Chapter 7

Classifier Evaluation

7.1 Introduction

Classifiers induced from empirical data can be evaluated along at least two dimensions: Performance and explanatory features. By performance is meant assessment of how well the classifier does in classifying new cases, according to some specified performance criterion. By explanatory features is meant how interpretable the structure of the induced classifier is, so that one might gain some insight into how the classification or decision making process is carried out. How these two evaluation dimensions are to weighted is a matter of the intended role of the induced classifier. If the classifier is to operate in a fully automated environment, then performance may be the only feature of interest. Conversely, if the classifier induction is part of a larger data mining or knowledge discovery process, then the interpretability of the classifier will be increasingly more important, and a decrease in performance may be acceptable. Classifiers that are to function as parts of interactive decision support systems lie somewhere in the middle of these two extremes.

Different machine learning methods for classification vary in how much they facilitate the knowledge discovery aspect, depending on the type of classifiers they produce. A point that is often held forth in favour of methods that produce decision tree or rule sets is that the models are directly readable and interpretable. In contrast, methods such as logistic regression or artificial neural networks are more difficult to interpret and may require familiarity with sophisticated statistical concepts. This chapter will not touch upon the issue of classifier interpretability, but will concern itself only with the performance evaluation aspect.

A classifier κ is in the following viewed as a realization of a function that, when applied to an object $x \in U$ in an information system $\mathcal{A}$, assigns a classification $\hat{d}_\kappa(x)$ to x. The true actual classification of x is denoted $d(x)$. In the following, unless otherwise stated and relevant for the exposition, no distinction will be made between a classifier and the function it realizes.

$$\hat{d}_\kappa : U \to V_d \tag{7.1}$$

We assume in the following that κ is forced to make a classification when presented with an object. In the case that the classifier fails to recognize an object, a default classification is assumed invoked. Extensions that incorporate rejection and degrees of doubt by κ are possible [179].

Section 7.2 discusses how to conduct classification experiments in a structured and systematic way so that reliable performance estimates can be obtained, while Section 7.3 reviews a simple way of collating the produced classifications. Section 7.4 then focuses on evaluation methods for binary classifiers. Two key evaluation dimensions are discussed in Section 7.4.2, and frameworks for graphically assessing these are presented in Section 7.4.3 and Section 7.4.4. Different quantitative performance measures are discussed in Section 7.4.5, while Section 7.4.6 outlines techniques for statistical comparison of such measures.

7.2 Partitioning the Examples

In supervised learning we are given a set of labeled example objects in a decision system $\mathcal{A}$, and want to construct a mapping $\hat{d}_\kappa$ that maps elements in U to elements in V_d, using only attributes contained in A. If we use the full set of examples $\mathcal{A}$ to construct the classifier, we may obtain a model that, if the chosen the learning paradigm lends itself to it, can be inspected and generate hypotheses that explain the observations. In some applications this might be valuable, but then we do not have any data left to assess the performance of the classifier in an unbiased manner. Applying the classifier to the dataset from which it was induced will obviously give an unfair and biased assessment, since a classifier can always be constructed that reproduces its originating data material perfectly. A perfect (or near-perfect, in the case of inconsistent labels) mapping can easily be constructed through rote learning, for instance by collecting all the provided examples in a lookup table. Since the full set of examples $\mathcal{A}$ was used to construct $\hat{d}_\kappa$, we have no way of assessing how well $\hat{d}_\kappa$ generalizes to new and unseen example objects.

To this end and since in practice $\mathcal{A}$ is almost always a finite and limited collection of possible examples, it is customary to randomly divide the examples in $\mathcal{A}$ into two disjoint subsets, a *training set* and a *test set*. The training set is used to construct κ, while the test set is used to assess its performance. Under the assumption that the two sets comprise independent samples, this ensures us that the performance estimate will be unbiased.

Every algorithm for classifier construction has a set of parameters or options which can be tweaked and tuned, and it is natural to select these settings so that the chosen performance criterion is optimized. Performance assessment for this purpose should *not* be made using the test set, since we will then have no way of estimating the classifier's true performance. And using the training set for parameter tuning may also not

desirable, since we may then be likely to *overfit* our data. By overfitting is meant that the constructed mapping is overly geared towards reproducing the training set, and may have captured noise and other data impurities that may be present. An overfitted model is not likely to generalize well to unseen examples.

These considerations may lead us to divide $\mathcal{A}$ into three disjoint sets of examples instead of two, namely into a training set, a test set and a *hold-out set*. While the classifier κ is induced from the training set, the purpose of the hold-out set is to function as a "test set during training", i.e., as a set of examples apart from those considered when constructing the mapping defined by κ, used for guiding the training process. The performance assessment on this hold-out set can be used to tune the training parameters, decide on when training should be stopped, or to aid in pruning or simplification of the model derived from the training set. The test set is kept completely separate, and is only used at the last moment to evaluate the performance of the classifier that was constructed from the training and hold-out sets.

7.2.1 Systematic Partitioning Methods

The reliability of the performance estimated from a single partitioning can be questioned. It could be that the random division of examples used for training and testing was a particularly "lucky" or "unlucky" split. One way of reducing the possible impact of the split is to repeat the described training and test process several times with different random splits, and to average the performance estimates from each iteration. However, this way the training and testing sets from iteration i will almost surely partially overlap with the training and testing sets from iteration j. Hence it would be desirable to take a more systematic approach where also the possibility of similar partitionings is taken into account.

