Model Selection and Evaluation

Table of Contents

Z.-H. Zhou, *Machine Learning*,
https://doi.org/10.1007/978-981-15-1967-3_2

2.1 Empirical Error and Overfitting

In general, the proportion of incorrectly classified samples to the total number of samples is called *error rate*, that is, if a out of m samples are misclassified, then the error rate is $E = a/m$. Accordingly, $1 - a/m$ is called *accuracy*, i.e., accuracy $= 1 -$ error rate. More generally, the difference between the output predicted by the learner and the ground-truth output is called *error*. The error calculated on the training set is called *training error* or *empirical error*, and the error calculated on the new samples is called *generalization error*. Clearly, we wish to have a learner with a small generalization error. However, since the details of the new samples are unknown during the training phase, we can only try to minimize the empirical error in practice. Quite often, we obtain learners that perform well on the training set with a small or even zero empirical error, that is, 100% accuracy. However, are they the learners we need? Unfortunately, such learners are not good in most cases.

Accuracy is often expressed as percentages: $(1 - \frac{a}{m}) \times 100\%$.

Here, the "error" refers to the expectation of errors.

Later chapters will introduce different learning algorithms for minimizing the empirical error.

The good learners we are looking for are those performing well on the new samples. Hence, good learners should learn general rules from the training examples such that the learned rules apply to all potential samples. However, when the learner learns the training examples "too well", it is likely that some peculiarities of the training examples are taken as general properties that all potential samples will have, resulting in a reduction in generalization performance. In machine learning, this phenomenon is known as *overfitting*, and the opposite is known as *underfitting*, that is, the learner failed to learn the general properties of training examples. ◘ Figure 2.1 illustrates the difference between overfitting and underfitting.

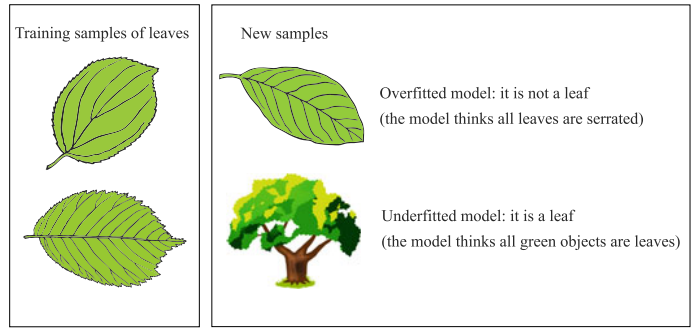


Fig. 2.1 An intuitive analogy of overfitting and underfitting

Among many possible reasons, the overly strong learning ability is a common cause for overfitting since such learners can learn the non-general peculiarities of training examples. By contrast, underfitting is usually due to weak learning ability. In practice, underfitting is relatively easy to overcome. For example, we can do more branching in decision tree learning or adding more training epochs in neural network learning. However, as we will see later, overfitting is a fundamental difficulty in machine learning, and almost every learning algorithm has implemented some mechanisms to deal with overfitting. Nevertheless, we should realize that overfitting is unavoidable, and all we can do is to alleviate or reduce the risk of it. This argument can be briefly justified as follows. Machine learning problems are often NP-hard or even harder, but practical learning algorithms have to finish learning within polynomial time. Hence, if overfitting is avoidable, then minimizing the empirical error will lead to the optimal solution, and therefore we have a constructive proof of P=NP. In other words, overfitting is unavoidable as long as we believe in P≠NP.

In practice, there are often multiple candidate learning algorithms, and even the same learning algorithm may produce different models under different parameter settings. Then, which learning algorithm should we choose, and which parameter settings should we use? This problem is referred to as *model selection*. The ideal solution is to evaluate all candidate models and select the one with the smallest generalization error. However, as mentioned earlier, we cannot obtain the generalization error directly, while the empirical error suffers from overfitting. So, how can we evaluate and select models in practice?

2.2 Evaluation Methods

In general, we can evaluate the generalization error through testing experiments. To do so, we use a *testing set* to estimate the learner's ability to classify the new samples, and use the *testing error* as an approximation to the generalization error. Generally, we assume that the testing samples are independent and identically sampled from the ground-truth sample distribution. Note that the testing set and the training set should be mutually exclusive as much as possible, that is, testing samples should avoid appearing in the training set or be used anyhow in the training process.

Here, we only consider the generalization error, but in real-world applications, we often consider more factors such as computational cost, memory cost, and interpretability.

Why should testing samples avoid appearing in the training set? To understand this, let us consider the following scenario. Suppose we use the same set of ten questions for both the exercise and exam, then does the exam reflect students' learning outcomes? The answer is "no" because some students can get

good grades even if they only know how to solve those ten questions. Analogously, the generalization ability we wish the model to have is the same as we want students to study and master the knowledge. Accordingly, the training examples correspond to the exercises, and the testing samples correspond to the exam. Hence, the estimation could be too optimistic if the testing samples are already seen in the training process.

However, given the only data set of m samples $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$, how can we do both training and testing? The answer is to produce both a training set S and a testing set T from the data set D. We discuss a few commonly used methods as follows.

2.2.1 Hold-Out

The *hold-out* method splits the data set D into two disjoint subsets: one as the training set S and the other as the testing set T, where $D = S \cup T$ and $S \cap T = \varnothing$. We train a model on the training set S and then calculate the testing error on the testing set T as an estimation of the generalization error.

Taking binary classification problems as an example, let D be a data set with 1000 samples, and we split it into a training set S with 700 samples and a testing set T with 300 samples. After being trained on S, suppose the model misclassified 90 samples on T, then we have the error rate $(90/300) \times 100\% = 30\%$, and accordingly, the accuracy $1 - 30\% = 70\%$.

It is worth mentioning that the splitting should maintain the original data distribution to avoid introducing additional bias. Taking classification problems as an example, we should try to preserve the class ratio in different subsets, and the sampling methods that maintain the class ratio are called *stratified sampling*. For example, suppose we have a data set D containing 500 positive examples and 500 negative examples, and we wish to split it into a training set S with 70% of the examples and a testing set T with 30% of the examples. Then, a stratified sampling method will ensure that S contains 350 positive examples and 350 negative examples, and T contains 150 positive examples and 150 negative examples. Without stratified sampling, the different class ratios in S and T can lead to biased error estimation since the data distributions are changed.

See Exercise 2.1.

However, even if the class ratios match, there still exist different ways of splitting the original data set D. For example, we can sort the samples in D and then use the first 350 samples for training with the rest for testing. Different ways of splitting will result in different training and testing sets, and accordingly, different model evaluation results. Therefore, a single trial of hold-out testing usually leads to unreliable error estimation. In

practice, we often perform the hold-out testing multiple times, where each trial splits the data randomly, and we use the average error as the final estimation. For example, we can randomly split the data set 100 times to produce 100 evaluation results and then take the average as the hold-out error estimation.

The hold-out method splits D into a training set and a testing set, but the model we wish to evaluate is the one trained on D. Hence, we have a dilemma. If we place most samples in the training set S, then the trained model is an excellent approximation to the model trained on D. However, the evaluation is less reliable due to the small size of T. On the other hand, if we place more samples in the testing set T, then the difference between the model trained on S and the model trained on D becomes substantial, that is, the fidelity of evaluation becomes lower. There is no perfect solution to this dilemma, and we must make a trade-off. One routine is to use around 2/3 to 4/5 of the examples for training and the rest for testing.

We can also check other statistical quantities such as standard deviation.

The dilemma can be explained with *bias-variance* which will be discussed in Sect. 2.5. The variance of the evaluation result is large when the testing set is small, and the bias of the evaluation result is large when the training set is small.

Generally speaking, a testing set should contain at least 30 samples (Mitchell 1997).

2.2.2 Cross-Validation

Cross-validation splits data set D into k disjoint subsets with similar sizes, that is, $D = D_1 \cup D_2 \cup \cdots \cup D_k$, $D_i \cap D_j = \varnothing (i \neq j)$. Typically, each subset D_i tries to maintain the original data distribution via stratified sampling. In each trial of cross-validation, we use the union of $k-1$ subsets as the training set to train a model and then use the remaining subset as the testing set to evaluate the model. We repeat this process k times and use each subset as the testing set precisely once. Finally, we average over k trials to obtain the evaluation result. Since the stability and fidelity of cross-validation largely depend on the value of k, it is also known as k-fold cross-validation. The most commonly used value of k is 10, and the corresponding method is called 10-fold cross-validation. Other common values of k include 5 and 20. ◘ Figure 2.2 illustrates the idea of 10-fold cross-validation.

There are special cases, such as Leave-One-Out, which will be discussed shortly.

