Huawei AI Academy Training Materials

Machine Learning

Huawei Technologies Co., Ltd.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services, and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services, and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees, or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express, or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base Bantian, Longgang, Shenzhen 518129

Website: http://e.huawei.com

Contents

1 Machine Learning

Machine learning is currently a mainstream research direction in the field of artificial intelligence (AI), involving multiple disciplines such as probability theory, statistics, and convex optimization. This chapter describes the definition of machine learning algorithms, the machine learning process, common machine learning algorithms, and the concepts such as hyper parameters, gradient descent, and cross validation.

1.1 Machine Learning Algorithms

1.1.1 Overview

Machine learning (including deep learning) is the study of learning algorithms. A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*. For example, identifying spam emails is a task *T*. We can easily finish the task because we have accumulated a lot of experience in our daily life. The experience may cover emails, spam messages, and even TV ads. By summarizing the experience, we find that emails that are sent from unknown users and contain words such as "discount" and "zero risk" are more likely to be spam emails. Based on the knowledge of spam emails, we can determine whether an unread email is a spam email, as shown in the left part in Figure 1-1. So can we write a computer program to simulate the above process? As shown in the right part in Figure 1-1, we can prepare a large number of emails and manually filter out spam ones as the computer program experience *E*. However, the computer program cannot automatically summarize the experience. In this case, a machine learning algorithm is needed to train the computer program. A trained computer program is called a model. Generally, a larger number of emails used for training indicates a better trained model, that is, a larger value of the performance measure *P*.

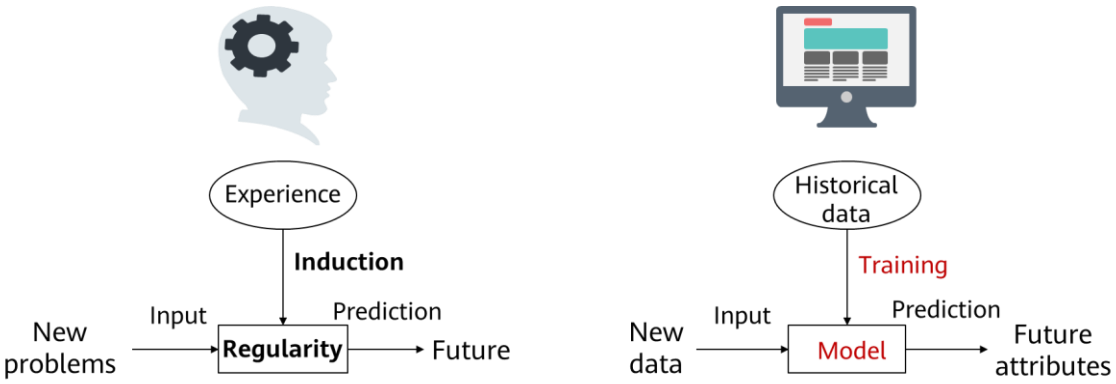


Figure 1-1 Machine learning mode

It is very difficult to identify spam emails by using conventional programming methods. In theory, we should be able to find a set of rules that are met by any spam emails but not by normal emails. This approach to problem solving using explicit programming is called a rule-based approach. In practice, it is almost impossible to find such a set of rules. Therefore, a statistics-based approach is used by machine learning to solve problems. As we all know, machine learning is an algorithm that enables machines to automatically learn rules based on samples. Compared with the rule-based approach, statistics-based approach can learn more complex rules or rules difficult to describe, and therefore can process more complex tasks.

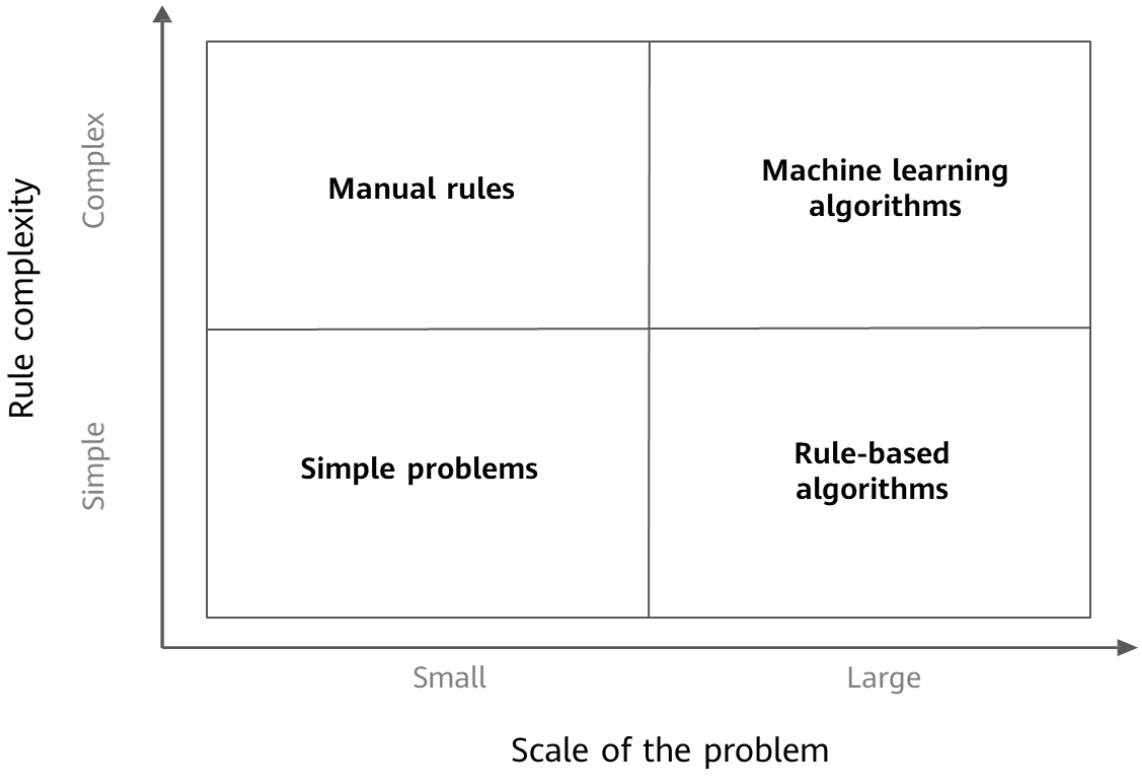


Figure 1-2 Application scenarios of machine learning algorithms

Machine learning algorithms are highly expressive and can solve many problems in the AI field. However, this does not mean that machine learning is the first choice in any case.

As shown in 0, machine learning applies to complex solutions or scenarios involving a large amount of data and unknown data probability distribution. It is also applicable to other scenarios but often generates higher costs than conventional methods. Take the second quadrant in 0 as an example. If the problem has a scale small enough for the problem to be manually solved, you do not need to use machine learning algorithms. Generally, machine learning is applicable to:

- Scenarios with complex rules or rules difficult to describe, for example, facial recognition and speech recognition.
- Scenarios with data distribution changing over time and constant readaptation of programs required, for example, sales trend forecast.

1.1.2 Rational Understanding of Machine Learning Algorithms

The essence of machine learning algorithms is function fitting. Assuming that f is a target function, the objective of machine learning algorithms is to output a hypothesis function g, so that $g(x)$ approaches $f(x)$ as far as possible for the input x in any definition domain. A simple example is the probability density estimation in statistics. According to the law of large numbers, the height of all Chinese should be subject to a normal distribution. Although the probability density function f of the normal distribution is unknown, we can estimate the mean and variance of the distribution by using the sampling method, and then estimate f.

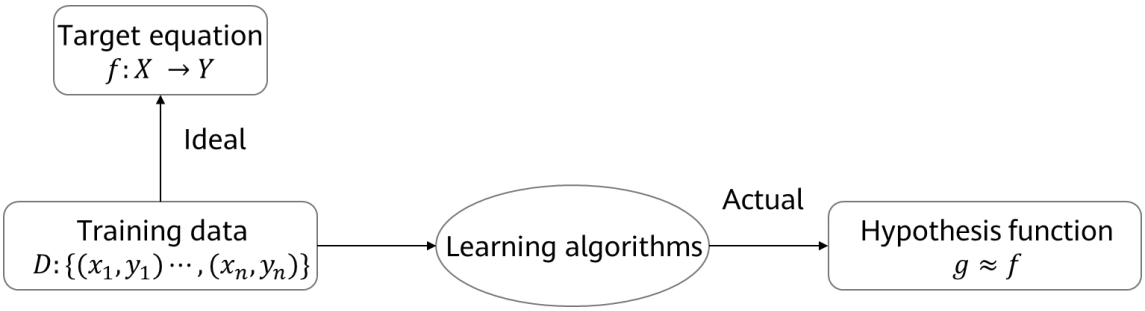


Figure 1-3 Relationship between the hypothesis function and target function

This practice also applies to a more complex situation, as shown in 0. For a given task, we can collect a large amount of training data. The data must satisfy a certain target function f. Otherwise, it is meaningless to learn such a task. Machine learning algorithms can provide, by analyzing the training data, a hypothesis function g that is as similar to the target function f as possible. Therefore, the output of machine learning algorithms cannot be the same as the target function. However, with increasing training data, the hypothesis function g gradually approaches the target function f to achieve satisfactory precision.

Notably, the existence of the target function f is sometimes highly abstract. For a typical image classification task, the target function is a mapping from an image set to a category set. To enable a computer program to process logical information such as images and categories, you need to map the images or categories to a scalar, a vector, or a matrix in a particular encoding manner. For example, you can assign a sequence number starting with 0 to each category to map the category to a scalar. Different one-hot vectors can also be used to represent different categories, and this manner is referred

to as one-hot encoding. The image encoding mode is slightly complex, and is generally represented by a three-dimensional matrix. With this encoding mode, we can consider the definition domain of the target function *f* as a set of three-dimensional matrices, and its value range as a set of a series of label numbers. Although the encoding process is not a part of machine learning algorithms, in some cases, the selection of encoding mode also affects efficiency of machine learning algorithms.

1.1.3 Main Problems Solved by Machine Learning

Machine learning algorithms can deal with many types of tasks. The most typical types of tasks are classification, regression, and clustering. Classification and regression are major types of prediction tasks, accounting from 80% to 90%. The output of classification is discrete category labels, and the output of regression is continuous numbers.

