CHAPTER

9

ALGORITHM-INDEPENDENT MACHINE LEARNING

9.1 INTRODUCTION

In the previous chapters we have seen many learning algorithms and techniques for pattern recognition. When confronting such a range of algorithms, everyone has wondered at one time or another which one is "best." Of course, some algorithms may be preferred because of their lower computational complexity; others may be preferred because they take into account some prior knowledge of the form of the data (e.g., discrete, continuous, unordered list, string, ...). Nevertheless, there are classification problems for which such issues are of little or no concern, or we wish to compare algorithms that are equivalent in regard to them. In these cases we are left with the question, Are there any reasons to favor one algorithm over another? For instance, given two classifiers that perform equally well on the training set, it is frequently asserted that the *simpler* classifier can be expected to perform better on a test set. But is this version of *Occam's razor* really so evident? Likewise, we frequently prefer or impose *smoothness* on a classifier's decision functions. Do simpler or "smoother" classifiers generalize better, and, if so, why? In this chapter we address these and related questions concerning the foundations and philosophical underpinnings of statistical pattern classification. Now that the reader has intuition and experience with individual algorithms, these issues in the theory of learning may be better understood.

OCCAM'S RAZOR

In some fields there are strict conservation laws and constraint laws—such as the conservation of energy, charge, and momentum in physics, as well as the second law of thermodynamics, which states that the entropy of an isolated system can never decrease. These hold regardless of the number and configuration of the forces at play. Given the usefulness of such laws, we naturally ask, Are there analogous results in pattern recognition, ones that do not depend upon the particular choice of classifier or learning method? Are there any fundamental results that hold regardless of the cleverness of the designer, the number and distribution of the patterns, and the nature of the classification task?

Of course it is very valuable to know that there exists a constraint on classifier accuracy, the Bayes error rate, and it is sometimes useful to compare performance

to this theoretical limit. Alas, in practice we rarely, if ever, know the Bayes error rate. Even if we did know this error rate, it would not help us much in designing a classifier except to tell us that further training and data collection is futile. Thus the Bayes error is generally of theoretical interest. What other fundamental principles and properties might be of greater use in designing classifiers?

Before we address such problems, we should clarify the meaning of the title of this chapter. "Algorithm-independent" here refers, first, to those mathematical foundations that do not depend upon the particular classifier or learning algorithm used. Our upcoming discussion of bias and variance is just as valid for methods based on neural networks or the nearest-neighbor or model-dependent maximum-likelihood. Second, we mean techniques that can be used in conjunction with different learning algorithms, or provide guidance in their use. For example, cross-validation and resampling techniques can be used with any of a large number of training methods. Of course by the very general notion of an algorithm, these too are algorithms, technically speaking, but we discuss them because of their breadth of applicability and their independence from details of the learning techniques used.

In this chapter we shall see, first, that no pattern classification method is inherently superior to any other, or even to random guessing; it is the type of problem, prior distribution, and other information that determine which form of classifier should provide the best performance. We shall then explore several ways to quantify and adjust the "match" between a learning algorithm and the problem it addresses. In any particular problem there are differences between classifiers, of course, and thus we show that with certain assumptions we can estimate their accuracies (even, for instance, before the candidate classifier is fully trained) and compare different classifiers. Finally, we shall see methods for integrating component or "expert" classifiers, which themselves might implement quite different algorithms.

We shall present the results that are most important for pattern recognition practitioners, generally skipping over mathematical proofs that can be found in the original research referenced in the Bibliographical and Historical Remarks section.

9.2 LACK OF INHERENT SUPERIORITY OF ANY CLASSIFIER

We now turn to the central question posed above: If we are interested solely in the generalization performance, are there any reasons to prefer one classifier or learning algorithm over another? If we make no prior assumptions about the nature of the classification task, can we expect any classification method to be superior or inferior overall? Can we even find an algorithm that is overall superior to (or inferior to) random guessing?

9.2.1 No Free Lunch Theorem

As summarized in the *No Free Lunch Theorem*, the answer to these and several related questions is "no." If the goal is to obtain good generalization performance, there are no context-independent or usage-independent reasons to favor one learning or classification method over another. If one algorithm seems to outperform another in a particular situation, it is a consequence of its fit to the particular pattern recognition problem, not the general superiority of the algorithm. When confronting a new pattern recognition problem, appreciation of this theorem reminds us to focus on the aspects that matter most—prior information, data distribution, amount of training data, and cost or reward functions. The theorem also justifies a healthy skepticism

regarding studies that purport to demonstrate the overall superiority of a particular learning or recognition algorithm.

First we should consider more closely the method by which we judge the generalization performance of a classifer. Up to here, we have estimated such performance by means of a test data set, sampled independently, as is the training set. In some cases, this approach has some unexpected drawbacks when applied to comparing classifiers. In a discrete problem, for example, when the training set and test set are very large, they necessarily overlap, and we are testing on training patterns. Further, virtually any powerful algorithm such as the nearest-neighbor algorithm, unpruned decision trees, or neural networks with sufficient number of hidden nodes can learn the training set perfectly. Second, for low-noise or low-Bayes error cases, if we use an algorithm powerful enough to learn the training set, then the upper limit of the i.i.d. error decreases as the training set size increases.

OFF-TRAINING SET ERROR

Thus, in order to compare learning algorithms, we will use the *off-training set error*—the error on points *not* in the training set, as will become clear. If the training set is very large, then the maximum size of the off-training set data is necessarily small.

For simplicity consider a two-category problem, where the training set $\mathcal{D}$ consists of patterns $\mathbf{x}^i$ and associated category labels $y_i = \pm 1$ for $i = 1, \ldots, n$ generated by the unknown target function to be learned, $F(\mathbf{x})$, where $y_i = F(\mathbf{x}^i)$. In most cases of interest there is a random component in $F(\mathbf{x})$ and thus the same input could lead to different categories, giving nonzero Bayes error. At first we shall assume that the feature set is discrete; this simplifies notation and allows the use of summation and probabilities rather than integration and probability densities. The general conclusions hold in the continuous case as well, but the required technical details would cloud our discussion.

Let $\mathcal{H}$ denote the (discrete) set of hypotheses, or possible sets of parameters to be learned. A particular hypothesis $h(\mathbf{x}) \in \mathcal{H}$ could be described by quantized weights in a neural network, or parameters $\boldsymbol{\theta}$ in a functional model, or sets of decisions in a tree, and so on. Furthermore, $P(h)$ is the prior probability that the algorithm will produce hypothesis h after training; note that this is *not* the probability that h is correct. Next, $P(h|\mathcal{D})$ denotes the probability that the algorithm will yield hypothesis h when trained on the data $\mathcal{D}$. In deterministic learning algorithms such as the nearest-neighbor and decision trees, $P(h|\mathcal{D})$ will be everywhere zero except for a single hypothesis h. For stochastic methods (such as neural networks trained from random initial weights), or stochastic Boltzmann learning, $P(h|\mathcal{D})$ can be a broad distribution. Let E be the error for a zero-one or other loss function.

How shall we judge the generalization quality of a learning algorithm? Because we are not given the target function, the natural measure is the expected value of the error given $\mathcal{D}$, summed over all possible targets. This scalar value can be expressed as a weighted "inner product" between the distributions $P(h|\mathcal{D})$ and $P(F|\mathcal{D})$, as follows:

$$\mathcal{E}[E|\mathcal{D}] = \sum_{h,F} \sum_{\mathbf{x} \notin \mathcal{D}} P(\mathbf{x})[1 - \delta(F(\mathbf{x}), h(\mathbf{x}))] P(h|\mathcal{D}) P(F|\mathcal{D}), \tag{1}$$

where for the moment we assume there is no noise. The familiar Kronecker delta function, $\delta(\cdot, \cdot)$, has value 1 if its two arguments match, and value 0 otherwise. Equation 1 states that the expected error rate, given a fixed training set $\mathcal{D}$, is related to the sum over all possible inputs weighted by their probabilities, $P(\mathbf{x})$, as well as

the "alignment" or "match" of the learning algorithm, $P(h|\mathcal{D})$, to the actual posterior $P(F|\mathcal{D})$. The important insight provided by this equation is that without prior knowledge concerning $P(F|\mathcal{D})$, we can prove little about any particular learning algorithm $P(h|\mathcal{D})$, including its generalization performance.

The expected off-training-set classification error when the true function is $F(\mathbf{x})$ and the probability for the kth candidate learning algorithm is $P_k(h(\mathbf{x})|\mathcal{D})$ is given by

$$\mathcal{E}_k(E|F,n) = \sum_{\mathbf{x}\notin\mathcal{D}} P(\mathbf{x})[1-\delta(F(\mathbf{x}),h(\mathbf{x}))]P_k(h(\mathbf{x})|\mathcal{D}). \tag{2}$$

Although we shall not provide a formal proof, we are now in a position to give a precise statement of the No Free Lunch Theorem.*

■ **Theorem 9.1. (No Free Lunch)** For any two learning algorithms $P_1(h|\mathcal{D})$ and $P_2(h|\mathcal{D})$, the following are true, independent of the sampling distribution $P(\mathbf{x})$ and the number n of training points:

1. Uniformly averaged over all target functions F, $\mathcal{E}_1(E|F,n) - \mathcal{E}_2(E|F,n) = 0$
2. For any fixed training set $\mathcal{D}$, uniformly averaged over F, $\mathcal{E}_1(E|F,\mathcal{D}) - \mathcal{E}_2(E|F,\mathcal{D}) = 0$
3. Uniformly averaged over all priors $P(F)$, $\mathcal{E}_1(E|n) - \mathcal{E}_2(E|n) = 0$
4. For any fixed training set $\mathcal{D}$, uniformly averaged over $P(F)$, $\mathcal{E}_1(E|\mathcal{D}) - \mathcal{E}_2(E|\mathcal{D}) = 0$

Part 1 says that uniformly averaged over all target functions the expected off-training set error for all learning algorithms is the same, that is,

$$\sum_F \sum_{\mathcal{D}} P(\mathcal{D}|F)\,[\mathcal{E}_1(E|F,n) - \mathcal{E}_2(E|F,n)] = 0, \tag{3}$$

for any two learning algorithms. In short, no matter how clever we are at choosing a "good" learning algorithm $P_1(h|\mathcal{D})$ and a "bad" algorithm $P_2(h|\mathcal{D})$ (perhaps even random guessing, or a constant output), if all target functions are equally likely, then the "good" algorithm will not outperform the "bad" one. Stated more generally, there are no i and j such that for all $F(\mathbf{x})$, $\mathcal{E}_i(E|F,n) > \mathcal{E}_j(E|F,n)$. Furthermore, no matter what algorithm we use, there is at least one target function for which random guessing is a better algorithm.

With the assumption that the training set can be learned by all algorithms we consider, Part 2 states that even if we know $\mathcal{D}$, then averaged over all target functions no learning algorithm yields an off-training set error that is superior to any other, that is,

$$\sum_F [\mathcal{E}_1(E|F,\mathcal{D}) - \mathcal{E}_2(E|F,\mathcal{D})] = 0. \tag{4}$$

Parts 3 and 4 concern nonuniform target function distributions and have related interpretations (Problems 2–5). Example 1 provides an elementary illustration.

*The clever name for the theorem was suggested by David Haussler.

EXAMPLE 1 No Free Lunch for Binary Data

Consider input vectors consisting of three binary features, and a particular target function $F(\mathbf{x})$, as given in the table. Suppose (deterministic) learning algorithm 1 assumes every pattern is in category ω_1 unless trained otherwise, and algorithm 2 assumes every pattern is in ω_2 unless trained otherwise. Thus when trained with $n = 3$ points in $\mathcal{D}$, each algorithm returns a single hypothesis, h_1 and h_2, respectively. In this case the expected errors on the off-training set data are $\mathcal{E}_1(E|F, \mathcal{D}) = 0.4$ and $\mathcal{E}_2(E|F, \mathcal{D}) = 0.6$.

	x	F	h_1	h_2
$\mathcal{D}$	000	1	1	1
	001	−1	−1	−1
	010	1	1	1
	011	−1	1	−1
	100	1	1	−1
	101	−1	1	−1
	110	1	1	−1
	111	1	1	−1

For this target function $F(\mathbf{x})$, clearly algorithm 1 is superior to algorithm 2. But note that the designer does not *know* $F(\mathbf{x})$—indeed, we assume we have no prior information about $F(\mathbf{x})$. The fact that all targets are equally likely means that $\mathcal{D}$ provides no information about $F(\mathbf{x})$. If we wish to compare the algorithms overall, we therefore must average over all such possible target functions consistent with the training data. Part 2 of Theorem 9.1 states that averaged over all possible target functions, there is no difference in off-training set errors between the two algorithms. For each of the 2^5 distinct target functions consistent with the $n = 3$ patterns in $\mathcal{D}$, there is exactly one other target function whose output is inverted for each of the patterns outside the training set, and this ensures that the performances of algorithms 1 and 2 will also be inverted, so that the contributions to the formula in Part 2 cancel. Thus indeed Part 2 of the Theorem as well as Eq. 4 are obeyed.

Figure 9.1 illustrates a result derivable from Part 1 of Theorem 9.1. Each of the six squares represents the set of all possible classification problems; note that this is *not* the standard feature space. If a learning system performs well—higher than average generalization accuracy—over some set of problems, then it must perform worse than average elsewhere, as shown in Fig. 9.1 a. No system can perform well throughout the full set of functions (Fig. 9.1 d); to do so would violate the No Free Lunch Theorem.

In sum, all statements of the form "learning/recognition algorithm 1 is better than algorithm 2" are ultimately statements about the relevant target functions. Hence there is a "conservation theorem" in generalization: For every possible learning algorithm for binary classification the sum of performance over all possible target functions is exactly zero. Thus we cannot achieve positive performance on some problems without getting an equal and opposite amount of negative performance on other problems. While we may hope that we never have to apply any particular algorithm to certain problems, all we can do is trade performance on problems we do not expect to encounter with those that we do expect to encounter. This, along with the other results from the No Free Lunch Theorem, stresses that it is the *assumptions*

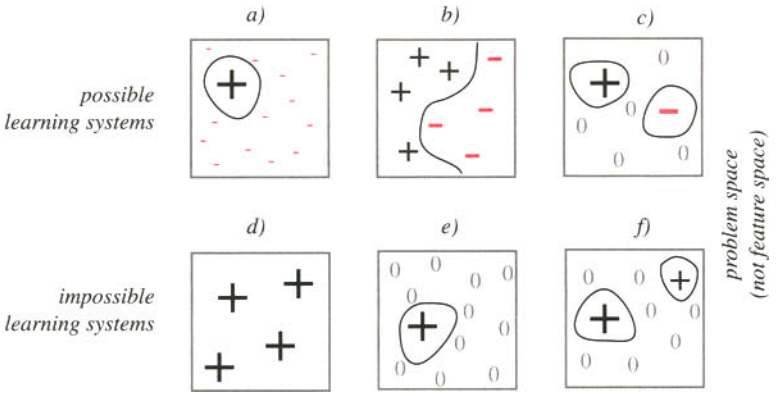


FIGURE 9.1. The No Free Lunch Theorem shows the generalization performance on the off-training set data that *can* be achieved (top row) and also shows the performance that *cannot* be achieved (bottom row). Each square represents all possible classification problems consistent with the training data—this is *not* the familiar feature space. A + indicates that the classification algorithm has generalization higher than average, a − indicates lower than average, and a 0 indicates average performance. The size of a symbol indicates the amount by which the performance differs from the average. For instance, part a shows that it is possible for an algorithm to have high accuracy on a small set of problems so long as it has mildly poor performance on all other problems. Likewise, part b shows that it is possible to have excellent performance throughout a large range of problem, but this will be balanced by very poor performance on a large range of other problems. It is impossible, however, to have good performance throughout the full range of problems, shown in part d. It is also impossible to have higher-than-average performance on some problems while having average performance everywhere else, shown in part e.

about the learning domains that are relevant. Another practical import of the theorem is that even popular and theoretically grounded algorithms will perform poorly on some problems, ones in which the learning algorithm and the posterior happen not to be "matched," as governed by Eq. 1. Practitioners must be aware of this possibility, which arises in real-world applications. Expertise limited to a small range of methods, even powerful ones such as neural networks, will not suffice for all classification problems. Experience with a broad range of techniques is the best insurance for solving arbitrary new classification problems.

*9.2.2 Ugly Duckling Theorem

While the No Free Lunch Theorem shows that in the absence of assumptions we should not prefer any learning or classification algorithm over another, an analogous theorem addresses features and patterns. Roughly speaking, the Ugly Duckling Theorem states that in the absence of assumptions there is no privileged or "best" feature representation, and that even the notion of similarity between patterns depends implicitly on assumptions that may or may not be correct.

Because we are using discrete representations, we can use logical expressions or "predicates" to describe a pattern, much as in Chapter 8. If we denote a binary feature attribute by f_i, then a particular pattern might be described by the predicate "f_1 *AND* f_2," another pattern might be described as "*NOT* f_2," and so on. Likewise

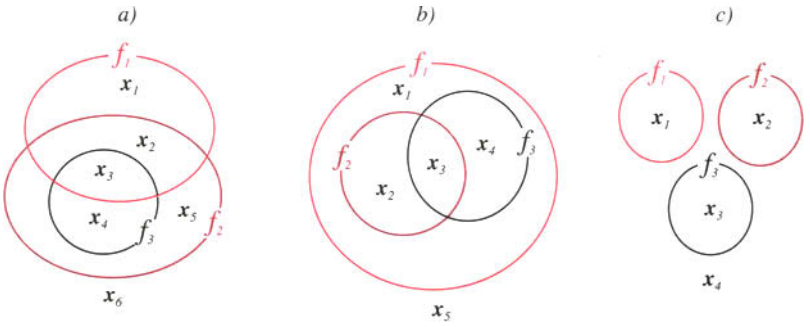


FIGURE 9.2. Patterns $\mathbf{x}_i$, represented as d-tuples of binary features f_i, can be placed in Venn diagram (here $d = 3$); the diagram itself depends upon the classification problem and its constraints. For instance, suppose f_1 is the binary feature attribute `has_legs`, f_2 is `has_right_arm` and f_3 the attribute `has_right_hand`. Thus in part a pattern $\mathbf{x}_1$ denotes a person who has legs but neither arm nor hand; $\mathbf{x}_2$ a person who has legs and an arm, but no hand; and so on. Notice that the Venn diagram expresses the biological constraints associated with real people: it is impossible for someone to have a right hand but no right arm. Part c expresses different constraints, such as the biological constraint of mutually exclusive eye colors. Thus attributes f_1, f_2 and f_3 might denote `brown`, `green`, and `blue`, respectively, and a pattern $\mathbf{x}_i$ describes a real person, whom we can assume cannot have eyes that differ in color.

we could have a predicate involving the patterns themselves, such as $\mathbf{x}_1$ *OR* $\mathbf{x}_2$. Figure 9.2 shows how patterns can be represented in a Venn diagram.

Below we shall need to count predicates, and for clarity it helps to consider a particular Venn diagram, such as that in Fig. 9.3. This is the most general Venn diagram based on two features, because for every configuration of f_1 and f_2 there is indeed a pattern. Here predicates can be as simple as "$\mathbf{x}_1$," or more complicated, such as "$\mathbf{x}_1$ *OR* $\mathbf{x}_2$ *OR* $\mathbf{x}_4$," and so on.

RANK

The *rank* r of a predicate is the number of the simplest or indivisible elements it contains. The tables below show the predicates of rank 1, 2, and 3 associated with the Venn diagram of Fig. 9.3.* Not shown is the fact that there is but one predicate of rank $r = 4$, the disjunction of the $\mathbf{x}_1, \ldots, \mathbf{x}_4$, which has the logical value `True`. If we

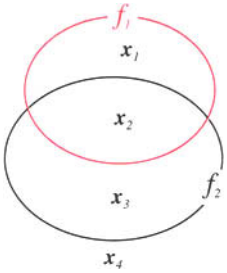


FIGURE 9.3. The Venn for a problem with no constraints on two features. Thus all four binary attribute vectors can occur.

*Technically speaking, we should use set operations rather than logical operations when discussing the Venn diagram, writing $\mathbf{x}_1 \cup \mathbf{x}_2$ instead of $\mathbf{x}_1$ OR $\mathbf{x}_2$. Nevertheless, we use logical operations here for consistency with the rest of the text.

let n be the total number of regions in the Venn diagram (i.e., the number of distinct possible patterns), then there are $\binom{n}{r}$ predicates of rank r, as shown at the bottom of the table.

rank $r = 1$

$\mathbf{x}_1$	f_1 *AND NOT* f_2
$\mathbf{x}_2$	f_1 *AND* f_2
$\mathbf{x}_3$	f_2 *AND NOT* f_1
$\mathbf{x}_4$	*NOT* (f_1 *OR* f_2)

$\binom{4}{1} = 4$

rank $r = 2$

$\mathbf{x}_1$ OR $\mathbf{x}_2$	f_1
$\mathbf{x}_1$ OR $\mathbf{x}_3$	f_1 XOR f_2
$\mathbf{x}_1$ OR $\mathbf{x}_4$	NOT f_2
$\mathbf{x}_2$ OR $\mathbf{x}_3$	f_2
$\mathbf{x}_2$ OR $\mathbf{x}_4$	NOT(f_1 AND f_2)
$\mathbf{x}_3$ OR $\mathbf{x}_4$	NOT f_1

$\binom{4}{2} = 6$

rank $r = 3$

$\mathbf{x}_1$ OR $\mathbf{x}_2$ OR $\mathbf{x}_3$	f_1 OR f_2
$\mathbf{x}_1$ OR $\mathbf{x}_2$ OR $\mathbf{x}_4$	f_1 OR NOT f_2
$\mathbf{x}_1$ OR $\mathbf{x}_3$ OR $\mathbf{x}_4$	NOT(f_1 AND f_2)
$\mathbf{x}_2$ OR $\mathbf{x}_3$ OR $\mathbf{x}_4$	f_2 OR NOT f_1

$\binom{4}{3} = 4$

The total number of predicates in the absence of constraints is

$$\sum_{r=0}^{n} \binom{n}{r} = (1+1)^n = 2^n, \tag{5}$$

and thus for the $d = 4$ case of Fig. 9.3, there are $2^4 = 16$ possible predicates (Problem 9). Note that Eq. 5 applies only to the case where there are no constraints; for Venn diagrams that do incorporate constraints, such as those in Fig. 9.2, the formula does not hold (Problem 10).

Now we turn to our central question: In the absence of prior information, is there a principled reason to judge any two distinct patterns as more or less similar than two other distinct patterns? A natural and familiar measure of similarity is the number of features or attributes shared by two patterns, but even such an obvious measure presents conceptual difficulties.

To appreciate such difficulties, consider first a simple example. Suppose attributes f_1 and f_2 represent `blind_in_right_eye` and `blind_in_left_eye`, respectively. If we base similarity on shared features, person $\mathbf{x}_1 = \{1, 0\}$ (blind only in the right eye) is maximally different from person $\mathbf{x}_2 = \{0, 1\}$ (blind only in the left eye). In particular, in this scheme $\mathbf{x}_1$ is more similar to a totally blind person and to a normally sighted person than he is to $\mathbf{x}_2$. But this result may prove unsatisfactory; we can easily envision many circumstances where we would consider a person blind in just the right eye to be "similar" to one blind in just the left eye. Such people might be permitted to drive automobiles, for instance. Furthermore, a person blind in just one eye would differ significantly from totally blind person who would not be able to drive.

A second, related point is that there are always multiple ways to represent vectors (or tuples) of attributes. For instance, in the above example, we might use alternative features f_1' and f_2' to represent `blind_in_right_eye` and `same_in_both_eyes`, respectively, and then the four types of people would be represented as shown in the tables.

	f_1	f_2
$\mathbf{x}_1$	0	0
$\mathbf{x}_2$	0	1
$\mathbf{x}_3$	1	0
$\mathbf{x}_4$	1	1

f_1'	f_2'
0	1
0	0
1	0
1	1

Of course there are other representations, each more or less appropriate to the particular problem at hand. In the absence of prior information, though, there is no principled reason to prefer one of these representations over another.