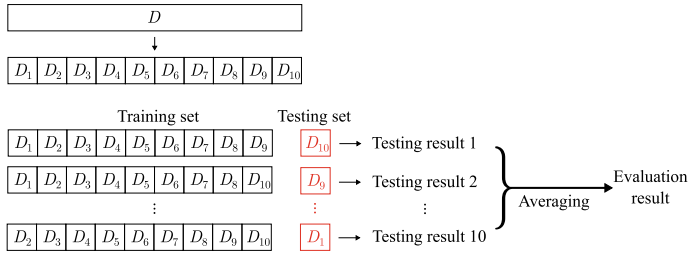


Fig. 2.2 10-fold cross-validation

Like hold-out, there are different ways of splitting the data set D into k subsets. To decrease the error introduced by splitting, we often repeat the random splitting p times and average the evaluation results of p times of k-fold cross-validation. For example, a common case is 10-time 10-fold cross-validation.

Both "10-time 10-fold cross-validation" and "100-time hold-out" run 100 evaluation experiments.

For a data set D with m samples, a special case of cross-validation is Leave-One-Out (LOO), which lets $k = m$. In such a case, the random splitting does not matter since there is only one way of splitting the m samples into m subsets. In LOO, each subset contains a single sample, and the training set is only one sample less than the original data set D; thus in most cases, the evaluation from LOO is very close to the ideal evaluation of training the model on D. Therefore, the results of LOO evaluations are often considered accurate. However, LOO has a flaw that the computational cost of training m models could be prohibitive for large data sets (e.g., 1 million samples imply 1 million models), and it can be even worse if we take parameter tuning into consideration. Besides, LOO is not necessarily more accurate than other evaluation methods since the NFL theorem also applies to evaluation methods.

See Exercise 2.2.

See Sect. 1.4 for the NFL theorem.

2.2.3 Bootstrapping

What we want to evaluate is the model trained with D. However, no matter we use hold-out or cross-validation, the training set is always smaller than D. Hence, the estimation bias is unavoidable due to the size difference between the training set and D. We can reduce the bias by using LOO, but its computational complexity is often prohibitive. However, is it possible to reduce the impact of the small training set while still be computational efficient?

See Chap. 12 for more information about the relationship between the complexity of samples and the generalization ability.

One solution is *bootstrapping*, which employs the bootstrap sampling technique (Efron and Tibshirani 1993). Given a data set D containing m samples, bootstrapping samples a data set D' by randomly picking one sample from D, copying it to D', and then placing it back to D so that it still has a chance to be picked next time. Repeating this process m times results in the bootstrap sampling data set D' containing m samples. Due to replacement, some samples in D may not appear in D', while others may appear more than once. Let us do a quick estimation: the chance of not being picked in m rounds is $(1 - \frac{1}{m})^m$, and hence taking the limit gives

The original meaning of bootstrap is to remove the strap of boots. The term comes from a story in the eighteenth century book *Baron Munchausen's Narrative of his Marvellous Travels and Campaigns in Russia*, in which Baron Munchausen pulls himself out of a swamp with his straps. Bootstrapping is also called *repeatable sampling* or *sampling with replacement*.

e is Euler's number.

$$\lim_{m\to\infty}\left(1-\frac{1}{m}\right)^m=\frac{1}{e}\approx 0.368, \tag{2.1}$$

which means that roughly 36.8% of the original samples do not appear in the data set D'. Then, we can use D' as the training set and $D \backslash D'$ as the testing set such that both the evaluated model and the actual model that we wish to evaluate on D are using m training examples. Besides, we still have a separate testing set containing about 1/3 of the original examples that are not used for training. The evaluation result obtained via this approach is called *out-of-bag estimate*.

"\" is the subtraction of sets.

Bootstrapping is particularly useful when the data set is small, or when there is no effective way of splitting training and testing sets. Besides, bootstrapping can create multiple data sets, which can be useful for methods such as ensemble learning. Nevertheless, since the original data distribution has changed by bootstrapping, the estimation is also biased. Therefore, when we have abundant data, hold-out and cross-validation are often used instead.

See Chap. 8 for ensemble learning.

2.2.4 Parameter Tuning and Final Model

Most learning algorithms have parameters to set, and different parameter settings often lead to models with significantly different performance. Hence, the model evaluation and selection is not just about selecting the learning algorithms but also about the configuration of parameters. The process of finding the right parameters is called *parameter tuning*.

Machine learning typically involves two types of parameters. The first one is the algorithm parameters, also known as *hyper-parameters*, which are usually less than 10. The other one is the model parameters, which can be many, e.g., large-scale deep learning models can have more than 10 billion parameters. Both types of parameters are tuned similarly, that is, one generates candidate models and then selects via an evaluation method. The difference is that hyper-parameters are usually configured manually, whereas candidate models are generated by learning, e.g., parameters of neural networks that stop training at different iterations.

Readers may think there is no essential difference between parameter tuning and algorithm selection: each parameter setting leads to one model, and we select the one that produces the best results as the final model. This idea is basically sound; however, there is one issue: since parameters are often real-valued, it is impossible to try all parameter settings. Therefore, in practice, we usually set a range and a step size for each parameter, e.g., a range of [0, 0.2] and a step size of 0.05, which lead to only five candidate parameter settings. Such a trade-off between computational cost and quality of estimation makes the learning feasible, though the selected parameter setting is usually not optimal. In reality, even after making such a trade-off, parameter tuning can still be quite challenging. We can make a simple estimation. Suppose that the algorithm has three parameters and each considers only five candidate values, then we need to assess $5^3 = 125$ models for each pair of training and testing sets. Powerful learning algorithms often have quite many parameters to be configured, resulting in a heavy workload of parameter tuning. The quality of parameter tuning is often vital in real-world applications.

We should note that the training process does not use all data since part of the data is hold-out for model evaluation and selection. Therefore, after we have determined the algorithm and parameters via model selection, the entire data set should be used to re-train a model as the final delivery.

Last but not least, we should distinguish the data used for model selection from the testing data encountered after model selection. We often call the data set used in the model selection a *validation set*. For example, we may split data into a training set for training models, a validation set for model selection and parameter tuning, and a testing set for estimating the generalization ability of models.

2.3 Performance Measure

In order to evaluate the generalization ability of models, we need not only practical and effective estimation methods but also some performance measures that can quantify the generalization ability. Different performance measures reflect the varied demands of tasks and produce different evaluation results. In other words, the quality of a model is a relative concept that depends on the algorithm and data as well as the task requirement.

See Chap. 9 for the performance measures of clustering.

In prediction problems, we are given a data set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$, where y_i is the ground-truth label of the sample $\boldsymbol{x}_i$. To evaluate the performance of a learner f, we compare its prediction $f(\boldsymbol{x})$ to the ground-truth label y.

For regression problems, the most commonly used performance measure is the Mean Squared Error (MSE):

$$E(f; D) = \frac{1}{m}\sum_{i=1}^{m}(f(\boldsymbol{x}_i) - y_i)^2. \tag{2.2}$$

More generally, for a data distribution $\mathcal{D}$ and a probability density function $p(\cdot)$, the MSE is written as

$$E(f; \mathcal{D}) = \int_{\boldsymbol{x}\sim D}(f(\boldsymbol{x}) - y)^2 p(\boldsymbol{x})d\boldsymbol{x}. \tag{2.3}$$

The rest of this section will introduce some common performance measures for classification problems.

2.3.1 Error Rate and Accuracy

At the beginning of this chapter, we discussed error rate and accuracy, which are the most commonly used performance

measures in classification problems, including both binary classification and multiclass classification. Error rate is the proportion of misclassified samples to all samples, whereas accuracy is the proportion of correctly classified samples instead. Given a data set D, we define error rate as

$$E(f; D) = \frac{1}{m}\sum_{i=1}^{m}\mathbb{I}(f(\boldsymbol{x}_i) \neq y_i), \tag{2.4}$$

and accuracy as

$$\begin{aligned}\mathrm{acc}(f; D) &= \frac{1}{m}\sum_{i=1}^{m}\mathbb{I}(f(\boldsymbol{x}_i) = y_i) \\ &= 1 - E(f; D).\end{aligned} \tag{2.5}$$

More generally, for a data distribution $\mathcal{D}$ and a probability density function $p(\cdot)$, error rate and accuracy can be, respectively, written as

$$E(f; \mathcal{D}) = \int_{\boldsymbol{x}\sim D}\mathbb{I}(f(\boldsymbol{x}) \neq y)p(\boldsymbol{x})d\boldsymbol{x}, \tag{2.6}$$

$$\begin{aligned}\mathrm{acc}(f; \mathcal{D}) &= \int_{\boldsymbol{x}\sim D}\mathbb{I}(f(\boldsymbol{x}) = y)p(\boldsymbol{x})d\boldsymbol{x} \\ &= 1 - E(f; \mathcal{D}).\end{aligned} \tag{2.7}$$

2.3.2 Precision, Recall, and *F*1

Error rate and accuracy are frequently used, but they are not suitable for all tasks. Taking our watermelon problem as an example, suppose we use a learned model to classify a new batch of watermelons. The error rate tells us the proportion of misclassified watermelons to all watermelons in this batch. However, we may want to know "What percentage of the picked watermelons are ripe?" or "What percentage of all ripe watermelons were picked out?" Unfortunately, the error rate is unable to answer such questions, and hence we need other performance measures.