A classification task requires a computer program to specify a specific one of the *k* categories for the input. To accomplish this task, machine learning algorithms usually output a mapping from the definition domain *D* to the category labels {1, 2, ..., k}. The image classification algorithm mentioned above deals with classification tasks.

In a regression task, a computer program needs to predict the output for the given input. The output of a machine learning algorithm is usually a mapping from the definition domain *D* to the real number domain *R*. An example of this task type is to predict the claim amount of an insured person (to set the insurance premium) or predict the security price. Classification tasks can also be classified as regression tasks. You can obtain the classification result by predicting the probability that an image belongs to each category.

A clustering task divides data into multiple categories based on the internal similarity of the data. Different from a classification task, the dataset of a clustering task does not contain a manually added category label. Clustering algorithms try to achieve higher data similarity in the same category than that between different categories, so as to implement classification. Clustering algorithms can be applied to scenarios such as image retrieval and user profiling.

1.2 Machine Learning Classification

Machine learning can be classified into supervised learning and unsupervised learning. The training data used for supervised learning contains manually added labels, while that of unsupervised learning does not. If some data in a dataset contains labels but most data does not, this type of machine learning is called semi-supervised learning. Reinforcement learning focuses on multi-step decision-making and automatically collects data for learning during interaction with the environment.

1.2.1 Supervised Learning

Figuratively speaking, supervised learning allows a computer to compare its answers with standard ones when handling multiple-choice questions. The computer tries to adjust its model parameters, trying to make the predicted answers as consistent as possible with the standard ones, and finally learns how to complete the task. Supervised learning can train an optimal model with required performance based on samples with known labels. This trained model can map input to output to predict unknown data.

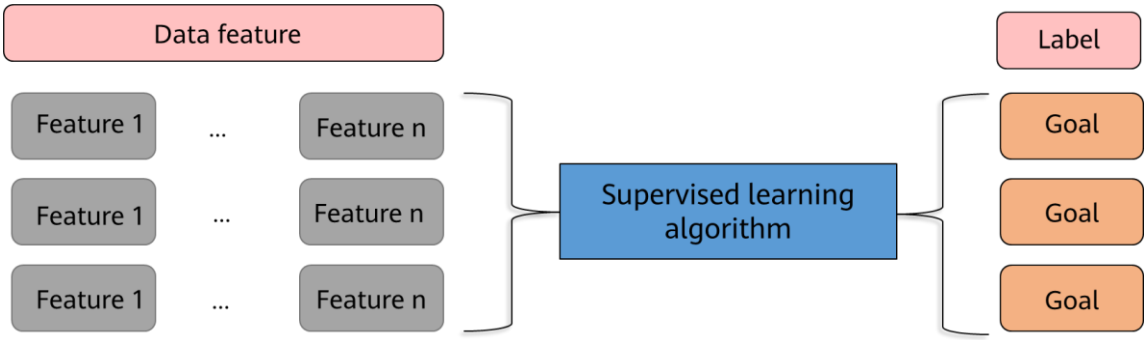


Figure 1-4 Supervised learning algorithm

0 shows the functions of supervised learning algorithms in a highly simplified manner. The features mentioned in the figure can be simply understood as data items. Although this understanding is incomplete in some sense, it does not affect the description of supervised learning algorithms. A supervised learning algorithm takes features as input and the predicted value of the target as output. 0 provides an example. In this example, whether a user enjoys sports is predicted based on the weather. A similar algorithm can be applied to scenarios such as product recommendation. Each row in the table is a training sample, which records a weather feature of a specific day and whether the user enjoys sports.

Feature			Objective
Weather	Temperature	Wind Speed	Enjoy Sports
Sunny	Warm	Strong	Yes
Rainy	Cold	Fair	/
Sunny	Cold	Weak	Yes

Figure 1-5 Example data

The input (features) and output (targets) of supervised learning algorithms can be continuous or discrete. When the value of the target variable is continuous, the output of a supervised learning algorithm is called a regression model. A regression model reflects the features of attribute values of samples in a sample dataset. A function is used to express the sample mapping relationship and further discover the dependency between attribute values. The attribute values include features and target. Regression models are widely used in time series forecasting, for example, predicting how much profit stocks can bring in the next week and what is the temperature in Celsius tomorrow. Correspondingly, when the target variable takes a discrete value, the output of the learning algorithm is referred to as a classification model. You can map samples in a sample dataset to a given category by using a classification model, for example, whether a traffic jam will occur on a highway during tomorrow's morning rush hours, and which of a CNY 5 voucher and a 25% discount will attract more customers.

Although the value range of a regression model can be an infinite set, the output of a classification model is usually limited. This is because the size of a dataset cannot increase infinitely, and the number of categories in the dataset is the same as the number of training samples at most. Therefore, the number of categories cannot be infinite. When a classification model is trained, a category set *L* usually needs to be manually specified for the model to select a category for output. The size of the category set *L* is generally denoted as *K*, which indicates the number of possible categories.

1.2.2 Unsupervised Learning

Compared with supervised learning, unsupervised learning is like letting a computer handle multiple-choice questions without telling it the right answers. In this case, it is difficult for the computer to give correct answers. However, by analyzing the relationship between these questions, the computer can classify the questions so that the multiple choice questions in each category have the same answer. As shown in 0, unsupervised learning algorithms do not require sample labeling, but directly model the input datasets.

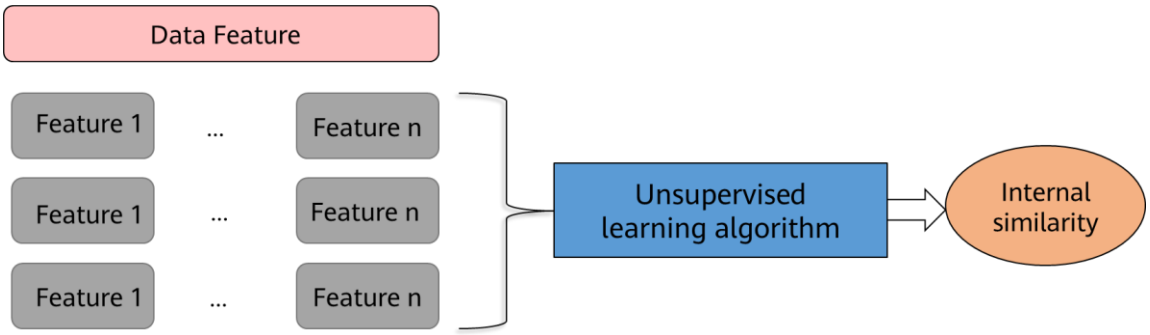


Figure 1-6 Unsupervised learning algorithm

The clustering algorithm is a typical unsupervised learning algorithm. The algorithm only needs to put things with a high similarity together. For a new sample, you only need to calculate the similarity between the new sample and an existing sample, and then classify the new sample based on the similarity. Biologists have long been using the idea of clustering to study interspecies relationships. As shown in Figure 1-7, after the iris is drawn on a two-dimensional plane based on the sepals and petals, the iris is obviously divided into three clusters. Samples in a sample dataset are classified into several categories based on the clustering model. Samples belonging to the same category have high similarity. The application scenarios of the clustering model are as follows: which audience likes to watch movies of the same subject, and which of these components are damaged in a similar way.

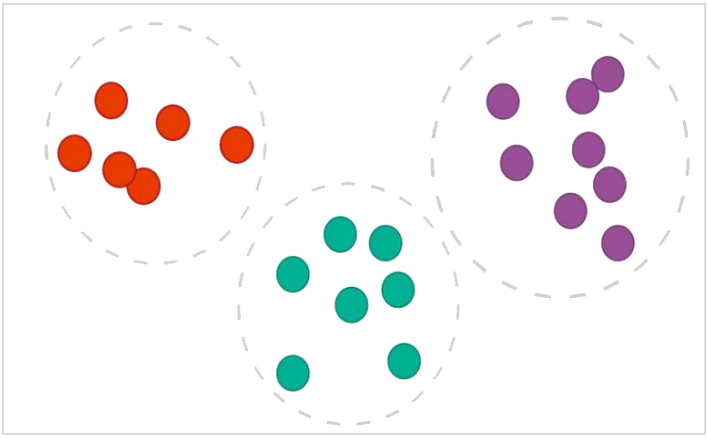


Figure 1-7 Clustering algorithm example

1.2.3 Semi-supervised Learning

Semi-supervised learning is a fusion of supervised learning and unsupervised learning. It is a machine learning task that automatically uses a large amount of unlabeled data to assist learning of a small amount of labeled data. In a conventional supervised learning algorithm, a computer program needs to learn a large number of labeled training samples to build a model for predicting the labels of new samples. For example, in a classification task, a label indicates the category of a sample while in a regression task, a label is a real-value output of the sample. As our data collection and storage capabilities are developing, we can easily gain a large amount of unlabeled data from many tasks. However, labeling the data is labor-consuming and time-consuming. For example, a user needs to mark interested websites for website recommendation. However, few users are willing to spend a lot of time marking websites. Therefore, the number of marked websites is small. There are countless websites on the Web that can be used as unmarked data.

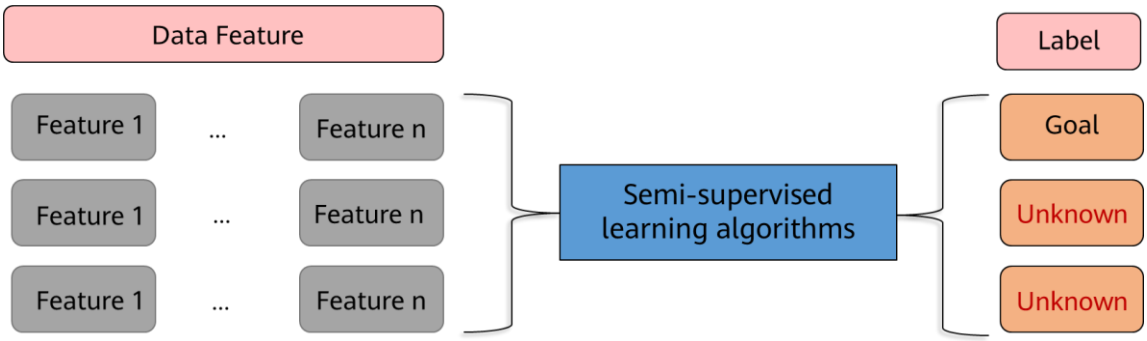


Figure 1-8 Semi-supervised learning algorithm

As shown in 0, semi-supervised learning does not require manual labeling of all samples as supervised learning does, nor is semi-supervised learning completely independent from targets as unsupervised learning does. In a dataset used for semi-supervised learning, only a few samples are labeled. The iris classification task shown in Figure 1-7 is used as an example. A small amount of supervision information is added to the dataset, as shown in 0. Red objects indicate Setosa samples, green objects indicate Versicolor samples,

purple objects indicate Virginica samples, and gray objects indicate unknown samples. Assume that the output of the clustering algorithm, which has been introduced in unsupervised learning, is shown in the gray dashed circle in the figure. Collect statistics on the number of samples of each category in these circles, and use the category with the largest number of samples as the cluster category. For example, the cluster in the upper left corner belongs to Setosa, and the cluster in the upper right corner belongs to Virginica. By combining unsupervised learning algorithms with supervision information, semi-supervised learning algorithms can bring higher accuracy with lower labor cost.

