontinuous penalty (30) by a close continuous one, thereby enabling the use of numerical optimization. This is motivated by the observation that both (28) and (29) (30) can be viewed as two points on a continuum of penalties, such as

$$\eta_q(\theta_1, \cdots, \theta_p) = \sum_{j=1}^{p} |\theta_j|^q \quad \text{("bridge")} \tag{33}$$

(Frank and Friedman, 1993), or

$$\eta_w(\theta_1, \cdots, \theta_p) = \sum_{j=1}^{p} \frac{(\theta_j/w)^2}{1+(\theta_j/w)^2} \quad (\text{"weight decay"}) \tag{34}$$

(Wiegand, Huberman and Rumelhart, 1991). With the "bridge" penalty (33) $q = 2$ yields the ridge penalty (28), whereas subset selection (29) (30) is approached in the limit as $q \to 0$. For "weight decay" (34) the ridge penalty (28) is approached as $w \to \infty$, while subset selection is approached in the opposite extreme $w \to 0$. Both penalties are continuous functions of the parameters $\{\theta_1, \cdots, \theta_p\}$ for $(q, w) > 0$, and the smaller their values the more (33) (34) encourage diverse parameter values, behaving more like the subset selection penalty (29) (30). Care must be taken, however, not to approach subset selection too closely (q, w too small) since as q, w become smaller, an increasing number of local minima are included in the objective function (26), (33) or (34), making the numerical optimization more difficult. In the limit $q, w = 0$, there are so many local minima the optimization effectively becomes combinatorial.

A particular parametric form (25) characterizes a parametric family of functions, but that characterization may not be unique. There may be a wide variety of parameterizations that specify the same parametric family. Consider, for example, the linear case (32). In this case the parameterization is in terms of a linear expansion in a set of (basis) functions $\{B_k(\boldsymbol{x})\}_1^p$ with the parameters $\{\theta_1, \cdots, \theta_p\}$ being the expansion coefficients. Clearly one could obtain an equivalent representation

$$h(\boldsymbol{x}|\alpha_1, \cdots, \alpha_p) = \sum_{l=1}^{p} \alpha_l C_l(\boldsymbol{x}) \tag{35}$$

where each

$$C_l(\boldsymbol{x}) = \sum_{k=1}^{p} \beta_{lk} B_k(\boldsymbol{x}), \quad l = 1, p$$

is a (specified) linear combination of $\{B_k(\boldsymbol{x})\}_1^p$ and

$$\theta_k = \sum_{l=1}^{p} \alpha_l \beta_{lk}, \quad k = 1, p.$$

Although (32) and (35) have a different form they both specify the same family of functions.

If all members of the parametric family are taken to be equally likely (constant penalty function in (26), or $\lambda = 0$) then the particular form of its representation does not matter. Different representations will yield different solutions for the parameter values in (26) but they will correspond (uniquely)

to the same function (27). This continues to be true if a penalty is applied that depends on the actual function, i.e. $\varphi[h(\boldsymbol{x}|\theta_1, \cdots, \theta_p)]$, and not on its representation (32) (35). However, this is not the case for (generic) penalties that are expressed purely in terms of parameter values such as (28), (29), (33) or (34). Changing the representation (e.g. (32) to (35)) changes the function penalty $\phi[g(\boldsymbol{x})]$ giving rise to different solutions to (26) (27). Therefore, specifying a penalty of this type only specifies a function penalty in the context of a particular representation of the parametric family, and changing the representation changes which functions are favored within the family. Although obvious, this point is often overlooked.

The ridge penalty (28) can be viewed as the Euclidean distance (squared) from the origin to a point $\theta = \{\theta_1, \cdots, \theta_p\}$ in the p-dimensional parameter space. It thus places highest prior probability on the function $h(\boldsymbol{x}|\theta_1, \cdots, \theta_p)$ corresponding to all parameters set to zero. There is clearly nothing fundamental about the origin; a similar ridge penalty could be defined in term of Euclidean distance squared from any prespecified point $\theta_0 = \{\theta_1^{(0)}, \cdots, \theta_p^{(0)}\}$

$$\eta_r(\theta_1, \cdots, \theta_p) = \sum_{j=1}^{p} (\theta_j - \theta_j^{(0)})^2. \tag{36}$$

There is also nothing special about simple Euclidean distance; one could, for example, multiply each of the squares in (36) by a scale factor w_j making some parameters more important than others in defining the penalty.

One method of defining a distance penalty that is very popular in the neural network community consists of defining a particular (one-dimensional) path ("curve") through the p-dimensional parameter space, and restricting solutions to those that lie on the specified path. The penalty is taken to be the distance along the path (arc-length) from some prespecified point θ_0 also lying on the path. A one dimensional locus of points ("curve") in the p-dimensional parameter space can be defined by specifying a set of p functions

$$\theta_j = \Theta_j(s), \quad j = 1, p, \tag{37}$$

each of the same (one-dimensional) argument s. Changing the value of s changes the point $\theta = \{\theta_1, \cdots, \theta_p\}$ in the parameter space, and the locus of all such points, corresponding to all values of s $(-\infty < s < \infty)$, maps out a one-dimensional manifold (curve) in the parameter space. The nature or topology of such a curve is determined by the particular functions $\{\Theta_j(s)\}_1^p$ (37) used to define it.

One (but not the only) way to specify these functions is through a set of differential equations

$$\frac{d\Theta_j(s)}{ds} = \frac{\partial U}{\partial \theta_j} \tag{38a}$$

where $U(\theta_1, \cdots, \theta_p)$ is (yet another) specified function of the parameters. The solutions are

$$\Theta_j(s) = \int_{s_0}^{s} \frac{d\Theta_j}{ds}(s')ds' + \theta_j^{(0)} \tag{38b}$$

with the derivatives defined in (38a), and $\{\theta_j^{(0)} = \Theta_j(s_0)\}_1^p$ represents and arbitrary starting point (on the curve). Here (38) the path is defined by picking a starting point $\{\theta_j^{(0)}\}_1^p$ and then taking repeated (infinitesimal) steps in the direction of the gradient (38a) of the chosen function $U(\theta_1, \cdots, \theta_p)$ in the parameter space. The penalty $\eta(\theta_1, \cdots, \theta_p)$ (26) is taken to be infinite for points $\{\theta_1, \cdots, \theta_p\}$ not lying on the path and proportional to distance along the path from the starting point $\{\theta_1^{(0)}, \cdots, \theta_p^{(0)}\}$ otherwise.

Defining a penalty in this framework requires the specification of a starting point $\{\theta_1^{(0)}, \cdots,$
$\theta_p^{(0)}\}$ and a function $U(\theta_1, \cdots, \theta_p)$ whose gradient determines the particular path. A particularly convenient choice for the latter is

$$U(\theta_1, \cdots, \theta_p) = \sum_{i=1}^{N} [y_i - h(\boldsymbol{x}_i|\theta_1, \cdots, \theta_p)]^2 \tag{39}$$

This is just the data part of the combined (penalized) criterion (26) whose minimum defines the solution approximation $\hat{f}(\boldsymbol{x})$ (27). Common choices for a starting point are all parameters set to zero (or nearly so if the gradient happens to be zero at that point) or a point chosen at random in the parameter space. Beginning there, a small step is taken in the direction of the gradient (38a) (39). The gradient is evaluated at the new point so obtained, and the next step is taken along this new gradient direction. These steps are continued until the gradient becomes zero, indicating a (local) minimum of (39).

The (numerical) procedure for obtaining the path (37) – (39) is just the steepest descent procedure for minimizing (39) and thereby providing the solution to (26) for $\lambda = 0$. This is the solution approximation $\hat{f}(\boldsymbol{x})$ under a prior that considers all functions $h(\boldsymbol{x}|\theta_1, \cdots, \theta_p)$ equally likely. For positive values of the strength parameter, $\lambda > 0$, a penalty is incurred proportional to the path length along the steepest-descent trajectory from the starting point. If the step lengths are equal this is proportional to the number of steps taken, and even when they are not equal, the number of steps is most often used

$$\eta(\theta_1, \cdots, \theta_p) = \#\,[\text{gradient descent steps}]. \tag{40}$$

This ($\lambda > 0$) will cause the solution approximation (26) (40) to be another point on the steepest descent path uphill from the minimum of (39) (or (26) with $\lambda = 0$). One can view the value of λ as a charge or cost associated with each step and increasing its value will reduce the number of steps associated

with the solution approximation. This one-to-one correspondence between a λ value and the number of allowed steps implies that the latter could equally serve as the regulating strength parameter.

There is nothing inherently fundamental in choosing (39) to characterize the path of solutions to be considered. It corresponds to a prior belief that the target function is likely to be close to one of those on the path, preferably closer to the starting point. This may or may not be true and there may well be another path, defined by another $U(\theta_1, \cdots, \theta_p)$, that serves this purpose better in terms of coming closer to the target function (earlier). Any such path, however, should include the point that minimizes (39) (data part of (26)) so that there is not a discontinuity in the criterion (26) at $\lambda = 0$. Another important property that should apply to any path selected for this purpose is that (39) be a monotonically decreasing function of the number of steps taken along the path. That is, each step should be in a descent direction of (39) (data part of (26)). Since the penalty (26) (40) is monotonically increasing, this insures a unique minimum of (26) along the path and a one-to-one correspondence between λ-values and the number of resulting steps associated with the solution. Choosing (39) to define the path meets all of these criteria but there are other paths that also meet them. For example, a trajectory generated by any other numerical optimization algorithm (such as conjugate gradients, Gauss-Newton, etc.) shares these properties.

Like those defined by (28) (29) (33) (34), penalties defined in this manner (39) (40) depend on the particular parametric family chosen as well as the choice of a particular representation (e.g. (32) or (35)) of the functions in that family. In addition they depend on the particular one-dimensional path of eligible solutions chosen in the parameter space. If this path depends on the training data (as in (39)) it will not even be known in advance. As a result it is difficult to interpret such a penalty in terms of the actual functions it tends to favor as solutions to (26) (27), making the incorporation of actual a priori knowledge difficult. In fact, this data dependence clouds the interpretation connecting the penalty to such prior knowledge through the Bayesian framework (24). It is clear, however, that penalties of this type do favor certain solutions over others and those that are favored depend on (the definition of) the particular path that is chosen.

3.3. Nonparametric penalties.

Constraining an approximating function $\hat{f}(\boldsymbol{x})$ to be a member of a specified parametric family (25) is one way an approximation method can require its solutions to be smooth. In all applications of this approach the parametric form $h(\boldsymbol{x}|\theta_1, \cdots, \theta_p)$ is a smooth function of $\boldsymbol{x}$ for all (allowable) values of the parameters $\{\theta_1, \cdots, \theta_p\}$. As noted in Section 2.0, it is the smoothness property that is basically used to infer function values in $\boldsymbol{x}$-regions where data is sparse, which is everywhere in high dimensional $\boldsymbol{x}$-spaces. However, the parametric constraint (25) imposes more than just smoothness, since a

large number of parametric families produce a smooth function (of $\boldsymbol{x}$) and the constraint restricts the solution to be in just one of them. This has motivated the formulation of smoothness penalties based on differential operators, that do not (explicitly) involve parametric restrictions. These penalties have the form

$$\varphi[g(\boldsymbol{x})] = \int_{R^n} |Dg(\boldsymbol{x})|^2 d\boldsymbol{x}, \tag{41}$$

where $Dg(\boldsymbol{x})$ is a differential operator. Examples of such differential operators are the (norm of the) gradient

$$|Dg(\boldsymbol{x})|^2 = \sum_{j=1}^{n} \left(\frac{\partial g}{\partial x_j}\right)^2 \tag{42a}$$

or the Laplacian

$$|Dg(\boldsymbol{x})|^2 = \sum_{j=1}^{n}\sum_{k=1}^{n} \left(\frac{\partial^2 g}{\partial x_j \partial x_k}\right)^2. \tag{42b}$$

Embodied in these (and other) definitions is the notion that wigglyness is the opposite of smoothness and ought to be penalized; the penalties (41) (42a,b) clearly get larger as $g(\boldsymbol{x})$ oscillates more.

These nonparametric penalties (41 - 42) seem more general than their parametric counterparts (25) in that they appear to directly enforce smoothness without any parametric constraints. However this is not necessarily the case. Solutions to (20) involving penalties of the form (41) generally are members of particular parametric families with parameter penalties similar to (28). The number of parameters p, however is usually equal to the training sample size N. For example, solutions to (20) with penalty (41) (42) lead to basis function expansions (32) with $p = N$ and

$$B_k(\boldsymbol{x}) = \begin{cases} r_k^{2n+1}, & n = \text{odd} \\ r_k^{2n} \log r_k, & n = \text{even} \end{cases} \tag{42a}$$

with $r_k = [(\boldsymbol{x} - \boldsymbol{x}_k)^T(\boldsymbol{x} - \boldsymbol{x}_k)]^{1/2}$ being the Euclidean distance from $\boldsymbol{x}$ to each sample point $\boldsymbol{x}_k$. Wahba (1990) and Girosi, Jones and Poggio (1993) establish similar relationships for a wide variety of other penalties based on differential operators and their corresponding parametric representations in terms of basis function expansions (32).