Such questions often arise in applications like information retrieval and web search. For example, in information retrieval, we often want to know "What percentage of the retrieved information is of interest to users?" and "How much of the information the user is interested in is retrieved?" For such questions, *precision* and *recall* are better choices.

In binary classification problems, there are four combinations of the ground-truth class and the predicted class, namely true positive, false positive, true negative, and false negative,

and we denote the number of samples in each case as TP, FP, TN, and FN, respectively. Then, $TP + FP + TN + FN =$ total number of samples. The four combinations can be displayed in a *confusion matrix*, as shown in ◘ Table 2.1. Then, the precision P and the recall R are, respectively, defined as

$$P = \frac{TP}{TP + FP}, \tag{2.8}$$

$$R = \frac{TP}{TP + FN}. \tag{2.9}$$

◘ **Tab. 2.1** The confusion matrix of binary classification

Ground-truth class	Predicted class	
	Positive	Negative
Positive	*TP*	*FN*
Negative	*FP*	*TN*

Precision and recall are contradictory. Generally speaking, the recall is often low when the precision is high, and the precision is often low when the recall is high. For example, to pick more ripe watermelons, we can increase the number of picked watermelons because, in an extreme case, if we pick all watermelons, then all ripe watermelons are picked as well. However, by doing so, the precision would be very low. On the other hand, if we wish the proportion of ripe watermelons to be high, then we should only pick watermelons that we are sure of. However, doing so could miss many ripe watermelons, and hence the recall becomes low. Typically, we can achieve high precision and high recall at the same time only in simple problems.

Quite often, we can use the learner's predictions to sort the samples by how likely they are positive. That is, the samples that are most likely to be positive are at the top of the ranking list, and the samples that are least likely to be positive are at the bottom. Starting from the top of the ranking list, we can incrementally label the samples as positive to calculate the precision and recall at each increment. Then, plotting the precisions as y-axis and the recalls as x-axis gives the Precision-Recall Curve (P-R curve). The plots of P-R curves are called P-R plots. ◘ Figure 2.3 gives an example of P-R curve.

Taking information retrieval as an example, the precision and recall can be calculated by sequentially returning each piece of information that the user might be interested in.

Also called *PR curve* or *PR plot*.

P-R plots intuitively show the overall precision and recall of learners. When comparing two learners, if the P-R curve of one learner entirely encloses the curve of another learner, then the performance of the first learner is superior. For example, in ◘ Figure 2.3, learner A is better than learner C. However, when the P-R curves intersect, such as curve A and curve B, we cannot

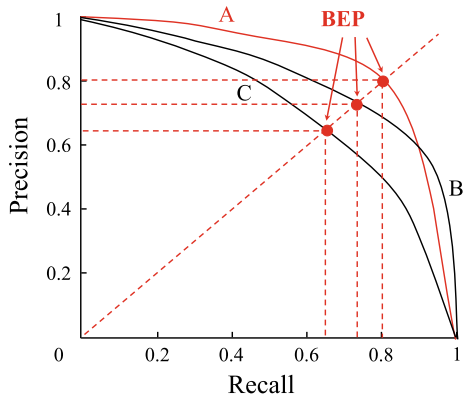


Fig. 2.3 P-R curve and break-even points

say which learner is generally better and can only compare them at a specific precision or recall. Nevertheless, people often insist on finding out the best learner even if there exist intersections. A reasonable solution is to compare the areas under the P-R curves, which, to some extent, represent the proportion of cases when both precision and recall are relatively high. However, the areas are not easy to compute, and hence we often seek alternative performance measures that consider precision and recall simultaneously.

One alternative is Break-Even Point (BEP), which is the value when precision and recall are equal. For example, in ◘ Figure 2.3, the BEP of learner C is 0.64, and learner A is better than learner B, according to the BEP.

However, BEP could be oversimplified, and a more commonly used alternative is $F1$-measure:

$$F1 = \frac{2 \times P \times R}{P + R} = \frac{2 \times TP}{\text{total number of samples} + TP - TN}. \tag{2.10}$$

$F1$ is the harmonic mean of precision and recall: $\frac{1}{F1} = \frac{1}{2} \cdot \left(\frac{1}{P} + \frac{1}{R}\right)$.

In some applications, the importance of precision and recall are different. For example, precision is more critical in recommender systems since it is more desirable that the recommended content is of interest to the user and disturbs the user as little as possible. On the other hand, recall is more critical in criminal information retrieval systems since we wish to miss as few criminals as possible. The general form of $F1$-measure is F_β, which allows us to specify our preference over precision and recall, and is defined as

F_β is the weighted harmonic mean: $\frac{1}{F_\beta} = \frac{1}{1+\beta^2} \cdot \left(\frac{1}{P} + \frac{\beta^2}{R}\right)$.

The Harmonic mean emphasizes more on smaller values compared to the arithmetic mean ($\frac{P+R}{2}$) and the geometric mean ($\sqrt{P \times R}$).

$$F_\beta = \frac{(1+\beta^2) \times P \times R}{(\beta^2 \times P) + R}, \tag{2.11}$$

where $\beta > 0$ gives the relative importance of recall to precision (Van Rijsbergen 1979). When $\beta = 1$, it reduces to the standard $F1$; when $\beta > 1$, recall is more important; when $\beta < 1$, precision is more important.

Sometimes we may have multiple confusion matrices in binary classification problems. For example, there is one confusion matrix for each round of training and testing. Also, there are multiple confusion matrices when we do training and testing on multiple data sets to estimate the overall performance. Besides, there is one confusion matrix for every class in multiclass classification problems. In all of these cases, we need to investigate the overall precision and recall on n binary confusion matrices.

A straightforward approach is to calculate the precision and the recall for each confusion matrix, denoted by (P_1, R_1), $(P_2, R_2), \ldots, (P_n, R_n)$. By taking the averages, we have the macro-P, the macro-R, and the macro-$F1$:

$$\text{macro-}P = \frac{1}{n}\sum_{i=1}^{n} P_i, \tag{2.12}$$

$$\text{macro-}R = \frac{1}{n}\sum_{i=1}^{n} R_i, \tag{2.13}$$

$$\text{macro-}F1 = \frac{2 \times \text{macro-}P \times \text{macro-}R}{\text{macro-}P + \text{macro-}R}. \tag{2.14}$$

We can also calculate element-wise averages across the confusion matrices to get $\overline{TP}$, $\overline{FP}$, $\overline{TN}$, $\overline{FN}$, and then take the averages to obtain the micro-P, the micro-R, and the micro-$F1$:

$$\text{micro-}P = \frac{\overline{TP}}{\overline{TP} + \overline{FP}}, \tag{2.15}$$

$$\text{micro-}R = \frac{\overline{TP}}{\overline{TP} + \overline{FN}}, \tag{2.16}$$

$$\text{micro-}F1 = \frac{2 \times \text{micro-}P \times \text{micro-}R}{\text{micro-}P + \text{micro-}R}. \tag{2.17}$$

2.3.3 ROC and AUC

Since the predictions from learners are often in the form of real values or probabilities, we can compare the predicted values against a classification threshold, that is, classify a sample as positive if the prediction value is greater than the threshold and

classify it as negative otherwise. For example, typical neural networks predict real values in the interval [0.0, 1.0] for testing samples. We can compare the predicted values with 0.5, and classify a sample as positive if its predicted value is greater than 0.5, and negative otherwise. Hence, the predicted real values or probabilities directly determine the generalization ability. In practice, we sort the testing samples by the predicted real values or probabilities in descending order such that potential positive samples are at the top of the list. After that, we put a *cut point* in the sorted list and classify the samples above it as positive and the rest as negative.

See Chap. 5 for neural networks.

The position of the cut point depends on the specific application. For example, we move the cut point toward the top of the list if precision is more critical than recall, and move it toward the bottom otherwise. Consequently, the ranking quality reflects the learner's "expected generalization ability" for different tasks or the generalization ability for "typical cases". The Receiver Operating Characteristics (ROC) curve follows the above idea to measure the generalization ability of learners.