Figure 1-9 Iris dataset with supervision information

1.2.4 Reinforcement Learning

Reinforcement learning is mainly used to solve multi-step decision-making problems in scenarios such as chess, electronic games, and visual navigation. Different from the problems studied in supervised learning and unsupervised learning, it is often difficult to find accurate answers for multi-step decision-making problems. Taking chess as an example, it takes about 10^{170} operations to exhaust the game's results (There are no more than 10^{80} atoms in the universe.). So for a given situation, it is generally difficult to find the perfect move.

Another feature of multi-step decision-making problems is that it is easy to define a reward function to evaluate task completion. The reward function of chess can be defined as whether to win the game. The reward function for electronic games can be defined as a score. The goal of reinforcement learning is to find an action strategy π to maximize the value of the reward function.

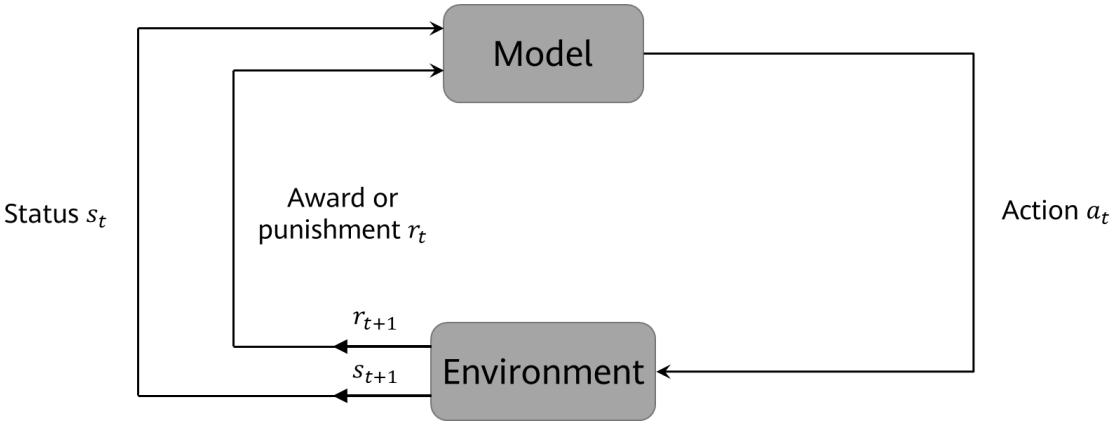


Figure 1-10 Reinforcement learning algorithm

As shown in 0, the two most important parts of a reinforcement learning algorithm are the model and environment. In different environments, the model can determine its own actions, and different actions may have different impacts on the environments. For example, a computer can give the answer of a question at will, and the teacher will rate the answer. If that is the only case, the computer will not have learned how to do questions, because the teacher's grades do not apply to the training process. In this case, the importance of status as well as reward and punishment is reflected. A higher test score can make the teacher satisfied, thus giving the computer a certain reward. Conversely, a lower test score could be a penalty for the computer. As a progressive computer, it is certain to adjust its own model parameters so that the answers given by itself will receive more rewards. In this process, no one provides training data for machine learning algorithms, or tells the reinforcement learning system how to generate the correct action. All data and reward signals are dynamically generated and learned during the interaction between the model and the environment. Both good and bad behaviors can help model learning.

1.3 Machine Learning Process

1.3.1 Overview

A complete machine learning project includes data collection, data cleansing, feature extraction and selection, model training, model evaluation, as well as model deployment and integration, as shown in 0. This section first describes concepts related to data and data cleansing. These concepts are the basis for understanding feature selection. After selecting proper features, you need to train and evaluate a model based on these features. The entire process can be completed only after continuous feedback and iterations to achieve satisfactory results. Finally, you need to further deploy the model in specific application scenarios for practice.

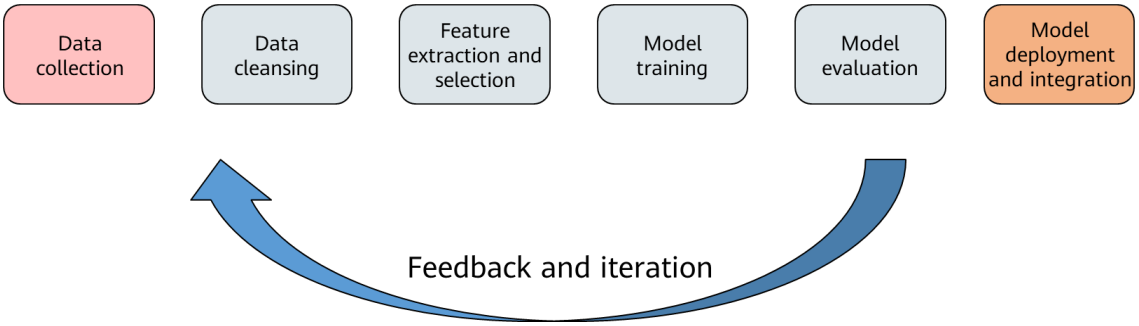


Figure 1-11 Machine learning process

1.3.2 Data Collection

A dataset is a collection of data used in machine learning tasks. Each piece of data is called a sample. Events or attributes that reflect the performance or nature of a sample in a particular aspect are called features. A training set is a dataset used in the training process, where each sample is referred to as a training sample. Learning (training) is the process of creating a model from data. The process of using a model for prediction is called testing, and the dataset used is called a test set. Each sample in the test set is called a test sample.

		Feature 1	Feature 2	Feature 3	Label
	No.	Area	School Districts	Direction	House Price
Training set	1	100	8	South	1000
	2	120	9	Southwest	1300
	3	60	6	North	700
	4	80	9	Southeast	1100
Test set	5	95	3	South	850

Figure 1-12 Dataset example

0 shows a typical dataset. In this dataset, each row represents one sample, and each column represents one feature or label. After a task (for example, predicting house prices based on the areas, school districts, and directions) is determined, the features and labels are determined. Therefore, the table header of a dataset cannot be changed throughout a machine learning project. The splitting of the training set and test set is relatively flexible. Researchers can determine which samples belong to the training set based on experience. If the proportion of the test set is too low, model testing will be excessively random. As a result, the performance of the model cannot be properly evaluated. If the proportion of the training set is too low, the sample utilization may be low and model learning is insufficient. Therefore, it is recommended that the training set account for

80% of the total number of samples, and the test set account for 20%. In this example, there are four samples in the training set and one sample in the test set.

1.3.3 Data Cleansing

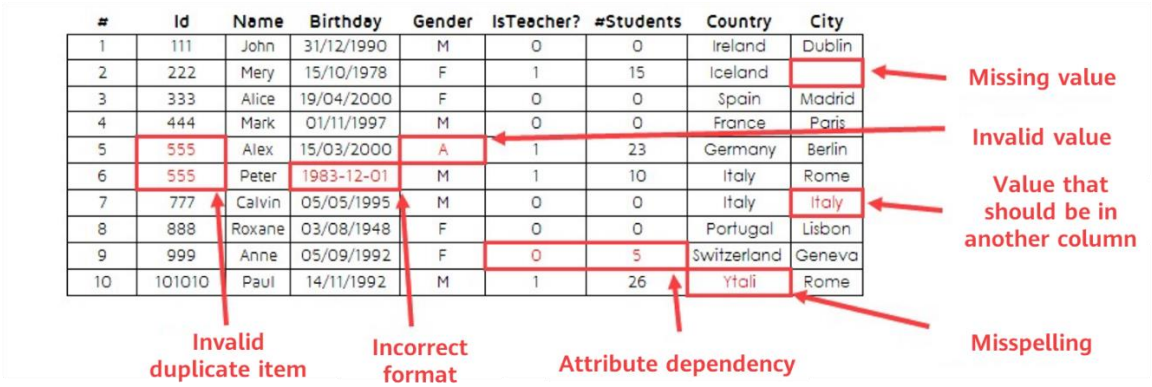
Data is crucial to models. It is the ceiling of model capabilities. Without good data, there is no good model. However, real data may have some quality problems, as shown in 0. Typical data quality problems include:

(1) Incompleteness: contains missing values or the data that lacks attributes.

(2) Noise: contains incorrect records or exceptions.

(3) Inconsistency: contains inconsistent records.

Such data is called dirty data. The process of filling in missing values, as well as detecting and eliminating exceptions is called data cleansing. In addition, data preprocessing usually includes data dimension reduction and data normalization. The purpose of data dimension reduction is to simplify data attributes to prevent dimension explosion. The purpose of data normalization is to normalize the dimensions of each feature to reduce the training difficulty. This section describes only data cleansing.



#	Id	Name	Birthday	Gender	IsTeacher?	#Students	Country	City
1	111	John	31/12/1990	M	0	0	Ireland	Dublin
2	222	Mery	15/10/1978	F	1	15	Iceland	
3	333	Alice	19/04/2000	F	0	0	Spain	Madrid
4	444	Mark	01/11/1997	M	0	0	France	Paris
5	555	Alex	15/03/2000	A	1	23	Germany	Berlin
6	555	Peter	1983-12-01	M	1	10	Italy	Rome
7	777	Calvin	05/05/1995	M	0	0	Italy	Italy
8	888	Roxane	03/08/1948	F	0	0	Portugal	Lisbon
9	999	Anne	05/09/1992	F	0	5	Switzerland	Geneva
10	101010	Paul	14/11/1992	M	1	26	Ytali	Rome

Figure 1-13 Dirty data

Most machine learning models process features, which are usually numeric representations of input variables that can be used in the model. In most cases, collected data can be used by algorithms only after being preprocessed. The preprocessing operations include:

- Data filtering
- Processing of lost data
- Processing of possible exceptions, errors, or abnormal values
- Combination of data from multiple data sources
- Data consolidation

The workload of data cleansing is usually heavy. Research shows that cleansing and organizing data account for 60% of data scientists' time in machine learning research, as shown in 0. On the one hand, this shows the difficulty of data cleansing: Different ways and contents of data collection require different methods of data cleansing. On the other hand, this also shows that data cleansing plays an important role in subsequent model

training and optimization: If data is thoroughly cleaned, the model is less susceptible to interference from abnormal data, ensuring the model training effect.

