CHAPTER 2

Binary classification and related tasks

IN THIS CHAPTER and the next we take a bird's-eye view of the wide range of different tasks that can be solved with machine learning techniques. 'Task' here refers to whatever it is that machine learning is intended to improve performance of (recall the definition of machine learning on p.3), for example, e-mail spam recognition. Since this is a classification task, we need to learn an appropriate classifier from training data. Many different types of classifiers exist: linear classifiers, Bayesian classifiers, distance-based classifiers, to name a few. We will refer to these different types as models; they are the subject of Chapters 4–9. Classification is just one of a range of possible tasks for which we can learn a model: other tasks that will pass the review in this chapter are class probability estimation and ranking. In the next chapter we will discuss regression, clustering and descriptive modelling. For each of these tasks we will discuss what it is, what variants exist, how performance at the task could be assessed, and how it relates to other tasks. We will start with some general notation that is used in this chapter and throughout the book (see Background 2.1 for the relevant mathematical concepts).

The objects of interest in machine learning are usually referred to as *instances*. The set of all possible instances is called the *instance space*, denoted $\mathscr{X}$ in this book. To illustrate, $\mathscr{X}$ could be the set of all possible e-mails that can be written using the Latin

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

alphabet.[1] We furthermore distinguish between the *label space* $\mathscr{L}$ and the *output space* $\mathscr{Y}$. The label space is used in supervised learning to label the examples. In order to achieve the task under consideration we need a *model*: a mapping from the instance space to the output space. For instance, in classification the output space is a set of classes, while in regression it is the set of real numbers. In order to learn such a model we require a *training set* Tr of *labelled instances* $(x, l(x))$, also called *examples*, where $l : \mathscr{X} \to \mathscr{L}$ is a labelling function.

Based on this terminology and notation, and concentrating on supervised learning of predictive models for the duration of the chapter, Table 2.1 distinguishes a number of specific scenarios. The most commonly encountered machine learning scenario is where the label space coincides with the output space. That is, $\mathscr{Y} = \mathscr{L}$ and we are trying to learn an approximation $\hat{l} : \mathscr{X} \to \mathscr{L}$ to the true labelling function l, which is only known through the labels it assigned to the training data. This scenario covers both classification and regression. In cases where the label space and the output space differ, this usually serves the purpose of learning a model that outputs more than just a label – for instance, a score for each possible label. In this case we have $\mathscr{Y} = \mathbb{R}^k$, with $k = |\mathscr{L}|$ the number of labels.

Matters may be complicated by *noise*, which can take the form of *label noise* – instead of $l = l(x)$ we observe some corrupted label l' – or *instance noise* – instead of x we observe an instance x' that is corrupted in some way. One consequence of noisy data is that it is generally not advisable to try to match the training data exactly, as this may lead to overfitting the noise. Some of the labelled data is usually set aside for evaluating or testing a classifier, in which case it is called a *test set* and denoted by Te. We use superscripts to restrict training or test set to a particular class: e.g., $Te^{\oplus} = \{(x, l(x)) | x \in Te, l(x) = \oplus\}$ is the set of positive test examples, and $Te^{\ominus}$ is the set of negative test examples.

The simplest kind of input space arises when instances are described by a fixed number of *features*, also called attributes, predictor variables, explanatory variables or independent variables. Indicating the set of values or *domain* of a feature by $\mathscr{F}_i$, we then have that $\mathscr{X} = \mathscr{F}_1 \times \mathscr{F}_2 \times \ldots \times \mathscr{F}_d$, and thus every instance is a d-vector of feature values. In some domains the features to use readily suggest themselves, whereas in other domains they need to be constructed. For example, in the spam filter example in the Prologue we constructed a large number of features, one for each word in a vocabulary, counting the number of occurrences of that word in the e-mail. Even when features are given explicitly we often want to transform them to maximise their usefulness for the task at hand. We will discuss this in considerable detail in Chapter 10.

[1] It is perhaps worth emphasising that an instance space like this is an unimaginably vast set (e.g., the set of all possible text messages of 160 characters using only lower-case letters, spaces and full stops is 28^{160}, a number too large for most pocket calculators), and that only a minuscule fraction of this set carries enough meaning to be possibly encountered in the real world.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

We briefly review some important concepts from discrete mathematics. A *set* is a collection of objects, usually of the same kind (e.g., the set of all natural numbers $\mathbb{N}$ or the set of real numbers $\mathbb{R}$). We write $x \in A$ if x is an element of set A, and $A \subseteq B$ if all elements of A are also elements of B (this includes the possibility that A and B are the same set, which is equivalent to $A \subseteq B$ and $B \subseteq A$). The *intersection* and *union* of two sets are defined as $A \cap B = \{x | x \in A \text{ and } x \in B\}$ and $A \cup B = \{x | x \in A \text{ or } x \in B\}$. The *difference* of two sets is defined as $A \setminus B = \{x | x \in A \text{ and } x \notin B\}$. It is customary to fix a *universe of discourse* U such that all sets under consideration are subsets of U. The *complement* of a set A is defined as $\overline{A} = U \setminus A$. Two sets are *disjoint* if their intersection is empty: $A \cap B = \emptyset$. The *cardinality* of a set A is its number of elements and is denoted $|A|$. The *powerset* of a set A is the set of all its subsets $2^A = \{B | B \subseteq A\}$; its cardinality is $|2^A| = 2^{|A|}$. The *characteristic function* of a set A is the function $f : U \to \{\mathsf{true}, \mathsf{false}\}$ such that $f(x) = \mathsf{true}$ if $x \in A$ and $f(x) = \mathsf{false}$ if $x \in U \setminus A$.

If A and B are sets, the *Cartesian product* $A \times B$ is the set of all pairs $\{(x, y) | x \in A \text{ and } y \in B\}$; this generalises to products of more than two sets. A (binary) *relation* is a set of pairs $R \subseteq A \times B$ for some sets A and B; if $A = B$ we say the relation is over A. Instead of $(x, y) \in R$ we also write xRy. A relation over A is (*i*) *reflexive* if xRx for all $x \in A$; (*ii*) *symmetric* if xRy implies yRx for all $x, y \in A$; (*iii*) *antisymmetric* if xRy and yRx implies $x = y$ for all $x, y \in A$; (*iv*) *transitive* if xRy and yRz implies xRz for all $x, y, z \in A$. (*v*) *total* if xRy or yRx for all $x, y \in A$.

A *partial order* is a binary relation that is reflexive, antisymmetric and transitive. For instance, the *subset* relation $\subseteq$ is a partial order. A *total order* is a binary relation that is total (hence reflexive), antisymmetric and transitive. The $\leq$ relation on real numbers is a total order. If xRy or yRx we say that x and y are *comparable*; otherwise they are *incomparable*. An *equivalence relation* is a binary relation $\equiv$ that is reflexive, symmetric and transitive. The *equivalence class* of x is $[x] = \{y | x \equiv y\}$. For example, the binary relation 'contains the same number of elements as' over any set is an equivalence relation. Any two equivalence classes are disjoint, and the union of all equivalence classes is the whole set – in other words, the set of all equivalence classes forms a *partition* of the set. If $A_1, \ldots, A_n$ is a partition of a set A, i.e. $A_1 \cup \ldots \cup A_n = A$ and $A_i \cap A_j = \emptyset$ for all $i \neq j$, we write $A = A_1 \uplus \ldots \uplus A_n$.

To illustrate this, let T be a feature tree, and define a relation $\sim_T \subseteq \mathcal{X} \times \mathcal{X}$ such that $x \sim_T x'$ if and only if x and x' are assigned to the same leaf of feature tree T, then $\sim_T$ is an equivalence relation, and its equivalence classes are precisely the instance space segments associated with T.

Background 2.1. Useful concepts from discrete mathematics.

The sections in this chapter are devoted to the first three scenarios in Table 2.1:

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

Task	*Label space*	*Output space*	*Learning problem*
Classification	$\mathscr{L} = \mathscr{C}$	$\mathscr{Y} = \mathscr{C}$	learn an approximation $\hat{c} : \mathscr{X} \to \mathscr{C}$ to the true labelling function c
Scoring and ranking	$\mathscr{L} = \mathscr{C}$	$\mathscr{Y} = \mathbb{R}^{\lvert\mathscr{C}\rvert}$	learn a model that outputs a score vector over classes
Probability estimation	$\mathscr{L} = \mathscr{C}$	$\mathscr{Y} = [0,1]^{\lvert\mathscr{C}\rvert}$	learn a model that outputs a probability vector over classes
Regression	$\mathscr{L} = \mathbb{R}$	$\mathscr{Y} = \mathbb{R}$	learn an approximation $\hat{f} : \mathscr{X} \to \mathbb{R}$ to the true labelling function f

Table 2.1. Predictive machine learning scenarios.

classification in Section 2.1, scoring and ranking in Section 2.2 and class probability estimation in Section 2.3. To keep things manageable we mostly restrict attention to two-class tasks in this chapter and deal with more than two classes in Chapter 3. Regression, unsupervised and descriptive learning will also be considered there.

Throughout this chapter I will illustrate key concepts by means of examples using simple models of the kind discussed in the Prologue. These models will either be simple tree-based models, representative of grouping models, or linear models, representative of grading models. Sometimes we will even construct models from single features, a setting that could be described as *univariate machine learning*. We will start dealing with the question of how to *learn* such models from Chapter 4 onwards.

2.1 Classification

Classification is the most common task in machine learning. A *classifier* is a mapping $\hat{c} : \mathscr{X} \to \mathscr{C}$, where $\mathscr{C} = \{C_1, C_2, \ldots, C_k\}$ is a finite and usually small set of *class labels*. We will sometimes also use C_i to indicate the set of examples of that class. We use the 'hat' to indicate that $\hat{c}(x)$ is an estimate of the true but unknown function $c(x)$. Examples for a classifier take the form $(x, c(x))$, where $x \in \mathscr{X}$ is an instance and $c(x)$ is the true class of the instance. Learning a classifier involves constructing the function $\hat{c}$ such that it matches c as closely as possible (and not just on the training set, but ideally on the entire instance space $\mathscr{X}$).