3.4. Model Selection.

For a given training data set (7), choice of a penalty $\lambda\varphi[g(\boldsymbol{x})]$ (20) determines the solution approximation $\hat{f}(\boldsymbol{x})$. The preceding sections discussed the relationships among several popular (broad classes of) choices for such penalties. Specific choices are associated with particular procedures (discussed below). Thus (given the data) the accuracy (10) of the approximation will depend

on penalty choice. Clearly, the best penalty $\varphi[g(\boldsymbol{x})]$ for a particular target function $f(\boldsymbol{x})$ will be one that has small values for $g(\boldsymbol{x}) \approx f(\boldsymbol{x})$ and large values otherwise. This is why the penalty should incorporate as much (prior) information about $f(\boldsymbol{x})$ that is known to the user. This knowledge (by definition) cannot completely determine target $f(\boldsymbol{x})$, otherwise there is no need for supervised learning. It is the purpose of the data (first) part of the criterion (20) to use information from the training data to complement the prior information incorporated into the penalty to produce a usable approximation $\hat{f}(\boldsymbol{x})$ in terms of acceptably small values of *mse* (10).

Strict application of the Bayesian paradigm (21) (22) forbids the use of the training data in formulation of a prior probability model leading to a specific penalty term (20). The likelihood $\Pr[g(\boldsymbol{x})|\,\text{data}]$ captures the data information and the prior $\Pr[g(\boldsymbol{x})]$ captures all information outside the data. A problem with this approach that it presumes a correctly specified prior. This is difficult is practice. First, it is not always easy to accurately incorporate one's knowledge in terms of a mathematical formula for a penalty, and second, one may in fact posses very little knowledge concerning the nature of the target function. On the other hand a penalty is required to make (20) a well posed problem. In such cases, the pure Bayesian philosophy requires one to choose a prior (penalty) and simply hope that (if incorrect) it does not overwhelm the (information contained in) the training data.

The "empirical" Bayesian paradigm attempts to mitigate this problem. In this approach one incorporates prior knowledge into a penalty $\varphi[g(\boldsymbol{x})]$ (20) but allows the training data to choose the values of the "strength" parameter λ. This parameter is viewed as regulating the degree-of-belief in the corresponding prior information. If the training data appear highly compatible with the prior penalty, λ is given a larger value than would be the case if the data seemed to contradict the prior information (i.e. a surprise). In this way one "hedges one's bet" on the reliability of the chosen penalty to favor functions close to the true (but unknown) target function (low penalty value) and discourage competing functions far from it (high penalty value). If the penalty is deemed to be good in this sense an appropriately large value of λ is used, resulting in a substantial increase of accuracy, whereas if not, a small value is selected thereby controlling the corresponding damage. Thus, one wins big with a good penalty choice (method) but does not encounter a catastrophe with a bad choice.

A penalty with a corresponding strength parameter value $\lambda\varphi[g(\boldsymbol{x})]$ is deemed to be good to the extent that its use (20) yields an accurate estimate of the target function (10). This implies that for a given penalty $\varphi[g(\boldsymbol{x})]$ we want to choose the strength parameter to minimize *mse* (10) or equivalently the *mspe* (9)

$$mspe(\lambda) = \int E_{\epsilon}[y - \hat{f}(\boldsymbol{x} \mid \lambda)]^2 p(\boldsymbol{x}) d\boldsymbol{x} \tag{43}$$

with y given by (4). Here (43), the dependence of the solution to (20) on

the strength parameter λ (given $\varphi[g(\boldsymbol{x})]$) is explicitly identified. An optimal value for λ, λ^*, would minimize (43),

$$\lambda^* = \underset{\lambda}{\operatorname{argmin}}\, mspe(\lambda). \tag{44}$$

Unfortunately, the target $f(\boldsymbol{x})$ is unknown so that (43) (or equivalently (10)) cannot be explicitly evaluated (for each λ-value) and minimized (44).

If we have a data sample presumed to have been generated by the same (probability) model (4) (6), we can estimate (43) by

$$\widehat{mspe}(\lambda) = \frac{1}{N_T}\sum_{i=1}^{N_T}[y_i - \hat{f}(\boldsymbol{x}_i|\lambda)]^2 \tag{45}$$

and

$$\hat{\lambda} = \underset{\lambda}{\operatorname{argmin}}\, \widehat{mspe}(\lambda). \tag{46}$$

Here (45) N_T is the sample size. The training data (7) is just such a sample. However, it is clear from the definition of $\hat{f}$ (20) that (45) (46) will always yield $\hat{\lambda} = 0$ as a solution. By definition the unpenalized solution provides the best fit to the training data, but this is generally not the case for (future) data to be predicted. Thus, (45) (46) must be evaluated for such future data, which (again by definition) we don't have.

A common approach to this dilemma is to divide the training sample (7) into two (disjoint) parts, a "learning" sample

$$\{y_i, \boldsymbol{x}_i\}_1^{N_L} \tag{47}$$

and a "test" or "prediction" sample

$$\{y_i, \boldsymbol{x}_i\}_1^{N_T} \tag{48}$$

where $N_L + N_T = N$, the training sample size. The presumption here is that the test sample (48) is representative of future data to be predicted, which will be the case if the training sample is so representative and the division (47) (48) is done randomly. The learning sample (47) is used in (20) to obtain the approximations $\hat{f}(\boldsymbol{x}|\lambda)$ and the test sample is used to evaluate (45) for each of them. Then (46) is used to estimate the optimal value $\hat{\lambda}$ of the strength parameter λ. This approach is usually referred to as "cross-validation," and is often very effective.

One problem with this form of cross-validation is that the resulting estimate $\hat{\lambda}$ depends on the detailed manner in which the training sample is partitioned into learning and test samples, in terms of both their relative sizes and which particular observations are assigned to the respective samples. If the training sample is very large, so that both the learning and test

samples can also be large, then this does not represent a major difficulty; both will give reliable estimates of the corresponding predictive squared errors (20) (45). If this is not the case then this indeterminancy translates into increased inaccuracy in both $\hat{f}(x \mid \lambda)$ (20) and the estimate $\hat{\lambda}$ (45) (46). This has motivated modifications to this simple method of cross-validation.

Given the sizes of the respective (learning and test) samples (N_L and N_T) the ambiguity associated with the particular assignments to each can be eliminated by computing (20) (45) for all $\binom{N}{N_L}$ possible assignments and then minimizing the average of (45) over these replications. This is of course not feasible for even moderately small values of N, so it can be approximated by many repeated random divisions and averaging the results. Even this can represent too much of a computational burden when the evaluation of (20) is computationally intensive, so generally only a few replications are performed. In this case the procedure is often approximated by "K-fold" cross-validation. Here the training sample is divided into K disjoint subsamples of (approximately) equal size. Each is used in turn as a test sample (48) with the complement sample serving as the learning sample (47). This is like the random division approach except the sampling is done without replacement and repeated K-times. Thus, each training observation appears in one and only one of the test samples and is used exactly once to evaluate (the average of) (45). This can be expressed as

$$\widehat{mspe}(\lambda) = \frac{1}{N}\sum_{i=1}^{N}[y_i - \hat{f}_{\setminus i}(x_i|\lambda)]^2 \tag{49}$$

where $\hat{f}_{\setminus i}(x_i|\lambda)$ indicates the solution to (20) for that learning sample in which the ith observation does not appear. This approach does not remove the ambiguity associated with observation assignment to each of the K test samples; that is which observations are assigned together in each individual group. It does however insure that each and every training observation is used to estimate the *mspe* (λ) (49) the same number of times (once).

None of these cross-validation methods (so far) removes the ambiguity in the relative sizes N_L, N_T of the learning and test samples. This can be done in the context of K-fold cross-validation by setting $K = N$; that is each test subsample consists of one of the training observations ($N_T = 1$) and the corresponding learning sample consists of the remining observations ($N_L = N-1$). This is the well known (in statistics) "leave-one-out" estimate of *mspe* (λ) (49). In fact, $K < N$ fold cross-validation is often motivated as being a computationally faster approximation to the leave-one-out estimate, since it requires only $K < N$ solutions to (20). However, there is evidence that setting $K = N$ is not necessarily the best choice, and that more accurate estimates of $\hat{\lambda}$ can be derived for smaller values [Efron (1983), Breiman and Spector (1989)].

Criteria such as (45) (48) or (49) that are used to select a penalty strength parameter value are known as "model selection" criteria. Although

easiest to motivate, criteria based on cross-validation are not the only ones used for model selection. Most model selection criteria are based on trying to approximate *mspe* (λ) (43) from the training data and then minimizing it with respect to λ (44). Cross-validation is one way to attempt this but there are others such as AIC (Akaike, 1974), C_p (Mallows, 1973), PSE (Barron, 1984), GCV (Caven and Wahba, 1979), and Moody (1992). Notable exceptions to this paradigm for model selection are MDL (Rissanen, 1983) and BIC (Schwartz, 1978). Model selection is still a very active research topic and there is no universal agreement as to which methods are best. Complete treatment of this subject is beyond the scope of this article.

Once a model selection criterion is chosen, solutions to (20) (26) $\hat{f}(\boldsymbol{x}|\lambda)$ are obtained for a sequence of λ-values. Each are evaluated, (45) (48) or (49), and the (estimated) best one is chosen (46). One could also employ various directed single parameter optimization strategies such as repeated bisections or "golden" searches (see Gill, Murray, and Wright, 1981). These can reduce the number of required solutions to (20) (26) which can be especially beneficial if obtaining those solutions is computationally intensive.

In the special case where solutions are restricted to the steepest-descent path (38) (39) (40), and a single learning-test division (47) (48) is employed for cross-validation, one can minimize (26) and (45) simultaneously in a single operation. Beginning with the starting parameter values $\{\theta_1^{(0)}, \cdots, \theta_p^{(0)}\}$, steepest descent steps are taken in the parameter space until a (local) minimum of (39) is reached. This local minimum represents a solution of (26) for $\lambda = 0$. Each set of parameter values corresponding to all earlier steps along the path represents a solution to (26) for (all) larger values of λ. Therefore, beginning at the starting point, the approximation (27) corresponding to each set of parameter values $\{\theta_1, \cdots, \theta_p\}$, associated with each successive step, are evaluated on the left-out test sample (27) (45) as the gradient search proceeds (to the minimum) of (39). The set with the smallest value for (45) is selected (46). Therefore, only one minimization (steepest descent) provides all the approximations corresponding to all λ-values, as opposed to a separate minimization of (20) or (26) (27) to obtain $\hat{f}(\boldsymbol{x}|\lambda)$ for each λ-value. This implementational convenience is the principal motivation for this approach to model selection, rather than any belief that approximations corresponding to points along the steepest-descent path are especially likely to be close to the target function $f(\boldsymbol{x})$. A major computational disadvantage is that steepest-descent is used for the numerical optimization, which is generally one of the least efficient methods. For many $g(\boldsymbol{x})$ (20) or $h(\boldsymbol{x}|\theta_1, \cdots, \theta_p)$ (26) and penalties $\varphi[g(\boldsymbol{x})]$ or $\eta(\theta_1, \cdots, \theta_p)$, the minima can be found by direct linear algebra or more sophisticated numerical techniques such as conjugate gradient or Gauss-Newton methods (see Gill, Murray, and Wright, 1981).

Model selection criteria (based on estimating *mspe*) such as cross-validation (49) can be used to obtain an estimate of *mspe* for any approximation $\hat{f}(\boldsymbol{x})$. In particular, one can compare solutions to (20) for different types of

penalties $\varphi[g(\boldsymbol{x})]$ as well as for relative strengths λ for a given type. Since the penalty type is usually dictated by the method used, approximations derived from different methods can be compared, in any given problem, by cross-validation. This can be a quite useful way to choose (estimate) the best method for the problem at hand (training sample (7) and target function $f(\boldsymbol{x})$ (4) (6)). If some (or all) of the methods being compared themselves intrinsically involve model selection (for a strength parameter λ (45) (48) or (49)) then such comparisons must be done with care. One cannot use the minimized criterion value (with respect to the strength parameter) directly for comparison since it tends to provide an over optimistic (too small) estimate owing to the minimization. The degree of over optimism depends on the particular method. An "honest" (unbiased) estimate of *mspe* for a given method can be obtained by first (randomly) dividing the data sample into a training sample and a prediction sample. The training sample is used for all estimation including the strength parameter (perhaps by dividing it into learning and test samples) and the left out prediction sample is used only to estimate *mspe* for the complete procedure. This is known as "double" cross-validation if cross-validation (45) (48) or (49) is used to estimate the strength parameter. Although this "double" procedure provides an unbiased estimate, it still can be highly variable owing to the ambiguities associated with choice of a prediction sample discussed above. One can *estimate* which procedure is best, but not *determine* it, in any given situation.

3.5. Bias-Variance Trade-Off.

Model selection was motivated in the previous section through the empirical Bayes paradigm. It can also be motivated from another ("frequentist") perspective involving the so called "bias-variance trade-off." This involves the identification of two distinct mechanisms that contribute to approximation error (10); these are the bias and the variance.

