Chapter 1

Introduction and Overview

How is it that a committee of blockheads can somehow arrive together at highly reasoned decisions, despite the weak judgment of the individual members? How can the shaky, separate views of a panel of dolts be combined into a single opinion that is very likely to be correct? That this possibility of garnering wisdom from a council of fools can be harnessed and used to advantage may seem far-fetched and implausible, especially in real life. Nevertheless, this unlikely tack turns out to form the basis of boosting, an approach to machine learning that is the topic of this book. Indeed, at its core, boosting solves hard machine-learning problems by forming a very smart committee of grossly incompetent but carefully selected members.

To see how this might work in the context of machine learning, consider the problem of filtering out spam, or junk email. Spam is a modern-day nuisance, and one that is ideally handled by highly accurate filters that can identify and remove spam from the flow of legitimate email. Thus, to build a spam filter, the main problem is to create a method by which a computer can automatically categorize email as spam (junk) or ham (legitimate). The machine learning approach to this problem prescribes that we begin by gathering a collection of examples of the two classes, that is, a collection of email messages which have been labeled, presumably by a human, as spam or ham. The purpose of the machine learning algorithm is to automatically produce from such data a prediction rule that can be used to reliably classify new examples (email messages) as spam or ham.

For any of us who has ever been bombarded with spam, rules for identifying spam or ham will immediately come to mind. For instance, if it contains the word *viagra*, then it is probably spam. Or, as another example, email from one's spouse is quite likely to be ham. Such individual rules of thumb are far from complete as a means of separating spam from ham. A rule that classifies all email containing *viagra* as spam, and all other email as ham, will be wrong very often. On the other hand, such a rule is undoubtedly telling us something useful and non-trivial, and its accuracy, however poor, will nonetheless be significantly better than simply guessing entirely at random as to whether each email is spam or ham.

Intuitively, finding these weak rules of thumb should be relatively easy—so easy, in fact, that one might reasonably envision a kind of automatic "weak learning" program that, given any set of email examples, could effectively search for a simple prediction rule that may be rough and rather inaccurate, but that nonetheless provides some non-trivial guidance in separating the given examples as spam or ham. Furthermore, by calling such a weak learning program repeatedly on various subsets of our dataset, it would be possible to extract a whole collection of rules of thumb. The main idea of boosting is to somehow combine these weak and inaccurate rules of thumb into a single "committee" whose overall predictions will be quite accurate.

In order to use these rules of thumb to maximum advantage, there are two critical problems that we face: First, how should we choose the collections of email examples presented to the weak learning program so as to extract rules of thumb that will be the most useful? And second, once we have collected many rules of thumb, how can they be combined into a single, highly accurate prediction rule? For the latter question, a reasonable approach is simply for the combined rule to take a vote of the predictions of the rules of thumb. For the first question, we will advocate an approach in which the weak learning program is forced to focus its attention on the "hardest" examples, that is, the ones for which the previously chosen rules of thumb were most apt to give incorrect predictions.

Boosting refers to a general and provably effective method of producing a very accurate prediction rule by combining rough and moderately inaccurate rules of thumb in a manner similar to that suggested above. This book presents in detail much of the recent work on boosting, focusing especially on the *AdaBoost* algorithm which has undergone intense theoretical study and empirical testing. In this first chapter, we introduce AdaBoost and some of the key concepts required for its study. We also give a brief overview of the entire book.

1.1 Classification problems and machine learning

This book focuses primarily on *classification* problems in which the goal is to categorize objects into one of a relatively small set of classes. For instance, an optical-character recognition (OCR) system must classify images of letters into the categories A, B, C, etc. Medical diagnosis is another example of a classification problem in which the goal is to diagnose a patient. In other words, given the symptoms manifested by the patient, our goal is to categorize him or her as a sufferer or non-sufferer of a particular disease. The spam-filtering example is also a classification problem in which we attempt to categorize emails as spam or ham.

We focus especially on a machine-learning approach to classification problems. Machine learning studies the design of automatic methods for making predictions about the future based on past experiences. In the context of classification problems, machine-learning methods attempt to learn to predict the correct classifications of unseen examples through the careful examination of examples which were previously labeled with their correct classifications, usually by a human.

We refer to the objects to be classified as *instances*. Thus, an instance is a description of some kind which is used to derive a predicted classification. In the OCR example, the instances are the images of letters. In the medical-diagnosis example, the instances are descriptions of a patient's symptoms. The space of all possible instances is called the *instance space* or *domain*, and is denoted by $\mathcal{X}$. A *(labeled) example* is an instance together with an associated *label* indicating its correct classification. Instances are also sometimes referred to as (unlabeled) examples.

During training, a *learning algorithm* receives as input a *training set* of labeled examples called the *training examples*. The output of the learning algorithm is a prediction rule called a *classifier* or *hypothesis*. A classifier can itself be thought of as a computer program which takes as input a new unlabeled instance and outputs a predicted classification; so, in mathematical terms, a classifier is a function that maps instances to labels. In this book, we use the terms *classifier* and *hypothesis* fairly interchangeably, with the former emphasizing a prediction rule's use in classifying new examples, while the latter emphasizes the fact that the rule has been (or could be) generated as the result of some learning process. Other terms that have been used in the literature include *rule*, *prediction rule*, *classification rule*, *predictor*, and *model*.

To assess the quality of a given classifier, we measure its error rate, that is, its frequency of making incorrect classifications. To do this, we need a *test set*, a separate set of *test examples*. The classifier is evaluated on each of the test instances and its predictions are compared to the correct classifications of the test examples. The fraction of examples on which incorrect classifications were made is called the *test error* of the classifier. Similarly, the fraction of mistakes on the training set is called the *training error*. The fraction of correct predictions is called the (test or training) *accuracy*.

Of course, the classifier's performance on the training set is not of much interest since our purpose is to build a classifier that works well on unseen data. On the other hand, if there is no relationship at all between the training set and test set, then the learning problem is unsolvable; the future can only be predicted if it resembles the past. Therefore, in designing and studying learning algorithms, we generally assume that the training and test examples are taken from the *same* random source. That is, we assume that the examples are chosen randomly from some fixed but unknown distribution $\mathcal{D}$ over the space of labeled examples, and moreover that the training and test examples are generated by the *same* distribution. The *generalization error* of a classifier measures the probability of misclassifying a random example from this distribution $\mathcal{D}$; equivalently, the generalization error is the expected test error of the classifier on any test set generated by $\mathcal{D}$. The goal of the learning algorithm can now be stated clearly to be the minimization of the generalization error of the classifier that it produces.

To illustrate these concepts, consider the problem of diagnosing a patient with coronary artery disease. For this problem, an instance consists of a description of the patient including items such as sex, age, cholesterol level, chest pain type (if any), blood pressure and results of various other medical tests. The label or class associated with each instance is a diagnosis provided by a doctor as to whether or not the patient described actually suffers from the disease. During training, a learning algorithm is provided with a set of labeled examples and attempts to produce a classifier for predicting if new patients suffer from the disease. The goal is to produce a classifier that is as accurate as possible. Later, in Section 1.2.3, we describe experiments using a publicly available dataset for this problem.

1.2 Boosting

We can now make some of the informal notions about boosting described above more precise. Boosting assumes the availability of a *base* or *weak learning algorithm* which, given labeled training examples, produces a *base* or *weak classifier*. The goal of boosting is to improve the performance of the weak learning algorithm while treating it as a "black box" which can be called repeatedly, like a subroutine, but whose innards cannot be observed or manipulated. We wish to make only the most minimal assumptions about this learning algorithm. Perhaps the least we can assume is that the weak classifiers are not entirely trivial in the sense that their error rates are at least a little bit better than a classifier whose every prediction is a random guess. Thus, like the rules of thumb in the spam-filtering example, the weak classifiers can be rough and moderately inaccurate, but not entirely trivial and uninformative. This assumption, that the base learner produces a weak hypothesis that is at least slightly better than random guessing on the examples on which it was trained, is called the *weak learning assumption*, and it is central to the study of boosting.

As with the words *classifier* and *hypothesis*, we use the terms *base* and *weak* roughly interchangeably, with *weak* connoting mediocrity in performance, and *base* connoting use as a building block.

Like any learning algorithm, a boosting algorithm takes as input a set of training examples $(x_1, y_1), \ldots, (x_m, y_m)$ where each x_i is an instance from $\mathcal{X}$, and each y_i is the associated label or class. For now, and indeed for most of this book, we assume the simplest case in which there are only two classes, -1 and $+1$, although we do explore extensions to multiclass problems in Chapter 10.

A boosting algorithm's only means of learning from the data is through calls to the base learning algorithm. However, if the base learner is simply called repeatedly, always with the same set of training data, we cannot expect anything interesting to happen; instead, we expect the same, or nearly the same, base classifier to be produced over and over again so that little is gained over running the base learner just once. This shows that the boosting algorithm, if it is to actually improve on the base learner, must in some way manipulate the data that it feeds to it.

Indeed, the key idea behind boosting is to choose training sets for the base learner in such a fashion as to force it to infer something new about the data each time it is called. This can be accomplished by choosing training sets on which we can reasonably expect the performance of the preceding base classifiers to be very poor—poorer even than their regular weak performance. If this can be accomplished, then we can expect the base learner to output a new base classifier which is significantly different from its predecessors. This is because, although we think of the base learner as a weak and mediocre learning algorithm, we nevertheless expect it to output classifiers that make non-trivial predictions.

We are now ready to describe in detail the boosting algorithm AdaBoost, which incorporates these ideas, and whose pseudocode is shown as Algorithm 1.1. (Refer to Appendix A for clarification of the notation used here and throughout this book, as well as some brief, mathematical background.) AdaBoost proceeds in *rounds* or iterative calls to the base learner. For the purpose of choosing the training sets provided to the base learner on each round, AdaBoost maintains a distribution over the training examples. The distribution used on the t-th round is denoted D_t. The weight that this distribution assigns to training example i on round t is denoted $D_t(i)$. Intuitively, this weight is a measure of the importance of correctly classifying example i on the current round. Initially, all weights are set equally, but on each round, the weights of incorrectly classified examples are increased so that, effectively, hard examples get successively higher weight, forcing the base learner to focus its attention on them.

The base learner's job is to find a base classifier $h_t : \mathcal{X} \rightarrow \{-1, +1\}$ appropriate for the distribution D_t. Consistent with the earlier discussion, the quality of a

```
Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.
Initialize: $D_1(i) = 1/m$ for $i = 1, \ldots, m$.
For $t = 1, \ldots, T$:

  • Train weak learner using distribution $D_t$.
  • Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$.
  • Aim: select $h_t$ to minimalize the weighted error:
        $\epsilon_t \doteq \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].$
  • Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$.
  • Update, for $i = 1, \ldots, m$:
        $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$
        $= \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
    where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution).

Output the final hypothesis:
    $H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$
```

Algorithm 1.1: The boosting algorithm AdaBoost.

base classifier is measured by its error *weighted* by the distribution D_t:

$$\epsilon_t \doteq \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right] = \sum_{i:h_t(x_i)\neq y_i} D_t(i).$$

Here, $\Pr_{i \sim D_t}[\cdot]$ denotes probability with respect to the random selection of an example (as specified by its index i) according to distribution D_t. Thus, the weighted error ϵ_t is the chance of h_t misclassifying a random example if selected according to D_t. Equivalently, it is the sum of the weights of the misclassified examples. Notice that the error is measured with respect to the same distribution D_t on which the base classifier was trained.

The weak learner attempts to choose a weak hypothesis h_t with low weighted error ϵ_t. In this setting, however, we do not actually expect that this error will be especially small in an absolute sense, but only in a more general and relative sense; in particular, we expect it to be only a bit better than random, and typically far from zero. To emphasize this looseness in what we require of the weak learner, we say that the weak learner's aim is to *minimalize* the weighted error, using this word to signify a vaguer and less stringent diminishment than that connoted by *minimize*.

If a classifier makes each of its predictions entirely at random, choosing each predicted label to be -1 or $+1$ with equal probability, then its probability of misclassifying any given example will be exactly $1/2$. Therefore, the error of this classifier will always be $1/2$, regardless of the data on which the error is measured. Thus, a weak hypothesis with weighted error ϵ_t equal to $1/2$ can be obtained trivially by formulating each prediction as a random guess. The weak learning assumption then, for our present purposes, amounts to an assumption that the error of each weak classifier is bounded away from $1/2$ so that each ϵ_t is at most $1/2 - \gamma$ for some small positive constant γ. In this way, each weak hypothesis is assumed to be slightly better than random guessing by some small amount, as measured by its error. (This assumption will be refined considerably in Section 2.3.)

As for the weights $D_t(i)$ that AdaBoost calculates on the training examples, in practice, there are several ways in which these can be used by the base learner. In some cases the base learner can use these weights directly. In other cases an unweighted training set is generated for the base learner by selecting examples at random from the original training set. The probability of selecting an example in this case is set to be proportional to the weight of the example. These methods are discussed in more detail in Section 3.4.

Relating back to the spam-filtering example, the instances x_i correspond to email messages, and the labels y_i give the correct classification as spam or ham. The base classifiers are the rules of thumb provided by the weak learning program where the subcollections on which it is run are chosen randomly according to the distribution D_t.

Once the base classifier h_t has been received, AdaBoost chooses a parameter α_t as in Algorithm 1.1. Intuitively, α_t measures the importance that is assigned to h_t. The precise choice of α_t is unimportant for our present purposes; the rationale for this particular choice will become apparent in Chapter 3. For now, it is enough to observe that $\alpha_t > 0$ if $\epsilon_t < 1/2$, and that α_t gets larger as ϵ_t gets smaller. Thus, the more accurate the base classifier h_t, the more importance that we assign to it.

The distribution D_t is next updated using the rule shown in the algorithm. First, all of the weights are multiplied either by $e^{-\alpha_t} < 1$ for examples incorrectly classified by h_t, or by $e^{\alpha_t} > 1$ for correctly classified examples. Equivalently, since we are using labels and predictions in $\{-1, +1\}$, this update can be expressed more succinctly as a scaling of each example i by $\exp(-\alpha_t y_i h_t(x_i))$. Next, the resulting set of values is renormalized by dividing through by the factor Z_t to ensure that the new distribution D_{t+1} does indeed sum to 1. The effect of this rule is to increase the weights of examples misclassified by h_t, and to decrease the weights of correctly classified examples. Thus, the weight tends to concentrate on "hard" examples. Actually, to be more precise, AdaBoost chooses a new distribution D_{t+1} on which the last base classifier h_t is sure to do extremely poorly: It can be shown by a simple computation that the error of h_t with respect to distribution D_{t+1} is exactly $1/2$, that is, exactly the trivial error rate achievable through simple random guessing. In this way, as discussed above, AdaBoost tries on each round to force the base learner to learn something new about the data.

After many calls to the base learner, AdaBoost combines the many base classifiers into a single *combined* or *final classifier* H. This is accomplished by a simple weighted vote of the base classifiers. That is, given a new instance x, the combined classifier evaluates all of the base classifiers, and predicts with the weighted majority of the base classifiers' predicted classifications. Here, the vote of the t-th base classifier h_t is weighted by the previously chosen parameter α_t. The resulting formula for H's prediction is as shown in the algorithm.

1.2.1 A toy example

To illustrate how AdaBoost works, let us look at the tiny toy learning problem shown in Figure 1.1. Here, the instances are points in the plane which are labeled $+$ or $-$. In this case, there are $m = 10$ training examples, as shown in the figure: 5 are positive and 5 negative.

