Evaluation

Jianxin Wu

LAMDA Group
National Key Lab for Novel Software Technology
Nanjing University, China
wujx2001@gmail.com

April 30, 2020

Contents

It is very important to evaluate the performance of a learning algorithm or a recognition system. A systematic evaluation is often a comprehensive process, which may involve many aspects such as the speed, accuracy (or other accuracy-related evaluation metrics), scalability, reliability, etc. In this chapter, we focus on the evaluation of the accuracy and other accuracy-related evaluation metrics. We will also briefly discuss why perfect accuracy is impossible for many tasks, where the errors come from, and how confident we should be about the evaluation results.

1 Accuracy and error in the simple case

Accuracy is a widely used evaluation metric in classification tasks, which we have briefly mentioned in the previous chapter. In layman's language, accuracy is the percentage of examples that are correctly classified. The error rate is the percentage of examples that are wrongly (incorrectly) classified—i.e., one minus accuracy. We use Acc and Err to denote accuracy and error rate, respectively. Note that both accuracy and error rate are between 0 and 1 (e.g., $0.49 = 49\%$). However, as mentioned in the previous chapter, we have to resort to quite some assumptions to make this definition crystal clear.

Suppose we are dealing with a classification task, which means that the labels of all examples are taken from a set of categorical values $\mathbb{Y}$. Similarly, the features of all examples are from the set $\mathbb{X}$. We want to evaluate the accuracy of a learned mapping

$$f : \mathbb{X} \mapsto \mathbb{Y}\,.$$

We obey the i.i.d. assumption—that is, any example $(\boldsymbol{x}, y)$ satisfies the conditions that $\boldsymbol{x} \in \mathbb{X}$, $y \in \mathbb{Y}$, and an example is drawn independently from the same underlying joint distribution $p(\boldsymbol{x}, y)$, written as

$$(\boldsymbol{x}, y) \sim p(\boldsymbol{x}, y)\,.$$

The mapping is learned on a training set of examples D_{train}, and we also have a set of test examples D_{test} for evaluation purposes. Under these assumptions, we can define the following error rates.

- The *generalization* error. This is the expected error rate when we consider all possible examples that follow the underlying distribution, i.e.,

$$\text{Err} = \mathbb{E}_{(\boldsymbol{x},y)\sim p(\boldsymbol{x},y)}\left[[\![f(\boldsymbol{x}) \neq y]\!] \right], \tag{1}$$

  in which $[\![\cdot]\!]$ is the indicator function. If we know the underlying distribution *and* the examples and labels can be enumerated (e.g., the joint distribution is discrete) or analytically integrated (e.g., the joint distribution is a simple continuous one), the generalization error can be explicitly computed.

However, in almost all tasks, the underlying distribution is unknown. Hence, we often assume the generalization error exists as a fixed number between 0 and 1 but cannot be explicitly calculated. Our purpose is to provide a robust estimate of the generalization error.

- Approximate error. Because of the i.i.d. assumption, we can use a set of examples (denoted as D) to *approximate* the generalization error, *so long as D is sampled from the underlying distribution following the i.i.d. assumption*,

$$\text{Err} \approx \frac{1}{|D|} \sum_{(\boldsymbol{x},y)\in D} [\![f(\boldsymbol{x}) \neq y]\!] \,, \tag{2}$$

  in which $|D|$ is the number of elements in the set D. It is easy to verify that Equation 2 calculates the percentage of examples in D that are incorrectly classified ($f(\boldsymbol{x}) \neq y$).

- The *training* and *test* error. The training set D_{train} and the test set D_{test} are two obvious candidates for the D set in Equation 2. When $D = D_{\text{train}}$, the calculated error rate is called the training error; and when $D = D_{\text{test}}$, it is called the test error.

1.1 Training and test error

Why do we need a test set? The answer is simple: the training error is *not* a reliable estimate of the generalization error.

Consider the nearest neighbor classifier. It is easy to deduce that the training error will be 0 because the nearest neighbor of one training example must be itself.[1] However, we have good reasons to believe that NN search will lead to errors in most applications (i.e., the generalization error is larger than 0). Hence, the training error is too optimistic for estimating the generalization error of the 1-NN classifier.

The NN example is only an extreme case. However, in most tasks we will find that the training error is an *overly optimistic* approximation of the generalization error. That is, it is usually smaller than the true error rate. Because the mapping f is learned to fit the characteristics of the training set, it makes sense that f will perform better on D_{train} than on examples not in the training set.

The solution is to use a separate test set D_{test}. Because D_{test} is sampled i.i.d., we expect the examples in D_{train} and D_{test} to be different from each other with high probability (especially when the space $\mathbb{X}$ is a large one). That is, it is unlikely that examples in D_{test} have been utilized in learning f. Hence, the test error (which is calculated based on D_{test}) is a better estimate of the generalization error than the training error. In practice, we divide all the available

[1]If two examples $(\boldsymbol{x}_1, y_1)$ and $(\boldsymbol{x}_2, y_2)$ are in the training set, with $\boldsymbol{x}_1 = \boldsymbol{x}_2$ but $y_1 \neq y_2$ due to label noise or other reasons, the training error of the nearest neighbor classifier will not be 0. A more detailed analysis of this kind of irreducible error will be presented later in this chapter. For now we assume if $\boldsymbol{x}_1 = \boldsymbol{x}_2$ then $y_1 = y_2$.

examples into two *disjoint* sets: the training set and the test set, i.e.,

$$D_{\text{train}} \cap D_{\text{test}} = \emptyset\,.$$

1.2 Overfitting and underfitting

Over- and under-fitting are two adverse phenomena when learning a mapping f. We use a simple regression task to illustrate these two concepts and a commonly used evaluation metric for regression.

The regression mapping we study is $f : \mathbb{R} \mapsto \mathbb{R}$. In this simple task, the mapping can be written as $f(x)$, where the input x and output $f(x)$ are both real numbers. We assume the mapping f is a polynomial with degree d—i.e., the model is

$$y = f(x) + \epsilon = p_d x^d + p_{d-1} x^{d-1} + \cdots + p_1 x + p_0 + \epsilon\,. \tag{3}$$

This model has $d+1$ *parameters* $p_i \in \mathbb{R}$ $(0 \le i \le d)$, and the random variable ϵ models the noise of the mapping—i.e.,

$$\epsilon = y - f(x)\,.$$

The noise ϵ is *independent* of x. We then use a training set to learn these parameters.

Note that the polynomial degree d is not considered as a parameter of $f(x)$, which is specified prior to the learning process. We call d a *hyperparameter*. After both the hyperparameter d and all the parameters p_i are specified, the mapping f is completely learned.

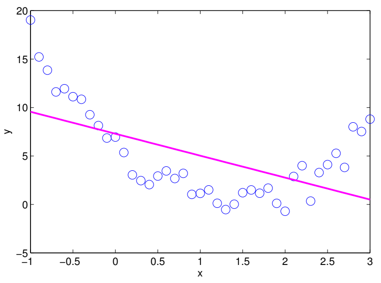
It is not difficult to find the optimal parameters for polynomial regression, which we leave as an exercise. In Figure 1, we show the fitted models for the same set of training data for different polynomial degrees. The data are generated by a degree 2 polynomial,

$$y = 3(x-0.2)^2 - 7(x-0.2) + 4.2 + \epsilon\,, \tag{4}$$

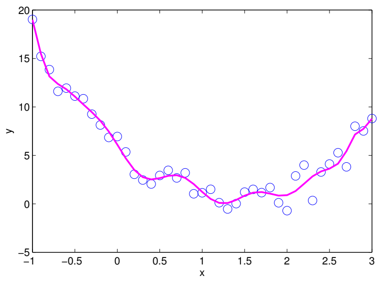
in which ϵ is i.i.d. sampled from a standard normal distribution—i.e.,

$$\epsilon \sim N(0,1)\,. \tag{5}$$

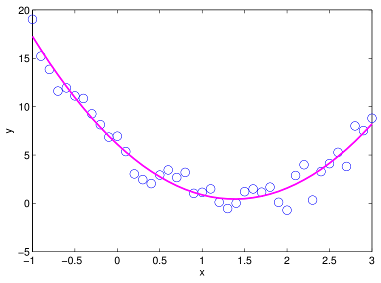
In Figure 1, the x- and y-axes represent x and y / $f(x)$ values, respectively. One blue marker corresponds to one training example, and the purple curve is the fitted polynomial regression model $f(x)$. In Figure 1a, $d=1$, and the fitted model is a linear one. It is easy to observe from the blue circles that the true relationship between x and y is nonlinear. Hence, the capacity or representation power of the model (degree 1 polynomial) is lower than the complexity in the data. This phenomenon is called *underfitting*, and we expect large errors when underfitting happens. As shown by the large discrepancy between the purple curve and the blue points, large regression errors indeed occur.



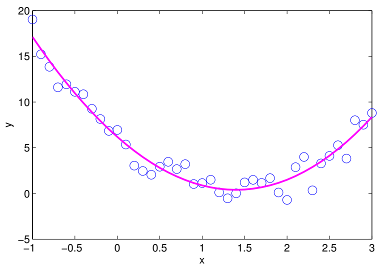
(a) Degree 1 polynomial



(b) Degree 15 polynomial



(c) Degree 2 polynomial



(d) Degree 3 polynomial

Figure 1: Fitting data with polynomials with different degrees of freedom. ()

Table 1: True (first row) and learned (second and third rows) parameters of polynomial regression.

	p_3	p_2	p_1	p_0
True		3.0000	-8.2000	6.8000
$d = 2$		2.9659	-9.1970	6.1079
$d = 3$	0.0489	2.8191	-8.1734	6.1821

Now we move on to Figure 1b (with $d = 15$). The learned polynomial curve is a complex one, with many upward and downward transitions along the curve. We observe that most blue points are very close to the purple curve, meaning that the fitting error is very small *on the training set*. However, since the data are generated using a degree 2 polynomial, its generalization (or test) error will be large, too. A higher order polynomial has larger representation power than a lower order one. The extra capacity is often used to model the noise in the examples (the normal noise ϵ in this case) or some peculiar property in the training set (which is not a property of the underlying distribution). This phenomenon is called *overfitting*.

Nowadays we are equipped with more computing resources, which means we can afford to use models with large capacity. Overfitting has become a more serious issue than underfitting in many modern algorithms and systems.

Figure 1c uses the correct polynomial degree ($d = 2$). It strikes a good balance between minimizing the training error and avoiding overfitting. As shown by this figure, most blue points are around the purple curve, and the curve is quite smooth. We expect it to also achieve small generalization error.

In Figure 1d, we set $d = 3$, which is slightly larger than the correct value ($d = 2$). There are indeed differences between the curves in Figure 1c and Figure 1d, although the differences are almost indistinguishable. We also listed the true and learned parameter values in Table 1. Both learned sets of parameters match the true values quite well. The p_3 parameter for the degree 3 polynomial is close to 0, which indicates that the term p_3x^3 has limited impact for predicting y. In other words, when the model capacity is only slightly higher than the data's true complexity, a learning algorithm has the potential to still properly fit the data. However, when the gap between model and data complexity is large (cf. Figure 1b), overfitting will cause serious problems.