The variance contribution is a result of the random nature of the training data (7). Training data sets sampled from the same system at different times will generally not be identical. This can be expressed in terms of an (unknown) probability (density) model $p(\boldsymbol{x})$ associated with the system, that gives the relative probability of observing a particular input vector $\boldsymbol{x}$ from that system. In addition, there is the randomness associated with the error ε (4) representing the effect of the unobserved inputs $\{z_1, \cdots, z_L\}$ (1) (if any) which is also expressed in terms of a probability distribution $F_\varepsilon(\varepsilon)$. A particular training data set (7) is presumed to be a random sample drawn according to these probability distributions with the outputs y (4) being the (deterministic) target function $f(\boldsymbol{x})$ evaluated at the random $\boldsymbol{x}$-points with random noise ε added to it.

The criterion (20) to be minimized depends on the training data through its first term. Therefore its value for a given argument $g(\boldsymbol{x})$ will be random as a consequence of the randomness associated with the training data. Thus the

location of its minimum $\hat{f}(x)$ will also be random; different training samples from the same system will give rise to different (estimated) approximations $\hat{f}(x)$. This is the source of the variance component of the approximation error. It can be quantified by the definition

$$var[\hat{f}(x)] = E[\{\hat{f}(x) - E[\hat{f}(x)]\}^2] \tag{50}$$

at each point x in the input variable space. Here (50), the expected value $E[\cdot]$ represents an average over all training samples (7) (of size N) that could be realized from the system with probabilities governed by $p(x)$ and $F_\epsilon(\epsilon)$; $E[\hat{f}(x)]$ is the average approximation value at x (over all such training samples). The var $[\hat{f}(x)]$ characterizes the (square of the) dispersion of $\hat{f}(x)$ values about its mean. The variance of the whole approximation is characterized as the point-wise variance (50) averaged over all points

$$var\hat{f} = \int var[\hat{f}(x)]p(x)dx. \tag{51}$$

Since the randomness of the training data gives rise to a variety (distribution) of different approximations $\hat{f}(x)$, and since some of these will be more accurate than others (10), the variance will contribute to approximation error (10). This can be quantified by the well know identity

$$E[\{f(x) - \hat{f}(x)\}^2] = var[\hat{f}(x)] + \{f(x) - E[\hat{f}(x)]\}^2 \tag{52}$$

with var$[\hat{f}(x)]$ given by (50). As in (50) the $E[\cdot]$ operation in (52) represents an average over all $\hat{f}(x)$ that can be obtained from (20) for all possible different training data sets that can be realized from the system with appropriate probabilities. Thus, the left hand side of (52) represents the average mean-squared-error (at each point x) for a particular method characterized by a penalty $\varphi[g(x)]$ and a given strength parameter value λ (20). This is equal to the corresponding variance of $\hat{f}(x)$ plus the last term in (52). This last term is the difference (squared) between the target function $f(x)$ and the average approximation value $E[\hat{f}(x)]$ (at the point x). This difference is called the "bias" of the $\hat{f}(x)$ values (at x).

$$bias[\hat{f}(x)] = f(x) - E[\hat{f}(x)]. \tag{53}$$

Defining the mean-squared-error (at x)

$$mse[\hat{f}(x)] = E[\{f(x) - \hat{f}(x)\}^2] \tag{54}$$

one has from (52)

$$mse[\hat{f}(x)] = var[\hat{f}(x)] + bias^2[\hat{f}(x)], \tag{55}$$

and for the corresponding global averages

$$mse\hat{f} = \int mse[\hat{f}(\boldsymbol{x})]p(\boldsymbol{x})d\boldsymbol{x} \tag{56a}$$

$$bias^2\hat{f} = \int bias^2[\hat{f}(\boldsymbol{x})]p(\boldsymbol{x})d\boldsymbol{x}, \tag{56b}$$

one has from (55)

$$mse\hat{f} = var\hat{f} + bias^2\hat{f}. \tag{56c}$$

The random mechanism (probability distributions) associated with the training data $(\boldsymbol{x}, \varepsilon)$ are presumed to be a property of the system producing the data and not under the user's control. This is referred to as an "observational" setting. In a "designed experiment" the user can choose the set of $\boldsymbol{x}$-values $\{\boldsymbol{x}_i\}_1^N$ but the errors ε (4) are still random. In both cases the user has control over the choice of penalty $\varphi[g(\boldsymbol{x})]$ (20) and its strength λ, with the goal of minimizing the mean-squared error (56). For a given penalty $\varphi[g(\boldsymbol{x})]$, increasing the value of $\lambda > 0$ reduces the variance (50) (51) because it increases the importance of the penalty part of (20) relative to the (random) data contribution (first term) of the criterion. If the penalty part does not depend on the training data [e.g. (28) (29) (33) (34)], then it is a deterministic quantity (functional) and has no random component; it contributes nothing to the variation of the criterion (20) and thus to the variation in the location of its minimum $\hat{f}(\boldsymbol{x})$. As the relative importance to the criterion (20) of the (random) data term is diminished by increasing the value of λ, the criterion variation itself is diminished, thereby decreasing the variation in the location of its minimum. In the limit $\lambda \to \infty$, the criterion becomes deterministic as does its minimum and $var\ \hat{f} = 0$ (51) (provided the minimum of $\varphi[g(\boldsymbol{x})]$ is unique).

By altering the value of the strength parameter λ the user has control over the variance of the approximation $\hat{f}(\boldsymbol{x})$; it can be reduced at will. However, decreasing the variance (51) does not necessarily decrease prediction error (56) owing to the bias contribution. Increasing the value of λ may increase the bias-squared, and that increase may more than offset the resulting decrease in variance, resulting in increased $mse\hat{f}$ (56). The rate of increase of $bias^2\hat{f}$ with increasing λ-values depends on the true target function and the chosen penalty $\varphi[g(\boldsymbol{x})]$ (20). As $\lambda \to \infty$ the bias-squared approaches the limit.

$$\lim_{\lambda\to\infty} bias^2\hat{f}(\boldsymbol{x}|\lambda) \to \int [f(\boldsymbol{x}) - g^*(\boldsymbol{x})]^2 p(\boldsymbol{x})d\boldsymbol{x} \tag{57}$$

where $g^*(\boldsymbol{x})$ is a function that minimizes the penalty

$$g^*(\boldsymbol{x}) = \underset{g(\boldsymbol{x})}{\operatorname{argmin}} \varphi[g(\boldsymbol{x})]. \tag{58}$$

(If the minimum (58) is not unique, then the one closest to the target $f(\boldsymbol{x})$ is used in (57)). This (57) is the mean-squared-error associated with the function(s) most favored by the chosen penalty (20) (58). If the target $f(\boldsymbol{x})$ happens to a minimizer of the penalty $f(\boldsymbol{x}) = g^*(\boldsymbol{x})$, then no bias increase is incurred by increasing the value of λ and the optimal choice would be $\lambda = \infty$ (minimum variance).

If the target $f(\boldsymbol{x})$ is not a minimizer of the chosen penalty (the usual case) then the dependence of the bias-squared on λ-value can be quite complicated in general. However, in some specific cases it can be determined. Suppose, for example that the approximation is restricted to lie in a parametric family (25) (the penalty $\varphi[g(\boldsymbol{x})] = \infty$ for functions not in that family) so that the estimate is obtained from (26) (27). Suppose further that the parameterization is linear (32); that is, the approximation is taken to be an expansion in a set $\{B_k(\boldsymbol{x})\}_1^p$ of predetermined (basis) functions. If the true target function happens to also be a member of the same parametric family

$$f(\boldsymbol{x}) = \sum_{k=1}^{p} \theta_k^* B_k(\boldsymbol{x}) \tag{59}$$

(for some $\{\theta_1^*, \cdots, \theta_p^*\}$) then one can show that the solution to (26) (27) for $\lambda = 0$ is unbiased (Gauss-Markov theorem)

$$bias^2 \hat{f}(\boldsymbol{x}|\lambda = 0) = 0.$$

(In fact, it is the unbiased estimator with minimum variance (50)(51) and thus minimum mean-squared-error (56).) In this case, the bias-squared will increase (monotonically) with increasing values of λ (26) (except in the unlikely case that $\{\theta_1^*, \cdots, \theta_p^*\}$ minimize the penalty $\eta(\theta_1, \cdots, \theta_p)$). If the true target function is not a member of the (linear) parametric family (25) (32) characterizing the approximation ((59) is not true) then the bias-squared will also monotonically increase with increasing λ-values but the minimum bias (for $\lambda = 0$) will not be zero

$$bias^2 \hat{f}(\boldsymbol{x}|\lambda = 0) = \underset{g(\boldsymbol{x}) \in \{h(\boldsymbol{x}|\theta_1, \cdots, \theta_p)\}}{\operatorname{argmin}} \int [f(\boldsymbol{x}) - g(\boldsymbol{x})]^2 p(\boldsymbol{x}) d\boldsymbol{x}.$$

This is the distance squared between the target $f(\boldsymbol{x})$ and the closest function in the (restricted) approximating family (25) (32) to it.

The variance decreases with increasing values of the strength parameter λ. In the special case (25) (32) the bias-squared increases with increasing λ-values. The mean-squared-error (56) is sum of these two effects. Thus, in increasing the value of λ one trades increasing bias-squared for decreasing variance. For some value $0 \leq \lambda \leq \infty$ this trade-off will be optimal in the sense of minimizing mean-squared-error. The optimal value will depend upon

the true target function $f(\boldsymbol{x})$, the detailed properties of the randomness of the training data $p(\boldsymbol{x})$ and $F_\epsilon(\varepsilon)$, all of which are unknown, and the training sample size N, and choice of penalty $\varphi[g(\boldsymbol{x})]$ (method) which are both known.

This analysis of the bias-variance trade off presumed that the penalty $\varphi[g(\boldsymbol{x})]$ (20) was chosen independently of the training data, thereby not contributing to the random variation of of the criterion and its solution $\hat{f}(\boldsymbol{x})$. This is not the case for the penalty (40), or any such penalty where the data is used to define the path through the parameter space (37) - (39). In these cases the analysis is a bit more complicated but a similar result obtains. Clearly the results for the variance are the same at the two extremes ($\lambda = \infty$ and $\lambda = 0$). For $\lambda = \infty$ the solution to (26) (27) is given by

$$\hat{f}(\boldsymbol{x}) = h(\boldsymbol{x}|\theta_1^{(0)}, \cdots, \theta_p^{(0)})$$

where $\{\theta_1^{(0)}, \cdots, \theta_p^{(0)}\}$ represents the (user defined) starting point. It thus has zero variance. The solution for $\lambda = 0$ is the (unpenalized) least-squares solution which as maximal variance. At points along the path in-between these extremes the solution is constrained to lie closer to the starting point (as measured along the path) so that its ability to react to changes in the training sample is thereby limited. The larger the value for λ the more the solution is constrained to lie close to the starting point so the variance is smaller. Thus the variance tends to decrease with increasing values of λ.

The behavior of the bias in this case (40) as a function of λ-value can be more complicated. Clearly if the target function is a member of the approximating (linear) parametric family (59) then the bias-squared achieves its minimal value (zero) at $\lambda = 0$. However, even in this case, there is no guarantee that the bias increases monotonically with increasing λ-value or even that it achieves its maximum value for $\lambda = \infty$ (starting point). When the $\lambda = 0$ solution is biased (target not a member of the parametric family) then it may not even be the one corresponding to minimum bias. Therefore, this type of regularization (40) provides variance control through choice of λ-value but very little control over the corresponding bias that is produced in its place.

The bias-variance trade-off characteristic was developed above strictly for the case of linear parameterizations (25) (32). Many popular approximation methods are of this form. For nonlinear parameterizations (25) things become more complicated. For example, even when the target $f(\boldsymbol{x})$ is a member of the same parametric family characterizing the approximations, the bias-squared may be nonzero for $\lambda = 0$. It is even possible for the bias-squared to decrease with increasing λ-values. (The variance always decreases with increasing λ-values.) However, the same general trend, established for the linear case above, seems to obtain in nearly all applications involving nonlinear parameterizations as well. The bias-squared tends to increase as the value of λ is increased (with the exception of (40)). This is due to the

fact that most nonlinear methods used in practice are not too far from being linear so they share many of the characteristics of linear approximations. Considering approximations from parametric families (25) is not a severe restriction since nearly all approaches reduce to (25) however they are implemented. For example, nonparametrically motivated penalties such as (41) (42) involve solutions that are members of parametric families (see Girosi, Jones and Poggio, 1993)

Since the optimal bias-variance trade-off (λ-value) is unknown, it must be estimated from the training data. This is, of course, the same model selection problem discussed in Section 3.4. The reason for discussing it in this framework is to show that model selection can be motivated from non-Bayesian ("frequentist") principles without appeal to prior probability distributions, or empirical Bayesian degree-of-belief ideas. Applying a penalty $\lambda\varphi[g(\boldsymbol{x})]$ (20) to a (least-squares) criterion is known in this setting as the "method-of-regulation" and λ (20) is referred to as the "regularization parameter".