Cross-Validation

The process of *cross-validation* (CV) is a way of getting more reliable estimates and more mileage out of possibly scarce data. In *k-fold* CV we randomly divide the set of examples into k disjoint "blocks" of examples, usually of equal size. Then we can apply a classifier trained using $k-1$ blocks to the remaining block to assess its performance. Repeating this for each of the k blocks enables us to average the estimates from each iteration to obtain an unbiased performance estimate. By this procedure, each example is guaranteed to be in the test set once and in the training size $k-1$ times.

An extreme variant of selecting k is to choose $k = |U|$, i.e., letting each test set consist of a single example. This is called *leave-one-out* CV, and, although potentially extremely computer-intensive, may be intuitively pleasing as it most closely mimics the true size of the training set.

Bootstrapping

Another approach to systematic partitioning of the training examples is the *bootstrap*. The basic idea of the procedure known as the *0.632-bootstrap* is to randomly sample a training set $\mathcal{A}^*$ of $|U|$ examples from $\mathcal{A}$ *with replacement*, and to assess the performance of the classifier on $\mathcal{A}^*$ and on the examples in $\mathcal{A}$ and not in $\mathcal{A}^*$. These two estimates are then weighted by 0.368 and 0.632[1] respectively, and added to obtain a bootstrap performance estimate. The process can then be repeated several times in order to compute the average bootstrap estimate.

Remarks

Of course, systematic partitioning methods can be combined with using hold-out sets, simply by doing a second-level split on the examples originally singled out for training. Salzberg [183] recommends using CV together with hold-out sets as a means of ensuring fair comparisons between classifiers.

If inspection of the model for KDD purposes is a primary goal, then systematic partitioning methods introduces a complicating factor. For what is then the model that goes along with the obtained performance estimate? Instead of a single model, we have instead a plethora of different models. A common way to interpret the performance estimate is as the estimate we would get if we had induced a model from the full database and had had more data with which to test it.

Resampling ideas can be employed not only for systematically partitioning the data into training and test examples, but can also be used in process of inducing a model from the training set. We can thus introduce resampling at two different levels: At the evaluation level and at the model induction level. If the latter is done, the induced model is really an ensemble of several submodels that are ultimately combined. Such meta-methods are sometimes called *bagging*[2] or *boosting*. In the bagging procedure by Breiman [19], a large number of bootstrapped training sets are sampled from the training set by the previously described bootstrap procedure, and one submodel is induced from each of these bootstrap samples. The final model is then defined as the aggregation of all the submodels by uniform voting, i.e., when an presented with an unseen example, the ensemble labels it with the class that is predicted by the greatest number of submodels. In boosting [60, 186], the training set chosen at any point depends on the performance of earlier classifiers. Examples that are incorrectly classified are then chosen more often than correctly classified examples. Bagging and boosting can often considerably improve classificatory performance [10].

Further general readings on CV, bootstrapping, and other resampling techniques can be found in, e.g., [52, 179].

[1] The number 0.632 is shorthand for $(1 - \frac{1}{e})$, the limit for large $|U|$ of the probability that a given example in $\mathcal{A}$ appears in $\mathcal{A}^*$.

[2] Short for "bootstrap aggregation".

7.3 Confusion Matrices

A *confusion matrix* C is a $|V_d| \times |V_d|$ matrix with integer entries that summarizes the performance of a classifier κ, applied to the objects in an information system $\mathcal{A}$.

Without loss of generality we may assume that V_d is the set of integers $\{0, \ldots, r(d) - 1\}$, as defined in Section 5.3.1. The entry $C(i, j)$ counts the number of objects that really belong to class i, but were classified by κ as belonging to class j.

$$C(i, j) = |\{x \in U \mid d(x) = i \text{ and } \hat{d}_\kappa(x) = j\}| \tag{7.2}$$

Obviously, it is desirable for the diagonal entries to be as large as possible. Probabilities are easily estimated from the confusion matrix C by dividing an entry by the sum of the row or column the entry appears in:

$$\Pr(d(x) = i \mid \hat{d}_\kappa(x) = j) = \frac{C(i, j)}{\sum_i C(i, j)} \tag{7.3}$$

$$\Pr(\hat{d}_\kappa(x) = j \mid d(x) = i) = \frac{C(i, j)}{\sum_j C(i, j)} \tag{7.4}$$

$$\Pr(d(x) = \hat{d}_\kappa(x)) = \frac{\sum_i C(i, i)}{\sum_i \sum_j C(i, j)} \tag{7.5}$$

Classification tasks with binary outcomes are so common in practice that the entries of 2×2 confusion matrices and the corresponding values of Equation 7.3 and Equation 7.4 are given special names. Consider the following confusion matrix:

		$\hat{d}_\kappa$	
		0	1
d	0	TN	FP
	1	FN	TP

The names given to the entries and derived quantities from 2×2 confusion matrices are listed in Table 7.1 The sensitivity of a classifier thus gives us a measure of how good it is in detecting that an event defined through X_1 has occurred, while the specificity gives us a measure of how good it is in picking up non-events defined through X_0. The positive and negative predictive values of a classifier gives us a measure of how "trustworthy" it is in its detections of events and non-events.