We must then still confront the problem of finding a principled measure of the similarity between two patterns, given some representation. The only plausible candidate measure in this circumstance would be the number of *predicates* (rather than the number of features) the patterns share. Consider two distinct patterns (in some representation) $\mathbf{x}_i$ and $\mathbf{x}_j$, where $i \neq j$. Regardless of the constraints in the problem (i.e., the Venn diagram), there are, of course, no predicates of rank $r = 1$ that are shared by the two patterns. There is but one predicate of rank $r = 2$; it is $\mathbf{x}_i$ *OR* $\mathbf{x}_j$. A predicate of rank $r = 3$ must contain three patterns, two of which are $\mathbf{x}_i$ and $\mathbf{x}_j$. Because there are d patterns total, there are then $\binom{d-2}{1} = d - 2$ predicates of rank 3 that are shared by $\mathbf{x}_i$ and $\mathbf{x}_j$. Likewise, for an arbitrary rank r, there are $\binom{d-2}{r-2}$ predicates shared by the two patterns, where $2 \leq r \leq d$. The total number of predicates shared by the two patterns is thus the sum

$$\sum_{r-2}^{d} \binom{d-2}{r-2} = (1+1)^{d-2} = 2^{d-2}. \tag{6}$$

Note the key result: Equation 6 is *independent* of the choice of $\mathbf{x}_i$ and $\mathbf{x}_j$ (so long as they are distinct). Thus we conclude that the number of predicates shared by two distinct patterns is *constant*, and *independent* of the patterns themselves (Problem 11). We conclude that if we judge similarity based on the number of predicates that patterns share, then any two distinct patterns are "equally similar." This is stated formally as the following theorem:

■ **Theorem 9.2. (Ugly Duckling)** Given that we use a finite set of predicates that enables us to distinguish any two patterns under consideration, the number of predicates shared by any two such patterns is constant and independent of the choice of those patterns. Furthermore, if pattern similarity is based on the total number of predicates shared by two patterns, then any two patterns are "equally similar." *

In summary, then, the Ugly Duckling Theorem states something quite basic yet often overlooked: There is no problem-independent or privileged or "best" set of features or feature attributes. Moreover, while the above was derived using d-tuples of binary values, it also applies to a continuous feature space too, if such a space is discretized (at any resolution). The theorem forces us to acknowledge that even the apparently simple notion of similarity between patterns is fundamentally based on implicit assumptions about the problem domain (Problem 12).

9.2.3 Minimum Description Length (MDL)

It is sometimes claimed that the minimum description length principle provides justification for preferring one type of classifier over another—specifically, "simpler" classifiers over "complex" ones. Briefly stated, the approach purports to find some irreducible, smallest representation of all members of a category (much like a "signal"); all variation among the individual patterns is then "noise." The argument is

*The theorem gets its fanciful name from the following counterintuitive statement: Assuming that similarity is based on the number of shared predicates, an ugly duckling A is as similar to beautiful swan B as beautiful swan C is to B, given that these items differ at all from one another.

that by simplifying recognizers appropriately, the signal can be retained while the noise is ignored. Because the principle is so often invoked, it is important to understand what properly derives from it, what does not, and how it relates to the No Free Lunch Theorem. To do so, however, we must first understand the notion of algorithmic complexity.

Algorithmic Complexity

Algorithmic complexity—also known as Kolmogorov complexity, Kolmogorov-Chaitin complexity, descriptional complexity, shortest program length, or algorithmic entropy—seeks the inherent complexity of a binary string. (We shall assume that both classifiers and patterns are described by such strings.) Algorithmic complexity can be explained by analogy to communication, the earliest application of information theory (Section A.7 of the Appendix). If the sender and receiver agree upon a specification method, such as an encoding or compression technique, then message x can then be transmitted as y and decoded given some fixed method L, denoted $L(y) = x$. The cost of transmission of x is the length of the transmitted message y, that is, $|y|$. The least such cost is hence the minimum length of such a message, denoted $\min_{y:L(y)=x} |y|$.

ABSTRACT COMPUTER

Algorithmic complexity is defined by analogy to communication, where instead of a fixed decoding method L, we consider programs running on an *abstract computer*—that is, one whose functions (memory, processing, etc.) are described operationally and without regard to hardware implementation. Consider an abstract computer that takes as a program a binary string y and outputs a string x and halts. In such a case we say that y is an abstract encoding or description of x.

A *universal* description should be independent of the specification (up to some additive constant)—it should not depend upon whether our specification is programmed in *C++*, *Lisp*, *Java*, and so on. Such a description would then allow us to reliably compare the complexities of different binary strings x_1 and x_2. Such a method would provide a measure of the inherent information content, the amount of data that must be transmitted in the absence of any other prior knowledge. The Kolmogorov complexity of a binary string x, denoted $K(x)$, is defined as the size of the *shortest* program string y (where y's length is measured in bits), that, without additional data, computes the string x and halts. Formally, we write

$$K(x) = \min_{y:U(y)=x} |y|, \tag{7}$$

TURING MACHINE

where U represents an abstract universal *Turing machine* or Turing computer. For our purposes it suffices to state that a Turing machine is "universal" in that it can implement any algorithm and compute any computable function. Kolmogorov complexity is a measure of the incompressibility of x and is analogous to minimal sufficient statistics, the optimally compressed representation of certain properties of a distribution (Chapter 3).

Consider the following examples. Suppose x consists solely of n 1s. This string is actually quite "simple." If we use some fixed number of bits k to specify a general program containing a loop for printing a string of 1s, we need merely $\log_2 n$ more bits to specify the iteration number n, the condition for halting. Thus the Kolmogorov complexity of a string of n 1s is $K(x) = O(\log_2 n)$. Next consider the transcendental number π, whose infinite sequence of seemingly random binary digits, $11.001001000011111101101010001\ldots_2$, actually contains only a few bits of

information: the size of the shortest program that can produce any arbitrarily large number of consecutive digits of π. Informally we say the algorithmic complexity of π is a constant; formally we write $K(\pi) = O(1)$, which means that $K(\pi)$ does not grow with increasing number of desired bits. Another example is a "truly" random binary string, which cannot be expressed as a shorter string; its algorithmic complexity is within a constant factor of its length. For such a string we write $K(x) = O(|x|)$, which means that $K(x)$ grows as fast as the length of x (Problem 13).

9.2.4 Minimum Description Length Principle

We now turn to a simple, "naive" version of the minimum description length principle and its application to pattern recognition. Given that all members of a category share some properties, yet differ in others, the recognizer should seek to learn the common or essential characteristics while ignoring the accidental or random ones. Kolmogorov complexity seeks to provide an objective measure of simplicity, and thus the description of the "essential" characteristics.

Suppose we seek to design a classifier using a training set $\mathcal{D}$. The *minimum description length (MDL) principle* states that we should minimize the sum of the model's algorithmic complexity and the description of the training data with respect to that model, that is,

$$K(h, \mathcal{D}) = K(h) + K(\mathcal{D} \text{ using } h). \tag{8}$$

Thus we seek the model h^* that obeys $h^* = \arg\min_h K(h, \mathcal{D})$, as explored in Problem 14. (Variations on the naive minimum description length principle use a *weighted* sum of the terms in Eq. 8.) In practice, determining the algorithmic complexity of a classifier depends upon a chosen class of abstract computers, and this means that the complexity can be specified only up to an additive constant.

A particularly clear application of the minimum description length principle is in the design of decision-tree classifiers (Chapter 8). In this case, a model h specifies the tree and the decisions at the nodes; thus the algorithmic complexity of the model is proportional to the number of nodes. The complexity of the data given the model could be expressed in terms of the entropy (in bits) of the data $\mathcal{D}$, the weighted sum of the entropies of the data at the leaf nodes. Thus if the tree is pruned based on an entropy criterion, there is an implicit global cost criterion that is equivalent to minimizing a measure of the general form in Eq. 8 (Computer exercise 1).

It can be shown theoretically that classifiers designed with a minimum description length principle are guaranteed to converge to the ideal or true model *in the limit of more and more data*. This is surely a very desirable property. However, such derivations cannot prove that the principle leads to superior performance in the *finite* data case; to do so would violate the No Free Lunch Theorem. Moreover, in practice it is often difficult to compute the minimum description length, because we may not be clever enough to find the "best" representation (Problem 17). Assume that there is some correspondence between a particular classifier and an abstract computer; in such a case it may be quite simple to determine the length of the string y necessary to create the classifier. But because finding the algorithmic complexity demands we find the *shortest* such string, we must perform a very difficult search through possible programs that could generate the classifier.

The minimum description length principle can be viewed from a Bayesian perspective. Using our current terminology, Bayes formula states

$$P(h|\mathcal{D}) = \frac{P(h)P(\mathcal{D}|h)}{P(\mathcal{D})} \tag{9}$$

for discrete hypotheses and data. The optimal hypothesis h^* is the one yielding the highest posterior probability, that is,

$$\begin{aligned} h^* &= \arg\max_h [P(h)P(\mathcal{D}|h)] \\ &= \arg\max_h [\log_2 P(h) + \log_2 P(\mathcal{D}|h)], \end{aligned} \tag{10}$$

much as we saw in Chapter 3. We note that a string x can be communicated or represented at a cost bounded below by $-\log_2 P(x)$, as stated in Shannon's optimal coding theorem. Shannon's theorem thus provides a link between the minimum description length (Eq. 8) and the Bayesian approaches (Eq. 10). The minimum description length principle states that simple models (small $K(h)$) are to be preferred, and thus amounts to a bias toward "simplicity." It is often easier in practice to specify such a prior in terms of a description length than it is using functions of distributions (Problem 16). We shall revisit the issue of the trade-off between simplifying the model and fitting the data in the bias-variance dilemma in Section 9.3.

It is found empirically that classifiers designed using the minimum description length principle work well in many problems. As mentioned, the principle is effectively a method for biasing priors over models toward "simple" models. The reasons for the many empirical success of the principle are not trivial, as we shall see in Section 9.2.5. One of the greatest benefits of the principle is that it provides a computationally clear approach to balancing model complexity and the fit of the data. In somewhat more heuristic methods, such as pruning neural networks, it is difficult to compare the algorithmic complexity of the network (e.g., number of units or weights) with the entropy of the data with respect to that model.

9.2.5 Overfitting Avoidance and Occam's Razor

Throughout our discussions of pattern classifiers, we have mentioned the need to avoid overfitting by means of regularization, pruning, inclusion of penalty terms, minimizing a description length, and so on. The No Free Lunch results throw such techniques into question. If there are no problem-independent reasons to prefer one algorithm over another, why is overfitting avoidance nearly universally advocated? For a given training error, why do we generally advocate simple classifiers with fewer features and parameters?

In fact, techniques for avoiding overfitting or minimizing description length are not inherently beneficial; instead, such techniques amount to a preference, or "bias," over the forms or parameters of classifiers. They are only beneficial if they happen to address problems for which they work. It is the match of the learning algorithm to the *problem*—not the imposition of overfitting avoidance—that determines the empirical success. There are problems for which overfitting avoidance actually leads to worse performance. The effects of overfitting avoidance depend upon the choice of representation too; if the feature space is mapped to a new, formally equivalent one, overfitting avoidance has different effects.

In light of the negative results from the No Free Lunch theorems, we might probe more deeply into the frequent empirical "successes" of the minimum description length principle and the more general philosophical principle of Occam's razor. In its original form, Occam's razor stated merely that "entities" (or explanations) should

not be multiplied beyond necessity, but it has come to be interpreted in pattern recognition as counseling that one should not use classifiers that are more complicated than are necessary, where "necessary" is determined by the quality of fit to the training data. Given the respective requisite assumptions, the No Free Lunch theorem proves that there is no benefit in "simple" classifiers (or "complex" ones, for that matter)—simple classifiers claim neither unique nor universal validity.

The frequent empirical "successes" of Occam's razor imply that the classes of problems addressed so far have certain properties. What might be the reason we explore problems that tend to favor simpler classifiers? A reasonable hypothesis is that through evolution, we have had strong selection pressure on our pattern recognition apparatuses to be computationally simple—require fewer neurons, less time, and so forth—and in general such classifiers tend to be "simple." We are more likely to ignore problems for which Occam's razor does not hold. Analogously, researchers naturally develop simple algorithms before more complex ones, as for instance in the progression from the Perceptron, to multilayer neural networks, to networks with pruning, to networks with topology learning, to hybrid neural net/rule-based methods, and so on—each more complex than its predecessor. Each method is found to work on some problems, but not ones that are "too complex." For instance the basic Perceptron is inadequate for optical character recognition; a simple three-layer neural network is inadequate for speaker-independent speech recognition. Hence our design methodology itself imposes a bias toward "simple" classifiers; we generally stop searching for a design when the classifier is "good enough." This principle of *satisficing*—creating an adequate though possibly nonoptimal solution—underlies much of practical pattern recognition as well as human cognition.

SATISFICING

Another "justification" for Occam's razor derives from a property we might strongly desire or expect in a learning algorithm. If we assume that adding more training data does not, on average, degrade the generalization accuracy of a classifier, then a version of Occam's razor can in fact be derived. Note, however, that such a desired property amounts to a nonuniform prior over target functions; while this property is surely desirable, it is a premise and cannot be "proven." Finally, the No Free Lunch theorem implies that we cannot use training data to create a scheme by which we can with some assurance distinguish new problems for which the classifier will generalize well from new problems for which the classifier will generalize poorly (Problem 8).

9.3 BIAS AND VARIANCE

Given that there is no general best classifier unless the probability over the class of problems is restricted, practitioners must be prepared to explore a number of methods or models when solving any given classification problem. Below we will define two ways to measure the "match" or "alignment" of the learning algorithm to the classification problem: the bias and the variance. The bias measures the accuracy or quality of the match: high bias implies a *poor* match. The variance measures the precision or specificity of the match: a high variance implies a *weak* match. Designers can adjust the bias and variance of classifiers, but the important bias-variance relation shows that the two terms are not independent; in fact, for a given mean-square error, they obey a form of "conservation law." Naturally, though, with prior information or even mere luck, classifiers can be created that have a different mean-square error.

9.3.1 Bias and Variance for Regression

Bias and variance are most easily understood in the context of regression or curve fitting. Suppose there is a true (but unknown) function $F(\mathbf{x})$ with continuous valued output with noise, and we seek to estimate $F(\cdot)$ based on n samples in a set $\mathcal{D}$ generated by $F(\mathbf{x})$. The regression function estimated is denoted $g(\mathbf{x};\ \mathcal{D})$ and we are interested in the dependence of this approximation on the training set $\mathcal{D}$. Due to random variations in data selection, for some data sets of finite size this approximation will be excellent, while for other data sets of the same size the approximation will be poor. The natural measure of the effectiveness of the estimator can be expressed as its mean-square deviation from the desired optimal. Thus we average over all training sets $\mathcal{D}$ of fixed size n and find (Problem 18)

$$\mathcal{E}_{\mathcal{D}}\left[(g(\mathbf{x};\ \mathcal{D}) - F(\mathbf{x}))^2\right] = \underbrace{(\mathcal{E}_{\mathcal{D}}[g(\mathbf{x};\ \mathcal{D}) - F(\mathbf{x})])^2}_{bias^2} + \underbrace{\mathcal{E}_{\mathcal{D}}\left[(g(\mathbf{x};\ \mathcal{D}) - \mathcal{E}_{\mathcal{D}}[g(\mathbf{x};\ \mathcal{D})])^2\right]}_{variance}. \quad (11)$$

BIAS

VARIANCE

The first term on the right-hand side is the *bias* (squared)—the difference between the expected value and the true (but generally unknown) value—while the second term is the *variance*. Thus a low bias means that on average we will accurately estimate F from $\mathcal{D}$. Furthermore, a low variance means that the estimate of F does not change much as the training set varies. Even if an estimator is unbiased (i.e., the $bias = 0$ and its expected value is equal to the true value), there can nevertheless be a large mean-square error arising from a large variance term.

BIAS-VARIANCE DILEMMA

Equation 11 shows that the mean-square error can be expressed as the sum of a bias and a variance term. The *bias-variance dilemma* or *bias-variance trade-off* is a general phenomenon: Procedures with increased flexibility to adapt to the training data (e.g., have more free parameters) tend to have lower bias but higher variance. Different classes of regression functions $g(\mathbf{x};\ \mathcal{D})$—linear, quadratic, sum of Gaussians, and so on—will have different overall errors; nevertheless, Eq. 11 will be obeyed.

Suppose, for example, that the true, target function $F(x)$ is a cubic polynomial of one variable, with noise, as illustrated in Fig. 9.4. We seek to estimate this function based on a sampled training set $\mathcal{D}$. Column a shows a very poor "estimate" $g(x)$—a fixed linear function, *independent* of the training data. For different training sets sampled from $F(x)$ with noise, $g(x)$ is unchanged. The histogram of this mean-square error of Eq. 11, shown at the bottom, reveals a spike at a fairly high error; because this estimate is so poor, it has a high bias. Furthermore, the variance of the constant model $g(x)$ is zero. The model in column b is also fixed, but happens to be a better estimate of $F(x)$. It too has zero variance, but a lower bias than the poor model in column a. Presumably the designer imposed some prior knowledge about $F(x)$ in order to get this improved estimate.

The model in column c is a cubic with trainable coefficients; it would learn $F(x)$ exactly if $\mathcal{D}$ contained infinitely many training points. Notice that the fit found for every training set is quite good. Thus the bias is low, as shown in the distribution at the bottom. The model in column d is linear in x, but its slope and intercept are determined from the training data. As such, the model in column d has a lower bias than the models in columns a and b.

In sum, for a given target function $F(x)$, if a model has many parameters (generally low bias), it will fit the data well but yield high variance. Conversely, if the

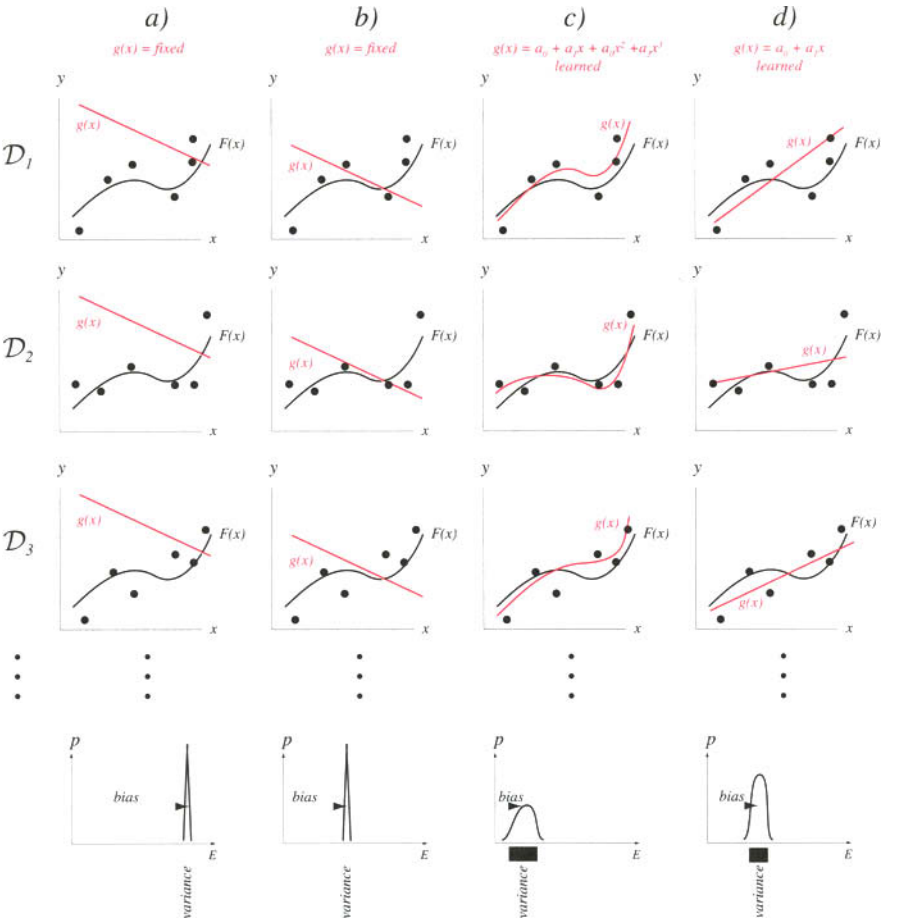


FIGURE 9.4. The bias-variance dilemma can be illustrated in the domain of regression. Each column represents a different model, and each row represents a different set of $n = 6$ training points, $\mathcal{D}_i$, randomly sampled from the true function $F(x)$ with noise. Probability functions of the mean-square error of $E = \mathcal{E}_{\mathcal{D}}[(g(x) - F(x))^2]$ of Eq. 11 are shown at the bottom. Column a shows a very poor model: a linear $g(x)$ whose parameters are held fixed, *independent* of the training data. This model has high bias and zero variance. Column b shows a somewhat better model, though it too is held fixed, independent of the training data. It has a lower bias than in column a and has the same zero variance. Column c shows a cubic model, where the parameters are trained to best fit the training samples in a mean-square-error sense. This model has low bias and a moderate variance. Column d shows a linear model that is adjusted to fit each training set; this model has intermediate bias and variance. If these models were instead trained with a very large number $n \rightarrow \infty$ of points, the bias in column c would approach a small value (which depends upon the noise), while the bias in column d would not; the variance of all models would approach zero.

model has few parameters (generally high bias), it may not fit the data particularly well, but this fit will not change much for different data sets (low variance). The best way to get low bias and low variance is the have prior information about the target function. We can virtually never get zero bias and zero variance; to do so would mean that there is only one learning problem to be solved, in which case the answer is already known. Furthermore, a large amount of training data will yield improved

performance so long as the model is sufficiently general to represent the target function. These considerations of bias and variance help to clarify the reasons we seek to have as much accurate prior information about the form of the solution, and as large a training set as feasible; the match of the algorithm to the problem is crucial.

9.3.2 Bias and Variance for Classification

While the bias-variance decomposition and dilemma are simplest to understand in the case of regression, we are most interested in their relevance to classification; here there are a few complications. In a two-category classification problem we let the target (discriminant) function have value 0 or +1, that is,

$$F(\mathbf{x}) = \Pr[y = 1|\mathbf{x}] = 1 - \Pr[y = 0|\mathbf{x}]. \tag{12}$$

On first consideration, the mean-square error we saw for *regression* (Eq. 11) does not appear to be the proper one for *classification*. After all, even if the mean-square error fit is poor, we can have accurate classification, possibly even the lowest (Bayes) error. This is because the decision rule under a zero-one loss selects the higher posterior $P(\omega_i|\mathbf{x})$, regardless of the *amount* by which it is higher. Nevertheless, by considering the expected value of y, we can recast classification into the framework of regression we saw before. To do so, we consider a discriminant function

$$y = F(\mathbf{x}) + \epsilon, \tag{13}$$

where ϵ is a zero-mean, random variable, for simplicity here assumed to be a centered binomial distribution with variance $\mathrm{Var}[\epsilon|\mathbf{x}] = F(\mathbf{x})(1 - F(\mathbf{x}))$. The target function can thus be expressed as

$$F(\mathbf{x}) = \mathcal{E}[y|\mathbf{x}], \tag{14}$$

and now the goal is to find an estimate $g(\mathbf{x}; \mathcal{D})$ that minimizes a mean-square error, such as in Eq. 11:

$$\mathcal{E}_{\mathcal{D}}[(g(\mathbf{x}; \mathcal{D}) - y)^2]. \tag{15}$$

In this way the regression methods of Section 9.3.1 can yield an estimate $g(\mathbf{x}; \mathcal{D})$ used for classification.