Let us suppose that our base learner finds classifiers defined by vertical or horizontal lines through the plane. For instance, such a base classifier defined by a vertical line might classify all points to the right of the line as positive, and all points to the left as negative. It can be checked that no base classifier of this form correctly classifies more than 7 of the 10 training examples, meaning that none has

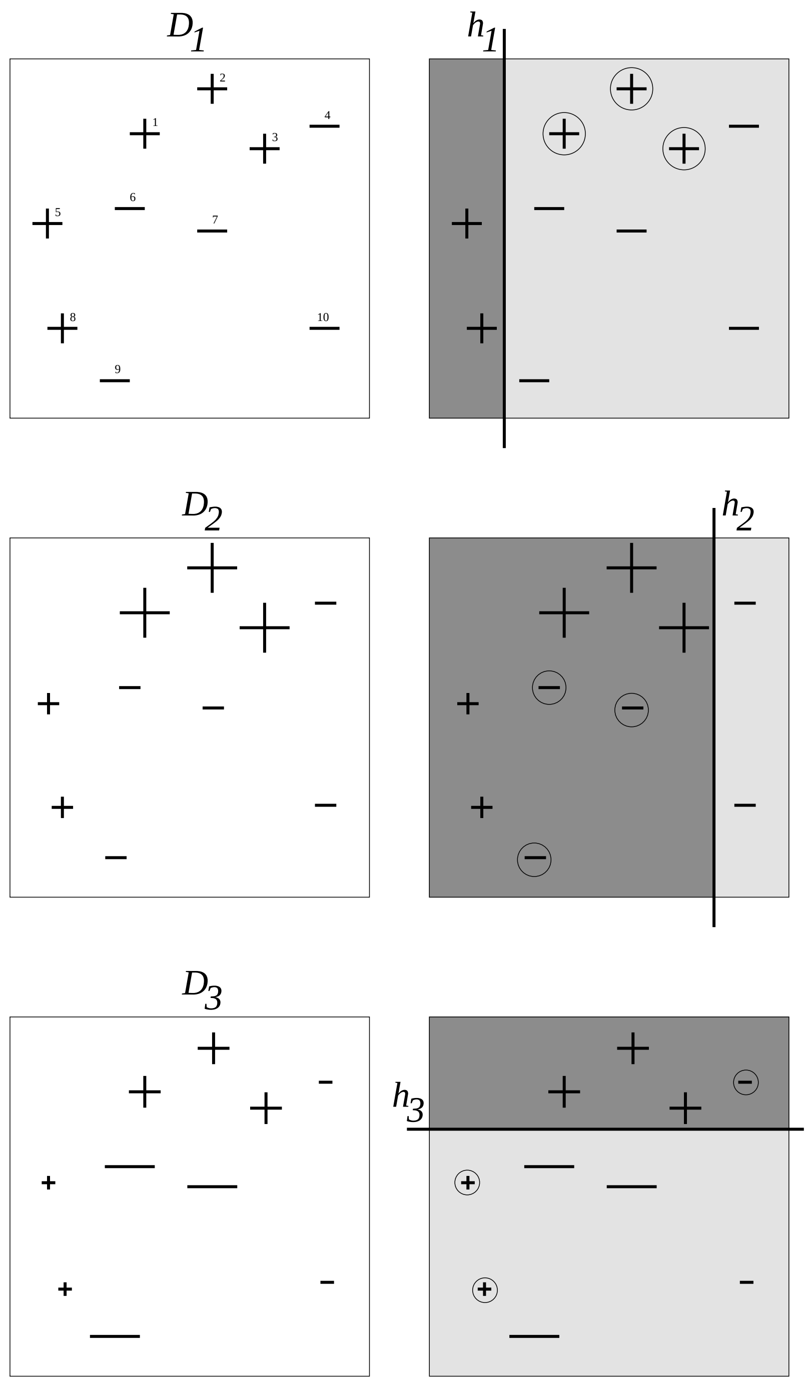


Figure 1.1: An illustration of how AdaBoost behaves on a tiny toy problem with $m = 10$ examples. Each row depicts one round, for $t = 1, 2, 3$. The left box in each row represents the distribution D_t, with the size of each example scaled in proportion to its weight under that distribution. Each box on the right shows the weak hypothesis h_t where darker shading indicates the region of the domain predicted positive. Examples that are misclassified by h_t have been circled.

	1	2	3	4	5	6	7	8	9	10	
$D_1(i)$	$\underline{0.10}$	$\underline{0.10}$	$\underline{0.10}$	0.10	0.10	0.10	0.10	0.10	0.10	0.10	$\epsilon_1 = 0.30, \alpha_1 \approx 0.42$
$e^{-\alpha_1 y_i h_1(x_i)}$	1.53	1.53	1.53	0.65	0.65	0.65	0.65	0.65	0.65	0.65	
$D_1(i)\ e^{-\alpha_1 y_i h_1(x_i)}$	0.15	0.15	0.15	0.07	0.07	0.07	0.07	0.07	0.07	0.07	$Z_1 \approx 0.92$
$D_2(i)$	0.17	0.17	0.17	0.07	0.07	$\underline{0.07}$	$\underline{0.07}$	0.07	$\underline{0.07}$	0.07	$\epsilon_2 \approx 0.21, \alpha_2 \approx 0.65$
$e^{-\alpha_2 y_i h_2(x_i)}$	0.52	0.52	0.52	0.52	0.52	1.91	1.91	0.52	1.91	0.52	
$D_2(i)\ e^{-\alpha_2 y_i h_2(x_i)}$	0.09	0.09	0.09	0.04	0.04	0.14	0.14	0.04	0.14	0.04	$Z_2 \approx 0.82$
$D_3(i)$	0.11	0.11	0.11	$\underline{0.05}$	$\underline{0.05}$	0.17	0.17	$\underline{0.05}$	0.17	0.05	$\epsilon_3 \approx 0.14, \alpha_3 \approx 0.92$
$e^{-\alpha_3 y_i h_3(x_i)}$	0.40	0.40	0.40	2.52	2.52	0.40	0.40	2.52	0.40	0.40	
$D_3(i)\ e^{-\alpha_3 y_i h_3(x_i)}$	0.04	0.04	0.04	0.11	0.11	0.07	0.07	0.11	0.07	0.02	$Z_3 \approx 0.69$

Table 1.1: The numerical calculations corresponding to the toy example in Figure 1.1. Calculations are shown for the 10 examples as numbered in the figure. Examples on which hypothesis h_t makes a mistake are indicated by underlined figures in the rows marked D_t.

an unweighted training error below 30%. On each round t, we suppose that the base learner always finds the base hypothesis of this form that has *minimum* weighted error with respect to the distribution D_t (breaking ties arbitrarily). We will see in this example how, using such a base learner for finding such weak base classifiers, AdaBoost is able to construct a combined classifier that correctly classifies all of the training examples in only $T = 3$ boosting rounds.

On round 1, AdaBoost assigns equal weight to all of the examples, as is indicated in the figure by drawing all examples in the box marked D_1 to be of the same size. Given examples with these weights, the base learner chooses the base hypothesis indicated by h_1 in the figure which classifies points as positive if and only if they lie to the left of this line. This hypothesis incorrectly classifies three points, namely, the three circled positive points, so its error ϵ_1 is 0.30. Plugging into the formula of Algorithm 1.1 gives $\alpha_1 \approx 0.42$.

In constructing D_2, the weights of the three points misclassified by h_1 are increased while the weights of all other points are decreased. This is indicated by the sizes of the points in the box marked D_2. See also Table 1.1 which shows the numerical calculations involved in running AdaBoost on this toy example.

On round 2, the base learner chooses the line marked h_2. This base classifier correctly classifies the three points missed by h_1 of relatively high weight at the expense of missing three other relatively low weight points which were correctly classified by h_1. Under distribution D_2, these three points have weight only around 0.07, so the error of h_2 with respect to D_2 is $\epsilon_2 \approx 0.21$, giving $\alpha_2 \approx 0.65$. In constructing D_3, the weights of these three misclassified points are increased while the weights of the other points are decreased.

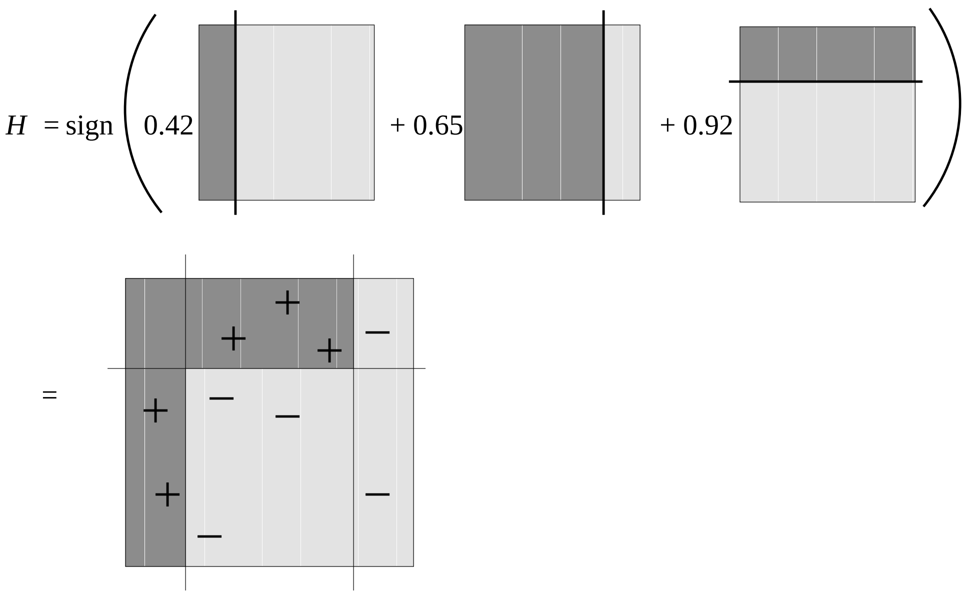


Figure 1.2: The combined classifier for the toy example of Figure 1.1 is computed as the sign of the weighted sum of the three weak hypotheses, $\alpha_1 h_1 + \alpha_2 h_2 + \alpha_3 h_3$, as shown at the top. This is equivalent to the classifier shown at the bottom. (As in Figure 1.1, the regions that a classifier predicts positive are indicated using darker shading.)

On round 3, classifier h_3 is chosen. Note that this classifier misses none of the points misclassified by h_1 and h_2 since these points have relatively high weight under D_3. Instead, it misclassifies three points which, because they were not misclassified by h_1 or h_2, are of very low weight under D_3. On round 3, $\epsilon_3 \approx 0.14$ and $\alpha_3 \approx 0.92$.

Note that our earlier remark that the error of each hypothesis h_t is exactly $1/2$ on the new distribution D_{t+1} can be verified numerically in this case from Table 1.1 (modulo small discrepancies due to rounding).

The combined classifier H is a weighted vote of h_1, h_2 and h_3 as shown in Figure 1.2, where the weights on the respective classifiers are α_1, α_2 and α_3. Although each of the composite weak classifiers misclassifies 3 of the 10 examples, the combined classifier, as shown in the figure, correctly classifies *all* of the training examples. For instance, the classification of the negative example in the upper right corner (instance #4), which is classified negative by h_1 and h_2 but positive by

h_3, is

$$\mathrm{sign}(-\alpha_1 - \alpha_2 + \alpha_3) = \mathrm{sign}(-0.15) = -1.$$

One might reasonably ask if such a rapid reduction in training error is typical for AdaBoost. The answer turns out to be "yes" in the following sense: given the weak learning assumption (that is, that the error of each weak classifier ϵ_t is at most $1/2 - \gamma$ for some $\gamma > 0$), we can prove that the training error of the combined classifier drops exponentially fast as a function of the number of weak classifiers combined. Although this fact, which is proved in Chapter 3, says nothing directly about generalization error, it does suggest that boosting, which is so effective at driving down the training error, may also be effective at producing a combined classifier with low generalization error. And indeed, in Chapters 4 and 5, we prove various theorems about the generalization error of AdaBoost's combined classifier.

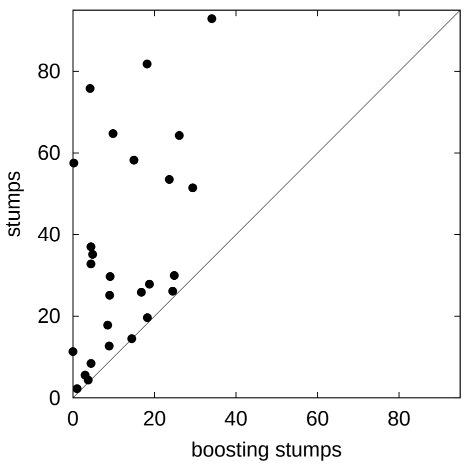
Note also that although we depend on the weak learning assumption to prove these results, AdaBoost does not need to know the "edge" γ referred to above, but rather adjusts and adapts to errors ϵ_t which may vary considerably, reflecting the varying levels of performance among the base classifiers. It is in this sense that AdaBoost is an *adaptive boosting* algorithm—which is exactly what the name stands for.[1] Moreover, this adaptiveness is one of the key qualities that makes AdaBoost practical.

1.2.2 Experimental performance

Experimentally, on data arising from many real-world applications, AdaBoost also turns out to be highly effective. To get a sense of AdaBoost's performance overall, we can compare it to other methods on a broad variety of publicly available benchmark datasets, an important methodology in machine learning since different algorithms can exhibit relative strengths that vary substantially from one dataset to the next. Here, we consider two base learning algorithms: one that produces quite weak and simple base classifiers called decision stumps; and the other, called C4.5, that is an established and already highly effective program for learning decision trees, which are generally more complex but also quite a bit more accurate than decision stumps. Both of these base classifiers are described further below in Sections 1.2.3 and 1.3.

Boosting algorithms work by improving the accuracy of the base learning algorithm. Figure 1.3 shows this effect on 27 benchmark datasets. In each scatterplot, each point shows the test error rate of boosting (x-coordinate) versus that of the

[1]This is also why *AdaBoost*, which is short for "adaptive boosting," is pronounced *ADD-uh-boost*, similar to *adaptation*.



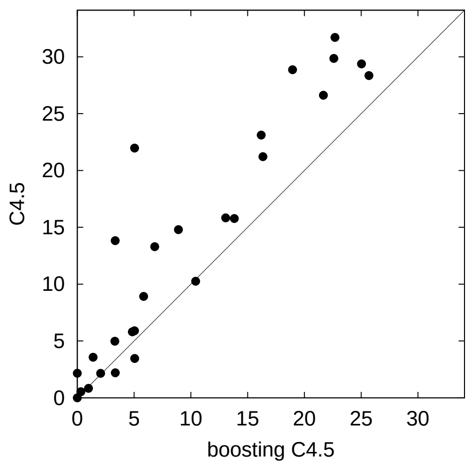


Figure 1.3: Comparison of two base learning algorithms—decision stumps and C4.5—with and without boosting. Each point in each scatterplot shows the test error rate of the two competing algorithms on one of 27 benchmark learning problems. The x-coordinate of each point gives the test error rate (in percent) using boosting, and the y-coordinate gives the error rate without boosting when using decision stumps (left plot) or C4.5 (right plot). All error rates have been averaged over multiple runs.

base learner (y-coordinate) on a single benchmark. All error rates have been averaged over multiple runs and multiple random splits of the given data into training and testing sets, and in these experiments, boosting was run for $T = 100$ rounds.

To "read" such a scatterplot, note that a point lands above the line $y = x$ if and only if boosting shows improvement over the base learner. Thus, we see that when using the relatively strong base learner C4.5, an algorithm that is very effective in its own right, AdaBoost is able to often provide quite a significant boost in performance. Even more dramatic is the improvement effected when using the rather weak decision stumps as base classifiers. In fact, this improvement is so substantial that boosting stumps is often even better than C4.5, as can be seen in Figure 1.4. On the other hand, overall, boosting C4.5 seems to give more accurate results than boosting stumps.

In short, empirically, AdaBoost appears to be highly effective as a learning tool for generalizing beyond the training set. How can we explain this capacity to extrapolate beyond the observed training data? Attempting to answer this question is a primary objective of this book.

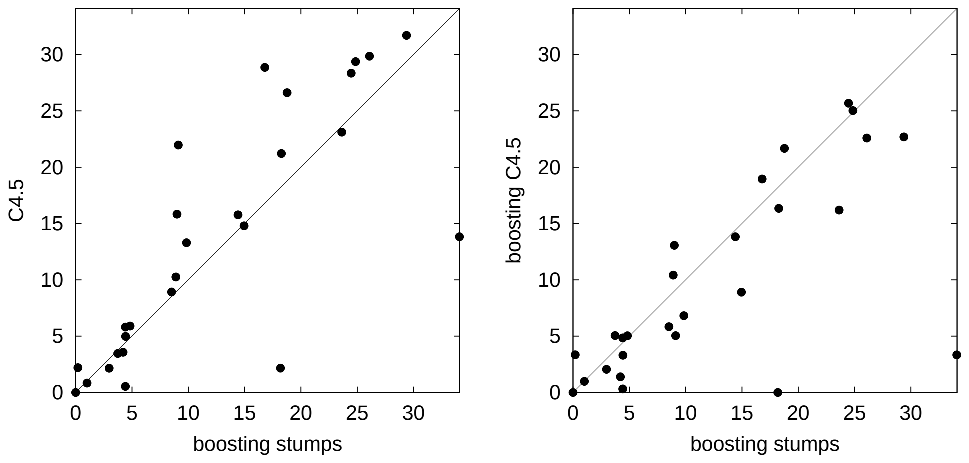


Figure 1.4: Comparison of boosting using decision stumps as the base learner versus unboosted C4.5 (left plot) and boosted C4.5 (right plot).