1.3 Choosing the hyperparameter(s) using validation

Unlike the parameters, which can be learned based on the training data, hyperparameters are usually not learnable. Hyperparameters are often related to the capacity or representation power of the mapping f. A few examples are listed here.

- d in polynomial regression. It is obvious that a larger d is directly related to higher representation power.

- k in k-NN. A large k value can reduce the effect of noise, but may reduce the discriminative power in classification. For example, if k equals the number of training examples, the k-NN classifier will return the same prediction for any example.

- Bandwidth h in kernel density estimation (KDE). We will introduce KDE in the chapter on probabilistic methods (Chapter 8). Unlike in the above examples, there is a theoretical rule that guides the choice of optimal bandwidth.

- γ in the RBF kernel for SVM. We will introduce the RBF kernel in the SVM chapter (Chapter 7). The choice of this hyperparameter, however, is neither learnable from data nor guided by a theoretical result.

Hyperparameters are often critical in a learning method. Wrongly changing the value of one hyperparameter (e.g., γ in the RBF kernel) may dramatically reduce the classification accuracy (e.g., from above 90% to less than 60%). We need to be careful about hyperparameters.

One widely adopted approach is to use a validation set. The validation set is provided in some tasks, in which we want to ensure that the training, validation, and test sets are *disjoint*. In scenarios where this split is not provided, we can randomly split the set of all available data into three disjoint sets: D_{train}, D_{val} (the validation set), and D_{test}.

Then, we can list a set of representative hyperparameter values—e.g., $k \in \{1, 3, 5, 7, 9\}$ for k-NN—and train a classifier using the training set and one of the hyperparameter values. The resulting classifier can be evaluated using the *validation* set to obtain a validation error. For example, we will obtain 5 validation k-NN error rates, one for each hyperparameter value. The hyperparameter value that leads to the smallest validation error rate will be used as the chosen hyperparameter value.

In the last step, we train a classifier using D_{train} and the chosen hyperparameter value to train a classifier, and its performance can be evaluated using D_{test}. When there are ample examples, this strategy often works well. For example, we can use $\frac{n}{10}$ examples for testing when there are n examples in total *and* when n is large. In the remaining $\frac{9n}{10}$ examples, we can use $\frac{n}{10}$ for validation and the remaining $\frac{8n}{10}$ for training. The ratio of sizes between train/validation or train/test is not fixed, but we usually set aside a large portion of examples for training. If a split between training and test examples has been pre-specified, we can split the training set into a smaller training set and a validation set when necessary.

The validation set can be used for other purposes, too. For example, the parameters of neural networks (including the convolutional neural network) are learned iteratively. The neural network model (i.e., the values of its parameters) is updated in every iteration. However, it is not trivial to determine when to stop the iterative updating procedure. Neural networks have very large capacity. Hence, the training error will continue to decrease even after many training iterations. But, after some number of iterations, the new updates may be mainly

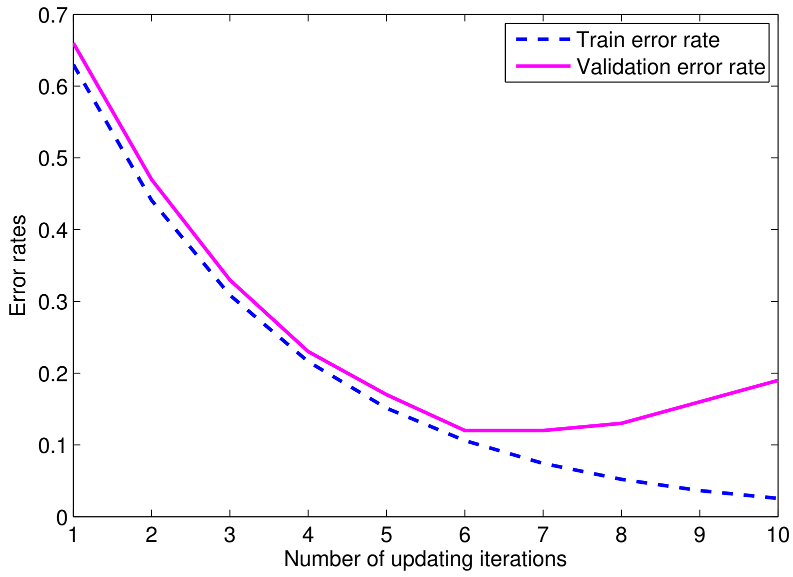


Figure 2: Illustration of the overfitting phenomenon. One unit in the x-axis can be one or more (e.g., 1000) updating iterations.

fitting the noise, which leads to overfitting, and the validation error will start to increase. Note that the validation and training sets are disjoint. Hence, both the validation error and the test error rates will increase if overfitting happens.

One useful strategy is to evaluate the model's validation error. The learning process can be terminated if the validation error becomes stable (not decreasing in a few consecutive iterations) or even starts to increase—that is, when it starts to continuously overfit. This strategy is called 'early stopping' in the neural network community.

Figure 2 illustrates the interaction between training and validation error rates. Note that this is only a schematic illustration. In practice, although the overall trend is that the training error reduces when more iterations are updated, it can have small ups and downs in a short period. The validation error can also be (slightly) smaller than the training error before overfitting occurs. In this example, it is probably a good idea to stop the training process after six iterations.

1.4 Cross validation

The validation strategy works well when there are a large number of examples. For example, if one million ($n = 10^6$) examples are available, the validation set has 100 000 examples ($\frac{n}{10}$), which is enough to provide a good estimate of the generalization error. However, when n is small (e.g., $n = 100$), this strategy may cease to work if we use a small portion of examples for validation (e.g., $\frac{n}{10} = 10$). If we set aside a large portion as the validation set (e.g., $\frac{n}{2} = 50$), the training set will be too small to learn a good classifier.

Cross-validation (CV) is an alternative strategy for small size tasks. A k-fold CV *randomly* divides the training set D_{train} into k (roughly) *even* and *disjoint* subsets $D_1, D_2, \ldots, D_k$, such that

$$\bigcup_{i=1}^{k} D_i = D_{\text{train}}\,, \tag{6}$$

$$D_i \cap D_j = \emptyset, \qquad \forall\, 1 \le i \ne j \le k\,, \tag{7}$$

$$|D_i| \approx \frac{|D_{\text{train}}|}{k}, \qquad \forall\, 1 \le i \le k\,. \tag{8}$$

The CV estimate of the error rate then operates following Algorithm 1.

Algorithm 1 Cross-validation estimate of the error rate

```
1: Input: $k$, $D_{\text{train}}$, a learning algorithm (with hyperparameters specified).
2: Randomly divide the training set into $k$ folds satisfying Equations 6–8.
3: for $i = 1, 2, \ldots, k$ do
4:     Construct a new training set, which consists of all the examples from
       $D_1, \ldots, D_{i-1}, D_{i+1}, \ldots, D_k$.
5:     Learn a classifier $f_i$ using this training set and the learning algorithm.
6:     Calculate the error rate of $f_i$ on $D_i$, denoted as $\text{Err}_i$.
7: end for
8: Output: Return the error rate estimated by cross-validation as
```

$$\text{Err}_{\text{CV}} = \frac{1}{k}\sum_{i=1}^{k} \text{Err}_i\,. \tag{9}$$

In Algorithm 1, the learning algorithm is run k times. In the i-th run, all examples except those in the i-th subset D_i are used for training, while D_i is used to compute the validation error Err_i. After all the k runs, the cross-validation estimate is the average of k validation errors Err_i ($1 \le i \le k$).

Note that every example is used *exactly once* to compute one of the validation error rates, and at the same time is used $k-1$ times for training one of the k classifiers. In other words, the CV strategy makes good use of the small number of examples and hence may provide a reasonable estimate of the error rate.

Because the number of training examples is small, often a small k is used—e.g., $k = 10$ or $k = 5$. Although k classifiers have to be learned, the computational costs are still acceptable because $|D_{\text{train}}|$ is small.

Furthermore, because there is randomness in the division of D_{train} into k folds, different Err_{CV} will be returned if we run Algorithm 1 multiple times. The difference can be large if $|D_{\text{train}}|$ is really small (e.g., a few tens). In this case, we can run k-fold CV multiple times and return the average of their estimates. A commonly used CV schedule is 10 times average of 10-fold cross-validation.

When the number of available examples is small, we may use all these examples as D_{train} (i.e., do not put aside examples as a separate test set). The cross validation error rate can be used to evaluate classifiers in this case.

2 Minimizing the cost/loss

Based on the above analyses of error rates, it is a natural idea to formulate a learning task as an optimization problem which minimizes the error rate. We will use the classification task to illustrate this idea.

Given a training set consists of n examples $(\boldsymbol{x}_i, y_i)$ $(1 \leq i \leq n)$, the training error rate of a mapping f is simply

$$\min_f \frac{1}{n} \sum_{i=1}^{n} [\![f(\boldsymbol{x}_i) \neq y_i]\!] \,. \tag{10}$$

In Equation 10, the exact form of the mapping f and its parameters are not specified. After these components are fully specified, optimizing Equation 10 will lead to an optimal (in the training error sense) mapping f.

However, the indicator function $[\![\cdot]\!]$ is not smooth—even not convex for most functional forms and parameterizations of f. Hence, as we have discussed in Chapter 2, solving Equation 10 will be very difficult. We may also have to require $f(\boldsymbol{x}_i)$ to be categorical (or integer) values, which makes designing the form of f more difficult.

When $[\![f(\boldsymbol{x}_i) \neq y_i]\!]$ is 1, the i-th training example is wrongly classified by f, which means some *cost* or *loss* has been attributed to the mapping f with respect to the example $(\boldsymbol{x}_i, y_i)$, and Equation 10 is minimizing the average loss on the training set. Hence, we also call this type of learning cost minimization or loss minimization.

A widely used strategy is to replace the difficult-to-optimize loss function $[\![\cdot]\!]$ with a new optimization-friendly loss function. For example, if the classification problem is binary—i.e., $y_i \in \{0, 1\}$ for all i—we can use the following *mean squared error* (MSE) loss

$$\min_f \frac{1}{n} \sum_{i=1}^{n} \left(f(\boldsymbol{x}_i) - y_i \right)^2 \,. \tag{11}$$

A few properties of the MSE loss makes it a good loss function:

- We can treat y_i as real values in MSE, and $f(\boldsymbol{x}_i)$ does *not* need to be categorical. During prediction, the output can be 1 if $f(\boldsymbol{x}_i) \geq 0.5$ and 0 otherwise.

- In order to minimize the loss, MSE encourages $f(\boldsymbol{x}_i)$ to be close to 0 if $y_i = 0$, and pushes $f(\boldsymbol{x}_i)$ to be around 1 if $y_i = 1$. In other words, the behavior of MSE mimics some *key characteristics* of the indicator function (the true loss function). We expect a classifier learned based on MSE to have a low training error rate too (even though it might not be the smallest possible).