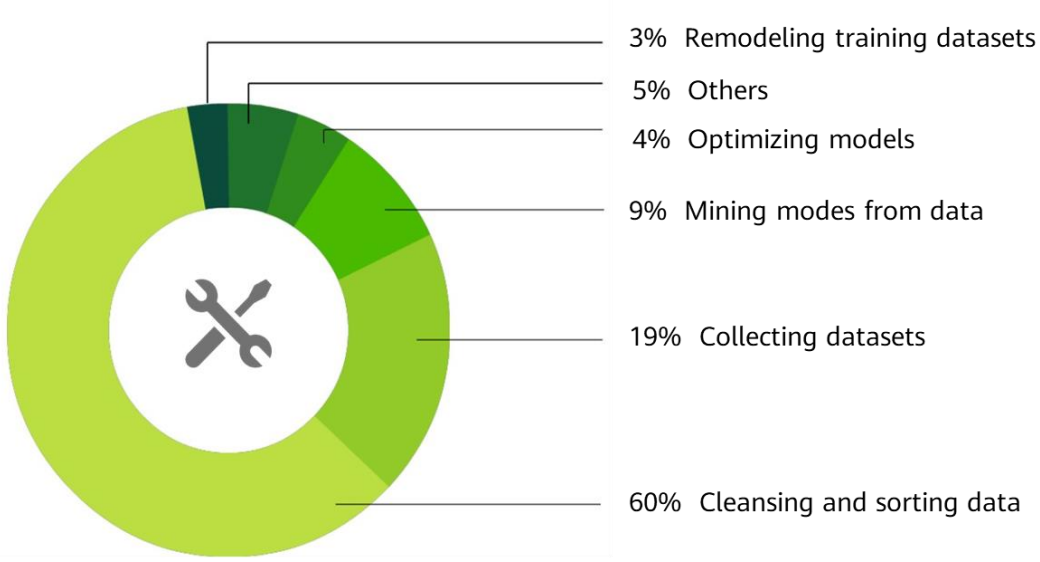


Figure 1-14 Statistics on data scientists' work in machine learning

1.3.4 Feature Selection

Generally, a dataset has many features, some of which may be redundant or irrelevant to the target. For example, in the task of predicting house prices based on the area, school district, and temperature of the day, the temperature of the day is obviously an irrelevant feature. Feature selection filters out redundant or irrelevant features, simplifying models and making them easier for users to interpret. In addition, feature selection can effectively reduce the model training time, prevent dimension explosion, improve model generalization performance, and prevent overfitting. The common methods for feature selection include filter, wrapper, and embedded.

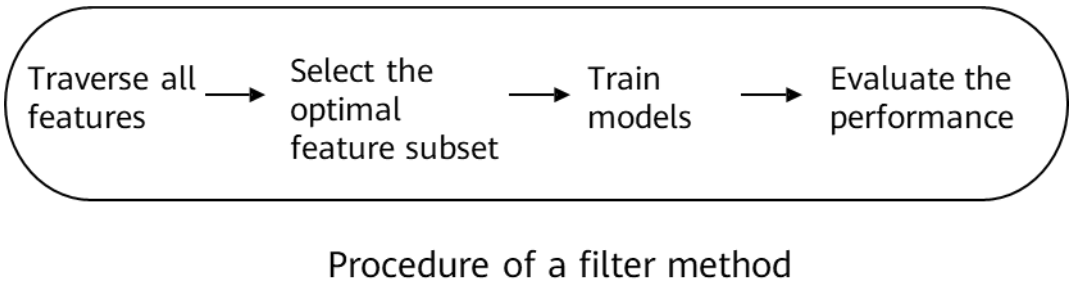


Figure 1-15 Filter method

Filter methods are independent of models during feature selection. By evaluating the correlation between each feature and the target attribute, these methods use a statistical measure to score each feature. Features are then sorted by score, which is helpful for preserving or eliminating specific features. 0 shows the machine learning process using a

filter method. Statistical measures commonly used in filter methods include Pearson correlation coefficient, chi-square coefficient, and mutual information. Because filter methods do not consider the relationship between features, they only tend to filter out redundant variables.

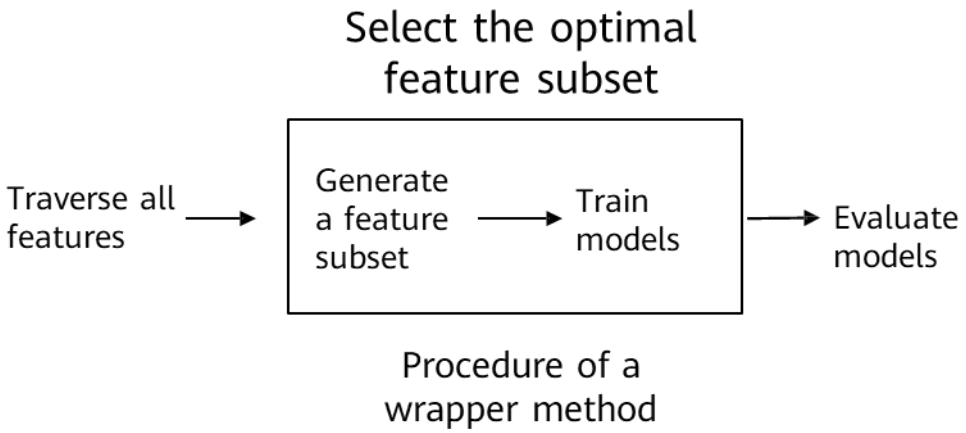


Figure 1-16 Wrapper method

Wrapper methods use a prediction model to score feature subsets. Consider feature selection as a search issue, in which wrappers use a prediction model to evaluate and compare different feature combinations. Higher accuracy of the prediction model indicates that the feature combination should be retained. 0 shows the machine learning process using a wrapper method. Common wrapper methods include the recursive feature elimination (RFE). Wrapper methods usually provide feature sets with the best performance for a specific type of models, but need to train a new model for each feature subset, which can be computationally expensive.

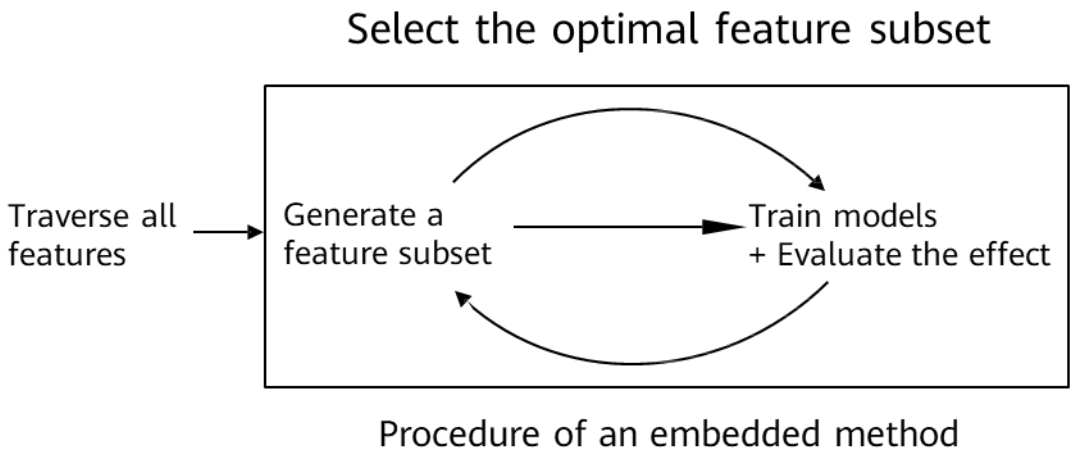


Figure 1-17 Embedded method

Embedded methods consider feature selection as a part of model building, as shown in 0. Unlike filter and wrapper methods, models using embedding methods dynamically learn how to select features during training. The most common type of embedded feature

selection methods is regularization methods. Regularization methods are also called penalization methods that introduce additional constraints into the optimization of a predictive algorithm to bias the model toward lower complexity and reduce the number of features. Common regularization methods include Ridge regression and Lasso regression.

1.3.5 Overall Procedure of Building a Model

After data cleansing and feature extraction, you need to start building the model. 0 shows how to build a model, using supervised learning as an example. The core of model building is model training, verification, and testing. This section briefly describes the training and prediction process based on an example.