1.2.3 A medical diagnosis example

As a more detailed example, let us return to the heart-disease dataset described briefly in Section 1.1. If we wanted to apply boosting on this dataset, what would we use for the base learner and base classifiers? Here we have many options, but perhaps the simplest rules of thumb one can imagine are ones which test on a single attribute describing the patient. For instance, such a rule might state:

> If the patient's cholesterol is at least 228.5, then predict that the patient has heart disease; otherwise, predict the patient is healthy.

In the experiments we are about to describe, we used base classifiers of just this form which are the *decision stumps* alluded to in Section 1.2.2. (In fact, the weak classifiers used in the toy example of Section 1.2.1 are also decision stumps.) It turns out, as will be seen in Section 3.4.2, that a base learner which does an exhaustive search for the best decision stump can be implemented very efficiently (where, as before, "best" means the one with lowest weighted training error with respect to a given distribution D_t over training examples). Table 1.2 shows the first six base classifiers produced by this base learner when AdaBoost is applied to this entire dataset.

To measure performance on such a small dataset, we can divide the data randomly into disjoint training and test sets. Because the test set for such a split is very small, we repeat this many times using a standard technique called cross validation. We then take the averages of the training and test errors for the different

rnd	if	then predict	else predict
1	thalamus normal	healthy	sick
2	number of major vessels colored by flouroscopy > 0	sick	healthy
3	chest pain type is asymptomatic	sick	healthy
4	ST depression induced by exercise relative to rest ≥ 0.75	sick	healthy
5	cholesterol ≥ 228.5	sick	healthy
6	resting electrocardiographic results are normal	healthy	sick

Table 1.2: The first six base classifiers found when using AdaBoost on the heart-disease dataset.

splits of the data. Figure 1.5 shows these average error rates for this dataset as a function of the number of base classifiers combined. Boosting steadily drives down the training error. The test error also drops quickly reaching a low point of 15.3% after only 3 rounds, a rather significant improvement over using just one of the base classifiers, the best of which has a test error of 28.0%. However, after reaching this low point, the test error begins to *increase* again so that after 100 rounds, the test error is up to 18.8%, and after 1000 rounds, up to 22.0%.

This deterioration in performance with continued training is an example of an important and ubiquitous phenomenon called *overfitting*. As the number of base classifiers becomes larger and larger, the combined classifier becomes more and more complex leading somehow to a deterioration of test-error performance. Overfitting, which has been observed in many machine-learning settings and which has also received considerable theoretical study, is consistent with the intuition that a simpler explanation of the data is better than a more complicated one, a notion sometimes called "Occam's razor." With more rounds of boosting, the combined classifier grows in size and complexity apparently overwhelming the good performance on the training set. This general connection between simplicity and accuracy is explored in Chapter 2. For boosting, exactly the kind of behavior observed in Figure 1.5 is predicted by the analysis in Chapter 4.

Overfitting is a significant problem because it means that we have to be very careful about when to stop boosting. If we stop too soon or too late, our performance on the test set may suffer significantly, as can be seen in this example. Moreover, performance on the training set provides little clue about when to stop training since the training error typically continues to drop even as the overfitting gets worse and worse.

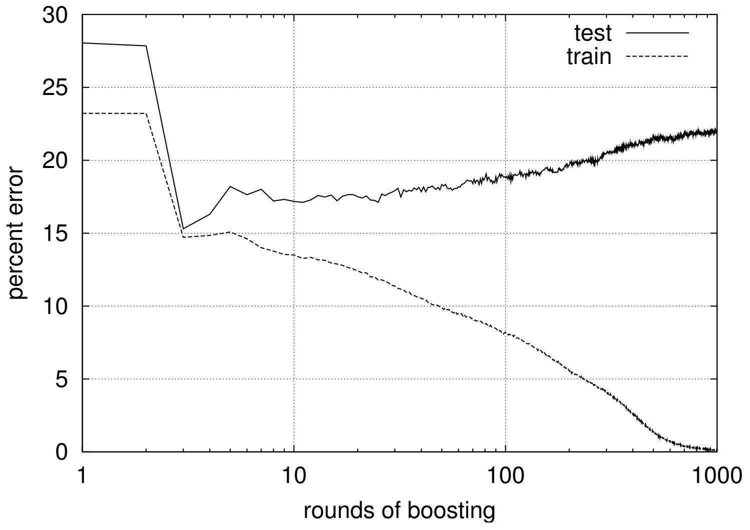


Figure 1.5: The training and test percent error rates obtained using boosting on the heart-disease dataset. Results are averaged over multiple train-test splits of the data.

1.3 Resistance to overfitting and the margins theory

This last example described a case in which boosting was used with very weak base classifiers. This is one possible use of boosting, namely, in conjunction with a very simple but truly mediocre weak learning algorithm. A rather different use of boosting is instead to boost the accuracy of a learning algorithm that is already quite good.

This is the approach taken in the next example. Here, rather than a very weak base learner, we used the well known and highly developed machine-learning algorithm C4.5 as the base learner. As mentioned earlier, C4.5 produces classifiers called *decision trees*. Figure 1.6 shows an example of a decision tree. The nodes are identified with tests with a small number of outcomes corresponding to the outgoing edges of the node. The leaves are identified with predicted labels. To classify an example, a path is traversed through the tree from the root to a leaf. The path is determined by the outcome of the tests that are encountered along the way, and the predicted classification is determined by the leaf that is ultimately reached. For instance, in the figure, a large, square, blue item would be classified '$-$' while a medium, round, red item would be classified '$+$'.

We tested boosting using C4.5 as the base learner on a benchmark dataset in

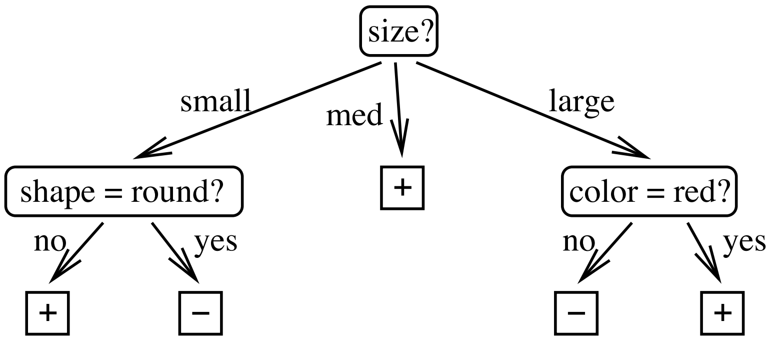


Figure 1.6: An example of a decision tree.

which the goal is to identify images of handwritten characters as letters of the alphabet. The features used are derived from the raw pixel images, including such items as the average of the x-coordinates of the pixels that are turned on. The dataset consists of 16,000 training examples and 4,000 test examples.

Figure 1.7 shows training and test error rates for AdaBoost's combined classifier on this dataset as a function of the number of decision trees (base classifiers) combined. A single decision tree produced by C4.5 on this dataset has a test error rate of 13.8%. In this example, boosting very quickly drives down the training error; in fact, after only five rounds, the training error is zero so that all training examples are correctly classified. Note that there is no reason why boosting cannot proceed beyond this point. Although the training error of the *combined* classifier is zero, the individual *base* classifiers continue to incur significant weighted error—around 5-6%—on the distributions on which they are trained so that ϵ_t remains in this range, even for large t. This permits AdaBoost to proceed with the reweighting of training examples and the continued training of base classifiers.

The test performance of boosting on this dataset is extremely good, far better than a single decision tree. And surprisingly, unlike the earlier example, on this dataset, the test error never increases, even after a thousand trees have been combined (by which point, the combined classifier involves more than two million decision nodes). Even after the training error hits zero, the test error continues to drop, from 8.4% on round 5 down to 3.1% on round 1000. This pronounced lack of overfitting seems to flatly contradict our earlier intuition that simpler is better. Surely, a combination of 5 trees is much, much simpler than a combination of 1000 trees (about 200 times simpler, in terms of raw size), and both perform equally well on the training set (perfectly, in fact). So how can it be that the far larger and more complex combined classifier performs so much better on the test set? This would

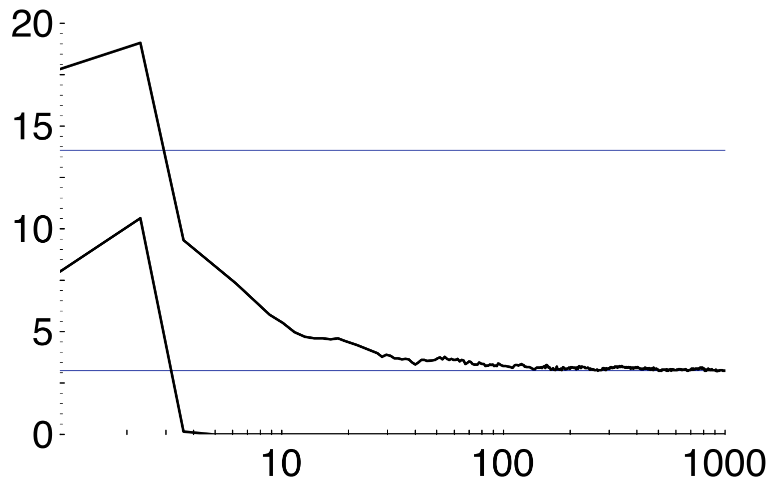


Figure 1.7: The training and test percent error rates obtained using boosting on an OCR dataset with C4.5 as the base learner. The top and bottom curves are test and training error, respectively. The top horizontal line shows the test error rate using just C4.5. The bottom line shows the final test error rate of AdaBoost after 1000 rounds. [Reprinted with permission of the Institute of Mathematical Statistics.]

appear to be a paradox.

One superficially plausible explanation is that the α_t's are converging rapidly to zero so that the number of base classifiers being combined is effectively bounded. However, as noted above, the ϵ_t's remain around 5-6% in this case, well below $1/2$, which means that the weights α_t on the individual base classifiers are also bounded well above zero so that the combined classifier is constantly growing and evolving with each round of boosting.

Such resistance to overfitting is typical of boosting, although, as we have seen in Section 1.2.3, boosting certainly *can* overfit. This resistance is one of the properties that make it such an attractive learning algorithm. But how can we understand this behavior?

In Chapter 5, we present a theoretical explanation of how, why and when AdaBoost works, and in particular, why it often does not overfit. Briefly, the main idea is the following. Our description above of AdaBoost's performance on the training set only took into account the training error, which is zero already after only five rounds. However, training error only tells part of the story in that it only reports the number of examples that are correctly or incorrectly classified. Instead, to understand AdaBoost, we also need to take into account how *confident* are the predictions being made by the algorithm. We will see that such confidence can be measured by a quantity called the *margin*. According to this explanation, although the training error—that is, whether or not the predictions are correct—is not changing after round 5, the confidence in those predictions is increasing dramatically with additional rounds of boosting. And this increase in the confidence in these predictions translates into better generalization performance.

This theory, for which we present both empirical and theoretical evidence, explains not only the lack of overfitting, but also provides a detailed framework for fundamentally understanding the conditions under which AdaBoost can fail or succeed.

1.4 Foundations and algorithms

The core analysis outlined above forms Part I of this book, a largely mathematical study of AdaBoost's capacity to minimize both the training and generalization error. Here, our focus is on understanding how, why and when AdaBoost is effective as a learning algorithm.

This analysis, including the margins theory, is paramount in our study of boosting; however, it is hardly the end of the story. Indeed, although an enticingly simple algorithm, AdaBoost turns out to be understandable from a striking number of disparate theoretical perspectives. Taken together, these provide a remarkably rich and encompassing illumination of the algorithm, in addition to practical generalizations and variations along multiple dimensions. Part II of the book explores three of these fundamental perspectives.

In the first of these, the interaction between a boosting algorithm and a weak learning algorithm is viewed as a game between these two players—not only a game in the informal, everyday sense, but also in the mathematical sense studied in the field of game theory. In fact, it turns out that AdaBoost is a special case of a more general algorithm for playing any game in a repeated fashion. This perspective, presented in Chapter 6, helps us to understand numerous properties of the algorithm, such as its limiting behavior, in broader, game-theoretic terms. We will see that notions that are central to boosting, such as margins and the weak learning assumption, have very natural game-theoretic interpretations. Indeed, the very idea of boosting turns out to be intimately entwined with one of the most fundamental theorems of game theory. This view also unifies AdaBoost with another branch of learning known as on-line learning.

AdaBoost can be further understood as an algorithm for optimizing a particular objective function measuring the fit of a model to the available data. In this way, AdaBoost can be seen as an instance of a more general approach that can be applied to a broader range of statistical learning problems, as we describe in Chapter 7. This view further leads to a unification of AdaBoost with the more established statistical method called logistic regression, and suggests how AdaBoost's predictions can be used to estimate the probability of a particular example being positive or negative.

From yet another vantage point, which turns out to be "dual" to the one given in Chapter 7, AdaBoost can be interpreted in a kind of abstract, geometric framework. Here, the fundamental operation is projection of a point onto a subspace. In this case, the "points" are in fact the distributions D_t computed by AdaBoost, which exist in a kind of "information geometric" space—one based on notions from information theory—rather than the usual Euclidean geometry. As discussed in Chapter 8, this view leads to a deeper understanding of AdaBoost's dynamics and underlying mathematical structure, and yields proofs of fundamental convergence properties.

Part III of this book focuses on practical, algorithmic extensions of AdaBoost. In the basic form shown in Algorithm 1.1, AdaBoost is intended for the simplest learning setting in which the goal is binary classification, that is, classification problems with only two possible classes or categories. To apply AdaBoost to a much broader range of real-world learning problems, the algorithm must be extended along multiple dimensions.

In Chapter 9, we describe an extension to AdaBoost in which the base classifiers themselves are permitted to output predictions that vary in their self-rated level of confidence. In practical terms, this modification of boosting leads to a dramatic speed-up in learning time. Moreover, within this framework, we derive two algorithms designed to produce classifiers that are not only accurate, but also understandable in form to humans.

Chapter 10 extends AdaBoost to the case in which there are more than two possible classes, as is very commonly the case in actual applications. For instance, if recognizing digits, there are ten classes, one for each digit. As will be seen, it turns out that there are quite a number of methods for modifying AdaBoost for this purpose, and we will see how a great many of these can be studied in a unified framework.

Chapter 11 extends AdaBoost for ranking problems, that is, problems in which the goal is to learn to rank a set of objects. For instance, the goal might be to rank credit card transactions according to the likelihood of each one being fraudulent so that those at the top of the ranking can be investigated.

Finally, in Part IV, we study a number of advanced theoretical topics.

The first of these provides an alternative approach for the understanding of AdaBoost's generalization capabilities, and one that explicitly takes into consideration intrinsic randomness or "noise" in the data that may prevent perfect generalization by *any* classifier. In such a setting, we show in Chapter 12 that the accuracy of (a variant of) AdaBoost will nevertheless converge to that of the best possible classifier, under appropriate assumptions. However, we also show that without these assumptions, AdaBoost's performance can be rather poor when the data is noisy.

AdaBoost can be understood in many ways, but at its foundation, it is a boosting algorithm in the original technical meaning of the word, that is, a provable method for driving down the error of the combined classifier by combining a number of weak classifiers. In fact, for this specific problem, AdaBoost is not the best possible; rather, there is another algorithm called "boost-by-majority" that is optimal in a very strong sense, as we will see in Chapter 13. However, this latter algorithm is not practical because it is not adaptive in the sense described in Section 1.2.1. Nevertheless, as we show in Chapter 14, this algorithm can be made adaptive by taking a kind of limit in which the discrete time steps in the usual boosting framework are replaced by a *continuous* sequence of time steps. This leads to a number of new boosting algorithms with favorable properties, especially tolerance to noise in the data, and connects further to AdaBoost which can be derived in this setting by taking a particular limit.

