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, every reader 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, or 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 limit, 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; 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 as for the nearest-neighbor or for 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 in this chapter because of their breadth of applicability and independence from the details of the learning techniques encountered up to here.

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 accuracy (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 any of a number of algorithms.

We shall present the results that are most important for pattern recognition practitioners, occasionally skipping over mathematical details 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*: on the criterion of generalization performance, there are no context- or problem-independent reasons to favor one learning or classification method over another. The apparent superiority of one algorithm or set of algorithms is due to the nature of the problems investigated and the distribution of data. It is an appreciation of the No Free Lunch Theorem that allows us, when confronting practical pattern recognition problems, 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 scepticism about studies that purport to demonstrate the overall superiority of a particular learning or recognition algorithm.

When comparing algorithms we sometimes focus on generalization error for points *not* in the training set $\mathcal{D}$, rather than the more traditional independent identically distributed or i.i.d. case. We do this for several reasons: First, 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. 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. In short, it is the *off-training set error* — the error on points *not* in the training set — that is a better measure for distinguishing algorithms. Of course, for most applications the final performance of a fielded classifier is the full i.i.d. error.

OFF-TRAINING SET ERROR

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 non-zero 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, etc. Further, $P(h)$ 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, $P(h|\mathcal{D})$ will be a broad distribution. For a general loss function $L(\cdot,\cdot)$ we let $E = L(\cdot,\cdot)$ be the scalar error or cost. While the natural loss function for regression is a sum-square error, for classification we focus on zero-one loss, and thus the generalization error is the expected value of E.

How shall we judge the generalization quality of a learning algorithm? Since 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 some candidate learning algorithm is $P_k(h(\mathbf{x})|\mathcal{D})$ is given by

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

With this background and the terminology of Eq. 2 we can now turn to a formal 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 error for all learning algorithms is the same, i.e.,

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

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 you use, there is at least one target function for which random guessing is a better algorithm.

Assuming the training set can be learned by all algorithms in question, then 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 error that is superior to any other, i.e.,

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

Parts *3* & *4* concern non-uniform target function distributions, and have related interpretations (Problems 2 – 5). Example 1 provides an elementary illustration.

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

[*] The clever name for the Theorem was suggested by David Haussler.

$n = 3$ 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 $\mathcal{E}_2(E|F, \mathcal{D}) = 0.6$.

	$\mathbf{x}$	F	h_1	h_2
	000	1	1	1
$\mathcal{D}$	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 a). No system can perform well throughout the full set of functions, 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. There is, hence, 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, and the other results from the No Free Lunch Theorem, stress that it is the *assumptions* 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.

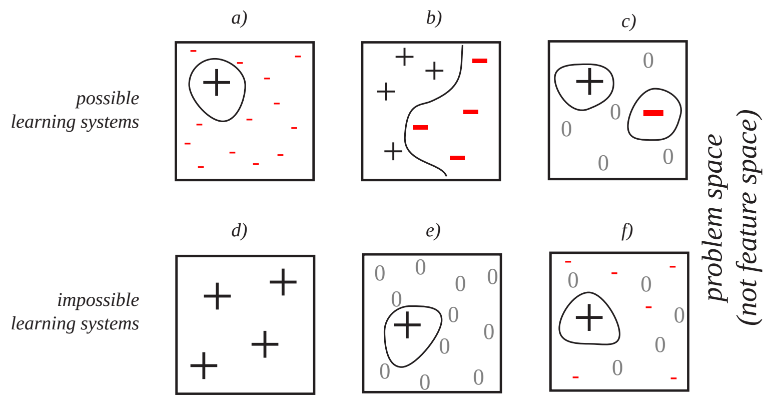

a)
b)
c)
d)
e)
f)
possible learning systems
impossible learning systems
problem space (not feature space)

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 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, 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, 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 d). It is also impossible to have higher than average performance on some problems, and average performance everywhere else, shown in e).

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 which may or may not be correct.