The ROC curve was initially developed for radar detection of enemy aircraft in World War II and then introduced to psychology and medical applications in the 1960s–1970s. Later on, it was introduced to machine learning (Spackman 1989). Similar to the P-R curve discussed in Sect. 2.3.2, we sort the samples by the predictions and then obtain two measures by gradually moving the cut point from the top toward the bottom of the ranked list. Using those two measures as x-axis and y-axis gives the ROC curve. Unlike precision and recall in P-R curves, the y-axis in ROC curves is True Positive Rate (TPR), and the x-axis is False Positive Rate (FPR). Reusing the notations in ◘ Table 2.1, these two measures are, respectively, defined as

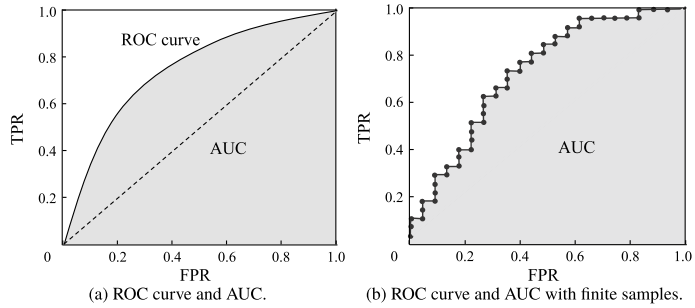
$$\text{TPR} = \frac{TP}{TP + FN}, \tag{2.18}$$

$$\text{FPR} = \frac{FP}{TN + FP}. \tag{2.19}$$

The plot showing ROC curves is called an ROC plot. ◘ Figure 2.4a gives an example of an ROC plot in which the diagonal corresponds to the "random guessing" model, and the point (0, 1) corresponds to the "ideal model" that places all positive samples before negative samples.

In practice, we only have finite pairs of (FPR, TPR) coordinates for drawing the ROC plot since the testing samples are finite. Hence, the ROC curve may not look smooth like the one in ◘ Figure 2.4a but is only an approximation, like the one shown in ◘ Figure 2.4b. The plotting process is as follows: given m^+ positive samples and m^- negative samples, we first

The same problem occurs when drawing P-R plots, but we deferred the discussion until now to facilitate the introduction of calculating AUC.



(a) ROC curve and AUC. (b) ROC curve and AUC with finite samples.

Fig. 2.4 An illustration of ROC curve and AUC

sort all samples by the learner's predictions, and then set the threshold to maximum, that is, predicting all samples as negative. At this moment, both TPR and FPR are 0, so we mark at coordinate (0, 0). Then, we gradually decrease the threshold to the predicted value of each sample along the sorted list, that is, the samples are classified as positive successively. Let (x, y) denote the previous coordinate, we put a mark at $(x, y + \frac{1}{m^+})$ if the current samples are true positive, and we put a mark at $(x + \frac{1}{m^-}, y)$ if the current samples are false positive. By connecting all adjacent marked points, we have the ROC curve.

Like P-R plots, we say learner A is better than learner B if A's ROC curve entirely encloses B's ROC curve. However, when there exist intersections, no learner is generally better than the other. One way of comparing intersected ROC curves is to calculate the areas under the ROC curves, that is, Area Under ROC Curve (AUC), as shown in ◘ Figure 2.4.

By its definition, AUC can be calculated by integrating the areas under the steps of ROC curve. Suppose that the ROC curve is obtained by sequentially connecting the points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, where $x_1 = 0$ and $x_m = 1$. Then, as illustrated in ◘ Figure 2.4b, the AUC is estimated as

$$\mathrm{AUC} = \frac{1}{2} \sum_{i=1}^{m-1} (x_{i+1} - x_i) \cdot (y_i + y_{i+1}). \tag{2.20}$$

AUC is closely related to ranking errors since it considers the ranking quality of predictions. Let m^+ denote the number of positive samples, m^- denote the number of negative samples, D^+ denote the set of positive samples, and D^- denote the set of negative samples. Then, the ranking *loss* is defined as

$$\ell_{\mathrm{rank}} = \frac{1}{m^+ m^-} \sum_{\boldsymbol{x}^+ \in D^+} \sum_{\boldsymbol{x}^- \in D^-} \left(\mathbb{I}(f(\boldsymbol{x}^+) < f(\boldsymbol{x}^-)) + \frac{1}{2} \mathbb{I}(f(\boldsymbol{x}^+) = f(\boldsymbol{x}^-)) \right). \tag{2.21}$$

For each pair of positive sample $\boldsymbol{x}^+$ and negative sample $\boldsymbol{x}^-$, the ranking loss applies a penalty of 1 if the predicted value of the positive sample is lower than that of the negative sample, and a penalty of 0.5 applies when the predicted values are equal. Suppose (x, y) is the coordinate of a positive sample on the ROC curve, then x is the proportion of negative samples ranked above this positive sample (i.e., FPR). Hence, the ranking loss ℓ_{rank} corresponds to the area above the ROC curve, that is,

$$\text{AUC} = 1 - \ell_{\text{rank}}. \tag{2.22}$$

2.3.4 Cost-Sensitive Error Rate and Cost Curve

In some problems, the consequences of making different errors are not the same. Taking medical diagnosis as an example, according to our earlier discussions, we receive the same amount of penalty for misclassifying someone as healthy or unhealthy. However, it turns out that misclassifying a sick patient as healthy is more serious since it risks the life of the patient. Another example is the access control system in which denying the access of normal users leads to unpleasant user experience while allowing intruders to enter causes security breach. In such cases, we need to assign *unequal costs* to different errors.

For binary classification problems, we can leverage domain knowledge to design a *cost matrix*, as shown in ◘ Table 2.2, where cost_{ij} represents the cost of misclassifying a sample of class i as class j. In general, $\text{cost}_{ii} = 0$, and $\text{cost}_{01} > \text{cost}_{10}$ if misclassifying class 0 as class 1 costs more than the other way around. The larger the difference between the costs is, the larger the difference between cost_{01} and cost_{10} will be.

Normally, we care more about the cost ratios rather than the absolute values, e.g., $\text{cost}_{01} : \text{cost}_{10} = 5 : 1$ is equivalent to $\text{cost}_{01} : \text{cost}_{10} = 50 : 10$.

◘ **Tab. 2.2** Cost matrix of binary classification

Ground-truth class	Predicted class	
	Class 0	Class 1
Class 0	0	cost_{01}
Class 1	cost_{10}	0

Almost all performance measures we discussed so far implicitly assumed equal-cost. For example, error rate (2.4) counts the number of errors without considering the different consequences. With unequal costs, however, we no longer minimize the counts but the *total cost*. For binary classification problems, we can call class 0 as the positive class and class 1 as the

negative class. Let D^+ and D^- denote, respectively, the set of positive samples and the set of negative samples. Then, based on ◘ Table 2.2, the *cost-sensitive* error rate is defined as

$$E(f; D; \text{cost}) = \frac{1}{m}\Bigg(\sum_{\boldsymbol{x}_i \in D^+} \mathbb{I}(f(\boldsymbol{x}_i) \neq y_i) \times \text{cost}_{01} + \sum_{\boldsymbol{x}_i \in D^-} \mathbb{I}(f(\boldsymbol{x}_i) \neq y_i) \times \text{cost}_{10} \Bigg). \tag{2.23}$$

Similarly, we can also define the distribution-based cost-sensitive error rate and the cost-sensitive version of accuracy. It is also possible to define cost-sensitive performance measures for multiclass cases by allowing i and j of cost_{ij} to take values other than 0 and 1.

With unequal costs, we find the expected total costs of learners from *cost curves* rather than ROC curves. The x-axis of cost curves is the probability cost of positive class:

See Exercise 2.7.

$$P(+)\text{cost} = \frac{p \times \text{cost}_{01}}{p \times \text{cost}_{01} + (1-p) \times \text{cost}_{10}}, \tag{2.24}$$

where $p \in [0, 1]$ is the probability of a sample being positive. The y-axis is the normalized cost which takes values from [0, 1]:

Normalization is the process of mapping values from different ranges to a fixed range, e.g., [0, 1]. See Exercise 2.8.

$$\text{cost}_{\text{norm}} = \frac{\text{FNR} \times p \times \text{cost}_{01} + \text{FPR} \times (1-p) \times \text{cost}_{10}}{p \times \text{cost}_{01} + (1-p) \times \text{cost}_{10}}, \tag{2.25}$$

where FPR is the false positive rate defined in (2.19) and $\text{FNR} = 1 - \text{TPR}$ is the false negative rate. We can draw a cost curve as follows: since every point (FPR, TPR) on the ROC curve corresponds to a line segment on the cost plane, we

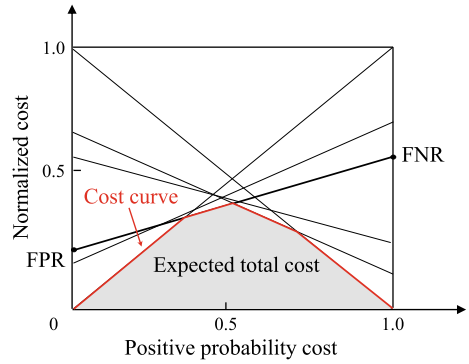


Fig. 2.5 The cost curve and expected total cost

can calculate the FNR and draw a line segment from (0, FPR) to (1, FNR). Then, the area under the line segment represents the expected total cost for the given p, FPR, and TPR. By converting all points on the ROC curve to line segments on the cost plane, the expected total cost is given by the area under the lower bound of all line segments, as shown in ◘ Figure 2.5.