Although this book is about foundations and algorithms, we also provide numerous examples illustrating how the theory we develop can be applied practically. Indeed, as seen earlier in this chapter, AdaBoost has many practical advantages. It is fast, simple and easy to program. It has no parameters to tune (except for the number of rounds T). It requires no prior knowledge about the base learner and so can be flexibly combined with any method for finding base classifiers. Finally, it comes with a set of theoretical guarantees given sufficient data and a base learner that can reliably provide only moderately accurate base classifiers. This is a shift in mind set for the learning-system designer: instead of trying to design a learning algorithm that is accurate over the entire space, we can instead focus on finding weak learning algorithms that only need to be better than random.

On the other hand, some caveats are certainly in order. The actual performance of boosting on a particular problem is clearly dependent on the data and the base learner. Consistent with the theory outlined above and discussed in detail in this book, boosting can fail to perform well given insufficient data, overly complex base classifiers or base classifiers that are too weak. Boosting seems to be especially susceptible to noise as we discuss in Section 12.3. Nonetheless, as seen in Section 1.2.2, on a wide range of real-world learning problems, boosting's performance overall is quite good.

To illustrate its empirical performance and application, throughout this book, we give examples of its use on practical problems such as human-face detection, topic identification, language understanding in spoken-dialogue systems, and natural-language parsing.

Summary

In this chapter, we have given an introduction to machine learning, classification problems and boosting, particularly AdaBoost and its variants, which are the focus of this book. We have given examples of boosting's empirical performance, as well as an overview of some of the highlights of its rich and varied theory. In the chapters ahead, we explore the foundations of boosting from many vantage points, and develop key principles in the design of boosting algorithms, while also giving examples of their application to practical problems.

Bibliographic notes

Boosting has its roots in a theoretical framework for studying machine learning called the PAC model, proposed by Valiant [220], which we discuss in more detail in Section 2.3. Working in this framework, Kearns and Valiant [133] proposed the question of whether a weak learning algorithm that performs just slightly better than random guessing can be boosted into one with arbitrarily high accuracy. Schapire [198] came up with the first provable polynomial-time boosting algorithm in 1989. A year later, Freund [88] developed a much more efficient boosting algorithm called boost-by-majority that is essentially optimal (see Chapter 13). The first experiments with these early boosting algorithms were carried out by Drucker, Schapire and Simard [72] on an OCR task. However, both algorithms were largely impractical because of their non-adaptiveness. AdaBoost, the first adaptive boosting algorithm, was introduced in 1995 by Freund and Schapire [95].

There are many fine textbooks which provide a broader treatment of machine learning, a field that overlaps considerably with statistics, pattern recognition and data mining. See, for instance, refs. [7, 22, 67, 73, 120, 134, 166, 171, 224, 225]. For alternative surveys of boosting and related methods for combining classifiers, see refs. [40, 69, 146, 170, 213].

The medical-diagnosis data used in Section 1.2.3 was collected from the Cleveland Clinic Foundation by Detrano et al. [66]. The letter recognition dataset used in Section 1.3 was created by Frey and Slate [97]. The C4.5 decision-tree learning algorithm used in Sections 1.2.2 and 1.3 is due to Quinlan [183], and is similar to the CART algorithm of Breiman et al. [39].

Drucker and Cortes [71], and Jackson and Craven [126] were the first to test AdaBoost experimentally. The experiments in Section 1.2.2 were originally reported by Freund and Schapire [93] from which the right plot of Figure 1.3 and left plot of Figure 1.4 were adapted. AdaBoost's resistance to overfitting was noticed early on by Drucker and Cortes [71], as well as Breiman [35] and Quinlan [182]. The experiments in Section 1.3, including Figure 1.7, are taken from Schapire et al. [201]. There have been numerous other systematic experimental studies of AdaBoost, such as refs. [15, 68, 162, 208], as well as Caruana and Niculescu-Mizil [42]'s large-scale comparison of several learning algorithms, including AdaBoost.

Exercises

1-1. Show that the error of h_t on distribution D_{t+1} is exactly $1/2$, that is,

$$\Pr_{i \sim D_{t+1}} [h_t(x_i) \neq y_i] = \tfrac{1}{2}.$$

1-2. For each of the following cases, explain how AdaBoost, as given in Algorithm 1.1, will treat a weak hypothesis h_t with weighted error ϵ_t. Also, in each case, explain how this behavior makes sense.

(a) $\epsilon_t = 1/2$.

(b) $\epsilon_t > 1/2$.

(c) $\epsilon_t = 0$.

1-3. In Figure 1.7, the training error and test error of the combined classifier H are seen to increase significantly on the second round. Give a plausible explanation why we might expect these error rates after two rounds to be higher than after only one.

Part I

Core Analysis

Chapter 2

Foundations of Machine Learning

Soon, we will embark on a theoretical study of AdaBoost in order to understand its properties, particularly its ability as a learning algorithm to generalize, that is, to make accurate predictions on data not seen during training. Before this will be possible, however, it will be necessary to take a step back to outline our approach to the more general problem of machine learning, including some fundamental general-purpose tools that will be invaluable in our analysis of AdaBoost.

We study the basic problem of inferring from a set of training examples a classification rule whose predictions are highly accurate on freshly observed test data. On first encounter, it may seem questionable whether this kind of learning should even be possible. After all, why should there be any connection between the training and test examples, and why should it be possible to generalize from a relatively small number of training examples to a potentially vast universe of test examples? Although such objections have indeed often been the subject of philosophical debate, in this chapter, we will identify an idealized but realistic model of the inference problem in which this kind of learning can be proved to be entirely feasible when certain conditions are satisfied. In particular, we will see that if we can find a *simple* rule that fits the training data well, and if the training set is not too small, then this rule will in fact generalize well, providing accurate predictions on previously unseen test examples. This is the basis of the approach presented in this chapter, and we will often use the general analysis on which it is founded to guide us in understanding how, why and when learning is possible.

We also outline in this chapter a mathematical framework for studying machine learning, one in which a precise formulation of the boosting problem can be clearly and naturally expressed.

instances:	1.2	2.8	8.0	3.3	5.0	4.5	7.4	5.6	3.8	6.6	6.1	1.7
labels:	$-$	$-$	$+$	$-$	$-$	$-$	$+$	$+$	$-$	$+$	$+$	$-$

Table 2.1: A sample dataset. Each column is a labeled example. For instance, the third example is the instance $x = 8.0$ with corresponding label $y = +1$.

Note that, unlike the rest of the book, this chapter omits nearly all of the proofs of the main results since these have largely all appeared in various texts and articles. See the bibliographic notes at the end of the chapter for references.

Also, Appendix A describes some of the notation and definitions used throughout this book, and briefly provides mathematical background in some of the relevant areas.

2.1 A direct approach to machine learning

We start with an introduction to our approach to machine learning. This will lead to the identification of criteria that are sufficient to assure generalization, laying the intuitive groundwork for the formal treatment that follows.

2.1.1 Conditions sufficient for learning

As described in Chapter 1, a learning algorithm takes as input a labeled set of training examples $(x_1, y_1), \ldots, (x_m, y_m)$ and must use these to formulate a hypothesis for classifying new instances. As before, and for most of the book, we assume that there are only two possible labels so that each $y_i \in \{-1, +1\}$.

Let us begin with an example in which the instances x_i are simply real numbers, and in which the training set looks something like the one in Table 2.1. Given such a dataset, how might we formulate a prediction rule? After examining these examples, most of us would eventually notice that the larger examples, above some cut-off, are positive, and the smaller examples are negative. This would lead most people to choose a rule based on the observed cut-off behavior exhibited by this data; in other words, they would eventually choose a threshold rule that predicts all instances x above some threshold ν are $+1$, and all instances below ν are -1, that is,

$$h(x) = \begin{cases} +1 & \text{if } x \geq \nu \\ -1 & \text{otherwise} \end{cases} \tag{2.1}$$

for some choice of ν, such as 5.3, or any other value between 5.0 (the largest negative example) and 5.6 (the smallest positive example). Such a rule—which

instances:	1.2	2.8	8.0	3.3	5.0	4.5	7.4	5.6	3.8	6.6	6.1	1.7
labels:	$-$	$-$	$+$	$-$	$-$	$-$	$+$	$+$	$+$	$-$	$+$	$-$

Table 2.2: A slightly different sample dataset.

is essentially the same in form as the decision stumps used in Section 1.2.3—seems so obvious and irresistibly attractive because it has the two properties that people seem to instinctively prefer: First of all, it is *consistent* with the given data, meaning that it predicts the correct labels on all of the given training examples. And secondly, the rule is *simple*.

This preference for simple explanations, as noted in Chapter 1, is often referred to as *Occam's Razor*, a central tenet, for instance, of mathematics and most areas of scientific inquiry. However, unlike consistency, simplicity is not at all simple to define, and seems vague, even mystical, although most of us feel that we recognize it when we see it. The notion of simplicity is closely related to our prior expectations: we expect the data to be explainable by a simple rule, and conversely, we usually consider a rule to be simple if it matches our expectations. One of the triumphs of modern research on learning has been the development of quantitative measures of simplicity and its role in generalization, as we will see shortly.

When faced with a harder dataset, such as the slightly modified version in Table 2.2, it is not so immediate how to find a simple and consistent rule. On the one hand, we might choose a simple threshold rule as before, but since none are consistent with this dataset, we will inevitably be forced to accept a small number of mistakes on the training set itself. Alternatively, we might choose a considerably more complex rule that *is* consistent, such as this one:

$$h(x) = \begin{cases} -1 & \text{if } x < 3.4 \\ +1 & \text{if } 3.4 \leq x < 4.0 \\ -1 & \text{if } 4.0 \leq x < 5.2 \\ +1 & \text{if } 5.2 \leq x < 6.3 \\ -1 & \text{if } 6.3 \leq x < 7.1 \\ +1 & \text{if } x \geq 7.1. \end{cases} \tag{2.2}$$

Thus, we face a trade-off between simplicity and fit to the data, one that will require balance and compromise.

Generally, then, we see that a natural approach to learning is to seek a hypothesis satisfying two basic criteria:

1. that it fits the data well, and
2. that it is simple.

As indicated in the example above, however, these two criteria are often in conflict: we can typically fit the data better at the cost of choosing a more complex hypothesis, and conversely, simpler hypotheses are prone to give a poorer fit to the data. This trade-off is quite general, and is at the heart of the most central issue in the study of machine learning.

This issue must be faced at some point in the design of every learning algorithm, since we must eventually make an explicit or implicit decision about the form of the hypotheses that will be used. The form that is chosen reflects an assumption about what we expect in the data, and it is the form that determines simplicity or complexity. For instance, we might choose to use threshold rules of the form given in Eq. (2.1), or we might instead use rules of a much freer form as in Eq. (2.2). In principle, the more we know about a particular learning problem, the simpler (and more restrictive) the form of the hypotheses that can be used.

We measure how well a particular hypothesis h fits the training data by its *training* (or *empirical*) *error*, that is, the fraction of the m training examples that it misclassifies, denoted

$$\widehat{\mathrm{err}}(h) \doteq \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}\{h(x_i) \neq y_i\}$$

where $\mathbf{1}\{\cdot\}$ is an indicator function that is 1 if its argument is true, and 0 otherwise. Although we seek a simple classifier h with low training error, the ultimate goal of learning is to find a rule that is highly accurate as measured on a separate test set. We generally assume that both the training examples and test examples are generated from the same distribution $\mathcal{D}$ on labeled pairs (x, y). With respect to this true distribution, the expected test error of a hypothesis h is called the *true* or *generalization error*; it is the same as the probability of misclassifying a single example (x, y) chosen at random from $\mathcal{D}$, and is denoted

$$\mathrm{err}(h) \doteq \Pr_{(x,y)\sim\mathcal{D}}\left[h(x) \neq y\right]. \tag{2.3}$$

Of course, a learning algorithm does not have the means to directly measure the generalization error $\mathrm{err}(h)$ that it aims to minimize. Instead, it must use the training error $\widehat{\mathrm{err}}(h)$ as an estimate or proxy for the generalization error. If working with a single hypothesis h, then the training error will be a reasonable estimate of the generalization error which it equals in expectation—in this sense, it is an *unbiased* estimator. However, generally, a learning algorithm will work with a large space of hypotheses, and will choose the hypothesis from this space with (approximately) minimum training error. Unfortunately, the training error of a hypothesis selected in this fashion will not be unbiased, but rather will almost certainly underestimate its true error. This is because, in selecting the hypothesis with minimum training error, the algorithm favors hypotheses whose training errors are, by chance, much lower than their true errors.

To get an intuition for this effect, imagine an experiment in which each student in a class is asked to predict the outcomes of ten coin flips. Clearly, any individual student will, in expectation, correctly predict just half of the flips. However, in a good-size class of perhaps fifty students, it is highly likely that one student will be very lucky and will correctly predict eight or even nine of the coin flips. This student will appear to possess remarkable powers of clairvoyance, when in fact, it was only chance that made the student's true prediction accuracy of 50% appear to be much higher. The larger the class, the greater will be this effect: at an extreme, for a very large class of over a thousand students, we will expect there to be one student that perfectly predicts all ten coin flips.

In the learning setting, for the same reasons, it is quite likely that the apparently good performance of the best hypothesis on the training set will in part be the result of having fit spurious patterns that appeared in the training data entirely by chance. Inevitably, this will cause the training error to be lower than the true error for the chosen hypothesis. Moreover, the amount of this bias depends directly on the simplicity or complexity of the hypotheses considered—the more complex and thus less restrictive the form of the hypotheses, the larger the space from which they are selected, and the greater will be the bias, just as in the example above. On the other hand, we will see that the bias can be controlled when a sufficiently large training set is available.

Finding the hypothesis with minimal or nearly minimal training error can often be accomplished through an exhaustive or heuristic search, even when working with an infinitely large space of hypotheses. As an example, consider our sample dataset in Table 2.2 when using threshold rules as in Eq. (2.1). Figure 2.1 shows a plot of the training error $\widehat{\mathrm{err}}(h)$ of such rules as a function of the threshold ν. As can be seen from the figure, even though the set of potential classifiers of this form is uncountably infinite, a training set of m examples partitions the classifiers into $m + 1$ subsets such that all of the rules in any single subset make the same predictions on the training examples, and thus have the same empirical error. (For instance, the threshold rule defined by setting the threshold ν to be 3.9 makes exactly the same predictions on all m of the examples as if we were instead to set ν to be 4.2.) As a result, we can find a classifier that minimizes the training error by first sorting the data according to the instance values x_i, and then computing the training error for all possible values of the threshold ν in a single scan through the sorted examples. In this case, we would find that the minimum training error is attained when ν is between 5.0 and 5.6. (More details of this method will be given in Section 3.4.2.)

In general, the idea of balancing simplicity against fit to the training data is

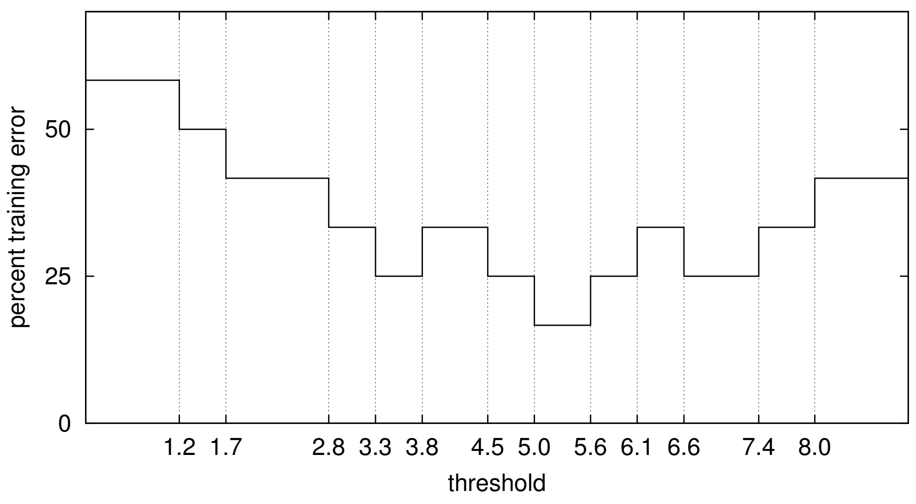


Figure 2.1: The empirical error, on the data in Table 2.2, of a threshold rule of the form given in Eq. (2.1), plotted as a function of the threshold ν.

key to many practical learning algorithms. For instance, decision-tree algorithms (see Section 1.3) typically grow a large tree that fits the data very well (usually too well), and then heavily prune the tree so as to limit its overall complexity as measured by its gross size.