- The square function is smooth, differentiable, and convex, which makes the minimization of MSE an easy task. Hence, in this formalization of loss minimization, we can treat the MSE as a simplification of the indicator function.

MSE is also a natural loss function for regression tasks (e.g., polynomial regression). In this book, we will encounter many other loss functions.

The above strategy replaces the mean indicator loss function with a mean squared error, which is called a *surrogate* loss function. If a surrogate loss is easier to minimize and its key characteristics mimic the original one, we can replace the original loss function with the surrogate to make the optimization tractable.

2.1 Regularization

As illustrated by Figure 1, both underfitting and overfitting are harmful. As we will show later in this book, a classifier with small capacity will have large *bias*, which constitutes part of the error rate. In contrast, a classifier with large capacity or representation power will have small bias but large *variance*, which is another major contributor to the error rate.

Since we do not know the inherent complexity in our task's examples, it is never easy to choose a classifier that has suitable capacity for a specific task. One commonly used strategy to deal with this difficulty is to use a mapping f with high capacity, but at the same time add a *regularization* term to it.

An ideal situation is that the bias will be small (because f is complex enough) and the variance may also be small (because the regularization term penalizes complex f). Let $\mathcal{R}(f)$ be a regularizer on f; the MSE minimization (Equation 11) now becomes

$$\min_f \frac{1}{n} \sum_{i=1}^{n} \left(f(\boldsymbol{x}_i) - y_i\right)^2 + \lambda \mathcal{R}(f)\,. \tag{12}$$

The hyperparameter $\lambda > 0$ is a tradeoff parameter that controls the balance between a small training set cost $\frac{1}{n}\sum_{i=1}^{n}\left(f(\boldsymbol{x}_i) - y_i\right)^2$ and a small regularization cost $\mathcal{R}(f)$.

The regularization term (also called the regularizer) has different forms for different classifiers. Some regularizers will be introduced later in this book.

2.2 Cost matrix

The simple accuracy or error rate criterion may work poorly in many situations. Indeed, as we have already discussed, minimizing the error rate in a severely imbalanced binary classification task may cause the classifier to classify all examples into the majority class.

Most imbalanced learning tasks are also *cost-sensitive*: making different types of errors will incur losses that are also imbalanced. Let us consider the small startup IT company with 24 employees once again. Company X has signed a contract with your company, and you (with your employee ID being 24) are assigned to this project. Company X produces part Y for a complex product Z, and decides to replace the quality inspector with an automatic system. The percentage of qualified parts is around 99% in company X, hence you have to deal with an imbalanced classification task. If a qualified part is predicted as defective, this part will be discarded, which will cost 10 RMB. In contrast, if a part is in fact defective but is predicted as qualified, it will be assembled into the final product Z and disable the entire product Z. Company X has to pay 1000 RMB for one such defective part. Hence, your problem is also obviously cost-sensitive.

The cost minimization framework can be naturally extended to handle cost-sensitive tasks. Let $(\boldsymbol{x}, y)$ be an example and $f(\boldsymbol{x})$ be a mapping that predicts the label for $\boldsymbol{x}$. y is the groundtruth label for $\boldsymbol{x}$, which is denoted as 0 if the part $\boldsymbol{x}$ is qualified and 1 if it is defective. For simplicity, we assume $f(\boldsymbol{x})$ is also binary (0 if it predicts $\boldsymbol{x}$ as qualified, and 1 for defective). We can define a 2×2 *cost matrix* to summarize the cost incurred in different cases:

$$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} 0 & 10 \\ 1000 & 10 \end{bmatrix} \tag{13}$$

In Equation 13, c_{ij} is the cost incurred when the groundtruth label y is i and the prediction $f(\boldsymbol{x})$ is j.

Note that even a correct prediction can incur certain cost. For example, $c_{11} = 10$ because a defective part will cost 10 RMB even if it is correctly detected as a defective one.

With an appropriate cost matrix, Equation 10 becomes

$$\min_f \frac{1}{n} \sum_{i=1}^{n} c_{y_i, f(\boldsymbol{x}_i)} , \tag{14}$$

which properly handles different costs.

Cost-sensitive learning is an important topic in learning and recognition. Although we will not cover its details in this book, we want to present a few notes on the cost matrix.

- Minimizing the error rate can be considered as a special case of cost minimization, in which the cost matrix is $\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$.
- It is, however, difficult to determine appropriate c_{ij} values in many real-world applications.
- The cost matrix can be easily extended to multi-class problems. In an m class classification problem, the cost matrix is of size $m \times m$, and c_{ij} is the cost when the groundtruth label is i but the prediction is j.

Table 2: Possible combinations for the groundtruth and the predicted label. ()

	Prediction $f(\boldsymbol{x}) = +1$	Prediction $f(\boldsymbol{x}) = -1$
True label $y = +1$	True positive	False negative
True label $y = -1$	False positive	True negative

2.3 Bayes decision theory

Bayes decision theory minimizes the cost function in the probabilistic sense. Assume the data $(\boldsymbol{x}, y)$ is a random vector, and the joint probability is $\Pr(\boldsymbol{x}, y)$; Bayes decision theory seeks to minimize the *risk*, which is the expected loss with respect to the joint density:

$$\sum_{\boldsymbol{x},y} c_{y,f(\boldsymbol{x})} \Pr(\boldsymbol{x}, y)\,. \tag{15}$$

Here we are using the notation for discrete random variables. If continuous or hybrid random distributions are involved, integrations (or a combination of summations and integrations) can be used to replace the summations.

The risk is the average loss over the true underlying probability, which can also be succinctly written as

$$\mathbb{E}_{(\boldsymbol{x},y)}[c_{y,f(\boldsymbol{x})}]\,. \tag{16}$$

By picking the mapping $f(\boldsymbol{x})$ that minimizes the risk, Bayes decision theory is optimal under these assumptions in the minimum cost sense.

3 Evaluation in imbalanced problems

Error rate is not a good evaluation metric in imbalanced or cost-sensitive tasks. It is also difficult to determine the cost (c_{ij}) values in Equation 13. In practice, we use another set of evaluation metrics (such as *precision* and *recall*) for these tasks.

In this section, we will only consider binary classification tasks. Following the commonly used terminology, we call the two classes *positive* and *negative*, respectively. The positive class often happens to be the minority class, that is, the class with fewer examples (e.g., defective parts). The negative class often happens to be the majority class, that is, the class with more examples (e.g., qualified parts). We use $+1$ and -1 to denote positive and negative classes, respectively.

3.1 Rates inside individual class

There are four possible combinations of the groundtruth label y and the prediction $f(\boldsymbol{x})$ for one example $\boldsymbol{x}$, which are summarized in Table 2.

We use two words to pinpoint one of these four possible cases, such as true positive or false negative. In each possible case, the second word refers to the predicted label (which is shown in red in Table 2). The first word describes whether the prediction is correct or not. For example, *false positive* indicates the predicted label is "positive" (+1) and this prediction is wrong ("false"); hence, the true label is "negative" (−1). We use the initials as abbreviations for these four cases: TP, FN, FP, TN.

Given a test set with $TOTAL$ number of examples, we also use these abbreviations to denote the number of examples falling into each of the four cases. For example, TP=37 means that 37 examples in the test set are true positives. The following quantities are defined based on TP, FN, FP, and TN.

- The total number of examples:
$$TOTAL = TP + FN + FP + TN$$
- The total number of positive examples (whose true label is +1):
$$P = TP + FN$$
- The total number of negative examples (whose true label is −1):
$$N = FP + TN$$
- Accuracy:
$$\text{Acc} = \frac{TP + TN}{TOTAL}$$
- Error rate:
$$\text{Err} = \frac{FP + FN}{TOTAL} = 1 - \text{Acc}$$
- True positive rate:
$$TPR = \frac{TP}{P},$$
which is the ratio between the number of true positives and the total number of positive examples
- False positive rate:
$$FPR = \frac{FP}{N},$$
which is the ratio between the number of false positives and the total number of negative examples
- True negative rate:
$$TNR = \frac{TN}{N},$$
which is the ratio between the number of true negatives and the total number of negative examples

- False negative rate:
  $$FNR = \frac{FN}{P}\,,$$
  which is the ratio between the number of false negatives and the total number of positive examples.

To help remember the last four rates, we note that the denominator can be deduced from the numerator in the definitions. For example, in $FPR = \frac{FP}{N}$, the numerator says "false positive" (whose true label is negative), which determines the denominator as N, the total number of negative examples.

These rates have been defined in different areas (such as statistics) and have different names. For example, TPR is also called *sensitivity* and FNR is the *miss rate*.

One simple way to understand these rates is to treat them as evaluation results only on examples of *single* class. For example, if we only use instances from the positive class ($y = +1$) to evaluate $f(\boldsymbol{x})$, the obtained accuracy and error rate are TPR and FNR, respectively. If only the negative class is used, the accuracy and error are TNR and FPR, respectively.

3.2 Area under the ROC curve

FPR and FNR calculate the error rates in two classes *separately*, which will not be affected by the class imbalance issue. However, we usually prefer one single number to describe the performance of one classifier, rather than two or more rates.

What is more, in most classifiers we can obtain many pairs of different (FPR, FNR) values. Suppose

$$f(\boldsymbol{x}) = 2\left(\llbracket \boldsymbol{w}^T\boldsymbol{x} > b \rrbracket - 0.5\right)\,,$$

that is, the prediction is positive ($+1$) if $\boldsymbol{w}^T\boldsymbol{x} > b$ (-1 if $\boldsymbol{w}^T\boldsymbol{x} \le b$). The parameters of this classifier include both $\boldsymbol{w}$ and b. After we obtain the optimal $\boldsymbol{w}$ parameters, we may utilize the freedom in the value of b to gradually change its value:

- When $b = -\infty$, the prediction is always $+1$ regardless of the values in $\boldsymbol{x}$. That is, FPR is 1 and FNR is 0.
- When b gradually increases, some test examples will be classified as negative. If the true label for one such example is positive, a false negative is created, hence the FNR will increase; if the true label for this example is negative, then it was a false positive when $b = -\infty$, hence, the FPR will be reduced. In general, when b increases, FPR will gradually decrease and FNR will gradually increase.
- Finally when $b = \infty$, the prediction is always -1. Hence, $FPR = 0$ and $FNR = 1$.