7.4 Binary Classifiers

In the following, since the situation is so common in practice, we will examine the case where κ is a binary classifier, i.e., where $V_d = \{0, 1\}$. As defined in Section 5.3.1, the

Quantity	Name	Description
$C(0,0)$	TN	True negatives.
$C(0,1)$	FP	False positives.
$C(1,0)$	FN	False negatives.
$C(1,1)$	TP	True positives.
$TP/(TP+FN)$	Sensitivity	$\Pr(\hat{d}_\kappa(x) = 1 \mid d(x) = 1)$
$TN/(TN+FP)$	Specificity	$\Pr(\hat{d}_\kappa(x) = 0 \mid d(x) = 0)$
$TP/(TP+FP)$	Positive predictive value (PPV)	$\Pr(d(x) = 1 \mid \hat{d}_\kappa(x) = 1)$
$TN/(TN+FN)$	Negative predictive value (NPV)	$\Pr(d(x) = 0 \mid \hat{d}_\kappa(x) = 0)$

Table 7.1: Names given to the entries and derived quantities from 2×2 confusion matrices.

corresponding decision classes will be denoted X_0 and X_1.

We start by decomposing κ into realizing two functions ϕ and θ so that $\hat{d}_\kappa(x) = \theta(\phi(x))$, and refine Equation 7.1 as shown below. The function ϕ maps an object x to a continuous estimate in the interval $[0,1]$, where $\phi(x)$ is to be interpreted as the classifier's certainty that x belongs to decision class X_1. The function θ interprets the output of ϕ, and decides how the mapping into $\{0,1\}$ from the intermediate representation $[0,1]$ is to take place.

$$\hat{d}_\kappa : U \xrightarrow{\phi} [0,1] \xrightarrow{\theta} \{0,1\} \tag{7.6}$$

The function ϕ is what is usually associated with a given classifier, and is what most machine learning paradigms implement, whether it being neural networks, decision trees, rule sets, logistic regression equations or something else. Letting $\mathit{certainty}(x, X_i)$ denote the classifier's degree of certainty that x belongs to decision class X_i, we have the following:

$$\phi(x) = \mathit{certainty}(x, X_1) \tag{7.7}$$

It is typically desirable for $\phi(x)$ to estimate the probability $\Pr(d(x) = 1 \mid x)$, as will be discussed in Section 7.4.2.

The function θ interprets $\phi(x)$ and decides on how this is to be converted into a final decision value. Here, θ will be assumed to be a simple threshold function that outputs decision value 1 if $\phi(x)$ is above a certain threshold $0 \leq \tau \leq 1$, and decision value 0 otherwise:

$$\theta(\phi(x)) = \begin{cases} 1 & \text{if } \phi(x) \geq \tau \\ 0 & \text{otherwise} \end{cases} \tag{7.8}$$

Other definitions of θ might under some circumstances be appropriate, for instance if our application requires the classifier to express 'doubt' as a separate classification.

7.4.1 Realizing ϕ

As mentioned, different classifiers may realize the certainty function ϕ in radically different fashions. For decision rules as defined in Section 6.3, variants of the voting procedures explored in Section 6.4 may be used. Using, e.g., Equation 6.12 and normalizing by all votes cast, for binary classifiers the certainty function reduces to Equation 7.9.

$$\phi(x) = \frac{votes((d = 1))}{votes((d = 0)) + votes((d = 1))} \tag{7.9}$$

7.4.2 Discrimination and Calibration

Two of the main performance dimensions along which binary outcome classifiers can be evaluated is *discrimination* and *calibration*. Calibration deals with probability estimation, while discrimination deals with classificatory abilities.

Discrimination measures how well the classifier is able to separate objects in decision class X_0 from objects in decision class X_1. Examples of measures of discrimination are accuracy and the area under ROC curves, further discussed in Section 7.4.5. Discrimination is in many ways a natural and intuitive performance measure to focus on, as it indicates how good a classifier is at doing what it was designed to do, namely at guessing the correct value for $d(x)$ when presented with object x.

So-called *calibration-in-the-large* measures how close the average intermediate model output is to the average actual outcome, computed on the basis of the whole sample. Calibration-in-the-large, also referred to as *bias*, thus gives an overall picture of whether a model is "optimistic" or "pessimistic", since it signals if the model outputs are systematically high or low. However, calibration-in-the-large is not very useful in practice since a model may be perfectly calibrated-in-the-large, and yet provide no information.[3]

A classifier is considered well *calibrated-in-the-small* when cases assigned a certainty value of $\phi(x)$ actually yield outcome 1 approximately $\phi(x) \times 100\%$ of the time. Hence, calibration-in-the-small measures how well the function ϕ realizes a probability estimate. In the following, calibration-in-the-small will simply be referred to as calibration.

Note that calibration is different from discrimination. A classifier may discriminate well, but be badly calibrated. Conversely, a classifier may be well calibrated, but discriminate poorly. A good discussion of calibration and discrimination can be found in [136].

A short comment on terminology might be in order. Discrimination and calibration are by no means the only names that these issues are known under. For instance,

[3] Consider for example the "default" model that for all inputs simply outputs the prevalence of disease.

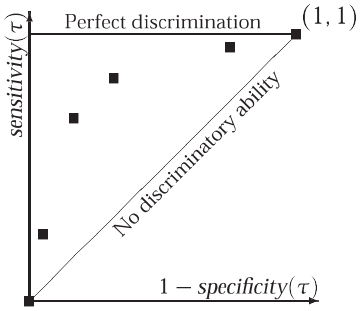

Figure 7.1: Points on an example ROC curve. The points are typically connected with straight line segments to form a curve. Each point corresponds to a different threshold value τ. A straight line from $(0,0)$ to $(1,1)$ indicates no discriminatory ability, i.e., the area under the curve is 0.5. The ideal ROC curve has an area under the curve of 1, i.e., it is a step function with segments from $(0,0)$ to $(0,1)$ to $(1,1)$.

discrimination is sometimes referred to as *resolution*, and *imprecision* is some places used instead of calibration. Hand [74] devotes a small section in his book to review and comment on some of the different terms that are employed in the literature.