2.1.2 Comparison to an alternative approach

As a brief aside, we take a moment to contrast the approach we are following to a well-studied alternative. When faced with data as in Tables 2.1 and 2.2, we may in some cases have additional information available to us. For instance, maybe we know that each example i corresponds to a person where x_i represents the person's height and y_i the person's gender (say $+1$ means male and -1 means female); in other words, the problem is to learn to predict a person's gender from his or her height. This information leads to some natural assumptions. Specifically, we might assume that height among men is normally distributed, and likewise among women, where naturally the means μ_+ and μ_- will be different with $\mu_+ > \mu_-$.

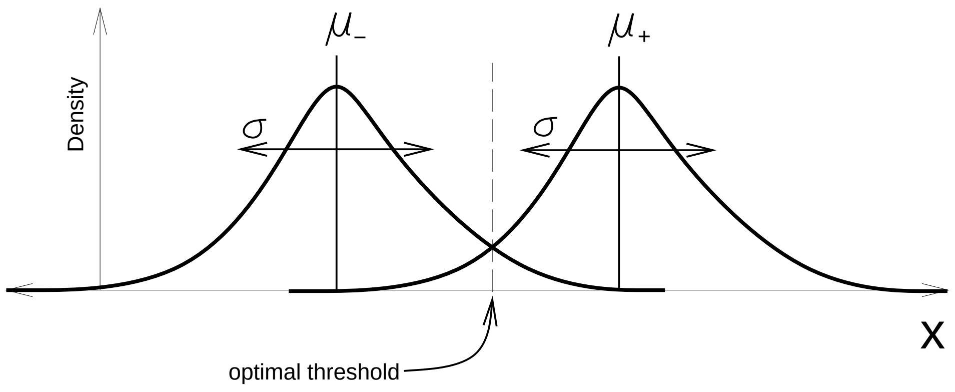


Figure 2.2: A classification problem in which both classes are normally distributed with equal variances but different means.

We might further assume equal variance σ^2 for these two normal distributions, and that the two classes (genders) occur with equal probability.

These assumptions suggest a different way of proceeding. Our goal is still to classify instances whose label is unknown, but now, if we know the values of the means μ_+ and μ_- then we can easily calculate the best possible classifier, which in this case simply predicts, for a given instance (height) x, with the mean that is closest; in other words, $+1$ if $x \geq (\mu_+ + \mu_-)/2$, and -1 otherwise. See Figure 2.2. Note that this classifier has the form of a threshold rule as in Eq. (2.1), although our rationale for using such a rule here is rather different. If the distribution parameters μ_+ and μ_- are unknown, they can be estimated from training data, and then used to classify test examples in the same way.

This is sometimes called a *generative* approach since we attempt to model how the data is being generated. In contrast, the approach we outlined in Section 2.1.1 is often said to be *discriminative* since the emphasis is on directly discriminating between the two classes.

When the assumptions we have made above about the data are valid, good generalization is assured in this simple case. Most importantly, such a guarantee of good performance is dependent on the data actually being generated by two normal distributions. If this assumption does not hold, then the performance of the resulting classifier can become arbitrarily poor as can be seen in Figure 2.3 where, because the two distributions are far from normal, the threshold between positive and negatives that is found by incorrectly assuming normality ends up being well away from optimal.

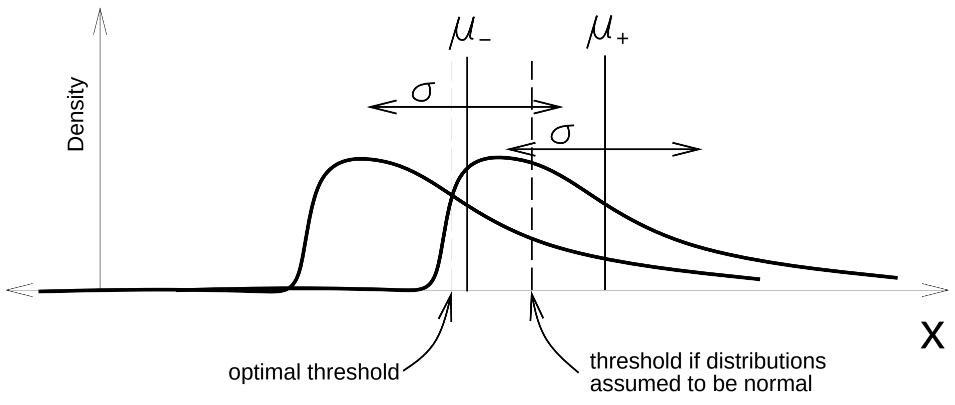


Figure 2.3: A classification problem in which both classes have the same skewed, non-normal distribution but with different means.

Normality assumptions lead naturally to the use of threshold rules as in Eq. (2.1), but we see in this example that an over-dependence on this assumption can yield poor performance. On the other hand, a discriminative approach to this problem in which the best threshold rule is selected based on its training error would be very likely to perform well in this case, since, after all, the optimal classifier is a simple threshold here as well. So because the discriminative approach is not based on distributional assumptions, it also can be more robust.

Moreover, if our only goal is to produce a good classifier, then estimating the distribution parameters of each class is entirely irrelevant to the task at hand. We do not care what the means of the distributions are, or even whether or not the distributions are normal. The only thing we care about in classification is which label y is more likely to correspond with any given instance x.

The generative approach can provide a powerful and convenient framework for incorporating into the learning process information that may be available about how the data is generated. However, its performance may be sensitive to the validity of those assumptions. The discriminative approach, on the other hand, attempts more directly to find a good hypothesis by searching for a simple one that makes accurate predictions on the training data without regard to underlying distributional assumptions. In this latter approach, preconceptions about the data are used to guide how we choose the form of the rules considered, but actual performance may be relatively robust.

2.2 General methods of analysis

We return now to the development of general methods for analyzing the generalization error of classifiers generated by learning algorithms. This is the essence of proving an algorithm's effectiveness. As we have discussed, success in learning depends intuitively on finding a classifier (1) that fits the training data well, that is, has low training error; and (2) that is simple. A third prerequisite, of course, is that the learner be provided with a sufficiently large training set. We will see that the generalization error depends on these same three interacting factors in a way that can be formalized in precise terms. On the other hand, the analysis we present does not in any way depend on the form of the distribution of data; for instance, we make no assumptions of normality. This reflects the general robustness of this approach which is largely immune to changes in the underlying distribution.

We begin by analyzing the generalization error of a single hypothesis and then move on to analyze whole families of hypotheses. Along the way, we will develop several different techniques for measuring the central notion of simplicity and complexity.

2.2.1 A single hypothesis

To start, consider just a single, fixed hypothesis h. Earlier, we discussed how the training error is used as a proxy for the true error. This motivates us to ask how much the training error $\widehat{\text{err}}(h)$ can differ from the true error $\text{err}(h)$ as a function of the number of training examples m. Note first that there is always a chance that the selected training set will be highly unrepresentative so that the training error will be a very poor estimate of the true error. This means that it is impossible to give a guarantee that holds with absolute certainty. Instead, we seek bounds that hold true *with high probability* over the choice of the random training set.

In fact, the problem can be seen to be equivalent to one involving coin flipping. When a training example (x_i, y_i) is selected at random, the probability that $h(x_i) \neq y_i$ is exactly $p = \text{err}(h)$, an event that we can identify with a flipped, biased coin coming up *heads*. In this way, the training set can be viewed as a sequence of m coin flips, each of which is *heads* with probability p. The problem then is to determine the probability that the fraction $\hat{p}$ of *heads* in the actual observed sequence of coin flips—that is, the training error—will be significantly different from p. We can write down explicitly the probability, say, of getting at most $(p - \varepsilon)m$ *heads*, which is exactly

$$\sum_{i=0}^{\lfloor (p-\varepsilon)m \rfloor} \binom{m}{i} p^i (1-p)^{m-i}. \tag{2.4}$$

This is a rather unwieldy expression, but fortunately, there exist a number of tools for bounding it. Foremost among these is the family of *Chernoff bounds*, including *Hoeffding's inequality*, one of the simplest and most widely used, which can be stated as follows:

Theorem 2.1 *Let $X_1, \ldots, X_m$ be independent random variables such that $X_i \in [0,1]$. Denote their average value by $A_m = \frac{1}{m}\sum_{i=1}^m X_i$. Then for any $\varepsilon > 0$ we have*

$$\Pr\left[A_m \geq \mathrm{E}\left[A_m\right] + \varepsilon\right] \leq e^{-2m\varepsilon^2} \tag{2.5}$$

and

$$\Pr\left[A_m \leq \mathrm{E}\left[A_m\right] - \varepsilon\right] \leq e^{-2m\varepsilon^2}. \tag{2.6}$$

Hoeffding's inequality applies to averages of arbitrary bounded and independent random variables. In the coin flipping example, we can take $X_i = 1$ (*heads*) with probability p and $X_i = 0$ (*tails*) with probability $1 - p$. Then A_m is equal to $\hat{p}$, the fraction of heads observed in a sequence of m flips; its expected value $\mathrm{E}\left[A_m\right]$ is p; and Eq. (2.6) tells us that the chance of at most $(p - \varepsilon)m$ *heads*, written explicitly in Eq. (2.4), is at most $e^{-2m\varepsilon^2}$.

In the learning setting, we can define the random variable X_i to be 1 if $h(x_i) \neq y_i$ and 0 otherwise. This means that the average value A_m is exactly $\widehat{\mathrm{err}}(h)$, the training error of h, and that $\mathrm{E}\left[A_m\right]$ is exactly the generalization error $\mathrm{err}(h)$. Thus, Theorem 2.1 (using Eq. (2.6)) implies that the probability of a training sample of size m for which

$$\mathrm{err}(h) \geq \widehat{\mathrm{err}}(h) + \varepsilon$$

is at most $e^{-2m\varepsilon^2}$. Said differently, given m random examples, we can deduce that with probability at least $1 - \delta$, for any $\delta > 0$, that the following upper bound holds on the generalization error of h:

$$\mathrm{err}(h) \leq \widehat{\mathrm{err}}(h) + \sqrt{\frac{\ln(1/\delta)}{2m}}.$$

(Here, we simply set $\delta = e^{-2m\varepsilon^2}$ and solve for ε.) This means, in quantifiable terms, that if h has low training error on a good-sized training set, then we can be quite confident that h's true error is also low.

Using Eq. (2.5) gives a corresponding lower bound on $\mathrm{err}(h)$. We can combine these using the *union bound*, which states simply that

$$\Pr\left[a \vee b\right] \leq \Pr\left[a\right] + \Pr\left[b\right]$$

for any two events a and b. Together, the two bounds imply that the chance that

$$\left|\mathrm{err}(h) - \widehat{\mathrm{err}}(h)\right| \geq \varepsilon$$

is at most $2e^{-2m\varepsilon^2}$, or that

$$|\mathrm{err}(h) - \widehat{\mathrm{err}}(h)| \le \sqrt{\frac{\ln(2/\delta)}{2m}}$$

with probability at least $1 - \delta$.

As with most of the results in this chapter, we do not prove Theorem 2.1. However, we note as an aside that the standard technique for proving Chernoff bounds is closely related to the method we use in Chapter 3 to analyze AdaBoost's training error. Indeed, as will be seen in Section 3.3, a special case of Hoeffding's inequality follows as a direct corollary of our analysis of AdaBoost.

2.2.2 Finite hypothesis spaces

Thus, we can bound the difference between the training error and the true error of a *single*, fixed classifier. To apply this to a learning algorithm, it might seem tempting to use the same argument to estimate the error of the single hypothesis that is chosen by the algorithm by minimizing the training error—after all, this is still only one hypothesis that we care about. However, such reasoning would be entirely fallacious. Informally, the problem is that in such an argument, the training errors are used twice—first to choose the seemingly best hypothesis, and then to estimate its true error. Said differently, the reasoning in Section 2.2.1 requires that we select the single hypothesis h *before* the training set is randomly chosen. The argument is invalid if h is itself a random variable that depends on the training set as it would be if selected to minimize the training error. Moreover, we have already argued informally in Section 2.1.1 that the hypothesis that appears best on the training set is very likely to have a true error that is significantly higher, whereas, for a single hypothesis, the training error is an unbiased estimate of the true error; this is another indication of the fallacy of such an argument.

Despite these difficulties, we will see now how we can analyze the error of the classifier produced by a learning algorithm, even if it is found by minimizing the training error. The intuitive arguments outlined in Section 2.1.1 indicate that such a bound will depend on the form of the hypotheses being used, since this form defines how simple or complex they are. Saying that a hypothesis has a particular form is abstractly equivalent to saying that it belongs to some set of hypotheses $\mathcal{H}$, since we can define $\mathcal{H}$ tautologically to be the set of all hypotheses of the chosen form. For instance, $\mathcal{H}$ might be the set of all threshold rules as in Eq. (2.1). We call $\mathcal{H}$ the *hypothesis class* or *hypothesis space*.

Generally, our approach will be to show that the training error of *every* hypothesis $h \in \mathcal{H}$ is close to its true error with high probability, leading to so-called *uniform error* or *uniform convergence* bounds. This condition will ensure that the hypothesis with minimum training error also has nearly minimal true error among all hypotheses in the class. For if

$$|\text{err}(h) - \widehat{\text{err}}(h)| \leq \varepsilon \tag{2.7}$$

holds for all $h \in \mathcal{H}$, and if $\hat{h}$ minimizes the training error $\widehat{\text{err}}(h)$, then $\hat{h}$ also approximately minimizes the true error since

$$\begin{aligned} \text{err}(\hat{h}) &\leq \widehat{\text{err}}(\hat{h}) + \varepsilon \\ &= \min_{h \in \mathcal{H}} \widehat{\text{err}}(h) + \varepsilon \\ &\leq \min_{h \in \mathcal{H}} (\text{err}(h) + \varepsilon) + \varepsilon \\ &= \min_{h \in \mathcal{H}} \text{err}(h) + 2\varepsilon. \end{aligned} \tag{2.8}$$

(We assumed that the minima above exist, as will be the case, for instance, if $\mathcal{H}$ is finite; it is straightforward to modify the argument if they do not.)

Although we assumed a two-sided bound as in Eq. (2.7), we will henceforth restrict our attention primarily to proving one-sided bounds of the form

$$\text{err}(h) \leq \widehat{\text{err}}(h) + \varepsilon \tag{2.9}$$

for all $h \in \mathcal{H}$. We do this for simplicity of presentation, but also for the reason that we typically are only interested in upper bounds on generalization error. This is because, first of all, there is no harm done if the learning algorithm manages to get lucky in picking a hypothesis whose generalization error is significantly *lower* than its training error. But more importantly, this case almost never occurs: since the learning algorithm is biased toward choosing hypotheses with already low training error, the generalization error is quite unlikely to be even lower. In fact, a closer look at the argument in Eq. (2.8) reveals that only one-sided uniform error bounds as in Eq. (2.9) were actually used.

To prove that such uniform bounds hold with high probability, the simplest case is when $\mathcal{H}$ is finite so that the hypothesis h is selected from a *finite* set of hypotheses that is fixed before observing the training set. In this case, we can use a simple argument based on the union bound: If we fix any *single* hypothesis $h \in \mathcal{H}$, then as in Section 2.2.1, we can use Theorem 2.1 to bound the probability of choosing a training set for which $\text{err}(h) - \widehat{\text{err}}(h) \geq \varepsilon$; this will be at most $e^{-2m\varepsilon^2}$. By the union bound, the chance that this happens for *any* hypothesis in $\mathcal{H}$ can be upper bounded simply by summing this probability bound over all the hypotheses in $\mathcal{H}$, which gives $|\mathcal{H}|e^{-2m\varepsilon^2}$. Thus, we obtain the following:

Theorem 2.2 *Let $\mathcal{H}$ be a finite space of hypotheses, and assume that a random training set of size m is chosen. Then for any $\varepsilon > 0$,*

$$\Pr\left[\exists h \in \mathcal{H} : \mathrm{err}(h) \geq \widehat{\mathrm{err}}(h) + \varepsilon\right] \leq |\mathcal{H}|e^{-2m\varepsilon^2}.$$

Thus, with probability at least $1 - \delta$,

$$\mathrm{err}(h) \leq \widehat{\mathrm{err}}(h) + \sqrt{\frac{\ln|\mathcal{H}| + \ln(1/\delta)}{2m}} \tag{2.10}$$

for all $h \in \mathcal{H}$.