In whatever way model selection is motivated, its degree of success depends on the relationship between the target function $f(\boldsymbol{x})$ and the penalty $\varphi[g(\boldsymbol{x})]$. The more influence the penalty term has, relative to the data term (20), the smaller the variance. The degree-of-influence is determined by the model selection process which, if successful, will allow such influence to increase to the extent that not too much bias is produced. The bias is bounded by the distance-squared (57) between the target and the function that minimizes the penalty (58), or the closest such minimizer if there is more than one. Therefore, the closer the penalty matches the target function in this sense, the more benefit is realized from the regulation process in terms of smaller mean-squared-error.

The ability of the penalty $\varphi[g(\boldsymbol{x})]$ to influence the resulting approximation $\hat{f}(\boldsymbol{x})$ (20) is also related to its specificity, which is (inversely) related to the multiplicity of its minimizing functions (58). Such multiple solution degeneracy can only be resolved by the training data through the least squares term in (20) giving it more influence, thereby reducing the relative influence of the penalty, and thus increasing the variance. For example, a penalty that restricts approximations to be in a parametric family (25) characterized by small number of parameters is generally more specific than one characterized by a larger number, since more (a larger class of) functions minimize the latter. Therefore, for the same value of λ (20), the former penalty will have more influence than the latter on the solution, thereby producing less variance but perhaps more bias. Penalties such as (28) (29) (33) (34) or (36) are the most specific (within a parametric family) since they usually have unique minimizing functions (58).

Regularization produces the greatest accuracy (lowest mean-squared-error) when the penalty is the most specific and most closely matched to the target function (57). If it is highly specific but not well matched then model selection will tend to diminish its influence by choosing a small value of λ

(20) and variance is not significantly reduced. If it is well matched but non specific, then even with large λ-values it has little effect on the approximation again resulting low variance reduction. Knowing how well a chosen penalty $\varphi[g(\boldsymbol{x})]$ matches the target function $f(\boldsymbol{x})$ depends on knowledge about the target function itself. The more knowledge, the more specific a well matched penalty can be made, thereby achieving more accuracy. One thus returns to the Bayesian notion of the relation of the penalty to a priori knowledge about the target (24).

With most approximation methods the penalty $\varphi[g(\boldsymbol{x})]$ is implicitly or explicitly incorporated into the method itself and not directly adjustable. Therefore, the user controls penalty choice through choice of method. It is the goal of the next section to review some of the more popular methods for supervised learning/function approximation from this point of view. That is, what are the nature of the penalties they impose, or equivalently, for what types of target functions will they tend to be most successful. No method dominates all others, or even can be successful, over all situations (target functions).

4.0. Selected methods.

In this section several popular methods for predictive learning / function approximation are reviewed from the perspective of the preceding discussion. This review is incomplete and necessarily brief. The purpose is to highlight their commonality and expose their differences.

All of the methods discussed here can be viewed as "dictionary" methods. Each establishes a set of predefined functions (dictionary)

$$\{b(\mathbf{x}|\gamma)\}_{\gamma} \tag{60}$$

where

$$\gamma = \{\gamma_1, \cdots, \gamma_q\} \tag{61}$$

are a set of parameters that characterize the functions and thus index each member ("entry") of the dictionary. As the joint set of parameters (61) range over all of their allowed values, the entire dictionary of functions (60) is created. The goal is to select a subset of entries characterized by specific sets of parameter values (61)

$$\{b(\mathbf{x}|\gamma_m)\}_{m=1}^{M} \subseteq \{b(\mathbf{x}|\gamma)\}_{\gamma} \tag{62}$$

whose linear combination

$$\hat{f}(\mathbf{x}) = \sum_{m=1}^{M} a_m b(\mathbf{x}|\gamma_m) \tag{63}$$

best (most accurately) approximate the target function $f(\mathbf{x})$. This approximator (63) lies within a subspace of the space of all functions, namely that

subset of functions that can be exactly represented by (63). As the values of $\{a_m, \gamma_m\}_1^M$ range over all of their allowed values, all functions (approximators) in the subspace are generated. The number of entries M that comprise the approximation (63) is either fixed (prespecified) in advance or is taken to be a model selection parameter (Section 3.4).

A dictionary is said to be "complete" with respect to a set (class) of target functions if any target function in that class can be exactly represented by the entire dictionary

$$f(\mathbf{x}) = \sum_{m=1}^{\infty} a_m^* b(\mathbf{x}|\gamma_m) \tag{64}$$

for some set of expansion coefficient $\{a_m^*\}_1^\infty$ values. The corresponding method is said to be a "universal" approximator for that class. The methods reviewed below are all universal approximators for the class of all continuous target functions.

The finite size N of the training sample (7) does not permit the use of the entire dictionary. As entries are added to (62) the approximation (63) is better able to fit the (random) training data and variance is thereby increased. However the bias squared

$$\text{bias}^2\left[\hat{f}(\mathbf{x})\right] = \arg \min_{\{a_m,\gamma_m\}_1^M} \int \left[f(\mathbf{x}) - \sum_{m=1}^{M} a_m b(\mathbf{x}|\gamma_m)\right]^2 p(\mathbf{x})d\mathbf{x} \tag{65}$$

is decreased since adding entries enlarges the function space spanned by the approximation. The approximating subspace can be defined as the set of all functions for which the squared bias (65) is zero. The goal is to choose a dictionary that best matches the target function in the sense that the bias squared (65) is kept low for small M so that the variance is also kept small, thereby producing small mean squared error (56).

For finite M (62) (63), this dictionary approach can be viewed as a penalized method (20) where the penalty functional $\phi[g(\mathbf{x})]$ is taken to be zero for all functions in the subspace spanned by the (basis) functions (62), and infinity for all other functions,

$$\phi[g(\mathbf{x})] = H\{bias^2[g(\mathbf{x})]\} \tag{66}$$

(65), where

$$H(\eta) = \begin{cases} 0 & \text{if } \eta = 0 \\ \infty & \text{otherwise} . \end{cases}$$

Under this constraint (66) the (learning) problem reduces to that of parametric learning (26) (27) with the parameters being

$$\boldsymbol{\theta} = \{\theta_m\}_1^p = \{a_m, \gamma_{1m}, \cdots, \gamma_{qm}\}_1^M . \tag{67}$$

An (additional) parametric penalty $\eta(\theta)$ (26) (if present, $\lambda > 0$) further restricts solutions in the approximation subspace to those functions characterized by parameter values that are favored by that particular penalty. Ridge (28), subset selection (29), weight decay (34), and the gradient descent penalty (40), along with no additional penalty at all ($\lambda = 0$), are among the most popular choices. If a penalty is involved ($\lambda > 0$) the value for λ is (also) determined through model selection (Section 3.4).

The various methods for predictive learning / function approximation described below (along with nearly all others) are primarily differentiated by their (often implied) choice of a particular dictionary (60). Thus, each is especially effective for those target functions that can be accurately approximated by a relatively small number of entries from its dictionary, and less effective when this is not the case. The methods discussed below employ distinctly different dictionaries so that if success is possible ($f(\mathbf{x}) >> \varepsilon$ (4)), there is a chance that at least one of them will be successful. No method is universally superior to all others over all target functions. Each is most appropriate for the particular classes of target functions that most closely match its dictionary in the above sense.

4.1. Projection pursuit and feed-forward neural networks.

These are among the most popular and highly promoted class of methods (especially neural networks). Their dictionaries are comprised of functions of the form

$$\{b(\mathbf{x}|\gamma,\gamma_0) = s(\gamma_0 + \gamma^t\mathbf{x})\}_{\gamma,\gamma_0} \tag{68}$$

where

$$\gamma^t\mathbf{x} = \sum_{j=1}^{n} \gamma_j x_j \tag{69}$$

is a linear combination of the input variables. In the case of neural networks (Rumelhart, Hinton, and Williams, 1986) the single variable ("activation") function $s(z)$, $z = \gamma_0 + \gamma^t\mathbf{x}$ (68), takes the form of a "sigmoid"; that is, it is an S-shaped function that assumes monotonically increasing values between zero and one, as the value of its argument z goes from $z = -\infty$ to $z = \infty$. The most popular choice is the "logistic" function

$$s(z) = \frac{1}{1 + e^{-z}}. \tag{70}$$

The arctangent function

$$s(z) = \frac{1}{\pi}\tan^{-1} z + \frac{1}{2} \tag{71}$$

is also popular. A particular choice (among the many possible S-shaped functions) is seldom crucial.

In the case of projection pursuit (Friedman and Stuetzle, 1981) the single argument function $s(z)$ is given some flexibility by (for example) expressing it in a parameterized form

$$s_\alpha(z) = s(z|\alpha) \tag{72}$$

where $\alpha = \{\alpha_1, \cdots, \alpha_p\}$ are additional parameters that characterize a wide class of (single argument) functions, not necessarily S-shaped. These parameters α are fit to the training data simultaneously with those of the linear combination (γ, γ_0), permitting the individual activation functions to adapt to the target function as well. This approach (72) basically enlarges the dictionary to include a wide variety of activation functions for each different linear combination (69) associated with the dictionary (68). This increased generality is achieved at the expense of introducing additional parameters (72). Therefore, for the same number of entries into the approximation (63) projection pursuit has less bias but more variance than neural networks. Whether this trade-off is beneficial or not will depend on the nature of the target function $f(\mathbf{x})$.

Neural networks as described above (63), (68) – (71), are known as "three-layer" networks. The first layer is comprised of the inputs $\mathbf{x} = \{x_1, \cdots, x_n\}$, the second "hidden" layer is the set of dictionary entries (63) (68), and the output $\hat{f}(\mathbf{x})$ is the third layer. Three layer networks are universal (64) for all continuous target functions, and are most often used in practice. Four layer networks can be constructed by introducing another hidden layer. This is accomplished by using the values of the individual dictionary entries in (63) as inputs to another set of functions taken from the same dictionary, rather than simply taking their linear combination to form the output $\hat{f}(\mathbf{x})$, as in (63). The resulting approximation takes the form

$$\hat{f}(\mathbf{x}) = \sum_{m=1}^{M_2} a_m^{(2)} b(a_{0m}^{(1)} + \sum_{k=1}^{M_1} a_{km}^{(1)} b(\gamma_{0k} + \sum_{j=1}^{n} \gamma_{jk} x_j)) \tag{73}$$

where M_1 and M_2 are respectively the number of entries ("units") in the two hidden layers. This (73) is an expansion of the same form as (63) but with a different dictionary. This dictionary is represented by the collection of functions

$$\left\{ b(\alpha_0 + \sum_{k=1}^{K} a_k b(\gamma_0 + \sum_{j=1}^{n} \gamma_{jk} x_j)) \right\}_{\alpha,\gamma}$$

as the (nonlinear) parameters (α, γ) range over their allowed values. The value of K is usually prespecified.

One can define networks of more than four layers in direct analogy with (73). In all such cases the dictionary is comprised of functions represented by the final hidden layer, as the parameters associated with all previous layers

range over their allowed values. Whether adding such layers improves performance will depend on how well the resulting dictionary matches the target function in the sense discussed above. Adding a layer to an existing network enlarges the dictionary by involving more parameters and thereby increases the variance while reducing the bias. For a given number of parameters (comparable variance) the resulting bias-squared (65) for a particular network architecture, and thus its accuracy (56), will depend on the particular target function encountered.

4.2. Radial basis functions.

The dictionary for this approach is comprised of the functions

$$\left\{r[((\mathbf{x}-\mathbf{t})^t H(\mathbf{x}-\mathbf{t}))^{\frac{1}{2}}]\right\}_{\mathbf{t},H} \tag{74}$$

characterized by the n parameters $\mathbf{t} \in R^n$, and the $\frac{n(n+1)}{2}$ parameters associated with the nonnegative definite matrix $H \in R^{n \times n}$. There are many popular choices for the single-argument functions $r(z)$, the most common being the Gaussian

$$r(z) = e^{-z^2}. \tag{75}$$

This dictionary is very large ($\frac{n(n+3)}{2}$ parameters) and its size is often reduced by placing constraints on the matrix H, thereby reducing the number of parameters. Requiring H to be diagonal

$$H = \mathrm{diag}(h_1, \cdots, h_n) \tag{76}$$

reduces the parameter count to $2n$, whereas requiring it to be a multiple of the $n \times n$ identity matrix

$$H = \rho^2 I_n \tag{77}$$

further reduces it to $n+1$. This (77) is the most popular choice in practice. Parameter count is also often further reduced by constraining all of the selected entries from (74) into (63) to involve the same matrix H.

As with the other methods, there is a bias-variance trade-off associated with the richness of the dictionary (as characterized by the number of parameters), and the number of entries in the expansion M (63). For the same variance, a smaller dictionary (fewer parameters) can include more entries. Whether the associated bias-squared (65) will be less than that of a small expansion from a richer dictionary (similar variance) will depend on the target function $f(\mathbf{x})$.