7.4.3 ROC Curves

A *receiver operating characteristic* (ROC) curve is a graphical representation of discrimination. The analysis of ROC curves is a classic methodology from signal detection theory [216] that is common in medical diagnosis [75], and that has recently begun to be used more generally in the machine learning field [173, 174].

As can be seen from Equation 7.8, the output of a classifier κ depends on a selected threshold value τ. An ROC curve captures the behavior of κ as the threshold τ is varied across the full spectrum of possible values. A very important and desirable feature of an ROC curve is that it describes the predictive behavior of a classifier independent of class distributions and classification error costs. There is a rich literature on the field of ROC analysis.

For each different value of the threshold τ we may obtain a different 2×2 confusion matrix C when we apply κ to the objects in an information system. Let the sensitivity and specificity of C as defined in Section 7.3 be denoted by $\mathit{sensitivity}(\tau)$ and $\mathit{specificity}(\tau)$. An ROC curve is defined through a collection of points as shown in Equation 7.10. The points $(0,0)$ and $(1,1)$ are also included as these points correspond to the situations where all objects are blindly classified as belonging to the same decision class. An example ROC curve is drawn in Figure 7.1.

$$\text{ROC} = \bigcup_{\tau} \{(1 - \mathit{specificity}(\tau),\ \mathit{sensitivity}(\tau))\} \cup \{(0,0),(1,1)\} \tag{7.10}$$

An ROC curve is said to *dominate* another ROC curve if it is lies consistently above

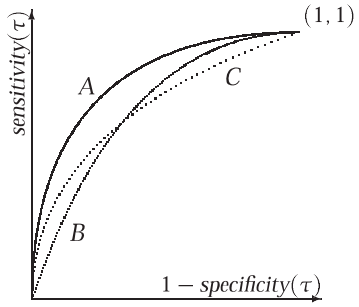

Figure 7.2: ROC curve A dominates both ROC curves B and C. Curve B does not dominate curve C, nor does C dominate B.

it in ROC space, illustrated in Figure 7.2. If the ROC curve for a classifier κ_1 clearly dominates the ROC curve of another classifier κ_2, then it is safe to say that κ_1 is better than κ_2. However, this is seldom the case since the ROC curves usually cross each other, in which case the choice of classifier might be unclear.

It should be noted that even though a binary classifier or test is better than another test with respect to an ROC analysis, then the better test is not necessarily the test that would be preferred in practice. The reason for this is that there are additional non-performance aspects of conducting a test that a plain ROC analysis does not take into account, such as the cost of acquiring the information the test is based on. For example, in a medical setting, some attributes may involve expensive drugs or therapy or incur considerable discomfort for the patient.

The ROC Convex Hull Method

When comparing several classifiers and plotting their respective ROC curves, the curves will in practice almost always cross each other somewhere, i.e., none of the classifiers will be optimal under all circumstances. Provost and Fawcett [173] propose a method for determining from ROC curves which classifier is optimal, for various class and cost distributions. Schematically, the process is as follows:

1. Compute the convex hull of all the points on the ROC curves. If a classifier does not contribute to a point on the hull, then that classifier can never be optimal.

2. Encode the class and cost distribution as a line and superimpose this as a tangent on the hull. The classifier that contributes to the line/hull intersection point is the best classifier under that particular class and cost distribution.

3. Tabulate under which class and cost distributions that the hull-contributing classifiers are optimal.

Interpreting the ROC Slope

If we examine how the axes in an ROC curve are defined and substitute Equation 7.8 and Equation 5.35 into their definitions from Section 7.3, we obtain the following:

$$\begin{aligned}\frac{\textit{sensitivity}(\tau)}{1-\textit{specificity}(\tau)} &= \frac{\Pr(\hat{d}_\kappa(x)=1 \mid d(x)=1)}{1-\Pr(\hat{d}_\kappa(x)=0 \mid d(x)=0)} \\ &= \frac{\Pr(\phi(x)\geq\tau \mid x\in X_1)}{1-\Pr(\phi(x)<\tau \mid x\in X_0)} \\ &= \frac{\Pr(\phi(x)\geq\tau \mid x\in X_1)}{\Pr(\phi(x)\geq\tau \mid x\in X_0)} \end{aligned} \tag{7.11}$$

An ROC curve is thus a plot of the cumulative distribution function for $\phi(x)$ in the X_0 subpopulation against that of the X_1 subpopulation. Consequently, the local slope of the ROC curve can be interpreted as a likelihood ratio, by computing the derivatives of the counter and denominator of Equation 7.11 with respect to τ.

Hilden [81] exploits this relationship by scaling the ROC coordinate axes by various factors that represent such things as prevalence and measures of utility, and discusses how the local slopes of the scaled ROC curves can then be interpreted. For details, see [81]. Hilden also discusses issues relating to the concavity and convexity of ROC curves.