In the simplest case we have only two classes which are usually referred to as *positive* and *negative*, ⊕and ⊖, or +1 and −1. Two-class classification is often called *binary classification* (or *concept learning*, if the positive class can be meaningfully called

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

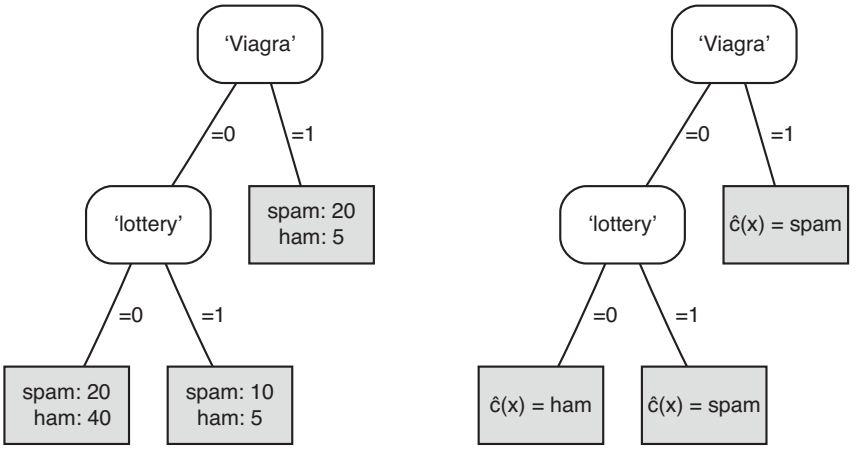

Figure 2.1. (left) A feature tree with training set class distribution in the leaves. **(right)** A decision tree obtained using the majority class decision rule.

a concept). Spam e-mail filtering is a good example of binary classification, in which spam is conventionally taken as the positive class, and ham as the negative class (clearly, positive here doesn't mean 'good'!). Other examples of binary classification include medical diagnosis (the positive class here is having a particular disease) and credit card fraud detection.

The feature tree in Figure 2.1 (left) can be turned into a classifier by labelling each leaf with a class. The simplest way to do this is by assigning the *majority class* in each leaf, resulting in the decision tree in Figure 2.1 (right). The classifier works as follows: if an e-mail contains the word 'Viagra' it is classified as spam (right-most leaf); otherwise, the occurrence of the word 'lottery' decides whether it gets labelled spam or ham.[2] From the numbers in Figure 2.1 we can get an idea how well this classifier does. The left-most leaf correctly predicts 40 ham e-mails but also mislabels 20 spam e-mails that contain neither 'Viagra' nor 'lottery'. The middle leaf correctly classifies 10 spam e-mails but also erroneously labels 5 ham e-mails as spam. The 'Viagra' test correctly picks out 20 spam e-mails but also 5 ham e-mails. Taken together, this means that 30 out of 50 spam e-mails are classified correctly, and 40 out of 50 ham e-mails.

Assessing classification performance

The performance of such classifiers can be summarised by means of a table known as a *contingency table* or *confusion matrix* (Table 2.2 (left)). In this table, each row refers to

[2] If you are keen to know how such a decision tree can be learned from data, you may want to take a sneak preview at Algorithm 5.1 on p.132.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

actual classes as recorded in the test set, and each column to classes as predicted by the classifier. So, for instance, the first row states that the test set contains 50 positives, 30 of which were correctly predicted and 20 incorrectly. The last column and the last row give the *marginals* (i.e., column and row sums). Marginals are important because they allow us to assess statistical significance. For instance, the contingency table in Table 2.2 (right) has the same marginals, but the classifier clearly makes a random choice as to which predictions are positive and which are negative – as a result the distribution of actual positives and negatives in either predicted class is the same as the overall distribution (uniform in this case).

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**30**	**20**	50
Actual ⊖	**10**	**40**	50
	40	60	100

	⊕	⊖	
⊕	**20**	**30**	50
⊖	**20**	**30**	50
	40	60	100

Table 2.2. (left) A two-class contingency table or confusion matrix depicting the performance of the decision tree in Figure 2.1. Numbers on the descending diagonal indicate correct predictions, while the ascending diagonal concerns prediction errors. **(right)** A contingency table with the same marginals but independent rows and columns.

From a contingency table we can calculate a range of performance indicators. The simplest of these is *accuracy*, which is the proportion of correctly classified test instances. In the notation introduced at the beginning of this chapter, accuracy over a test set Te is defined as

$$acc = \frac{1}{|Te|} \sum_{x \in Te} I[\hat{c}(x) = c(x)] \tag{2.1}$$

Here, the function $I[\cdot]$ denotes the *indicator function*, which is 1 if its argument evaluates to true, and 0 otherwise. In this case it is a convenient way to count the number of test instances that are classified correctly by the classifier (i.e., the estimated class label $\hat{c}(x)$ is equal to the true class label $c(x)$). For example, in Table 2.2 (left) the accuracy of the classifier is 0.70 or 70%, and in Table 2.2 (right) it is 0.50. Alternatively, we can calculate the *error rate* as the proportion of incorrectly classified instances, here 0.30 and 0.50, respectively. Clearly, accuracy and error rate sum to 1.

Test set accuracy can be seen as an *estimate* of the probability that an arbitrary instance $x \in \mathscr{X}$ is classified correctly: more precisely, it estimates the probability

$$P_{\mathscr{X}}(\hat{c}(x) = c(x))$$

(Notice that I write $P_{\mathscr{X}}$ to emphasise that this is a probability distribution over the instance space $\mathscr{X}$; I will often omit subscripts if this is clear from the context.) We

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

typically only have access to the true classes of a small fraction of the instance space and so an estimate is all we can hope to get. It is therefore important that the test set is as representative as possible. This is usually formalised by the assumption that the occurrence of instances in the world – i.e., how likely or typical a particular e-mail is – is governed by an unknown probability distribution on $\mathscr{X}$, and that the test set *Te* is generated according to this distribution.

It is often convenient – not to say necessary – to distinguish performance on the classes. To this end, we need some further terminology. Correctly classified positives and negatives are referred to as *true positives* and *true negatives*, respectively. Incorrectly classified positives are, perhaps somewhat confusingly, called *false negatives*; similarly, misclassified negatives are called *false positives*. A good way to think of this is to remember that positive/negative refers to the classifier's prediction, and true/false refers to whether the prediction is correct or not. So, a false positive is something that was incorrectly predicted as positive, and therefore an actual negative (e.g., a ham e-mail misclassified as spam, or a healthy patient misclassified as having the disease in question). In the previous example (Table 2.2 (left)) we have 30 true positives, 20 false negatives, 40 true negatives and 10 false positives.

The *true positive rate* is the proportion of positives correctly classified, and can be defined mathematically as

$$tpr = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]}{\sum_{x \in Te} I[c(x) = \oplus]} \tag{2.2}$$

True positive rate is an estimate of the probability that an arbitrary positive is classified correctly, that is, an estimate of $P_{\mathscr{X}}(\hat{c}(x) = \oplus | c(x) = \oplus)$. Analogously, the *true negative rate* is the proportion of negatives correctly classified (see Table 2.3 on p.57 for the mathematical definition), and estimates $P_{\mathscr{X}}(\hat{c}(x) = \ominus | c(x) = \ominus)$. These rates, which are sometimes called *sensitivity* and *specificity*, can be seen as per-class accuracies. In the contingency table, the true positive and negative rates can be calculated by dividing the number on the descending (good) diagonal by the row total. We can also talk about per-class error rates, which is the *false negative rate* for the positives (i.e., the number of misclassified positives or false negatives as a proportion of the total number of positives) and the *false positive rate* for the negatives (sometimes called the *false alarm rate*). These rates can be found by dividing the number on the ascending (bad) diagonal by the row total.

In Table 2.2 (left) we have a true positive rate of 60%, a true negative rate of 80%, a false negative rate of 40% and a false positive rate of 20%. In Table 2.2 (right) we have a true positive rate of 40%, a true negative rate of 60%, a false negative rate of 60% and a false positive rate of 40%. Notice that the accuracy in both cases is the average of the true positive rate and the true negative rate (and the error rate is the average of the false positive rate and the false negative rate). However, this is true only if the test set

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

contains equal numbers of positives and negatives – in the general case we need to use a *weighted* average, where the weights are the proportions of positives and negatives in the test set.

Example 2.1 (Accuracy as a weighted average). Suppose a classifier's predictions on a test set are as in the following table:

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**60**	**15**	75
Actual ⊖	**10**	**15**	25
	70	30	100

From this table, we see that the true positive rate is $tpr = 60/75 = 0.80$ and the true negative rate is $tnr = 15/25 = 0.60$. The overall accuracy is $acc = (60 + 15)/100 = 0.75$, which is no longer the average of true positive and negative rates. However, taking into account the proportion of positives $pos = 0.75$ and the proportion of negatives $neg = 1 - pos = 0.25$, we see that

$$acc = pos \cdot tpr + neg \cdot tnr \tag{2.3}$$

This equation holds in general: if the numbers of positives and negatives are equal, we obtain the unweighted average from the earlier example ($acc = (tpr + tnr)/2$).

Equation 2.3 has a neat intuition: good performance on either class contributes to good classification accuracy, but the more prevalent class contributes more strongly. In order to achieve good accuracy, a classifier should concentrate on the *majority class*, particularly if the class distribution is highly unbalanced. However, it is often the case that the majority class is also the least interesting class. To illustrate, suppose you issue a query to an internet search engine,[3] and suppose that for that particular query there is only one relevant page in every 1 000 web pages. Now consider a 'reluctant' search engine that doesn't return *any* answers – i.e., it classifies every web page as irrelevant to your query. Consequently, it will achieve 0% true positive rate and 100% true negative rate. Because $pos = 1/1000 = 0.1\%$ and $neg = 99.9\%$, the reluctant search engine's accuracy is very high (99.9%). Put differently, if we select a random web page uniformly

[3] An internet search engine can be seen as a binary classifier into the classes relevant and irrelevant, or interesting and not interesting, if we fix the query – not very realistic in practice, but a useful analogy for our purposes.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

Measure	*Definition*	*Equal to*	*Estimates*
number of positives	$Pos = \sum_{x \in Te} I[c(x) = \oplus]$		
number of negatives	$Neg = \sum_{x \in Te} I[c(x) = \ominus]$	$\lvert Te \rvert - Pos$	
number of true positives	$TP = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]$		
number of true negatives	$TN = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \ominus]$		
number of false positives	$FP = \sum_{x \in Te} I[\hat{c}(x) = \oplus, c(x) = \ominus]$	$Neg - TN$	
number of false negatives	$FN = \sum_{x \in Te} I[\hat{c}(x) = \ominus, c(x) = \oplus]$	$Pos - TP$	
proportion of positives	$pos = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[c(x) = \oplus]$	$Pos / \lvert Te \rvert$	$P(c(x) = \oplus)$
proportion of negatives	$neg = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[c(x) = \ominus]$	$1 - pos$	$P(c(x) = \ominus)$
class ratio	$clr = pos / neg$	Pos / Neg	
(*) accuracy	$acc = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[\hat{c}(x) = c(x)]$		$P(\hat{c}(x) = c(x))$
(*) error rate	$err = \frac{1}{\lvert Te \rvert} \sum_{x \in Te} I[\hat{c}(x) \neq c(x)]$	$1 - acc$	$P(\hat{c}(x) \neq c(x))$
true positive rate, sensitivity, recall	$tpr = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]}{\sum_{x \in Te} I[c(x) = \oplus]}$	TP / Pos	$P(\hat{c}(x) = \oplus \lvert c(x) = \oplus)$
true negative rate, specificity, negative recall	$tnr = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = \ominus]}{\sum_{x \in Te} I[c(x) = \ominus]}$	TN / Neg	$P(\hat{c}(x) = \ominus \lvert c(x) = \ominus)$
false positive rate, false alarm rate	$fpr = \frac{\sum_{x \in Te} I[\hat{c}(x) = \oplus, c(x) = \ominus]}{\sum_{x \in Te} I[c(x) = \ominus]}$	$FP / Neg = 1 - tnr$	$P(\hat{c}(x) = \oplus \lvert c(x) = \ominus)$
false negative rate	$fnr = \frac{\sum_{x \in Te} I[\hat{c}(x) = \ominus, c(x) = \oplus]}{\sum_{x \in Te} I[c(x) = \oplus]}$	$FN / Pos = 1 - tpr$	$P(\hat{c}(x) = \ominus \lvert c(x) = \oplus)$
precision, confidence	$prec = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]}{\sum_{x \in Te} I[\hat{c}(x) = \oplus]}$	$TP / (TP + FP)$	$P(c(x) = \oplus \lvert \hat{c}(x) = \oplus)$