2.4 Comparison Test

It seems straightforward to compare learners using evaluation methods and performance measures. For example, we use an evaluation method to measure the performance of learners and then compare them. However, how should we make the "comparison"? Should we check which of the measured values is better? Performance comparisons are indeed far more complicated than we thought due to the following reasons. Firstly, we wish to compare the generalization performance of learners, but evaluation methods only measure performance on testing sets, that is, the comparisons may not reflect the actual generalization performance. Secondly, testing performance depends on the choice of the testing set, e.g., the results on two different-sized testing sets, or two equal-sized sets but with different samples, could be different. Finally, many machine learning algorithms have some build-in random behavior, which means that we may obtain different results even for the same parameter settings and testing set. Then, what is the appropriate way of comparing the performance of learners?

Hypothesis testing is one of the techniques to compare the performance of learners. Suppose that we observe learner A outperforms learner B on a testing set. Then, hypothesis testing can help us check whether the generalization performance of learner A is better than that of learner B in the statistical sense and how significant it is. In the following discussions, we introduce two basic hypothesis tests and several methods to compare learners' performance. For ease of discussion, the rest of this section assumes error rate, denoted by ϵ, to be the default performance measure.

See Wellek (2010) for more information about hypothesis testing.

2.4.1 Hypothesis Testing

In hypothesis testing, a hypothesis is a statement or assumption about the learner's generalization error rate distribution, e.g., "$\epsilon = \epsilon_0$". In practice, however, we only have the testing error rate $\hat{\epsilon}$ but not the generalization error rate ϵ. Though $\hat{\epsilon}$ and ϵ may not be identical, they are, intuitively, likely to be close.

Hence, we can use the testing error rate distribution to infer the generalization error rate distribution.

A generalization error rate of ϵ means that the learner has a probability of ϵ to make an incorrect prediction. A testing error rate of $\hat{\epsilon}$ means that the learner misclassified $\hat{\epsilon} \times m$ samples in a testing set of m samples. Suppose the testing samples are drawn *i.i.d.* from the population distribution. Then, the probability that a learner with a generalization error rate of ϵ misclassifies m' samples and correctly classifies the rest is $\binom{m}{m'}\epsilon^{m'}(1-\epsilon)^{m-m'}$. Consequently, for a learner with a generalization error rate of ϵ, the probability of misclassifying $\hat{\epsilon} \times m$ samples, which is also the probability that the testing error rate being $\hat{\epsilon}$ on a testing set of m samples, is

$$P(\hat{\epsilon}; \epsilon) = \binom{m}{\hat{\epsilon} \times m} \epsilon^{\hat{\epsilon} \times m}(1-\epsilon)^{m-\hat{\epsilon} \times m}. \tag{2.26}$$

By solving $\partial P(\hat{\epsilon}; \epsilon)/\partial \epsilon = 0$ with the testing error rate, we observe that $P(\hat{\epsilon}; \epsilon)$ is maximized when $\epsilon = \hat{\epsilon}$, and $P(\hat{\epsilon}; \epsilon)$ decreases as $|\epsilon - \hat{\epsilon}|$ increases. The observation follows the binomial distribution, and, as shown in ◘ Figure 2.6, the learner is most likely to misclassify 3 samples out of 10 samples when $\epsilon = 0.3$.

We can use *binomial test* to verify hypotheses such as "$\epsilon \leqslant 0.3$", that is, the generalization error rate is not greater than 0.3. More generally, for the hypothesis "$\epsilon \leqslant \epsilon_0$", (2.27) gives the maximum observable error rate within a probability of $1 - \alpha$. The probability is also known as *confidence*, corresponding to the non-shaded part of ◘ Figure 2.6.

Common values of α include 0.05 and 0.1. We use a large α in ◘ Figure 2.6 for illustration purposes.

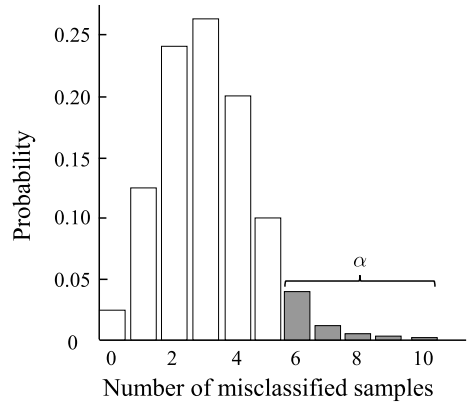


Fig. 2.6 Binomial distribution ($m = 10, \epsilon = 0.3$)

$$\bar{\epsilon} = \min \epsilon \ \text{ s.t. } \sum_{i=\epsilon \times m+1}^{m} \binom{m}{i} \epsilon_0^i (1-\epsilon_0)^{m-i} < \alpha. \tag{2.27}$$

"s.t." stands for "subject to", indicating that the expression on the right-hand side must be met while solving the expression on the left-hand side.

If the testing error rate $\hat{\epsilon}$ is greater than the critical value $\bar{\epsilon}$, then, according to the binomial test, the hypothesis "$\epsilon \leqslant \epsilon_0$" cannot be rejected at the significance level of α, that is, the learner's generalization error rate is not greater than ϵ_0 at the confidence level of $1-\alpha$; otherwise, we reject the hypothesis, that is, the learner's generalization error rate is greater than ϵ_0 at the significance level of α.

We can compute the critical value with the assistance of $\texttt{qbinom}(1-\alpha, m, \epsilon_0)$ in R or $\texttt{icdf('Binomial',}1-\alpha, m, \epsilon_0)$ in MATLAB.

We often obtain multiple testing error rates from cross-validation or by doing multiple hold-out evaluations. In such cases, we can use t-test. Let $\hat{\epsilon}_1, \hat{\epsilon}_2, \ldots, \hat{\epsilon}_k$ denote the k testing error rates, then the average testing error rate μ and variance σ^2 are, respectively,

R is an open-source scripting language for statistical computing. See
► http://www.r-project.org.

$$\mu = \frac{1}{k} \sum_{i=1}^{k} \hat{\epsilon}_i, \tag{2.28}$$

$$\sigma^2 = \frac{1}{k-1} \sum_{i=1}^{k} (\hat{\epsilon}_i - \mu)^2. \tag{2.29}$$

We can regard these k testing error rates as *i.i.d.* samples of the generalization error rate ϵ_0, and hence the variable

$$\tau_t = \frac{\sqrt{k}(\mu - \epsilon_0)}{\sigma} \tag{2.30}$$

follows a t-distribution with $k-1$ degrees of freedom, as shown in ◘ Figure 2.7.

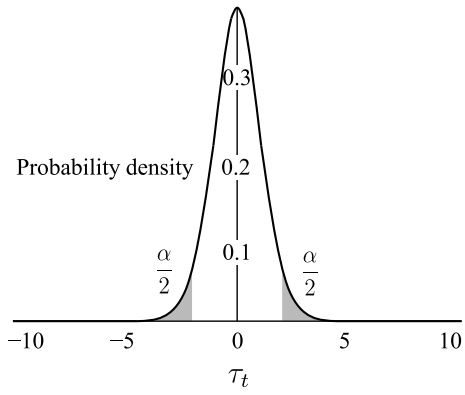


Fig. 2.7 t-distribution ($k = 10$)

For the hypothesis "$\mu = \epsilon_0$" and significance level α, we can calculate the maximum observable error rate (i.e., the critical value) within a probability of $1 - \alpha$, where ϵ_0 is the average testing error rate. Here, we employ a two-tailed hypothesis, and there are $\alpha/2$ shaded areas at both tails of the distribution, as shown in ◘ Figure 2.7. Let $(-\infty, t_{-\alpha/2}]$ and $[t_{\alpha/2}, \infty)$ denote the ranges of the two shaded areas, respectively. If τ_t is within the critical value range $[t_{-\alpha/2}, t_{\alpha/2}]$, then the hypothesis "$\mu = \epsilon_0$" cannot be rejected, that is, the generalization error rate is ϵ_0 at the confidence level of $1 - \alpha$; otherwise, we reject the hypothesis, that is, the generalization error rate is significantly different from ϵ_0 at this confidence level. 0.05 and 0.1 are commonly used significance levels, and ◘ Table 2.3 shows some commonly used critical values for t-test.