For simplicity we assume equal priors, $P(\omega_1) = P(\omega_2) = 0.5$, and thus the Bayes discriminant y_B has threshold 1/2 and the Bayes decision boundary is the set of points for which $F(\mathbf{x}) = 1/2$. For a given training set $\mathcal{D}$, if the classification error rate $\Pr[g(\mathbf{x}; \mathcal{D}) = y]$ averaged over predictions at $\mathbf{x}$ agrees with the Bayes discriminant,

$$\Pr[g(\mathbf{x}; \mathcal{D}) = y] = \Pr[y_B(\mathbf{x}) \neq y] = \min[F(\mathbf{x}), 1 - F(\mathbf{x})], \tag{16}$$

then indeed we have the lowest error. If not, then the prediction yields an increased error

$$\begin{aligned} \Pr[g(\mathbf{x}; \mathcal{D})] &= \max[F(\mathbf{x}), 1 - F(\mathbf{x})] \\ &= |2F(\mathbf{x}) - 1| + \Pr[y_B(\mathbf{x}) = y]. \end{aligned} \tag{17}$$

We average over all data sets of size n and find

$$\Pr[g(\mathbf{x}; \mathcal{D}) \neq y] = |2F(\mathbf{x}) - 1|\Pr[g(\mathbf{x}; \mathcal{D}) \neq y_B] + \Pr[y_B \neq y]. \tag{18}$$

BOUNDARY ERROR

Equation 18 shows that classification error rate is linearly proportional to $\Pr[g(\mathbf{x};\ \mathcal{D}) \neq y_B]$, which can be considered a *boundary error* in that it represents the incorrect estimation of the optimal (Bayes) boundary (Problem 19).

Because of random variations in training sets, the boundary error will depend upon $p(g(\mathbf{x};\ \mathcal{D}))$, the probability density of obtaining a particular estimate of the discriminant given $\mathcal{D}$. This error is merely the area of the tail of $p(g(\mathbf{x};\ \mathcal{D}))$ on the opposite side of the Bayes discriminant value 1/2, much as we saw in Chapter 2:

$$\Pr[g(\mathbf{x};\ \mathcal{D}) \neq y_B] = \begin{cases} \int_{1/2}^{\infty} p(g(\mathbf{x};\ \mathcal{D}))\, dg & \text{if } F(\mathbf{x}) < 1/2 \\ \int_{-\infty}^{1/2} p(g(\mathbf{x};\ \mathcal{D}))\, dg & \text{if } F(\mathbf{x}) \geq 1/2. \end{cases} \tag{19}$$

If we make the convenient assumption that $p(g(\mathbf{x};\ \mathcal{D}))$ is a Gaussian, we find (Problem 20)

$$\begin{aligned} \Pr[g(\mathbf{x};\ \mathcal{D}) \neq y_B] &= \Phi\left[\mathrm{Sgn}[F(\mathbf{x}) - 1/2]\frac{\mathcal{E}_{\mathcal{D}}[g(\mathbf{x};\ \mathcal{D})] - 1/2}{\sqrt{\mathrm{Var}[g(\mathbf{x};\ \mathcal{D})]}}\right] \\ &= \Phi\Big[\underbrace{\mathrm{Sgn}[F(\mathbf{x}) - 1/2][\mathcal{E}_{\mathcal{D}}[g(\mathbf{x};\ \mathcal{D})] - 1/2]}_{\textit{boundary bias}}\ \underbrace{\mathrm{Var}[g(\mathbf{x};\ \mathcal{D})]^{-1/2}}_{\textit{variance}}\Big], \end{aligned} \tag{20}$$

where

$$\Phi[t] = \frac{1}{\sqrt{2\pi}}\int_{t}^{\infty} e^{-1/2u^2}\, du = \frac{1}{2}[1 - \mathrm{erf}(t/\sqrt{2})] \tag{21}$$

and erf[·] is the familiar error function (Appendix Section A.5).

BOUNDARY BIAS

We have expressed this boundary error in terms of a *boundary bias*, in analogy with the simple bias-variance relation in regression (Eq. 11). Equation 20 shows that the effect of the variance term on the boundary error is highly nonlinear and depends on the value of the boundary bias. Furthermore, when the variance is small, this effect is particularly sensitive to the sign of the bias. In regression the estimation error is *additive* in *bias*^{2} and *variance*, whereas for classification there is a nonlinear and *multiplicative* interaction. In classification the sign of the *boundary bias* affects the role of variance in the error. For this reason low *variance* is generally important for accurate classification, while low *boundary bias* need not be. Or said another way, in classification, *variance* generally dominates *bias*. In practical terms, this implies that we need not be particularly concerned if our estimation is biased, so long as the variance is kept low. Numerous specific methods of classifier adjustment—pruning neural networks or decision trees, varying the number of free parameters, and so on—affect the bias and variance of a classifier; in Section 9.5 we shall discuss some methods applicable to a broad range of classification methods. Much as we saw in the bias-variance dilemma for regression, classification procedures with increased flexibility to adapt to the training data (e.g., have more free parameters) tend to have lower bias but higher variance.

As an illustration of *boundary bias* and *variance* in classifiers, consider a simple two-class problem in which samples are drawn from two-dimensional Gaussian distributions, each parameterized by vectors $p(\mathbf{x}|\omega_i) \sim N(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, for $i = 1, 2$. Here the true distributions have diagonal covariances, as shown at the top of Fig. 9.5. We

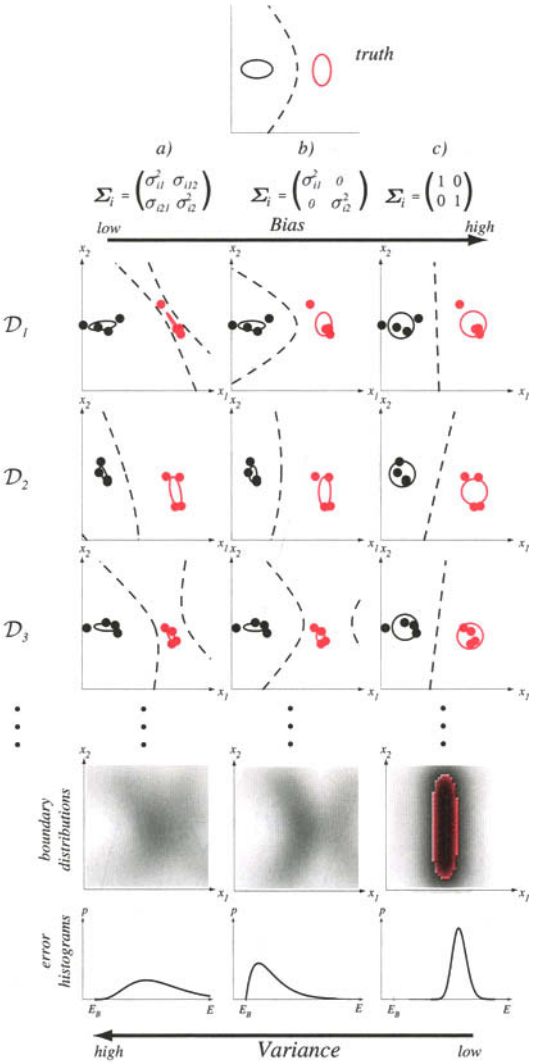


FIGURE 9.5. The (boundary) bias-variance trade-off in classification is illustrated with a two-dimensional Gaussian problem. The figure at the top shows the true distributions and the Bayes decision boundary. The nine figures in the middle show different learned decision boundaries. Each row corresponds to a different training set of $n = 8$ points selected randomly from the true distributions and labeled according to the true decision boundary. Column a shows case of a Gaussian model with fully general covariance matrices trained by maximum-likelihood. The learned boundaries differ significantly from one data set to the next; this learning algorithm has high variance. Column b shows the decision boundaries resulting from fitting a Gaussian model with diagonal covariances; in this case the decision boundaries vary less from one row to another. This learning algorithm has a lower variance than the one at the left. Finally, column c shows decision boundaries learning by fitting a Gaussian model with unit covariances (i.e., a linear model); notice that the decision boundaries are nearly identical from one data set to the next. This algorithm has low variance.

have just a few samples from each category and estimate the parameters in three different classes of models by maximum-likelihood. Column a shows the most general Gaussian classifiers; each component distribution can have arbitrary covariance matrix. Column b shows classifiers where each component Gaussian is constrained to have a diagonal covariance. Column c shows the most restrictive model: The covariances are equal to the identity matrix, yielding circular Gaussian distributions. Thus the left column corresponds to very low bias, and the right column corresponds to high bias.

Each row in Fig. 9.5 represents a different training set, randomly selected from the true distribution (shown at the top), and the resulting classifiers. Notice that most feature points in the high-bias cases retain their classification, regardless of the particular training set (i.e., such models have low variance), whereas the classification of a much larger range of points varies in the low-bias case (i.e., there is high variance). While in general a lower bias comes at the expense of higher variance, the relationship is nonlinear and multiplicative.

At the bottom of the figure, three density plots show how the location of the decision boundary varies across many different training sets. The leftmost density plot shows a very broad distribution (high variance). The rightmost plot shows a narrow, peaked distribution (low variance). To visualize the bias, imagine taking the spatial average of the decision boundaries obtained by running the learning algorithm on all possible data sets. The average of such boundaries for the leftmost algorithm will be equal to the true decision boundary—this algorithm has no bias. The rightmost average will be a vertical line, and hence there will be higher error—this algorithm has the highest bias of the three. Distributions of the generalization error are shown along the bottom.

For a given bias, the variance will decrease as n is increased. Naturally, if we had trained using a very large training set ($n \rightarrow \infty$), all error distributions become narrower and move to lower values of E. If a model is rich enough to express the optimal decision boundary, its error distribution for the large n case will approach a delta function at $E = E_B$, the Bayes error. As mentioned, to achieve the desired low generalization error it is more important to have low variance than to have low bias. The only way to get the ideal of zero bias and zero variance is to know the true model ahead of time (or be astoundingly lucky and guess it), in which case no learning was needed anyway. Bias and variance can be lowered with large training size n and accurate prior knowledge of the form of $F(\mathbf{x})$. Furthermore, as n grows, more parameters must be added to the model, g, so the data can be fit (reducing bias). For best classification based on a finite training set, it is desirable to match the form of the model to that of the (unknown) true distributions; this usually requires prior knowledge.

9.4 RESAMPLING FOR ESTIMATING STATISTICS

When we apply some learning algorithm to a new pattern recognition problem with unknown distribution, how can we determine the bias and variance? Figures 9.4 and 9.5 suggest a method using multiple samples, an inspiration for formal "resampling" methods, which we now discuss. Later we shall turn to our ultimate goal: using resampling and related techniques to improve classification (Section 9.5).

9.4.1 Jackknife

We begin with an example of how resampling can be used to yield a more informative estimate of a general statistic. Suppose we have a set $\mathcal{D}$ of n data points x_i ($i = 1, \ldots, n$), sampled from a one-dimensional distribution. The familiar estimate of the mean is, of course,

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i. \tag{22}$$

Likewise the estimate of the accuracy of the mean is the sample variance, given by

$$\hat{\sigma}^2 = \frac{(n-1)}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2. \tag{23}$$

MEDIAN

Suppose we were instead interested in the *median*, the point for which half of the distribution is higher, half lower. Although we could determine the median explicitly, there does not seem to be a straightforward way to generalize Eq. 23 to give a measure of the *error* or spread of our estimate of the median. The same difficulty applies to estimating the *mode* (the most frequently represented point in a data set), the 25th percentile, or any of a large number of statistics other than the mean. The jackknife* and bootstrap (Section 9.4.2) are two of the most popular and theoretically grounded resampling techniques for extending the above approach (based on Eqs. 22 and 23) to *arbitrary* statistics, of which the mean is just one instance.

MODE

In resampling theory, we frequently use statistics in which a data point is eliminated from the data; we denote this by means of a special subscript. For instance, the *leave-one-out* mean is

LEAVE-ONE-OUT MEAN

$$\mu_{(i)} = \frac{1}{n-1}\sum_{j\neq i}^{n} x_j = \frac{n\bar{x} - x_i}{n-1}, \tag{24}$$

that is, the sample average of the data set if the ith point is deleted. Next we define the jackknife estimate of the mean to be

$$\mu_{(\cdot)} = \frac{1}{n}\sum_{i=1}^{n}\mu_{(i)}, \tag{25}$$

that is, the mean of the leave-one-out means. It is simple to prove that the traditional estimate of the mean and the jackknife estimate of the mean are the same, that is, $\hat{\mu} = \mu_{(\cdot)}$ (Problem 23). Likewise, the jackknife estimate of the variance of the estimate obeys

$$\mathrm{Var}[\hat{\mu}] = \frac{n-1}{n}\sum_{i=1}^{n}(\mu_{(i)} - \mu_{(\cdot)})^2 \tag{26}$$

and, applied to the mean, is equivalent to the traditional variance of Eq. 23 (Problem 26).

*The jackknife method, which also goes by the name of "leave one out," was due to Maurice Quenouille. The playful name was chosen by John W. Tukey to capture the impression that the method was handy and useful in lots of ways.

The benefit of expressing the variance in the form of Eq. 26 is that it can be generalized to any other estimator $\hat{\theta}$, such as the median or 25th percentile or mode. To do so we need to compute the statistic with one data point "left out." Thus we let

$$\hat{\theta}_{(i)} = \hat{\theta}(x_1, x_2, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n) \tag{27}$$

take the place of $\mu_{(i)}$ and let $\hat{\theta}_{(\cdot)}$ take the place of $\mu_{(\cdot)}$ in Eqs. 25 and 26 above.

Jackknife Bias Estimate

BIAS

The notion of bias is more general than that described in Section 9.3; in fact it can be applied to the estimation of any statistic. The *bias* of an estimator θ is the difference between its true value and its expected value, that is,

$$bias = \theta - \mathcal{E}[\hat{\theta}]. \tag{28}$$

The jackknife method can be used estimate such a bias. The procedure is first to sequentially delete points x_i one at a time from $\mathcal{D}$ and compute the estimate $\hat{\theta}_{(\cdot)}$, where $\hat{\theta}_{(\cdot)} = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_{(i)}$. Then the jackknife estimate of the bias is (Problem 21)

$$bias_{jack} = (n-1)(\hat{\theta}_{(\cdot)} - \hat{\theta}). \tag{29}$$

We rearrange terms and thus see that the jackknife estimate of $\hat{\theta}$ is

$$\tilde{\theta} = \hat{\theta} - bias_{jack} = n\hat{\theta} - (n-1)\hat{\theta}_{(\cdot)}. \tag{30}$$

The benefit of using Eq. 30 is that it is unbiased for estimating the true bias (Problem 25).

Jackknife Variance Estimate

Now we seek the jackknife estimate of the variance of an arbitrary statistic θ. First, recall that the traditional variance is defined as

$$\mathrm{Var}[\hat{\theta}] = \mathcal{E}\left[[\hat{\theta}(x_1, x_2, \ldots, x_n) - \mathcal{E}[\hat{\theta}]]^2\right]. \tag{31}$$

The jackknife estimate of the variance, defined by analogy to Eq. 26, is

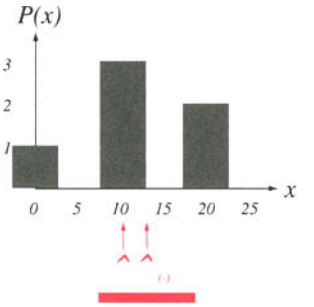
$$\mathrm{Var}_{jack}[\hat{\theta}] = \frac{n-1}{n}\sum_{i=1}^{n}[\hat{\theta}_{(i)} - \hat{\theta}_{(\cdot)}]^2. \tag{32}$$

EXAMPLE 2 Jackknife Estimate of Bias and Variance of the Mode

Consider an elementary example where we are interested in the mode of the following $n = 6$ points: $\mathcal{D} = \{0, 10, 10, 10, 20, 20\}$. It is clear from the histogram that the most frequently represented point is $\hat{\theta} = 10$. The jackknife estimate of the mode is

$$\hat{\theta}_{(\cdot)} = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_{(i)} = \frac{1}{6}[10 + 15 + 15 + 15 + 10 + 10] = 12.5,$$

where for $i = 2, 3, 4$ we used the fact that the mode of a distribution having two equal peaks is the point midway between those peaks. The fact that $\hat{\theta}_{(\cdot)} > \hat{\theta}$ reveals immediately that the jackknife estimate takes into account more of the full (skewed) distribution than does the standard mode calculation.



A histogram of $n = 6$ points whose mode is $\hat{\theta} = 10$ and whose jackknife estimate of the mode is $\hat{\theta}_{(\cdot)} = 12.5$. The square root of the jackknife estimate of the variance is a natural measure of the spread of probable values of the mode. This spread is indicated by the horizontal red bar.

The jackknife estimate of the bias of the estimate of the mode is given by Eq. 29:

$$bias_{jack} = (n-1)(\hat{\theta}_{(\cdot)} - \hat{\theta}) = 5(12.5 - 10) = 12.5.$$

Likewise, the jackknife estimate of the variance is given by Eq. 32:

$$\begin{aligned}\mathrm{Var}_{jack}[\hat{\theta}] &= \frac{n-1}{n}\sum_{i=1}^{n}(\hat{\theta}_{(i)} - \hat{\theta}_{(\cdot)})^2 \\ &= \frac{5}{6}[(10-12.5)^2 + 3(15-12.5)^2 + 2(10-12.5)^2] = 31.25.\end{aligned}$$

The square root of this variance, $\sqrt{31.25} \simeq 5.6$, serves as an effective standard deviation. A red bar of twice this width, shown below the histogram, reveals that the traditional mode lies within this tolerance to the jackknife estimate of the mode.

The jackknife resampling technique often gives us a more satisfactory estimate of a statistic such as the mode than do traditional methods, though it is more computationally complex (Problem 27).

9.4.2 Bootstrap

A "bootstrap" data set is one created by randomly selecting n points from the training set $\mathcal{D}$, with replacement. (Because $\mathcal{D}$ itself contains n points, there is nearly always duplication of individual points in a bootstrap data set.) In bootstrap estimation,* this selection process is independently repeated B times to yield B bootstrap data

*"Bootstrap" comes from subsequent versions of Rudolf Erich Raspe's wonderful stories "The adventures of Baron Munchhausen," in which the hero could pull himself up onto his horse by lifting his own bootstraps. A different but more common usage of the term applies to starting a computer, which must first run a program before it can run other programs.

sets, which are treated as independent sets. The bootstrap estimate of a statistic θ, denoted $\hat{\theta}^{*(\cdot)}$, is merely the mean of the B estimates on the individual bootstrap data sets:

$$\hat{\theta}^{*(\cdot)} = \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}^{*(b)}, \tag{33}$$

where $\hat{\theta}^{*(b)}$ is the estimate on bootstrap sample b.

Bootstrap Bias Estimate

The bootstrap estimate of the bias is (Problem 28)

$$bias_{boot} = \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}^{*(b)} - \hat{\theta} = \hat{\theta}^{*(\cdot)} - \hat{\theta}. \tag{34}$$

TRIMMED MEAN

Computer exercise 3 shows how the bootstrap can be applied to statistics that resist computational analysis, such as the "trimmed mean," in which the mean is calculated for a distribution in which some percentage (e.g., 5%) of the high and the low points in a distribution have been eliminated.

Bootstrap Variance Estimate

The bootstrap estimate of the variance is

$$\mathrm{Var}_{boot}[\theta] = \frac{1}{B}\sum_{b=1}^{B}\left[\hat{\theta}^{*(b)} - \hat{\theta}^{*(\cdot)}\right]^2. \tag{35}$$

If the statistic θ is the mean, then in the limit of $B \to \infty$, the bootstrap estimate of the variance is the traditional variance of the mean (Problem 22). Generally speaking, the larger the number B of bootstrap samples, the more satisfactory is the estimate of a statistic and its variance. One of the benefits of bootstrap estimation is that B can be adjusted to the computational resources; if powerful computers are available for a long time, then B can be chosen large. In contrast, a jackknife estimate requires exactly n repetitions: Fewer repetitions gives a poorer estimate that depends upon the random points chosen; more repetitions merely duplicates information already provided by some of the first n leave-one-out calculations.

9.5 RESAMPLING FOR CLASSIFIER DESIGN

The previous section addressed the use of resampling in estimating statistics, including the accuracy of an existing classifier, but only indirectly referred to the design of classifiers themselves. We now turn to a number of general resampling methods that have proven effective when used in conjunction with any in a wide range of techniques for training classifiers. These are related to methods for estimating and comparing classifier models that we will discuss in Section 9.6.

9.5.1 Bagging

ARCING

The generic term *arcing*—adaptive reweighting and combining—refers to reusing or selecting data in order to improve classification. In Section 9.5.2 we shall consider

the most popular arcing procedure, AdaBoost, but first we discuss briefly one of the simplest. Bagging—a name derived from "bootstrap aggregation"—uses multiple versions of a training set, each created by drawing $n' < n$ samples from $\mathcal{D}$ with replacement. Each of these bootstrap data sets is used to train a different *component classifier* and the final classification decision is based on the vote of each component classifier.* Traditionally the component classifiers are of the same general form—for example, all hidden Markov models, or all neural networks, or all decision trees—merely the final parameter values differ among them due to their different sets of training patterns.

COMPONENT CLASSIFIER

INSTABILITY

A classifier/learning algorithm combination is informally called *unstable* if "small" changes in the training data lead to significantly different classifiers and relatively "large" changes in accuracy. As we saw in Chapter 8, decision tree classifiers trained by a greedy algorithm can be unstable—a slight change in the position of a single training point can lead to a radically different tree. In general, bagging improves recognition for unstable classifiers because it effectively averages over such discontinuities. There are no convincing theoretical derivations or simulation studies showing that bagging will help all unstable classifiers, however.

Bagging is our first encounter with multiclassifier systems, where a final overall classifier is based on the outputs of a number of component classifiers. The global decision rule in bagging—a simple vote among the component classifiers—is the most elementary method of pooling or integrating the outputs of the component classifiers. We shall consider multiclassifier systems again in Section 9.7, with particular attention to forming a single decision rule from the outputs of the component classifiers.

9.5.2 Boosting

The goal of boosting is to improve the accuracy of any given learning algorithm. In boosting we first create a classifier with accuracy on the training set greater than average, and then add new component classifiers to form an ensemble whose joint decision rule has arbitrarily high accuracy on the training set. In such a case we say that the classification performance has been "boosted." In overview, the technique trains successive component classifiers with a subset of the training data that is "most informative" given the current set of component classifiers. Classification of a test point $\mathbf{x}$ is based on the outputs of the component classifiers, as we shall see.

For definiteness, consider creating three component classifiers for a two-category problem through boosting. First we randomly select a set of $n_1 < n$ patterns from the full training set $\mathcal{D}$ (without replacement); call this set $\mathcal{D}_1$. Then we train the first classifier, C_1, with $\mathcal{D}_1$. Classifier C_1 need only be a *weak learner*—that is, have accuracy only slightly better than chance. (Of course, this is the minimum requirement; a weak learner could have high accuracy on the training set. In that case the benefit of boosting will be small.) Now we seek a second training set, $\mathcal{D}_2$, that is the "most informative" given component classifier C_1. Specifically, half of the patterns in $\mathcal{D}_2$ should be correctly classified by C_1, half incorrectly classified by C_1 (Problem 30). Such an informative set $\mathcal{D}_2$ is created as follows: We flip a fair coin. If the coin is heads, we select remaining samples from $\mathcal{D}$ and present them, one by one to C_1 until C_1 misclassifies a pattern. We add this misclassified pattern to $\mathcal{D}_2$. Next we flip the

WEAK LEARNER

*In Section 9.7 we shall come across other names for component classifiers. For the present purposes we simply note that these are not classifiers of *component features*, but are instead members in an ensemble of classifiers whose outputs are pooled so as to implement a single classification rule.

coin again. If heads, we continue through $\mathcal{D}$ to find another pattern misclassified by C_1 and add it to $\mathcal{D}_2$ as just described; if tails, we find a pattern that C_1 classifies correctly. We continue until no more patterns can be added in this manner. Thus half of the patterns in $\mathcal{D}_2$ are correctly classified by C_1, half are not. As such, $\mathcal{D}_2$ provides information complementary to that represented in C_1. Now we train a second component classifier C_2 with $\mathcal{D}_2$.

Next we seek a third data set, $\mathcal{D}_3$, which is not well classified by voting by C_1 and C_2. We randomly select a training pattern from those remaining in $\mathcal{D}$ and then classify that pattern with C_1 and with C_2. If C_1 and C_2 disagree, we add this pattern to the third training set $\mathcal{D}_3$; otherwise we ignore the pattern. We continue adding informative patterns to $\mathcal{D}_3$ in this way; thus $\mathcal{D}_3$ contains those not well represented by the combined decisions of C_1 and C_2. Finally, we train the last component classifier, C_3, with the patterns in $\mathcal{D}_3$.