The second bound (Eq. (2.10)) on the generalization performance of any hypothesis h captures in a single formula the three factors noted earlier which determine the success of a learning algorithm: With regard to the training error $\widehat{\mathrm{err}}(h)$, the first of these factors, we see that the formula is consistent with the intuition that better fit to the training data implies better generalization. The second factor is the number of training examples m, where again the bound captures quantitatively the unarguable notion that more data is better. Finally, the formula contains a term that depends on the form of the hypotheses being used, that is, on the class $\mathcal{H}$ from which they are chosen. Note that if hypotheses in $\mathcal{H}$ are written down in some way, then the number of bits needed to give each hypothesis $h \in \mathcal{H}$ a unique name is about $\lg|\mathcal{H}|$. Thus, we can think of $\ln|\mathcal{H}|$—the term appearing in the formula, which is only off by a constant from $\lg|\mathcal{H}|$—as roughly the "description length" of the hypotheses being used, rather a natural measure of $\mathcal{H}$'s complexity, the third factor affecting generalization performance. In this way, the formula is consistent with Occam's Razor, the notion that, all else being equal, simpler hypotheses perform better than more complex ones.

2.2.3 Infinite hypothesis spaces

We next consider infinitely large hypothesis spaces. The results of the last section are useless in such cases, for instance, when using threshold rules of the form given in Eq. (2.1) where there are infinitely many choices for the threshold ν, and thus are infinitely many hypotheses of this form. However, this hypothesis space has an important property that was noted in Section 2.1: any training set of m (distinct) examples partitions the infinitely large space into just $m + 1$ equivalence classes such that the predictions of any two functions in the same equivalence class are identical on all m points. Said differently, the number of distinct labelings of the m training points that can be produced by functions in $\mathcal{H}$ is at most $m + 1$. See Figure 2.4 for an example.

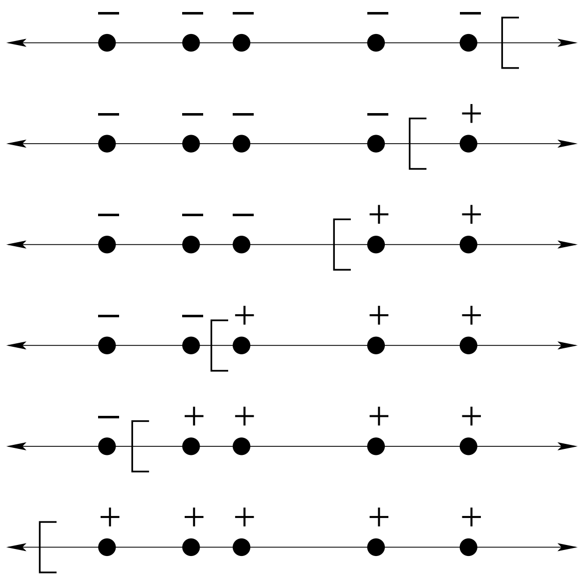


Figure 2.4: Five points on the real line, and a listing of all six possible labelings of the points using threshold functions of the form shown in Eq. (2.1). The left bracket on each line "[" shows a sample threshold ν realizing the given labeling.

Thus, in this case, although there are infinitely many hypotheses in $\mathcal{H}$, there are, in a sense, effectively only $m+1$ hypotheses of any relevance with respect to a fixed set of m examples. It would be tempting indeed to regard $m+1$ as the "effective" size of the hypothesis space, and to then attempt to apply Theorem 2.2 with $|\mathcal{H}|$ replaced by $m+1$. This would give very reasonable bounds. Unfortunately, such an argument would be flawed because the finite set of effective hypotheses that is induced in this way *depends on the training data* so that the argument suggested earlier for proving Theorem 2.2 using the union bound cannot be applied. Nevertheless, using more sophisticated arguments, it turns out to be possible to prove that this alluring idea actually works so that, modulo some adjusting of the constants, $|\mathcal{H}|$ can be replaced in Theorem 2.2 with the "effective" size of $\mathcal{H}$ as measured by the number of labelings of $\mathcal{H}$ on a finite sample.

To make these ideas formal, for any hypothesis class $\mathcal{H}$ over $\mathcal{X}$ and for any finite sample $S = \{x_1, \ldots, x_m\}$, we define the set of *dichotomies* or *behaviors* $\Pi_{\mathcal{H}}(S)$ to be all possible labelings of S by functions in $\mathcal{H}$. That is,

$$\Pi_{\mathcal{H}}(S) \doteq \{\langle h(x_1), \ldots, h(x_m)\rangle : h \in \mathcal{H}\}.$$

We also define the *growth function* $\Pi_{\mathcal{H}}(m)$ which measures the maximum number of dichotomies for any set S of size m:

$$\Pi_{\mathcal{H}}(m) \doteq \max_{S:|S|=m} |\Pi_{\mathcal{H}}(S)|.$$

For instance, when $\mathcal{H}$ is the class of threshold functions, $\Pi_{\mathcal{H}}(m) = m+1$, as we have already seen.

We can now state a more general result than Theorem 2.2 that is applicable to both finite and infinite hypothesis spaces. Ignoring constants, this theorem has replaced $|\mathcal{H}|$ by the growth function $\Pi_{\mathcal{H}}(m)$.

Theorem 2.3 *Let $\mathcal{H}$ be any*[1] *space of hypotheses, and assume that a random training set of size m is chosen. Then for any $\varepsilon > 0$,*

$$\Pr\left[\exists h \in \mathcal{H} : \text{err}(h) \geq \widehat{\text{err}}(h) + \varepsilon\right] \leq 8\Pi_{\mathcal{H}}(m)e^{-m\varepsilon^2/32}.$$

Thus, with probability at least $1-\delta$,

$$\text{err}(h) \leq \widehat{\text{err}}(h) + \sqrt{\frac{32(\ln \Pi_{\mathcal{H}}(m) + \ln(8/\delta))}{m}} \tag{2.11}$$

for all $h \in \mathcal{H}$.

[1]To be strictly formal, we have to restrict the class $\mathcal{H}$ to be measurable with respect to an appropriate probability space. Here, and throughout this book, we ignore this finer point which we implicitly assume to hold.

So our attention naturally turns to the growth function $\Pi_{\mathcal{H}}(m)$. In "nice" cases, such as for threshold functions, the growth function is only polynomial in m, that is, $O(m^d)$ for some constant d dependent on $\mathcal{H}$. In such a case, $\ln \Pi_{\mathcal{H}}(m)$ is roughly $d \ln m$ so that the term on the far right of Eq. (2.11), as a function of the training set size m, approaches zero at the favorable rate $O(\sqrt{(\ln m)/m})$.

However, the growth function need not always be polynomial. For instance, consider the class of all hypotheses that are defined to be $+1$ on some finite but unrestricted set of intervals of the real line and -1 on the complement (where instances here are points on the line). An example of a hypothesis in this class is given by the classifier in Eq. (2.2) which is $+1$ on the intervals $[3.4, 4.0)$, $[5.2, 6.3)$ and $[7.1, \infty)$, and -1 on all other points. This hypothesis space is so rich that for *any* labeling of any set of distinct training points, there always exists a consistent classifier which can be constructed simply by choosing sufficiently small intervals around each of the positively labeled instances. Thus, the number of dichotomies for any set of m distinct points is exactly 2^m, the worst possible value of the growth function. In such a case, $\ln \Pi_{\mathcal{H}}(m) = m \ln 2$ so that the bound in Theorem 2.3 is useless, being of order $\theta(1)$. On the other hand, the very richness that makes it so easy to find a hypothesis consistent with any training set also strongly suggests that the generalization capabilities of such hypotheses will be very weak indeed.

So we have seen one case in which $\Pi_{\mathcal{H}}(m)$ is polynomial, and another in which it is 2^m for all m. It is a remarkable fact of combinatorics that these are the *only* two behaviors that are possible for the growth function, no matter what the space $\mathcal{H}$. Moreover, as we will soon see, statistically tractable learning turns out to correspond exactly to the former case, with the exponent of the polynomial acting as a natural measure of the complexity of the class $\mathcal{H}$.

To characterize this exponent, we now define some key combinatorial concepts. First, when all 2^m possible labelings of a set S of size m can be realized by hypotheses in $\mathcal{H}$, we say that S is *shattered* by $\mathcal{H}$. Thus, S is shattered by $\mathcal{H}$ if $|\Pi_{\mathcal{H}}(S)| = 2^{|S|}$. Further, we define the *Vapnik-Chervonenkis (VC) dimension* of $\mathcal{H}$ to be the cardinality of the largest set S shattered by $\mathcal{H}$. If arbitrarily large finite sets can be shattered by $\mathcal{H}$ then the VC-dimension is ∞.

For instance, for threshold functions as in Eq. (2.1), the VC-dimension is 1 since a single point can be labeled $+1$ or -1 by such rules (which means the VC-dimension is at least 1), but no pair of points can be shattered since if the leftmost point is labeled $+1$, the rightmost point must also be labeled $+1$ (thus, the VC-dimension is strictly less than 2). For the unions-of-intervals example above, we saw that any set of distinct points is shattered, so the VC-dimension is ∞ in this case.

Indeed, when the VC-dimension is infinite, $\Pi_{\mathcal{H}}(m) = 2^m$, by definition. On the other hand, when the VC-dimension is a finite number d, the growth function turns out to be polynomial, specifically, $O(m^d)$. This follows from a beautiful combinatorial fact known as Sauer's Lemma:

Lemma 2.4 (Sauer's Lemma) *If $\mathcal{H}$ is a hypothesis class of VC-dimension $d < \infty$, then for all m*

$$\Pi_{\mathcal{H}}(m) \leq \sum_{i=0}^{d} \binom{m}{i}.$$

(We follow the convention that $\binom{n}{k} = 0$ if $k < 0$ or $k > n$.) For $m \leq d$, this bound is equal to 2^m (and indeed, by d's definition, the bound matches $\Pi_{\mathcal{H}}(m)$ in this case). For $m \geq d \geq 1$, the following bound is often useful:

$$\sum_{i=0}^{d} \binom{m}{i} \leq \left(\frac{em}{d}\right)^d \tag{2.12}$$

(where, as usual, e is the base of the natural logarithm).

We can plug this bound immediately into Theorem 2.3 to obtain the following:

Theorem 2.5 *Let $\mathcal{H}$ be a hypothesis space of VC-dimension $d < \infty$, and assume that a random training set of size m is chosen where $m \geq d \geq 1$. Then for any $\varepsilon > 0$,*

$$\Pr\left[\exists h \in \mathcal{H} : \mathrm{err}(h) \geq \widehat{\mathrm{err}}(h) + \varepsilon\right] \leq 8\left(\frac{em}{d}\right)^d e^{-m\varepsilon^2/32}.$$

Thus, with probability at least $1 - \delta$,

$$\mathrm{err}(h) \leq \widehat{\mathrm{err}}(h) + O\left(\sqrt{\frac{d\ln(m/d) + \ln(1/\delta)}{m}}\right) \tag{2.13}$$

for all $h \in \mathcal{H}$.

As before, this second bound on the generalization error captures the three factors discussed earlier. However, the complexity of $\mathcal{H}$, which was earlier measured by $\ln|\mathcal{H}|$, now is measured instead by the VC-dimension d. This is an important result because it shows that limiting the VC-dimension of a hypothesis class can be used as a general tool for avoiding overfitting.

The VC-dimension may not at first seem intuitive as a measure of complexity. However, it can be shown that the VC-dimension can also be used to provide a general lower bound on the number of examples needed for learning. Thus, in this sense, VC-dimension fully characterizes the (statistical) complexity of learning. Moreover, VC-dimension is related to our earlier complexity measure in that it can never exceed $\lg|\mathcal{H}|$ (since, for any set S, $|\Pi_{\mathcal{H}}(S)|$ is trivially upper-bounded by

$|\mathcal{H}|$, and since, by definition of VC-dimension, there exists a set S of size d for which $|\Pi_{\mathcal{H}}(S)| = 2^d$).

In addition, VC-dimension turns out often (but not always!) to be equal to the number of parameters defining hypotheses in $\mathcal{H}$. For instance, suppose that examples are points $\mathbf{x}$ in $\mathbb{R}^n$, and that hypotheses in $\mathcal{H}$ are *linear threshold functions* of the form

$$h(\mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{x} > 0 \\ -1 & \text{else} \end{cases}$$

for some weight vector $\mathbf{w} \in \mathbb{R}^n$. Then it can be shown (see Lemma 4.1) that the VC-dimension of this class is exactly n, which matches the number of parameters, that is, the number of dimensions in the vector $\mathbf{w}$ that defines each hypothesis in $\mathcal{H}$.

2.2.4 A more abstract formulation

The framework above, particularly Theorem 2.3, can be stated in more abstract terms that we will sometimes find easier to work with. However, the reader may wish to skip this technical section and come back to it when needed later in the book.

Briefly, let $\mathcal{Z}$ be any set, let $\mathcal{A}$ be a family of subsets of $\mathcal{Z}$, and let $\mathcal{D}$ be a distribution over $\mathcal{Z}$. We consider the problem of estimating from a random sample S the probability of each set $A \in \mathcal{A}$. We wish to show that the empirical probability $\Pr_{z \in S}[z \in A]$ (that is, probability when z is chosen uniformly at random from S) is likely to be close to the true probability $\Pr_{z \in \mathcal{D}}[z \in A]$, for every set $A \in \mathcal{A}$. For a single, fixed set A, this can be done using Hoeffding's inequality (Theorem 2.1). Likewise, if $\mathcal{A}$ is finite, then this is likely to be true for all $A \in \mathcal{A}$ by an application of the union bound. But when $\mathcal{A}$ is infinite, we need to generalize the machinery developed in Section 2.2.3.

For any finite subset S of $\mathcal{Z}$, we consider the restriction of $\mathcal{A}$ to S, denoted $\Pi_{\mathcal{A}}(S)$, that is, the intersection of S with each set $A \in \mathcal{A}$:

$$\Pi_{\mathcal{A}}(S) \doteq \{S \cap A : A \in \mathcal{A}\}.$$

Analogous to $\Pi_{\mathcal{H}}(S)$, this collection can be viewed as the set of all "in-out behaviors" of sets $A \in \mathcal{A}$ on the points in S. As before, the growth function is the maximum cardinality of this set over all sets S of size m:

$$\Pi_{\mathcal{A}}(m) \doteq \max_{S:|S|=m} |\Pi_{\mathcal{A}}(S)|.$$

With these definitions, Theorem 2.3 becomes:

Theorem 2.6 *Let $\mathcal{A}$ be a family of subsets of $\mathcal{Z}$, and suppose that a random set S of m points is chosen independently from $\mathcal{Z}$, each point selected according to the same distribution $\mathcal{D}$. Then for any $\varepsilon > 0$,*

$$\Pr\left[\exists A \in \mathcal{A} : \Pr_{z\sim\mathcal{D}}[z \in A] \geq \Pr_{z\sim S}[z \in A] + \varepsilon\right] \leq 8\Pi_{\mathcal{A}}(m)e^{-m\varepsilon^2/32}.$$

Thus, with probability at least $1 - \delta$,

$$\Pr_{z\sim\mathcal{D}}[z \in A] \leq \Pr_{z\sim S}[z \in A] + \sqrt{\frac{32(\ln \Pi_{\mathcal{A}}(m) + \ln(8/\delta))}{m}}$$

for all $A \in \mathcal{A}$.

To obtain the formulation given in Section 2.2.3 as a special case, we first let $\mathcal{Z} = \mathcal{X} \times \{-1, +1\}$, the space of all possible labeled examples. Then, for a given hypothesis space $\mathcal{H}$, we define a family $\mathcal{A}$ of subsets A_h, one for each $h \in \mathcal{H}$, where A_h is the set of examples on which h makes a mistake. That is,

$$\mathcal{A} = \{A_h : h \in \mathcal{H}\} \tag{2.14}$$

and

$$A_h = \{(x, y) \in \mathcal{Z} : h(x) \neq y\}.$$

Then it can be verified with these definitions that

$$\Pi_{\mathcal{H}}(m) = \Pi_{\mathcal{A}}(m) \tag{2.15}$$

and that Theorem 2.6 yields exactly Theorem 2.3.

2.2.5 Consistent hypotheses

We have seen that, when possible, it is sometimes desirable for a learning algorithm to produce a hypothesis that is *consistent* with the training data so that it makes no mistakes at all on the training set. Of course, our preceding analyses hold; this is just a special case in which $\widehat{\text{err}}(h) = 0$. However, the bounds so obtained in this fashion are particularly loose, giving bounds on the order of $1/\sqrt{m}$. In fact, for consistent hypotheses, it turns out that the square roots on all of our convergence bounds can generally be removed (with some adjusting of constants), giving the far faster convergence rate of just $1/m$ (ignoring log terms).