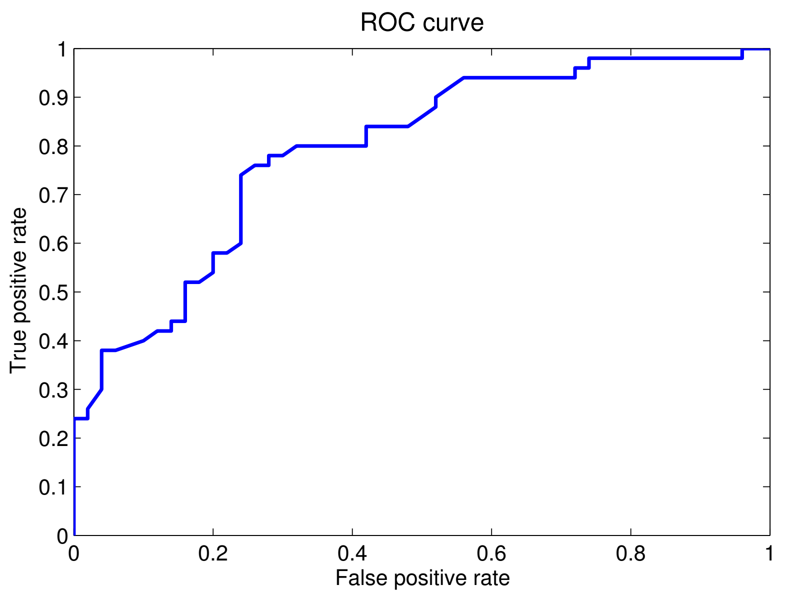


Figure 3: One example of the Receiver Operating Characteristics (ROC) curve.

The Receiver Operating Characteristics (ROC) curve records the process of these changes in a curve. The name ROC was first invented to describe the performance of finding enemy targets by radar, whose original meaning is not very important in our usage of this curve.

As shown in Figure 3, the x-axis is the false positive rate FPR and the y-axis is the true positive rate TPR. Because (why?)

$$TPR = 1 - FNR\,,$$

when b sweeps from ∞ to $-\infty$ by gradually decreasing b, the (FPR, TPR) pair will move from the coordinate $(0,0)$ to $(1,1)$, and the curve is non-decreasing.

Because the ROC curve summarizes the classifier's performance over the entire operational range (by changing the value of b), we can use the area under the ROC curve (called the AUC-ROC) as a single-number metric for its evaluation, which is also suitable in imbalanced tasks. In a classifier other than the linear classifier $\boldsymbol{w}^T\boldsymbol{x} + b$, we often can find another parameter that acts as a decision threshold (similar to b), and then can generate the ROC by altering this parameter.

The ROC of a classifier that randomly guesses its answer should be a line segment that connects $(0,0)$ and $(1,1)$ (i.e., the diagonal), and its AUC-ROC is 0.5. Hence, we expect any reasonable classifier to have its AUC-ROC larger than 0.5 in a binary classification problem.

The best ROC consists of two line segments: a vertical one from $(0,0)$ to $(0,1)$ followed by a horizontal one from $(0,1)$ to $(1,1)$. The classifier corresponding to this ROC always correctly classifies all test examples. Its AUC-ROC is 1.

3.3 Precision, recall, and F-measure

Precision and recall are two other measures suitable for evaluating imbalanced tasks. They are defined as

$$\text{Precision} = \frac{TP}{TP + FP}\,, \tag{17}$$

$$\text{Recall} = \frac{TP}{P}\,. \tag{18}$$

Note that the recall measure is simply another name for true positive rate.

Consider an image retrieval application. The test set has 10 000 images in total, in which 100 are related to *Nanjing University* (which we treat as positive class examples). When you type "Nanjing University" as the retrieval request, the retrieval system returns 50 images to you, which are those predicted as related to Nanjing University (i.e., predicted as positive). The returned results contain some images that indeed are related to Nanjing University (TP), but some are irrelevant (FP). Now two measures are important when you evaluate the retrieval system:

- Are the returned results precise? You want most (if not all) returned images to be positive/related to Nanjing University. This measure is the *precision*, the percentage of true positives among those that are predicted as positive—i.e., $\frac{TP}{TP+FP}$.
- Are all positive examples recalled? You want most (if not all) positive images to be included in the returned set. This measure is the *recall* (or TPR), the percentage of true positives among all positive images in the entire test set—i.e., $\frac{TP}{P}$.

It is not difficult to observe that these two performance measures are still meaningful in imbalanced tasks.

There are two widely used ways to summarize precision and recall into one number. Similar to the ROC curve, we can alter the threshold parameter in a classifier or a retrieval system and generate many pairs of (precision, recall) values, which form the precision–recall (PR) curve. Then, we can use the AUC-PR (area under the precision-recall curve) as the evaluation metric. Figure 4 shows one example PR curve.

One notable difference between the PR and ROC curves is that the PR curve is no longer non-decreasing. In most tasks, we will observe PR curves that zigzag, like the one in Figure 4.

The AUC-PR measure is more discriminating than AUC-ROC. One may observe AUC-ROC scores that are very close to each other for many classifiers. In this case, AUC-PR may be a better evaluation metric.

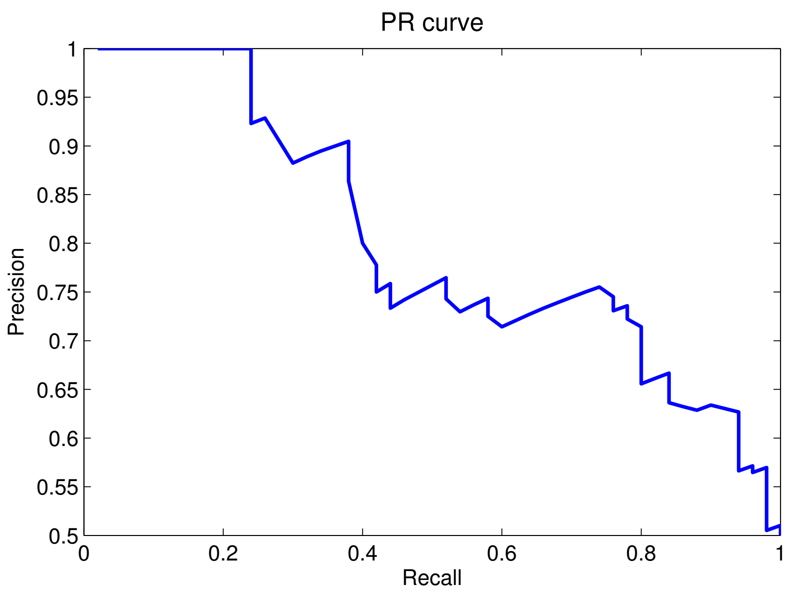


Figure 4: One example of the Precision–Recall (PR) curve.

The second way to combine precision and recall is the F-measure, which is defined as

$$F = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \,. \tag{19}$$

The F-measure is the harmonic mean of precision and recall, which is always between 0 and 1. A higher F-measure indicates a better classifier. It is also easy to see that

$$F = \frac{2TP}{2TP + FP + FN} \,. \tag{20}$$

The F-measure treats precision and recall equally. An extension of the F-measure is defined for a fixed $\beta > 0$, as

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} \,. \tag{21}$$

Either the precision or recall is considered as more important than the other, depending on the value of β. Note that the F-measure is a special case of F_β when $\beta = 1$.

An intuitive way to understand the relative importance between precision and recall is to consider extreme β values. If $\beta \to 0$, $F_\beta \to$ precision—that is, precision is more important for small β values. If $\beta \to \infty$, $F_\beta \to$ recall—that is, recall is more important for large β. When $\beta = 1$, the F-measure is symmetric about precision and recall—i.e., $F_1(\text{precision}, \text{recall}) = F_1(\text{recall}, \text{precision})$. Hence, they are equally important when $\beta = 1$. More properties and implications of F_β will be discussed in the exercises.

4 Can we reach 100% accuracy?

Till now, we have introduced ways to evaluate a system's accuracy or error in a few different tasks. What we really want is, in fact, a learning algorithm or recognition system that has *no* error. Can we implement such a system with 100% accuracy?

The answer to this question hinges on the data in your task. In the probabilistic interpretation, the data are expressed as random variables $(\boldsymbol{x}, y)$, and the examples are sampled i.i.d. from an underlying distribution $\Pr(\boldsymbol{x}, y)$ (or use the density $p(\boldsymbol{x}, y)$ for continuous random variables). If there exists an instance $\boldsymbol{x}^1$ and two different values of y (y^1 and y^2, and $y^1 \neq y^2$), such that $\Pr(\boldsymbol{x}^1, y^1) > 0$ and $\Pr(\boldsymbol{x}^1, y^2) > 0$, then a classifier with 100% (generalization) accuracy is impossible. If this classifier correctly classifies the example $(\boldsymbol{x}^1, y^1)$, it will err on $(\boldsymbol{x}^1, y^2)$, and vice versa.

4.1 Bayes error rate

We assume any classifier $f(\boldsymbol{x})$ is a valid function, which can only map $\boldsymbol{x}$ to a single fixed element y—that is, $f(\boldsymbol{x}^1) = y^1$ and $f(\boldsymbol{x}^1) = y^2$ cannot both happen if $y^1 \neq y^2$. This assumption is a valid one because in a classifier, we expect a deterministic answer. Hence, in either the training or test set, any instance $\boldsymbol{x}$ will only be assigned one groundtruth label y, and the prediction is also unique.

Now we can consider three cases for the above example.

- The true label for the instance $\boldsymbol{x}^1$ is set to be y^1. Then, the example $(\boldsymbol{x}^1, y^2)$ is considered as an error. Because $\Pr(\boldsymbol{x}^1, y^2) > 0$, this example will contribute at least $\Pr(\boldsymbol{x}^1, y^2)$ to the generalization error. That is, no matter what classifier is used, $(\boldsymbol{x}^1, y^2)$ will contribute $\Pr(\boldsymbol{x}^1, y^2)$ to this classifier's generalization error.

- The true label for the instance $\boldsymbol{x}^1$ is set to be y^2. Then, the example $(\boldsymbol{x}^1, y^1)$ is considered as an error. Because $\Pr(\boldsymbol{x}^1, y^1) > 0$, this example will contribute at least $\Pr(\boldsymbol{x}^1, y^1)$ to the generalization error. That is, no matter what classifier is used, $(\boldsymbol{x}^1, y^1)$ will contribute $\Pr(\boldsymbol{x}^1, y^1)$ to this classifier's generalization error.

- The true label for the instance $\boldsymbol{x}^1$ is set to be neither y^1 nor y^2. Then, both examples $(\boldsymbol{x}^1, y^1)$ and $(\boldsymbol{x}^1, y^2)$ are errors. Because both $\Pr(\boldsymbol{x}^1, y^1) > 0$ and $\Pr(\boldsymbol{x}^1, y^2) > 0$, these two examples will contribute at least $\Pr(\boldsymbol{x}^1, y^1) + \Pr(\boldsymbol{x}^1, y^2)$ to the generalization error of any possible classifier.

These three cases exhaust all possible cases for $(\boldsymbol{x}^1, y^1)$ and $(\boldsymbol{x}^1, y^2)$. If there are more than two y values that make $\Pr(\boldsymbol{x}^1, y) > 0$, analyses can be made in the same way. Hence, we reach the following conclusions:

- Errors are inevitable if there exist $\boldsymbol{x}^1$ and y^1, y^2 ($y^1 \neq y^2$) such that $\Pr(\boldsymbol{x}^1, y^1) > 0$ and $\Pr(\boldsymbol{x}^1, y^2) > 0$. Note that this statement is classifier agnostic—it is true regardless of the classifier.