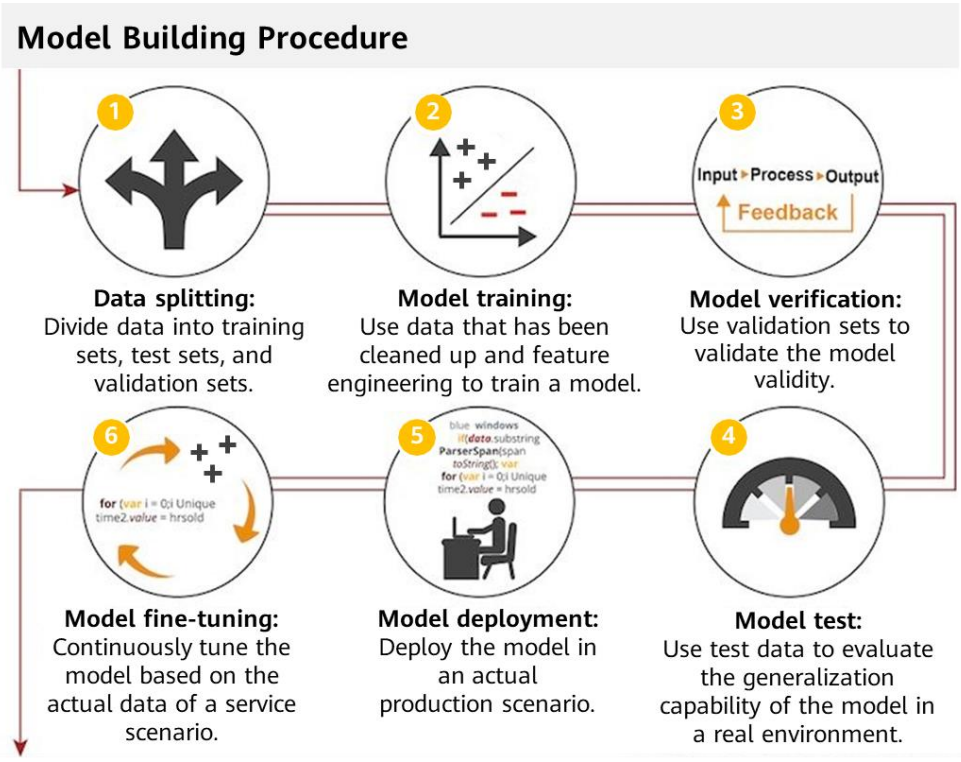


Figure 1-18 Model building procedure

In the example in this section, we use a classification model to predict whether a person needs to change the supplier for a particular feature. Assume that the dataset in 0 is the cleaned dataset. The task of the model is to predict the target as accurately as possible based on the known features. During the training, the model can learn the mapping relationship between features and the target based on samples in the training set. After training, we may obtain the following model:

Code:

```
def model(city, age):
    if city == "Miami": return 0.7
    if city == "Orlando": return 0.2
    if age > 42: return 0.05 * age + 0.06
    else: return 0.01 * age + 0.02
```

The output of the model is the probability that the target is the true value. As we know, the model accuracy increases as the training data increases. So why not use all the data for training, but use part of it as test set? This is because we are concerned about the performance of the model in the face of unknown data, not known data. It can be understood that the training set is like the question bank that students studied when preparing for an exam. No matter how high the accuracy rate of students in the question bank is not surprising, because the question bank is always limited. As long as the students' memory is good enough, all the answers can be memorized. Only through an examination can we really check the students' mastery of knowledge, because the questions appear in the examination are never seen by the students. The test set is equivalent to a test paper prepared by the researcher for the model. That is, in the entire dataset (including the training set and test set), the model can read only the features of the training set and test set. The targets of the test set can only be used by the researcher to evaluate the performance of the model.

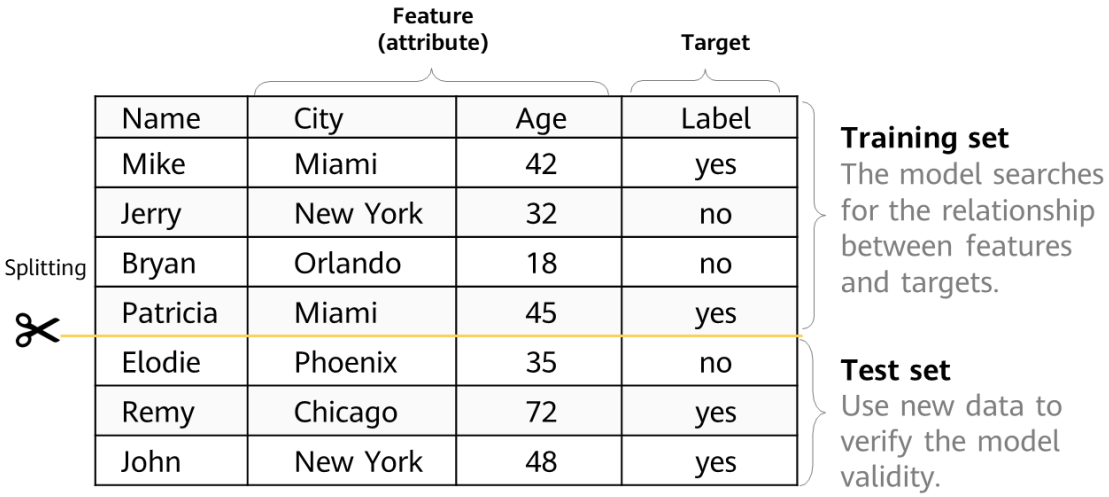


Figure 1-19 Training set and test set

1.3.6 Model Evaluation

What Is a Good Model? The most important evaluation indicator is the generalization capability of a model, that is, the prediction accuracy of a model regarding actual service data. In addition, some engineering indicators can also be used to evaluate a model. Interpretability describes the intuitive degree of the prediction result of a model. The prediction rate refers to the average time for a model to predict each sample. Plasticity refers to the degree to which the prediction rate can be accepted with the increase of the service volume in the actual service process.

The goal of machine learning is that the model obtained after learning should perform well on new samples, not just on samples used for training. The capability of applying a model to new samples is called generalization or robustness. The difference between the sample result predicted by the model obtained after learning and the actual sample result is called an error. The training error refers to the error that you get when you run the model on the training set, and the generalization error refers to the error that you

get when you run the model on new samples (test set). Obviously, we prefer a model with a smaller generalization error.

Once the form of a model is given, all possible functions constitute a space, which is hypothesis space. Machine learning algorithms are searching for a suitable fitting function in a hypothesis space. If the mathematical model is too simple or the training time is too short, the training error of the model will be large. This phenomenon is called underfitting. For the former cause, a more complex model needs to be used for retraining. For the latter cause, the underfitting phenomenon can be effectively alleviated only by prolonging the training time. However, to accurately determine the causes of underfitting often requires certain experience and methods. On the contrast, overfitting refers to the phenomenon that the training error of a model is very small (because the model is complex) but the generalization capability is weak, that is, the generalization error is relatively large. There are many ways to mitigate overfitting. The common ones are as follows: appropriately simplifying the model; ending training before overfitting occurs; using the Dropout and Weight Decay methods. 0 shows the underfitting, good fitting, and overfitting results for the same dataset.

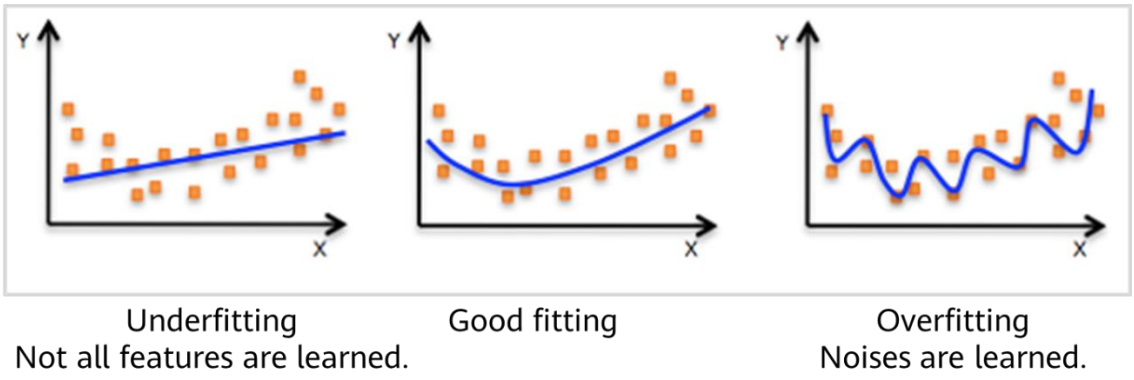


Figure 1-20 Underfitting, good fitting, and overfitting

Model capacity, also called model complexity, refers to model's capability of fitting functions. When the capacity suits the task complexity and the amount of training data provided, the algorithm effect is usually optimal. Models with insufficient capacity cannot solve complex tasks, and underfitting may occur. As shown in the left part in 0, the data distribution is in a tick shape, but the model is linear and cannot describe the data distribution well. A high-capacity model can solve complex tasks, but overfitting may occur if the capacity is higher than that required by a task. As shown in the right part in 0, the model attempts to fit data with an extremely complex function. Although the training error is reduced, it is conceivable that such a model cannot well predict the target value of a new sample. The effective capacity of a model is limited by algorithms, parameters, and regularization methods.

In general, the generalization error can be broken down into the following forms:

$$\text{Total error} = \text{Bias}^2 + \text{Variance} + \text{Unresolvable error}$$

Bias and variance are two subforms that we should pay attention to. As shown in 0, variance is the offset of the prediction result from the average value, and is the error

caused by the model's sensitivity to small fluctuations in the training set. Bias is the difference between the average prediction value and the correct value we are trying to predict. Unresolvable errors refer to errors caused by imperfections of models and finiteness of data. Theoretically, if there is infinite amount of data and a perfect model, the error can be eliminated. However, there is no such situation in practice, so the generalization error can never be eliminated.

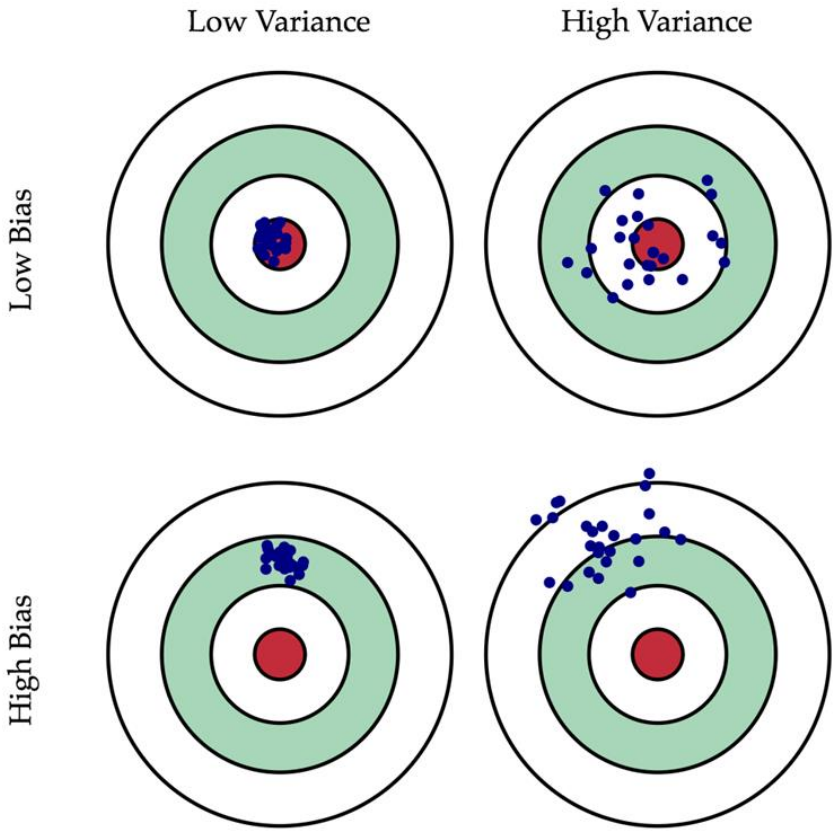


Figure 1-21 Variance and bias

Ideally, we want a model that can accurately capture the rules in the training data and summarize the invisible data (new data). However, it is usually impossible for the model to complete both tasks at the same time. As shown in 0, the training error decreases as the model complexity increases. As the model complexity increases, the test error decreases to a certain point and then increases in the reverse direction, forming a convex curve. The model complexity at the lowest test error point is the ideal model complexity.