Now consider the use of the ensemble of three trained component classifiers for classifying a test pattern $\mathbf{x}$. Classification is based on the votes of the component classifiers. Specifically, if C_1 and C_2 agree on the category label of $\mathbf{x}$, we use that label; if they disagree, then we use the label given by C_3 (Fig. 9.6).

We skipped over a practical detail in the boosting algorithm: how to choose the number of patterns n_1 to train the first component classifier. We would like the final system to be trained with *all* patterns in $\mathcal{D}$ of course; moreover, because the final decision is a simple vote among the component classifiers, we would like to have a

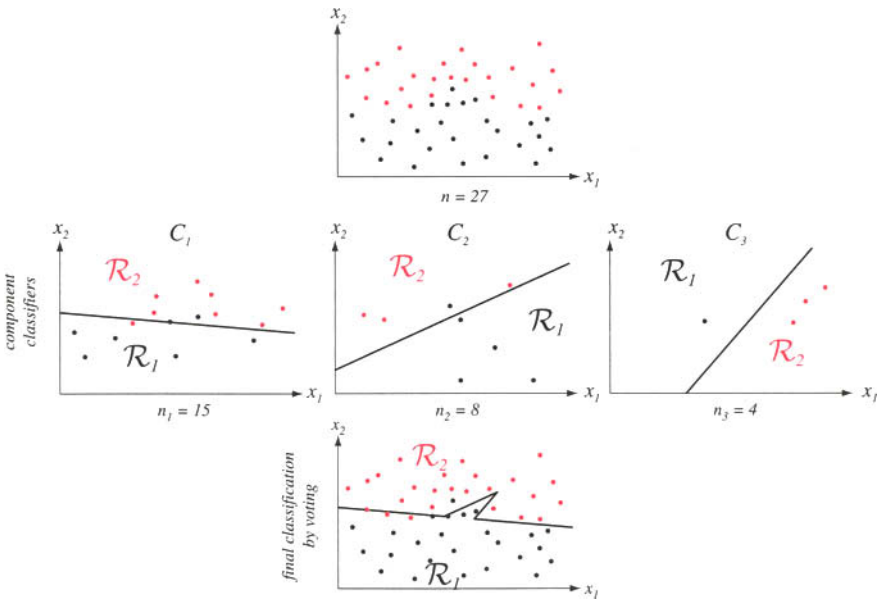


FIGURE 9.6. A two-dimensional two-category classification task is shown at the top. The middle row shows three component (linear) classifiers C_k trained by LMS algorithm (Chapter 5), where their training patterns were chosen through the basic boosting procedure. The final classification is given by the voting of the three component classifiers and yields a nonlinear decision boundary, as shown at the bottom. Given that the component classifiers are weak learners (i.e., each can learn a training set at least slightly better than chance), the ensemble classifier will have a lower training error on the full training set $\mathcal{D}$ than does any single component classifier. Of course, the ensemble classifier has lower error than a single linear classifier trained on the entire data set.

roughly equal number of patterns in each (i.e., $n_1 \simeq n_2 \simeq n_3 \simeq n/3$). A reasonable first guess is to set $n_1 \simeq n/3$ and create the three component classifiers. If the classification problem is very simple, however, component classifier C_1 will explain most of the data and thus n_2 (and n_3) will be much less than n_1, and not all of the patterns in the training set $\mathcal{D}$ will be used. Conversely, if the problem is extremely difficult, then C_1 will explain only a small amount of the data, and nearly all the patterns will be informative with respect to C_1; thus n_2 will be unacceptably large. Thus in practice we may need to run the overall boosting procedure a few times, adjusting n_1 in order to use the full training set and, if possible, get roughly equal partitions of the training set. A number of simple heuristics can be used to improve the partitioning of the training set as well (Computer exercise 6).

The above boosting procedure can be applied recursively to the component classifiers themselves, giving a 9-component or even 27-component full classifier. In this way, a very low training error can be achieved, even a vanishing training error if the problem is separable.

AdaBoost

There are a number of variations on basic boosting. The most popular, AdaBoost—from "adaptive boosting"—allows the designer to continue adding weak learners until some desired low training error has been achieved. In AdaBoost each training pattern receives a weight that determines its probability of being selected for a training set for an individual component classifier. If a training pattern is accurately classified, then its chance of being used again in a subsequent component classifier is reduced; conversely, if the pattern is not accurately classified, then its chance of being used again is raised. In this way, AdaBoost "focuses in" on the informative or "difficult" patterns. Specifically, we initialize the weights across the training set to be uniform. On each iteration k, we draw a training set at random according to these weights, and then we train component classifier C_k on the patterns selected. Next we increase weights of training patterns misclassified by C_k and decrease weights of the patterns correctly classified by C_k. Patterns chosen according to this new distribution are used to train the next classifier, C_{k+1}, and the process is iterated.

We again let the patterns and their labels in $\mathcal{D}$ be denoted $\mathbf{x}^i$ and y_i, respectively and let $W_k(i)$ be the kth (discrete) distribution over all these training samples. Thus the AdaBoost procedure is as follows:

■ **Algorithm 1. (AdaBoost)**

begin initialize $\mathcal{D} = \{\mathbf{x}^1, y_1, \ldots, \mathbf{x}^n, y_n\}$, k_{max}, $W_1(i) = 1/n, i = 1, \ldots, n$
  $k \leftarrow 0$
  do $k \leftarrow k + 1$
    train weak learner C_k using $\mathcal{D}$ sampled according to $W_k(i)$
    $E_k \leftarrow$ training error of C_k measured on $\mathcal{D}$ using $W_k(i)$
    $\alpha_k \leftarrow \frac{1}{2}\ln[(1 - E_k)/E_k]$
    $W_{k+1}(i) \leftarrow \frac{W_k(i)}{Z_k} \times \begin{cases} e^{-\alpha_k} & \text{if } h_k(\mathbf{x}^i) = y_i \text{ (correctly classified)} \\ e^{\alpha_k} & \text{if } h_k(\mathbf{x}^i) \neq y_i \text{ (incorrectly classified)} \end{cases}$
  until $k = k_{max}$
return C_k and α_k for $k = 1$ to k_{max} (ensemble of classifiers with weights)
end

Note that in line 5 the error for classifier C_k is determined with respect to the distribution $W_k(i)$ over $\mathcal{D}$ on which it was trained. In line 7, Z_k is simply a normalizing constant computed to ensure that $W_k(i)$ represents a true distribution, and $h_k(\mathbf{x}^i)$ is the category label ($+1$ or -1) given to pattern $\mathbf{x}^i$ by component classifier C_k. Naturally, the loop termination of line 8 could instead use the criterion of sufficiently low training error of the ensemble classifier.

The final classification decision of a test point $\mathbf{x}$ is based on a discriminant function that is merely the weighted sums of the outputs given by the component classifiers:

$$g(\mathbf{x}) = \left[\sum_{k=1}^{k_{max}} \alpha_k h_k(\mathbf{x})\right]. \tag{36}$$

The classification decision for this two-category case is then simply $\text{Sgn}[g(\mathbf{x})]$.

Except in pathological cases, so long as each component classifier is a weak learner, the total training error of the ensemble can be made arbitrarily low by setting the number of component classifiers, k_{max}, sufficiently high. To see this, notice that the training error for weak learner C_k can be written as $E_k = 1/2 - G_k$ for some positive value G_k. Thus the ensemble training error is (Problem 32)

$$E = \prod_{k=1}^{k_{max}} \left[2\sqrt{E_k(1-E_k)}\right] = \prod_{k=1}^{k_{max}} \sqrt{1-4G_k^2}$$

$$\leq \exp\left(-2\sum_{k=1}^{k_{max}} G_k^2\right), \tag{37}$$

as illustrated in Fig. 9.7. It is sometimes beneficial to increase k_{max} beyond the value needed for zero ensemble training error because this may improve generalization.

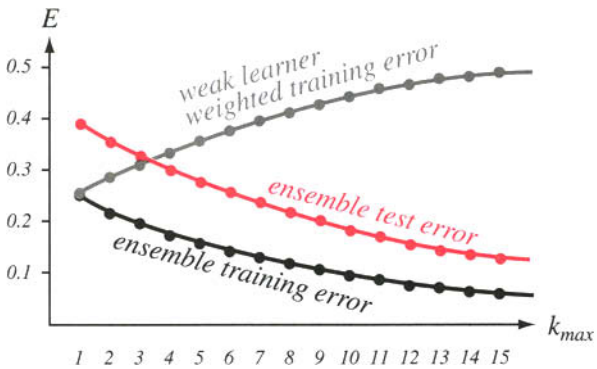


FIGURE 9.7. AdaBoost applied to a weak learning system can reduce the training error E exponentially as the number of component classifiers, k_{max}, is increased. Because AdaBoost "focuses on" *difficult* training patterns, the training error of each successive component classifier (measured on its own weighted training set) is generally larger than that of any previous component classifier (shown in gray). Nevertheless, so long as the component classifiers perform better than chance (e.g., have error less than 0.5 on a two-category problem), the weighted ensemble decision of Eq. 36 ensures that the training error will decrease, as given by Eq. 37. It is often found that the test error decreases in boosted systems as well, as shown in red.

While a large k_{max} could in principle lead to overfitting, simulation experiments have shown that overfitting rarely occurs, even when k_{max} is extremely large.

At first glance, it appears that boosting violates the No Free Lunch Theorem in that an ensemble classifier seems always to perform better than any single component classifier on the full training set. After all, according to Eq. 37 the training error drops exponentially fast with the number of component classifiers. The theorem is not violated, however: Boosting only improves classification if the component classifiers perform *better than chance*, but this cannot be guaranteed *a priori*. If the component classifiers cannot learn the task better than chance, then we do not have a strong match between the problem and model and should choose an alternative learning algorithm. Moreover, the exponential reduction in error on the training set does not ensure reduction of the *off-training set error* or generalization, as we saw in Section 9.2.1. Nevertheless, AdaBoost has proven effective in many real-world applications.

9.5.3 Learning with Queries

In the previous sections we assumed that there was a set of labeled training patterns $\mathcal{D}$ and employed resampling methods to reuse patterns to improve classification. In some applications, however, the patterns are *unlabeled*. In Chapter 10 we shall return to the problem of learning when no labels are available, but here we assume there exists some (possibly costly) way of labeling any pattern. Our current challenge is thus to determine which unlabeled patterns would be most informative (i.e., improve the classifier the most) if they were labeled and used as training patterns. These are the patterns we will present as a *query* to an *oracle*—a teacher who can label any pattern without error. This approach is called variously learning with queries, active learning, or interactive learning and is a special case of a resampling technique. A further refinement on this approach is *cost-based learning* in which there is a cost for obtaining new data. Here the task is to minimize an overall cost, which depends both on the classifier accuracy and the cost of data collection.

QUERY
ORACLE

COST-BASED LEARNING

Learning with queries might be appropriate, for example, when we want to design a classifier for handwritten numerals using unlabeled pixel images scanned from documents from a corpus too large for us to label every pattern. We could start by randomly selecting some patterns, presenting them to an oracle, and then training the classifier with the returned labels. We then use learning with queries to select unlabeled patterns from our set to present to a human (the oracle) for labeling. Informally, we would expect that the most valuable patterns would be near the decision boundaries.

More generally we begin with a preliminary, weak classifier that has been developed with a small set of labeled samples. There are two related methods for then selecting an informative pattern—that is, a pattern for which the current classifier is least certain. In *confidence-based query selection* the classifier computes discriminant functions $g_i(\mathbf{x})$ for the c categories, $i = 1, \ldots, c$. An informative pattern $\mathbf{x}$ is one for which the two largest discriminant functions have nearly the same value; such patterns lie near the current decision boundaries. Several search heuristics can be used to find such points efficiently (Problem 31).

CONFIDENCE BASED QUERY SELECTION

The second method, *voting-based* or *committee-based query selection*, is similar to the previous method but is applicable to multiclassifier systems—that is, ones comprising several component classifiers (Section 9.7). Each unlabeled pattern is presented to each of the k component classifiers; the pattern that yields the greatest disagreement among the k resulting category labels is considered the most infor-

VOTING-BASED QUERY SELECTION

mative pattern, and is thus presented as a query to the oracle. Voting-based query selection can be used even if the component classifiers do not provide analog discriminant functions—for instance, decision trees, rule-based classifiers, or simple k-nearest neighbor classifiers. In both confidence-based and voting-based methods, the pattern labeled by the oracle is then used for training the classifier in the traditional way. (We shall return in Section 9.7 to training an ensemble of classifiers.)

Clearly such learning with queries does not directly exploit information about the prior distribution of the patterns. In particular, in most problems the distributions of query patterns will be large near the final decision boundaries (where patterns are informative) rather than at the region of highest prior probability (where they are typically less informative), as illustrated in Fig. 9.8. One benefit of learning with queries is that we need not guess the form of the underlying distribution, but can instead use nonparametric techniques, such as nearest-neighbor classification, that allow the decision boundary to be found directly.

If the set of unlabeled samples for queries is not large, we can nevertheless exploit learning with queries if there is a way to *generate* query patterns. Suppose we have a only small set of labeled handwritten characters. Suppose too we have image processing algorithms for altering these images to generate new, surrogate patterns for queries to an oracle. For instance, the pixel images might be rotated, scaled, sheared, be subject to random pixel noise, or have their lines thinned. Furthermore, we might be able to generate new patterns "in between" two labeled patterns by interpolating

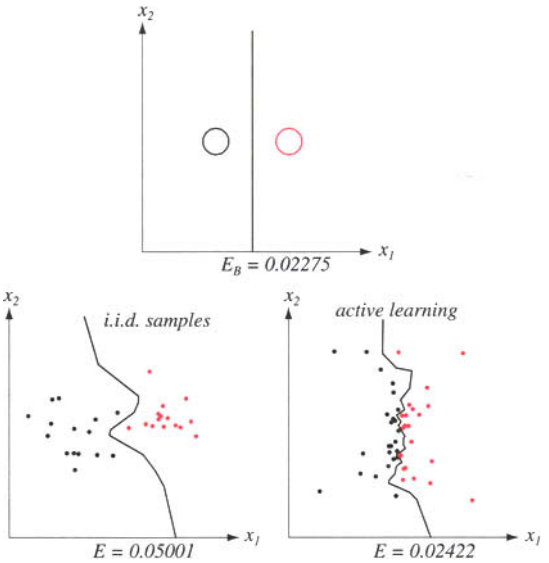


FIGURE 9.8. Active learning can be used to create classifiers that are more accurate than ones using i.i.d. sampling. The figure at the top shows a two-dimensional problem with two equal circular Gaussian priors; the Bayes decision boundary is a straight line and the Bayes error E_B equals 0.02275. The bottom figure on the left shows a nearest-neighbor classifier trained with $n = 30$ labeled points sampled i.i.d. from the true distributions. Note that most of these points are far from the decision boundary. The figure at the right illustrates active learning. The first four points were sampled near the extremes of the feature space. Subsequent query points were chosen midway between two points already used by the classifier, one randomly selected from each of the two categories. In this way, successive queries to the oracle "focused in" on the true decision boundary. The final generalization error of this classifier, $E = 0.02422$, is lower than the one trained using i.i.d. samples, $E = 0.05001$.

or somehow mixing them in a domain-specific way. With such generated query patterns the classifier can explore regions of the feature space about which it is least confident (Fig. 9.8).

9.5.4 Arcing, Learning with Queries, Bias and Variance

In Chapter 3 and many other places, we have stressed the need for training a classifier on samples drawn from the distribution on which it will be tested. Resampling in general—and learning with queries in particular—seems to violate this recommendation. Why can a classifier trained on a strongly weighted distribution of data be expected to do well—or better!—than one trained on the i.i.d. sample? Why doesn't resampling lead to *worse* performance, to the extent that the resampled distribution differs from the i.i.d. one?

Indeed, if we were to take a model of the true distribution and train it with a highly skewed distribution obtained by learning with queries, the final classifier accuracy might be unacceptably low. Consider, however, two interrelated points about resampling methods and altered distributions. The first is that resampling methods are generally used with techniques that do *not* attempt to model or fit the full category distributions. Thus even if we suspect we know that the prior distributions for two categories have a particular mathematical form, we might nevertheless use a nonparametric method such as nearest neighbor, radial basis function, or RCE classifiers when using learning with queries. Thus in learning with queries we are not fitting parameters in a model, as described in Chapter 3, but instead are seeking decision boundaries more directly.

The second point is that as the number of component classifiers is increased, techniques such as general boosting and AdaBoost effectively broaden that class of implementable functions, as illustrated in Fig. 9.6. While the final classifier might indeed be characterized as parametric, it is in an expanded space of parameters, one larger than that of the first component classifier.

In broad overview, resampling, boosting and related procedures are heuristic methods for adjusting the class of implementable decision functions. As such they allow the designer to try to "match" the final classifier to the problem by indirectly adjusting the bias and variance. The power of these methods is that they can be used with an arbitrary classification technique such as the two-layer Perceptron, which would otherwise prove extremely difficult to adjust to the complexity of an arbitrary problem.

9.6 ESTIMATING AND COMPARING CLASSIFIERS

There are at least two reasons for wanting to know the generalization rate of a classifier on a given problem. One is to see if the classifier performs well enough to be useful; another is to compare its performance with that of a competing design. Estimating the final generalization performance invariably requires making assumptions about the classifier or the problem or both, and can fail if the assumptions are not valid. We should stress, then, that all the following methods are heuristic. Indeed, if there were a foolproof method for choosing which of two classifiers would generalize better on an arbitrary new problem, we could incorporate such a method into the learning and violate the No Free Lunch Theorem. Occasionally our assumptions are explicit (as in parametric models), but more often than not they are implicit and difficult to identify or relate to the final estimation (as in empirical methods).

9.6.1 Parametric Models

One approach to estimating the generalization rate is to compute it from the assumed parametric model. For example, in the two-class multivariate normal case, we might estimate the probability of error using the Bhattacharyya or Chernoff bounds (Chapter 2), substituting estimates of the means and the covariance matrix for the unknown parameters. However, there are three problems with this approach. First, such an error estimate is often overly optimistic; characteristics that make the training samples peculiar or unrepresentative will not be revealed. Second, we should always suspect the validity of an assumed parametric model; a performance evaluation based on the same model cannot be believed unless the evaluation is unfavorable. Finally, in more general situations where the distributions are not simple, it is very difficult to compute the error rate exactly, even if the probabilistic structure is known completely.

9.6.2 Cross-Validation

VALIDATION SET

In simple validation we randomly split the set of labeled training samples $\mathcal{D}$ into two parts: one is used as the traditional training set for adjusting model parameters in the classifier. The other set—the *validation set*—is used to estimate the generalization error. Since our ultimate goal is low generalization error, we train the classifier until we reach a minimum of this validation error, as sketched in Fig. 9.9. It is essential that the validation (or the test) set not include points used for training the parameters in the classifier—a methodological error known as "testing on the training set." *

m-FOLD CROSS-VALIDATION

A simple generalization of the above method is *m-fold cross-validation*. Here the training set is randomly divided into m disjoint sets of equal size n/m, where n is

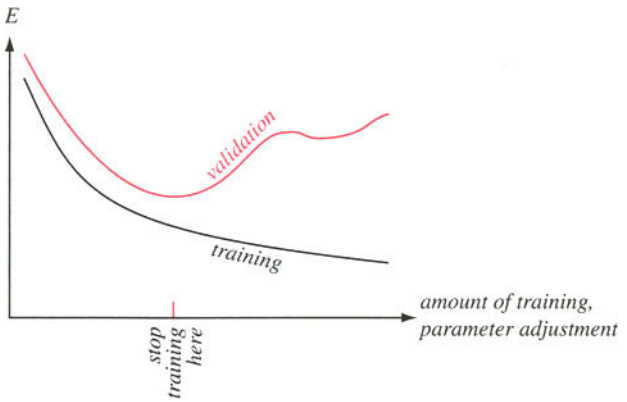


FIGURE 9.9. In validation, the data set $\mathcal{D}$ is split into two parts. The first (e.g., 90% of the patterns) is used as a standard training set for setting free parameters in the classifier model; the other (e.g., 10%) is the validation set and is meant to represent the full generalization task. For most problems, the training error decreases monotonically during training, as shown in black. Typically, the error on the validation set decreases, but then increases, an indication that the classifier may be overfitting the training data. In validation, training or parameter adjustment is stopped at the first minimum of the validation error. In the more general method of cross-validation, the performance is based on multiple independently formed validation sets.

*A related but less obvious problem arises when a classifier undergoes a long series of refinements guided by the results of repeated testing on the same test data. This form of "training on the test data" often escapes attention until new test samples are obtained.

again the total number of patterns in $\mathcal{D}$. The classifier is trained m times, each time with a different set held out as a validation set. The estimated performance is the mean of these m errors. In the limit where $m = n$, the method is in effect the leave-one-out approach to be discussed in Section 9.6.3.

Such techniques can be applied to virtually every classification method, where the specific form of learning or parameter adjustment depends upon the general training method. For example, in neural networks of a fixed topology (Chapter 6), the amount of training is the number of epochs or presentations of the training set. Alternatively, the number of hidden units can be set via cross-validation. Likewise, the width of the Gaussian window in Parzen windows (Chapter 4), and an optimal value of k in the k-nearest neighbor classifier can be set by validation or cross-validation.

Validation is heuristic and need not (indeed cannot) give improved classifiers in every case. Nevertheless, validation is extremely simple, and for many real-world problems it is found to improve generalization accuracy. There are several heuristics for choosing the portion γ of $\mathcal{D}$ to be used as a validation set ($0 < \gamma < 1$). Nearly always, a smaller portion of the data should be used as validation set ($\gamma < 0.5$) because the validation set is used merely to set a *single* global property of the classifier (i.e., when to stop adjusting parameters) rather than the large number of classifier parameters learned using the training set. If a classifier has a large number of free parameters or degrees of freedom, then a larger portion of $\mathcal{D}$ should be used as a training set, that is, γ should be reduced. A traditional default is to split the data with $\gamma = 0.1$, which has proven effective in many applications. Finally, when the number of degrees of freedom in the classifier is small compared to the number of training points, the predicted generalization error is relatively insensitive to the choice of γ.

ANTI-CROSS-VALIDATION

We reiterate that cross-validation is a heuristic and need not work on every problem. Indeed, there are problems for which *anti-cross-validation* is effective—halting the adjustment of parameters when the validation error is the first local *maximum*. As such, in any particular problem, designers must be prepared to explore different values of γ, and possibly abandon the use of cross-validation altogether if performance cannot be improved (Computer exercise 7).

Cross-validation is, at base, an empirical approach that tests the classifier experimentally. Once we train a classifier using cross-validation, the validation error gives an estimate of the accuracy of the final classifier on the unknown test set. If the true but unknown error rate of the classifier is p and if k of the n' independent, randomly drawn test samples are misclassified, then k has the binomial distribution

$$P(k) = \binom{n'}{k} p^k (1-p)^{n'-k}. \tag{38}$$

Thus, the fraction of test samples misclassified is exactly the maximum-likelihood estimate for p (Problem 40):

$$\hat{p} = \frac{k}{n'}. \tag{39}$$

The properties of this estimate for the parameter p of a binomial distribution are well known. In particular, Fig. 9.10 shows 95% confidence intervals as a function of $\hat{p}$ and n'. For a given value of $\hat{p}$, the probability is 0.95 that the true value of p lies in the interval between the lower and upper curves marked by the number n' of test samples (Problem 37). These curves show that unless n' is fairly large, the maximum-likelihood estimate must be interpreted with caution. For example, if no errors are

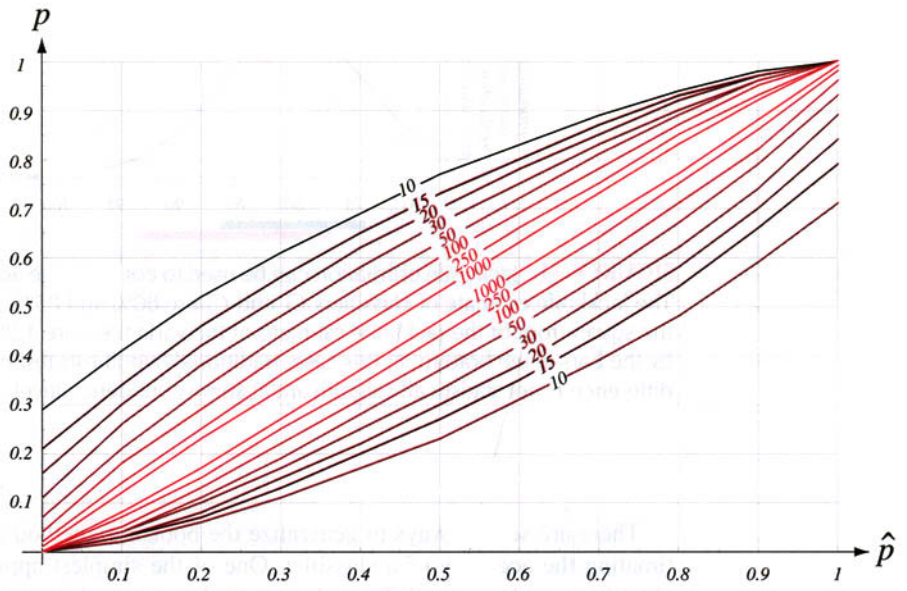


FIGURE 9.10. The 95% confidence intervals for a given estimated error probability $\hat{p}$ can be derived from a binomial distribution of Eq. 38. For each value of $\hat{p}$, the true probability has a 95% chance of lying between the curves marked by the number of test samples n'. The larger the number of test samples, the more precise the estimate of the true probability and hence the smaller the 95% confidence interval.

made on 50 test samples, with probability 0.95 the true error rate is between zero and 8%. The classifier would have to make no errors on more than 250 test samples to be reasonably sure that the true error rate is below 2%.