To get an intuitive feeling for why this is so, consider again the problem of estimating the bias p of a coin, as in Section 2.2.1. It turns out that a coin with bias p close to 0 or 1 is much easier to estimate (in the sense of requiring fewer samples) than a coin with bias close to $1/2$. This is reflected in the bounds that can be proved. According to Hoeffding's inequality (Theorem 2.1), if $\hat{p}$ is the observed fraction of heads in m flips, then

$$p \leq \hat{p} + \sqrt{\frac{\ln(1/\delta)}{2m}} \tag{2.16}$$

with probability at least $1 - \delta$. In other words, the true probability p is within $O(1/\sqrt{m})$ of its estimate $\hat{p}$.

Now let us consider what happens when $\hat{p} = 0$, that is, when there are no heads in the observed sequence, corresponding to the case of a hypothesis that is consistent with the entire training set. The probability of getting *no* heads in m flips is exactly $(1-p)^m \leq e^{-pm}$. This means that if $p \geq \ln(1/\delta)/m$ then $\hat{p}$ will be zero with probability at most δ. Turning this statement around implies that when $\hat{p} = 0$, we can conclude that

$$p < \frac{\ln(1/\delta)}{m}$$

with probability at least $1 - \delta$. Note that this estimate is $O(1/m)$ rather than $O(1/\sqrt{m})$ as in Eq. (2.16).

This style of argument can be applied in the learning setting as well, yielding results such as those summarized in the following theorem:

Theorem 2.7 *Let $\mathcal{H}$ be a space of hypotheses, and assume that a random training set S of size m is chosen.*

If $\mathcal{H}$ is finite, then with probability at least $1 - \delta$,

$$\text{err}(h) \leq \frac{\ln|\mathcal{H}| + \ln(1/\delta)}{m} \tag{2.17}$$

for every $h \in \mathcal{H}$ that is consistent with S.

More generally, for any (finite or infinite) $\mathcal{H}$, with probability at least $1 - \delta$,

$$\text{err}(h) \leq \frac{2\lg\Pi_{\mathcal{H}}(2m) + 2\lg(2/\delta)}{m} \tag{2.18}$$

for every $h \in \mathcal{H}$ that is consistent with S. If $\mathcal{H}$ has VC-dimension d, where $m \geq d \geq 1$, then with probability at least $1 - \delta$,

$$\text{err}(h) \leq \frac{2d\lg(2em/d) + 2\lg(2/\delta)}{m} \tag{2.19}$$

for every $h \in \mathcal{H}$ that is consistent with S.

2.2.6 Compression-based bounds

We have described two general techniques for analyzing a learning algorithm, one based simply on counting the number of hypotheses in the class $\mathcal{H}$, and the other based on the VC-dimension of the class $\mathcal{H}$. In this section, we briefly describe a third approach.

It has already been noted that $\lg |\mathcal{H}|$, our first complexity measure, is closely related to the number of bits needed to describe each hypothesis $h \in \mathcal{H}$. Thus, from this perspective, the learning algorithm must find a relatively short description that can be used to reconstruct labels for the training examples. This idea depends on the description being in bits, and thus on $\mathcal{H}$ being finite.

Nevertheless, even when $\mathcal{H}$ is infinite, it will often be the case that the hypotheses produced by a particular algorithm can be given a short description, not in bits, but in terms of *training examples* themselves. For instance, for the class of threshold functions as in Eq. (2.1), each classifier is defined by the threshold ν. Moreover, as we have seen, the behavior of such classifiers is equivalent with respect to a particular training set for all thresholds ν lying between two adjacent data points. Thus, a learning algorithm can choose ν to be one of those data points, and in so doing, produces a classifier that can be described by just one of the training examples. (Alternatively, if it seems more natural, the learner can take ν to be the midpoint between two adjacent data points. This hypothesis can be described by two of the training examples.)

We call such an algorithm whose hypotheses can be represented by κ of the training examples a *compression scheme of size* κ. Thus, formally, such an algorithm is associated with a function K that maps κ-tuples of labeled examples to hypotheses h in some space $\mathcal{H}$. Given training examples $(x_1, y_1), \ldots, (x_m, y_m)$, such an algorithm chooses some indices $i_1, \ldots, i_\kappa \in \{1, \ldots, m\}$, and outputs the hypothesis

$$h = K((x_{i_1}, y_{i_1}), \ldots, (x_{i_\kappa}, y_{i_\kappa}))$$

determined by the corresponding examples.

For such algorithms, which arise quite often and quite naturally, bounds on the generalization error can be derived in much the same way as for the case of finite $\mathcal{H}$. In particular, the following can be proved:

Theorem 2.8 *Suppose a learning algorithm based on a compression scheme of size κ is provided with a random training set of size m. Then with probability at least $1 - \delta$, the hypothesis h produced by this algorithm satisfies*

$$\text{err}(h) \le \left(\frac{m}{m-\kappa}\right) \widehat{\text{err}}(h) + \sqrt{\frac{\kappa \ln m + \ln(1/\delta)}{2(m-\kappa)}}.$$

Furthermore, with probability at least $1 - \delta$*, any consistent hypothesis* h *produced by this algorithm satisfies*

$$\mathrm{err}(h) \leq \frac{\kappa \ln m + \ln(1/\delta)}{m - \kappa}.$$

Thus, for such algorithms, it is the size κ of the compression scheme that acts as a complexity term.

2.2.7 Discussion

We have explored three general methods for analyzing learning algorithms. The techniques are closely related, differing primarily in the complexity measure employed. These bounds are extremely useful in understanding the *qualitative* behavior of learning algorithms. As we have discussed already, the bounds describe the dependence of the generalization error on the training error, the number of examples, and the complexity of the chosen hypothesis. Furthermore, these bounds are quite helpful in understanding the important and ubiquitous phenomenon of overfitting: Disregarding δ and log factors, each of the bounds in Theorems 2.2, 2.5 and 2.8 have the form

$$\mathrm{err}(h) \leq \widehat{\mathrm{err}}(h) + \tilde{O}\left(\sqrt{\frac{C_{\mathcal{H}}}{m}}\right) \tag{2.20}$$

where $C_{\mathcal{H}}$ is some measure of the complexity of $\mathcal{H}$. As the complexity of the hypotheses used is permitted to increase, the training error tends to decrease causing the first term in Eq. (2.20) to decrease. However, this also causes the second term to increase. The result is an idealized "learning curve" like the one in Figure 2.5 which pretty well matches the kind of overfitting behavior often observed in practice.

On the other hand, these bounds are usually too loose to be applied *quantitatively* on actual learning problems. In most cases, the bounds suggest that, with the amount of training data available, only very simple hypothesis classes can be used, while in actual practice, quite large hypothesis classes are used regularly and with good results. The problem is that the bounds are overly pessimistic, holding as they do for *all* distributions, including those "worst-case" distributions which make learning as difficult as possible. Thus, while the uniform nature of the bounds is an unquestionable strength that lends generality and robustness to the results, this same uniformity can also be a weakness in the sense that the results may better characterize the theoretical worst case than the actual case encountered in practice.

One way to tighten the bounds may be to take into account additional quantities that can be measured on the training set. Bounds of the type given in the theorems

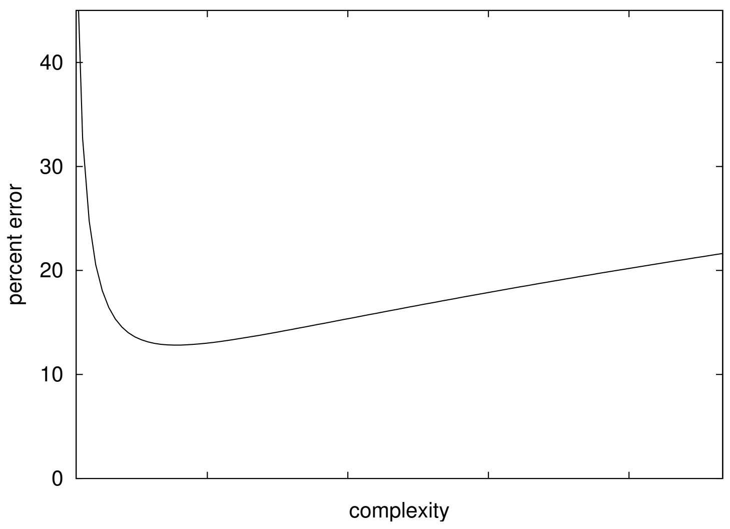


Figure 2.5: An idealized plot of the generalization error of a classifier with varying complexity, as predicted by a bound of the form in Eq. (2.20).

above take into account only the training error $\widehat{\mathrm{err}}(h)$, but other quantities can be considered. For instance, Chapter 5 describes bounds on the generalization error of boosting algorithms which depend on properties of the *margin distribution* of the training set.

2.3 A foundation for the study of boosting algorithms

We next introduce an idealized framework for the mathematical study of machine learning, one that admits absolute guarantees on performance. As we will see, this model of learning will allow us to define the concept of boosting in precise terms, and will thus provide a foundation for the development and analysis of boosting algorithms.

2.3.1 Absolute guarantees on performance

As just discussed, the mode of analysis studied in Section 2.2 is very general and quite agnostic in the sense that we have made no assumptions about the form of the distribution $\mathcal{D}$ generating the labeled examples (x, y). On the one hand, this generality has made it possible to state results that are applicable in a very broad range of settings without the need for prior assumptions about the data distribution. On the other hand, this same generality has also precluded us from providing *absolute* guarantees on performance. Rather, the bounds that we have discussed tell us that the generalization error will be small *if* we can place our hands during training on a simple hypothesis with low training error on a sufficiently large training set. The bounds do not tell us when it will be possible to obtain a hypothesis that has, say, 99% generalization accuracy. This can only be deduced, according to the bounds, *after* the training error has been observed.

Even as the training set becomes extremely large, the bounds do not guarantee low generalization error. Indeed, such guarantees are impossible in such generality since the distribution $\mathcal{D}$ may be such that the label y is intrinsically unpredictable. For instance, since we assume nothing about $\mathcal{D}$, it is possible that y is equally likely to be $+1$ or -1, independent of x; in such an extreme case, no amount of training or computation can result in a hypothesis with generalization error less than 50%. Thus, to place absolute guarantees on the generalization error, we must accept additional assumptions about the form of the generating distribution $\mathcal{D}$.

Suppose our goal is to learn a classifier with nearly perfect accuracy. We have just seen that this is not always possible, and indeed, realistically, this is almost never attainable in practice. Nevertheless, achieving the highest possible accuracy is the ultimate goal of learning, and as such, understanding theoretically when near perfect accuracy can be achieved is a fundamental question.

For starters, as we have seen, when there is intrinsic randomness in the labels y themselves, so that y is not strictly determined by x, there is no way to find a hypothesis h which can predict y perfectly for any x, since such a hypothesis does not exist. Thus, the first assumption we will need to adopt is that there exists a functional relationship between the instances x and the labels y; in other words, we assume that there exists a *target function*

$$c : \mathcal{X} \to \{-1, +1\}$$

such that, for any x, the associated label y is equal to $c(x)$ with probability 1. That is,

$$\Pr_{(x,y)\sim\mathcal{D}}[y = c(x) \mid x] = 1.$$

This is equivalent to simply regarding $\mathcal{D}$ as a distribution over $\mathcal{X}$, and assuming that examples are of the form $(x, c(x))$ so that each example x is deterministically assigned the label $c(x)$.

Even with such deterministic labeling of the data, learning may be impossible. For instance, if the target function c is an entirely arbitrary function over $\mathcal{X}$, then no finite training set can provide any connection to examples not seen during training; therefore, we cannot hope to find a hypothesis that makes accurate predictions, other than on the examples seen during training. Thus, for generalization to be plausible, we also must make assumptions about the nature of the target function so that there can exist some relationship between the labels of the training data and those on test examples. We can summarize any such knowledge about c by assuming that c comes from some known class of functions $\mathcal{C}$, called the *target (function) class*.

So our problem of understanding the circumstances under which nearly perfect learning is possible can be rephrased as that of determining for which classes $\mathcal{C}$, embodying our assumptions about the target c, is such learning achievable. Here, again, our goal is to learn a hypothesis h with nearly perfect accuracy, that is, with generalization error below ϵ for any specified value of $\epsilon > 0$, though naturally more data will need to be allowed for smaller values of ϵ. Moreover, to avoid further assumptions, we ask that learning be possible for any distribution $\mathcal{D}$ over the instance space $\mathcal{X}$, and any target function $c \in \mathcal{C}$. Finally, since there is always the possibility that a highly unrepresentative random training sample will be selected, we allow learning to fail entirely with probability $\delta > 0$, where δ should be controllable in a manner similar to ϵ.

So this notion of learning requires that, with high probability over the choice of the random sample, a hypothesis h be found that is nearly perfect, or approximately correct, hence the name, *probably approximately correct* or *PAC* learnability. To distinguish from what is to follow, we also call this *strong learnability*. Formally, then, a class $\mathcal{C}$ is *strongly PAC learnable* if there exists a learning algorithm A such that for any distribution $\mathcal{D}$ over the instance space $\mathcal{X}$, and for any $c \in \mathcal{C}$ and for any positive values of ϵ and δ, algorithm A takes as input $m = m(\epsilon, \delta)$ examples $(x_1, c(x_1)), \ldots, (x_m, c(x_m))$ where x_i is chosen independently at random according to $\mathcal{D}$, and produces a hypothesis h such that

$$\Pr\left[\mathrm{err}(h) > \epsilon\right] \leq \delta.$$

Here, $\mathrm{err}(h)$ is the generalization error of h with respect to distribution $\mathcal{D}$, and the probability is over the random selection of training examples $x_1, \ldots, x_m$. Note that the training set size m can depend on ϵ and δ, and we typically require these to be polynomial in $1/\epsilon$ and $1/\delta$. Further, the sample size will usually also depend on properties of the class $\mathcal{C}$.

When computability is not an issue, we can immediately apply the results of Section 2.2.5 to obtain quite general PAC results. In particular, for any class $\mathcal{C}$, consider an algorithm A that, given any sample, selects h to be any function in $\mathcal{C}$ that is consistent with the observed data. By assumption, such a hypothesis h must exist (although the computational considerations of finding it might be prohibitive). Thus, our hypothesis space $\mathcal{H}$, in this case, is identical to $\mathcal{C}$. When $\mathcal{C}$ is finite, applying Theorem 2.7, we thus get immediately that with probability at least $1 - \delta$,

$$\text{err}(h) \le \frac{\ln |\mathcal{C}| + \ln(1/\delta)}{m}.$$

Setting the right-hand side equal to ϵ, this means that A is PAC (so that $\text{err}(h) \le \epsilon$ with probability at least $1 - \delta$) when given a sample of size

$$m = \left\lceil \frac{\ln |\mathcal{C}| + \ln(1/\delta)}{\epsilon} \right\rceil.$$

Similarly, for $\mathcal{C}$ infinite, the algorithm can be shown to be PAC for some sample size that depends on the VC-dimension of $\mathcal{C}$ (provided it is finite).

Thus, our generalization bounds indicate generally that only a moderate size sample is statistically adequate for PAC learning. What remains is the problem of finding an efficient learning algorithm, an issue that cannot be ignored in the real world. Therefore, we often add an efficiency requirement that the learning algorithm A compute h in polynomial time. Characterizing *efficient* PAC learnability turns out to be much more difficult and involved. Indeed, computational tractability has very often been found to be far more limiting and constraining in the design of learning systems than any statistical considerations. In other words, it is often the case that a learning problem cannot be solved, even when more than enough data has been provided to ensure statistical generalization, solely because the associated computational problem is intractable. (For instance, it is known that this is the case when $\mathcal{C}$ is the class of all polynomial-size formulas constructed using the usual operations (AND, OR and NOT) over n boolean variables.)

2.3.2 Weak Learnability and Boosting Algorithms

As indicated above, requiring nearly perfect generalization is usually unrealistic, sometimes for statistical reasons and often for purely computational reasons. What happens if we drop this overly stringent requirement? In other words, rather than requiring a generalization error below, say, 1%, what if we were content with error below 10%? or 25%? In the most extreme case, what if our goal is merely to find a hypothesis whose error is just slightly below the trivial baseline of 50% which is attainable simply by guessing every label at random? Surely, learning must become easier when (far) less-than-perfect accuracy is deemed sufficient.