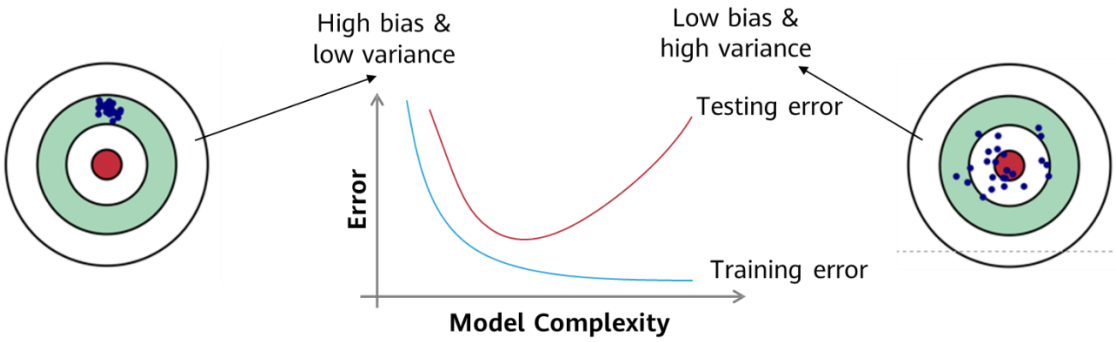


Figure 1-22 Relationship between model complexity and error

The commonly used counters for evaluating the performance of a regression model are mean absolute error (MAE), mean square error (MSE), and R^2. Assume that the actual target value of the test sample is $y_1, y_2, \ldots, y_m$ and the corresponding predicted value is $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m$. The preceding counters are defined as follows:

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i|$$

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{m}(y_i - \bar{y}_i)^2}$$

TSS indicates the difference between samples, and RSS indicates the difference between the predicted value and sample value. The values of MAE and MSE counters are non-negative. A value closer to 0 indicates better model performance. The value of R^2 is not greater than 1. A value closer to 1 indicates better model performance.

Estimated amount / Actual amount	yes	no	Total
yes	TP	FN	P
no	FP	TN	N
Total	P'	N'	$P+N$

Figure 1-23 Binary-classification confusion matrix

The confusion matrix is used to evaluate the performance of a classification model, as shown in 0. The confusion matrix is a *k*-dimensional square matrix, where *k* represents the quantity of all categories. The value in row *i* and column *j* in the table indicates the number of samples that belong to category *i* but are determined as category *j* by the model. Ideally, for a high accuracy classifier, most samples should be located in the

diagonal of the table while values outside the diagonal are 0 or close to 0. Each symbol in the binary-classification confusion matrix shown in 0 is described as follows:

(1) *P*: positive, indicating the number of real positive cases in the data.

(2) *N*: negative, indicating the number of real negative cases other than *P* in the data.

(3) *TP*: true positive, indicating the number of positive cases that are correctly classified by the classifier.

(4) *TN*: true negative, indicating the number of negative cases that are correctly classified by the classifier.

(5) *FP*: false positive, indicating the number of positive cases that are incorrectly classified by the classifier.

(6) *FN*: false negative, indicating the number of negative cases that are incorrectly classified by the classifier.

0 lists other concepts in the binary-classification confusion matrix.

Measurement	Ratio
Accuracy and recognition rate	$\frac{TP+TN}{P+N}$
Error rate and misclassification rate	$\frac{FP+FN}{P+N}$
Sensitivity, true positive rate, and recall	$\frac{TP}{P}$
Specificity and true negative rate	$\frac{TN}{N}$
Precision	$\frac{TP}{TP+FP}$
F_1, harmonic mean of the recall rate and precision	$\frac{2 \times precision \times recall}{precision + recall}$
F_β, where β is a non-negative real number	$\frac{(1+\beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$

Figure 1-24 Other concepts of a confusion matrix

The following describes the concepts of precision rate and recall rate by taking literature retrieval as an example. The precision rate describes the proportion of documents that are really related to the search subject in all the documents that are retrieved. The recall rate describes the proportion of the retrieved documents related to the retrieval subject to all the related documents in the document library.

At the end of this section, we use an example to describe the calculation of the binary-classification confusion matrix. Assume that a classifier that can identify whether the object in an image is a cat, and now we use 200 images to verify the performance measures of the model. Among the 200 images, objects in 170 images are cats, while others are not. 0 shows the performance of the model. The identification result of the model is that objects in 160 images are cats, while others are not. The precision rate of

the model can be calculated as follows: 140/160 = 87.5%; the recall rate is 140/170 = 82.4%; the accuracy rate is (140 + 10)/200 = 75%.

Estimated amount / Actual amount	*yes*	*no*	Total
yes	140	30	170
no	20	10	30
Total	160	40	200

Figure 1-25 Confusion matrix instance

1.4 Parameters and Hyperparameters in Models

Parameters are a part of a model that is learned from historical training data and key to machine learning algorithms. Generally, model parameters are not manually set by researchers, but obtained from data estimation or learning. Determining parameter values of a model is equivalent to defining the functions of the model. Therefore, model parameters are usually saved as a part of the learning model. Parameters are also an integral part of model prediction. Some examples of model parameters include weights in artificial neural networks, support vectors in support vector machines (SVMs), and coefficients in linear or logistic regression.

A model contains not only parameters but also hyperparameters. Different from a parameter, a hyperparameter is an external configuration of a model, and is usually used in the process of estimating a model parameter. The most fundamental difference between them is that parameters are automatically learned by a model, while hyperparameters are manually set. Model hyperparameters usually need to be adjusted for different prediction modeling problems. In addition to being directly specified by researchers, model hyperparameters can also be set using a heuristic method. Common model hyperparameters include the penalty item coefficient in Lasso/Ridge regression, the learning rate, quantity of iterations, batch size, activation function, and quantity of neurons in a training neural network, as well as C and σ of SVMs, K in KNN, the quantity of decision tree models in a random forest.

Model training generally refers to optimizing model parameters, and this process is completed by using the gradient descent algorithm. Based on the training effect of the model, you can use a series of hyperparameter search algorithms to optimize the hyperparameters of the model. This section successively describes gradient descent

algorithms, the concept of validation set, and hyperparameter search and cross validation.

1.4.1 Gradient Descent

A gradient descent algorithm uses the negative gradient direction of the current position as the search direction, which is the steepest direction. See the left part in 0. The gradient descent formula is as follows:

$$w_{k+1} = w_k - \eta \nabla f_{w_k}(x)$$

In the formula, η indicates the learning rate, and w represents a parameter of the model. As w approaches the target value, a variation amount of w gradually decreases. When the value of the target function changes little or the maximum number of iterations of gradient descent is reached, algorithm convergence occurs. It should be noted that when the gradient descent algorithm is used to calculate the minimum value of the non-convex function, different initial values may lead to different results, as shown in the right part in 0.

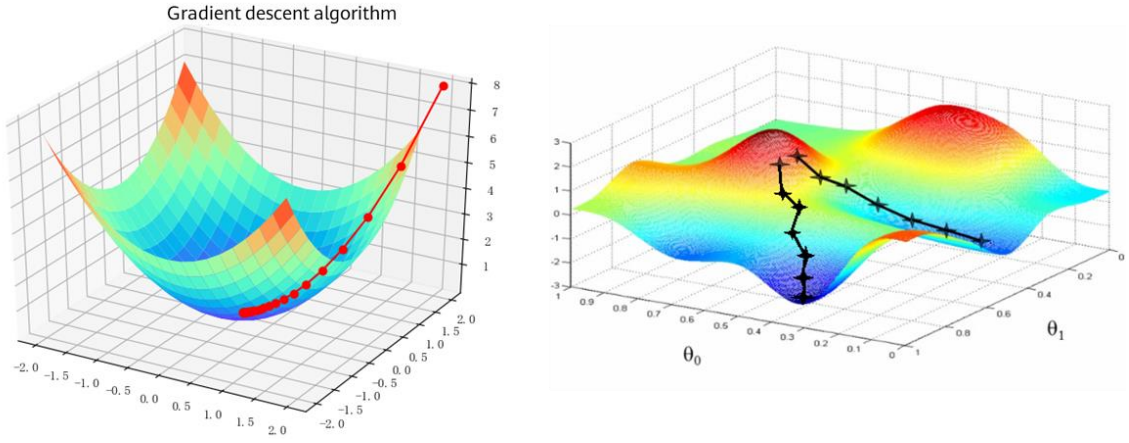
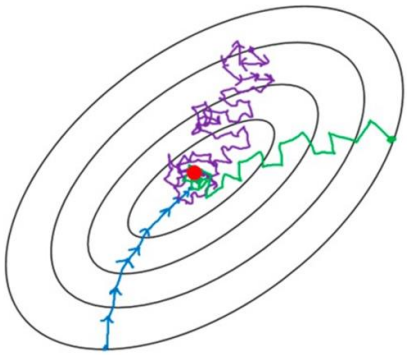


Figure 1-26 Gradient descent algorithm

You can apply one of the following gradient descent algorithms to model training:

- Batch gradient descent (BGD): uses the samples in all datasets to update the weight parameter based on the average gradient value at the current point.
- Stochastic gradient descent (SGD): randomly selects a sample from a dataset to update the weight parameter based on the gradient value of the sample.
- Mini-batch gradient descent (MBGD): combines the features of BGD and SGD and selects the average gradient value of n samples from a dataset to update the weight parameter.