Since we are using discrete representations, we can use logical expressions or "predicates" to describe a pattern, much as in Chap. **??**. 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 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, since for every configuration of f_1 and f_2 there is

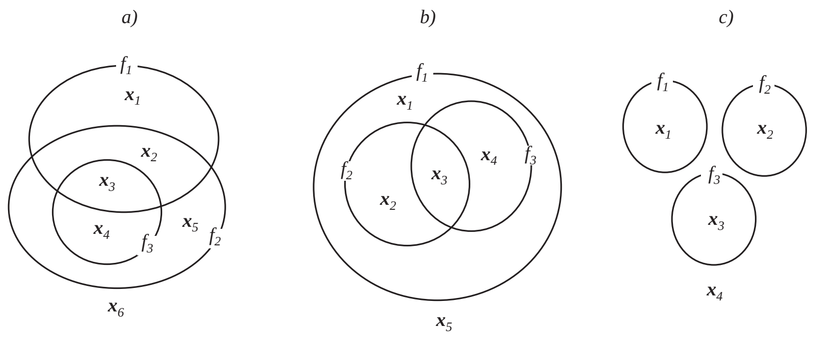


Figure 9.2: Patterns $\mathbf{x}_i$, represented as d-tuples of binary features f_i, can be placed in Venn diagram (here $d = 3$); the diagram itself depends upon the classification problem and its constraints. For instance, suppose f_1 is the binary feature attribute `has_legs`, f_2 is `has_right_arm` 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.

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.

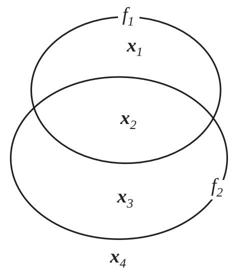


Figure 9.3: The Venn for a problem with no constraints on two features. Thus all four binary attribute vectors can occur.

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 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

* 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.

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 $\mathbf{x}_3$	NOT(f_1 AND f_2)
$\mathbf{x}_2$ OR $\mathbf{x}_3$ OR $\mathbf{x}_4$	f_2 OR NOTf_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. Further, 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	f'_1	f'_2
$\mathbf{x}_1$	0	0	0	1
$\mathbf{x}_2$	0	1	0	0
$\mathbf{x}_3$	1	0	1	0
$\mathbf{x}_4$	1	1	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 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 $r = 2$, i.e., $\mathbf{x}_i\ OR\ \mathbf{x}_j$. A predicate of rank $r = 3$ must contain three patterns, two of which are $\mathbf{x}_i$ and $\mathbf{x}_j$. Since 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: Eq. 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:

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 simple yet important: 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 spaces too, if such as 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 principle argues 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

[*] The Theorem gets its fanciful name from the following counter-intuitive statement: Assuming 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 from one another.

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 to quantify an inherent complexity of a binary string. (We shall assume both classifiers and patterns are described by such strings.) Algorithmic complexity can be explained by analogy to communication, the earliest application of information theory (App. **??**). If the sender and receiver agree upon a specification method L, such as an encoding or compression technique, then message x can then be transmitted as y, denoted $L(y) = x$ or $y : L(y) = x$. 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 $\min\limits_{|y|} : L(y) = x$; this minimal length is the entropy of x under the specification or transmission method L.

ABSTRACT COMPUTER

Algorithmic complexity is defined by analogy to entropy, where instead of a specification method L, we consider programs running on an *abstract computer*, i.e., 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), so that we can compare the complexities of different binary strings. Such a method would provide a measure of the inherent information content, the amount of data which 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 y, measured in bits, that without additional data computes the string x and halts. Formally, we write

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

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 (Chap. **??**).

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 $K(x) = \mathcal{O}(\log_2 n)$. Next consider the transcendental number π, whose infinite sequence of seemingly random binary digits, $11.0010010000111111011010101001\ldots_2$, 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 $K(\pi) = \mathcal{O}(1)$, which means $K(\pi)$ 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 $K(x) = \mathcal{O}(|x|)$, 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, i.e.,

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

Thus we seek the model h^* that obeys $h^* = \arg\min_h K(h, \mathcal{D})$ (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 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 (Chap. **??**). 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 Theorems. Moreover, in practice it is often difficult to compute the minimum description length, since we may not be clever enough to find the "best" representation (Problem 17). Assume 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 since 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, i.e.,

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

much as we saw in Chap. **??**. 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 tradeoff between simplifying the model and fitting the data in the bias-variance dilemma in Sec. 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 Sect. 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 (Computer exercise **??**).