Table 2.3. A summary of different quantities and evaluation measures for classifiers on a test set *Te*. Symbols starting with a capital letter denote absolute frequencies (counts), while lower-case symbols denote relative frequencies or ratios. All except those indicated with (*) are defined only for binary classification. The right-most column specifies the instance space probabilities that these relative frequencies are estimating.

over all web pages, the probability of selecting a positive is only 0.001, and these are the only pages on which the reluctant engine makes an error. However, we are not normally selecting pages from the web uniformly, and hence accuracy is not a meaningful quantity in this context. To be of any use at all, a search engine should achieve a much better true positive rate, which usually comes at the expense of a worse true negative rate (and hence a drop in accuracy).

We conclude from this example that, if the minority class is the class of interest and very small, accuracy and performance on the majority class are not the right quantities to optimise. For this reason, an alternative to true negative rate called *precision* is usually considered in such cases. Precision is a counterpart to true positive rate in the following sense: while true positive rate is the proportion of predicted positives among the actual positives, precision is the proportion of actual positives among the predicted positives. In Example 2.1 the classifier's precision on the test set is 60/70 = 85.7%. In

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

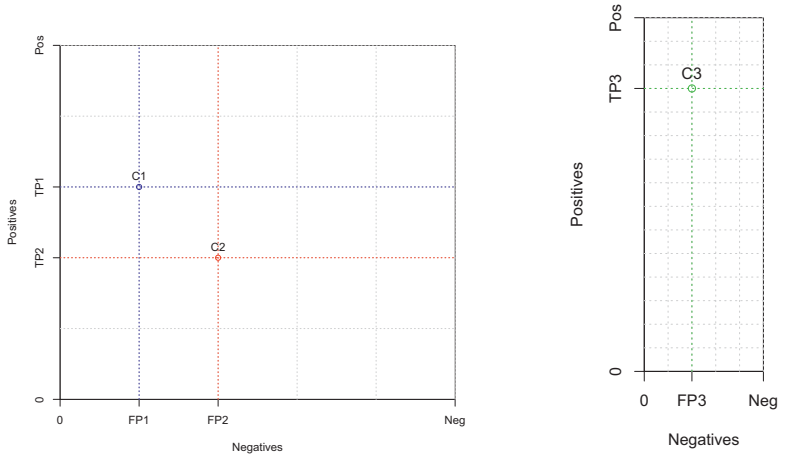

Figure 2.2. (left) A coverage plot depicting the two contingency tables in Table 2.2. The plot is square because the class distribution is uniform. **(right)** Coverage plot for Example 2.1, with a class ratio $clr = 3$.

the reluctant search engine example we have not only 0 true positive rate (which in this context is usually called *recall*) but also 0 precision, which clearly demonstrates the problem with a search engine that doesn't return any answers. Table 2.3 summarises the evaluation measures introduced in this section.

Visualising classification performance

I will now introduce an important tool for visualising the performance of classifiers and other models called a *coverage plot*. If you look at two-class contingency tables such as the ones depicted in Table 2.2, you realise that, even though the table contains nine numbers, only four of those can be chosen freely. For instance, once you've determined the true/false positives/negatives, the marginals are fixed. Or if you know the true positives, true negatives, total number of positives and size of the test set, you can reconstruct all other numbers. Statisticians say that the table has four *degrees of freedom*.[4]

Often we are particularly interested in the following four numbers that completely determine the contingency table: the number of positives *Pos*, the number of negatives *Neg*, the number of true positives *TP* and the number of false positives *FP*. A coverage plot visualises these four numbers by means of a rectangular coordinate system and a point. Imagine a rectangle with height *Pos* and width *Neg*. Imagine furthermore that all positives live on the y-axis of this rectangle, and all negatives on the x-axis. We don't

[4] More generally, a k-class contingency table has $(k+1)^2$ entries and k^2 degrees of freedom.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

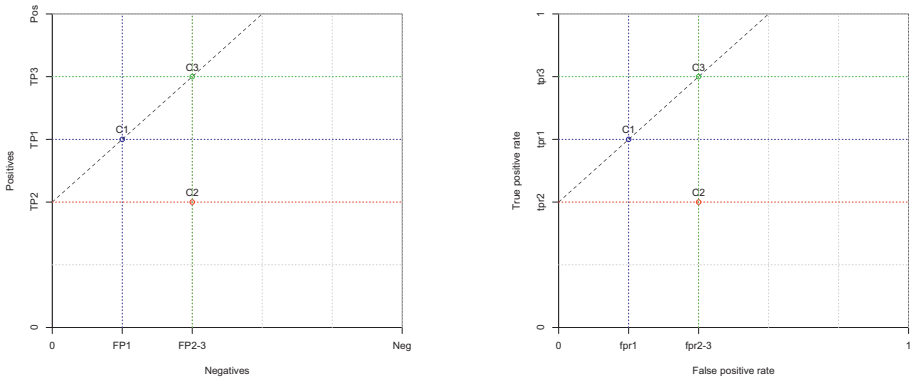

Figure 2.3. (left) C1 and C3 both dominate C2, but neither dominates the other. The diagonal line indicates that C1 and C3 achieve equal accuracy. **(right)** The same plot with normalised axes. We can interpret this plot as a merger of the two coverage plots in Figure 2.2, employing normalisation to deal with the different class distributions. The diagonal line now indicates that C1 and C3 have the same average recall.

really care how positives and negatives are ordered on their respective axes, as long as *positive predictions come before negative predictions*. This gives us enough information to depict the whole contingency table as a single point within the rectangle (Figure 2.2).

Consider the two classifiers marked C1 and C2 in Figure 2.2 (left). One reason why coverage plots are so useful is that we can immediately see that C1 is better than C2. How do we know that? Well, C1 has both more true positives and fewer false positives than C2, and so is better in both respects. Put differently, C1 achieves better performance than C2 on *both* classes. If one classifier outperforms another classifier on all classes, the first one is said to *dominate* the second.[5] However, things are not always that straightforward. Consider a third classifier C3, better than C1 on the positives but worse on the negatives (Figure 2.3 (left)). Although both C1 and C3 dominate C2, neither of them dominates the other. Which one we prefer depends on whether we put more emphasis on the positives or on the negatives.

We can make this a little bit more precise. Notice that the line segment connecting C1 and C3 has a slope of 1. Imagine travelling up that line: whenever we gain a true positive, we also lose a true negative (or gain a false positive, which is the same thing). This doesn't affect the sum of true positives and true negatives, and hence the accuracy is the same wherever we are on the line. It follows that C1 and C3 have the same accuracy. *In a coverage plot, classifiers with the same accuracy are connected by line segments with slope 1*. If true positives and true negatives are equally important, the

[5] This terminology comes from the field of *multi-criterion optimisation*. A dominated solution is one that is not on the *Pareto front*.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

choice between C1 and C3 is arbitrary; if true positives are more important we should choose C3, if true negatives are more important we prefer C1.

Now consider Figure 2.3 (right). What I have done here is renormalise the axes by dividing the x-axis by *Neg* and the y-axis by *Pos*, resulting in a plot in the unit square with true positive rate on the y-axis and false positive rate on the x-axis. In this case the original coverage plot was already square ($Pos = Neg$), so the relative position of the classifiers isn't affected by the normalisation. However, since the normalised plot will be square regardless of the shape of the original plot, normalisation is a way to combine differently shaped coverage plots, and thus to combine results on test sets with different class distributions. Suppose you would normalise Figure 2.2 (right): since C3's true and false positive rates are 80% and 40%, respectively (see Example 2.1 on p.56), its position in a normalised plot is exactly the same as the one labelled C3 in Figure 2.3 (right)! In other words, classifiers occupying different points in different coverage spaces (e.g., C3 in Figure 2.2 (right) and C3 in Figure 2.3 (left)) can end up in the same point in a normalised plot.

What is the meaning of the diagonal line connecting C1 and C3 in Figure 2.3 (right)? It can't have the same meaning as in the coverage plot, because in a normalised plot we know the true and false positive rates but not the class distribution, and so we cannot calculate accuracy (refer back to Equation 2.3 on p.56 if you want to remind yourself why). The line is defined by the equation $tpr = fpr + y_0$, where y_0 is the y-intercept (the value of tpr where the line intersects the y-axis) . Now consider the average of the true positive rate and the true negative rate, which we will call *average recall*, denoted *avg-rec*.[6] On a line with slope 1 we have $avg\text{-}rec = (tpr + tnr)/2 = (tpr + 1 - fpr)/2 = (1 + y_0)/2$, which is a constant. *In a normalised coverage plot, line segments with slope 1 connect classifiers with the same average recall.* If recall on the positives and the negatives are equally important, the choice between C1 and C3 is arbitrary; if positive recall is more important we should choose C3, if negative recall is more important we prefer C1.

In the literature, normalised coverage plots are referred to as *ROC plots*, and we will follow that convention from now on.[7] ROC plots are much more common than coverage plots, but both have their specific uses. Broadly speaking, you should use a coverage plot if you explicitly want to take the class distribution into account, for instance when you are working with a single data set. An ROC plot is useful if you want to combine results from different data sets with different class distributions. Clearly, there are many connections between the two. Since an ROC plot is always square, lines of constant average recall (so-called average recall *isometrics*) do not only have

[6] Remember that recall is just a different name for true positive rate; negative recall is then the same as the true negative rate, and average recall is the average of positive recall (or true positive rate) and negative recall (or true negative rate). It is sometimes called *macro-averaged accuracy*.