9.6.3 Jackknife and Bootstrap Estimation of Classification Accuracy

A method for comparing classifiers closely related to cross-validation is to use the jackknife or bootstrap estimation procedures (Sections 9.4.1 and 9.4.2). The application of the jackknife approach to classification is straightforward. We estimate the accuracy of a given algorithm by training the classifier n separate times, each time using the training set $\mathcal{D}$ from which a different single training point has been deleted. This is merely the $m = n$ limit of m-fold cross-validation. Each resulting classifier is tested on the single deleted point, and the jackknife estimate of the accuracy is then simply the mean of these leave-one-out accuracies. Here the computational complexity may be very high, especially for large n (Problem 29).

The jackknife, in particular, generally gives good estimates because each of the the n classifiers is quite similar to the classifier being tested (differing solely due to a single training point). Likewise, the jackknife estimate of the variance of this estimate is given by a simple generalization of Eq. 32. A particular benefit of the jackknife approach is that it can provide measures of confidence or statistical significance in the comparison between two classifier designs. Suppose that trained classifier C_1 has an accuracy of 80% while C_2 has accuracy of 85%, as estimated by the jackknife procedure. Is C_2 really better than C_1? To answer this, we calculate the jackknife estimate of the variance of the classification accuracies and use traditional hypothesis testing to see if C_1's apparent superiority is statistically significant (Fig. 9.11).

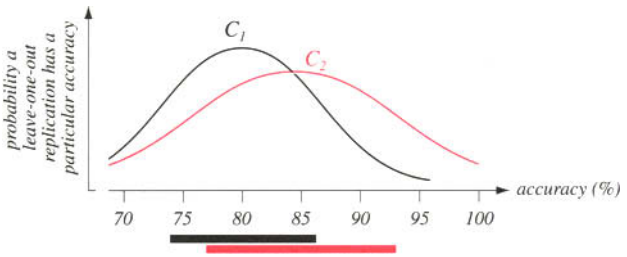


FIGURE 9.11. Jackknife estimation can be used to compare the accuracies of classifiers. The jackknife estimate of classifiers C_1 and C_2 are 80% and 85%, and full widths (twice the square root of the jackknife estimate of the variances) are 12% and 15%, as shown by the bars at the bottom. In this case, traditional hypothesis testing could show that the difference is not statistically significant at some confidence level.

There are several ways to generalize the bootstrap method to the problem of estimating the accuracy of a classifier. One of the simplest approaches is to train B classifiers, each with a different bootstrap data set, and test on other bootstrap data sets. The bootstrap estimate of the classifier accuracy is simply the mean of these bootstrap accuracies. In practice, the high computational complexity of bootstrap estimation of classifier accuracy is sometimes worth possible improvements in that estimate (Section 9.5.1).

9.6.4 Maximum-Likelihood Model Comparison

ML-II

Recall first the maximum-likelihood parameter estimation methods discussed in Chapter 3. Given a model with unknown parameter vector $\boldsymbol{\theta}$, we find the value $\hat{\boldsymbol{\theta}}$ that maximizes the probability of the training data, that is, $p(\mathcal{D}|\hat{\boldsymbol{\theta}})$. Maximum-likelihood *model comparison* or maximum-likelihood *model selection*—sometimes called ML-II—is a direct generalization of those techniques. The goal here is to choose the *model* that best explains the training data, in a way that will become clear below.

We again let $h_i \in \mathcal{H}$ represent a candidate hypothesis or model (assumed discrete for simplicity) and let $\mathcal{D}$ represent the training data. The posterior probability of any given model is given by Bayes rule:

$$P(h_i|\mathcal{D}) = \frac{P(\mathcal{D}|h_i)P(h_i)}{p(\mathcal{D})} \propto P(\mathcal{D}|h_i)P(h_i), \tag{40}$$

EVIDENCE

where we will rarely need the normalizing factor $p(\mathcal{D})$. The data-dependent term, $P(\mathcal{D}|h_i)$, is the *evidence* for h_i; the second factor, $P(h_i)$, is our subjective prior over the space of hypotheses—it rates our confidence in different models even before the data arrive. In practice, the data-dependent term dominates in Eq. 40, and hence the priors $P(h_i)$ are often neglected in the computation. In maximum-likelihood model comparison, we find the maximum-likelihood parameters for each of the candidate models, calculate the resulting likelihoods, and select the model with the largest such likelihood in Eq. 40 (Fig. 9.12).

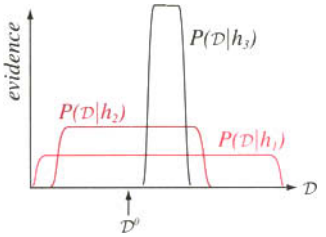


FIGURE 9.12. The evidence (i.e., probability of generating different data sets given a model) is shown for three models of different expressive power or complexity. Model h_1 is the most expressive, because with different values of its parameters the model can fit a wide range of data sets. Model h_3 is the most restrictive of the three. If the actual data observed is $\mathcal{D}^0$, then maximum-likelihood model selection states that we should choose h_2, which has the highest evidence. Model h_2 "matches" this particular data set better than do the other two models, and it should be selected.

9.6.5 Bayesian Model Comparison

Bayesian model comparison uses the full information over priors when computing posterior probabilities in Eq. 40. In particular, the evidence for a particular hypothesis is an integral,

$$P(\mathcal{D}|h_i) = \int p(\mathcal{D}|\boldsymbol{\theta}, h_i) p(\boldsymbol{\theta}|\mathcal{D}, h_i)\, d\boldsymbol{\theta}, \tag{41}$$

where, as before, $\boldsymbol{\theta}$ describes the parameters in the candidate model. It is common for the posterior $P(\boldsymbol{\theta}|\mathcal{D}, h_i)$ to be peaked at $\hat{\boldsymbol{\theta}}$, and thus the evidence integral can often be approximated as

$$P(\mathcal{D}|h_i) \simeq \underbrace{P(\mathcal{D}|\hat{\boldsymbol{\theta}}, h_i)}_{\substack{\text{best-fit} \\ \text{likelihood}}} \underbrace{p(\hat{\boldsymbol{\theta}}|h_i)\Delta\boldsymbol{\theta}}_{\text{Occam factor}}. \tag{42}$$

OCCAM FACTOR

Before the data arrive, model h_i has some broad range of model parameters, denoted by $\Delta^0\boldsymbol{\theta}$ and shown in Fig. 9.13. After the data arrive, a smaller range is commensurate or compatible with $\mathcal{D}$, denoted $\Delta\boldsymbol{\theta}$. The *Occam factor* in Eq. 42,

$$\begin{aligned} \text{Occam factor} &= p(\hat{\boldsymbol{\theta}}|h_i)\Delta\boldsymbol{\theta} = \frac{\Delta\boldsymbol{\theta}}{\Delta^0\boldsymbol{\theta}} \\ &= \frac{\text{param. vol. commensurate with } \mathcal{D}}{\text{param. vol. commensurate with any data}}, \end{aligned} \tag{43}$$

is the ratio of two volumes in parameter space: (1) the volume that can account for data $\mathcal{D}$ and (2) the prior volume, accessible to the model without regard to $\mathcal{D}$. The Occam factor has magnitude less than 1.0; it is simply the factor by which the hypothesis space collapses by the presence of data. The more the training data, the smaller the range of parameters that are commensurate with it, and thus the greater this collapse in the parameter space and the larger the Occam factor (Fig. 9.13).

Naturally, once the posteriors for different models have been calculated by Eqs. 40 and 42, we select the single one having the highest such posterior. (Ironically, the Bayesian model selection procedure is itself not truly Bayesian, because a Bayesian procedure would average over *all* possible models when making a decision.)

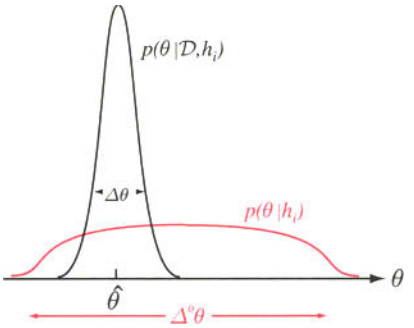


FIGURE 9.13. In the absence of training data, a particular model h_i has available a large range of possible values of its parameters, denoted $\Delta^0\theta$. In the presence of a particular training set $\mathcal{D}$, a smaller range is available. The Occam factor, $\Delta\theta/\Delta^0\theta$, measures the fractional decrease in the volume of the model's parameter space due to the presence of training data $\mathcal{D}$. In practice, the Occam factor can be calculated fairly easily if the evidence is approximated as a p-dimensional Gaussian, centered on the maximum-likelihood value $\hat{\boldsymbol{\theta}}$.

The evidence for h_i—that is, $P(\mathcal{D}|h_i)$—was ignored in a maximum-likelihood setting of parameters $\hat{\boldsymbol{\theta}}$; nevertheless, it is the central term in our comparison of models. As mentioned, in practice the evidence term in Eq. 40 dominates the prior term, and it is traditional to ignore such priors, which are often highly subjective or problematic anyway (Problem 39, Computer exercise 9). This procedure represents an inherent bias toward simple models (small $\Delta\boldsymbol{\theta}$); models that are overly complex (large $\Delta\boldsymbol{\theta}$) are automatically self-penalizing and "overly complex" is a data-dependent concept.

In the general case, the full integral of Eq. 41 is too difficult to calculate analytically or even numerically. Nevertheless, if $\boldsymbol{\theta}$ is p-dimensional and the posterior can be assumed to be a Gaussian, then the Occam factor can be calculated directly (Problem 38), yielding

$$P(\mathcal{D}|h_i) \simeq \underbrace{P(\mathcal{D}|\hat{\boldsymbol{\theta}}, h_i)}_{\substack{\text{best-fit}\\ \text{likelihood}}} \underbrace{p(\hat{\boldsymbol{\theta}}|h_i)(2\pi)^{k/2}|\mathbf{H}|^{-1/2}}_{\text{Occam factor}}, \tag{44}$$

where

$$\mathbf{H} = \frac{\partial^2 \ln p(\boldsymbol{\theta}|\mathcal{D}, h_i)}{\partial \boldsymbol{\theta}^2} \tag{45}$$

is a Hessian matrix—a matrix of second-order derivatives—and measures how "peaked" the posterior is around the value $\hat{\boldsymbol{\theta}}$. Note that this Gaussian approximation does not rely on the fact that the underlying model of the distribution of the *data* in feature space is or is not Gaussian. Rather, it is based on the assumption that the evidence distribution arises from a large number of independent uncorrelated processes and is governed by the Law of Large Numbers. The integration inherent in Bayesian methods is simplified using this Gaussian approximation to the evidence. Because calculating the needed Hessian via differentiation is nearly always simpler than a high-dimensional numerical integration, the Bayesian method of model selection

is not at a severe computational disadvantage relative to its maximum-likelihood counterpart.

There may be a problem due to degeneracies in a model: Several parameters could be relabeled and leave the classification rule (and hence the likelihood) unchanged. The resulting degeneracy leads, in essence, to an "overcounting" that alters the effective volume in parameter space. Degeneracies are especially common in neural network models where the parameterization comprises many equivalent weights (Chapter 6). For such cases, we must multiply the right-hand side of Eq. 42 by the degeneracy of $\hat{\boldsymbol{\theta}}$ in order to scale the Occam factor, and thereby obtain the proper estimate of the evidence (Problem 43).

Bayesian Model Selection and the No Free Lunch Theorem

There seems to be a fundamental contradiction between two of the deepest ideas in the foundation of statistical pattern recognition. On the one hand, the No Free Lunch Theorem states that in the absence of prior information about the problem, there is no reason to prefer one classification algorithm over another. On the other hand, Bayesian model selection is theoretically well-founded and seems to show how to reliably choose the better of two algorithms.

Consider two "composite" algorithms—algorithm A and algorithm B—each of which employs two others (algorithm 1 and algorithm 2). For any problem, algorithm A uses Bayesian model selection and applies the "better" of algorithm 1 and algorithm 2. Algorithm B uses *anti*-Bayesian model selection and applies the "worse" of algorithm 1 and algorithm 2. It appears that algorithm A will reliably outperform algorithm B throughout the full class of problems—in contradiction with Part 1 of the No Free Lunch Theorem.

What is the resolution of this apparent contradiction? In Bayesian model selection we ignore the prior over the space of models, $\mathcal{H}$, effectively assuming that it is uniform. This assumption therefore does not take into account how those models correspond to underlying target functions—that is, mappings from input to category labels. Accordingly, Bayesian model selection usually corresponds to a nonuniform prior over target functions. Moreover, depending on the arbitrary choice of model, the precise nonuniform prior will vary. In fact, this arbitrariness is very well known in statistics, and good practitioners rarely apply the *principle of indifference*, assuming a uniform prior over models, as Bayesian model selection requires. Indeed, there are many "paradoxes" described in the statistics literature that arise from not being careful to have the prior over models be tailored to the choice of models (Problem 39). The No Free Lunch Theorem allows that for some particular nonuniform prior there may be a learning algorithm that gives better than chance—or even optimal—results. Apparently Bayesian model selection corresponds to nonuniform priors that seem to match many important real-world problems.

PRINCIPLE OF INDIFFERENCE

9.6.6 The Problem-Average Error Rate

The examples we have given thus far suggest that the problem with having only a small number of samples is that the resulting classifier will not perform well on new data—it will not generalize well. Thus, we expect the error rate to be a function of the number n of training samples, typically decreasing to some minimum value as n approaches infinity. To investigate this analytically, we must carry out the following familiar steps:

1. Estimate the unknown parameters from samples.
2. Use these estimates to determine the classifier.
3. Calculate the error rate for the resulting classifier.

In general this analysis is very complicated. The answer depends on everything—on the particular training patterns, on the way they are used to determine the classifier, and on the unknown, underlying probability structure. However, by using histogram approximations to the unknown probability densities and averaging appropriately, it is possible to draw some illuminating conclusions.

Consider a case in which two categories have equal prior probabilities. Suppose that we partition the feature space into some number m of disjoint cells $\mathcal{C}_1, \ldots, \mathcal{C}_m$. If the conditional densities $p(\mathbf{x}|\omega_1)$ and $p(\mathbf{x}|\omega_2)$ do not vary appreciably within any cell, then instead of needing to know the actual value of $\mathbf{x}$, we need only know into which cell $\mathbf{x}$ falls. This reduces the problem to the discrete case. Let $p_i = P(\mathbf{x} \in \mathcal{C}_i|\omega_1)$ and $q_i = P(\mathbf{x} \in \mathcal{C}_i|\omega_2)$. Then, because we have assumed that $P(\omega_1) = P(\omega_2) = 1/2$, the vectors $\mathbf{p} = (p_1, \ldots, p_m)^t$ and $\mathbf{q} = (q_1, \ldots, q_m)^t$ determine the probability structure of the problem. If $\mathbf{x}$ falls in $\mathcal{C}_i$, the Bayes decision rule is to decide ω_i if $p_i > q_i$. The resulting Bayes error rate is given by

$$P(E|\mathbf{p}, \mathbf{q}) = \frac{1}{2}\sum_{i=1}^{m} \min[p_i, q_i]. \tag{46}$$

When the parameters $\mathbf{p}$ and $\mathbf{q}$ are unknown and must be estimated from a set of training patterns, the resulting error rate will be larger than the Bayes rate. The exact error probability will depend on the set of training patterns and the way in which they are used to obtain the classifier. Suppose that half of the samples are labeled ω_1 and half are labeled ω_2, with n_{ij} being the number that fall in $\mathcal{C}_i$ and are labeled ω_j. Suppose further that we design the classifier by using the maximum-likelihood estimates $\hat{p}_i = 2n_{i1}/n$ and $\hat{q}_i = 2n_{i2}/n$ as if they were the true values. Then a new feature vector falling in $\mathcal{C}_i$ will be assigned to ω_1 if $n_{i1} > n_{i2}$. With all of these assumptions, it follows that the probability of error for the resulting classifier is given by

$$P(E|\mathbf{p}, \mathbf{q}, \mathcal{D}) = \frac{1}{2}\sum_{n_{i1}>n_{i2}} q_i + \frac{1}{2}\sum_{n_{i1}\le n_{i2}} p_i. \tag{47}$$

To evaluate this probability of error, we need to know the true conditional probabilities $\mathbf{p}$ and $\mathbf{q}$ and the set of training patterns, or at least the numbers n_{ij}. Different sets of n randomly chosen patterns will yield different values for $P(E|\mathbf{p}, \mathbf{q}, \mathcal{D})$. We can use the fact that the numbers n_{ij} have a multinomial distribution to average over all of the possible sets of n random samples and obtain an average probability of error $P(E|\mathbf{p}, \mathbf{q}, n)$. Roughly speaking, this is the typical error rate one should expect for n samples. However, evaluation of this average error rate still requires knowing the underlying problem—that is, the values for $\mathbf{p}$ and $\mathbf{q}$. If $\mathbf{p}$ and $\mathbf{q}$ are quite different, the average error rate will be near zero, while if $\mathbf{p}$ and $\mathbf{q}$ are quite similar, it will be near 0.5.

A sweeping way to eliminate this dependence of the answer upon the problem is to average the answer over all possible problems. That is, we assume some prior distribution for the unknown parameters $\mathbf{p}$ and $\mathbf{q}$, and we average $P(E|\mathbf{p}, \mathbf{q}, n)$ with respect to $\mathbf{p}$ and $\mathbf{q}$. The resulting *problem-average probability of error* $\bar{P}(E|m, n)$

will depend only on the number m of cells, the number n of samples, and the prior distributions.

Of course, choosing the prior distributions is a delicate matter. By favoring easy problems we can make $\bar{P}$ approach zero, and by favoring hard problems we can make $\bar{P}$ approach 0.5. We would like to choose a prior distribution corresponding to the class of problems we typically encounter, but there is no obvious way to do that. A bold approach is merely to assume that problems are "uniformly distributed"—that is, that the vectors $\mathbf{p}$ and $\mathbf{q}$ are distributed uniformly over the simplexes

$$p_i \geq 0, \qquad \sum_{i=1}^{m} p_i = 1, \tag{48}$$

$$q_i \geq 0, \qquad \sum_{i=1}^{m} q_i = 1.$$

Note that this uniform distribution over the space of $\mathbf{p}$ and $\mathbf{q}$ does not correspond to some purported uniform distribution over possible distributions or target functions, the issue pointed out in Section 9.6.5.

Figure 9.14 summarizes simulation experiments and shows curves of $\bar{P}$ as a function number of cells for fixed numbers of training patterns. With an infinite number of training patterns the maximum-likelihood estimates are perfect, and $\bar{P}$ is the average of the Bayes error rate over all problems. The corresponding curve for $\bar{P}(E|m, \infty)$ decreases rapidly from 0.5 at $m = 1$ to the asymptotic value of 0.25 as m approaches infinity. The fact that $\bar{P} = 0.5$ if $m = 1$ is not surprising, because if there is only one cell the decision must be based solely on the prior probabilities. The fact that $\bar{P}$ approaches 0.25 as m approaches infinity is aesthetically pleasing, because this value is halfway between the extremes of 0.0 and 0.5. The fact that the problem-average error rate is so high merely shows that many hopelessly difficult classification prob-

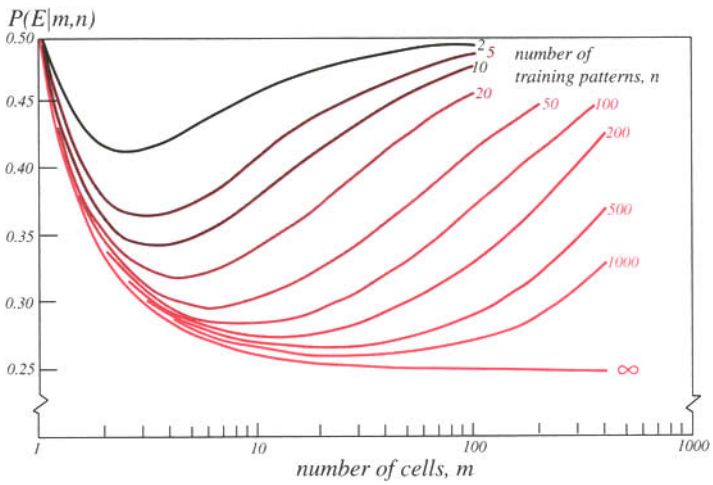


FIGURE 9.14. The probability of error E on a two-category problem for a given number of samples, n, can be estimated by splitting the feature space into m cells of equal size and classifying a test point according to the label of the most frequently represented category in the cell. The graphs show the average error of a large number of random problems having the given n and m indicated.

lems are included in this average. Clearly, it would be rash indeed to conclude that the "average" pattern recognition problem will have this error rate.

However, the most interesting feature of these curves is that for every curve involving a finite number of samples there is an optimal number of cells. This is directly related to the fact that with a finite number of samples the performance will worsen if too many features are used. In this case it is clear why there exists an optimal number of cells for any given n and m. At first, increasing the number of cells makes it easier to distinguish between the distributions represented by the vectors $\mathbf{p}$ and $\mathbf{q}$, thereby allowing improved performance. However, if the number of cells becomes too large, there will not be enough training patterns to fill them. Eventually, the number of patterns in most cells will be zero, and we must return to using just the ineffective *a priori* probabilities for classification. Thus, for any finite n, $\bar{P}(E|m, n)$ must approach 0.5 as m approaches infinity.

The value of m for which $\bar{P}(E|m, n)$ is minimum is quite small. For $n = 500$ samples, it is somewhere around $m = 200$ cells. Suppose that we were to form the cells by dividing each feature axis into l intervals. Then with d features we would have $m = l^d$ cells. If $l = 2$, which is extremely crude quantization, this implies that using more than four or five binary features will lead to worse rather than better performance. This is a very pessimistic result, but then so is the statement that the average error rate is 0.25. These numerical values are a consequence of the prior distribution chosen for the problems, and they are of no significance when one is facing a particular problem. The main thing to be learned from this analysis is that the performance of a classifier certainly does depend on the number of training patterns, and that if this number is fixed, increasing the number of features beyond a certain point raises the variance unacceptably and will be counterproductive.

9.6.7 Predicting Final Performance from Learning Curves

Training on very large data sets can be computationally intensive, requiring days, weeks, or even months on powerful machines. If we are exploring and comparing several different classification techniques, the total training time needed may be unacceptably long. What we seek, then, is a method to compare classifiers without the need of training all of them fully on the complete data set. If we can determine the most promising model quickly and efficiently, all we then must do is train *this* model fully.

One method is to use a classifier's performance on a relatively *small* training set to predict its performance on the ultimate large training set. Such performance is revealed in a type of learning curve in which the test error is plotted versus the size of the training set. Figure 9.15 shows the error rate on an independent test set after the classifier has been fully trained on $n' \leq n$ points in the training set. (Note that in this form of learning curve the training error decreases monotonically and does not show "overtraining" evident in curves such as Fig. 9.9.)