4.3. Recursive partitioning tree-structured methods.

There are a wide variety of methods of this type. The dictionary for all of them takes the form

$$\{I(\mathbf{x} \in R)\}_R \tag{78}$$

where $I(\eta)$ is an indicator function of the truth of its (logical) argument η

$$I(\eta) = \begin{cases} 1 & \text{if } \eta \text{ is true,} \\ 0 & \text{otherwise .} \end{cases}$$

The symbol R in (78) represents subregions of the space of all possible values of $\mathbf{x} \in R^n$

$$R \subseteq R^n. \tag{79}$$

It is characterized by a set of parameters that define the subregion. For example, if the subregions are constrained to be axis oriented hyper-rectangles (the usual case) one has

$$I(\mathbf{x} \in R) = \prod_{j=1}^{n} I(u_j \leq x_j < v_j) \tag{80}$$

where the $2n$ parameters $\{u_j, v_j\}_1^n$ are the respective lower and upper limits of the region on each input axis.

Recursive partitioning methods (78) nearly always restrict the entries into (63) to represent disjoint regions so that for any set of input values $\mathbf{x}$, only one entry (indicator function (78)) will be nonzero. Thus, in this case (63) can be equivalently expressed as

$$\mathbf{x} \in R_m \Longrightarrow \hat{f}(\mathbf{x}) = a_m \tag{81}$$

where $\{R_m\}_1^M$ represent the chosen entries in (63). This (81) is a piecewise-constant function; all values of $\mathbf{x}$ in the same region R_m have the same approximation value a_m.

Even with the axis-oriented simplification (80) the corresponding optimization problem (20) (66) is combinatorial (rather than numerical) and represents a formidable problem for M (63) not small. For this reason, approximate solutions using greedy optimization strategies are employed based on recursive partitioning. The various approaches associated with methods of this type differ on the particular details of how the partitioning is carried out. Described here is the technique employed by CART (Breiman, et. al., 1984). Most of the other methods use fairly similar strategies. An initial region R_0 consisting of the entire input space is considered first. This region is optimally divided into two regions R_1 and R_2 by a split on one of the input variables v at a split point t. Such a split is defined by:

if $\mathbf{x} \in R_0$ then
    if $x_v \leq t$ then $\mathbf{x} \in R_1$
    else $\mathbf{x} \in R_2$
end if

The values for v and t are jointly chosen so that when the "parent" entry $I(\mathbf{x} \in R_0)$ is replaced by its two "daughters" $I(\mathbf{x} \in R_1)$ and $I(\mathbf{x} \in R_2)$ in

(63) the best fit to the training data is achieved. Each of these newly created daughter regions (R_1 and R_2) is then optimally split in turn by replacing R_0 by R_1 (respectively R_2), and by replacing R_1 and R_2 by their respective daughters, in the above split definition. As each region is successively split, its corresponding entry in (63) is replaced by its two newly created daughter regions.

This recursive application of optimal splitting is continued until a large number of regions are created giving rise to a relatively large expansion (M big) in (63). These regions are then optimally recombined through their unions with other adjacent regions to form a smaller set of (larger) regions to be used in the final approximation. Optimality for this recombination process is based on an estimate of future prediction error using some model selection criterion such as cross-validation (Section 3.4). This strategy of producing a large number of regions which are then recombined to form a smaller optimal subset ("pruning") helps overcome some of the limitations associated with the greedy top-down recursive partitioning approach used for their formation.

The discontinuous piecewise-constant nature of approximations based on recursive partitioning (81) limits their accuracy on continuous (smooth) target functions, especially near the region boundaries. This limitation is mitigated however by other attractive properties of the procedure. One such property is input variable subset selection. Each split chooses the best input variable to divide an existing region (parent) into its two daughter subregions. Thus, input variables that have little or no effect on the variation of the target function are less likely to be chosen for splitting. In this way the resulting approximation tends to include only input variables that are "relevant" to target function variation. The corresponding limits (80) on such nonrelevant variables x_j include their entire range ($u_j = -\infty$, $v_j = \infty$) so that

$$I(-\infty \leq x_j \leq \infty) = 1 \tag{82}$$

for all values of x_j. This effectively removes each such variable from (80) and thus (63). Moreover, this subset selection strategy extends to local regions of the input variable space as well. As the initial (top down) partitioning proceeds, the corresponding regions include smaller and smaller subregions of the input space thereby becoming more and more local. Since the split of any parent region only involves the training data in the subvolume of the input space which it spans, the input variables most relevant in that local subvolume are most likely to be chosen for its split, and the splits of its daughters and further grandchildren. A particular target function $f(\mathbf{x})$ may (globally) have a strong dependence on many of the input variables, but it may depend on only a (different) few of them in different (local) regions of the input space. If so, then this approach can take advantage of this "local relevance" property due to its recursive nature. Approximations derived by the methods discussed above (Sections 4.1 - 4.2) involve all of the input variables everywhere in the

input space and thus are less able to take advantage of this property (if it exists).

The one area where recursive partitioning strategies tend to dominate all others is in interpretability. Owing to the recursive (nested) nature of the partitioning it can be represented graphically as a binary tree. Each node of the tree represents a subregion of the input space. The root node represents the entire space. The two edges connecting an interior node to its two daughter nodes represent the split of that region. The daughter nodes represent the subregions produced by the split of that region. The terminal nodes of the tree represent the regions (63) (80) associated with the final partition. Each interior node is labeled by the splitting variable v and split point t that divided it into its two daughters. The terminal nodes are labeled with the approximation value a_m (81) associated with its region. The boundaries (80) of any terminal region can simply be obtained by traversing the tree from the root node to that region, updating the boundaries with the split points on the splitting variables, at each nonterminal node visited in the traversal. Thus, the binary tree contains all of the information (parameter values) associated with the approximation (63) (80) in a single easily interpreted graphic. [See Breiman et al. (1984) for many such examples.] The binary tree associated with recursive partitioning methods is considered an integral part of the methodology, and such methods are usually referred to as "tree-structured" methods.

4.4. Tensor product methods.

The dictionary for methods of this type are the functions

$$\left\{\prod_{j=1}^{n}\phi(x_j|\gamma_j)\right\}_{(\gamma_1,\cdots,\gamma_n)}. \tag{83}$$

Each of these functions (83) is a (tensor) product of functions of a single variable. Each factor $\phi(x_j|\gamma_j)$ in (83) is a function of one of the inputs x_j and is characterized by one or more parameters γ_j. All implementations of tensor product methods allow the constant function

$$\phi(x_j|\gamma_0) = 1 \tag{84}$$

to appear as a factor associated with one or more of the variables in (83) so that the dictionary can be equivalently expressed as

$$\left\{\prod_{k=1}^{K}\phi(x_{v(k)}|\gamma_k)\right\}_{(K,\{v(k),\gamma_k\}_1^K)} \tag{85}$$

where the number of (nonconstant) factors K, the input variables associated with each of these factor $\{v(k)\}_1^K$, and $\{\gamma_k\}_1^K$, are parameters characterizing each dictionary entry, with $0 \leq K \leq n$ and $1 \leq v(k) \leq n$. By definition $K = 0$ produces the constant function with value always equal to one.

The goal of tensor product methods is generally to favor entries (85) into the approximation (63) with relatively small number of factors K, $K << n$, by judiciously choosing the input variable set $\{v(k)\}_{k=1}^K$ and corresponding parameter values $\{\gamma_k\}_1^K$ associated with each one. This incorporates both the global and local input variable subset selection capabilities associated with recursive partitioning methods (Section 4.3). In fact from (80) one sees that axis oriented recursive partitioning results in tensor product approximations where each factor (85) is an indicator function

$$\phi(x_j|u_j,v_j) = I(u_j \leq x_j < v_j) \tag{86}$$

and $\{v(k)\}_1^K$ are the factors for which $u_j > -\infty$ and $v_j < \infty$ [see (82)].

More refined tensor product methods use a paradigm similar to recursive partitioning but employ continuous and more flexible functions for each of the factors $\phi(x_{v(k)}|\gamma_k)$ appearing in the products (83) (85). This permits more accurate approximation of target functions that are continuous (no sharp approximation discontinuity at the region boundaries), and of a wider class of target functions. For example, multivariate adaptive regression splines ("MARS", Friedman, 1991) uses (two-sided truncated power) spline functions

$$\phi(x_j|s_j,t_j) = [s_j(x_j - t_j)]_+ \tag{87}$$

where the subscript "+" indicates the positive part of the argument,

$$\eta_+ = \max(\eta, 0). \tag{88}$$

The parameter $s_j = \pm 1$ indicates the (left / right) sense of the truncation, and t_j is the "knot" location of the spline.

As with recursive partitioning, optimizing the approximation (63) (85) over all possible parameter values is not computationally feasible. The MARS strategy (Friedman, 1991) approximates this with a forward stepwise procedure (in analogy with recursive partitioning) that attempts to incrementally add entries to (63), and factors (85) (87) to the products that enter, simultaneously. The approximation is initialized with the constant entry

$$b(\mathbf{x}|\gamma_0) = 1. \tag{89}$$

At each successive step (iteration) two additional entries of the form

$$b(\mathbf{x}|\gamma_l)\,\phi(x_v|1,t) \quad \text{and} \quad b(x|\gamma_l)\,\phi(x_v|-1,t) \tag{90}$$

are incorporated into (63). Here (90) the $\phi(x_v|s,t)$ are the spline factors (87) and $b(\mathbf{x}|\gamma_l)$ is one of the entries (products) already in the approximation that was entered at an earlier step. The particular choices for l, v, and t are taken to be the ones that provide the best fit to the training data of the resulting (incremented) approximation. The values of the parameters associated with the selected previously entered term γ_l are not refit. This forward stepwise procedure is iterated until a large number of entries are incorporated into (63). Entries are then selectively deleted in a backward stepwise strategy, to find the best subset of entries, using a cross-validation criterion (49).

The MARS approach reduces to that of recursive partitioning if the factors (86) are used in place of (87) and the "parent" entry $b(\mathbf{x}|\gamma_l)$ is removed from the approximation (63) when its two "daughters" (90) are included. As noted above, using continuous functions (87) allows more accurate approximation of continuous target functions. Not removing the parent increases the generality of the procedure by widening the class of approximators (increasing the size of the dictionary). The MARS approach thereby retains all of the advantages of recursive partitioning while overcoming its principal disadvantages. Although not as interpretable as those of recursive partitioning (binary tree), MARS approximations retain a high degree of interpretability through a technique known as the ANOVA decomposition. See Friedman (1991) for many examples. For another tensor product approach see Breiman (1991).

With the tensor product approach (83) (85) (86) there is no requirement the individual factors $\varphi(x_j|\gamma_j)$ be functions from the same parametric family. Different families could be associated with different input variables x_j. In particular, different parametric families could be employed for input variables of different type (Section 1.1). This enables tensor product methods (including tree-structured recursive partitioning – Section 4.3) to handle categorical input variables, and input variables with missing values, in a much more natural way than other methods [Breiman et al. (1984) and Friedman (1993)]. Methods designed for real valued inputs (projection pursuit, neural networks, radial basis functions, etc.) are generally forced to use "dummy variables" (Section 1.1) to represent categorically valued inputs, thereby reducing their generality and effectiveness when such variables are present. Missing valued inputs also present more difficulty for these methods than for those based on tensor products.

4.5 Discussion.

Five methods (projection pursuit, feed-forward neural networks, radial basis functions, recursive partitioning decision trees, and tensor product methods) were presented (Sections 4.1 – 4.4). The first two (projection pursuit and neural networks) shared some similar properties whereas the last three were fairly different from the first two and from each other. They were primarily discussed and compared in terms of the types (properties) of the target

functions for which each is most appropriate in terms of the dictionaries they employ. As pointed out repeatedly, none of these methods dominate any of the others, over all possible target function, in terms of accuracy. These methods were also compared in terms of the interpretability of (the structure of) the approximations each produces. Some were seen to be much more interpretable than others.

There are other properties associated with approximation methods that, while very important, were not emphasized here. These properties include training speed (computation), and ease of use. The latter property involves the expertise and experience with the method required on the part of the user to make it perform to its potential in situations (target functions) for which it is most appropriate. Specific implementations of each of these methods involve procedural ("tuning") parameters, that can be adjusted by a user, to increase performance in a particular situation. Judicious choices for the values of such parameters can often greatly improve accuracy whereas poor choices can severely degrade performance. Thus a particular technique can be quite powerful (in a particular situation) in the hands of an expert with that method, and ineffectual when used by a novice. There is no way to separate the power of a method from the expertise of the person applying it. Some methods, however, are less sensitive than others to variations in the values of their tuning parameters and thus more robust against suboptimal choices. For such robust methods, default tuning parameter values can often be provided that provide acceptable performance over a wide variety of situations. In this context, the tree structured methods (Section 4.3), adaptive splines (Section 4.4), and to a somewhat lesser extent, projection pursuit (Section 4.1) are the more robust of the methods discussed here. This is likely due to the fact that these methods were originally developed in the field of statistics where ease of use (and interpretability) are often regarded as being equally important as accuracy.

The inability to separate the performance of a method from the expertise of the person that applies it complicates the interpretation of empirical performance comparisons between methods. The author(s) performing a comparison may have differential skills in their ability (and perhaps motivation) to tune the respective methods being compared. This can especially be the case when the purpose of the comparison is to validate a particular procedure, often one developed by the author(s). The developer of a particular method generally has the most skill (and motivation) to tune it well, in comparison to the other procedures, and this should be taken into account when interpreting the corresponding results. In addition, when a comparison (contest) involves data from a specific field (domain of application) then the relative domain specific knowledge of the user can greatly affect the outcome. Clever use of such knowledge can often be far more important in determining the outcome (approximation accuracy), than choice of a particular approximation method used in the context of such knowledge. Purely methodological

comparisons are most easily interpreted when the same person(s) apply all of the respective methods to the same sets of data and have no differential skills among the methods, nor preferential interest in the outcome.