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 counselling 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 non-optimal 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 non-uniform prior over learning algorithms — 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 it 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)

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

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 on average we accurately estimate F from $\mathcal{D}$. Further, a low variance means 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, etc. — 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) at the left, 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. Further, 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 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 the fit found for every training set is quite good. Thus the bias is low, as shown in the histogram at the bottom. The model in d) is linear in x, but its slope and intercept are determined from the training data. As such, the model in d) has a lower bias than the models in 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 model has few parameters (generally high bias), it may not fit the data particularly well, but this fit will not change much as 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 there is only one learning problem to be solved, in which case the answer is already

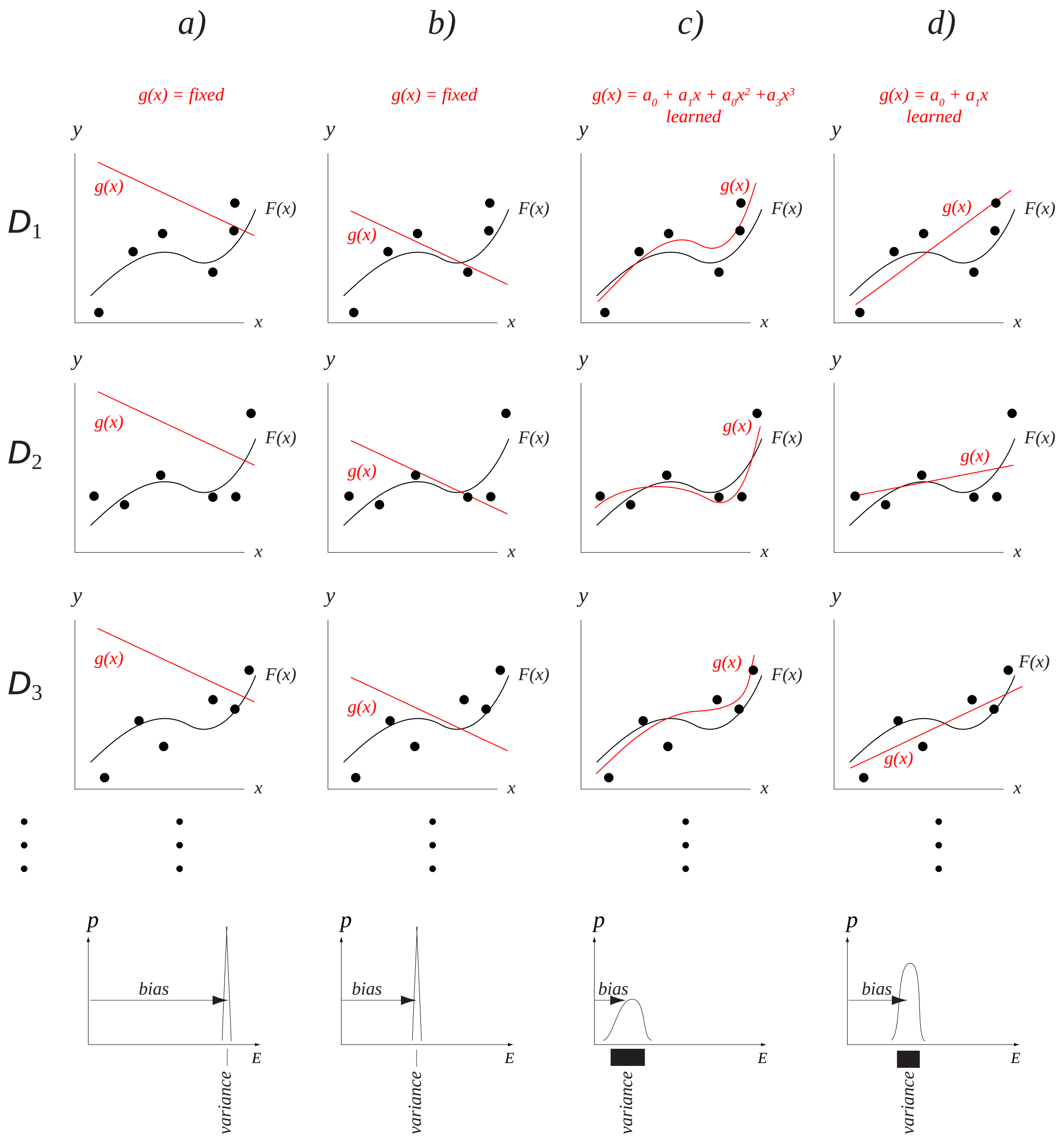


Figure 9.4: The bias-variance dilemma can be illustrated in the domain of regression. Each column represents a different model, each row a different set of $n = 6$ training points, $\mathcal{D}_i$, randomly sampled from the true function $F(x)$ with noise. Histograms of the mean-square error of $E \equiv \mathcal{E}_{\mathcal{D}}[(g(x) - F(x))^2]$ 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 a) and 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 $n \to \infty$ of points, the bias in c) would approach a small value (which depends upon the noise), while the bias in d) would not; the variance of all models would approach zero.

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, i.e.,

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

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 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 $\text{Var}[\epsilon|\mathbf{x}] = F(\mathbf{x})(1 - F(\mathbf{x}))$. 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 $g(\mathbf{x};\ \mathcal{D})$ which 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 Sect. 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