[7] ROC stands for *receiver operating characteristic*, a term originating from *signal detection theory*.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

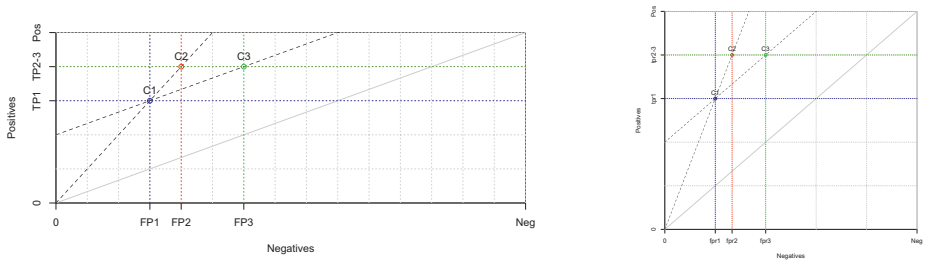

Figure 2.4. (left) In a coverage plot, accuracy isometrics have a slope of 1, and average recall isometrics are parallel to the ascending diagonal. **(right)** In the corresponding ROC plot, average recall isometrics have a slope of 1; the accuracy isometric here has a slope of 3, corresponding to the ratio of negatives to positives in the data set.

a slope of 1 but are parallel to the ascending diagonal. The latter property carries over to coverage plots. To illustrate, in the coverage plot in Figure 2.4, C1 and C2 have the same accuracy (they are connected by a line segment with slope 1), and C1 and C3 have the same average recall (they are connected by a line segment parallel to the diagonal). You can also argue that C2 has both higher accuracy and higher average recall than C3 (why?). In the corresponding ROC plot, the average recall isometric has a slope of 1, and the accuracy isometric's slope is $Neg/Pos = 1/clr$.

2.2 Scoring and ranking

Many classifiers compute scores on which their class predictions are based. For instance, in the Prologue we saw how SpamAssassin calculates a weighted sum from the rules that 'fire' for a particular e-mail. Such scores contain additional information that can be beneficial in a number of ways, which is why we perceive scoring as a task in its own right. Formally, a *scoring classifier* is a mapping $\hat{\mathbf{s}} : \mathcal{X} \to \mathbb{R}^k$, i.e., a mapping from the instance space to a k-vector of real numbers. The boldface notation indicates that a scoring classifier outputs a vector $\hat{\mathbf{s}}(x) = (\hat{s}_1(x), \ldots, \hat{s}_k(x))$ rather than a single number; $\hat{s}_i(x)$ is the score assigned to class C_i for instance x. This score indicates how likely it is that class label C_i applies. If we only have two classes, it usually suffices to consider the score for only one of the classes; in that case, we use $\hat{s}(x)$ to denote the score of the positive class for instance x.

Figure 2.5 demonstrates how a feature tree can be turned into a scoring tree. In order to obtain a score for each leaf, we first calculate the ratio of spam to ham, which is 1/2 for the left leaf, 2 for the middle leaf and 4 for the right leaf. Because it is often more convenient to work with an additive scale, we obtain scores by taking the logarithm of the class ratio (the base of the logarithm is not really important; here we have

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

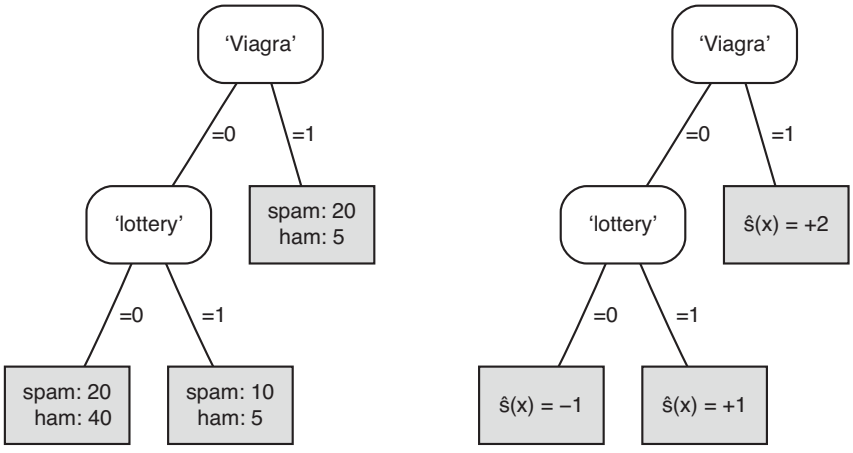

Figure 2.5. (left) A feature tree with training set class distribution in the leaves. **(right)** A scoring tree using the logarithm of the class ratio as scores; spam is taken as the positive class.

taken base-2 logarithms to get nice round numbers). Notice that the majority class decision tree corresponds to thresholding $\hat{s}(x)$ at 0: i.e., predict spam if $\hat{s}(x) > 0$ and ham otherwise.

If we take the true class $c(x)$ as $+1$ for positive examples and -1 for negative examples, then the quantity $z(x) = c(x)\hat{s}(x)$ is positive for correct predictions and negative for incorrect predictions: this quantity is called the *margin* assigned by the scoring classifier to the example.[8] We would like to reward large positive margins, and penalise large negative values. This is achieved by means of a so-called *loss function* $L : \mathbb{R} \mapsto [0, \infty)$ which maps each example's margin $z(x)$ to an associated loss $L(z(x))$. We will assume that $L(0) = 1$, which is the loss incurred by having an example on the decision boundary. We furthermore have $L(z) \geq 1$ for $z < 0$, and usually also $0 \leq L(z) < 1$ for $z > 0$ (Figure 2.6). The average loss over a test set Te is $\frac{1}{|Te|} \sum_{x \in Te} L(z(x))$.

The simplest loss function is *0–1 loss*, which is defined as $L_{01}(z) = 1$ if $z \leq 0$ and $L(z) = 0$ if $z > 0$. The average 0–1 loss is simply the proportion of misclassified test examples:

$$\frac{1}{|Te|} \sum_{x \in Te} L_{01}(z(x)) = \frac{1}{|Te|} \sum_{x \in Te} I[c(x)\hat{s}(x) \leq 0] = \frac{1}{|Te|} \sum_{x \in Te} I[c(x) \neq \hat{c}(x)] = err$$

where $\hat{c}(x) = +1$ if $\hat{s}(x) > 0$, $\hat{c}(x) = 0$ if $\hat{s}(x) = 0$, and $\hat{c}(x) = -1$ if $\hat{s}(x) < 0$. (It is sometimes more convenient to define the loss of examples on the decision boundary as $1/2$). In other words, 0–1 loss ignores the magnitude of the margins of the examples, only

[8] Remember that in Chapter 1 we talked about the margin of a classifier as the distance between the decision boundary and the nearest example. Here we use margin in a slightly more general sense: each example has a margin, not just the nearest one. This will be further explained in Section 7.3.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

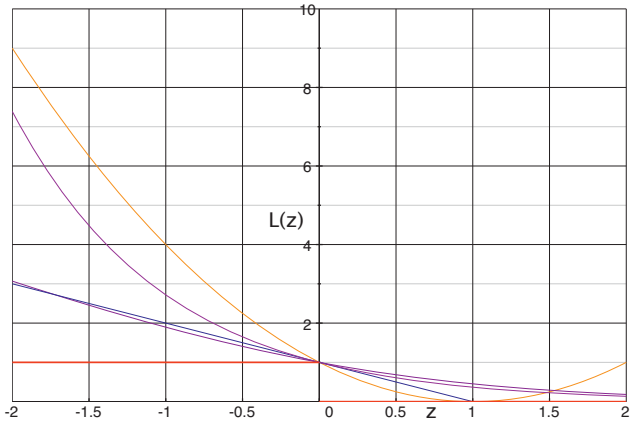

Figure 2.6. Loss functions: from bottom-left *(i)* 0–1 loss $L_{01}(z) = 1$ if $z \leq 0$, and $L_{01}(z) = 0$ if $z > 0$; *(ii)* hinge loss $L_{\mathrm{h}}(z) = (1 - z)$ if $z \leq 1$, and $L_{\mathrm{h}}(z) = 0$ if $z > 1$; *(iii)* logistic loss $L_{\log}(z) = \log_2(1 + \exp(-z))$; *(iv)* exponential loss $L_{\exp}(z) = \exp(-z)$; *(v)* squared loss $L_{\mathrm{sq}}(z) = (1 - z)^2$ (this can be set to 0 for $z > 1$, just like hinge loss).

taking their sign into account. As a result, 0–1 loss doesn't distinguish between scoring classifiers, as long as their predictions agree. This means that it isn't actually that useful as a search heuristic or objective function when learning scoring classifiers. Figure 2.6 pictures several loss functions that are used in practice. Except for 0–1 loss, they are all *convex*: linear interpolation between any two points on the curve will never result in a point below the curve. Optimising a convex function is computationally more tractable.

One loss function that will be of interest later is the *hinge loss*, which is defined as $L_{\mathrm{h}}(z) = (1 - z)$ if $z \leq 1$, and $L_{\mathrm{h}}(z) = 0$ if $z > 1$. The name of this loss function comes from the fact that the loss 'hinges' on whether an example's margin is greater than 1 or not: if so (i.e., the example is on the correct side of the decision boundary with a distance of at least 1) the example incurs zero loss; if not, the loss increases with decreasing margin. In effect, the loss function expresses that it is important to avoid examples having a margin (much) less than 1, but no additional value is placed on achieving large positive margins. This loss function is used when training a ☞*support vector machine* (Section 7.3). We will also encounter *exponential loss* later when we discuss ☞*boosting* in Section 11.2.

Assessing and visualising ranking performance

It should be kept in mind that scores are assigned by a classifier, and are not a property inherent to instances. Scores are not estimated from 'true scores' – rather, a scoring classifier has to be learned from examples in the form of instances x labelled with

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

classes $c(x)$, just as a classifier. (The task where we learn a function $\hat{f}$ from examples labelled with true function values $(x, f(x))$ is called ☞*regression* and is covered in Section 3.2.) Often it is more convenient to keep the order imposed by scores on a set of instances, but ignore their magnitudes – this has the advantage, for instance, of being much less sensitive to outliers. It also means that we do not have to make any assumptions about the scale on which scores are expressed: in particular, a ranker does not assume a particular score threshold for separating positives from negatives. A *ranking* is defined as a total order on a set of instances, possibly with ties.[9]

Example 2.2 (Ranking example). The scoring tree in Figure 2.5 produces the following ranking: [20+,5−][10+,5−][20+,40−]. Here, 20+ denotes a sequence of 20 positive examples, and instances in square brackets [...] are tied. By selecting a split point in the ranking we can turn the ranking into a classification. In this case there are four possibilities: (A) setting the split point before the first segment, and thus assigning all segments to the negative class; (B) assigning the first segment to the positive class, and the other two to the negative class; (C) assigning the first two segments to the positive class; and (D) assigning all segments to the positive class. In terms of actual scores, this corresponds to (A) choosing any score larger than 2 as the threshold; (B) choosing a threshold between 1 and 2; (C) setting the threshold between −1 and 1; and (D) setting it lower than −1.