Interpreting the ROC Integral

The area under the ROC curve (AUC) is a measure of how well the classifier is able to discriminate objects in X_0 from objects in X_1. AUC is usually computed using the trapezoidal rule for integration, but parametric methods that produce smooth curves and compute areas from these are also in use. An AUC of 0.5 represents no discriminatory ability, while an AUC of 1 represents perfect discrimination. Note that due to the shape of a typical ROC curve, the trapezoidal rule is prone to slightly underestimate the true area.

$$AUC = \int_0^1 \textit{sensitivity}(\tau)\, d\textit{specificity}(\tau) \tag{7.12}$$

The AUC can be shown to have a nice and intuitive probabilistic interpretation: Let $S_\kappa(\square) \subseteq X_0 \times X_1$ be defined as below, where "$\square$" denotes a comparison operator. A pair (x_0, x_1) is said to be *concordant* if it is a member of $S_\kappa(<)$, and *discordant* if it is a member of $S_\kappa(>)$. The set $S_\kappa(=)$ constitutes the set of *ties*. Obviously, it is desirable for a pair to be concordant. Under the assumption that tie-resolution results in half the ties being correctly ranked, the so-called *c-index* [77] equals the AUC estimated using the trapezoidal rule of integration. The *c*-index measures the probability that, given a

pair (x_0, x_1) randomly drawn from $X_0 \times X_1$, the function ϕ realized by classifier κ will rank x_0 and x_1 correctly, i.e., define a concordant pair.

$$S_\kappa(\square) = \{(x_0, x_1) \in X_0 \times X_1 \mid \phi(x_0) \square \phi(x_1)\} \tag{7.13}$$

$$c\text{-index} = \frac{|S_\kappa(<)| + \frac{1}{2}|S_\kappa(=)|}{|X_0||X_1|} \tag{7.14}$$

Statisticians may recognize the c-index as being the parameter of the Mann-Whitney-Wilcoxon rank sum test.

Selecting a Threshold

An ROC curve displays a model's discriminatory performance across a spectrum of thresholds. But if a classifier is to be implemented in practice, a value for the threshold must be decided upon. How this value should be selected is obviously a function of how one weighs the cost of false positives against the cost of false negatives. If the costs are equally weighted, the threshold that produces the "northwestern-most" point on the ROC curve, i.e., the point closest to $(0, 1)$, is an intuitive candidate. However, if we can quantify the costs and know the prevalence of disease, we can compute the threshold τ^* that minimizes the expected total cost as shown below. If π_i denotes the a priori probability that x is a member of X_i and misclassification incurs a cost c_i, the optimal threshold τ^* can be computed as follows [74]:

$$\tau^* = \arg\min_\tau \{\pi_0 c_0(1 - \textit{sensitivity}(\tau)) + \pi_1 c_1(1 - \textit{specificity}(\tau))\} \tag{7.15}$$

Hand [74] briefly relates how τ^* can be found by considering the local slope of the ROC curve, and how this can be used to identify a range of "good" threshold values.

7.4.4 Calibration Plots

A *calibration plot* is a graphical method for assessing how well calibrated a binary classifier is. Points in a calibration plot are generated as described below. Table 7.2 gives a small example of this process.

1. Sort the pairs $(d(x), \phi(x))$ according to the value of $\phi(x)$.
2. Partition the sorted vector of pairs into g groups.
3. Compute the average actual outcome $\bar{d}_i$ and the average classifier output $\bar{\phi}_i$ for each group i, and add $(\bar{d}_i, \bar{\phi}_i)$ to the plot.

Group	$d(x)$	$\phi(x)$
	0	0.04
1	0	0.21
	0	0.41
$(\bar{d}_1, \bar{\phi}_1)$	***0.00***	***0.22***
	1	0.46
2	0	0.51
	0	0.59
$(\bar{d}_2, \bar{\phi}_2)$	***0.33***	***0.52***
	1	0.61
3	1	0.75
	1	0.80
	1	0.92
$(\bar{d}_3, \bar{\phi}_3)$	***1.00***	***0.77***

Table 7.2: Generating points in a calibration plot. In this small example there are 10 $(d(x), \phi(x))$ pairs that are sorted according to $\phi(x)$ and form $g = 3$ groups. Each group defines a plot point by computing the averaged values within each group. Sometimes group sums are plotted instead of group averages.

If the classifier is well calibrated, the points in the calibration plot should tend to lie around the 45-degree line that crosses through $(0, 0)$. An example calibration plot can be found in Figure 7.3.

How to select the number of groups g is not entirely apparent. Usually, a predetermined number is simply chosen at will. Furthermore, all groups are usually set to have approximately the same size, but having variable-sized groups is also a possibility [84].

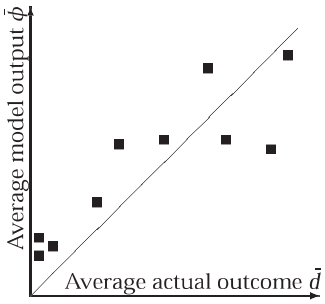

Figure 7.3: An example calibration plot with $g = 10$ groups. For a well calibrated classifier, the points should be close to the 45-degree line through $(0, 0)$.

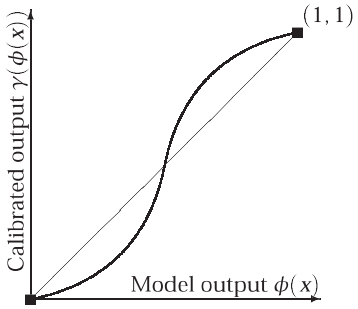

Figure 7.4: An example calibration function γ. Here, γ pushes the original model output $\phi(x)$ away from the center and towards the extreme values 0 and 1. If ϕ has a tendency to densely cluster the object certainties around a mean value, such a choice of γ may be appropriate.

Calibrating a Model

A model that may not be very well calibrated can in principle almost always be transformed into a model that is fairly well calibrated, while preserving the model's discriminatory abilities. This can be done by introducing a monotone strictly increasing function γ as shown in Equation 7.16, and using this to remap $\phi(x)$. Hence, we can imagine $\gamma(\phi(x))$ as being a calibrated version of our classifier.

$$\hat{d}_\kappa : U \xrightarrow{\phi} [0,1] \xrightarrow{\gamma} [0,1] \xrightarrow{\theta} \{0,1\} \tag{7.16}$$

The purpose of the "calibration function" γ is to stretch or compress the function ϕ in such a way that the transformation results in a better calibrated model, under the constraint that the discriminatory ability of the calibrated model does not worsen. If γ is a monotone strictly increasing function, then the ROC curve will remain unchanged. Figure 7.4 displays an example calibration function γ.