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

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 mis-estimation of the optimal (Bayes) boundary (Problem 19).

BOUNDARY ERROR

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 Chap. **??**, Fig. **??**:

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

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

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

where

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

and erf[·] is the familiar error function (App. **??**).

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. Further, 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 *boundarybias* affects the role of variance in the error. For this reason low *variance* is generally important for accurate classification while low *boundarybias* need not be. Or said another way, in classification, *variance* generally dominates *bias*. In practical terms, this implies 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, etc. — affect the bias and variance of a classifier; in Sect. 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.

BOUNDARY BIAS

As an illustration of *boundarybias* 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 have just a few samples from each category and estimate the parameters in three different classes of models by maximum likelihood. Column a) at the left shows the most general Gaussian classifiers; each component distribution can have arbitrary covariance matrix. Column b) at the middle shows classifiers where each component Gaussian is constrained to have a diagonal covariance. Column c) at the right 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 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 left-most density plot shows a very broad distribution (high variance). The right-most 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 left-most algorithm will be equal to the true decision boundary — this algorithm has no bias. The right-most average will be a vertical line, and hence there will be higher error — this algorithm has the highest bias of the three. Histograms 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 ($n \to \infty$), all error histograms become narrower and move to lower values of E. If a model is rich enough to express the optimal decision boundary, its error histogram 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 $F(\mathbf{x})$. Further, 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 & 9.5 suggest a method using multiple samples, an inspiration for formal "resampling"