0 shows the comparison of the three gradient descent algorithms. BGD is the most stable during running. However, because this algorithm traverses all samples for each update, it consumes a large quantity of computing resources. In SGD, randomly selecting samples for each update improves the calculation efficiency but also brings instability. The loss function may be unstable or even is reversely displaced when it decreases to the lowest point. MBGD is a balance between SGD and BGD, and is also a most commonly used gradient descent algorithm in machine learning at present.



BGD
Uses **all** training samples for training each time.

SGD
Uses **one** training sample for training each time.

MBGD
Uses a certain number of training samples for training each time.

Figure 1-27 Efficiency comparison of gradient descent algorithms

1.4.2 Validation Set and Hyperparameter Search

A training set is a set of samples used in model training. In the training process, gradient descent algorithms improve the prediction accuracy of the model on the samples in the training set as much as possible. As a result, the performance of the model on the training set is better than that on unknown datasets. To measure the generalization capability of a model, a part of the entire dataset is randomly selected as the test set before training, as shown in 0. Samples in the test set do not participate in training, and therefore are unknown to the model. It can be approximately considered that the performance of the model on the test set is the performance of the model on unknown samples.

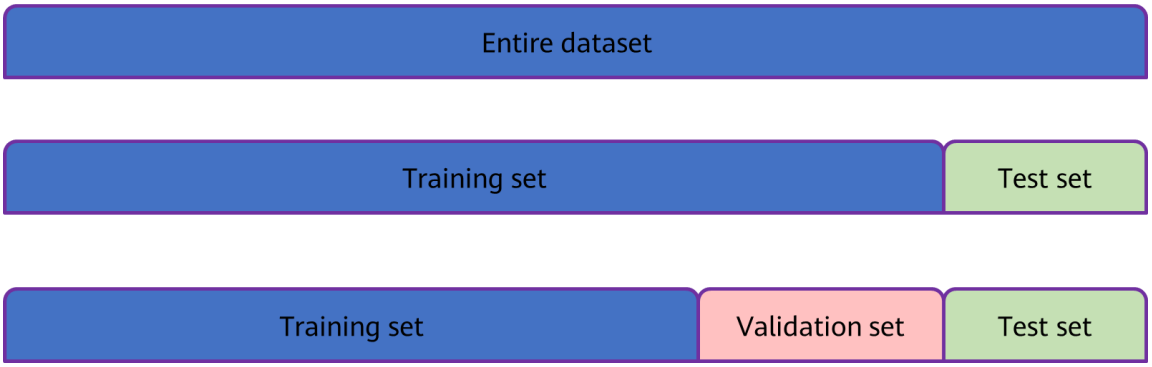


Figure 1-28 Training set, validation set, and test set

The goal of hyperparameter optimization is to improve the generalization capability of models. The most intuitive idea is to try different hyperparameter values, evaluate the performance of these models on the test set, and select the model with the strongest generalization capability. The problem with this idea is that the test set cannot participate in model training in any form, even in hyperparameter searches. Therefore, we need to randomly select some samples from the training set to form a validation set. Samples in the validation set do not participate in training either, and are used only to verify the effect of hyperparameters. Generally, a model needs to be repeatedly optimized on the training set and validation set to finally determine the parameters and

hyperparameters, and to evaluate the performance of the model on the test set. Common methods used to search for model hyperparameters include grid search, random search, heuristic intelligent search, and Bayesian search.

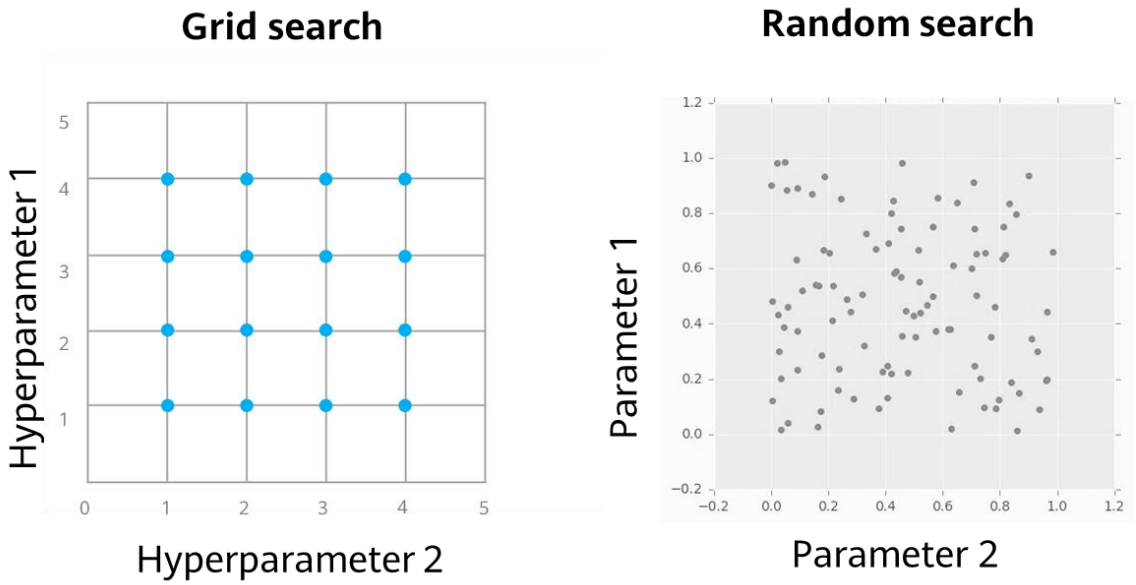


Figure 1-29 Grid search and random search

Grid search attempts to exhaustively search all possible hyperparameter combinations to form a hyperparameter value grid, as shown in the left part in 0. In practice, the scope and step size of the grid often need to be manually specified. Grid search works well when the number of hyperparameters is relatively small, so it is feasible in general machine learning algorithms. However, in a case such as a neural network, grid search is too expensive and time-consuming, and therefore is generally not used.

When the hyperparameter search space is large, the effect of random search is better than that of grid search. See the right part in 0. In random search, each setting is sampled from the distribution of possible parameter values, in an attempt to find the best subset of hyperparameters. Search is performed within a coarse range, which then will be narrowed based on where the best result appears. In practice, some hyperparameters are more important than others. In this case, the important hyperparameters will directly affect the search deviation, and other hyperparameters may not be optimized well.

1.4.3 Cross Validation

There are two main problems in the preceding methods of dividing validation sets: The randomness of sample division is very large, and the validation results are not persuasive; the number of samples that can be used for model training is further reduced. To solve the problems, you can divide the training set into *k* groups for *k*-fold cross validation. During *k*-fold cross validation, *k* rounds of training and validation are performed: A group of data is used as the validation set in turn, and the remaining *k*-1 groups of data are used as the training set. In this way, you will obtain *k* models and their classification

accuracies on validation sets. The average value of the *k* classification accuracies can be used as the performance indicator of the model generalization capability.

The *k*-fold cross validation can prevent the randomness of validation set division, and the validation result is more persuasive. However, *k*-fold cross validation requires the training of *k* models. If the dataset is large, the training time is long. Therefore, *k*-fold cross validation is generally applicable to small datasets.

The value of *k* in *k*-fold cross validation is also a hyperparameter, which needs to be determined through experiments. In an extreme case, the value of *k* is the same as the number of samples in the training set. This practice is called leave-one-out cross validation, in which a training sample is left as the validation set during each training. The training effect of leave-one-out cross validation is better, because almost all training samples participate in training. However, leave-one-out cross validation will last for a longer time, so it only applies to very small datasets.

1.5 Common Machine Learning Algorithms

1.5.1 Overview

As shown in Figure 1-30, there are many common algorithms for machine learning. This section briefly describes the principles and basic ideas of these algorithms. For details, refer to related books.