Although in principle a model can be calibrated a posteriori, it may in practice not be clear how to construct a suitable calibration function γ. An intuitive approach is to compute the linear regression line of the points in the calibration plot, and to let γ be the simple function that would "tilt" this line to equal the 45-degree line that crosses through $(0,0)$. This approach is explored by Vinterbo and Ohno-Machado [230].

7.4.5 Performance Measures

Section 7.2 discussed *how* reliable performance estimates could be gathered, but was not specific as to *which* performance measure that should be gathered. This section discusses some popular performance measures and dimensions along which classifier performance may be assessed.

Accuracy and Risk

Two measures for assessing a classifier's discriminatory performance are *accuracy* and its generalization, *risk*. The proportion of correctly classified objects, defined by Equation 7.5, is called the accuracy of the classifier, and is by far the most popular performance measure in the machine learning literature. More often than not, accuracy (or *error rate*, defined as one minus accuracy) is the only performance measure reported in comparison studies. Accuracy is an intuitive quantity to relate to, but its use as the only performance measure is questionable [174]. The main reason for this is that the measure does not capture situations where different types of classification errors have different costs, nor does it adjust for skewed outcome class distributions.

Bayes decision rule, given by Equation 7.18, states that an object x should be assigned to the most probable decision class when x is given. Bayes decision rule is intuitive, and can be shown to be theoretically optimal with respect to maximizing the accuracy, or, equivalently, minimizing the error rate [179, 185]. However, in practice, the Bayes error rate can rarely be obtained. Doing so would require perfect knowledge of all distributions and conditional probabilities involved, something that is not practically achievable.[4]

$$\hat{d}_\kappa(x) = \arg\max_k \Pr(d(x) = k \mid x) \tag{7.18}$$

An aspect that Equation 7.18 does not take into account is that not all errors are equal. Making a wrong decision of a certain kind may be dramatically more costly than making a different type of decision error. A common way of making up for this is to introduce a *loss function* L, where $L(j,k)$ denotes the loss or cost incurred by making decision j when the true decision class is k. The risk function for a classifier κ is the expected loss when using it, as a function of the unknown decision class k. Bayes decision rule for a general loss function is given by Equation 7.19. Again, since a classifier κ can only hope to form a fair approximation of $\Pr(d(x) = k \mid x)$, the theoretically minimal risk is rarely obtainable in practice.

$$\hat{d}_\kappa(x) = \arg\min_k \sum_j L(j,k) \Pr(d(x) = k \mid x) \tag{7.19}$$

With a loss function such that $L(k,k) = 0$ and 1 otherwise, Equation 7.19 reduces to Equation 7.18. Such a loss function is called a *0/1 loss function*.

Note that a binary classifier as defined in Section 7.4 with a fixed threshold for θ of $\tau = 0.5$ amounts to emulating Equation 7.18. Other threshold values amounts to

[4] The so-called *naive Bayes rule* approximates $\Pr(d(x) = k \mid x)$ as below. The naive Bayes classifier often works very well in practice, and excellent classification may be obtained even when the probability estimates contain large errors [46].

$$\Pr(d(x) = k \mid x) = \frac{\Pr(d(x) = k)}{\Pr(x)} \prod_{a \in A} \Pr((a = a(x)) \mid d(x) = k) \tag{7.17}$$

incorporating some kind of loss information, and hence emulates the more general Equation 7.19.

Domingos [45] proposes a method for using the loss function L together with a bagging procedure to relabel the training examples in such a way that the relabeled set can be trained by a procedure that focuses on minimizing error rate. The error rate of the relabeled data is then hopefully similar to the risk of the original data. An obvious advantage of such a "wrapper" approach is that the actual model induction technique does not have to be modified or tailored to take loss information into account.

The Area Under the ROC Curve

The AUC value derived from an ROC curve is a measure of a model's discriminatory performance. Collapsing a full ROC curve into a single-number statistic necessarily results in a loss of distinction of some kind. For example, several different curves may have the same AUC value, and a focus on the AUC alone glosses over the fact that the classifiers may both be optimal under different operating conditions. Still, the AUC is generally accepted as the best ROC-derived single-number statistic to use for performance assessment.[5]

The Brier Score

The *Brier score* [20] is defined as the average squared difference between the classifier's raw output value $\phi(x)$ and the actual binary decision value $d(x)$. The Brier score is also referred to as the *probability score*.

$$B = \frac{1}{|U|} \sum_{x \in U} (\phi(x) - d(x))^2 \tag{7.20}$$

The Brier score B carries information about both calibration and discrimination, and can be decomposed in several ways. A popular decomposition is given in Equation 7.21, usually attributed to Murphy [128]. The term $\bar{d}$ denotes the average actual outcome of objects in U, while CI and DI denote indices of calibration and discrimination respectively.

$$B = \bar{d}(1 - \bar{d}) - \mathrm{CI} + \mathrm{DI} \tag{7.21}$$

Arkes et al. [9] discuss the Murphy decomposition, and propose an alternative decomposition of the Brier score that lends itself nicely to graphical visualization in a *covariance graph*. Their decomposition involves readily interpretable components.

[5] However, the unquestioned use of the area under ROC curves is not without its critics. Hilden [81] argues that, for clinicians, the c-index or AUC may give "the right answer to the wrong question" since they are in practice almost never asked to decide which of two cases is diseased and which is non-diseased.