Suppose x and x' are two instances such that x receives a lower score: $\hat{s}(x) < \hat{s}(x')$. Since higher scores express a stronger belief that the instance in question is positive, this would be fine except in one case: if x is an actual positive and x' is an actual negative. We will call this a *ranking error*. The total number of ranking errors can then be expressed as $\sum_{x \in Te^{\oplus}, x' \in Te^{\ominus}} I[\hat{s}(x) < \hat{s}(x')]$. Furthermore, for every positive and negative that receive the same score – a *tie* – we count half a ranking error. The maximum number of ranking errors is equal to $|Te^{\oplus}| \cdot |Te^{\ominus}| = Pos \cdot Neg$, and so the *ranking error rate* is defined as

$$rank\text{-}err = \frac{\sum_{x \in Te^{\oplus}, x' \in Te^{\ominus}} I[\hat{s}(x) < \hat{s}(x')] + \frac{1}{2} I[\hat{s}(x) = \hat{s}(x')]}{Pos \cdot Neg} \tag{2.4}$$

and analogously the *ranking accuracy*

$$rank\text{-}acc = \frac{\sum_{x \in Te^{\oplus}, x' \in Te^{\ominus}} I[\hat{s}(x) > \hat{s}(x')] + \frac{1}{2} I[\hat{s}(x) = \hat{s}(x')]}{Pos \cdot Neg} = 1 - rank\text{-}err \tag{2.5}$$

[9] A total order with ties should not be confused with a partial order (see Background 2.1 on p.51). In a total order with ties (which is really a total order on equivalence classes), any two elements are comparable, either in one direction or in both. In a partial order some elements are incomparable.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

Ranking accuracy can be seen as an estimate of the probability that an arbitrary positive–negative pair is ranked correctly.

Example 2.3 (Ranking accuracy). We continue the previous example considering the scoring tree in Figure 2.5, with the left leaf covering 20 spam and 40 ham, the middle leaf 10 spam and 5 ham, and the right leaf 20 spam and 5 ham. The 5 negatives in the right leaf are scored higher than the 10 positives in the middle leaf and the 20 positives in the left leaf, resulting in $50 + 100 = 150$ ranking errors. The 5 negatives in the middle leaf are scored higher than the 20 positives in the left leaf, giving a further 100 ranking errors. In addition, the left leaf makes 800 half ranking errors (because 20 positives and 40 negatives get the same score), the middle leaf 50 and the right leaf 100. In total we have 725 ranking errors out of a possible $50 \cdot 50 = 2500$, corresponding to a ranking error rate of 29% or a ranking accuracy of 71%.

The coverage plots and ROC plots introduced in the previous section for visualising classifier performance provide an excellent tool for visualising ranking performance too. If *Pos* positives and *Neg* negatives are plotted on the vertical and horizontal axes, respectively, then each positive–negative pair occupies a unique 'cell' in this plot. If we order the positives and negatives on decreasing score, i.e., examples with higher scores are closer to the origin, then we can clearly distinguish the correctly ranked pairs at the bottom right, the ranking errors at the top left, and the ties in between (Figure 2.7). The number of cells in each area gives us the number of correctly ranked pairs, ranking errors and ties, respectively. The diagonal lines cut the ties area in half, so the area below those lines corresponds to the ranking accuracy multiplied by $Pos \cdot Neg$, and the area above corresponds to the ranking error rate times that same factor.

Concentrating on those diagonal lines gives us the piecewise linear curve shown in Figure 2.7 (right). This curve, which we will call a *coverage curve*, can be understood as follows. Each of the points marked A, B, C and D specifies the classification performance, in terms of true and false positives, achieved by the corresponding ranking split points or score thresholds from Example 2.2. To illustrate, C would be obtained by a score threshold of 0, leading to $TP2 = 20 + 10 = 30$ true positives and $FP2 = 5 + 5 = 10$ false positives. Similarly, B would be obtained by a higher threshold of 1.5, leading to $TP1 = 20$ true positives and $FP1 = 5$ false positives. Point A would result if we set the threshold unattainably high, and D if we set the threshold trivially low.

Why are these points connected by straight lines? How can we interpolate between, say, points C and D? Suppose we set the threshold exactly at -1, which is the score

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

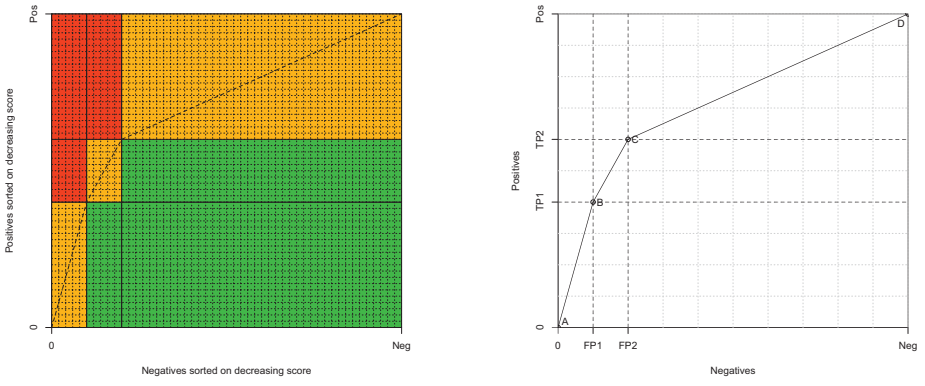

Figure 2.7. (left) Each cell in the grid denotes a unique pair of one positive and one negative example: the green cells indicate pairs that are correctly ranked by the classifier, the red cells represent ranking errors, and the orange cells are half-errors due to ties. **(right)** The coverage curve of a tree-based scoring classifier has one line segment for each leaf of the tree, and one (*FP*, *TP*) pair for each possible threshold on the score.

assigned by the left leaf of the tree. The question is now what class we predict for the 20 positives and 40 negatives that filter down to that leaf. It would seem reasonable to decide this by tossing a fair coin, leading to half of the positives receiving a positive prediction (on average) and half of them a negative one, and similar for the negatives. The total number of true positives is then $30 + 20/2 = 40$, and the number of false positives is $10 + 40/2 = 30$. In other words, we land exactly in the middle of the CD line segment. We can apply the same procedure to achieve performance half-way BC, by setting the threshold at 1 and tossing the same fair coin to obtain uniformly distributed predictions for the 10 positives and 5 negatives in the middle leaf, leading to $20 + 10/2 = 25$ true positives and $5 + 5/2 = 7.5$ false positives (of course, we cannot achieve a non-integer number of false positives in any trial, but this number represents the expected number of false positives over many trials). And what's more, by biasing the coin towards positive or negative predictions we can achieve expected performance anywhere on the line.

More generally, a coverage curve is a piecewise linear curve that rises monotonically from $(0,0)$ to (Neg, Pos) – i.e., *TP* and *FP* can never decrease if we decrease the decision threshold. Each segment of the curve corresponds to an equivalence class of the instance space partition induced by the model in question (e.g., the leaves of a feature tree). Notice that the number of segments is never more than the number of test instances. Furthermore, the slope of each segment is equal to the ratio of positive to negative test instances in that equivalence class. For instance, in our example the first segment has a slope of 4, the second segment slope 2, and the third segment slope 1/2

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

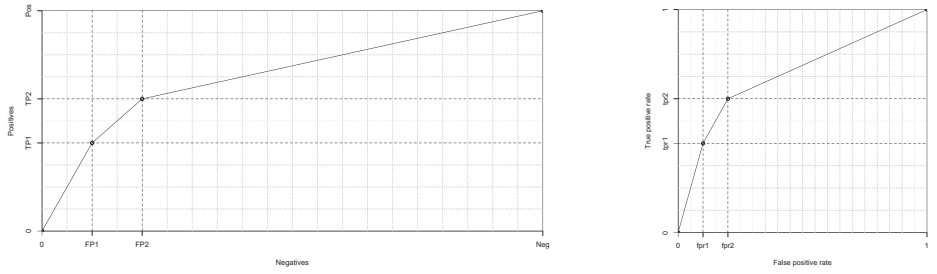

Figure 2.8. (left) A coverage curve obtained from a test set with class ratio $clr = 1/2$. **(right)** The corresponding ROC curve is the same as the one corresponding to the coverage curve in Figure 2.7 (right).

— exactly the scores assigned in each leaf of the tree! This is not true in general, since the coverage curve depends solely on the ranking induced by the scores, not on the scores themselves. However, it is not a coincidence either, as we shall see in the next section on class probability estimation.

An *ROC curve* is obtained from a coverage curve by normalising the axes to $[0,1]$. This doesn't make much of a difference in our running example, but in general coverage curves can be rectangular whereas ROC curves always occupy the unit square. One effect this has is that slopes are multiplied by $Neg/Pos = 1/clr$. Furthermore, while in a coverage plot the area under the coverage curve gives the absolute number of correctly ranked pairs, in an ROC plot *the area under the ROC curve is the ranking accuracy* as defined in Equation 2.5 on p.64. For that reason people usually write *AUC* for 'Area Under (ROC) Curve', a convention I will follow.

Example 2.4 (Class imbalance). Suppose we feed the scoring tree in Figure 2.5 on p.62 an extended test set, with an additional batch of 50 negatives. The added negatives happen to be identical to the original ones, so the net effect is that the number of negatives in each leaf doubles. As a result the coverage curve changes (because the class ratio changes), but the ROC curve stays the same (Figure 2.8). Note that the AUC stays the same as well: while the classifier makes twice as many ranking errors, there are also twice as many positive–negative pairs, so the ranking error rate doesn't change.

Let us now consider an example of a coverage curve for a grading classifier. Figure 2.9 (left) shows a linear classifier (the decision boundary is denoted B) applied to a

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

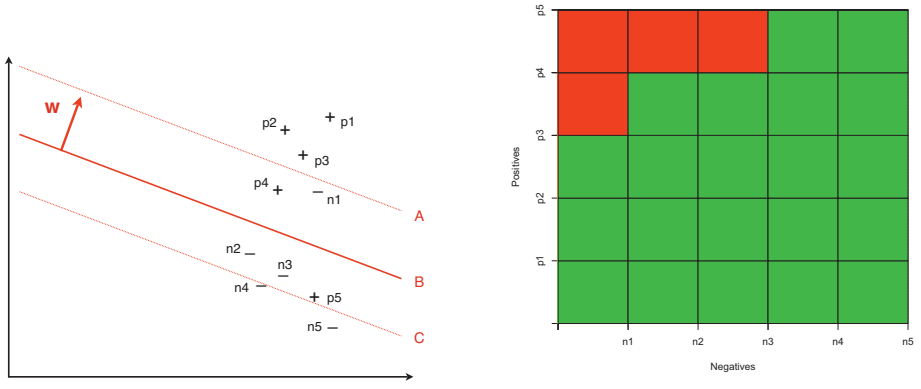

Figure 2.9. (left) A linear classifier induces a ranking by taking the signed distance to the decision boundary as the score. This ranking only depends on the orientation of the decision boundary: the three lines result in exactly the same ranking. **(right)** The grid of correctly ranked positive–negative pairs (in green) and ranking errors (in red).

small data set of five positive and five negative examples, achieving an accuracy of 0.80. We can derive a score from this linear classifier by taking the distance of an example from the decision boundary; if the example is on the negative side we take the negative distance. This means that the examples are ranked in the following order: p1 – p2 – p3 – n1 – p4 – n2 – n3 – p5 – n4 – n5. This ranking incurs four ranking errors: n1 before p4, and n1, n2 and n3 before p5. Figure 2.9 (right) visualises these four ranking errors in the top-left corner. The AUC of this ranking is $21/25 = 0.84$.