For many real-world problems, such learning curves decay monotonically and can be adequately described by a power-law function of the form

$$E_{test} = a + \frac{b}{n'^{\alpha}} \tag{49}$$

where a, b and $\alpha \geq 1$ depend upon the task and the classifier. In the limit of very large n', the training error equals the test error, because both the training and test sets represent the full problem space. Thus we also model the training error as a

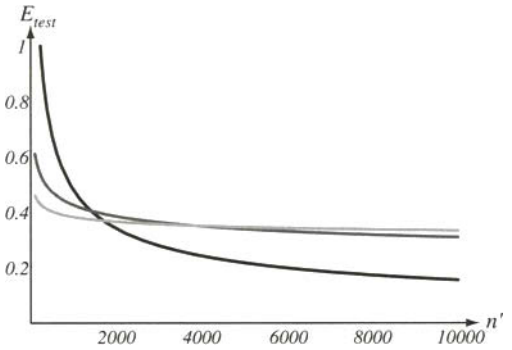


FIGURE 9.15. The test error for three classifiers, each fully trained on the given number n' of training patterns, decreases in a typical monotonic power-law function. Notice that the rank order of the classifiers trained on $n' = 500$ points differs from that for $n' = 10000$ points and the asymptotic case.

power-law function, having the same asymptotic error,

$$E_{train} = a - \frac{c}{n'^{\beta}}. \tag{50}$$

If the classifier is sufficiently powerful, this asymptotic error, a, is equal to the Bayes error. Furthermore, such a powerful classifier can learn perfectly the small training sets and thus the training error (measured on the n' points) will vanish at small n', as shown in Fig. 9.16.

Now we seek to estimate the asymptotic error, a, from the training and test errors on small and intermediate size training sets. From Eqs. 49 and 50 we find

$$E_{test} + E_{train} = 2a + \frac{b}{n'^{\alpha}} - \frac{c}{n'^{\beta}}, \tag{51}$$

$$E_{test} - E_{train} = \frac{b}{n^{\alpha}} + \frac{c}{n^{\beta}}.$$

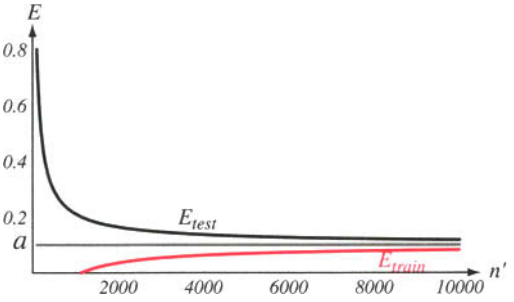


FIGURE 9.16. Test and training error of a classifier fully trained on data subsets of different size n' selected randomly from the full set $\mathcal{D}$. At low n', the classifier can learn the category labels of the points perfectly, and thus the training error vanishes there. In the limit $n' \to \infty$, both training and test errors approach the same asymptotic value, a. If the classifier is sufficiently powerful and the training data is sampled i.i.d., then a is the Bayes error rate, E_B.

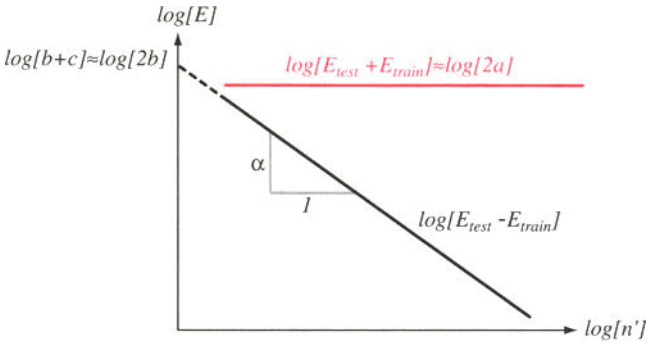


FIGURE 9.17. If the test and training errors versus training set size obey the power-law functions of Eqs. 49 and 50, then the log of the sum and log of the difference of these errors are straight lines on a log-log plot. The estimate of the asymptotic error rate a is then simply related to the height of the $\log[E_{test} + E_{train}]$ line, as shown.

If we make the assumption of $\alpha = \beta$ and $b = c$, then Eq. 51 reduces to

$$E_{test} + E_{train} = 2a, \tag{52}$$
$$E_{test} - E_{train} = \frac{2b}{n'^{\alpha}}.$$

Given this assumption, it is a simple matter to measure the training and test errors for small and intermediate values of n', plot them on a log-log scale, and estimate a, as shown in Fig. 9.17. Even if the approximations $\alpha = \beta$ and $b = c$ do not hold in practice, the difference $E_{test} - E_{train}$ nevertheless still forms a straight line on a log-log plot and the sum, $s = b+c$, can be found from the height of the $\log[E_{test}+E_{train}]$ curve. The weighted sum $cE_{test} + bE_{train}$ will be a straight line for some empirically set values of b and c, constrained to obey $b + c = s$, enabling a to be estimated (Problem 42). Once a has been estimated for each in the set of candidate classifiers, the one with the lowest a is chosen and must be trained on the full training set $\mathcal{D}$.

9.6.8 The Capacity of a Separating Plane

Consider the partitioning of a d-dimensional feature space by a hyperplane $\mathbf{w}^t\mathbf{x} + w_0 = 0$, as might be trained by the Perceptron algorithm (Chapter 5). Suppose that we are given n sample points in general position—that is, with no subset of $d + 1$ points falling in a $(d-1)$-dimensional subspace. Assume each point is labeled either ω_1 or ω_2. Of the 2^n possible dichotomies of n points in d dimensions, a certain fraction $f(n, d)$ are said to be linear dichotomies. These are the labelings for which there exists a hyperplane separating the points labeled ω_1 from the points labeled ω_2. It can be shown (Problem 41) that this fraction is given by

$$f(n,d) = \begin{cases} 1 & \text{for } n \le d+1 \\ \frac{2}{2^n}\sum_{i=0}^{d}\binom{n-1}{i} & \text{for } n > d+1, \end{cases} \tag{53}$$

as plotted in Fig. 9.18 for several values of d.

To understand the issue more fully, consider the one-dimensional case with four points; according to Eq. 53, we have $f(n = 4, d = 1) = 0.5$. The table shows schematically all 16 of the equally likely labels for four patterns along a line. (For

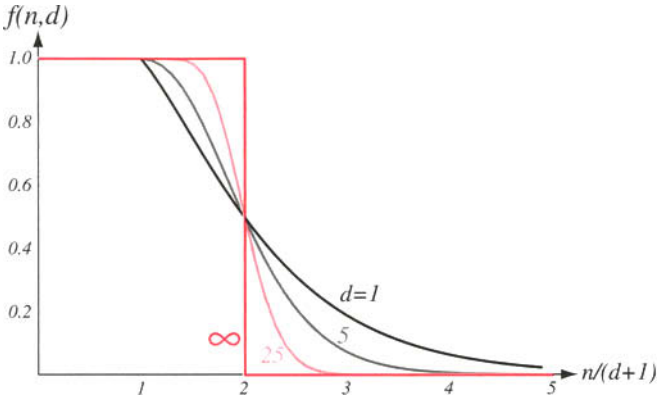


FIGURE 9.18. The fraction of dichotomies of n points in d dimensions that are linear, as given by Eq. 53.

instance, 0010 indicates that the labels are assigned $\omega_1\omega_1\omega_2\omega_1$.) The × marks those arrangements that are linearly separable—that is, in which a single point decision boundary can separate all ω_1 patterns from all ω_2 patterns. Indeed as given by Eq. 53, 8 of the 16—half—are linearly separable.

Labels	Linearly Separable?	Labels	Linearly Separable?
0000	×	1000	×
0001	×	1001	
0010		1010	
0011	×	1011	
0100		1100	×
0101		1101	
0110		1110	×
0111	×	1111	×

CAPACITY

Note from Fig. 9.18 that all dichotomies of $d + 1$ or fewer points are linear. This means that a hyperplane is not overconstrained by the requirement of correctly classifying $d + 1$ or fewer points. In fact, if d is large, it is not until n is a sizable fraction of $2(d + 1)$ that the problem begins to become difficult. At $n = 2(d + 1)$, which is sometimes called the *capacity* of a hyperplane, half of the possible dichotomies are still linear. Thus, a linear discriminant is not effectively overdetermined until the number of samples is several times as large as the dimensionality of the feature space or subset of the problems. This is often expressed as follows: "Generalization begins only after learning ends." Alternatively, we cannot expect a linear classifier to "match" a problem, on average, if the dimension of the feature space is greater than $n/2 - 1$.

9.7 COMBINING CLASSIFIERS

We have already mentioned classifiers whose decision is based on the outputs of component classifiers (Sections 9.5.1 and 9.5.2). Such full classifiers are variously

MIXTURE OF EXPERT

called *mixture-of-expert* models, ensemble classifiers, modular classifiers, or occasionally pooled classifiers. Such classifiers are particularly useful if each of its component classifiers is highly trained (i.e., an "expert") in a different region of the feature space. We first consider the case where each component classifier provides probability estimates. Later, in Section 9.7.2 we consider the case where component classifiers provide rank order information or one-of-c outputs.

9.7.1 Component Classifiers with Discriminant Functions

MIXTURE MODEL

We assume that each pattern is produced by a *mixture model*, in which first some fundamental process or function indexed by r (where $1 \leq r \leq k$) is randomly chosen according to distribution $P(r|\mathbf{x}, \boldsymbol{\theta}_0^0)$ where $\boldsymbol{\theta}_0^0$ is a parameter vector. Next, the selected process r emits an output $\mathbf{y}$ (e.g., a category label) according to $P(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_r^0)$, where the parameter vector $\boldsymbol{\theta}_r^0$ describes the state of the process. (The superscript 0 indicates the properties of the generating model. Below, terms without this superscript refer to the parameters in a classifier.) The overall probability of producing output $\mathbf{y}$ is then the sum over all the processes according to

$$P(\mathbf{y}|\mathbf{x}, \boldsymbol{\Theta}^0) = \sum_{r=1}^{k} P(r|\mathbf{x}, \boldsymbol{\theta}_0^0) P(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}^0), \tag{54}$$

where $\boldsymbol{\Theta}^0 = [\boldsymbol{\theta}_0^0, \boldsymbol{\theta}_1^0, \ldots, \boldsymbol{\theta}_k^0]^t$ represents the vector of all relevant parameters. Equation 54 describes a *mixture distribution*, which could be discrete or continuous.

MIXTURE DISTRIBUTION

Figure 9.19 shows the basic architecture of an ensemble classifier whose task is to classify a test pattern $\mathbf{x}$ into one of c categories; this architecture matches the assumed mixture model. A test pattern $\mathbf{x}$ is presented to each of the k component classifiers,

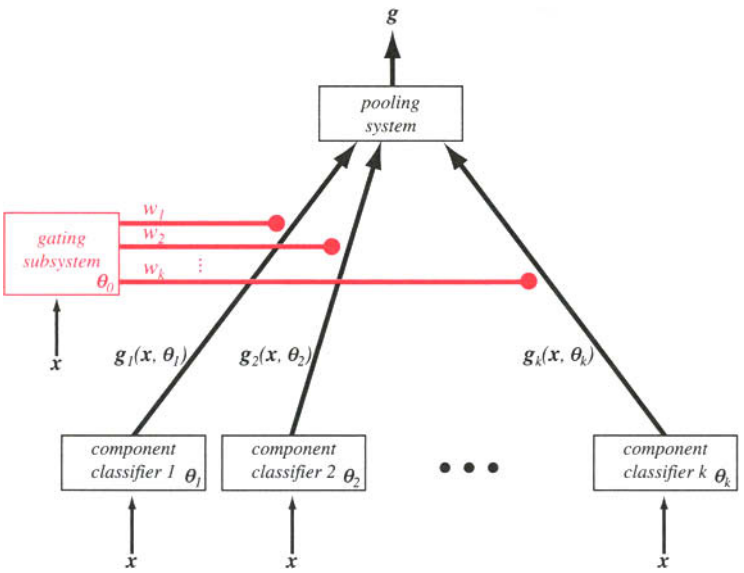


FIGURE 9.19. The mixture-of-experts architecture consists of k component classifiers or "experts," each of which has trainable parameters $\boldsymbol{\theta}_i$, $i = 1, \ldots, k$. For each input pattern $\mathbf{x}$, each component classifier i gives estimates of the category membership $g_{ir} = P(\omega_r|\mathbf{x}, \boldsymbol{\theta}_i)$. The outputs are weighted by the gating subsystem governed by parameter vector $\boldsymbol{\theta}_0$ and are pooled for ultimate classification.

each of which emits c scalar discriminant values, one for each category. The c discriminant values from component classifier r are grouped and marked $\mathbf{g}(\mathbf{x}, \boldsymbol{\theta}_r)$ in the figure, with

$$\sum_{j=1}^{c} g_{rj} = 1 \qquad \text{for all } r. \tag{55}$$

GATING SUBSYSTEM

All discriminant values from component classifier r are multiplied by a scalar weight w_r, governed by the *gating subsystem*, which has a parameter vector $\boldsymbol{\theta}_0$. Below we shall use the conditional mean of the mixture density, which can be calculated from Eq. 54:

$$\boldsymbol{\mu} = \mathcal{E}[\mathbf{y}|\mathbf{x}, \boldsymbol{\Theta}] = \sum_{r=1}^{k} w_r \boldsymbol{\mu}_r, \tag{56}$$

where $\boldsymbol{\mu}_r$ is the conditional mean associated with $P(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_r^0)$.

The mixture-of-experts architecture is trained so that each component classifier models a corresponding process in the mixture model, and the gating subsystem models the mixing parameters $P(r|\mathbf{x}, \boldsymbol{\theta}_0^0)$ in Eq. 54. The goal is to find parameters that maximize the log-likelihood for n training patterns $\mathbf{x}^1, \ldots, \mathbf{x}^n$ in set $\mathcal{D}$:

$$l(\mathcal{D}, \boldsymbol{\Theta}) = \sum_{i=1}^{n} \ln \left(\sum_{r=1}^{k} P(r|\mathbf{x}^i, \boldsymbol{\theta}_0) P(\mathbf{y}^i|\mathbf{x}^i, \boldsymbol{\theta}_r) \right). \tag{57}$$

A straightforward approach is to use gradient descent on the parameters, where the derivatives are (Problem 44)

$$\frac{\partial l(\mathcal{D}, \boldsymbol{\Theta})}{\partial \boldsymbol{\theta}_r} = \sum_{i=1}^{n} P(r|\mathbf{y}^i, \mathbf{x}^i) \frac{\partial}{\partial \boldsymbol{\theta}_r} \ln[P(\mathbf{y}^i|\mathbf{x}^i, \boldsymbol{\theta}_r)] \quad \text{for } r = 1, \ldots, k \tag{58}$$

and

$$\frac{\partial l(\mathcal{D}, \boldsymbol{\Theta})}{\partial g_r} = \sum_{i=1}^{n} (P(r|\mathbf{y}^i, \mathbf{x}^i) - w_r^i). \tag{59}$$

Here $P(r|\mathbf{y}^i, \mathbf{x}^i)$ is the posterior probability of process r conditional on the input and output being $\mathbf{x}^i$ and $\mathbf{y}^i$, respectively. Moreover, w_r^i is the prior probability $P(r|\mathbf{x}^i)$ that process r is chosen given the input is $\mathbf{x}^i$. Gradient descent according to Eq. 59 moves the prior probabilities to the posterior probabilities. The Expectation-Maximization (EM) algorithm can be used to train this architecture as well (Chapter 3).

WINNER-TAKE-ALL

The final decision rule is simply to choose the category corresponding to the maximum discriminant value after the pooling system. An alternative, *winner-take-all* method is to use the decision of the single component classifier that is "most confident"—that is, has the largest single discriminant value g_{rj}. While the winner-take-all method is provably suboptimal, it nevertheless is simple and can work well if the component classifiers are experts in separate regions of the input space.

We have skipped over a problem: How many component classifiers should be used? Of course, if we have prior information about the number of component processes that generated the mixture density, this should guide our choice of k. In the

absence of such information, we may have to explore different values of k, thereby tailoring the bias and variance of the full ensemble classifier. Typically, if the true number of components in the mixture density is k^*, a mixture-of-experts more than k^* component classifiers will generalize better than one with fewer than k^* component classifiers because the extra component classifiers learn to duplicate one another and hence become redundant.

9.7.2 Component Classifiers without Discriminant Functions

Occasionally we seek to form an ensemble classifier from highly trained component classifiers, some of which might not themselves compute discriminant functions. For instance, we might have four component classifiers—a k-nearest-neighbor classifier, a decision tree, a neural network, and a rule-based system—all addressing the same problem. While a neural network would provide analog values for each of the c categories, the rule-based system would give only a single category label (i.e., a one-of-c representation) and the k-nearest-neighbor classifier would give only rank order of the categories.

In order to integrate the information from the component classifiers we must convert the their outputs into discriminant values obeying the constraint of Eq. 55 so we can use the framework of Fig. 9.19. The simplest heuristics to this end are the following:

Analog: If the outputs of a component classifier are analog values $\tilde{g}_i$, we can use the *softmax* transformation,

SOFTMAX

$$g_i = \frac{e^{\tilde{g}_i}}{\sum_{j=1}^{c} e^{\tilde{g}_j}}. \tag{60}$$

to convert them to values g_i.

Rank order: If the output is a rank order list, we assume the discriminant function is linearly proportional to the rank order of the item on the list. Of course, the resulting g_i should then be properly normalized, and thus sum to 1.0.

One-of-c: If the output is a one-of-c representation, in which a single category is identified, we let $g_j = 1.0$ for the j corresponding to the chosen category, and 0.0 otherwise.

The table gives a simple illustration of these heuristics.

Analog value		Rank order		One-of-c	
$\tilde{g}_i$	g_i	$\tilde{g}_i$	g_i	$\tilde{g}_i$	g_i
0.4	0.158	3rd	4/21 = 0.194	0	0.0
0.6	0.193	6th	1/21 = 0.048	1	1.0
0.9	0.260	5th	2/21 = 0.095	0	0.0
0.3	0.143	1st	6/21 = 0.286	0	0.0
0.2	0.129	2nd	5/21 = 0.238	0	0.0
0.1	0.111	4th	3/21 = 0.143	0	0.0

Once the outputs of the component classifiers have been converted to effective discriminant functions in this way, the component classifiers are themselves held fixed, but the gating network is trained as described in Eq. 59. This method is particularly useful when several highly trained component classifiers are pooled to form a single decision.

SUMMARY

The No Free Lunch Theorem states that in the absence of prior information about the problem there are no reasons to prefer one learning algorithm or classifier model over another. Given that a finite set of feature values are used to distinguish the patterns under consideration, the Ugly Duckling Theorem states that the number of predicates shared by any two different patterns is constant and does not depend upon the choice of the two objects. Together, these theorems highlight the need for insight into proper features and matching the algorithm to the data distribution: There is no problem-independent "best" learning or pattern recognition system nor feature representation. In short, formal theory and algorithms taken alone are not enough; pattern classification is an empirical subject.

Two ways to describe the match between classifier and problem are the bias and variance. The bias measures the accuracy or quality of the match (high bias implies a *poor* match), and the variance measures the precision or specificity of the match (a high variance implies a *weak* match). The bias-variance dilemma states that learning procedures with increased flexibility to adapt to the training data (e.g., have more free parameters) tend to have lower bias but higher variance. In classification there is a nonlinear relationship between bias and variance, and low variance tends to be more important for classification than low bias. If classifier models can be expressed as binary strings, the minimum-description-length principle states that the best model is the one with the minimum sum of such a model description and the training data with respect to that model. This general principle can be extended to cover model-specific heuristics such as weight decay and pruning in neural networks, regularization in specific models, and so on.

The basic insight underlying resampling techniques—such as the bootstrap, jackknife, boosting, and bagging—is that multiple data sets selected from a given data set enable the value and ranges of arbitrary statistics to be computed. In classification, boosting techniques such as AdaBoost adjust the match of full classifier to the problem (and thus the bias and variance) even for an arbitrary basic classification method. In learning with queries, the classifier system presents query patterns to an oracle for labeling. Such learning is most efficient if *informative* patterns—ones for which the classifier is least certain—are presented as queries.

There are a number of methods for estimating the final accuracy of classifiers and thus comparing them. Each is based on assumptions—for example, that the parametric model is known or that the form of its learning curve is known. Cross-validation, jackknife, and bootstrap methods are closely related techniques that use subsets of the training data to estimate classifier accuracy. Maximum-likelihood (ML-II) and Bayesian methods—extensions of methods for setting *parameters*—can be used to compare and choose among *models*. A key term in Bayesian model selection is the Occam factor, which describes how the allowable volume in parameter space shrinks due to constraints imposed by the training data. The method penalizes "overly complex" models, where such complexity is a data-dependent property.

There are a number of methods for combining the outputs of separate component or "expert" classifiers, such as linear weighting, winner-takes-all, and so on. Overall classification is generally better when the decision rules of the component classifiers differ and provide complementary information.

BIBLIOGRAPHICAL AND HISTORICAL REMARKS

The No Free Lunch Theorem appears in reference [110] as well as in Wolpert's collection of contributions on the foundations of the theory of generalization [109]. Schaffer's "conservation law in generalization" is a reformulation of one of the parts of the theorem, and was the inspiration for Fig. 9.1 [83]. The Ugly Duckling Theorem was proven in reference [105], which also explores some of its philosophical implications [79].

The foundational work on Kolmogorov complexity appears in references [57, 58, 93] and [94], but a short elementary overview [14] and Chaitin's [15] and particularly Li and Vitányi's [66] books are far more accessible. Barron and Cover were the first to use a minimum description length (MDL) principle to estimate densities [7]. There are several versions of MDL [80, 81], such as the Akaike Information Criterion (AIC) [1, 2] and the Bayes Information Criterion (BIC) [86] (which differ from MDL by relative weighting of model penalty). Likewise, the Network Information Criterion (NIC) can be used to compare neural networks of the same architecture [73]. More generally, neural network pruning and general regularization methods can be cast as "minimum description" principles, but with different measures for model and fit of the data [65].

Convincing theoretical and philosophical justifications of Occam's razor have been elusive. Karl Popper has argued that Occam's razor is without operational value, because there is no clear criterion or measure of simplicity [76], a point echoed by other philosophers [92]. It is worth pointing out alternatives to Occam's razor, which Isaac Newton cast in **Principia** as "Natura enim simplex est, et rerum causis superfluis non luxuriat," or "for nature indeed is simple, and does not luxuriate in superfluous causes" [74]. The first alternative stems from Epicurus (342?–270? B.C.), who, in a letter to Pythocles, stated what we now call the principle of multiple explanations or principle of indifference: If several theories are consistent with the data, retain all such theories [29]. The second is a restatement of Bayes approach: The probability of a model or hypothesis being true is proportional to the designer's prior belief in the hypothesis multiplied by the conditional probability of the data given the hypothesis in question. Occam's razor, here favoring "simplicity" in classifiers, can be motivated by considering the cost (difficulty) of designing the classifier and the principle of bounded rationality—that we often settle for an adequate but not necessarily the optimal solution [89]. An empirical study showing that simple classifiers often work well can be found in reference [45].

The basic bias-variance decomposition and bias-variance dilemma [37] in regression appear in many statistics books [16, 41]. Geman et al. give a very clear presentation in the context of neural networks, but their discussion of *classification* is only indirectly related to their mathematical derivations for *regression* [35]. Our presentation for classification (zero-one loss) is based on Friedman's important paper [32]; the bias-variance decomposition has been explored in other nonquadratic cost functions as well [42].

Quenouille introduced the jackknife in 1956 [78]. The theoretical foundations of resampling techniques are presented in Efron's clear book [28], and practical guides

to their use include references [25] and [36]. Papers on bootstrap techniques for error estimation include reference [48]. Breiman has been particularly active in introducing and exploring resampling methods for estimation and classifier design, such as bagging [11] and general arcing [13]. AdaBoost [31] builds upon Schapire's analysis of the strength of weak learnability [84] and Freund's early work in the theory of learning [30]. Boosting in multicategory problems is a bit more subtle than in two-category problems we discussed [85]. Angluin's early work on queries for concept learning [3] was generalized to active learning by Cohn and many others [18, 20] and is fundamental to some efforts in collecting large databases, as discussed in references [95, 96] and [100].