The computational resources required to obtain a solution approximation (training speed) can be an important consideration, especially when there are (multiple) tuning parameters to be adjusted. Exploration to obtain a good set of values may require repeated training runs using different values in each one. Restricted computational resources can limit the number of such runs, and thereby the potential accuracy of the result. Again the methods originating from statistics (recursive partitioning, adaptive splines, and to a lesser extent projection pursuit) tend to allow (much) faster training owing to the (perhaps excessive) regard in that field for rapid computability.

The methods discussed here (Sections 4.1 - 4.4) far from exhaust all of those that have been proposed for function approximation (supervised learning). Although these are among the more popular, there exist a plethora of other competitive proposals in the respective literatures that study this problem. Many of these can be regarded as variations on the methods discussed here. For example, most methods that employ interpoint distances in the input variable space bear a strong similarity to radial basis functions (Section 4.2). There are also a great many adaptations of both the feed-forward neural network (Section 4.1), and tree-structured (Section 4.3) paradigms. As noted, a few variations on the tensor product approach have also been proposed. The goal here was to choose a particular method from each distinctive class of methods as being representative of its class in terms of the types of target functions for which that class of methods is appropriate, as dictated by its dictionary.

5.0 Multiple outputs.

All of the discussion so far has concerned the approximation of a single target function $f(\boldsymbol{x})$ (4) from a training sample (7). Most applications of supervised learning only involve a single output quantity of interest. Sometimes however the system under study produces several output variables (2), such as the quality control example mentioned in Section 1.0. The problem then becomes one of approximating several target functions $\{f_k(\boldsymbol{x})\}_1^K$ (2) of the same set of input variables $\boldsymbol{x}$, using the same input training data $\{\boldsymbol{x}_i\}_1^N$ with different outputs $\{y_{i1}\}_1^N, \cdots, \{y_{iK}\}_1^N$.

Multiple outputs ($K > 1$) (2) do not fundamentally change the approximation problem. One could approach this in a straightforward way by simply treating the approximation of each separate target function $f_k(\boldsymbol{x})$ ($1 \leq k \leq K$) as a different (unrelated) problem. With this strategy one would develop a separate approximation $\hat{f}_k(\boldsymbol{x})$ for each output using separate training sets $\{y_{i1}, \boldsymbol{x}_i\}_1^N, \cdots, \{y_{iK}, \boldsymbol{x}_i\}_1^N$, without regard to the commonality of their input variables $\boldsymbol{x}$, nor to the fact the same system gives rise to all of them. If the respective target functions $\{f_k(\boldsymbol{x})\}_1^K$ have very different prop-

erties in terms of their dependencies on the inputs $\boldsymbol{x}$ then this is the best strategy. In fact, one might even envision using different methods for approximating different targets in this case, if one's a priori knowledge concerning the respective target functions suggested that this might be appropriate.

One's priori knowledge might however indicate that all of the target functions should have a similar dependence on the input variables. This would dictate that the same (appropriate) method be used for approximating all of them. In addition, if the target functions were sufficiently similar, there may be gains to be made by limiting the dissimilarity of the resulting approximations $\{\hat{f}_k(\boldsymbol{x})\}_1^K$, perhaps by imposing a penalty for the degree of such dissimilarity. As with all such constraining penalties, bias (53) (56b) will tend to be increased and the variance (50) (51) will decrease. If there is sufficient similarity among the target functions, the increase in bias may be small enough so that the resulting decrease in variance gives rise to smaller mean-squared error (54) (56c). If not, mean-squared error will increase and the exercise will be counter productive. One might even envision associating a strength parameter λ with such a dissimilarity penalty and using (an additional level of) model selection to try to estimate its value. There has been however very little work in this area and the development of such approaches remains an open research topic.

Another motivation for attempting to connect or combine the approximations of different target functions, associated with the same inputs, is computational. If the method being employed is sufficiently computationally intensive (e.g. feed-forward neural networks or adaptive radial basis functions) there may not be sufficient computing resources available to separately approximate several target functions in a reasonable time. With most (nonlinear) function approximation methods the dominant part of the computation is concerned with minimizing the objective criterion (26) with respect to the nonlinear parameters $\{\gamma_m\}_1^M$ (63) associated with choosing each of the dictionary entries $\{b(\boldsymbol{x}|\gamma_m)\}_1^M$ (62). This suggests that considerable computation can be saved in the multiple output problem by sharing a single set of dictionary entries among all of the corresponding approximations,

$$\hat{f}_k(\boldsymbol{x}) = \sum_{m=1}^{M} \hat{a}_{km} b(\boldsymbol{x}|\hat{\gamma}_m) \tag{91}$$

Here (91) each target function $f_k(\boldsymbol{x})$ is approximated by (different) linear combinations of the same set of dictionary entries. All of the (linear) expansion coefficient $\{\hat{a}_{km}\}_{m=1,k=1}^{M,K}$ and the nonlinear parameters $\{\hat{\gamma}_m\}_1^M$ associated with the (common) set of dictionary entries are obtained by

$$(\hat{A}, \hat{\Gamma}) = \underset{A,\Gamma}{\operatorname{argmin}} \sum_{k=1}^{K} \sum_{i=1}^{N} [y_{ki} - \sum_{m=1}^{M} a_{km} b(\boldsymbol{x}_i|\gamma_m)]^2 + \lambda \cdot \eta(A, \Gamma). \tag{92}$$

Here (92) matrix notation is used to represent the respective collections of parameters

$$A = [a_{km}],\ \Gamma = (\gamma_1, \cdots, \gamma_M). \tag{93}$$

This (91) (92) constrains all the approximations $\{\hat{f}_k(\boldsymbol{x})\}_1^K$ to lie in the same function subspace spanned by the (commonly) chosen dictionary functions. Implementations of projection pursuit (Friedman, 1985), feed-forward neural networks (Rumelhart, Hinton, and Williams, 1986), recursive partitioning (CART-Breiman, Friedman, Olshen, and Stone, 1984), and MARS (Hastie, Tibshirani, Buja, 1992) all use this approach (91–93) for multiple outputs. The value of the single (common) strength parameter λ is chosen by optimizing a common collective model selection criterion. For example, this analog for cross-validation (49) would be

$$\hat{\lambda} = \underset{\lambda}{\operatorname{argmin}} \sum_{k=1}^{K} \sum_{i=1}^{N} [y_{ki} - \hat{f}_{k\backslash i}(\boldsymbol{x}_i|\lambda)]^2, \tag{94}$$

Here (94), $\hat{f}_{k\backslash i}(\boldsymbol{x}|\lambda)$ is the solution to (91–93) for a given value of λ, with the ith training observation removed.

In addition to reducing computation, this approach to multiple outputs imposes a similarity constraint on the respective approximations (91); they are all expansions in the same set of dictionary entries and they are all regularized by the same value for the strength parameter $\hat{\lambda}$ (94). For example, if subset selection (29) were used, the value of $\hat{\lambda}$ would regulate the (same) number M of (identical) dictionary entries appearing in each approximation (91). If the respective target functions $\{f_k(\boldsymbol{x})\}_1^K$ happen to exhibit a great deal of common structure then this may be a virtue (as discussed above) by reducing the variance component (50) of the approximation mean-squared error (54) (56c), for each approximation (91). This is however by no means assured since the (compromise) value of λ chosen by (94) can be considerably larger that[1 n some (or all) of the separate strength parameters associated with completely separate (unrelated) approximations $\{\hat{f}_k(\boldsymbol{x})\}_1^K$.

The imposition of such a similarity constraint is also often used to (heuristically) motivate this strategy for multiple outputs, by arguing that it "takes advantage" of possible similarities among the target functions associated with the respective outputs. Although these motivations are plausible, this procedure has serious flaws that can cause considerable degradation of accuracy as compared to independently approximating each target function separately. First, (91) (92) is a "natural" generalization of the single output case (20) only in a very restrictive setting. Even when the respective errors $\boldsymbol{\varepsilon} = \{\varepsilon_1, \cdots, \varepsilon_K\}$ (2) are assumed to have a (joint) Gaussian distribution

$$\Pr[\varepsilon_1, \cdots, \varepsilon_K|\boldsymbol{x}] \sim e^{-\frac{1}{2}\boldsymbol{\varepsilon}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\varepsilon}} \tag{95}$$

with $(K \times K)$ covariance matrix Σ, the multiple output generalization of the maximum a posteriori probability (MAP) criterion (23) (24) reduces to (92) (assuming (91)) only if Σ is a multiple of the identity matrix

$$\Sigma = \sigma^2 I_K. \tag{96}$$

This (96) requires all of the respective response errors ε_k (2) to have the same variance and to be completely uncorrelated with each other. This is not likely to be (even approximately) true in general. The correct multiple output generalization of the MAP procedure under these assumptions (91) (95) and Σ known is

$$(\hat{A}, \hat{\Gamma}) = \underset{A,\Gamma}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - \hat{f}(x_i))^T \Sigma^{-1} (y_i - \hat{f}(x_i)) + \lambda \cdot \eta(A, \Gamma) \tag{97}$$

where vector notation is used to represent the output variables and their corresponding approximations in the K-dimensional output space

$$y = \{y_k\}_1^K \text{ and } \hat{f}(x) = \{\hat{f}_k(x)\}_1^K \tag{98}$$

with $\hat{f}_k(x)$ given by (91). If Σ (95) is unknown (the usual case) it can be estimated by

$$\hat{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{f}(x_i))\,(y_i - \hat{f}(x_i))^T \tag{99}$$

and this estimate (99) substituted in (97). However this estimate (99) depends on the parameters (A, Γ) through (91) so that the process would have to be iterated. Starting with an initial guess for Σ (say $\Sigma = I_K$) the solution to (97) is found. The resulting estimates (91) are then used to re-estimate Σ (99) for the next iteration, and so on. This iterative procedure is equivalent (Bates and Watts, 1988) to solving

$$(\hat{A}, \hat{\Gamma}) = \underset{A,\Gamma}{\operatorname{argmin}} \log \; det \; \hat{\Sigma}(A, \Gamma) + \lambda \cdot \eta(A, \Gamma) \tag{100}$$

where $\hat{\Sigma}(A, \Gamma)$ is a function of the parameters through (91) (99). Using (99) (100) in place of (92) would overcome the limitation (96) and probably produce superior results in most cases. Of course this (99) (100) is a computationally much more complex criterion than (92). For example, all of the fast updating tricks used to speed up least-squares minimization (92) do not work for (99) (100). Simple numerical optimization methods such as steepest descent ("back propagation") can still be applied (as is routine with feed-forward neural networks using (92)) with this criterion (99) (100), but there may be more of a problem with local (suboptimal) minima owing to its increased complexity.

Examination of (97) (99) elucidates the major problem associated with using the usual approach (92) to multiple outputs. By the inclusion of $\hat{\Sigma}^{-1}$ (99) the proper criterion (97) gives more relative weight to those target functions that can be most accurately approximated by the chosen method. This may be due to a smaller associated (irreducible error) ε_k (2) or to the fact that the method is better matched to those particular target functions (57) (58). This relative weighting gives each target function its proper influence in determining the parameters associated with the (common) dictionary entries. Using (92), by contrast, allows those targets that are difficult to approximate to overly influence (and perhaps dominate) entry (parameter) selection. This can (and often does) cause severe degradation in (relative) approximation accuracy for those outputs for which high accuracy is possible, in exchange for modest (relative) gain for those that cannot be reasonably approximated under any circumstance. This effect is aggravated further by the use of an (unweighted) model selection criterion, such as (94), to estimate the common strength parameter value $\hat{\lambda}$. Here again the poorly approximated target functions will dominate the model selection giving rise to possibly highly suboptimal λ-values for those target functions that could have been accurately approximated separately.

If one had a priori knowledge of which target functions are the ones most likely to be accurately approximated, the criterion (92) could be modified so as to give more influence to the corresponding outputs. This knowledge could be incorporated into a conjecture for an appropriate matrix Σ (97). For example, setting it to the diagonal matrix

$$\Sigma = diag(\sigma_1^2, \cdots, \sigma_K^2)$$

would assign a relative influence (weight) of $1/\sigma_k^2$ to each corresponding output y_k in determining the common set of dictionary entries (91). The model selection criterion (94) could be similarly weighted to emphasize the importance of the outputs in estimating the value $\hat{\lambda}$ of the (common) strength parameter. However, such a priori knowledge is seldom available so that (92) (94) is usually the default choice.

Even if the proper criterion (100) is utilized, and the model selection criterion (94) is appropriately modified, one still has the restrictions of a common set of dictionary entries (91), and a common strength parameter value λ, associated with this multiple output strategy. It is easy to imagine a great many situations where the target functions $\{f_k(\boldsymbol{x})\}_1^K$ (2) are sufficiently different so that these restrictions impede performance, perhaps on all of them, compared to separate independent approximations.