- If a non-deterministic label is inevitable for an instance $\boldsymbol{x}^1$ (as in this particular example), we shall compare the two probabilities $\Pr(\boldsymbol{x}^1, y^1)$ and $\Pr(\boldsymbol{x}^1, y^2)$. If $\Pr(\boldsymbol{x}^1, y^1) > \Pr(\boldsymbol{x}^1, y^2)$, we set the groundtruth label for $\boldsymbol{x}^1$ to y^1 so that its contribution of unavoidable error $\Pr(\boldsymbol{x}^1, y^2)$ is small (compared to the case when the groundtruth label is set to y^2). If $\Pr(\boldsymbol{x}^1, y^2) > \Pr(\boldsymbol{x}^1, y^1)$, we must set the groundtruth label to y^2.

- Let us consider the general situation. There are m classes ($m \geq 2$), which are denoted by $y = i$ ($1 \leq i \leq m$) for the i-th class. For any instance $\boldsymbol{x}$, we should set its groundtruth label as

  $$y^\star = \arg\max_y \Pr(\boldsymbol{x}, y)$$

  to obtain the smallest unavoidable error, which is

  $$\sum_{y \neq y^\star} \Pr(\boldsymbol{x}, y)$$

  for $\boldsymbol{x}$.

If we repeat the above analyses for all possible instances $\boldsymbol{x} \in \mathbb{X}$, the generalization error of any classifier has a lower bound, as

$$\sum_{\boldsymbol{x} \in \mathbb{X}} \sum_{y \neq y^\star(\boldsymbol{x})} \Pr(\boldsymbol{x}, y) = 1 - \sum_{\boldsymbol{x} \in \mathbb{X}} \Pr(\boldsymbol{x}, y^\star(\boldsymbol{x}))\,, \tag{22}$$

in which $y^\star(\boldsymbol{x}) = \arg\max_y \Pr(\boldsymbol{x}, y)$ is the class index that makes $\Pr(\boldsymbol{x}, y)$ the largest for $\boldsymbol{x}$.

Equation 22 defines the *Bayes error rate*, which is a theoretical bound of the smallest error rate *any* classifier can attain. Comparing Equations 16 and 22, it is easy to see that the loss incurred by Bayes decision theory is exactly the Bayes error rate if the cost matrix is $\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$.

4.2 Groundtruth labels

The Bayes error rate is a theoretical bound, which assumes the joint density $p(\boldsymbol{x}, y)$ or joint probability $\Pr(\boldsymbol{x}, y)$ is known and can be analytically computed or exhaustively enumerated. The underlying distribution, however, is unknown in almost all real-world tasks. How then shall the groundtruth labels been decided?

In some applications, labels of training examples are accessible. For example, you try to predict day $t+1$'s NYSE closing index value based on the previous n days' closing NYSE index values, which is a regression task. On day $t+2$, you have observed the label of this regression task for day $t+1$. Hence, you can collect the training instances and labels using NYSE history data. A model learned from the history data can be used to predict next day's trading trend, although that prediction may be highly inaccurate—suppose a parallel universe does exist and our universe is split into two different parallel ones after the New York stock exchange closes on day $t+1$, the closing NYSE index values may be dramatically different in the two universes.

In most cases, the groundtruth labels are not observable and are obtained via human judgments. An expert in the task domain will manually specify labels for instances. For example, an experienced doctor will look at a patient's bone X-ray image and decide whether a bone injury exists or not. This type of human annotation is the major labeling method.

Human annotations are susceptible to errors and noise. In some difficult cases, even an expert will have difficulty in determining an accurate groundtruth label. An exhausted expert may make mistakes during labeling, especially when he or she has to label many examples. The labeling process can be extremely time-consuming, which also leads to high financial pressure—imagine you have to hire a top-notch doctor for 100 business days to label your data! In cases where both $\Pr(\boldsymbol{x}^1, y^1) > 0$ and $\Pr(\boldsymbol{x}^1, y^2) > 0$, the expert has to choose from y^1 or y^2, which may be difficult in some applications.

In summary, the labels can be a categorical value (for classification), a real number (for regression), a vector or matrix of real numbers, or even more complex data structures. The groundtruth labels may also contain uncertainty, noise, or error.

In this introductory book, we will focus on simple classification and regression tasks, and assume the labels are error and noise free *unless otherwise specified* (mainly because methods that explicitly handle label errors are advanced and beyond the scope of this book).

4.3 Bias–variance decomposition

Now that the error is inevitable, it is helpful to study what components contribute to the error. Hopefully, this understanding will help us in reducing the part of error that is larger than the Bayes error rate. We will use regression as an example to illustrate the *bias–variance decomposition*. Similar decompositions also exist for classifiers, but are more complex.

We need quite many assumptions to set up the stage for bias–variance decomposition. First, we have a function $F(\boldsymbol{x}) \in \mathbb{R}$, which is the function that generates our training data. However, the training data is susceptible to noise. A training example $(\boldsymbol{x}, y)$ follows

$$y = F(\boldsymbol{x}) + \epsilon \,, \tag{23}$$

in which $\epsilon \sim N(0, \sigma^2)$ is Gaussian random noise, which is *independent* of $\boldsymbol{x}$.

Next, we can sample (i.i.d.) from Equation 23 to generate different training sets, which may contain different numbers of training examples. We use a random variable D to represent the training set.

Third, we have a regression model f, which will generate a mapping $f(\boldsymbol{x}; D)$ by learning from the training set D, and predicts $f(\boldsymbol{x}; D)$ for any instance $\boldsymbol{x}$. We put D in prediction notation to emphasize that the prediction is based on the mapping learned from D. When different samples of D (different training sets) are used, we expect different prediction results for the same $\boldsymbol{x}$.

However, we assume that the regression method is deterministic. That is, given the same training set many times, it will produce the same mapping and predictions.

Finally, because of the i.i.d. sampling assumption, we just need to consider one specific instance $\boldsymbol{x}$ and examine the sources of error in predicting the regression output for $\boldsymbol{x}$.

An important note about these assumptions: since the underlying function F, the regression learning process, and $\boldsymbol{x}$ are deterministic, the randomness comes solely from the training set D. For example, $F(\boldsymbol{x})$ is deterministic, but $f(\boldsymbol{x}; D)$ is a random variable since it depends on D. For notational simplicity, we will simplify $\mathbb{E}_D[f(\boldsymbol{x}; D)]$ as $\mathbb{E}[f(\boldsymbol{x})]$, but it is essential to remember that the expectation is with respect to the distribution of D, and $f(\boldsymbol{x})$ means $f(\boldsymbol{x}; D)$.

Now we have all the tools to study $\mathbb{E}[(y - f(\boldsymbol{x}))^2]$, the generalization error in the squared error sense. Because we only consider one fixed example $\boldsymbol{x}$, we will write $F(\boldsymbol{x})$ as F and $f(\boldsymbol{x})$ as f. F is deterministic (hence $\mathbb{E}[F] = F$), and $\mathbb{E}[f]$ means $\mathbb{E}_D[f(\boldsymbol{x}; D)]$ (which is also a fixed value).

The error is then

$$\mathbb{E}[(y - f)^2] = \mathbb{E}[(F - f + \epsilon)^2] \,.$$

Because the noise ϵ is independent of all other random variables, we have

$$\begin{aligned}
\mathbb{E}[(y - f)^2] &= \mathbb{E}[(F - f + \epsilon)^2] && (24)\\
&= \mathbb{E}\left[(F - f)^2 + \epsilon^2 + 2(F - f)\epsilon\right] && (25)\\
&= \mathbb{E}[(F - f)^2] + \sigma^2 \,. && (26)
\end{aligned}$$

Note that

$$\mathbb{E}[\epsilon^2] = (\mathbb{E}[\epsilon])^2 + \mathrm{Var}(\epsilon) = \sigma^2 \,,$$

and

$$\mathbb{E}[(F - f)\epsilon] = \mathbb{E}[F - f]\mathbb{E}[\epsilon] = 0$$

because of the independence.

We can further expand $\mathbb{E}[(F - f)^2]$, as

$$\mathbb{E}[(F - f)^2] = (\mathbb{E}[F - f])^2 + \mathrm{Var}(F - f) \,. \tag{27}$$

For the first term in the RHS of Equation 27, because $\mathbb{E}[F - f] = F - \mathbb{E}[f]$, we have

$$(\mathbb{E}[F - f])^2 = (F - \mathbb{E}[f])^2 \,.$$

For the second term, because F is deterministic, we have

$$\mathrm{Var}(F - f) = \mathrm{Var}(-f) = \mathrm{Var}(f) = \mathbb{E}\left[(f - \mathbb{E}[f])^2\right] \,,$$

i.e., it equals the variance of $f(\boldsymbol{x}; D)$.

Putting all these results together, we have

$$\mathbb{E}[(y-f)^2] = (F-\mathbb{E}[f])^2 + \mathbb{E}\left[(f-\mathbb{E}[f])^2\right] + \sigma^2\,, \tag{28}$$

which is the bias–variance decomposition for regression. This decomposition states that the generalization error for any example $\boldsymbol{x}$ comes from three parts: the squared *bias*, the *variance*, and the noise.

- $F-\mathbb{E}[f]$ is called the bias, whose exact notation is

  $$F(\boldsymbol{x}) - \mathbb{E}_D[f(\boldsymbol{x};D)]\,. \tag{29}$$

  Because an expectation is taken on $f(\boldsymbol{x};D)$, the bias is not dependent on a training set. Hence, it is determined by the regression model—e.g., are you using a degree 2 or degree 15 polynomial? After we fix the form of our regression model, the bias is fixed, too.

- $\mathbb{E}\left[(f-\mathbb{E}[f])^2\right]$ is the variance of the regression *with respect to the variation in the training set.* The exact notation for the variance is

  $$\mathbb{E}_D\left[(f(\boldsymbol{x};D) - \mathbb{E}_D[f(\boldsymbol{x};D)])^2\right]\,. \tag{30}$$

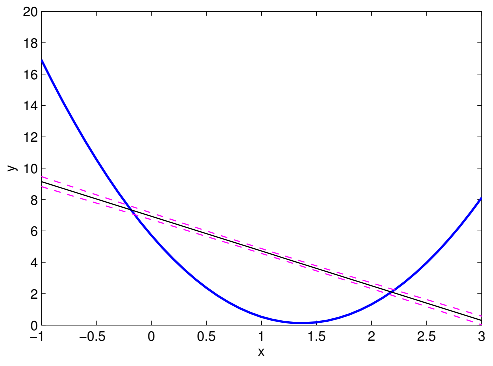
- σ^2 is the variance of the noise, which is irreducible (cf. the Bayes error rate in classification). Even if we know the underlying function $F(\boldsymbol{x})$ exactly and set $f=F$, the generalization error is still $\sigma^2>0$.

- This decomposition can be applied to any $\boldsymbol{x}$. Although we omitted $\boldsymbol{x}$ from the notations in Equation 28, the bias and variance will have different values when $\boldsymbol{x}$ changes.