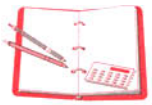
Cross-validation was introduced by Cover [23] and has been used extensively in conjunction with classification methods such as neural networks. Estimates of error under different conditions include references [34, 104] and [111], and an excellent paper that derives the size of test set needed for accurate estimation of classification accuracy is reference [39]. Bowyer and Phillip's book covers empirical evaluation techniques in computer vision [10], many of which apply to more general classification domains.

The roots of maximum-likelihood model selection stem from Bayes himself, but one of the earlier technical presentations is reference [38]. Interest in Bayesian model selection was revived in a series of papers by MacKay, whose primary interest was in applying the method to neural networks and interpolation [67, 68, 69, 70]. These model selection methods have subtle relationships to minimum-description-length (MDL) [80] and so-called maximum entropy approaches—topics that would take us a bit beyond our central concerns. Cortes and her colleagues pioneered the analysis of learning curves for estimating the final quality of a classifier in references [21] and [22]. No rate of convergence results can be made in the arbitrary case for finding the Bayes error, however [6]. Hughes [46] first carried out the required computations displayed in Fig. 9.14.

Books on techniques for combining general classifiers include references [55] and [56] and those for combining neural nets in particular include references [9] and [88]. Perrone and Cooper described the benefits that arise when expert classifiers disagree [75]. Dasarathy's book [24] has a nice mixture of theory (focusing more on sensor fusion than multiclassifier systems per se) and a collection of important original papers, including references [43, 61] and [97]. The simple heuristics for converting 1-of-c and rank order outputs to numerical values enabling integration were discussed in reference [63]. The hierarchical mixture-of-experts architecture and learning algorithm were first described in references [51] and [52]. A specific hierarchical multiclassifier technique is stacked generalization [12, 90, 91] and [108], where for instance Gaussian kernel estimates at one level are pooled by yet other Gaussian kernels at a higher level.

We have skipped over a great deal of work from the formal field of computational learning theory. Such work is generally preoccupied with convergence properties, asymptotics, and computational complexity, and usually relies on simplified or general models. Anthony and Biggs' short, clear and elegant book is an excellent introduction to the field [5]; broader texts include references [49, 53] and [72]. Perhaps the work from the field most useful for pattern recognition practitioners comes from weak learnability and boosting, mentioned above. The probably approximately correct (PAC) framework, introduced by Valiant [99], has been very influential in computation learning theory, but has had only minor influence on the development of practical pattern recognition systems. A somewhat broader formulation, probably almost Bayes (PAB), is described in reference [4]. The work by Vapnik and Cher-

vonenkis on structural risk minimization [103], and later Vapnik-Chervonenkis (VC) theory [101, 102], derives (among other things) expected error bounds; it too has proven influential to the theory community. Alas, the bounds derived are somewhat loose in practice, as presented in references [19] and [107].



PROBLEMS

Section 9.2

1. One of the "conservations laws" for generalization states that the positive generalization performance of an algorithm in some learning situations must be offset by negative performance elsewhere. Consider a very simple learning algorithm that seems to contradict this law. For each test pattern, the prediction of the *majority learning algorithm* is merely the category most prevalent in the training data.
   (a) Show that averaged over all two-category problems of a given number of features the off-training set error is 0.5.
   (b) Repeat (a) but for the *minority learning algorithm*, which always predicts the category label of the category *least* prevalent in the training data.
   (c) Use your answers from (a) and (b) to illustrate Part 2 of the No Free Lunch Theorem (Theorem 9.1).
2. Prove Part 1 of Theorem 9.1, that is, uniformly averaged over all target functions F, $\mathcal{E}_1(E|F,n) - \mathcal{E}_2(E|F,n) = 0$. Summarize and interpret this result in words.
3. Prove Part 2 of Theorem 9.1, that is, for any fixed training set $\mathcal{D}$, uniformly averaged over F, $\mathcal{E}_1(E|F,\mathcal{D}) - \mathcal{E}_2(E|F,\mathcal{D}) = 0$. Summarize and interpret this result in words.
4. Prove Part 3 of Theorem 9.1, that is, uniformly averaged over all priors $P(F)$, $\mathcal{E}_1(E|n) - \mathcal{E}_2(E|n) = 0$. Summarize and interpret this result in words.
5. Prove Part 4 of Theorem 9.1, i.e., for any fixed training set $\mathcal{D}$, uniformly averaged over $P(F)$, $\mathcal{E}_1(E|\mathcal{D}) - \mathcal{E}_2(E|\mathcal{D}) = 0$. Summarize and interpret this result in words.
6. Suppose you call an algorithm better if it performs slightly better than average over *most* problems, but very poorly on a small number of problems. Explain why the NFL Theorem does not preclude the existence of algorithms "better" in this sense.
7. Show by simple counterexamples that the averaging in the different Parts of the No Free Lunch Theorem (Theorem 9.1) must be "uniformly." For instance imagine that the sampling distribution is a Dirac delta distribution centered on a single target function, and algorithm 1 guesses the target function exactly while algorithm 2 disagrees with algorithm 1 on every prediction.
   (a) Part 1
   (b) Part 2
   (c) Part 3
   (d) Part 4

8. State how the No Free Lunch theorems imply that you cannot use training data to distinguish between new problems for which you generalize well from those for which you generalize poorly. Argue by *reductio ad absurdum*—that is, that if you could distinguish such problems, then the No Free Lunch Theorem would be violated.

9. Prove the relation $\sum_{r=0}^{n} \binom{n}{r} = (1+1)^n = 2^n$ of Eq. 5 two ways:

   (a) State the polynomial expansion of $(x+y)^n$ as a summation of coefficients and powers of x and y. Then, make a simple substitution for x and y.

   (b) Prove the relation by induction. Let $K(n) = \sum_{r=0}^{n} \binom{n}{r}$. First confirm that the relation is valid for $n = 1$—that is, that $K(1) = 2^1$. Now prove that $K(n+1) = 2K(n)$ for arbitrary n.

10. Consider the number of different Venn diagrams for k binary features $f_1, \ldots, f_k$. (Figure 9.2 shows several of these configurations for the $k = 3$ case.)

    (a) How many functionally different Venn diagrams exist for the $k = 2$ case? Sketch all of them. For each case, state how many different regions exists—that is, how many functionally different patterns can be represented.

    (b) Repeat part (a) for the $k = 3$ case.

    (c) How many functionally different Venn diagrams exist for the arbitrary k case?

11. While the text outlined a proof of the Ugly Duckling Theorem (Theorem 9.2), this problem asks you to fill in some of the details and explain some of its implications.

    (a) The discussion in the text assumed that the classification problem had no constraints, and thus could be described by the most general Venn diagram, in which all predicates of a given rank r were present. How do the derivations change, if at all, if we know that there are constraints provided by the problem, as in Fig. 9.2 (b) and (c)?

    (b) Someone sees two cars, A and B, made by the same manufacturer in the same model year, both are four-door and have the same engine type, but they differ solely in that one is red while the other is green. Car C is made by a different manufacturer, has a different engine, is two-door, and is blue. Explain in as much detail as possible why, even in this seemingly clear case, that in fact there are no prior reasons to view cars A and B as any "more similar" than cars B and C.

12. Suppose we describe patterns by means of predicates of a particular rank r^*. Show that the Ugly Duckling Theorem (Theorem 9.2) applies to any single level r^* and thus applies to all predicates up to an arbitrary maximum level.

13. Make some simple assumptions and state, using $O(\cdot)$ notation, the Kolmogorov complexity of the following binary strings:

    (a) $\underbrace{0101101110111110\ldots}_{n}$

    (b) $\underbrace{000\ldots00100\ldots000}_{n}$

    (c) $e = 10.10110111111000010\ldots_2$

    (d) $2e = 101.01101111110000101\ldots_2$

(e) The binary digits of π, but where every 100th digit is changed to the numeral 1.

(f) The binary digits of π, but where every nth digit is changed to the numeral 1.

14. Recall the notation from our discussion of the No Free Lunch Theorem and of Kolmogorov complexity. Suppose we use a learning algorithm with uniform $P(h|\mathcal{D})$. In that case $K(h, \mathcal{D}) = K(\mathcal{D})$ in Eq. 8. Explain and interpret this result.

15. Consider two binary strings x_1 and x_2. Explain why the Kolmogorov complexity of the pair obeys $K(x_1, x_2) \leq K(x_1) + K(x_2) + c$ for some positive constant c.

16. Consider designing a tree-based classifier of the general type explored in Chapter 8 employing the minimum-description length principle and alternatively the imposition of priors over the form of classifier.

(a) Suppose we design a tree classifier using a minimum-description length principle where the total entropy (in bits) consists of two terms: the entropy of the data with respect to the tree and the number of nodes in the tree. This is formally equivalent to a training the tree by maximum-likelihood techniques with a prior that favors smaller trees. Determine the functional form of this corresponding prior $P(K)$, where K is the total number of nodes in the tree. State any assumptions you must make.

(b) Suppose a tree classifier is trained by maximum-likelihood techniques where the prior on the number of nodes decreases with the number of nodes according to and exponential, that is, $P(K) \propto e^{-K}$. Determine the functional form of an equivalent minimum-description length criterion that will lead to the same final classifier.

BERRY PARADOX

17. The *Berry paradox* is related to the famous liar's paradox ("this statement is false"), as well as to a number of paradoxes in set theory explored by Bertrand Russell and Kurt Gödel. The Berry paradox shows indirectly how the notion of Kolmogorov complexity can be difficult and subtle. Consider describing positive integers by means of sentences—for instance, "the number of fingers on a human hand" or "the number of primes less than a million." Explain why the definition "the least number that cannot be defined in less than twenty words" is paradoxical, and explain informally how this relates to difficulties in computing Kolmogorov complexity.

Section 9.3

18. Expand the left-hand side of Eq. 11 to get the right-hand side, which expresses the mean-square error as a sum of a $bias^2$ and *variance*. Can *bias* ever be negative? Can *variance* ever be negative?

19. Fill in the steps leading to Eq. 18, that is,

$$\Pr[g(\mathbf{x};\ \mathcal{D}) \neq y] = |2F(\mathbf{x}) - 1|\Pr[g(\mathbf{x};\ \mathcal{D}) \neq y_B] + \Pr[y_B \neq y]$$

where the target function is $F(\mathbf{x})$, $g(\mathbf{x};\ \mathcal{D})$ is the computed discriminant value, and y_B is the Bayes discriminant value.

20. Assume that the probability of obtaining a particular discriminant value for pattern $\mathbf{x}$ for a training algorithm trained with data $\mathcal{D}$, denoted $p(g(\mathbf{x};\ \mathcal{D}))$, is a Gaussian. Use this Gaussian assumption and Eq. 19 to derive Eq. 20.

Section 9.4

21. Derive the jackknife estimate of the *bias* in Eq. 29.

22. Prove that in the limit of $B \to \infty$, the bootstrap estimate of the variance of the mean is the same as the standard estimate of the variance of the mean.

23. Prove that Eq. 24 for the average of the leave one out means, $\mu_{(\cdot)}$, is equivalent to Eq. 22 for the sample mean, $\hat{\mu}$.

CONSISTENT

24. We say that an estimator is *consistent* if it converges to the true value in the limit of infinite data. Prove that the standard mean of Eq. 22 is *not* consistent for the distribution $p(x) \sim \tan^{-1}(x - a)$ for any finite real constant a.

25. Prove that the jackknife estimate of an arbitrary statistic θ given in Eq. 30 is unbiased for estimating the true bias.

26. Verify that Eq. 26 for the jackknife estimate of the variance of the mean is formally equivalent to the variance implied by the traditional estimate given in Eq. 23.

27. Consider n points in one dimension. Use $O(\cdot)$ notation to express the computational complexity associated with each of the following estimations.

(a) The jackknife estimate of the mean

(b) The jackknife estimate of the median

(c) The jackknife estimate of the standard deviation

(d) The bootstrap estimate of the mean

(e) The bootstrap estimate of the median

(f) The bootstrap estimate of the standard deviation

28. Derive Eq. 34 for the bootstrap estimate of the bias.

Section 9.5

29. What is the computational complexity of full jackknife estimate of accuracy and variance for an unpruned nearest-neighbor classifier (Chapter 4)?

30. In standard boosting applied to a two-category problem, we must create a data set that is "most informative" with respect to the current state of the classifier. Why does this imply that *half* of its patterns should be classified correctly, rather than *none* of them? In a c-category problem, what portion of the patterns should be misclassified in a "most informative" set?

31. In active learning, learning can be speeded by creating patterns that are "informative"—that is, those for which the two largest discriminants are approximately equal. Consider the two-category case where for any point $\mathbf{x}$ in feature space, discriminant values g_1 and g_2 are returned by the classifier. Write pseudocode that takes two points—$\mathbf{x}_1$ classified as ω_1 and $\mathbf{x}_2$ classified as ω_2—and rapidly finds a new point $\mathbf{x}_3$ that is "near" the current decision boundary and hence is "informative." Assume only that the discriminant functions are monotonic along the line linking $\mathbf{x}_1$ and $\mathbf{x}_2$.

32. Consider AdaBoost with an arbitrary number of component classifiers.

(a) State clearly any assumptions you make, and derive Eq. 37 for the ensemble training error of the full boosted system.

(b) Recall that the training error for a weak learner applied to a two-category problem can be written $E_k = 1/2 - G_k$ for some positive value G_k. The training error for the first component classifier is $E_1 = 0.25$. Suppose that $G_k = 0.05$ for all $k = 1$ to k_{max}. Plot the upper bound on the ensemble test error given by Eq. 37, such as shown in Fig. 9.7.

(c) Suppose that G_k decreases as a function of k. Specifically, repeat part (b) with the assumption $G_k = 0.05/k$ for $k = 1$ to k_{max}.

Section 9.6

33. The No Free Lunch Theorem implies that if all problems are equally likely, then cross-validation must fail as often as it succeeds. Show this as follows: Consider algorithm 1 to be standard cross-validation and consider algorithm 2 to be *anti-cross-validation*, which advocates choosing the model that does *worst* on a validation set. Argue that if cross-validation were better than anti-cross-validation overall, the No Free Lunch Theorem would be violated.

34. Suppose we believe that the data for a pattern classification task from one category comes either from a uniform distribution $p(x) \sim U(x_l, x_u)$ or from a normal distribution, $p(x) \sim N(\mu, \sigma^2)$, but we have no reason to prefer one over the other. Our sample data is $\mathcal{D} = \{0.2, 0.5, 0.4, 0.3, 0.9, 0.7, 0.6\}$.

(a) Find the maximum-likelihood values of x_l, and x_u for the uniform model.

(b) Find the maximum-likelihood values of μ and σ for the Gaussian model.

(c) Use maximum-likelihood model selection to decide which model should be preferred.

35. Suppose we believe that the data for a pattern classification task from one category comes either from a uniform distribution bounded below by 0—that is, $p(x) \sim U(0, x_u)$—or from a normal distribution, $p(x) \sim N(\mu, \sigma^2)$, but we have no reason to prefer one over the other. Our sample data is $\mathcal{D} = \{0.2, 0.5, 0.4, 0.3, 0.9, 0.7, 0.6\}$.

(a) Find the maximum-likelihood values of x_u for the uniform model.

(b) Find the maximum-likelihood values of μ and σ for the Gaussian model.

(c) Use maximum-likelihood model selection to decide which model should be preferred.

(d) State qualitatively the difference between your solution here and that to Problem 34, without necessarily having to solve that problem. In particular, what are the implications from the fact that the two candidate models have different numbers of parameters?

36. Consider three candidate one-dimensional distributions, each parameterized by an unknown value for its "center":

- Gaussian: $p(x) \sim N(\mu, 1)$
- Triangle: $p(x) \sim T(\mu, 1) = \begin{cases} 1 - |x - \mu| & \text{for } |x - \mu| < 1 \\ 0 & \text{otherwise} \end{cases}$
- Uniform: $p(x) \sim U(\mu - 1, \mu + 1)$

We are given the data $\mathcal{D} = \{-0.9, -0.1, 0., 0.1, 0.9\}$, and thus, clearly the maximum-likelihood solution $\hat{\mu} = 0$ applies to each model.

(a) Use maximum-likelihood model selection to determine the best model for this data. State clearly any assumptions you make.

(b) Suppose we are sure for each model that the center must lie in the range $-1 \leq \mu \leq 1$. Calculate the Occam factor for each model and the data given.

(c) Use Bayesian model selection to determine the "best" model given $\mathcal{D}$.

37. Use Eq. 38 and generate curves of the form shown in Fig. 9.10. Prove analytically that the curves are symmetric with respect to the interchange $\hat{p} \rightarrow (1 - \hat{p})$ and $p \rightarrow (1 - p)$. Explain the reasons for this symmetry.

38. Let model h_i be described by a k-dimensional parameter vector $\boldsymbol{\theta}$. State your assumptions and show that the Occam factor can be written as

$$p(\hat{\boldsymbol{\theta}}|h_i)(2\pi)^{k/2}|\mathbf{H}|^{-1/2},$$

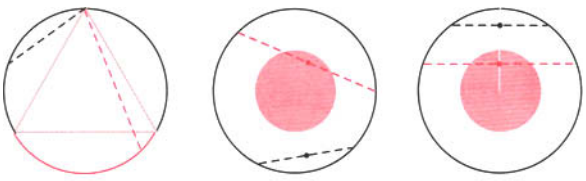
as given in Eq. 44, where the Hessian $\mathbf{H}$ is a matrix of second-order derivatives defined in Eq. 45.

BERTRAND'S PARADOX

39. *Bertrand's paradox* shows how the notion of "uniformly distributed" models can be problematic, and it leads us to question the principle of indifference (cf. Computer exercise 9). Consider the following problem: Given a circle, find the probability that a "randomly selected" chord has length greater than the side of an inscribed equilateral triangle.

Here are three possible solutions to this problem and their justifications, illustrated in the figure:

1. By definition, a chord strikes the circle at two points. We can arbitrarily rotate the figure so as to place one of those points at the top. The other point is equally likely to strike any other point on the circle. As shown in the left-hand side of the figure, one-third of such points (red) will yield that a chord with length greater than that of the side of an inscribed equilateral triangle. Thus the probability a chord is longer than the length of the side of an inscribed equilateral triangle is $P = 1/3$.
2. A chord is uniquely determined by the location of its midpoint. Any such midpoint that lies in the circular disk whose radius is half that of the full circle will yield a chord with length greater than that of an inscribed equilateral triangle. Because the area of this red disk is one-fourth that of the full circular disk, the probability is $P = 1/4$.
3. We can arbitrarily rotate the circle such that the midpoint of a chord lies on a vertical line. If the midpoint lies closer to the center than half the radius of the circle, the chord will be longer than the side of the inscribed equilateral triangle. Thus the probability is $P = 1/2$.



Explain why there is little or no reason to prefer one of the solution methods over another, and thus the solution to the problem itself is ill-defined. Use your answer to reconcile Bayesian model selection with the No Free Lunch Theorem (Theorem 9.1).

40. If k of n' independent, randomly chosen test patterns are misclassified, then as given in Eq. 38, k has a binomial distribution

$$P(k) = \binom{n'}{k} p^k (1-p)^{n'-k}$$

Prove that the maximum-likelihood estimate for p is then $\hat{p} = k/n'$, as given in Eq. 39.

41. Derive the relation for $f(n, d)$, the fraction of dichotomies of n randomly chosen points in d dimensions that are linearly separable, given by Eq. 53. Explain why $f(n, d) = 1$ for $n \leq d + 1$.

42. Write pseudocode for an algorithm to determine the large n' limit of the test error, given the assumption of a power-law decrease in error described by Eq. 52 and illustrated in Fig. 9.17.

43. Suppose a standard three-layer neural network having d input units, n_H hidden units, a single bias unit and $c = 2$ output units is trained on a two-category problem (Chapter 6). What is the degeneracy of the final assignment of weights? That is, how many ways can the weights be relabeled with the decision rule being unchanged? Explain how this degeneracy would need to be incorporated into a Bayesian model selection.

Section 9.7

44. Let $\mathbf{x}^i$ and $\mathbf{y}^i$ denote input and output vectors, respectively, and r index component processes ($1 \leq r \leq k$) in a mixture model. Use Bayes theorem,

$$P(r|\mathbf{y}^i, \mathbf{x}^i) = \frac{P(r|\mathbf{x}^i)P(\mathbf{y}^i|\mathbf{x}^i, r)}{\sum_{q=1}^{k} P(q|\mathbf{x}^i)P(\mathbf{y}^i|\mathbf{x}^i, q)},$$

to derive the derivatives Eqs. 58 and 59 used for gradient descent learning in the mixture of experts model.

45. Suppose a mixture-of-experts classifier has k Gaussians component classifiers of arbitrary mean and covariance in d dimensions, $N(\boldsymbol{\mu}_r, \boldsymbol{\Sigma}_r)$. Derive the specific learning rules for the parameters of each component classifier and for the gating subsystem, special cases of Eqs. 58 and 59.

COMPUTER EXERCISES

Several exercises will make use of the following three-dimensional data sampled from four categories, denoted ω_i.

	ω_1			ω_2			ω_3			ω_4		
Sample	x_1	x_2	x_3	x_1	x_2	x_3	x_1	x_2	x_3	x_1	x_2	x_3
1	2.5	3.4	7.9	4.2	4.9	11.3	2.9	15.5	4.6	16.9	12.4	0.2
2	4.3	4.4	7.1	11.7	5.3	10.5	3.6	13.9	9.8	12.1	16.8	2.1
3	7.1	0.8	6.3	8.4	11.1	6.6	10.3	6.1	12.3	13.7	12.1	5.5
4	1.4	−0.2	2.5	8.2	10.4	4.9	8.2	5.5	7.1	11.9	13.4	3.4
5	3.9	4.3	3.4	5.3	7.7	8.8	13.3	4.7	11.7	14.5	15.5	2.8
6	3.2	6.8	5.1	7.9	4.5	9.5	6.6	8.1	16.7	15.6	14.9	4.4
7	7.3	6.5	7.1	10.7	6.9	10.9	12.2	5.1	5.9	16.2	12.3	3.2
8	−0.7	3.1	8.1	9.6	9.7	7.3	15.6	3.3	10.7	12.2	16.3	3.2
9	2.8	5.9	2.2	8.2	11.2	6.3	4.6	10.1	13.8	14.5	12.9	−0.9
10	6.1	7.6	4.3	5.3	10.1	4.9	9.1	4.4	8.9	15.8	15.6	4.5

Section 9.2

1. Consider the use of the minimum description length principle for the design of a binary decision tree classifier (Chapter 4). Each question at a node is of the form "Is $x_i > \theta$?" (or alternatively "Is $x_i < \theta$?"). Specify each such question with five bits: Two bits specify the feature queried (x_1, x_2, or x_3), a single bit specifies whether the comparison is $>$ or $<$, and four bits specify each θ as an integer $0 \leq \theta \leq 16$. Assume that the Kolmogorov complexity of the classifier is, up to an additive constant, the sum of the bits of all questions. Assume too that the Kolmogorov complexity of the data, given the tree classifier, is merely the entropy of the data at the leaves, also measured in bits.

(a) Train your tree with the data from the four-category problem in the table above. Starting at the root, grow your tree a single node at a time, continuing until each node is as pure as possible. Plot as a function of the total number of nodes the Kolmogorov complexity of (1) the classifier, (2) the data with respect to the classifier, and (3) their sum (Eq. 8). Show the tree (including the questions at its nodes) having the minimum description length.

(b) The minimum description length principle gives a principled method for comparing classifiers in which the resolution of parameters (e.g., weights, thresholds, etc.) can be altered. Repeat part (a) but using only three bits to specify each threshold θ in the nodes.

(c) Assume that the additive constants for the Kolmogorov complexities of your above classifiers are equal. Which of all the classifiers has the minimum description length?