Although the multiple output strategy characterized by (91) (92) (94) is the one most commonly advocated, the above discussion indicates that it should be used with caution. If sufficient computing resources are available a more robust strategy is to treat the approximation of each target function $f_k(\boldsymbol{x})$, associated with each output y_k (2), as a separate independent problem.

Only when all the respective target functions $f_k(\boldsymbol{x})$, and the corresponding noise ϵ_k (2), have highly similar structure will the collective strategy (91) (92) (94) produce superior results. The degree of improvement in such favorable settings however is not likely to be sufficient to overcome the risk associated with the (possibly) severe loss in accuracy incurred in settings unfavorable to this approach. Alternative methods for dealing with multiple outputs are currently under investigation (Breiman and Friedman, 1994). These can take advantage of target function similarity (irrespective of the noise structure) to improve accuracy when such similarity exists, but do not sacrifice performance (compared to separate independent approximations) when it does not.

6.0. Classification (categorical outputs)

As noted in Section 1.1, situations in which the output variable takes on unorderable categorical values constitute an important (often occurring) class of problems. These types of problems dominate in the fields of engineering, pattern recognition, and machine learning. The categorical output value can be viewed as a label that classifies an object (observation) i, characterized by input variable values $\boldsymbol{x}_i$, to one of K groups. Each group is identified by one of the K (categorical) values y can assume

$$y \in \{a_1, \cdots, a_K\}. \tag{101}$$

Labeling these groups as $G_1, \cdots, G_K$ we can define classification group (class) membership as

$$y_i = a_k \Rightarrow i \in G_k. \tag{102}$$

Since we assume (define) that an output y_i must take on one (and only one) of its categorical values (101) for each observation, the assignment rule (102) implies that object i must belong to one (and only one) of the groups (classes) in $\{G_k\}_1^K$. The classification problem can thus be viewed as deriving a procedure (rule) for assigning each observation to its correct group or class. Supervised learning in this setting consists of using a training sample (7) whose correct classifications $\{y_i = a_k\}_1^N$ (101) are known, to obtain a good classification (prediction) rule $r(\boldsymbol{x})$ for assigning future observations (of unknown class) to their correct groups (102), given only their input values $\boldsymbol{x}$.

In order to derive a good classification rule degree-of-goodness (accuracy) must be defined. Since the output variable is categorically valued, error (lack-of-accuracy) measures like (8a), (8b), or (10) have no meaning, since (continuous) distances (differences) do not exist between categorical values; two categorical values are either equal, or they are not equal. A natural measure of error would therefore be the fraction of times that the prediction rule assigns (future) objects to an incorrect group

$$\Delta[r(\boldsymbol{x})] = \frac{1}{N_T} \sum_{i=1}^{N_T} I(r(\boldsymbol{x}_i) \neq y_i). \tag{103}$$

Here (103), $\{y_i, \boldsymbol{x}_i\}_1^{N_T}$ are a (test) sample of observations not used (in any way) in the training of $r(\boldsymbol{x})$.

Just as with real valued outputs (4), a given set of measured input variable values $\boldsymbol{x}$ may not uniquely specify a categorical output value. There may be other unobserved inputs $\{z_j\}$ (1) whose values affect the output (class membership), but are not in any way controlled. Therefore specifying a particular set of input values $\boldsymbol{x}$ determines a probability distribution of the output values

$$\{\Pr(y = a_k|\boldsymbol{x})\}_1^K \tag{104}$$

induced by the varying values of the unobserved inputs $\{z_j\}$ (1). In the case of a real valued output this effect is accounted for by the statistical model (4). Introducing the probability distribution (104) for the value of y, at each set of input values $\boldsymbol{x}$, represents the corresponding statistical model for a categorical output. As with all probability distributions, one has

$$\sum_{k=1}^{K} \Pr(y = a_k|\boldsymbol{x}) = 1 \tag{105}$$

for all values of $\boldsymbol{x}$. At each $\boldsymbol{x}$, the (categorical) output takes on a random value according to a multinomial distribution with respective probabilities given by (104).

At each $\boldsymbol{x}$-value, the derived decision rule $r(\boldsymbol{x})$ assigns a unique predicted (categorical) value. The probability that this prediction will be incorrect is the sum of the probabilities (104) for those categorical values (101) not equal to the predicted value

$$\Pr[y \neq r(\boldsymbol{x})] = \sum_{k=1}^{K} I(r(\boldsymbol{x}) \neq a_k) \Pr(y = a_k|\boldsymbol{x}) \tag{106}$$

so that the population ($N_T \to \infty$) analog of (103) is

$$\Delta[r(\boldsymbol{x})] = \int [\sum_{k=1}^{K} I(r(\boldsymbol{x}) \neq a_k) \Pr(y = a_k|\boldsymbol{x})] p(\boldsymbol{x}) d\boldsymbol{x}. \tag{107}$$

Thus (107) is the analogous error measure, for categorical outputs, to (9) for real valued outputs.

The error measures (103) (107) assume that all misclassification errors $y \neq r(\boldsymbol{x})$ are equally costly, irregardless of the values of y and $r(\boldsymbol{x})$. This is not always the case. For example, in medical diagnosis the cost (or "loss") associated with misclassifying someone as being well, when they are actually severely ill, is generally greater than the converse misclassification. This can be accommodated by specificing a "loss"

$$L(r(\boldsymbol{x}), y) \tag{108}$$

associated with assigning the (categorical) value $r(\boldsymbol{x})$ when the true value is actually y. By convention the loss (108) associated with a correct assignment is defined to be zero, and to be positive for incorrect assignment. The expected (average) loss of assigning $r(\boldsymbol{x})$ to an object with input values $\boldsymbol{x}$ is

$$R(\boldsymbol{x}) = \sum_{k=1}^{K} L(r(\boldsymbol{x}), a_k) \Pr(y = a_k|\boldsymbol{x}). \tag{109}$$

Thus (109) is called the misclassification "risk" at $\boldsymbol{x}$, and this quantity averaged over all $\boldsymbol{x}$-values

$$R = \int [\sum_{k=1}^{K} L(r(\boldsymbol{x}), a_k) \Pr(y = a_k|\boldsymbol{x})] p(\boldsymbol{x}) d\boldsymbol{x} \tag{110}$$

becomes the (generalized) lack-of-accuracy measure analogous to (107). If all assignment errors are deemed to be equally costly then this is specified by

$$L(r(\boldsymbol{x}), a_k) = I(r(\boldsymbol{x}) \neq a_k) \tag{111}$$

and (110) reduces to (107), which simply counts the proportion of incorrect assignments. In any case the loss matrix (108) is dictated by the nature of the problem and specified by the user.

If the respective conditional probabilities (104) were know for all values of $\boldsymbol{x}$ then (109) (110) is minimized by the classification assignment rule

$$r^*(\boldsymbol{x}) = \underset{r}{\operatorname{argmin}} \sum_{k=1}^{K} L(r, a_k) \Pr(y = a_k|\boldsymbol{x}). \tag{112}$$

This (112) is known as the "Bayes" rule and it achieves the lowest value of misclassification risk (109) (110) among all possible assignment rules for the particular problem at hand. The corresponding minimizing (average) risk R^* is obtained by substituting $r^*(\boldsymbol{x})$ (112) for $r(\boldsymbol{x})$ in (110), and is called the "Bayes risk" for that problem. The Bayes risk characterizes the irreducible error associated with the problem in analogy with the irreducible mean-squared prediction error (9)

$$\int \sigma^2(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}.$$

for real valued outputs.

Knowing the true class probabilities (104) (for all $\boldsymbol{x}$) is analogous to knowing the true target function $f(\boldsymbol{x})$ (4) in the real valued output case. In practice they are never known and it is the goal of supervised learning to obtain approximations (estimates) for them using the training data (7). One

way to do this is to use the "dummy variable" technique described in Section 1.1. the K-valued categorical output variable y is mapped to K real valued variables $\{d_1, \cdots, d_k\}$ each one of which assumes only two values ($d_k = 0$ or $d_k = 1$) through the definition

$$d_k = \begin{cases} 1 & \text{if } y = a_k \\ 0 & \text{otherwise.} \end{cases} \tag{113}$$

For a given categorical value $y = a_k$ all of the dummy variables but one (d_k) have the value zero, and

$$\sum_{k=1}^{k} d_k = 1. \tag{114}$$

The collective values $\{d_k\}_1^K$ (113) clearly contain the same information as the single (categorical value) of y (101), and predicting their values for given input $\boldsymbol{x}$ is equivalent to predicting the value of y. The linear degeneracy (114) implies that it is sufficient to predict only $K - 1$ of them. The value of introducing the (real-valued) dummy variables (113) stems from the relation

$$\Pr(y = a_k|\boldsymbol{x}) = \Pr(d_k = 1|\boldsymbol{x}) = E(d_k|\boldsymbol{x}). \tag{115}$$

The right most quantity in (115) has the defining form (6) of a real-valued target function $f_k(\boldsymbol{x})$ based on a real valued output d_k,

$$\{f_k(\boldsymbol{x}) = E(d_k|\boldsymbol{x})\}_1^K. \tag{116}$$

If these target functions (116) were known they could be used (115) to derive the optimal (Bayes) prediction rule (112). Since they are unknown, supervised learning is used to obtain approximations

$$\{\hat{f}_k(\boldsymbol{x})\}_1^K \tag{117}$$

for them using training data (7) (113). Thus, the supervised learning problem for a (single) K-valued categorical output y (101) is equivalent to the corresponding problem of (multiple) function approximation (116) (117) involving K real-valued outputs (113). The training data for obtaining each approximation (117) is

$$\{d_{ki}, \boldsymbol{x}_i\}_1^N, \quad k = 1, K, \tag{118}$$

with d_{ki} derived from (categorical) y_i (7) through (113).

The dummy variable trick converts a classification problem (102) to that of multiple output function approximation (Section 5.0). All of the function approximation methods discussed herein (and elsewhere) are eligible candidates to be used for this purpose. The one that is most appropriate for any

given problem will depend (as always) on the (properties of) the (true) target functions (116) encountered.

The multiple output problem associated with classification (116) (117) has special characteristics that are not usually present in the general problem of multiple outputs (Section 5.0). These stem from the constraints

$$\{0 \leq f_k(\boldsymbol{x}) \leq 1\}_1^K \text{ and } \sum_{k=1}^K f_k(\boldsymbol{x}) = 1 \tag{119}$$

on the true target functions (116). It is reasonable to expect that imposing the corresponding constraints (119) on the resulting approximations (117) should improve accuracy since information (outside the training data) is being incorporated, that is known to be true from the nature of the problem. Some function approximation methods (117) when used in the classification context (116) automatically produce estimates (117) that satisfy (119). An example of this is recursive partitioning, using a common set of dictionary entries (regions) for each (dummy) output (113) (116). Most methods however do not produce approximations (117) that satisfy these constraints (119) when applied in this context.

The second constraint in (119) can be met by replacing each derived approximation by

$$\hat{h}_k(\boldsymbol{x}) = \hat{f}_k(\boldsymbol{x}) / \sum_{l=1}^K \hat{f}_l(\boldsymbol{x}) \tag{120}$$

where $\{\hat{f}_k(\boldsymbol{x})\}_1^K$ are the (unconstrained) approximations produced by the chosen method. The first constraint in (119) is more difficult to enforce. One proposal (Denker and LeCun, 1991) is to make the replacement

$$h_k(\boldsymbol{x}) = \frac{e^{g_k(\boldsymbol{x})}}{\sum_{l=1}^K e^{g_l(\boldsymbol{x})}} \tag{121}$$

during training (20). That is,

$$\{\hat{g}_k(\boldsymbol{x})\}_1^K = \underset{\{g_k(\boldsymbol{x})\}_1^K}{\text{argmin}} \sum_{k=1}^K \sum_{i=1}^N [d_{ki} - \frac{e^{g_k(\boldsymbol{x})}}{\sum_{l=1}^K e^{g_l(\boldsymbol{x})}}]^2 + \lambda \cdot \varphi(\{g_k(\boldsymbol{x})\}_1^K), \tag{122}$$

$$\{\hat{f}_k(\boldsymbol{x}) = e^{\hat{g}_k(\boldsymbol{x})} / \sum_{l=1}^K e^{\hat{g}_l(\boldsymbol{x})}\}_1^K. \tag{123}$$

The penalty in (122) is dictated by the function approximation method used and specifies the form of each $\hat{g}_k(\boldsymbol{x})$, that is the particular set of dictionary entries (60) (62). This approach (122) (123) is called the "softmax" method.

It (usually) requires the use of a common basis function set (91), with common regularization (94). Also fast least-squares optimization tricks used by some of the function approximation methods are not valid for (122), so that relatively slow optimization methods (such as steepest-descent) must be used. Even when this is done the minimization of (122) is often more difficult than its function approximation analog (20).