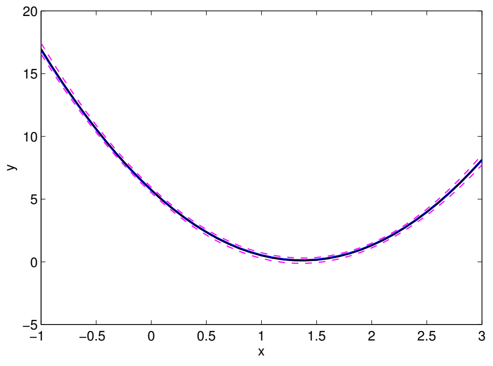
For the regression task in Equation 4, we can compute the bias and variance for any x, and the results are shown in Figure 5. Three regression models are considered: degree 1, 2, and 15 polynomials. In Figure 5, F is shown as the blue curve; $\mathbb{E}[f]$ is the black curve; the two purple curves are above and below $\mathbb{E}[f]$ by one standard deviation of f, respectively. Hence, the difference between the black and blue curves equals the bias, and the squared difference between the black and purple curves equals the variance at every x coordinate. In order to compute $\mathbb{E}[f]$ and $\mathrm{Var}(f)$, we i.i.d. sampled 100 training sets with the same size.

When the model has enough capacity (e.g., degree 2 or 15 polynomials), the bias can be very small. The black and blue curves in Figures 5b and Figure 5c are almost identical. However, the linear (degree 1 polynomial) model is too simple for this task, and its bias is huge: the distance between the black and blue curves is very large at most points.

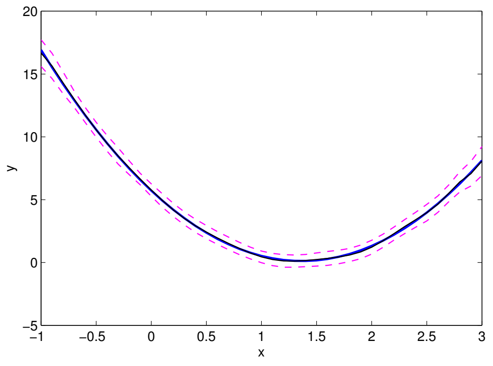
Also note that the bias and variance are not constant when x changes. In Figure 5a, the bias term changes quickly when x changes; while in Figure 5c, the variance is largest when x is close to both ends of the exhibited x range.



(a) degree 1 polynomial



(b) degree 2 polynomial



(c) degree 15 polynomial

Figure 5: Illustration of bias and variance in a simple polynomial regression task. The blue curve is the groundtruth, the black curve is the mean of 100 regression models learned from different training sets, and the two purple curves are the mean plus/minus one standard deviation of the regression models. ()

However, we do not want the model to be too complex. Although Figures 5c and Figure 5b both exhibit small distances between the black and blue curves (biases), the degree 15 polynomial model in Figure 5c shows quite large variances. That is, a slight change in the training set can lead to large variations in the learned regression results. We call this type of model *unstable*. In short, we want a learning model that has enough capacity (to have small bias) and is stable (to have small variance).

These requirements are contradictory to each other in most cases. For example, the linear model (cf. Figure 5a) is stable but its capacity is low. Adding regularization to complex models is one viable approach. Averaging many models (i.e., model ensemble) is also widely used.

5 Confidence in the evaluation results

At the end of this chapter, we briefly discuss the following question: after you have obtained an estimate of your classifier's error rate, how confident are you about this estimation?

One factor that affects the confidence is the size of your test or validation set. In some tasks such as the ImageNet image classification challenge, the test or validation set size is large (50 000 images in its validation set).[2] Researchers use the test or validation error rate with high confidence.

In some other problems with moderate numbers of examples, we can randomly divide all available examples into a training set and a test set. This split will be repeated k times (with $k = 10$ being a typical value), and the error rates evaluated in all k splits are averaged. The sample standard deviations (standard deviations computed from the k error rates) are almost always reported along with the average error rates.

When there are even fewer (e.g., a few hundreds) examples, within each of the k splits, cross-validation is used to calculate the cross-validation error rate in each split. The average and sample standard deviation of the k splits are reported.

5.1 Why averaging?

As shown by the bias–variance decomposition, the bias will not change after the classifier has been fixed. Hence, the error rate's variations are caused by different training and test sets. Hence, by reducing this variation we have more confidence in the error rate estimate.

Let E be a random variable corresponding to the error rate. Let E_1, E_2, $\dots$, E_k be k samples of E, computed from i.i.d. sampled training and test sets (with the same training and test set size). The error rate is often modeled as a

[2] http://www.image-net.org/

normal distribution—i.e., $E \sim N(\mu, \sigma^2)$. It is easy to verify that the average

$$\bar{E} = \frac{1}{k} \sum_{j=1}^{k} E_j \sim N\left(\mu, \frac{\sigma^2}{k}\right) , \tag{31}$$

that is, the average of multiple *independent* error rates will reduce the variance by a factor of k. This fact means that averaging can reduce the variance of the error rate estimates.

However, if we split a set of examples k times, the k training and k test sets cannot be independent because they are split from the same set of examples. Hence, we expect the average $\bar{E}$ to have a smaller variance than E, but not as small as $\frac{\sigma^2}{k}$. The smaller variance is still useful in providing a better estimate than using a single train/test split.

5.2 Why report the sample standard deviation?

If we know both $\bar{E}$ and σ (the population/true standard deviation, not the sample standard deviation), we can deduce how confident we are about $\bar{E}$.

Because $\bar{E} \sim N\left(\mu, \frac{\sigma^2}{k}\right)$, we have

$$\frac{\bar{E} - \mu}{\sigma/\sqrt{k}} \sim N(0, 1) .$$

In Figure 6 we show the p.d.f. of the standard normal $N(0, 1)$. The area of the green region is the probability $\Pr(|X| \leq 1)$ if $X \sim N(0, 1)$, which is 0.6827; the area of the green plus two blue regions is the probability $\Pr(|X| \leq 2)$ if $X \sim N(0, 1)$, which is 0.9545.

Because $\frac{\bar{E}-\mu}{\sigma/\sqrt{k}} \sim N(0, 1)$, we have $\Pr\left(\left|\frac{\bar{E}-\mu}{\sigma/\sqrt{k}}\right| \leq 2\right) = 0.9545$, or,

$$\Pr\left(\bar{E} - \frac{2\sigma}{\sqrt{k}} \leq \mu \leq \bar{E} + \frac{2\sigma}{\sqrt{k}}\right) > 0.95 . \tag{32}$$

That is, although we do *not* know the generalization error μ, a pretty confident (95%) estimate can be provided by $\bar{E}$ and σ. We know it is very likely (95% confident) that μ will be in the interval

$$\left[\bar{E} - \frac{2\sigma}{\sqrt{k}}, \bar{E} + \frac{2\sigma}{\sqrt{k}}\right] .$$

A smaller variance σ^2 means the confidence interval is smaller. Hence, it is useful to report the variance (or standard deviation).

However, two caveats exist about this interpretation. First, we do not know the true population standard deviation σ, which will be replaced by the sample standard deviation computed from the k splits. Hence, Equation 32 will not be correct anymore. Fortunately, the distribution after using the sample standard

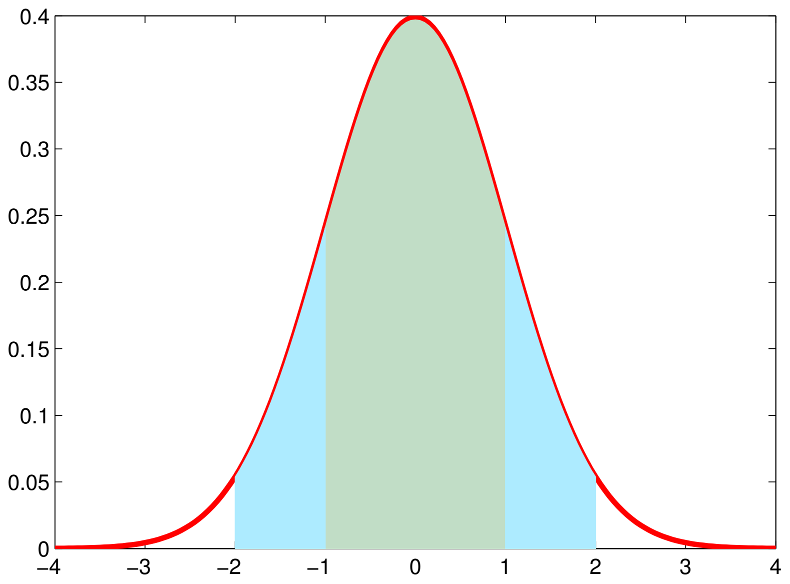


Figure 6: The standard normal probability density function and the one- and two-sigma ranges. The one sigma range has an area of 0.6827, and the two sigma range has an area of 0.9545. ()

deviation has a closed form, the Student's t-distribution. We will discuss the t-distribution more soon.

Second, $\bar{E}$ is a random variable in Equation 32. In practice, we need to replace it with the sample mean $\bar{e}$ computed from the k splits in one experiment. However, the interval

$$\left[\bar{e} - \frac{2\sigma}{\sqrt{k}}, \bar{e} + \frac{2\sigma}{\sqrt{k}}\right]$$

is deterministic, hence does *not* have a probability associated with it. Now, a 95% confidence means the following:

In one experiment, we will split the data k times, computing one $\bar{e}$ and one confidence interval accordingly. We will obtain 100 $\bar{e}$ and 100 confidence intervals in 100 experiments. Then, among the 100 experiments, around 95 times μ will fall into the respective intervals, but around 5 times it may not.

5.3 Comparing two classifiers

We have two classifiers f_1 and f_2, and we want to evaluate which one is better for a problem based on a set of examples D. We may evaluate f_1 and f_2, and estimate f_1's error rate as 0.08 ± 0.002 (meaning the sample mean and standard deviation for the error of f_1 are 0.08 and 0.002, respectively). Similarly, the evaluation result for f_2 is 0.06 ± 0.003. Now because

$$\frac{0.08 - 0.06}{0.002 + 0.003} = 4\,, \tag{33}$$

we know the distance between the two sample means is very large compared to the sum of the two sample standard deviations.[3] We are confident enough ($> 99\%$) to say that f_2 is better than f_1 on this problem.

However, if f_1 is 0.08 ± 0.01 and f_2 is evaluated as 0.06 ± 0.012, we have

$$\frac{0.08 - 0.06}{0.012 + 0.01} = 0.91\,,$$

that is, the one standard deviation confidence interval (whose confidence is < 0.6827) of f_1 and f_2's estimates overlap with each other. Hence, there is not enough confidence to say which one of the two classifiers is better.

Student's t-test[4] is useful for such comparisons. Full description of the t-test involves many different cases and details. In this chapter, we only introduce an application of the paired t-test, which can be used to compare two classifiers on a wide range of problems.