Section 9.3

2. Illustrate the bias-variance decomposition and the bias-variance dilemma for regression through simulations. Let the target function be $F(x) = x^2$ with Gaussian noise of variance 0.1. First, randomly generate 100 data sets, each of size $n = 10$, by selecting a value of x uniformly in the range $-1 \leq x \leq 1$ and then applying $F(x)$ with noise. Train any free parameters a_i (by minimum square error criterion) in each of the regression functions in parts (a) – (d), one data set at a time. Then make a histogram of the sum-square error of Eq. 11 (cf. Fig. 9.4). For each model use your results to estimate the bias and the variance.

(a) $g(x) = 0.5$

(b) $g(x) = 1.0$

(c) $g(x) = a_0 + a_1 x$

(d) $g(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$

(e) Repeat parts (a)–(d) for 100 data sets of size $n = 100$.

(f) Summarize all your above results, with special consideration of (i) the bias-variance decomposition and dilemma, and (ii) the effect of the size of the data set.

Section 9.4

TRIMMED MEAN

3. The *trimmed mean* of a distribution is merely the sample mean of the distribution from which some portion α (e.g., 0.1) of the highest and of the lowest points have been deleted. The trimmed mean is, of course, less sensitive to the presence of outliers than is the traditional sample mean.

(a) Show how in the limit $\alpha \to 0.5$, the trimmed mean of a distribution is the median.

(b) Let the data $\mathcal{D}$ be the x_3 values of the 10 patterns in category ω_2 in the table above. Write a program to determine (i) the jackknife estimate of the median of $\mathcal{D}$ and (ii) the jackknife estimate of the variance of this estimate.

(c) Repeat part (b) but for the $\alpha = 0.1$ trimmed mean and its variance.

(d) Repeat part (b) but for the $\alpha = 0.2$ trimmed mean and its variance.

(e) Repeat parts (b)–(d) but where $\mathcal{D}$ has an additional ("outlier") point at $x_3 = 20$.

(f) Interpret your results, with special attention to the sensitivity of the trimmed mean to outliers.

Section 9.5

4. Write a program to implement the AdaBoost procedure (Algorithm 1) with component classifiers whose linear discriminants are trained by the basic LMS algorithm (Chapter 5).

(a) Apply your system to the problem of discriminating the 10 points in ω_1 from the 10 points in ω_2 in the table above. Plot your training error as a function of the number of component classifiers. Be sure the graph extends to a k_{max} sufficiently high that the training error vanishes.

(b) Define a "supercategory" consisting of all the patterns in ω_1 and ω_2 in the table, and define another supercategory for the ω_3 and ω_4 patterns. Repeat part (a) for discriminating these supercategories.

(c) Compare and interpret your graphs in (a) and (b), paying particular attention to the relative difficulties of the classification problems.

5. Explore the value of active learning in a two-dimensional two-category problem in which the priors are Gaussians, $p(\mathbf{x}|\omega_i) \sim N(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ with $\boldsymbol{\mu}_1 = \binom{+5}{+5}$, $\boldsymbol{\mu}_2 = \binom{-5}{-5}$, $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = \begin{pmatrix} 20 & 0 \\ 0 & 20 \end{pmatrix}$, and $P(\omega_1) = P(\omega_2) = 0.5$. Throughout this problem, restrict data to be in the domain $-10 \leq x_i \leq +10$, for $i = 1, 2$.

(a) State by inspection the Bayes classifier. This will be the decision used by the oracle in part (c).

(b) Generate a training set of 100 points, 50 labeled ω_1 sampled according to $p(\mathbf{x}|\omega_1)$ and likewise 50 patterns according to $p(\mathbf{x}|\omega_2)$. Train a nearest-neighbor classifier (Chapter 4) using your data, and plot the decision boundary in two dimensions.

(c) Now assume there is an oracle, which can label any query pattern according to your answer in part (a), which we exploit through a particular form of active learning. To begin the learning, choose 10 points according to a uniform distribution in the domain $-10 \leq x_i \leq +10$, for $i = 1, 2$. Apply labels to these points according to the oracle to get $\mathcal{D}_1$ and $\mathcal{D}_2$, for each category. Now generate new query points as follows. Randomly choose a point from $\mathcal{D}_1$ and a point from $\mathcal{D}_2$; create a query point midway between these two points. Label the point according to the oracle and add it to the appropriate $\mathcal{D}_j$. Continue until the total number of labeled points is 100. Now create a nearest-neighbor classifier using all points, and plot the decision boundary in two dimensions.

(d) Compare qualitatively your classifiers from parts (a), (b) and (c), and discuss your results.

6. In simple boosting using three component classifiers we would like to use all n training points and to have a roughly equal number of patterns used in each component classifier (i.e., $n_1 \simeq n_2 \simeq n_3 \simeq n/3$).

(a) Generate a training set $\mathcal{D}$ consisting of $n = 300$ two-dimensional points, 150 in each of two categories. Draw your samples from uniform distributions in squares defined by their opposite corners, in particular

- $p(\mathbf{x}|\omega_1) \sim U\left(\binom{0}{0}, \binom{3}{3}\right)$
- $p(\mathbf{x}|\omega_2) \sim U\left(\binom{1}{1}, \binom{4}{4}\right)$

(b) Assume each component classifier is a simple monothetic binary decision tree consisting of a root node, two offspring nodes and four leaf nodes and trained on an entropy impurity (Chapter 8). Train a three-component classifier system using boosting. Start with $n_1 = n/3 = 100$ patterns.

(c) Perform a simple heuristic search for an appropriate for a good value of n_1 based on the values of n_2 and n_3 from your simulation in part (b). In particular, if you find n_1 is too low, re-initialize n_1 to be halfway between the maximum possible value $n_1 = n = 300$ and the current value. Alternatively, if you find n_1 is too high, re-initialize n_1 to be halfway between the minimum value $n_1 = 0$ and the current value.

(d) How many full applications of the boosting algorithm are needed to achieve an "acceptable" starting value for n_1? Be sure to state what you mean by "acceptable."

Section 9.6

7. Explore a case where validation need not yield an improved classifier. Throughout, the classifier will be k-nearest-neighbor (Chapter 4), where k will be set by validation. Consider a two-category problem in two dimensions, with uniform prior distributions throughout the range $0 \leq x_i \leq 1$ for $i = 1, 2$.

(a) First form a test set $\mathcal{D}_{test}$ of 20 points—10 points in ω_1 and 10 in ω_2—randomly chosen according to a uniform distribution.

(b) Next generate 100 points—50 patterns in each category. Set $\gamma = 0.1$ and split this set into a training set $\mathcal{D}_{train}$ (90 points) and validation set $\mathcal{D}_{val}$ (10 points).

(c) Now form a k-nearest-neighbor classifier in which k is increased until the first *minimum* of the validation error is found. (Restrict k to odd values, to avoid ties.) Now determine the error of your classifier using the test set.

(d) Repeat part (c), but instead find the k that is the first *maximum* of the validation error.

(e) Repeat parts (c) and (d) five times, noting the test error in all 10 cases.

(f) Discuss your results—in particular, how they depend or do not depend upon the fact that the data were all uniformly distributed.

8. Consider three candidate one-dimensional distributions, each parameterized by an unknown value for its "center":

- Gaussian: $p(x) \sim N(\mu, \sigma^2)$
- Triangle: $p(x) \sim T(\mu, 1) = \begin{cases} 1 - |x - \mu| & \text{for } |x - \mu| < 1 \\ 0 & \text{otherwise} \end{cases}$
- Uniform: $p(x) \sim U(\mu - 2, \mu + 2)$

Suppose we are sure that for each model the center must lie in the range $-1 < \mu < 1$, and for the Gaussian that $0 \leq \sigma^2 \leq 1$. Suppose too that we are given the data $\mathcal{D} = \{-0.9, -0.1, 0.0, 0.1, 0.9\}$. Clearly, the maximum-likelihood solution $\hat{\mu} = 0$ applies to each model.

(a) Estimate the Occam factor in each case.

(b) Use Bayesian model selection to choose the best of these models.

9. Problem 39 describes Bertrand's paradox, which involves the probability that a circle's chord "randomly chosen" will have length greater than that of an inscribed equilateral triangle.

(a) Write a program to generate chords according to the logic of solution (1) in Problem 39. Generate 1000 such chords and estimate empirically the probability that a chord has length greater than that of an inscribed equilateral triangle.

(b) Repeat part (a), assuming the logic underlying solution (2).

(c) Repeat part (a), assuming the logic underlying solution (3).

(d) Explain why there is little or no reason to prefer one of the solution methods over another, and thus the solution to the stated problem is ill-defined.

(e) Relate your answers above to the No Free Lunch Theorem (Theorem 9.1) and Bayesian model selection.

Section 9.7

10. Create a multiclassifier system for the data in the table above. As in Computer exercise 4, define two supercategories, where the 20 points in ω_1 and ω_2 form one category, ω_A, and the remaining 20 points form ω_B.

(a) Let the first component classifier be based on Gaussian priors, where the mean $\boldsymbol{\mu}_i$ is arbitrary and the covariance is estimated by maximum-likelihood (Chapter 3). What is training error measured using ω_A and ω_B?

(b) Let the second component classifier also be based on Gaussian priors, but where the covariance is arbitrary. Now what is the training error measured using ω_A and ω_B?

(c) Train your two-component classifier by gradient descent (Eqs. 58 and 59). What is the training error of the full system?

BIBLIOGRAPHY

[1] Hirotugu Akaike. On entropy maximization principle. In Paruchuri R. Krishnaiah, editor, *Applications of Statistics*, pages 27–42. North-Holland, Amsterdam, 1977.

[2] Hirotugu Akaike. A Bayesian analysis of the minimum AIC procedure. *Annals of the Institute of Statistical Mathematics*, 30A(1):9–14, 1978.

[3] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[4] Svetlana Anoulova, Paul Fischer, Stefan Pölt, and Hans-Ulrich Simon. Probably almost Bayes decisions. *Information and Computation*, 129(1):63–71, 1996.

[5] Martin Anthony and Norman Biggs, editors. *Computational Learning Theory: An Introduction*. Cambridge University Press, Cambridge, UK, 1992.

[6] András Antos, Luc Devroye, and László Györfi. Lower bounds for Bayes error estimation. *IEEE Transactions on Pattern Analysis and Applications*, PAMI-21(7):643–645, 1999.

[7] Andrew R. Barron and Thomas M. Cover. Minimum complexity density estimation. *IEEE Transactions on Information Theory*, IT-37(4):1034–1054, 1991.

[8] Charles H. Bennett, Péter Gács, Ming Li, Paul M. B. Vitányi, and Wojciech H. Zurek. Information distance. *IEEE Transactions on Information Theory*, IT-44(4):1407–1423, 1998.

[9] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, UK, 1995.

[10] Kevin W. Bowyer and P. Jonathon Phillips, editors. *Empirical Evaluation Techniques in Computer Vision*. IEEE Computer Society, Los Alamitos, CA, 1998.

[11] Leo Breiman. Bagging predictors. *Machine Learning*, 26(2):123–140, 1996.

[12] Leo Breiman. Stacked regressions. *Machine Learning*, 24(1):49–64, 1996.

[13] Leo Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–824, 1998.

[14] Gregory J. Chaitin. Information-theoretic computational complexity. *IEEE Transactions on Information Theory*, IT-20(1):10–15, 1974.

[15] Gregory J. Chaitin. *Algorithmic Information Theory*. Cambridge University Press, Cambridge, UK, 1987.

[16] Vladimir Cherkassky and Filip Mulier. *Learning from Data: Concepts, Theory, and Methods*. Wiley, New York, 1998.

[17] Bertrand S. Clarke and Andrew R. Barron. Information theoretic asymptotics of Bayes methods. *IEEE Transactions on Information Theory*, IT-36(3):453–471, 1990.

[18] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[19] David Cohn and Gerald Tesauro. How tight are the Vapnik-Chervonenkis bounds? *Neural Computation*, 4(2):249–269, 1992.

[20] David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. Active learning with statistical models. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 705–712. MIT Press, Cambridge, MA, 1995.

[21] Corinna Cortes, Larry D. Jackel, and Wan-Ping Chiang. Limits on learning machine accuracy imposed by data quality. In Gerald Tesauro, David S. Touretzky, and Tood K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 239–246. MIT Press, Cambridge, MA, 1995.

[22] Corinna Cortes, Larry D. Jackel, Sara A. Solla, Vladimir Vapnik, and John S. Denker. Learning curves: Asymptotic values and rate of convergence. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 327–334. Morgan Kaufmann, San Francisco, CA, 1994.

[23] Thomas M. Cover. Learning in pattern recognition. In Satoshi Watanabe, editor, *Methodologies of Pattern Recognition*, pages 111–132. Academic Press, New York, 1969.

[24] Belur V. Dasarathy, editor. *Decision Fusion*. IEEE Computer Society, Washington, DC, 1994.

[25] Anthony Christopher Davison and David V. Hinkley, editors. *Bootstrap Methods and Their Application*. Cambridge University Press, Cambridge, UK, 1997.

[26] Tom G. Dietterich. Overfitting and undercomputing in machine learning. *Computing Surveys*, 27(3):326–327, 1995.

[27] Robert P. W. Duin. A note on comparing classifiers. *Pattern Recognition Letters*, 17(5):529–536, 1996.

[28] Bradley Efron. *The Jackknife, the Bootstrap and Other Resampling Plans*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1982.

[29] Epicurus and Eugene Michael O'Connor (Editor). *The Essential Epicurus: Letters, Principal Doctrines, Vatican Sayings, and Fragments*. Prometheus Books, New York, 1993.

[30] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

[31] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1995.

[32] Jerome H. Friedman. On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

[33] Kenji Fukumizu. Active learning in multilayer perceptrons. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 295–301. MIT Press, Cambridge, MA, 1996.

[34] Keinosuke Fukunaga and Raymond R. Hayes. Effects of sample size in classifier design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-11:873–885, 1989.

[35] Stewart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Networks*, 4(1):1–58, 1992.

[36] Phillip I. Good. *Resampling Methods: A Practical Guide to Data Analysis*. Birkhauser, Boston, MA, 1999.

[37] Ulf Grenander. On empirical spectral analysis of stochastic processes. *Arkiv Matematiki*, 1(35):503–531, 1951.

[38] Stephen F. Gull. Bayesian inductive inference and maximum entropy. In Gary L. Ericson and C. Ray Smith, editors, *Maximum Entropy and Bayesian Methods in Science and Engineering 1: Foundations*, volume 1, pages 53–74. Kluwer, Boston, MA, 1988.

[39] Isabelle Guyon, John Makhoul, Richard Schwartz, and Vladimir Vapnik. What size test set gives good error rate estimates? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-20(1):52–64, 1998.

[40] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-12(10):993–1001, 1990.

[41] Trevor J. Hastie and Robert J. Tibshirani. *General Additive Models*. Chapman & Hall, London, 1990.

[42] Thomas Heskes. Bias/variance decompositions for likelihood-based estimators. *Neural Computation*, 10(6):1425–1433, 1998.

[43] Imad Y. Hoballah and Pramod K. Varshney. An information theoretic approach to the distributed detection problem. *IEEE Transactions on Information Theory*, IT-35(5):988–994, 1989.

[44] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[45] Robert C. Holte. Very simple classification rules perform well on most commonly used data sets. *Machine Learning*, 11(1):63–91, 1993.

[46] Gordon F. Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory*, IT-14(1):55–63, 1968.

[47] Robert A. Jacobs. Bias/variance analyses of mixtures-of-experts architectures. *Neural Computation*, 9(2):369–383, 1997.

[48] Anil K. Jain, Richard C. Dubes, and Chaur-Chin Chen. Bootstrap techniques for error estimation. *IEEE Transactions on Pattern Analysis and Applications*, PAMI-9(5):628–633, 1998.

[49] Sanjay Jain, Daniel Osherson, James S. Royer, and Arun Sharma. *Systems that Learn: An Introduction to Learning Theory*. MIT Press, Cambridge, MA, second edition, 1999.

[50] Chuanyi Ji and Sheng Ma. Combinations of weak classifiers. *IEEE Transactions on Neural Networks*, 8(1):32–42, 1997.

[51] Michael I. Jordan and Robert A. Jacobs. Hierarchies of adaptive experts. In John E. Moody, Stephen J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 985–992. Morgan Kaufmann, San Mateo, CA, 1992.

[52] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214, 1994.

[53] Michael J. Kearns. *The Computational Complexity of Machine Learning*. MIT Press, Cambridge, MA, 1990.

[54] Gary D. Kendall and Trevor J. Hall. Optimal network construction by minimum description length. *Neural Computation*, 5(2):210–212, 1993.

[55] Josef Kittler. Combining classifiers: A theoretical framework. *Pattern Analysis and Applications*, 1(1):18–27, 1998.

[56] Josef Kittler, Mohamad Hatef, Robert P. W. Duin, and Jiri Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Applications*, PAMI-20(3):226–239, 1998.

[57] Andreæi N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information and Transmission*, 1(1):3–11, 1965.

[58] Andreæi N. Kolmogorov and Vladimir A. Uspensky. On the definition of an algorithm. *American Mathematical Society Translations, Series 2*, 29:217–245, 1963.

[59] Eun Bae Kong and Thomas G. Dietterich. Error-correcting output coding corrects bias and variance. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 313–321. Morgan Kaufmann, San Francisco, CA, 1995.

[60] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. MIT Press, Cambridge, MA, 1995.

[61] Roman Krzysztofowicz and Dou Long. Fusion of detection probabilities and comparison of multisensor systems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-20(3):665–677, 1990.

[62] Sanjeev R. Kulkarni, Gábor Lugosi, and Santosh S. Venkatesh. Learning pattern classification – A survey. *IEEE Transactions on Information Theory*, IT-44(6):2178–2206, 1998.

[63] Dar-Shyang Lee and Sargur N. Srihari. A theory of classifier combination: The neural network approach. In *Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 14–16. IEEE Press, Los Alamitos, CA, 1995.

[64] Sik K. Leung-Yan-Cheong and Thomas M. Cover. Some equivalences between Shannon entropy and Kolmogorov complexity. *IEEE Transactions on Information Theory*, IT-24(3):331–338, 1978.

[65] Ming Li and Paul M. B. Vitányi. Inductive reasoning and Kolmogorov complexity. *Journal of Computer and System Sciences*, 44(2):343–384, 1992.

[66] Ming Li and Paul M. B. Vitányi. *Introduction to Kol-*

mogorov Complexity and Its Applications. Springer, New York, second edition, 1997.

[67] David J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992.

[68] David J. C. MacKay. Bayesian model comparison and backprop nets. In John E. Moody, Stephen J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 839–846. Morgan Kaufmann, San Mateo, CA, 1992.

[69] David J. C. MacKay. The evidence framework applied to classification networks. *Neural Computation*, 4(5):698–714, 1992.

[70] David J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

[71] Armand Maurer. *The Philosophy of William of Ockham in the Light of Its Principles*. Pontifical Institute of Mediaval Studies, Toronto, Ontario, 1999.

[72] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[73] Noboru Murata, Shuji Hoshizawa, and Shun-ichi Amari. Learning curves, model selection and complexity in neural networks. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 607–614. MIT Press, Cambridge, MA, 1993s.

[74] Isaac Newton and Alexander Koyre (Editor). *Isaac Newton's Philosophiae Naturalis Principia Mathematica*. Harvard University Press, Cambridge, MA, third edition, 1972.

[75] Michael Perrone and Leon N Cooper. When networks disagree: Ensemble methods for hybrid neural networks. In Richard J. Mammone, editor, *Artificial Neural Networks for Speech and Vision*, pages 126–142. Chapman & Hall, London, 1993.

[76] Karl Raimund Popper. *Conjectures and Refutations: The Growth of Scientific Knowledge*. Routledge Press, New York, fifth edition, 1992.

[77] Maurice H. Quenouille. Approximate tests of correlation in time series. *Journal of the Royal Statistical Society B*, 11:18–84, 1949.

[78] Maurice H. Quenouille. Notes on bias in estimation. *Biometrika*, 43:353–360, 1956.

[79] Willard V. Quine. *Ontological Relativity and other Essays*. Columbia University Press, New York, 1969.

[80] Jorma Rissanen. Modelling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[81] Jorma Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific, Singapore, 1989.

[82] Steven Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3):317–327, 1997.

[83] Cullen Schaffer. A conservation law for generalization performance. In William W. Cohen and Haym Hirsh, editors, *Proceeding of the Eleventh International Conference on Machine Learning*, pages 259–265. Morgan Kaufmann, San Francisco, CA, 1994.

[84] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[85] Robert E. Schapire. Using output codes to boost multiclass learning problems. In *Machine Learning: Proceedings of the Fourteenth International Conference*, pages 313–321. Morgan Kaufmann, San Mateo, CA, 1997.

[86] Gideon Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978.

[87] Holm Schwarze and John Hertz. Discontinuous generalization in large committee machines. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 399–406. Morgan Kaufmann, San Mateo, CA, 1994.

[88] Amanda J. C. Sharkey, editor. *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*. Springer-Verlag, London, 1999.

[89] Herbert Simon. Theories of bounded rationality. In C. Bartlett McGuire and Roy Radner, editors, *Decision and Organization: A Volume in Honor of Jacob Marschak*, chapter 8, pages 161–176. North-Holland, Amsterdam, 1972.

[90] Padhraic Smyth and David Wolpert. Stacked density estimation. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10, pages 668–674. MIT Press, Cambridge, MA, 1998.

[91] Padhraic Smyth and David Wolpert. An evaluation of linearly combining density estimators via stacking. *Machine Learning*, 36(1/2):59–83, 1999.

[92] Elliott Sober. *Simplicity*. Oxford University Press, New York, 1975.

[93] Ray J. Solomonoff. A formal theory of inductive inference. Part I. *Information and Control*, 7(1):1–22, 1964.

[94] Ray J. Solomonoff. A formal theory of inductive inference. Part II. *Information and Control*, 7(2):224–254, 1964.

[95] David G. Stork. Document and character research in the Open Mind Initiative. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR99)*, pages 1–12, Bangalore, India, September 1999.

[96] David G. Stork. The Open Mind Initiative. *IEEE Expert Systems and Their Application*, 14(3):19–20, 1999.

[97] Stelios C. A. Thomopoulos, Ramanarayanan Viswanathan, and Dimitrios C. Bougoulias. Optimal decision fusion in multiple sensor systems. *IEEE Transactions on Aerospace and Electronic Systems*, AES-23(5):644–653, 1987.

[98] Godfried T. Toussaint. Bibliography on estimation of misclassification. *IEEE Transactions on Information Theory*, IT-20(4):472–279, 1974.

[99] Les Valiant. Theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[100] Jean-Marc Valin and David G. Stork. Open Mind speech recognition. In *Proceedings of the Automatic*

Speech Recognition and Understanding Workshop (ASRU99), Keystone, CO, 1999.

[101] Vladimir Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, New York, 1995.

[102] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[103] Vladimir Vapnik and Aleksei Chervonenkis. *Theory of Pattern Recognition*. Nauka, Moscow, Russian edition, 1974.

[104] Šarūnas Raudys and Anil K. Jain. Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *IEEE Transactions on Pattern Analysis and Applications*, PAMI-13(3):252–264, 1991.

[105] Satoshi Watanabe. *Pattern Recognition: Human and Mechanical*. Wiley, New York, 1985.

[106] Ole Winther and Sara A. Solla. Optimal Bayesian online learning. In Kwok-Yee Michael Wong, Irwin King, and Dit-Yan Yeung, editors, *Theoretical Aspects of Neural Computation: A Multidisciplinary Perspective*, pages 61–70. Springer, New York, 1998.

[107] Greg Wolff, Art Owen, and David G. Stork. Empirical error-confidence curves for neural network and Gaussian classifiers. *International Journal of Neural Systems*, 7(3):363–371, 1996.

[108] David H. Wolpert. On the connection between in-sample testing and generalization error. *Complex Systems*, 6(1):47–94, 1992.

[109] David H. Wolpert, editor. *The Mathematics of Generalization*. Addison-Wesley, Reading, MA, 1995.

[110] David H. Wolpert. The relationship between PAC, the statistical physics framework, the Bayesian framework, and the VC framework. In David H. Wolpert, editor, *The Mathematics of Generalization*, pages 117–214. Addison-Wesley, Reading, MA, 1995.

[111] Frank J. Wyman, Dean M. Young, and Danny W. Turner. A comparison of asymptotic error rate for the sample linear discriminant function. *Pattern Recognition*, 23(7):775–785, 1990.