Requiring the constraints (119) on the derived approximations (117) is generally not necessary and in fact can be counter productive if their enforcement (say by softmax (122)) leads to implementation difficulties. First, conforming to these constraints (119) is neither a necessary nor sufficient condition for accuracy. A set of approximations (117) that violate them may in fact be more accurate than another set that obeys them. Second, the goal of the classification exercise is an accurate decision rule

$$\hat{r}(\boldsymbol{x}) = \underset{r}{\operatorname{argmin}} \sum_{k=1}^{K} L(r, a_k)\hat{f}_k(\boldsymbol{x}) \tag{124}$$

based on the approximators (117), and not necessarily the accurate approximation of the target functions (116) themselves (10). It is true that if the approximators are able to achieve perfect accuracy ($mse = 0$ (10)), then the resulting rule (124) will achieve the minimum (Bayes) risk (110) (112). This need not imply however that an imperfect set of approximations (117) ($mse > 0$ (10)) necessarily produce a more accurate decision rule (124) than an even less perfect set (greater mse (10)).

This lack of a direct (one-of-one) connection between approximation error (10) on the target functions (116) and misclassification risk

$$\hat{R} = \int \hat{r}(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} \tag{125}$$

of the resulting decision rule (124) can be easily illustrated by considering the simple loss function (111). In this case the Bayes (minimum misclassification rate) rule (112) (115) (116) becomes

$$r^*(\boldsymbol{x}) = a_{k^*(\boldsymbol{x})}, \tag{126}$$

$$k^*(\boldsymbol{x}) = \underset{1 \le k \le K}{\operatorname{argmax}} f_k(\boldsymbol{x}),$$

where $a_{k^*(\boldsymbol{x})}$ is one of the categorical output values (101). The corresponding rule derived from the training data is

$$\hat{r}(\boldsymbol{x}) = a_{\hat{k}(\boldsymbol{x})}, \tag{127}$$

$$\hat{k}(\boldsymbol{x}) = \underset{1 \le k \le K}{\operatorname{argmax}} \hat{f}_k(\boldsymbol{x}).$$

Comparing (126) and (127) one sees that the only requirement for obtaining the optimal decision at $\boldsymbol{x}$ is that the approximation, corresponding to the target function (116) with greatest value there, be the largest valued among all the respective approximations (117) at $\boldsymbol{x}$; that is $k^*(\boldsymbol{x}) = \hat{k}(\boldsymbol{x})$. The actual values of all of the approximations (117) could be very far from their respective targets (116), and need not even have the same ordering. As long as the right one has the largest (approximated) value, the optimal decision will be made. Although illustrated here (126) (127) for the simple loss function (111), a similar effect clearly obtains for more general loss structures (108).

The fact that function approximation accuracy (10) does not necessarily directly translate into classification accuracy (125) (and conversely) implies that in any given situation the best method for approximating the target probabilities (115) (116) may not necessarily be the best one for use in constructing a decision rule (124). Sometimes the probabilities (115) (116) are themselves of direct interest and other times only a decision rule is required. Therefore, even in the same situation different methods may be appropriate for the two different goals. Also, certain methods that have generally limited success in function approximation (10) can be quite effective when used in the classification context solely to construct a decision rule (124). An example of this is tree-structured recursive partitioning methods (Section 4.3) and nearest neighbor methods (Duda and Hart, 1973) not discussed here.

Owing to the discontinuous piecewise-constant nature of their approximations, recursive partitioning is seldom competitive with other methods (in terms of accuracy) for function approximation. Even though it tends to provide inaccurate estimates of the target probabilities (115) (116), it generally has better success at estimating which one is the largest (127) at a given point $\boldsymbol{x}$ in the input space. Therefore, recursive partitioning methods tend to be more competitive when used for classification than in the function (or probability) approximation context, particularly when the simple loss structure (111) is used. The same tends to be true for nearest neighbor techniques.

The dummy variable approach (113) (115) (116) (117) (124) provides a connection between the classification problem (101) (102) and (real valued) function approximation. It thereby allows all of the methodology developed for function approximation to be brought to bear on the classification problem. With this dummy variable – function approximation approach, the construction of the decision rule (124) can be considered a "post-process" applied to the approximations (117) after they are obtained from the function approximation procedure that was used. Generally the procedure attempts to obtain accurate approximations using a criterion like (9) (10) to assess accuracy, without regard to the fact that the ultimate goal is classification accuracy of the resulting decision rule (124). Unfortunately, directly using misclassification risk (110) (124) as a criterion for (joint) function approximation is generally not feasible since unlike (20) it does not vary continuously as the approximations (117) change. As was noted above, the approximat-

ing functions (117) can change (dramatically) without effecting the decision rule (124) (127) (and thus misclassification risk) at all. This rules out the use of numerical optimization methods for training, requiring instead more costly combinatorial optimization techniques if misclassification risk were to be used directly as the criterion to be minimized.

Another way to try to improve the use of function approximation on classification problems is to implement more elaborate "post-processors" than simply directly inserting the approximations (117) into (124) to construct the decision rule. The idea is to regard the function approximation part of the combined procedure as a "pre-processor" for transforming the original input variables $\boldsymbol{x} = \{x_j\}_1^n$ to a new set

$$\boldsymbol{z} = \{z_k = \hat{f}_k(\boldsymbol{x})\}_1^K \tag{128}$$

(117) which are then used as inputs to a (traditional) classification procedure. Nearest neighbor classification (Lippmann, 1989) and linear discriminant analysis (Hastie, Tibshirani and Buja, 1992), and (Ripley, 1994), have been suggested for this latter classification on the transformed inputs $\boldsymbol{z}$ (128). An attractive feature of this approach is that the constraints (119) are irrelevant to the function approximation preprocess since they are automatically enforced by the post classification part of the combined procedure. The approximating functions (117) are simply regarded as new "features" (128) and the function approximation preprocess is viewed as an (automatic) feature selection mechanism for the (post) classifier employed.

To the extent that the approximations (117) accurately represent their corresponding targets (115) (116) they ought to provide a good set of features (128) for separating the individual classes (102). Using the true target functions (115) (116) for the transformations (128) would cause the n-dimensional input space to be mapped onto a $K-1$ dimensional simplex imbedded in the K-dimensional Euclidean space of the transformed variables (probabilities) $\boldsymbol{z}$. The K vertices of this simplex are points in the positive direction along each z_k-axis at unit distance from the origin. Each such vertex can be identified with one of the classes $y = a_k$ (102) since it corresponds to the point for which the probability for that class (115) (116) is one, and those for all the other classes $y \neq a_k$ are zero. The optimal "post-process" in this case (Bayes decision rule (126) with equal losses (111)) would be to assign each point $\boldsymbol{z}$ to the class associated with the vertex closest to it in Euclidean distance

$$k^*(\boldsymbol{z}) = \underset{1 \leq k \leq K}{\operatorname{argmin}} \sum_{l=1}^{K} [\delta(k,l) - z_l]^2 \tag{129}$$

with

$$\delta(k,l) = \begin{cases} 1 & \text{if } k = l \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, if the approximations (117) provided perfectly accurate representations of their corresponding targets (115) (116) then (126) (129) represents

the optimal post-processor. Any other post-processor would yield suboptimal performance to the extend that its resulting decision rule differed from (126).

In practice, however, the chosen function approximation method ("preprocess") will not provide perfect approximations to the true target probabilities (115) (116). It is therefore possible that a (suboptimal) post-processor, different from (126) (129), will yield superior performance when used in conjunction with these suboptimal function approximations. This will tend to be the case as the approximations tend towards lesser accuracy. Heuristic arguments and some empirical evidence (Hastie, Tibshirani, and Buja, 1992, and Ripley, 1993) indicate that this is sometimes (but not always) the case. The connections between (suboptimal) function approximators (preprocess), and various possible classification post-processors to be used in conjunction with them, is as yet not well understood. As usual, it will no doubt depend on the particular problem at hand, as characterized by the true target function probabilities (115) (116), as well as the particular function approximation method used.

Because accurate function approximations (117) are not (necessarily) required to produce accurate classification decision rules (124), there tends to be less difference (in terms of accuracy) among the various methods for classification than there is for (direct) function approximation. Nevertheless, substantial classification accuracy differences can (and sometimes do) exist between chosen methods in particular problems. As with the function approximation problem, method choice should reflect one's (a priori) knowledge concerning the (true) nature of the class probabilities (115) (116).

As is the case with function approximation, classification methods differ widely on the interpretive value of the decision rule they produce, in terms of the usable information provided concerning the relationships between the probability distributions of the respective classes in the input space. As might be expected, the relative merits of the respective classification methods follow those of their corresponding function approximation counterparts. Classification methods based on tree-structured recursive partitioning (Section 4.3) and tensor product splines (Section 4.4) tend to have more interpretive value than those based on projection pursuit (Section 4.1), and especially more than those based on feed-forward neural networks (Section 4.1), radial basis functions (Section 4.2) and nearest-neighbor methods (Duda and Hart, 1973).

Bibliography

Akaike, H. (1974). A new look at statistical model identification. *IEEE Trans. Auto. Control* **19** 716–723.

Barron, A. (1984) Predicted squared error: a criterion for automatic model selection. In *Self-Organizing Methods in Modeling.* S. Farrow, ed., Marcel Dekker, New York.

Bates, D. M. and Watts, D. G. (1988). *Nonlinear Regression Analysis and its Applications.* Wiley, New York, NY.

Bellman, R. E. (1961). *Adaptive Control Proceses.* Princeton University Press.

Breiman, L. (1991). The Π-method for estimating multivariate functions from noisy data. *Technometrics* **33** 125–160.

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

Breiman L. and Friedman, J. H. (1994). A new approach to multiple outputs through stacking. Stanford University, Department of Statistics, Technical Report LCS114.

Breiman, L. and Spector, P. (1989). Submodel selection and evaluation in regression X random case. *Internat. Statist. Rev.* (to appear).

Craven, P. and Wahba, G. (1979). Smoothing noisy data with spline functions. Estimating the correct degree of smoothing by the method of generalized cross-validation. *Numer. Math.* **31** 317–403.

Denker, J. S. and LeCun, Y. (1991). Transforming neural-net output levels to probability distributions. In *Advances in Neural Information Processing Systems 3.* Lippmann, Moody, and Touretzky eds. Morgan Kaufman, San Mateo, CA.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis.* Wiley, New York, NY.

Efron, B. (1983). Estimating the error rate of a prediction rule. *J. Amer. Statist. Assoc.* **78** 316–333.

Frank, I. E. and Friedman, J. H. (1993). A statistical view of some chemometrics regression tools (with discussion). *Technometrics* **35** 109–148.

Friedman, J. H. (1985). Classification and multiple response regression through projection pursuit. Stanford University, Department of Statistics, Technical Report LCS012.

Friedman, J. H. (1991). Multivariate adaptive regression splines (with discussion). *Ann. Statist.* **19** 1–141.

Friedman, J. H. (1993). Estimating functions of mixed ordinal and categorical variables using adaptive splines. In: *New Directions in Statistical Data Analysis and Robustness*, Morgenthaler, Ronchetti, and Stahel, eds. Birkhauser

Friedman, J. H. and Stuetzle, W. (1981). Projection pursuit regression. *J. Amer. Statist. Assoc.* **76** 817–823.

Furnival, G. M. and Wilson, R. M. (1974). Regression by leaps and bounds. *Technometrics* **16** 499–512.

Gill, P. E., Murray, W. and Wright, M. H. (1981). *Practical Optimization.* Academic Press.

Girosi, F., Jones, M. and Poggio, T (1993). Priors, stabilizers and basis functions: from regularization to radial, tensor, and additive splines. Massachusetts Institute of Technology Artificial Intelligence Laboratory Technical Report A. I. 1430.

Hastie, T., Buja, A., and Tibshirani, R. (1992). Flexible discriminant analysis. *J. Amer. Statist. Assoc.* (to appear).

Holland, J. (1975). *Adaptation in Artificial and Neural Systems.* University of Michigan Press. Ann Arbor, MI.

Kolmogorov, A. N. (1957). On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Dokl. Akad. Nauk. USSR* **114** 953–956 (In Russian).

Lippmann, R. (1989). Pattern classification using neural networks. *IEEE Communications Magazine* **11** 47–64.

Lorentz, G. G. (1986). *Approximation of Functions.* Chelsea, New York, NY.

Mallows, C. L. (1973). Some comments on C_p. *Technometrics* **15** 661–675.

Moody, J. E. (1992) . The *effective* number of parameters: an analysis of generalization and regularization in nonlinear learning systems. In *Advances in Neural Information Processing Systems 4*, Moody, Hanson, and Lippmann, eds., Morgan Kaufmann Publishers, San Mateo, CA.

Ripley, B. D. (1994). Neural networks and related methods for classification (with discussion). *J. Roy. Statist. Soc. B* **56** (to appear).

Rissanen, Y. (1983). A universal prior for integers and estimation by minimum description length. *Ann. Statist.* **6** 416–431.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Rumelhart, McClelland, eds. MIT Press, Cambridge, MA.

Schwartz, G. (1978). Estimating the dimension of a model. *Ann. Statist.* **6** 461–464.

Wahba, G. (1990). *Spline Models for Observational Data.* SIAM, Philadelphia.

Wiegand, A. S., Huberman, B. A. and Rumelhart, D. (1991). Predicting the future: a connectionist approach. *Intl. J. Neural Syst.* **1** 193–209.