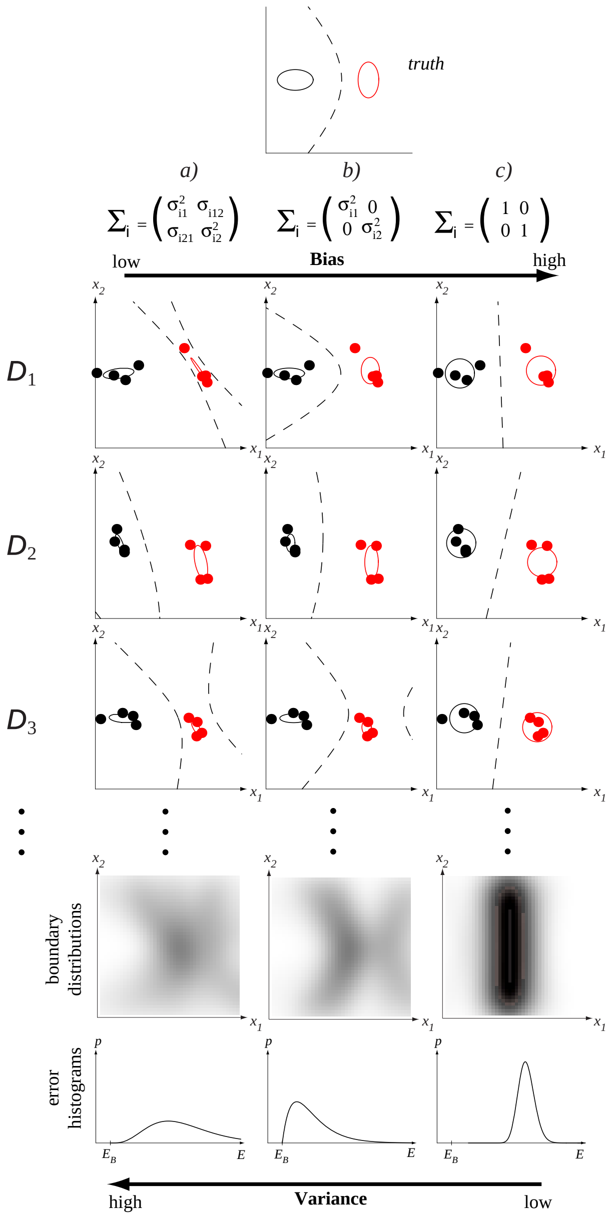

truth
a)
b)
c)
$\Sigma_i = \begin{pmatrix} \sigma_{i1}^2 & \sigma_{i12} \\ \sigma_{i21} & \sigma_{i2}^2 \end{pmatrix}$
$\Sigma_i = \begin{pmatrix} \sigma_{i1}^2 & 0 \\ 0 & \sigma_{i2}^2 \end{pmatrix}$
$\Sigma_i = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
low
Bias
high
D_1
D_2
D_3
x_1
x_2
boundary distributions
error histograms
p
E_B
E
high
Variance
low

Figure 9.5: The (boundary) bias-variance tradeoff in classification can be illustrated with a two-dimensional Gaussian problem. The figure at the top shows the (true) decision boundary of the Bayes classifier. The nine figures in the middle show nine 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 the decision boundaries learning by fitting a Gaussian model with fully general covariance matrices 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) at the right 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.

methods, which we now discuss. Later we shall turn to our ultimate goal: using resampling and related techniques to improve classification (Sect. 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 standard deviation, given by

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

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* 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 (Sect. 9.4.2) are two of the most popular and theoretically grounded resampling techniques for extending the above approach (based on Eqs. 22 & 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

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

i.e., 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, i.e., $\hat{\mu} = \mu_{(\cdot)}$ (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 & 26 above.

Jackknife bias estimate

The notion of bias is more general that that described in Sect. 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, i.e., BIAS

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

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 $\hat{\theta}_{(\cdot)}$. 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 θ itself 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 a quadratic function, 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:

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

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

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

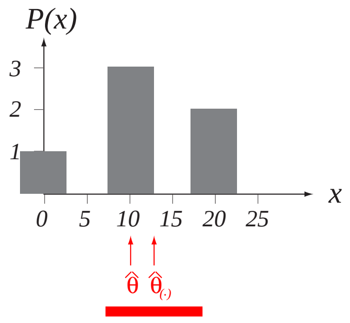
where as before $\hat{\theta}_{(\cdot)} = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_{(i)}$.

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 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 range of probable values of the mode. This range 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. (Since $\mathcal{D}$ 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 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 **??**)

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

Computer exercise 45 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.

TRIMMED MEAN

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

* "Bootstrap" comes from 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.

for training classifiers. These are related to methods for estimating and comparing classifier models that we will discuss in Sect. 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 Sect. 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 — i.e., 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 Chap. **??**, 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 since it effectively averages over such discontinuities. There are no convincing theoretical derivations or simulation studies showing that bagging will help all stable 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 Sect. 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 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*, i.e., 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

WEAK LEARNER

[*] In Sect. 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.

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 29). 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 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 which 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 the combined system C_1 and C_2. We randomly select a training pattern from those remaining in $\mathcal{D}$, and 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 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 but little 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 **??**).