From this grid we obtain the coverage curve in Figure 2.10. Because of its stepwise character, this curve looks quite different from the coverage curves for scoring trees that we saw earlier in this section. The main reason is the absence of ties, which means that all segments in the curve are horizontal or vertical, and that there are as many segments as examples. We can generate this stepwise curve from the ranking as follows: starting in the lower left-hand corner, we go up one step if the next example in the ranking is positive, and right one step if the next example is negative. The result is a curve that goes three steps up (for p1–3), one step to the right (for n1), one step up (p4), two steps to the right (n2–3), one step up (p5), and finally two steps to the right (n4–5).

We can actually use the same procedure for grouping models if we handle ties as follows: in case of a tie between p positive examples and n negative examples, we go p steps up and *at the same time* n steps to the right. Looking back at Figure 2.7 on p.66, you will see that this is exactly what happens in the diagonal segments spanning the orange rectangles which arise as a result of the ties in the leaves of the decision tree. Thus, the principles underlying coverage and ROC curves are the same for both

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

grouping and grading models, but the curves themselves look quite different in each case. *Grouping model ROC curves have as many line segments as there are instance space segments in the model; grading models have one line segment for each example in the data set*. This is a concrete manifestation of something I mentioned in the Prologue: grading models have a much higher 'resolution' than grouping models; this is also called the model's *refinement*.

Notice the three points in Figure 2.10 labelled A, B and C. These points indicate the performance achieved by the decision boundaries with the same label in Figure 2.9. As an illustration, the middle boundary B misclassifies one out of five positives ($tpr = 0.80$) and one out of five negatives ($fpr = 0.80$). Boundary A doesn't misclassify any negatives, and boundary C correctly classifies all positives. In fact, while they should all have the same orientation, their exact location is not important, as long as boundary A is between p3 and n1, boundary B is between p4 and n2, and boundary C is between p5 and n4. There are good reasons why I chose exactly these three boundaries, as we shall see shortly. For the moment, observe what happens if we use all three boundaries to turn the linear model into a grouping model with four segments: the area above A, the region between A and B, the bit between B and C, and the rest below C. The result is that we no longer distinguish between n1 and p4, nor between n2–3 and p5. The ties just introduced change the coverage curve to the dotted segments in Figure 2.10. Notice that this results in a larger AUC of 0.90. Thus, *by decreasing a model's refinement we sometimes achieve better ranking performance*. Training a model is not just about amplifying significant distinctions, but also about diminishing the effect of misleading distinctions.

Turning rankers into classifiers

I mentioned previously that the main difference between rankers and scoring classifiers is that a ranker only assumes that a higher score means stronger evidence for the positive class, but otherwise makes no assumptions about the scale on which scores are expressed, or what would be a good score threshold to separate positives from negatives. We will now consider the question how to obtain such a threshold from a coverage curve or ROC curve.

The key concept is that of the accuracy isometric. Recall that in a coverage plot points of equal accuracy are connected by lines with slope 1. All we need to do, therefore, is to draw a line with slope 1 through the top-left point (which is sometimes called *ROC heaven*) and slide it down until we touch the coverage curve in one or more points. Each of those points achieves the highest accuracy possible with that model. In Figure 2.10 this method would identify points A and B as the points with highest accuracy (0.80). They achieve this in different ways: e.g., model A is more conservative on the positives.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

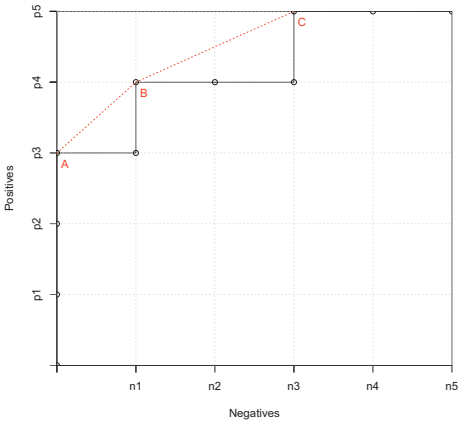

Figure 2.10. The coverage curve of the linear classifier in Figure 2.9. The points labelled A, B and C indicate the classification performance of the corresponding decision boundaries. The dotted lines indicate the improvement that can be obtained by turning the grading classifier into a grouping classifier with four segments.

A similar procedure can be followed with ROC plots, as long as you keep in mind that all slopes have to be multiplied by the reciprocal of the class ratio, $1/clr = Neg/Pos$.

Example 2.5 (Tuning your spam filter). You have carefully trained your Bayesian spam filter, and all that remains is setting the decision threshold. You select a set of six spam and four ham e-mails and collect the scores assigned by the spam filter. Sorted on decreasing score these are 0.89 (spam), 0.80 (spam), 0.74 (ham), 0.71 (spam), 0.63 (spam), 0.49 (ham), 0.42 (spam), 0.32 (spam), 0.24 (ham), and 0.13 (ham). If the class ratio of 3 spam against 2 ham is representative, you can select the optimal point on the ROC curve using an isometric with slope 2/3. As can be seen in Figure 2.11, this leads to putting the decision boundary between the sixth spam e-mail and the third ham e-mail, and we can take the average of their scores as the decision threshold (0.28).

An alternative way of finding the optimal point is to iterate over all possible split points – from before the top ranked e-mail to after the bottom one – and calculate the number of correctly classified examples at each split: 4 – 5 – 6 – 5 – 6 – 7 – 6 – 7 – 8 – 7 – 6. The maximum is achieved at the same split point, yielding an accuracy of 0.80. A useful trick to find out which accuracy an isometric in an ROC plot represents is to intersect the isometric with the descending diagonal.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

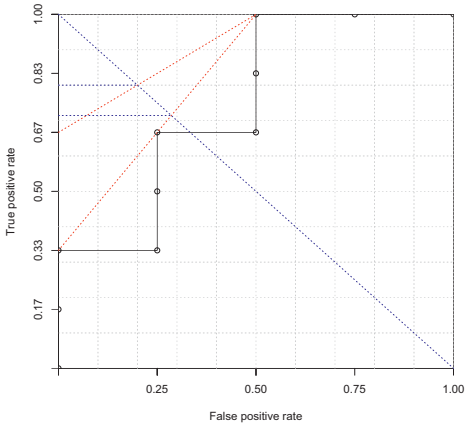

Figure 2.11. Selecting the optimal point on an ROC curve. The top dotted line is the accuracy isometric, with a slope of 2/3. The lower isometric doubles the value (or prevalence) of negatives, and allows a choice of thresholds. By intersecting the isometrics with the descending diagonal we can read off the achieved accuracy on the y-axis.

Since accuracy is a weighted average of the true positive and true negative rates, and since these are the same in a point on the descending diagonal, we can read off the corresponding accuracy value on the y-axis.

If the class distribution in the data is *not* representative, we can simply adjust the slope of the isometric. For example, if ham is in fact twice as prevalent, we use an isometric with slope 4/3. In the previous example this leads to three optimal points on the ROC curve.[10] Even if the class ratio in the data is representative, we may have other reasons to assign different weights to the classes. To illustrate, in the spam e-mail situation our spam filter may discard the false positives (ham e-mails misclassified as spam) so we may want to drive the false positive rate down by assigning a higher weight to the negatives (ham). This is often expressed as a *cost ratio* $c = c_{\mathrm{FN}}/c_{\mathrm{FP}}$ of the cost of false negatives in proportion to the cost of false positives, which in this case would be set to a value smaller than 1. The relevant isometrics then have a slope of $1/c$ in a coverage plot, and $1/(c \cdot clr)$ in an ROC plot. The combination of cost ratio and class ratio gives a precise context in which the classifier is deployed and is referred to as the

[10] It seems reasonable to choose the middle of these three points, leading to a threshold of 0.56. An alternative is to treat all e-mails receiving a score in the interval [0.28, 0.77] as lying on the decision boundary, and to randomly assign a class to those e-mails.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

operating condition.

If the class or cost ratio is highly skewed, this procedure may result in a classifier that assigns the same class to all examples. For instance, if negatives are 1 000 times more prevalent than positives, accuracy isometrics are nearly vertical, leading to an unattainably high decision threshold and a classifier that classifies everything as negative. Conversely, if the profit of one true positive is 1 000 times the cost of a false positive, we would classify everything as positive – in fact, this is the very principle underlying spam e-mail! However, often such one-size-fits-all behaviour is unacceptable, indicating that accuracy is not the right thing to optimise here. In such cases we should use average recall isometrics instead. These run parallel to the ascending diagonal in both coverage and ROC plots, and help to achieve similar performance on both classes.

The procedure just described learns a decision threshold from labelled data by means of the ROC curve and the appropriate accuracy isometric. This procedure is often preferable over fixing a decision threshold in advance, particularly if scores are expressed on an arbitrary scale – for instance, this would provide a way to finetune the SpamAssassin decision threshold to our particular situation and preferences. Even if the scores are probabilities, as in the next section, these may not be sufficiently well estimated to warrant a fixed threshold of 0.5.

2.3 Class probability estimation

A *class probability estimator* – or probability estimator in short – is a scoring classifier that outputs probability vectors over classes, i.e., a mapping $\hat{\mathbf{p}} : \mathscr{X} \to [0,1]^k$. We write $\hat{\mathbf{p}}(x) = \big(\hat{p}_1(x), \ldots, \hat{p}_k(x)\big)$, where $\hat{p}_i(x)$ is the probability assigned to class C_i for instance x, and $\sum_{i=1}^{k} \hat{p}_i(x) = 1$. If we have only two classes, the probability associated with one class is 1 minus the probability of the other class; in that case, we use $\hat{p}(x)$ to denote the estimated probability of the positive class for instance x. As with scoring classifiers, we usually do not have direct access to the true probabilities $p_i(x)$.