Suppose we evaluate f_1 and f_2 on n datasets, yielding errors E_{ij} where $i \in \{1, 2\}$ is the classifier index and j ($1 \le j \le n$) is the index to the datasets. Individually, E_{1j} and E_{2j} may not be confidently compared for most of the j values/datasets. However, this is a *paired* comparison because for any j, the same dataset is used by f_1 and f_2. Hence, we can study the properties of $E_{1j} - E_{2j}$ for all $1 \le j \le n$. Furthermore, we assume the datasets are not dependent on each other, thus different j values lead to *independent* $E_{1j} - E_{2j}$ random variables. Because the sum of two normally distributed variables has once again a normal distribution, we also assume $E_{1j} - E_{2j}$ is *normally distributed*.

For notational simplicity, we denote

$$X_j = E_{1j} - E_{2j}\,, \qquad 1 \le j \le n\,.$$

Under the above assumptions, we know X_j are i.i.d. samples from

$$X \sim N(\mu, \sigma^2)\,,$$

but the parameters μ and σ are unknown. The average

$$\bar{X} = \frac{1}{n}\sum_{j=1}^{n} X_j \sim N\left(\mu, \frac{\sigma^2}{n}\right)\,.$$

Hence,

$$\frac{\bar{X} - \mu}{\sigma/\sqrt{n}} \sim N(0, 1)\,.$$

[3]The rationale behind Equation 33 will be made clear in Chapter 6.

[4]William Sealy Gosset designed this test's statistics, and published his results under the pen name "Student."

Note that a particular parameter value $\mu = 0$ is of special interest to us. When $\mu = 0$, we have $\mathbb{E}[\bar{X}] = \mathbb{E}[X_j] = \mathbb{E}[E_{1j} - E_{2j}] = 0$, that is, on average f_1 and f_2 have the same error rate. When $\mu > 0$, f_1's error is higher than that of f_2, and f_1's error is smaller than that of f_2 if $\mu < 0$.

The t-test answers the following question: do f_1 and f_2 have different error rates? The *null hypothesis* is that $\mu = 0$, meaning there is no significant difference between f_1 and f_2. If f_1 is a new algorithm proposed by you and f_2 is a method in the literature, you may want the null hypothesis to be *rejected* (i.e., with enough evidence to believe it is not true) because you hope your new algorithm is better.

How can we confidently reject the null hypothesis? We can define a *test statistic* T, whose distribution can be derived *by assuming the null hypothesis is true* (i.e., when $\mu = 0$). For example, assuming $\mu = 0$, σ is known, and $T = \frac{\bar{X}}{\sigma/\sqrt{n}}$, we know the distribution of T is the standard normal $N(0, 1)$.

In the next step, you can compute the value of the statistic T based on your data, denoted as t, and say $t = 3.1$. Because T is a standard normal, we have $\Pr(|T| > 3) = 0.0027$, or 0.27%: it is a *small probability event* to observe $t = 3.1$ in one experiment—something must have gone wrong if you are not extremely unlucky.

The only thing that can be wrong is the assumption "the null hypothesis is true." Hence, when we observe unusual or extreme values for t, we have confidence to reject the null hypothesis.

Precisely speaking, we can specify a *significance level* α, which is a small number between 0 and 1. When assuming the null hypothesis is true, but the probability of observing an extreme value $T = t$ is smaller than α—i.e.,

$$\Pr(|T| > t) < \alpha \,,$$

we conclude that we are rejecting the null hypothesis at confidence level α.

In practice, σ is unknown. Let e_{ij} and x_j be samples for E_{ij} and X_j in one experiment, respectively. We can compute the sample mean and sample standard deviation for X as

$$\bar{x} = \frac{1}{n}\sum_{j=1}^{n} x_j \,, \tag{34}$$

$$s = \sqrt{\frac{1}{n-1}\sum_{j=1}^{n}(x_j - \bar{x})^2} \,, \tag{35}$$

which correspond to random variables $\bar{X}$ and S, respectively. Note that $\frac{1}{n-1}$ is used instead of $\frac{1}{n}$.

The new paired t-test statistics is

$$T = \frac{\bar{X}}{S/\sqrt{n}} \,, \tag{36}$$

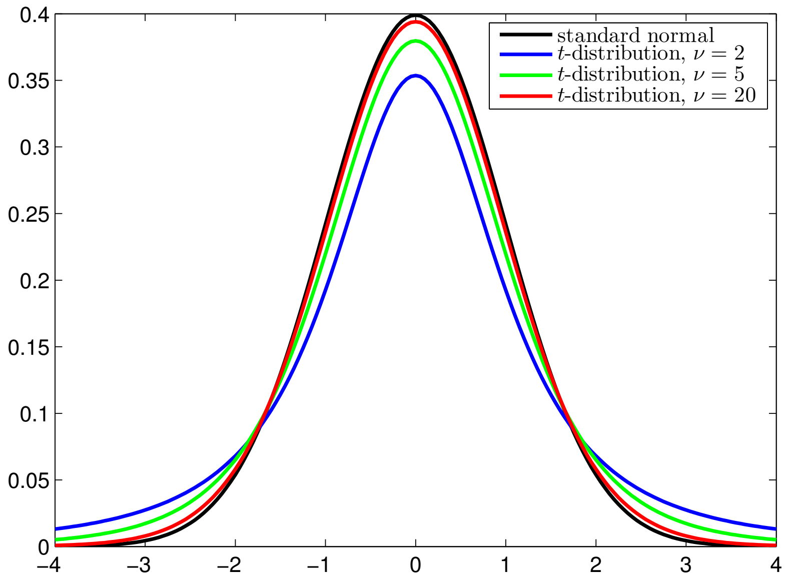


Figure 7: Probability density function of the Student's t-distribution with different degrees of freedom. ()

which replaces σ with S. Consequently, T is no longer the standard normal distribution. Fortunately, we know that T follows a Student's t-distribution with degrees of freedom $\nu = n - 1$. Note that the degrees of freedom are *not* n.

A t-distribution has one integer parameter: the degrees of freedom ν. Its p.d.f. is

$$p(t) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\,\Gamma(\frac{\nu}{2})} \left(1 + \frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}}, \tag{37}$$

in which Γ is the gamma function defined as

$$\Gamma(t) = \int_0^\infty x^{t-1} e^{-x} \, \mathrm{d}x \, .$$

As shown in Figure 7, the t-distribution is symmetric about 0 and looks like the standard normal distribution. However, the t-distribution has more density at its tails than the standard normal distribution. As the degrees of freedom ν grow, the t-distribution gets closer to $N(0,1)$. When $\nu \to \infty$, the t-distribution converges to the standard normal distribution.

Often the degrees of freedom are small. For one fixed degrees of freedom ν and a given significance level α, a critical value $c_{\nu,\alpha/2} > 0$ can be found in statistical tables, satisfying

$$\Pr(|T| > c_{\nu,\alpha/2}) = \alpha \, .$$

Because the t-distribution is symmetric and the region of extremal values can be at either side with equal probability, $\alpha/2$ is used in finding the critical value.

Hence, for one experiment, we can compute the sample statistic

$$t = \frac{\bar{x}}{s/\sqrt{n}} \,,$$

and reject the null hypothesis if

$$|t| > c_{\nu,\alpha/2} \,.$$

The paired t-test for comparing two classifiers is summarized in Algorithm 2.

Algorithm 2 The paired t-test for comparing two classifiers

```
1: Input: Two classifiers $f_1$ and $f_2$, whose error rate estimates on $n$ datasets are $e_{ij}$ ($i \in \{1,2\}$; $1 \le j \le n$).
2: Choose a significance level $\alpha$ (widely used values are 0.05 and 0.01).
3: Find the critical value $c_{n-1,\alpha/2}$.
4: $x_j \leftarrow e_{1j} - e_{2j}$, for $1 \le j \le n$.
5: $\bar{x} \leftarrow \frac{1}{n}\sum_{j=1}^{n} x_j$, $s \leftarrow \sqrt{\frac{1}{n-1}\sum_{j=1}^{n}(x_j - \bar{x})^2}$.
6: $t \leftarrow \frac{\bar{x}}{s/\sqrt{n}}$.
7: if $|t| > c_{n-1,\alpha/2}$ then
8:     We believe that $f_1$ and $f_2$ have different error rates. (The null hypothesis is rejected at confidence level $\alpha$.)
9: else
10:    We do not believe there is significant difference between the error rates of $f_1$ and $f_2$.
11: end if
```

Note that the paired t-test requires

- the errors e_{ij} are paired;
- the errors are independent for different j; and,
- the difference x_j is normally distributed.

Although slight or moderate violation of the third assumption is usually acceptable, the first two (paired and independent) assumptions cannot be violated.

Algorithm 2 specifies a *two-tailed* test, which only cares about whether f_1 and f_2 have the same/similar error rates or not. In many scenarios we need a *one-tailed* version of the test—for example, if you want to show that your new algorithm f_1 is better than f_2.

In the one-tailed test, if you want to show the error rate of f_1 is smaller than that of f_2, you need to change the critical value to $c_{n-1,\alpha}$ and further require $t < -c_{n-1,\alpha}$. If you want to show that f_2 is better than f_1, you need to have $t > c_{n-1,\alpha}$.

If statistical tables are not handy but you have a computer at your disposal (which happens to have a suitable software package installed on it), you can compute the critical value by one line of code in your favorite software. For example, let us compute the two-tailed critical value for the paired t-test, with $\nu = 6$ and $\alpha = 0.05$.

Because the t-distribution is symmetric, we have

$$\Pr(T > c_{\nu,\alpha/2}) = \alpha/2\,.$$

Let Φ denote the c.d.f. of a t-distribution with ν degrees of freedom; we have

$$\Phi(c_{\nu,\alpha/2}) = 1 - \alpha/2\,.$$

Hence,

$$c_{\nu,\alpha/2} = \Phi^{-1}(1 - \alpha/2)\,,$$

in which $\Phi^{-1}(\cdot)$ is the inverse c.d.f. function.

The Matlab/Octave function `tinv(p,`ν`)` calculates the inverse c.d.f. value for probability `p` and degrees of freedom ν. One single Matlab/Octave command

```
tinv(1-0.05/2,6)
```

tells us that the critical value for a two-tailed, 6 degrees of freedom, significance level 0.05, paired t-test is 2.4469. Similarly, for the one-tailed test, we have the critical value computed as

```
tinv(1-0.05,6) ,
```

which is 1.9432.

The t-test is a relatively simple statistical test which can be adopted to compare classification methods (or other learning algorithms/systems). Many other tests (e.g., various rank tests) are useful in this respect too. We hope its introduction in this chapter will help readers in understanding other statistical tests, but will not go into details of these tests.

One final note: whenever you want to apply a statistical test to your data, the first thing to check is whether your data satisfy the assumptions of that particular test!

Exercises

1. In a binary classification problem, we know $P = N = 100$ (i.e., there are 100 positive and 100 negative examples in the test set). If $FPR = 0.3$ and $TPR = 0.2$, then what is the precision, recall, F_1 score? What is its accuracy and error rate?