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 which determines its probability of being selected for a training set for an individual component classifier. If a training pattern is accurately

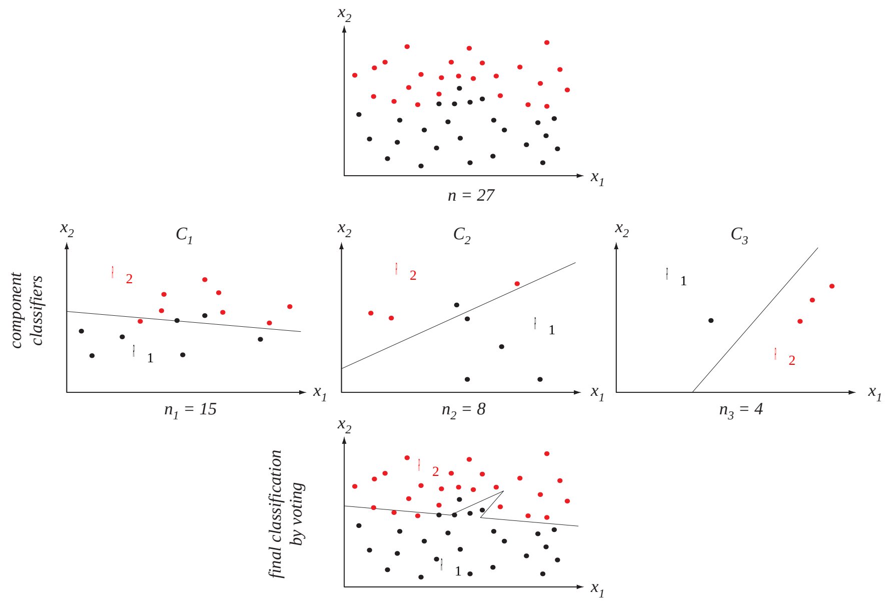


Figure 9.6: A two-dimensional two-category classification task is shown at the top. The middle row shows three component (linear) classifiers C_k trained by LMS algorithm (Chap. **??**), where their training patterns were chosen through the basic boosting procedure. The final classification is given 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 better than chance), then the ensemble classifier will have a lower training error on the full training set $\mathcal{D}$ than does any single component classifier.

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 these weights across the training set to to be uniform. On each iteration k, we draw a training set at random according to these weights, and 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 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. The AdaBoost procedure is then:

Algorithm 1 (AdaBoost)

```
1 begin initialize $\mathcal{D} = \{\mathbf{x}^1, y_1, \mathbf{x}^2, y_2, \ldots, \mathbf{x}^n, y_n\}, k_{max}, W_1(i) = 1/n, i = 1, \ldots, n$
2        $k \leftarrow 0$
3        do  $k \leftarrow k + 1$
4            Train weak learner $C_k$ using $\mathcal{D}$ sampled according to distribution $W_k(i)$
5            $E_k \leftarrow$ Training error of $C_k$ measured on $\mathcal{D}$ using $W_k(i)$
6            $\alpha_k \leftarrow \frac{1}{2}\ln[(1 - E_k)/E_k]$
```

```
7           $W_{k+1}(i) \leftarrow \frac{W_k(i)}{Z_k} \times \begin{cases} e^{-\alpha_k} & \text{if } h_k(\mathbf{x}^i) = y_i \text{ (correctly classified)} \\ e^{\alpha_k} & \text{if } h_k(\mathbf{x}^i) \neq y_i \text{ (incorrectly classified)} \end{cases}$
8         until $k = k_{max}$
9     return $C_k$ and $\alpha_k$ for $k = 1$ to $k_{max}$ (ensemble of classifiers with weights)
10 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 insure 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 31):

$$\begin{aligned} E = \prod_{k=1}^{k_{max}} \left[2\sqrt{E_k(1-E_k)} \right] &= \prod_{k=1}^{k_{max}} \sqrt{1 - 4G_k^2} \\ &\leq \exp\left(-2 \sum_{k=1}^{k_{max}} G_k^2 \right), \end{aligned} \tag{37}$$

as illustrated in Fig. 9.7. It is sometimes beneficial to increase k_{max} beyond the value needed for zero ensemble training error since this may improve generalization. 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 alternate learning algorithm. Moreover, the exponential reduction in error on the training set does not insure reduction of the *off-training set error* or generalization, as we saw in Sect. 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 there was a set of labeled training patterns $\mathcal{D}$ and employed resampling methods to reuse patterns to improve classification. In