The critical values $t_{\alpha/2}$ can be computed by $\mathrm{qt}(1 - \alpha/2, k - 1)$ in R or $\mathrm{icdf}('\mathrm{T}', 1 - \alpha/2, k - 1)$ in MATLAB.

◘ **Tab. 2.3** Commonly used critical values for two-tailed t-test

α	k				
	2	5	10	20	30
0.05	12.706	2.776	2.262	2.093	2.045
0.10	6.314	2.132	1.833	1.729	1.699

Both methods introduced above compare the generalization performance of a single learner. In the following section, we discuss several hypothesis testing methods for comparing the generalization performance of multiple learners.

2.4.2 Cross-Validated t-Test

For two learners A and B, let $\epsilon_1^A, \epsilon_2^A, \ldots, \epsilon_k^A$ and $\epsilon_1^B, \epsilon_2^B, \ldots, \epsilon_k^B$ denote their testing error rates obtained from k-fold cross-validation, where i indicates the ith fold. Then, we can use k-fold cross-validated paired t-tests to compare the two learners. The basic idea is that if the performance of the two learners is the same, then the testing error rates should be the same on the same training and testing sets, that is, $\epsilon_i^A = \epsilon_i^B$.

To be specific, for the k pairs of testing error rates obtained from the k-fold cross-validation, we calculate the difference of each pair of results as $\Delta_i = \epsilon_i^A - \epsilon_i^B$. Then, the mean of the differences should be zero if the two learners have the same performance. Consequently, based on the differences $\Delta_1, \Delta_2, \ldots, \Delta_k$, we perform a t-test on the hypothesis "learner A and learner B have the same performance". We calculate the mean μ and variance σ^2 of the differences, and if

$$\tau_t = \left| \frac{\sqrt{k}\mu}{\sigma} \right| \tag{2.31}$$

is less than the critical value $t_{\alpha/2,k-1}$ at the significance level of α, then the hypothesis cannot be rejected, that is, there is no significant difference in the learners' performance; otherwise, these two learners have significantly different performance, and the one with the lower mean error rate is superior. Here, $t_{\alpha/2,k-1}$ is the critical value of a t-distribution with $k-1$ degrees of freedom and a tail of $\alpha/2$.

The above hypothesis test assumes the testing error rates are $i.i.d.$ samples of the generalization error rate. However, due to the finite training data, the training sets of different rounds are often overlapped in evaluation methods such as cross-validation. Therefore, the testing error rates are indeed not independent, resulting in an overestimated probability for the hypothesis to be true. To alleviate the problem, we can use "5×2 cross-validation" (Dietterich 1998).

As the name suggests, 5×2 cross-validation repeats two-fold cross-validation five times, where the data is randomly shuffled before each two-fold cross-validation such that the data splitting is different in the five rounds of cross-validations. For example, for two learners A and B, we obtain their testing error rates of the ith two-fold cross-validation. Then, we calculate the difference between their error rates of the first fold, denoted by Δ_i^1, and the difference between their error rates of the second fold, denoted by Δ_i^2. To alleviate the dependency of testing error rates, we calculate the variance of each two-fold cross-validation as $\sigma_i^2 = \left(\Delta_i^1 - \frac{\Delta_i^1+\Delta_i^2}{2}\right)^2 + \left(\Delta_i^2 - \frac{\Delta_i^1+\Delta_i^2}{2}\right)^2$; however, only the mean of the first two-fold cross-validation is calculated as $\mu = 0.5(\Delta_1^1 + \Delta_1^2)$. The variable

$$\tau_t = \frac{\mu}{\sqrt{0.2\sum_{i=1}^{5}\sigma_i^2}} \tag{2.32}$$

follows a t-distribution with five degrees of freedom, where its two-tailed critical value $t_{\alpha/2,5}$ is 2.5706 when $\alpha = 0.05$, and 2.0150 when $\alpha = 0.1$.

2.4.3 McNemar's Test

For binary classification problems, the hold-out method estimates not only the testing error rates of both learner A and learner B, but also the classification difference of the two learners, that is, the numbers of both correct, both incorrect, and

one correct while the other incorrect. These numbers form a *contingency table*, as shown in Table 2.4.

Tab. 2.4 The contingency table of two learners

Algorithm B	Algorithm A	
	Correct	Incorrect
Correct	e_{00}	e_{01}
Incorrect	e_{10}	e_{11}

If the performance of the two learners are the same, then we should have $e_{01} = e_{10}$. The variable $|e_{01} - e_{10}|$ follows a Gaussian distribution. McNemar's test considers the variable

$$\tau_{\chi^2} = \frac{(|e_{01} - e_{10}| - 1)^2}{e_{01} + e_{10}}, \tag{2.33}$$

Since $e_{01} + e_{10}$ is often small, we need the continuity correction, that is, -1 in the numerator.

which follows a chi-square distribution with one degree of freedom, that is, the distribution of the sum of squared standard normal random variables. At the significance level of α, the hypothesis cannot be rejected if the variable is less than the critical value χ^2_α, that is, there is no significant difference between the performance of those two learners; otherwise, the hypothesis is rejected, that is, the performance of those two learners is significantly different, and the learner with smaller average error rate is superior. The critical value of χ^2 test with one degree of freedom is 3.8415 when $\alpha = 0.05$ and 2.7055 when $\alpha = 0.1$.

The critical values χ^2_α can be computed by qchisq$(1-\alpha, k-1)$ in R or icdf('Chisquare', $1-\alpha, k-1$) in MATLAB, where $k = 2$ is the number of algorithms being compared.

2.4.4 Friedman Test and Nemenyi Post-hoc Test

Both the cross-validated t-test and McNemar's test compare two algorithms on a single data set. However, in some cases, comparisons are made for multiple algorithms on multiple data sets. In such cases, we can compare each pair of algorithms on each data set using a cross-validated t-test or a McNemar's test. Alternatively, we can use the following ranking-based Friedman test to compare all algorithms on all data sets at once.

Suppose that we are comparing algorithms A, B, and C on four data sets D_1, D_2, D_3, and D_4. We first use either hold-out or cross-validation to obtain each algorithm's testing result on each data set. Then, we sort the algorithms on each data set by their testing performance and assign the ranks 1, 2, ..., accordingly, where the algorithms with the same testing performance share the averaged rank. For example, as shown in Table 2.5, on data sets D_1 and D_3, A is the best, B is the second, and

C is the last; on data set D_2, A is the best, and B and C have the same performance. After collecting all the ranks, we calculate the average rank of each algorithm as the last row of ◘ Table 2.5.

◘ **Tab. 2.5** The ranking table of algorithms

Data set	Algorithm A	Algorithm B	Algorithm C
D_1	1	2	3
D_2	1	2.5	2.5
D_3	1	2	3
D_4	1	2	3
Average rank	1	2.125	2.875

According to the Friedman test, the algorithms with the same performance should have the same average rank. Let k denote the number of algorithms, N denote the number of data sets, and r_i denote the average rank of the ith algorithm. Here, we ignore the ties to simplify our discussion. Then, the mean and the variance of r_i are $(k+1)/2$ and $(k^2-1)/12N$, respectively. The variable

$$\begin{aligned}\tau_{\chi^2} &= \frac{k-1}{k} \cdot \frac{12N}{k^2-1} \sum_{i=1}^{k} \left(r_i - \frac{k+1}{2} \right)^2 \\ &= \frac{12N}{k(k+1)} \left(\sum_{i=1}^{k} r_i^2 - \frac{k(k+1)^2}{4} \right)\end{aligned} \tag{2.34}$$

follows a χ^2 distribution with $k-1$ degrees of freedom when k and N are large.

The "original Friedman test" described above is too conservative, and hence the following variable is often used instead:

$$\tau_F = \frac{(N-1)\tau_\chi^2}{N(k-1)-\tau_\chi^2}, \tag{2.35}$$

where τ_{χ^2} is given by (2.34). τ_F follows a F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom. ◘ Table 2.6 shows some commonly used critical values for F-test.

The "original Friedman test" requires a large k (e.g., > 30), and tends to return no significant difference when k is small.

The critical values for F-test can be computed by qf$(1-\alpha, k-1, (k-1)(N-1))$ in R or icdf('F', $1-\alpha, k-1, (k-1)*(N-1)$) in MATLAB.