One way to understand the probabilities $\hat{p}_i(x)$ is as estimates of the probability $P_{\mathscr{C}}(c(x') = C_i | x' \sim x)$, where $x' \sim x$ stands for 'x' is similar to x'. In other words, how frequent are instances of this class among instances similar to x? The intuition is that the more (or less) frequent they are, the more (or less) confident we should be in our belief that x belongs to that class as well. What we mean with similarity in this context will depend on the models we are considering – we will illustrate it here by means of a few two-class examples. First, assume a situation in which any two instances are similar to each other. We then have $P_{\mathscr{C}}(c(x') = \oplus | x' \sim x) = P_{\mathscr{C}}(c(x') = \oplus)$ which is simply estimated by the proportion *pos* of positives in our data set (I am going to drop the subscript $\mathscr{C}$ from now on). In other words, in this scenario we predict $\hat{p}(x) = pos$ regardless

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

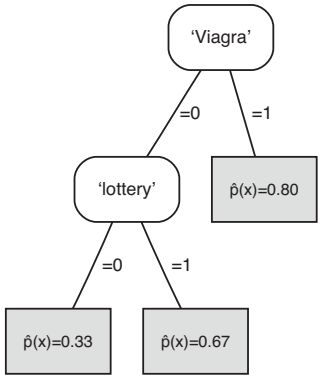

Figure 2.12. A probability estimation tree derived from the feature tree in Figure 1.4.

of whether we know anything about x's true class. At the other extreme, consider a situation in which no two instances are similar unless they are the same, i.e., $x' \sim x$ if $x' = x$, and $x' \not\sim x$ otherwise. In this case we have $P(c(x') = \oplus | x' \sim x) = P(c(x) = \oplus)$, which – because x is fixed – is 1 if $c(x) = \oplus$ and 0 otherwise. Put differently, we predict $\hat{p}(x) = 1$ for all known positives and $\hat{p}(x) = 0$ for all known negatives, but we can't generalise this to unseen instances.

A feature tree allows us to strike a balance between these extreme and simplistic scenarios, using the similarity relation $\sim_T$ associated with feature tree T: $x' \sim_T x$ if, and only if, x and x' are assigned to the same leaf of the tree. In each leaf we then predict the proportion of positives assigned to that leaf. For example, in the right-most leaf in Figure 1.4 on p.32 the proportion of positives is $40/50 = 0.80$, and thus we predict $\hat{p}(x) = 0.80$ for all instances x assigned to that leaf; similarly for the other two leaves (Figure 2.12). If we threshold $\hat{p}(x)$ at 0.5 (i.e., predict spam if the spam probability is 0.5 or more and predict ham otherwise), we get the same classifier as obtained by predicting the majority class in each leaf of the feature tree.

Assessing class probability estimates

As with classifiers, we can now ask the question of how good these class probability estimators are. A slight complication here is that, as already remarked, we do not have access to the true probabilities. One trick that is often applied is to define a binary vector $(I[c(x) = C_1], \ldots, I[c(x) = C_k])$, which has the i-th bit set to 1 if x's true class is C_i and all other bits set to 0, and use these as the 'true' probabilities. We can then define the *squared error* (*SE*) of the predicted probability vector $\hat{\mathbf{p}}(x) = \big(\hat{p}_1(x), \ldots, \hat{p}_k(x)\big)$ as

$$\mathrm{SE}(x) = \frac{1}{2} \sum_{i=1}^{k} (\hat{p}_i(x) - I[c(x) = C_i])^2 \tag{2.6}$$

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

and the *mean squared error* (*MSE*) as the average squared error over all instances in the test set:

$$\mathrm{MSE}(Te) = \frac{1}{|Te|} \sum_{x \in Te} \mathrm{SE}(x) \tag{2.7}$$

This definition of error in probability estimates is often used in *forecasting theory* where it is called the *Brier score*. The factor 1/2 in Equation 2.6 ensures that the squared error per example is normalised between 0 and 1: the worst possible situation is that the wrong class is predicted with probability 1, which means two 'bits' are wrong. For two classes this reduces to a single term $(\hat{p}(x) - I[c(x) = \oplus])^2$ only referring to the positive class. Notice that, if a class probability estimator is 'categorical' – i.e., it assigns probability 1 to one class and probability 0 to the rest – it is effectively a classifier, and MSE reduces to accuracy as defined in Section 2.1.

Example 2.6 (Squared error). Suppose one model predicts $(0.70, 0.10, 0.20)$ for a particular example x in a three-class task, while another appears much more certain by predicting $(0.99, 0, 0.01)$. If the first class is the actual class, the second prediction is clearly better than the first: the SE of the first prediction is $((0.70-1)^2 + (0.10-0)^2 + (0.20-0)^2)/2 = 0.07$, while for the second prediction it is $((0.99-1)^2 + (0-0)^2 + (0.01-0)^2)/2 = 0.0001$. The first model gets punished more because, although mostly right, it isn't quite sure of it.

However, if the third class is the actual class, the situation is reversed: now the SE of the first prediction is $((0.70-0)^2 + (0.10-0)^2 + (0.20-1)^2)/2 = 0.57$, and of the second $((0.99-0)^2 + (0-0)^2 + (0.01-1)^2)/2 = 0.98$. The second model gets punished more for not just being wrong, but being presumptuous.

Returning to the probability estimation tree in Figure 2.12, we calculate the squared error per leaf as follows (left to right):

$$\mathrm{SE}_1 = 20(0.33-1)^2 + 40(0.33-0)^2 = 13.33$$
$$\mathrm{SE}_2 = 10(0.67-1)^2 + 5(0.67-0)^2 = 3.33$$
$$\mathrm{SE}_3 = 20(0.80-1)^2 + 5(0.80-0)^2 = 4.00$$

which leads to a mean squared error of $\mathrm{MSE} = \frac{1}{100}(\mathrm{SE}_1 + \mathrm{SE}_2 + \mathrm{SE}_3) = 0.21$. An interesting question is whether we can change the predicted probabilities in each leaf to obtain a lower mean squared error. It turns out that this is not possible: predicting probabilities obtained from the class distributions in each leaf is optimal in the sense of lowest MSE.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

For instance, changing the predicted probabilities in the left-most leaf to 0.40 for spam and 0.60 for ham, or 0.20 for spam and 0.80 for ham, results in a higher squared error:

$$\mathrm{SE}_1' = 20(0.40-1)^2 + 40(0.40-0)^2 = 13.6$$
$$\mathrm{SE}_1'' = 20(0.20-1)^2 + 40(0.20-0)^2 = 14.4$$

The reason for this becomes obvious if we rewrite the expression for two-class squared error of a leaf as follows, using the notation $n^{\oplus}$ and $n^{\ominus}$ for the numbers of positive and negative examples in the leaf:

$$n^{\oplus}(\hat{p}-1)^2 + n^{\ominus}\hat{p}^2 = (n^{\oplus}+n^{\ominus})\hat{p}^2 - 2n^{\oplus}\hat{p} + n^{\oplus} = (n^{\oplus}+n^{\ominus})\left[\hat{p}^2 - 2\dot{p}\hat{p} + \dot{p}\right]$$
$$= (n^{\oplus}+n^{\ominus})\left[(\hat{p}-\dot{p})^2 + \dot{p}(1-\dot{p})\right]$$

where $\dot{p} = n^{\oplus}/(n^{\oplus}+n^{\ominus})$ is the relative frequency of the positive class among the examples covered by the leaf, also called the *empirical probability*. As the term $\dot{p}(1-\dot{p})$ does not depend on the predicted probability $\hat{p}$, we see immediately that we achieve lowest squared error in the leaf if we assign $\hat{p} = \dot{p}$.

Empirical probabilities are important as they allow us to obtain or finetune probability estimates from classifiers or rankers. If we have a set S of labelled examples, and the number of examples in S of class C_i is denoted n_i, then the empirical probability vector associated with S is $\dot{\mathbf{p}}(S) = (n_1/|S|, \ldots, n_k/|S|)$. In practice, it is almost always a good idea to *smooth* these relative frequencies to avoid issues with extreme values (0 or 1). The most common way to do this is to set

$$\dot{p}_i(S) = \frac{n_i + 1}{|S| + k} \tag{2.8}$$

This is called the *Laplace correction*, after the French mathematician Pierre-Simon Laplace, who introduced it for the case $k = 2$ (also known as Laplace's rule of succession). In effect, we are adding uniformly distributed *pseudo-counts* to each of the k alternatives, reflecting our prior belief that the empirical probabilities will turn out uniform.[11] We can also apply non-uniform smoothing by setting

$$\dot{p}_i(S) = \frac{n_i + m \cdot \pi_i}{|S| + m} \tag{2.9}$$

This smoothing technique, known as the *m-estimate*, allows the choice of the number of pseudo-counts m as well as the prior probabilities π_i. The Laplace correction is a special case of the m-estimate with $m = k$ and $\pi_i = 1/k$.

If all elements of S receive the same predicted probability vector $\hat{\mathbf{p}}(S)$ – which happens if S is a segment of a grouping model – then a similar derivation to the one above

[11] This can be modelled mathematically by a prior probability distribution known as a *Dirichlet prior*.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

allows us to write the total incurred squared error over S in terms of estimated and empirical probabilities as

$$\begin{aligned}\mathrm{SE}(S) = \sum_{x \in S} \mathrm{SE}(x) &= \sum_{x \in S} \frac{1}{2} \sum_{i=1}^{k} (\hat{p}_i(x) - I[c(x) = C_i])^2 \\ &= \frac{1}{2}|S| \sum_{i=1}^{k} (\hat{p}_i(S) - \dot{p}_i(S))^2 + \frac{1}{2}|S| \sum_{i=1}^{k} (\dot{p}_i(S)(1 - \dot{p}_i(S))\end{aligned}$$

The first term of the final expression is called the *calibration loss*, and measures squared error with respect to the empirical probabilities. It can be reduced to 0 in grouping models where we are free to choose the predicted probabilities for each segment, as in probability estimation trees. Models with low calibration loss are said to be well-calibrated. The second term is called the *refinement loss*; this depends only on the empirical probabilities, and is smaller if they are less uniform.

This analysis suggests that the best way of obtaining probability estimates is from empirical probabilities, obtained from the training set or from another set of labelled examples specifically set aside for the purpose. However, there are two issues we need to consider here. The first is that with some models we must make sure that the predicted probabilities obey the ranking imposed by the model. The second is that with grading models we don't have immediate access to empirical probabilities, since each example tends to get assigned an equivalence class of its own. We will now discuss this in a bit more detail.

Turning rankers into class probability estimators

Consider again Example 2.5 on p.70, and imagine the scores are not probabilities but on some unknown scale, so that the spam filter is a ranker rather than a class probability estimator. Since each test example receives a different score, the 'empirical probabilities' are either 0 (for negative examples) or 1 (for positive examples), leading to a sequence of $\dot{p}$-values of 1 – 1 – 0 – 1 – 1 – 1 – 0 – 1 – 1 – 1 – 0 – 0 in order of decreasing scores. The obvious problem is that these $\dot{p}$-values do not obey the order imposed by the scores, and so cannot be used directly to obtain probability estimates. Smoothing the empirical probabilities using Laplace correction doesn't really address this problem, since all it does is replace 0 with 1/3 and 1 with 2/3. We need a different idea.