2. (Linear regression) Consider a set of n examples $(\boldsymbol{x}_i, y_i)$ $(1 \leq i \leq n)$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. A *linear regression* model assumes

$$y = \boldsymbol{x}^T \boldsymbol{\beta} + \epsilon$$

for any example $(\boldsymbol{x}, y)$, where ϵ is a random variable modeling the regression error and $\boldsymbol{\beta} \in \mathbb{R}^d$ are the parameters of this model. For the i-th example, we have $\epsilon_i = y_i - \boldsymbol{x}_i^T \boldsymbol{\beta}$.

(a) Express the linear regression task as an optimization problem over the training set, using the training examples, the parameters $\boldsymbol{\beta}$, and the squared error ($\sum_{i=1}^n \epsilon_i^2$, which is the MSE times the number of examples).

(b) We can organize the training examples $\boldsymbol{x}_i$ into a $n \times d$ matrix X, whose i-th row is the vector $\boldsymbol{x}_i^T$. Similarly, we can organize y_i into a vector $\boldsymbol{y} \in \mathbb{R}^n$, with y_i in the i-th row. Rewrite the optimization problem in (a) using X and $\boldsymbol{y}$.

(c) Find the optimal values for $\boldsymbol{\beta}$. For now, assume $X^T X$ is invertible. This solution is called the *ordinary linear regression* solution.

(d) When there are more dimensions than examples—i.e., when $d > n$—will $X^T X$ be invertible?

(e) If we add a regularizer

$$\mathcal{R}(\boldsymbol{\beta}) = \boldsymbol{\beta}^T \boldsymbol{\beta}$$

with a tradeoff parameter λ ($\lambda > 0$) to a linear regression, what effect will that regularizer have? Linear regression with this regularizer is called the *ridge regression*, and this regularizer is a special case of the *Tikhonov regularization*.

(f) Express the optimization problem in ridge regression using X, $\boldsymbol{y}$, $\boldsymbol{\beta}$ and λ. Find the solution.

(g) Ordinary linear regression will encounter difficulties when $X^T X$ is not invertible. How will ridge regression help in this aspect?

(h) What will be the ridge regression solution if $\lambda = 0$? What if $\lambda = \infty$?

(i) Can we learn a good λ value by treating λ as a regular parameter (instead of a hyperparameter)—that is, by minimizing the ridge regression loss function jointly over λ and $\boldsymbol{\beta}$ on the training set (without using a validation set)?

3. (Polynomial regression) The polynomial regression model $y = f(x) + \epsilon$ assumes the mapping f is a polynomial. A degree d polynomial is of the form

$$f(x) = \sum_{i=0}^{d} p_i x^i \,, \tag{38}$$

with $d+1$ parameters p_i $(0 \le i \le d)$. Use ordinary linear regression to find the optimal parameters for polynomial regression. (Hint: Set the parameters of the linear regression to $\boldsymbol{\beta} = (p_0, p_1, \dots, p_d)^T$.)

4. (F_β measure) Answer the following two questions about the F_β measure.

   (a) Prove that $0 \le F_\beta \le 1$ for any $\beta \ge 0$.

   (b) When β takes different values, the F_β measure places different relative importance on the precision and recall. Which one (precision or recall) is more important if $\beta > 1$? Which one is more important if $0 \le \beta < 1$? (Hint: What is the speed of F_β's change when the precision or recall changes?)

5. (AUC-PR and AP) We have not discussed the details of how the AUC-PR measurement is calculated. For a binary classification task, we assume every example $\boldsymbol{x}$ has a score $f(\boldsymbol{x})$, and sort the test examples in descending order of these scores. Then, for every example, we set the classification threshold as the current example's score (i.e., only this example and examples before it are classified as positive). A pair of precision and recall values are computed at this threshold. The PR curve is drawn by connecting nearby points using line segments. Then, AUC-PR is the area under the PR curve.

   Let (r_i, p_i) denote the i-th recall and precision rates $(i = 1, 2, \dots)$. When computing the area, the contribution between r_i and r_{i-1} is calculated using the *trapezoidal* interpolation $(r_i - r_{i-1})\frac{p_i + p_{i-1}}{2}$, in which $r_i - r_{i-1}$ is the length on the x-axis, and p_i and p_{i-1} are the lengths of two vertical lines in the y-axis. Summing over all i values, we obtain the AUC-PR score. Note that we assume the first pair $(r_0, p_0) = (0, 1)$, which is a pseudo-pair corresponding to the threshold $+\infty$.

   (a) For the test set with 10 examples (indexed from 1 to 10) in Table 3, calculate the precision (p_i) and recall (r_i) when the threshold is set as the current example's $f(x_i)$ value. Use class 1 as positive, and fill these values in Table 3. Fill the trapezoidal approximation $(r_i - r_{i-1})\frac{p_i + p_{i-1}}{2}$ in the "AUC-PR" column for the i-th row, and fill their sum in the last row.

   (b) *Average precision* (AP) is another way to summarize the PR curve into one number. Similar to AUC-PR, AP approximates the contribution between r_i and r_{i-1} using a rectangle, as $(r_i - r_{i-1})p_i$. Fill in this approximation into the "AP" column for the i-th row, and fill their sum in

Table 3: Calculation of AUC-PR and AP.

index	label	score	precision	recall	AUC-PR	AP
0			1.0000	0.0000	-	-
1	1	1.0				
2	2	0.9				
3	1	0.8				
4	1	0.7				
5	2	0.6				
6	1	0.5				
7	2	0.4				
8	2	0.3				
9	1	0.2				
10	2	0.1				
					(?)	(?)

the last row. Both AUC-PR and AP summarize the PR curve, hence they should be similar to each other. Are they?

(c) Both AUC-PR and AP are sensitive to the order of labels. If the label of the 9-th and the 10-th rows are exchanged, what is the new AUC-PR and AP?

(d) Write a program to calculate both AUC-PR and AP based on the labels, scores, and the positive class. Validate your program's correctness using the example test set in Table 3.

6. We can use the k-NN method for regression. Let $D = \{\boldsymbol{x}_i, y_i\}_{i=1}^n$ be a training set, where the labels $y \in \mathbb{R}$ are generated by $y = F(\boldsymbol{x}) + \epsilon$, in which the true regression function F is contaminated by noise ϵ to generate the labels y. We assume the random noise ϵ is independent of anything else, $\mathbb{E}[\epsilon] = 0$ and $\mathrm{Var}(\epsilon) = \sigma^2$.

   For any test example $\boldsymbol{x}$, the k-NN method finds its k (k is a positive integer) nearest neighbors in D, denoted by $\boldsymbol{x}_{nn(1)}, \boldsymbol{x}_{nn(2)}, \ldots, \boldsymbol{x}_{nn(k)}$, where $1 \le nn(i) \le n$ is the index of the i-th nearest neighbor. Then, the prediction for $\boldsymbol{x}$ is

$$f(\boldsymbol{x}; D) = \frac{1}{k} \sum_{i=1}^{k} y_{nn(i)} \,.$$

   (a) What is the bias–variance decomposition for $\mathbb{E}[(y - f(\boldsymbol{x}; D))^2]$, in which y is the label for $\boldsymbol{x}$? Do not use abbreviations (Equation 28 uses abbreviations, e.g., $\mathbb{E}[f]$ should be $\mathbb{E}_D[f(\boldsymbol{x}; D)]$.) Use $\boldsymbol{x}$, y, F, f, D and σ to express the decomposition.

   (b) Use $f(\boldsymbol{x}; D) = \frac{1}{k} \sum_{i=1}^{k} y_{nn(i)}$ to compute $\mathbb{E}[f]$ (abbreviations can be used from here on).

(c) Replace the f term in the decomposition by using $\boldsymbol{x}$ and y.

(d) What is the variance term? How will it change when k changes?

(e) What is the squared bias term? How will it change with k? (Hint: consider $k = n$)?

7. (Bayes decision theory) Consider a binary classification task, in which the label $y \in \{1, 2\}$. If an example $x \in \mathbb{R}$ belongs to class 1, it is generated by the class conditional p.d.f. $p(x|y = 1) = N(-1, 0.25)$, and a class 2 example is sampled from the class conditional distribution $p(x|y = 2) = N(1, 0.25)$. Suppose $\Pr(y = 1) = \Pr(y = 2) = 0.5$.

   (a) What is the p.d.f. $p(x)$?

   (b) Let us use the cost matrix $\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$. Show that for any x, if we choose $f(x) = \arg\max_y p(y|x)$ to be our prediction for x, the cost $\mathbb{E}_{(x,y)}[c_{y,f(x)}]$ is minimized, and hence it is the optimal solution. Is this rule optimal if $y \in \{1, 2, \ldots, C\}$ $(C > 2)$ (i.e., in a multi-class classification problem)?

   (c) Using the cost matrix $\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ and Bayes decision theory, which classification strategy will be optimal for this task? What is the Bayes risk in this example?

   (d) If the cost matrix is $\left[\begin{smallmatrix} 0 & 10 \\ 1 & 0 \end{smallmatrix}\right]$ (i.e., when the true label is 1 but the prediction is 2, the cost is increased to 10). What is the new decision rule?

8. (Stratified sampling) Let D be a training set with only 10 examples, whose labels are 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, respectively. This dataset is both small in size and imbalanced. We need cross-validation during evaluation, and 2-fold CV seems a good choice.

   (a) Write a program to *randomly* split this dataset into two subsets, with five examples in each subset. Repeat this random split 10 times. The histogram of class 1 examples in these two subsets can be $(0, 2)$ or $(1, 1)$—one subset has zero (two) and the other has two (zero) class 1 examples, or every subset has exactly one class 1 example. In your 10 splits, how many times does $(0, 2)$ appear? (Note: this number can be different if you perform the experiments multiple times.)

   (b) What is the probability that $(0, 2)$ will appear in one random split of these 10 examples?

   (c) In your 2-fold CV evaluation, if the split's class 1 distribution in the two subsets is $(0, 2)$, how will it affect the evaluation?

   (d) One commonly used way to avoid this issue to use *stratified sampling*. In stratified sampling, we perform the train/test split for *every* class separately. Show that if stratified sampling is used, the distribution of class 1 examples will always be $(1, 1)$.

9. (Confusion matrix) In a classification problem with K classes, the cost matrix is of size $K \times K$: $C = [c_{ij}]$, in which c_{ij} is the cost when one example belongs to class i but is predicted to be in class j. Similarly, a confusion matrix is a $K \times K$ matrix: $A = [a_{ij}]$, in which a_{ij} is the number of class i examples that are classified as belonging to class j.

   Let the confusion matrix be computed based on a test set with N examples. We often normalize the confusion matrix to obtain $\hat{A}$, by $\hat{a}_{ij} = \frac{a_{ij}}{\sum_{k=1}^{K} a_{ik}}$. Hence, the sum of all elements in any row of $\hat{A}$ equals 1. We call $\hat{A}$ the normalized confusion matrix.

   (a) Prove that the total cost for the test set equals $\operatorname{tr}(C^T A)$.

   (b) In an imbalanced classification problem, do you prefer the confusion matrix or the normalized one? Why?

10. (McNemar's test) Find resources about McNemar's test.[5] *Carefully* read through these resources until you believe you have understood when this test can be applied and how to apply it to compare two classifiers.

[5] For example, in your school's library or search on the internet.