Tab. 2.6 Commonly used critical values for F-test

$\alpha = 0.05$									
N	k								
	2	3	4	5	6	7	8	9	10
4	10.128	5.143	3.863	3.259	2.901	2.661	2.488	2.355	2.250
5	7.709	4.459	3.490	3.007	2.711	2.508	2.359	2.244	2.153
8	5.591	3.739	3.072	2.714	2.485	2.324	2.203	2.109	2.032
10	5.117	3.555	2.960	2.634	2.422	2.272	2.159	2.070	1.998
15	4.600	3.340	2.827	2.537	2.346	2.209	2.104	2.022	1.955
20	4.381	3.245	2.766	2.492	2.310	2.179	2.079	2.000	1.935

$\alpha = 0.1$									
N	k								
	2	3	4	5	6	7	8	9	10
4	5.538	3.463	2.813	2.480	2.273	2.130	2.023	1.940	1.874
5	4.545	3.113	2.606	2.333	2.158	2.035	1.943	1.870	1.811
8	3.589	2.726	2.365	2.157	2.019	1.919	1.843	1.782	1.733
10	3.360	2.624	2.299	2.108	1.980	1.886	1.814	1.757	1.710
15	3.102	2.503	2.219	2.048	1.931	1.845	1.779	1.726	1.682
20	2.990	2.448	2.182	2.020	1.909	1.826	1.762	1.711	1.668

The performance of algorithms is significantly different if the hypothesis "algorithms' performance is the same" is rejected. Then, we use a *post-hoc test* to further distinguish the algorithms. A common choice is the Nemenyi post-hoc test, which calculates the critical difference CD of the average rank difference as

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}. \tag{2.36}$$

q_α is the critical value of Tukey distribution, which can be computed by qtukey$(1-\alpha, k,$ inf) / sqrt(2) in R.

Table 2.7 shows some commonly used values of q_α for $\alpha = 0.05$ and $\alpha = 0.1$. If the average rank difference of two algorithms is greater than the critical difference, then the hypothesis "algorithms' performance is the same" is rejected at the corresponding confidence level.

Tab. 2.7 Commonly used values of q_α for Nemenyi test

α	k								
	2	3	4	5	6	7	8	9	10
0.05	1.960	2.344	2.569	2.728	2.850	2.949	3.031	3.102	3.164
0.10	1.645	2.052	2.291	2.459	2.589	2.693	2.780	2.855	2.920

Taking the data in Table 2.5 as an example, we first calculate $\tau_F = 24.429$ according to (2.34) and (2.35). Then, from Table 2.6, we realize τ_F is greater than the critical value 5.143 when $\alpha = 0.05$. Hence, the hypothesis "algorithms' perfor-

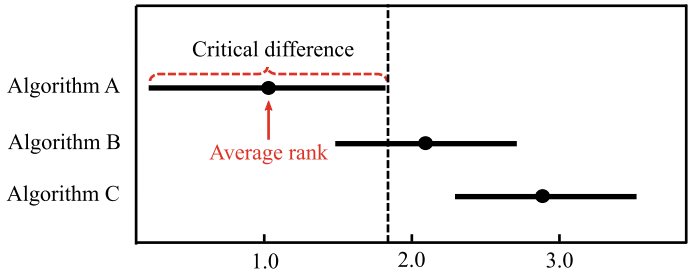


Fig. 2.8 The plot of the Friedman test

mance is the same" is rejected. We proceed with the Nemenyi post-hoc test. From ◘ Table 2.7, we find $q_{0.05} = 2.344$ for $k = 3$, and hence the critical difference is $CD = 1.657$ according to (2.36). Based on the average ranks in ◘ Table 2.5, neither the difference between algorithms A and B nor the difference between algorithms B and C is greater than the critical difference, that is, there is no significant difference between their performance. However, the test confirms that the performance of algorithms A and C are significantly different since their difference is greater than the critical difference.

We can use a plot to illustrate the Friedman test, e.g., ◘ Figure 2.8 illustrates the Friedman test for ◘ Table 2.5, where the y-axis shows the algorithms, and the x-axis shows the average ranks. The dots mark the average ranks of algorithms, and the line segments centered at the dots are the corresponding critical difference. The performance of the two algorithms is not significantly different if their line segments overlap; otherwise, their performance is significantly different. From ◘ Figure 2.8, we can easily observe that there is no significant difference between algorithms A and B since their line segments overlap. On the other hand, algorithm A is better than algorithm C since their line segments do not overlap while A has a higher rank.

2.5 Bias and Variance

In addition to estimating the generalization performance of learning algorithms, people often wish to understand "why" learning algorithms have such performance. An essential tool for understanding the generalization performance of algorithms is the *bias-variance decomposition*, which decomposes the expected generalization error of learning algorithms.

For different training sets, the learning outcomes are often different, although the training samples are drawn from the same distribution. Let $\boldsymbol{x}$ be a testing sample, y_D be the label of $\boldsymbol{x}$ in the data set D, y be the ground-truth label of $\boldsymbol{x}$, and

Potential noise may lead to $y_D \neq y$.

$f(\boldsymbol{x}; D)$ be the output of $\boldsymbol{x}$ predicted by the model f trained on D. Then, in regression problems, the expected prediction of a learning algorithm is

$$\bar{f}(\boldsymbol{x}) = \mathbb{E}_D\left[f(\boldsymbol{x}; D)\right]. \tag{2.37}$$

The variance of using different equal-sized training sets is

$$var(\boldsymbol{x}) = \mathbb{E}_D\left[\left(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x})\right)^2\right]. \tag{2.38}$$

The noise is

$$\varepsilon^2 = \mathbb{E}_D\left[\left(y_D - y\right)^2\right]. \tag{2.39}$$

The difference between the expected output and the ground-truth label is called bias, that is,

$$bias^2(\boldsymbol{x}) = \left(\bar{f}(\boldsymbol{x}) - y\right)^2. \tag{2.40}$$

For ease of discussion, we assume the expectation of noise is zero, i.e., $\mathbb{E}_D[y_D - y] = 0$. By expanding and combining the polynomial, we can decompose the expected generalization error as follows:

$$\begin{aligned}
E(f; D) &= \mathbb{E}_D\left[\left(f(\boldsymbol{x}; D) - y_D\right)^2\right] \\
&= \mathbb{E}_D\left[\left(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x}) + \bar{f}(\boldsymbol{x}) - y_D\right)^2\right] \\
&= \mathbb{E}_D\left[\left(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x})\right)^2\right] + \mathbb{E}_D\left[\left(\bar{f}(\boldsymbol{x}) - y_D\right)^2\right] \\
&\quad + \mathbb{E}_D\left[2\left(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x})\right)\left(\bar{f}(\boldsymbol{x}) - y_D\right)\right] \\
&= \mathbb{E}_D\left[\left(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x})\right)^2\right] + \mathbb{E}_D\left[\left(\bar{f}(\boldsymbol{x}) - y_D\right)^2\right] \\
&= \mathbb{E}_D\left[\left(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x})\right)^2\right] + \mathbb{E}_D\left[\left(\bar{f}(\boldsymbol{x}) - y + y - y_D\right)^2\right] \\
&= \mathbb{E}_D\left[\left(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x})\right)^2\right] + \mathbb{E}_D\left[\left(\bar{f}(\boldsymbol{x}) - y\right)^2\right] \\
&\quad + \mathbb{E}_D\left[\left(y - y_D\right)^2\right] + 2\mathbb{E}_D\left[\left(\bar{f}(\boldsymbol{x}) - y\right)\left(y - y_D\right)\right] \\
&= \mathbb{E}_D\left[\left(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x})\right)^2\right] + \left(\bar{f}(\boldsymbol{x}) - y\right)^2 + \mathbb{E}_D\left[\left(y_D - y\right)^2\right].
\end{aligned} \tag{2.41}$$

Since the noise does not rely on f, the last term equals to 0 according to (2.37).

The last term equals to 0 since the expectation of noise is 0.

That is,

$$E(f; D) = bias^2(\boldsymbol{x}) + var(\boldsymbol{x}) + \varepsilon^2, \tag{2.42}$$

which means the generalization error can be decomposed into the sum of bias, variance, and noise.

Bias (2.40) measures the difference between the learning algorithm's expected prediction and the ground-truth label, that is, expressing the fitting ability of the learning algorithm. Variance (2.38) measures the change of learning performance caused by changes to the equal-sized training set, that is, expressing the impact of data disturbance on the learning outcome. Noise (2.39) represents the lower bound of the expected generalization error that can be achieved by any learning algorithms for the given task, that is, the inherent difficulty of the learning problem. The bias-variance decomposition tells us that the generalization performance is jointly determined by the learning algorithm's ability, data sufficiency, and the inherent difficulty of the learning problem. In order to achieve excellent generalization performance, a small bias is needed by adequately fitting the data, and the variance should also be kept small by minimizing the impact of data disturbance.