Looking at Figure 2.11, we see that $\dot{p} = 1$ corresponds to a vertical segment of the ROC curve, and $\dot{p} = 0$ to a horizontal segment. The problem we have is caused by having a vertical segment following a horizontal one, or, more generally, a segment with steeper slope following a flatter segment. We will call a sequence of segments with increasing slope a *concavity*, as it forms a 'dent' in the ROC curve. A curve without concavities is a *convex* ROC curve. Our curve has two concavities: one formed by the third, fourth and fifth example, and the other by the sixth, seventh and eighth example.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

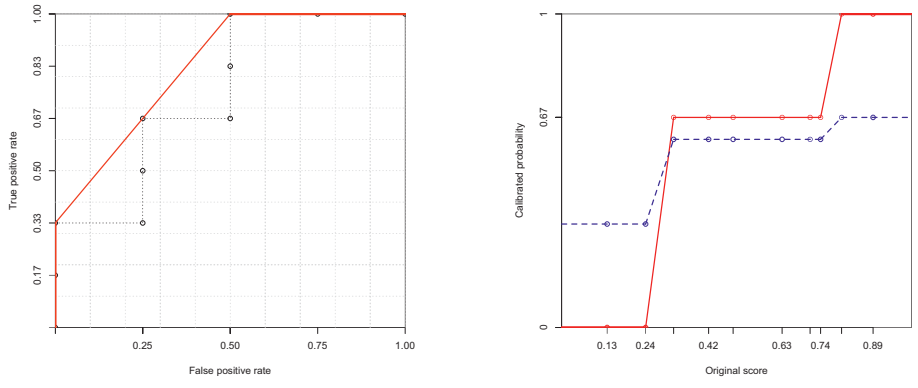

Figure 2.13. (left) The solid red line is the convex hull of the dotted ROC curve. **(right)** The corresponding calibration map in red: the plateaus correspond to several examples being mapped to the same segment of the convex hull, and linear interpolation between example scores occurs when we transition from one convex hull segment to the next. A Laplace-corrected calibration map is indicated by the dashed line in blue: Laplace smoothing compresses the range of calibrated probabilities but can sometimes affect the ranking.

Suppose now that the third to the fifth example all receive the same score, say 0.7; and the sixth to the eight example are also tied, say at 0.4. In that case the ROC curve would have six segments, with empirical probabilities 1 – 1 – 2/3 – 2/3 – 0 – 0. As we see, the $\dot{p}$-values are now decreasing with the scores; in other words, the concavities have disappeared and the ROC curve has become convex.

More generally speaking, *concavities in ROC curves can be remedied by combining segments through tied scores*. This is achieved by identifying what are sometimes called *adjacent violators*. For instance, in the sequence 1 – 1 – 0 – 1 – 1 – 0 – 1 – 1 – 0 – 0, the third and fourth example are adjacent violators, because they violate the rule that scores should be decreasing from left to right in the sequence (or, mathematically more accurate, they should be non-increasing). This is remedied by assigning them both their average score, leading to the sequence 1 – 1 – [1/2 – 1/2] – 1 – 0 – 1 – 1 – 0 – 0. The newly introduced segment now forms an adjacent violator pair with the fourth example, so we give them all their mean score, leading to the sequence 1 – 1 – [2/3 – 2/3 – 2/3] – 0 – 1 – 1 – 0 – 0.[12] The second 0 – 1 – 1 concavity is treated identically, and the final sequence is 1 – 1 – [2/3 – 2/3 – 2/3] – [2/3 – 2/3 – 2/3] – 0 – 0.

The result is illustrated in Figure 2.13. On the left, we see how the two concavities are replaced with two diagonal line segments with the same slope. These diagonal segments coincide with the accuracy isometric that gives the three 'outermost' points

[12] These two steps can be combined into one: once a pair of adjacent violators is found, we can scan to the left and right to include examples with the same score as the left and right example in the pair, respectively.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

involved in the concavities the same accuracy (Figure 2.11 on p.71). Jointly, the red segments constitute the *convex hull* of the ROC curve, which is the unique convex curve through the outermost points of the original ROC curve. The convex hull has a higher AUC than the original curve, because it replaces (some of) the ranking errors of the original curve with half-errors due to ties. In our example the original ranking incurs 6 out of 24 ranking errors (AUC = 0.75), while the convex hull turns all of these into half-errors (AUC = 0.83).

Once we have determined the convex hull, we can use the empirical probabilities in each segment of the convex hull as calibrated probabilities. Figure 2.13 (right) shows the resulting *calibration map*, which is a piecewise linear, non-decreasing curve mapping original scores on the x-axis to calibrated probabilities on the y-axis. Also shown is an alternative calibration map giving probability estimates after Laplace correction: for the given sequence these are 2/3 – 2/3 – [3/5 – 3/5 – 3/5] – [3/5 – 3/5 – 3/5] – 1/3 – 1/3, giving rise to a much compressed range of probability estimates.

Let's now look at this process from the point of view of mean squared error, calibration and refinement. The original scores had a mean squared error of $\frac{1}{10}[(0.89-1)^2 + (0.80-1)^2 + (0.74-0)^2 + (0.71-1)^2 + (0.63-1)^2 + (0.49-0)^2 + (0.42-1)^2 + (0.32-1)^2 + (0.24-0)^2 + (0.13-0)^2] = 0.19$. Notice that this is entirely incurred by the calibration loss, as all empirical probabilities are either 0 or 1 and thus the refinement loss is zero. The calibrated scores have a mean squared error of $\frac{1}{10}[(1-1)^2 + (1-1)^2 + (0.67-0)^2 + (0.67-1)^2 + (0.67-1)^2 + (0.67-0)^2 + (0.67-1)^2 + (0.67-1)^2 + (0-0)^2 + (0-0)^2] = 0.13$. Now the entire mean squared error is incurred by refinement loss as the estimated probabilities are equal to the empirical ones in each segment by construction. We have traded an increase in refinement loss for a decrease in calibration loss; since the latter is larger than the former, the overall error decreases. The increase in refinement loss comes from the construction of the convex hull, which introduces diagonal segments. The technical term for this process of obtaining calibrated scores through the convex hull of the ROC curve is *isotonic calibration*, as the mathematical problem underlying it is called isotonic regression. Some caution is in order when applying isotonic calibration, as it is easy to overfit the data in this process. In the calibration map in Figure 2.13 (right), both the horizontal transition points and the vertical levels are directly obtained from the given data, and may not generalise well to unseen data. This is why it is advisable to apply the Laplace correction to the empirical probabilities, even though it will increase the calibration loss on the given data.

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

2.4 Binary classification and related tasks: Summary and further reading

In this chapter we have looked at binary classification, a ubiquitous task that forms the starting point of a lot of work in machine learning. Although we haven't talked much about learning in this chapter, my philosophy is that you will reach a better understanding of machine learning models and algorithms if you first study the tasks that these models are meant to address.

☞ In Section 2.1 we defined the binary classification task and introduced an important tool to assess performance at such a task, namely the two-by-two contingency table. A wide range of performance indicators are derived from the counts in a contingency table. I introduced the coverage plot, which visualises a contingency table as a rectangle with size *Pos* up and size *Neg* across, and within that rectangle a point with y-coordinate *TP* and x-coordinate *FP*. We can visualise several models evaluated on the same data set by several points, and use the fact that accuracy is constant along line segments with slope 1 to visually rank these classifiers on accuracy. Alternatively, we can normalise the rectangle to be a unit square with true and false positive rate on the axes. In this so-called ROC space, line segments with slope 1 (i.e., those parallel to the ascending diagonal) connect points with the same average recall (sometimes also called macro-accuracy). The use of these kinds of plot in machine learning was pioneered by Provost and Fawcett (2001). Unnormalised coverage plots were introduced by Fürnkranz and Flach (2003).

☞ Section 2.2 considered the more general task of calculating a score for each example (or a vector of scores in the general case of more than two classes). While the scale on which scores are expressed is unspecified, it is customary to put the decision threshold at $\hat{s}(x) = 0$ and let the sign of the score stand for the prediction (positive or negative). Multiplying the score with the true class gives us the margin, which is positive for a correct prediction and negative for an incorrect one. A loss function determines how much negative margins are penalised and positive margins rewarded. The advantage of working with convex and continuously differentiable 'surrogate' loss functions (rather than with 0–1 loss, which is the loss function we ultimately want to optimise) is that this often leads to more tractable optimisation problems.

☞ Alternatively, we can ignore the scale on which scores are measured altogether and only work with their order. Such a ranker is visualised in coverage or ROC space by a piecewise continuous curve. For grouping models the line segments in these curves correspond to instance space segments (e.g., the leaves of a tree

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press

model) whereas for grading models there is a segment for each unique score assigned by the model. The area under the ROC curve gives the ranking accuracy (an estimate of the probability that a random positive is ranked before a random negative) and is known in statistics as the Wilcoxon-Mann-Whitney statistic These curves can be used to find a suitable operating point by translating the operating condition (class and cost distribution) into an isometric in ROC or coverage space. The origins of ROC curves are in signal detection theory (Egan, 1975); accessible introductions can be found in (Fawcett, 2006; Flach, 2010*b*).

☞ In Section 2.3 we looked at scoring models whose scores can be interpreted as estimates of the probability that the instance belongs to a particular class. Such models were pioneered in forecasting theory by Brier (1950) and Murphy and Winkler (1984), among others. We can assess the quality of class probability estimates by comparing them to the 'ideal' probabilities (1 for a positive, 0 for a negative) and taking mean squared error. Since there is no reason why the true probabilities should be categorical this is quite a crude assessment, and decomposing it into calibration loss and refinement loss provides useful additional information. We have also seen a very useful trick for smoothing relative frequency estimates of probabilities by adding pseudo-counts, either uniformly distributed (Laplace correction) or according to a chosen prior (m-estimate). Finally, we have seen how we can use the ROC convex hull to obtain calibrated class probability estimates. The approach has its roots in isotonic regression (Best and Chakravarti, 1990) and was introduced to the machine learning community by Zadrozny and Elkan (2002). Fawcett and Niculescu-Mizil (2007) and Flach and Matsubara (2007) show that the approach is equivalent to calibration by means of the ROC convex hull. (Note that in this chapter we have seen two different uses of the term 'convex': one in relation to loss functions, where convexity means that linear interpolation between any two points on the curve depicting the loss function will never result in a point below the curve; and the other in relation to the ROC convex hull, where it refers to the linearly interpolated boundary of a convex set which envelopes all points in the set.)

https://doi.org/10.1017/CBO9780511973000.004 Published online by Cambridge University Press