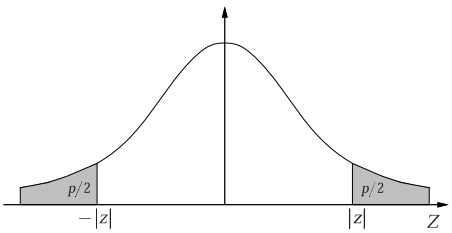

Figure 7.5: Statistical hypothesis testing (two-sided). Here, the test statistic Z is assumed to follow a standard Normal distribution under the null hypothesis H_0 that the two observed performance estimates are equal. For a two-sided test, the alternative hypothesis H_1 is that the two observed performance estimates are not equal. If we observe a realization z of Z that deviates significantly from 0, we reject H_0 and conclude that the two performance estimates are indeed different. The p-value indicates the probability that we erroneously reject H_0, i.e., the probability of a type I error.

7.4.6 Statistical Hypothesis Testing

In typical classifier comparison experiments, two or more classifiers are trained on the same training set and applied to the same test set, and a performance measure if each classifier is estimated. But if one classifier results in one performance estimate and another classifier results in another performance estimate, how much must the two estimates differ before one can say that the performances are significantly different in a statistical sense? Any such claim should always be accompanied by a *p-value* that indicates how statistically valid the claim is.

The starting point for performing hypothesis testing is a test statistic that under some null hypothesis H_0 has a known distribution. For example, H_0 could be that two performance estimates really are equal, with the alternative hypothesis H_1 being that they are different. The test statistic Z is a stochastic variable, and from our observations we can compute a realization z of Z. If z has a value that under H_0 would be very unlikely to observe, then we are prone to reject H_0. A p-value denotes the probability that we erroneously reject H_0.[6] Obviously, we would like this probability p to be small before we conclude that H_0 is to be rejected, typically $p < 0.05$ by convention. Figure 7.5 displays p-values graphically. A brief discussion of the use of p-values in decision making can be found in [239].

This section reviews a handful of statistical tests that can be used to compare discrimination and calibration between pairs of binary classifiers.

McNemar's Test

With two classifiers κ_1 and κ_2 applied to the same set of labeled objects using fixed suitable threshold values τ_1 and τ_2 for θ for each classifier, *McNemar's test* is the ap-

[6] The situation of erroneously rejecting H_0 is called a *type I error* in the statistical literature. The other situation, where we erroneously accept H_0, is called a *type II error*.

propriate statistical test to perform to detect statistically significant differences in accuracy [8, 43, 44, 58, 179, 183].

McNemar's test counts the number of objects that κ_1 classified wrongly but that κ_2 classified correctly, and the number of objects that κ_1 classified correctly but that κ_2 classified wrongly. The following small table is then obtained:

		$\hat{d}_{\kappa_1}$: error	$\hat{d}_{\kappa_1}$: correct
$\hat{d}_{\kappa_2}$	error	n_a	n_b
	correct	n_c	n_d

Under the null hypothesis H_0 that there is no difference between the accuracies of κ_1 and κ_2, we would expect n_b and n_c to be equal. An exact[7] test can be made since then n_b would follow a Binomial distribution with parameters $(n_b + n_c, \frac{1}{2})$. However, such a distribution can be very well approximated with the more easily manageable Normal distribution, even for quite small samples. McNemar's test uses the continuity-corrected Z statistic below, which under H_0 follows a standard Normal distribution. In some texts the χ^2_1-distributed quantity Z^2 is employed instead.

$$Z = \frac{|n_b - n_c| - 1}{\sqrt{n_b + n_c}} \sim N(0, 1) \tag{7.22}$$

Computing the observed value for Z enables us to perform a hypothesis test in the usual manner.

Altman [8] discusses McNemar's test in greater detail, and Salzberg [183] recommends that McNemar's test (or the exact Binomial variant on that test) should always be used when comparing the accuracy of classification algorithms.

Hanley-McNeil's Test

To statistically compare AUC values of two ROC curves derived from the same set of cases, *Hanley-McNeil's test* can be used.

Computations of the AUC of a classifier should always be accompanied by an estimate of the variability of that estimate. Hanley and McNeil [75] provide the following formula for the standard error $SE(AUC)$ of the AUC estimate computed using the trapezoidal rule of integration:

$$SE(AUC) = \sqrt{\frac{1}{|X_0||X_1|}} \sqrt{\begin{array}{l} AUC(1 - AUC) + \\ (|X_1| - 1)(Q_1 - AUC^2) + \\ (|X_0| - 1)(Q_2 - AUC^2) \end{array}} \tag{7.23}$$

[7] An exact test has precisely the distribution claimed under H_0.

The quantities Q_1 and Q_2 in Equation 7.23 are distribution-specific quantities. Hanley and McNeil claim that very good approximations can be obtained by Equation 7.24, and they also outline a more complex and non-parametric way of computing $SE(AUC)$.

$$Q_1 = \frac{AUC}{2 - AUC} \qquad Q_2 = \frac{2AUC^2}{1 + AUC} \tag{7.24}$$

Using Equation 7.23, one can compute the smallest sample size one needs for obtaining a sufficient degree of statistical precision.