Learning with such a weak demand on accuracy is called *weak learning*. In terms of its definition, the problem is only a slight modification of PAC learning in which we drop the requirement that the learning algorithm achieve error at most ϵ for *every* $\epsilon > 0$. Rather, we only make this requirement for some *fixed* ϵ, say $\epsilon = \frac{1}{2} - \gamma$, for some fixed but small "edge" $\gamma > 0$. Formally, then, a target class

$\mathcal{C}$ is *weakly PAC learnable* if for some $\gamma > 0$, there exists a learning algorithm A such that for any distribution $\mathcal{D}$ over the instance space $\mathcal{X}$, and for any $c \in \mathcal{C}$ and for any positive value of δ, algorithm A takes as input $m = m(\delta)$ examples $(x_1, c(x_1)), \ldots, (x_m, c(x_m))$ where x_i is chosen independently at random according to $\mathcal{D}$, and produces a hypothesis h such that

$$\Pr\left[\text{err}(h) > \tfrac{1}{2} - \gamma\right] \leq \delta.$$

Weak learnability arises as a natural relaxation of the overly demanding strong learning model. Indeed, on its face, one might be concerned that the model is now too weak, accepting as it does any hypothesis that is even slightly better than random guessing. Surely, it would seem, there must be many examples of classes that are weakly learnable (with accuracy only 51%) but not strongly learnable (to accuracy 99%).

This intuition, which turns out to be entirely incorrect, points to a fundamental question: Are the strong and weak learning models equivalent? In other words, is it the case that there exist classes that are weakly but not strongly learnable? Or, is it the case that any class that can be weakly learned can also be strongly learned? Boosting arose as an answer to exactly this theoretical question. The existence of boosting algorithms proves that the models are equivalent by showing constructively that any weak learning algorithm can be converted into a strong learning algorithm. Indeed, it is exactly this property that defines boosting in its true technical sense.

Formally, a *boosting algorithm B* is given access to a weak learning algorithm A for $\mathcal{C}$ which, when provided with $m_0(\delta)$ examples, is guaranteed to produce a weak hypothesis h with $\text{err}(h) \leq \frac{1}{2} - \gamma$ with probability at least $1 - \delta$. In addition, like any PAC algorithm, B is provided with $\epsilon > 0$, $\delta > 0$, and m labeled examples $(x_1, c(x_1)), \ldots, (x_m, c(x_m))$ for some $c \in \mathcal{C}$ (where $\mathcal{C}$ need not be known to B). Using its access to A, the boosting algorithm must produce its own (strong) hypothesis H such that

$$\Pr\left[\text{err}(H) > \epsilon\right] \leq \delta. \tag{2.21}$$

Further, there should only be a polynomial blow-up in complexity. In other words, B's sample size m should only be polynomial in $1/\epsilon$, $1/\delta$, $1/\gamma$ and m_0, and similarly, B's running time should be polynomially related to A's (as well as the other parameters). Clearly, applying such an algorithm to a weak learning algorithm for some class $\mathcal{C}$ demonstrates, by definition, that the class is strongly learnable as well. AdaBoost is indeed a boosting algorithm in this technical sense, as will be discussed in Section 4.3.

That strong and weak learnability are equivalent implies that learning, as we have defined it, is an "all or nothing" phenomenon in the sense that for every class

$\mathcal{C}$, either $\mathcal{C}$ is learnable with nearly perfect accuracy for every distribution, or $\mathcal{C}$ cannot even be learned in the most minimal way, on every distribution. There is nothing in between these two extremes.

2.3.3 Approaches to the analysis of boosting algorithms

In this chapter, we have explored two modes of analysis. In the first, the generalization error of a selected hypothesis is bounded in terms of measurable empirical statistics, most commonly its training error. No explicit assumptions are made about the data, and as a result, good generalization depends on an implicit assumption that a hypothesis with low training error can be found. In the second style of analysis, additional assumptions are made about the underlying data generation process, admitting absolute bounds on generalization. In the same way, boosting algorithms can be analyzed using either approach.

Every learning algorithm depends explicitly or implicitly upon assumptions since learning is quite impossible otherwise. Boosting algorithms are built upon the assumption of weak learnability, the premise that a method already exists for finding poor though not entirely trivial classifiers. In its original form, the boosting question begins by assuming that a given class $\mathcal{C}$ is weakly PAC learnable as defined above. With such an assumption, as we have seen, it is possible to prove strong absolute guarantees on the PAC learnability of $\mathcal{C}$, ensuring near perfect generalization.

However, in practical settings, this assumption may be too onerous, requiring that the labels be deterministic according to a target function from a known class $\mathcal{C}$, that weak learning hold for every distribution, and that the edge γ be known ahead of time. Practically, these requirements can be very difficult to check or guarantee. As we now discuss, there are many ways in which these assumptions can be weakened, and most (but not all) of the book is founded on such relaxed versions of the weak learning assumption.

To begin, for the sake of generality, we can usually drop any explicit assumptions about the data, returning to the more agnostic framework seen earlier in the chapter in which labeled examples (x, y) are generated by an arbitrary distribution $\mathcal{D}$ with no functional dependence assumed for the labels y. In this case, as in Chapter 1, the training set consists simply of labeled pairs $(x_1, y_1), \ldots, (x_m, y_m)$.

Furthermore, rather than the far-reaching assumption of weak PAC learnability described above, we can instead assume only that the weak hypotheses found by the given weak learning algorithm A have weighted *training* error bounded away from $1/2$. This leads to a different and weaker notion of weak learnability that is particular to the actual training set. Specifically, we say that the *empirical γ-weak learning assumption* holds if for any distribution D on the indices $\{1, \ldots, m\}$ of the training examples, the weak learning algorithm A is able to find a hypothesis h with weighted *training* error at most $1/2 - \gamma$:

$$\Pr_{i \sim D}\left[h(x_i) \neq y_i\right] \leq \tfrac{1}{2} - \gamma.$$

Thus, empirical weak learnability is with respect to distributions defined on a particular training set with particular labels, while weak PAC learnability is with respect to any distribution on the entire domain $\mathcal{X}$, and any labeling consistent with one of the targets in the class $\mathcal{C}$. These two notions are clearly related, but are also quite distinct. Even so, when clear from context, we often use shortened terminology, such as "weak learning assumption" and "weakly learnable," omitting "PAC" or "empirical."

This condition can be weakened still further. Rather than assuming that the weak learner A can achieve a training error of $1/2 - \gamma$ on *every* distribution on the training set, we can assume only that this happens on the *particular* distributions on which it is actually trained during boosting. In the notation of Algorithm 1.1, this means, for AdaBoost, simply that $\epsilon_t \leq \frac{1}{2} - \gamma$ for all t, for some $\gamma > 0$. This property clearly follows from the empirical γ-weak learning assumption, and also follows from the weak PAC learnability assumption, as will be seen in Section 4.3. Furthermore, in comparison to what was earlier assumed regarding the weak PAC learnability of a known class $\mathcal{C}$, this assumption is quite benign, and can be immediately verified in an actual learning setting.

We can even go one step further and drop the assumption that the edge γ is known; as discussed in Chapter 1, this is the defining property of a boosting algorithm that is *adaptive*, like AdaBoost.

In the resulting, fully relaxed framework, no assumptions at all are made about the data, analogous to the agnostic approach taken earlier in the chapter. Rather, generalization performance is analyzed in terms of the weighted training errors ϵ_t, as well as other relevant parameters, such as the complexity of the weak hypotheses. Although the ϵ_t's are not explicitly assumed to be bounded below $1/2$, when this is the case, such bounds should imply high generalization accuracy. Since it is the most general and practical, we will mostly follow this mode of analysis, particularly in Chapters 4 and 5 where bounds of just this form are proved.

Summary

This chapter has reviewed some fundamental results at the foundation of theoretical machine learning, tools we will use extensively in later chapters in the analysis of boosting algorithms. These general results formalize our intuition of the requirements for learning, namely, sufficient data, low training error and simplicity, the last of these being measurable in various ways including description length, VC-dimension and degree of compressibility. This understanding also captures the essential problem of learning, that being the careful balancing of the trade-off between fit to training data and simplicity. Finally, we looked at the formal PAC learning framework and the notion of weak learnability from which arises the basic question of the existence of boosting algorithms.

With this foundation, we can begin our analysis of AdaBoost.

Bibliographic notes

The overall approach to machine learning that we have adopted in this chapter, particularly Sections 2.1 and 2.2, was initially pioneered in the ground-breaking work of Vapnik and Chervonenkis [221, 222]. For a more complete treatment, see, for instance, the books by Vapnik [223, 224, 225], and by Devroye, Györfi and Lugosi [67]. The approach and analysis described in Sections 2.2.3 and 2.2.4, including Theorems 2.3, 2.5 and 2.6, Lemma 2.4, and the VC-dimension as applied in this setting, are all due to Vapnik and Chervonenkis [221]. However, the constants that appear in their versions of these theorems are slightly different from what we have given here, which are based instead on Theorem 12.5 of Devroye, Györfi and Lugosi [67]. This latter source includes an overview of some of the other versions that have been proved, some of which have better constants. Also, Lemma 2.4 was proved independently by Sauer [197]. A short proof of Eq. (2.12) is given by Kearns and Vazirani [134]. Examples of lower bounds on learning in terms of the VC-dimension include the work of Ehrenfeucht et al. [80], and Gentile and Helmbold [107]. Hoeffding's inequality (Theorem 2.1) is due to Hoeffding [123]. Regarding Theorem 2.7, Eq. (2.17) was proved by Vapnik and Chervonenkis [222], and later by Blumer et al. [27]. Eq. (2.18) was proved by Blumer et al. [28].

Further background on the generative (or Bayesian) approach described in Section 2.1.2 can be found, for instance, in Duda, Hart and Stork [73], and Bishop [22].

The approach given in Section 2.2.6, including Theorem 2.8, is due to Littlestone and Warmuth [154], and Floyd and Warmuth [85]. The minimum description length principle, though not discussed in this book, is the foundation for another, related approach to learning and statistics which is also based on compression—see, for instance, Grünwald's book [112].

The PAC learning model of Section 2.3.1 was proposed by Valiant [220]. Further background can be found in Kearns and Vazirani's book [134]. The notion of weak PAC learning in Section 2.3.2 was first considered by Kearns and Valiant [133]. This latter work also contains examples of learning problems which are computationally intractable even when provided with a statistically adequate amount of data, as mentioned in Section 2.3.1.

Some of the exercises in this chapter are based on material from [28, 154, 220, 221].

Exercises

2-1. For any hypothesis space $\mathcal{H}$, let $\hat{h}$ minimize the training error, and let h^* minimize the generalization error:

$$\begin{aligned}\hat{h} &= \arg\min_{h\in\mathcal{H}} \widehat{\text{err}}(h)\\ h^* &= \arg\min_{h\in\mathcal{H}} \text{err}(h).\end{aligned}$$

Prove that

$$\mathrm{E}\left[\widehat{\text{err}}(\hat{h})\right] \le \text{err}(h^*) \le \mathrm{E}\left[\text{err}(\hat{h})\right]$$

(where expectation is with respect to the choice of a random training set).

2-2. Show that the VC-dimension of any finite hypothesis space $\mathcal{H}$ is at most $\lg|\mathcal{H}|$. Also, show that this bound is tight; that is, for every $d \ge 1$, give an example of a hypothesis space $\mathcal{H}$ with VC-dimension equal to d for which $d = \lg|\mathcal{H}|$.

2-3. Let $\mathcal{X} = \{0,1\}^n$, and let $\mathcal{C}$ be the space of boolean monomials, that is, functions of the form

$$c(\mathbf{x}) = \begin{cases} +1 & \text{if } \prod_{j\in R} x_j = 1 \\ -1 & \text{else} \end{cases}$$

for some $R \subseteq \{1,\ldots,n\}$. In other words, $c(\mathbf{x}) = +1$ if and only if all of the variables $x_j = 1$, for all $j \in R$. Show that $\mathcal{C}$ is efficiently PAC learnable. That is, describe an efficient (polynomial-time) algorithm for finding a monomial consistent with any dataset (assuming one exists), and show that the PAC criterion (Eq. (2.21)) holds for a sample of size polynomial in $1/\epsilon$, $1/\delta$ and n.

2-4. Let $\mathcal{X} = \mathbb{R}^n$ and let $\mathcal{H}$ be the space of hypotheses defined by *axis-aligned rectangles*, that is, functions of the form

$$h(\mathbf{x}) = \begin{cases} +1 & \text{if } a_j \le x_j \le b_j \text{ for all } j = 1,\ldots,n \\ -1 & \text{else} \end{cases}$$

for some $a_1,\ldots,a_n,b_1,\ldots,b_n \in \mathbb{R}$. Compute exactly the VC-dimension of $\mathcal{H}$.

2-5. Let $\mathcal{X} = \mathbb{R}^n$ and let $\mathcal{C}$ be the space of functions defined by axis-aligned rectangles, as in Ex. 2-4. Show that $\mathcal{C}$ is efficiently PAC learnable (as in Ex. 2-3) using a compression scheme.

2-6. Let $\mathcal{X} = \mathbb{R}$ and let $\mathcal{C}$ be the space of functions defined by unions of at most n intervals, that is, functions of the form

$$c(\mathbf{x}) = \begin{cases} +1 & \text{if } x \in [a_1, b_1] \cup \cdots \cup [a_n, b_n] \\ -1 & \text{else} \end{cases}$$

for some $a_1, \ldots, a_n, b_1, \ldots, b_n \in \mathbb{R}$.

(a) Compute the VC-dimension of $\mathcal{C}$ exactly.

(b) Use the result from part (a) to show that $\mathcal{C}$ is efficiently PAC learnable (as in Ex. 2-3).

2-7. Show that Sauer's Lemma (Lemma 2.4) is tight. That is, for every $d \geq 1$, give an example of a space $\mathcal{H}$ with VC-dimension equal to d such that for each m,

$$\Pi_{\mathcal{H}}(m) = \sum_{i=0}^{d} \binom{m}{i}.$$

2-8. Let $\mathcal{H}$ be a countably infinite space of hypotheses. Let $g : \mathcal{H} \to (0, 1]$ be any function such that

$$\sum_{h \in \mathcal{H}} g(h) \leq 1.$$

Although g may look a bit like a probability distribution, it is just a function—any function—whose positive values happen to add up to a number not bigger than 1. Assume a random training set of size m has been chosen.

(a) Prove that, with probability at least $1 - \delta$,

$$\text{err}(h) \leq \widehat{\text{err}}(h) + \sqrt{\frac{\ln(1/g(h)) + \ln(1/\delta)}{2m}}$$

for all $h \in \mathcal{H}$.

(b) Suppose hypotheses in $\mathcal{H}$ are represented by bit strings and that $|h|$ denotes the number of bits needed to represent h. Show how to choose g to prove that, with probability at least $1 - \delta$,

$$\text{err}(h) \leq \widehat{\text{err}}(h) + O\left(\sqrt{\frac{|h| + \ln(1/\delta)}{m}}\right)$$

for all $h \in \mathcal{H}$.

(c) How does the bound in part (b) reflect the intuitive trade-off between fit to data and simplicity?

2-9. Show that Theorems 2.3 and 2.6 are equivalent. That is:

(a) For $\mathcal{A}$ constructed as in Eq. (2.14), verify Eq. (2.15), and show that Theorem 2.6 yields exactly Theorem 2.3.

(b) For a general family $\mathcal{A}$ of subsets, show that Theorem 2.3 can be used to prove Theorem 2.6.

2-10. Let the domain be $\mathcal{X}_n = \mathbb{R}^n$, and let $\mathcal{H}_n$ be the space of all decision stumps of the (simplified) form

$$h(\mathbf{x}) = \begin{cases} c_0 & \text{if } x_k \leq \nu \\ c_1 & \text{if } x_k > \nu \end{cases}$$

for some $c_0, c_1 \in \{-1, +1\}$, $k \in \{1, \ldots, n\}$ and $\nu \in \mathbb{R}$. (In Section 3.4.2, we will consider decision stumps in greater generality.)

(a) Show that $\Pi_{\mathcal{H}_n}(m) \leq 2nm$.

(b) Show that there exist positive constants a and b such that for all $n \geq 1$, the VC-dimension of $\mathcal{H}_n$ is at most $a + b \ln n$.