Generally speaking, bias and variance are conflicted with each other, and this is known as the bias-variance dilemma. ◘ Figure 2.9 gives an illustrating example. Given a learning problem and a learner, suppose we can control the degree of training. If we limit the degree of training such that the learner is undertrained, its fitting ability is limited, and hence the data disturbances have a limited impact on the learner, that is, bias dominates the generalization error. As the training proceeds, the learner's fitting ability improves, and hence the learner starts to learn the data disturbances, that is, variance starts to dominate the generalization error. After a large amount of training, the fitting ability of the learner becomes very strong, and hence slight disturbances in the training data will cause significant changes to the learner. At this point, the learner may start to learn the peculiarities of the training data, and hence overfitting occurs.

Many learning algorithms allow users to control the degree of training, such as the number of levels in decision trees, the number of training epochs in neural networks, and the number of base learners in ensemble learning methods.

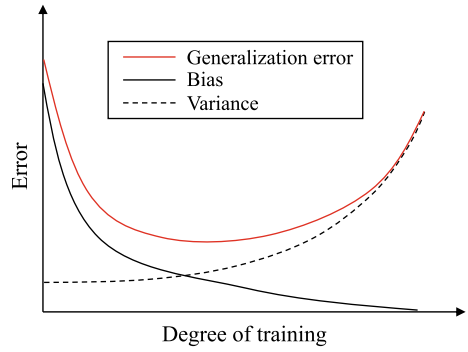


Fig. 2.9 Relationships between generalization error, bias, and variance

2.6 Further Reading

Bootstrap sampling has crucial applications in machine learning, and a detailed discussion can be found in Efron and Tibshirani (1993).

ROC curve was introduced to machine learning in the late 1980s (Spackman 1989), and AUC started to be widely used in the field of machine learning since the middle 1990s (Bradley 1997). However, using the area under the ROC curve to evaluate the expected performance of models has already been done much earlier in medical diagnosis (Hanley and McNeil 1983). Hand and Till (2001) extended the ROC curve from binary classification problems to multiclass classification problems. Fawcett (2006) surveyed the use of the ROC curve.

Drummond and Holte (2006) invented the cost curve. Other than the misclassification cost, there are many costs involved in the machine learning process, such as the testing cost, labeling cost, and feature cost. The misclassification cost can also be further divided into the class-based misclassification cost and sample-based misclassification cost. Cost-sensitive learning (Elkan 2001; Zhou and Liu 2006) is a research topic for learning under unequal cost settings.

Section 2.3.4 only discussed the class-based misclassification cost.

Dietterich (1998) pointed out the risk of using the regular k-fold cross-validation method, and proposed the 5×2 cross-validation method. Demsar (2006) discussed the hypothesis testing methods for comparing multiple algorithms.

Geman et al. (1992) proposed the bias-variance-covariance decomposition for regression problems, which was later shortened as bias-variance decomposition. Though bias and variance reveal the internal factors of errors, we can only derive the elegant form of (2.42) for regression problems based on MSE. For classification problems, however, deriving the bias-variance decomposition is difficult since the 0/1 loss function is discontinuous. There exist many empirical methods for estimating bias and variance (Kong and Dietterich 1995; Kohavi and Wolpert 1996; Breiman 1996; Friedman 1997; Domingos 2000).

Exercises

2.1 Given a data set of 1000 samples, where 500 samples are positive and 500 samples are negative. To perform a hold-out evaluation, we split the data set into a training set with 70% of the samples and a testing set with 30% of the samples. Estimate the total number of possible splittings.

2.2 Given a data set of 100 samples, where the positive and negative samples are half-half. Suppose that the model produced by a learning algorithm predicts every new sample as the majority class in the training set (random guessing if different classes have the same number of samples). Calculate the error rates of this model evaluated by 10-fold cross-validation and hold-out, respectively.

2.3 Given that the $F1$ value of learner A is greater than that of learner B, find out whether the BEP value of A is also greater than that of B.

2.4 Describe the relationships among TPR, FPR, Precision, and Recall.

2.5 Prove (2.22).

2.6 Describe the relationship between error rate and ROC curve.

2.7 Prove that every ROC curve has a corresponding cost curve, and vice versa.

2.8 The min-max normalization and z-score normalization are two commonly used normalization methods. Let x and x' denote the variable value before and after normalization, $x_{\min}$ and $x_{\max}$ denote the minimum and maximum value before normalization, $x'_{\min}$ and $x'_{\max}$ denote the minimum and maximum value after normalization, $\overline{x}$ denote the unnormalized mean, and σ_x denote the standard deviation. Then, the min-max normalization and z-score normalization are, respectively, defined in (2.43) and (2.44). Discuss the pros and cons of each method.

$$x' = x'_{\min} + \frac{x - x_{\min}}{x_{\max} - x_{\min}} \times (x'_{\max} - x'_{\min}), \tag{2.43}$$

$$x' = \frac{x - \bar{x}}{\sigma_x}. \tag{2.44}$$

2.9 Describe the process of χ^2 test.

2.10 * Describe the difference between using (2.34) and using (2.35) in the Friedman test.

Break Time

Short Story: *t*-Test, Beer, "Student", and William Gosset

In 1954, the Guinness corporation started to publish *Guinness World Records*.

In 1899, William Gosset (1876–1937), who majored in chemistry at the University of Oxford, joined Guinness Brewery in Dublin, Ireland after graduation and wished to apply his biology and chemistry knowledge to the brewing process. Gosset proposed *t*-test to reduce the cost of quality control of brewing, and published this work in *Biometrika* in 1908. In order to prevent the leak of the trade secret, the paper was published under the pseudonym of "Student", and this leads to the method's name "Student's *t*-test".

As a visionary corporation, Guinness Brewery grants its technical staff "sabbatical leave" just like in universities such that its staff can maintain a high level of technical skills. For this reason, Gosset had a chance to visit the lab led by Professor Karl Pearson (1857–1936) at University College London (UCL) in 1906–1907. Since *t*-test was published shortly after the visit, it is hard to tell whether it was developed at Guinness Brewery or during the visit at UCL. Nevertheless, the connection between "Student" and Gosset was found by statisticians from UCL, and this is not a surprise since Professor Pearson happened to be the editor-in-chief of *Biometrika*.

References

Bradley AP (1997) The use of the area under the roc curve in the evaluation of machine learning algorithms. Pattern Recognit 30(7):1145–1159
Breiman L (1996) Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California, CA
Demsar J (2006) Statistical comparison of classifiers over multiple data sets. J Mach Learn Res 7:1–30
Dietterich TG (1998) Approximate statistical tests for comparing supervised classification learning algorithms. Neural Comput 10(7):1895–1923
Domingos P (2000) A unified bias-variance decomposition. In: Proceedings of the 17th international conference on machine learning (ICML), pp 231–238. Stanford, CA
Drummond C, Holte RC (2006) Cost curves: an improved method for visualizing classifier performance. Mach Learn 65(1):95–130
Efron B, Tibshirani R (1993) An introduction to the bootstrap. Chapman & Hall, New York
Elkan C (2001) The foundations of cost-sensitive learning. In: Proceedings of the 17th international joint conference on artificial intelligence (IJCAI), pp 973–978. Seattle, WA
Fawcett T (2006) An introduction to roc analysis. Pattern Recognit Lett 27(8):861–874
Friedman JH (1997) On bias, variance, 0/1-loss, and the curse-of-dimensionality. Data Mining Knowl Disc 1(1):55–77
Geman S, Bienenstock E, Doursat R (1992) Neural networks and the bias/variance dilemma. Neural Comput 4(1):1–58
Hand DJ, Till RJ (2001) A simple generalisation of the area under the ROC curve for multiple class classification problems. Mach Learn 45(2):171–186
Hanley JA, McNeil BJ (1983) A method of comparing the areas under receiver operating characteristic curves derived from the same cases. Radiology 148(3):839–843
Kohavi R, Wolpert DH (1996) Bias plus variance decomposition for zero-one loss functions. In: Proceedings of the 13th international conference on machine learning (ICML), pp 275–283. Bari, Italy
Kong EB, Dietterich TG (1995) Error-correcting output coding corrects bias and variance. In: Proceedings of the 12th international conference on machine learning (ICML), pp 313–321. Tahoe City, CA
Mitchell T (1997) Machine learning. McGraw Hill, New York
Spackman KA (1989) Signal detection theory: valuable tools for evaluating inductive learning. In: Proceedings of the 6th international workshop on machine learning (IWML), pp 160–163. Ithaca, NY
Van Rijsbergen CJ (1979) Information retrieval, 2nd edn. Butterworths, London
Wellek S (2010) Testing statistical hypotheses of equivalence and noninferiority, 2nd edn. Chapman & Hall/CRC, Boca Raton
Zhou Z-H, Liu X-Y (2006) On multi-class cost-sensitive learning. In: Proceedings of the 21st national conference on artificial intelligence (AAAI), pp 567–572. Boston, WA