When we want to test the hypothesis that two estimated AUC values derived from the same cases are really different, we have to take into account that the AUC estimates are correlated since the data material is the same. If we know that one AUC value for one classifier is very low, this affects our knowledge of the AUC estimates for the other classifiers. Hanley and McNeil [76] provide a framework for statistically comparing two different AUC values derived from the same cases. The main contribution of their paper is a tabulation of values of the correlation quantity r in Equation 7.25, for different values of AUC_1, AUC_2 and the computed correlations[8] between the two classifiers' raw output values $\phi(x)$ for objects in the two decision classes X_0 and X_1.

$$SE(AUC_1 - AUC_2) = \sqrt{\begin{array}{l} SE^2(AUC_1) + \\ SE^2(AUC_2) - \\ 2rSE(AUC_1)SE(AUC_2) \end{array}} \tag{7.25}$$

The perhaps most important purpose of assessing the variability of the observed AUC differences, is so that we can construct a statistic to use for hypothesis testing. With the above definition of $SE(AUC_1 - AUC_2)$, Hanley and McNeil use the statistic Z defined below.

$$Z = \frac{AUC_1 - AUC_2}{SE(AUC_1 - AUC_2)} \sim N(0, 1) \tag{7.26}$$

Under the null hypothesis H_0 that the two ROC areas really are equal (and that the true distribution actually is Normal), the test statistic Z follows a standard Normal distribution. We can then reject H_0, i.e., conclude that the two ROC areas are not equal, if our observed value for Z is such that it does not lie within the allowed range we set up according to our desired level of significance.

Quantitative Assessment of Calibration

Calibration is usually only assessed qualitatively by visually inspecting calibration plots or covariance graphs, but quantitative assessment of calibration is also possible.

[8] Computed using either Pearson's product-moment correlation or Kendall's tau. See Hanley and McNeil [76] for details.

However, methods for this have often originally been developed with logistic regression models in mind, and carrying such statistics directly over to other model types may yield unpleasant surprises in applied research. For instance, undefined division-by-zero situations may occur. This because formulae may involve dividing by $\phi(x)$ or its complement, and logistic regression models can never output identically 0 or 1 but approach these extreme values asymptotically. For other model paradigms such as rule sets or decision trees, these extreme values are indeed possible outputs.

Hosmer-Lemeshow's test [83], originally intended for use with logistic regression models, constructs a statistic from the points in the calibration plot.[9] Simulations indicate that this statistic is well approximated by a χ^2-distribution with $g-2$ degrees of freedom under the null hypothesis that the model is appropriate calibration-wise. The Hosmer-Lemeshow statistic can thus be used to assess quantitatively whether or not the points in the calibration plot deviate from the 45-degree line through $(0,0)$ in a statistically significant fashion.

However, there are several pitfalls with using Hosmer-Lemeshow's test other than the ones already described. The statistic is only applicable if the groups sizes are above 5, and is undefined if the sum of model outputs within a group is zero. Furthermore, if the model outputs tend to be either small (< 0.1) or large (> 0.9), then the test should be used with caution [84].

Brier Score Tests

Statistics for assessing Brier scores often suffer from the same drawback as the Hosmer-Lemeshow statistic, namely that they have been developed with methods in mind that cannot output identically 0 or 1 but approach these extreme values asymptotically. However, if a model does not produce such extreme values on an analyzed dataset, such statistics may be useful after all. Bloch [16] reviews several statistics that can be used to test whether the actual outcomes $d(x)$ are compatible with the set of model outputs $\phi(x)$. Probabilities of some events must be explicitly chosen or known to use such approaches, however.

Statistics for comparing two Brier scores obtained from the same set of cases against each other exist, enabling one to test whether or not one model's Brier score is better than the Brier score of another model. Bloch [16] and Redelmeier et al. [178] discuss such statistics and the corresponding tests in detail.

Miscellaneous

Both Hanley-McNeil's and McNemar's tests assume that the two classifiers that are to be statistically compared have been applied to the same set of cases, and that we have their corresponding observed outputs available. This might not always be the

[9] Actually, the Hosmer-Lemeshow statistic operates on the groups sums instead of on the group averages.

situation, and if the datasets only partially overlap the methods are not applicable. Metz et al. [121] propose an algorithm that allows an ROC analysis to be done, even with only partially-paired data.

Other approaches, including non-parametric methods, than the one described for comparing correlated AUC estimates are possible, although such methods are less simple and immediate. A non-parametric alternative to Hanley-McNeil's test is provided by DeLong et al. [40].

AUC estimates are uncertain since the computed points on the ROC curve have an inherent degree of uncertainty associated with them. Another approach to ascertaining the uncertainty of the AUC estimate is to compute confidence intervals for the sensitivities and specificities, and thus obtain a "confidence envelope" for the ROC curve, i.e., a pair of curves that surround the ROC curve from above and below for which we can be certain "enough" that the true ROC curve lies within. Bounds for the AUC can thus be assessed via the bounds of the ROC curve. Computing confidence bounds for ROC curves is explored by Schäfer [184].

ROC analysis for more than two outcome classes can be imagined. Then, in the three-class case, an ROC curve would generalize to an ROC surface, with the volume under this surface corresponding to the AUC. This is further explored by, e.g., Dreiseitl et al. [48].

Lastly, it should be pointed out that combining statistical hypothesis testing with systematic partitioning methods as described in Section 7.2.1 is not entirely straightforward and should be done with caution. One reason for this is that the training and/or testing sets for the splits may partially overlap, thus introducing dependencies across the statistics computed per split. Dietterich [43, 44] analyzes this scenario for some popular statistics for comparing error rates, and shows through simulations that erroneous conclusions are likely to be drawn if inter-split dependencies are ignored. Dietterich also proposes a statistic that may be used for combining error rate comparisons with systematic partitioning methods, based on doing 2-fold CV five times. Alpaydin [7] improves this statistic further.