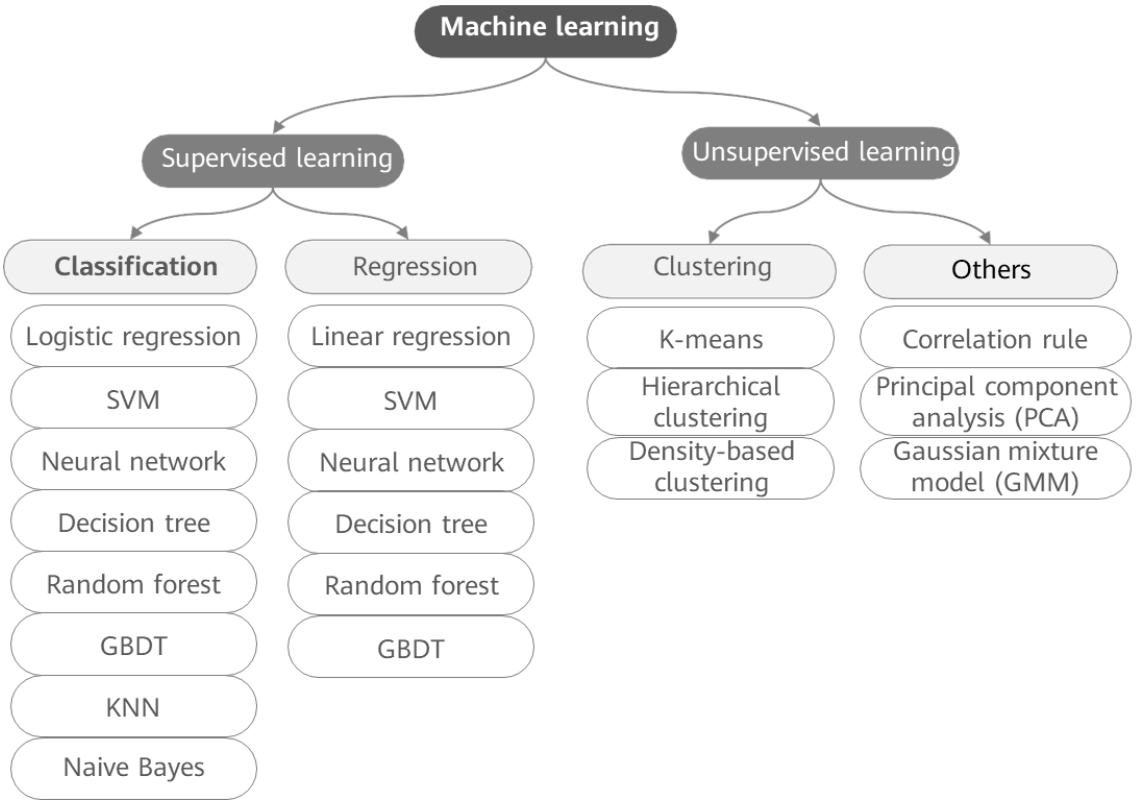


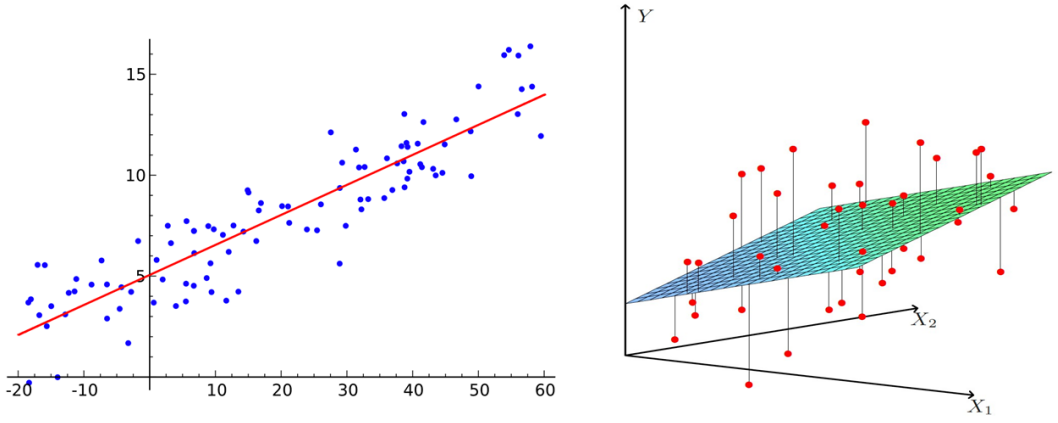
Figure 1-30 Common machine learning algorithms

1.5.2 Linear Regression

Linear regression, a type of supervised learning, is a statistical analysis method for determining the quantitative relationship between two or more variables through regression analysis in mathematical statistics. As shown in 0, the model function of linear regression is a hyperplane.

$$h(x) = w^T x + b$$

In the formula, *w* is a weight parameter, *b* is an offset, and *x* is a sample.



Unary linear regression　　Multi-dimensional linear regression

Figure 1-31 Linear regression

The relationship between the value predicted by the model and the actual value is as follows:

$$y = h(x) + \varepsilon$$

In the formula, *y* indicates the actual value, and ε indicates an error. The error is affected by many factors. According to the central limit theorem, the error follows normal distribution.

$$\varepsilon \sim N(0, \sigma^2)$$

The probability distribution of actual values can be obtained.

$$y \sim N(h(x), \sigma^2)$$

According to the maximum likelihood estimation, the target of model optimization is as follows:

$$\underset{h}{argmax} \prod_{i=1}^{m} P(Y = y_i | X = x_i) = \underset{h}{argmax} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(h(x_i) - y_i)^2}{2\sigma^2}\right)$$

In the formula, *argmax* indicates that a maximum value point is to be obtained, that is, *h*, which maximizes the value of the target function. In the target function, $\left(\sqrt{2\pi}\sigma\right)^{-1}$ is a constant irrelevant to *h*. Multiplying or dividing the target function by a constant does not change the position of the maximum or minimum value point. Therefore, the optimization target of the model can be expressed as follows:

$$\underset{h}{argmax}\prod_{i=1}^{m} exp\left(-\frac{(h(x_i)-y_i)^2}{2\sigma^2}\right)$$

Because the logarithmic function is monotonic, setting the target function to *ln* does not affect the maximum and minimum value points.

$$\underset{h}{argmax}\, ln\left(\prod_{i=1}^{m} exp\left(-\frac{(h(x_i)-y_i)^2}{2\sigma^2}\right)\right) = \underset{h}{argmax}\sum_{i=1}^{m} -\frac{(h(x_i)-y_i)^2}{2\sigma^2}$$

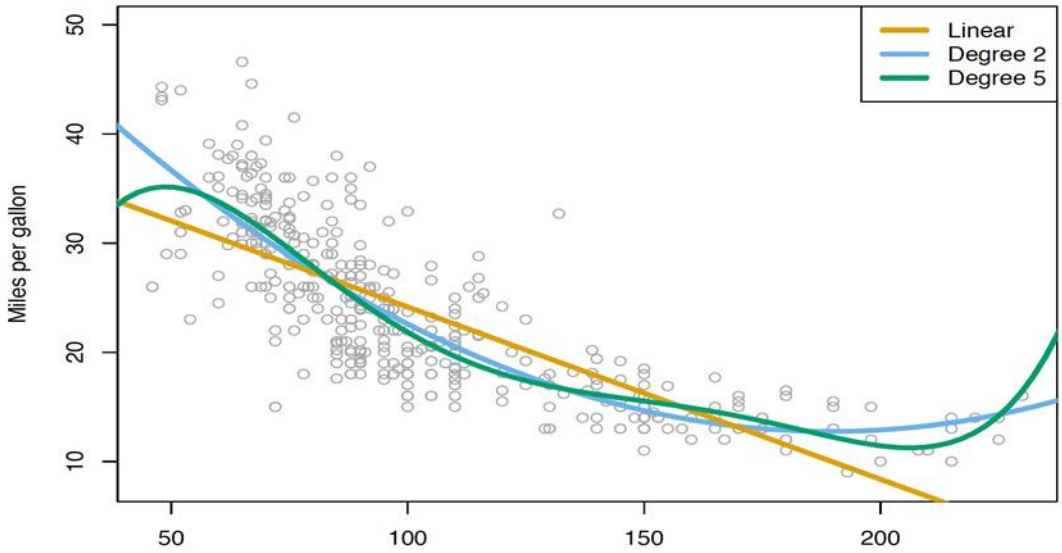
If the target function is set to a negative value, the original maximum value point is changed to the minimum value point. In addition, we can multiply the target function by a constant σ^2/m to convert the optimization target of the model into:

$$\underset{h}{argmin}\frac{1}{2m}\sum_{i=1}^{m}(h(x_i)-y_i)^2$$

Obviously, the loss function is:

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(h(x_i)-y_i)^2$$

We want the predicted value approaches the actual value as far as possible, that is, to minimize the loss value. We can use a gradient descent algorithm to calculate the weight parameter *w* when the loss function reaches the minimum, thereby complete model building.



Comparison between linear regression and polynomial regression

Figure 1-32 Comparison between linear regression and polynomial regression

Polynomial regression is an extension of linear regression. Generally, the complexity of a dataset exceeds the possibility of fitting performed by using a straight line. That is,

obvious underfitting occurs if the original linear regression model is used. The solution is to use polynomial regression, as shown in Figure 1-32.

$$h(x) = w_1x + w_2x^2 + \cdots + w_nx^n + b$$

In the formula, n indicates the number of polynomial regression dimensions.

Because the polynomial regression dimension is a hyperparameter, overfitting may occur if the dimension is selected unexpectedly. Applying regularization helps reduce overfitting. The most common regularization method is to add a square sum loss to the target function.

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(h(x_i) - y_i)^2 + \lambda\|w\|_2{}^2$$

In the formula, $\|\cdot\|_2$ indicates an L2 regular term. The linear regression model using this loss function is also known as a Ridge regression model. Similarly, a linear regression model with absolute loss is called a Lasso regression model.

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(h(x_i) - y_i)^2 + \lambda\sum\|w\|_1$$

In the formula, $\|\cdot\|_1$ indicates an L1 regular term.

1.5.3 Logistic Regression

The logistic regression model is used to solve classification problems. The model is defined as follows:

$$h(x) = P(Y = 1|X) = g(w^Tx + b)$$

In the formula, g represents a sigmoid function, w represents a weight, and b represents the bias. $w^Tx + b$ is a linear function of x. Therefore, same as linear regression, logistic regression is also a generalized linear model.

The sigmoid function is defined as follows:

$$g(x) = \frac{1}{1 + exp\{-x\}}$$

0 shows a sigmoid function.

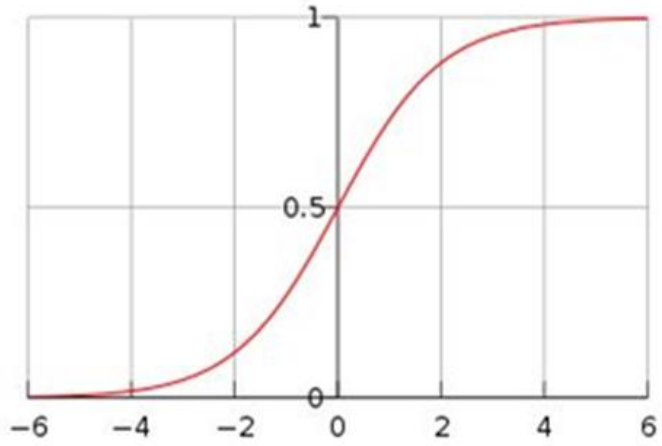


Figure 1-33 Sigmoid function

You can obtain a classification result corresponding to *x* by comparing the value relationship between *P(Y=1|X)* and the threshold *t*. The threshold *t* is a hyperparameter of the model, and can be any value. It is not difficult to see that when the threshold is large, the model tends to determine the sample as a negative example, so the precision rate is higher. When the threshold is small, the model tends to determine the sample as a positive example, so the recall rate is higher. Generally, you can use 0.5 as the threshold.

According to the maximum likelihood estimation, when the sample is a positive one, we hope that *P(Y=1|X)* is larger. When the sample is a negative one, we hope that *P(Y=0|X)* is larger. That is to say, for any sample, we want the value of the following formula to be as large as possible.

$$P = P(Y = 1|X)^y P(Y = 0|X)^{1-y}$$

Replace *P(Y=1|X)* and *P(Y=0|X)* with *h(x)*, we can obtain:

$$P = h(x)^y \cdot (1 - h(x))^{1-y}$$

Therefore, the target of model optimization is as follows:

$$\underset{h}{argmax} \prod_{i=1}^{m} P_i = \underset{h}{argmax} \prod_{i=1}^{m} h(x)^y (1 - h(x))^{1-y}$$

Similar to the derivation process of linear regression, the logarithm of the target function can be taken without changing the position of the maximum value point. Therefore, the optimization target of the model is equivalent to:

$$\underset{h}{argmax} \sum_{i=1}^{m} (y \ln h(x) + (1 - y) ln(1 - h(x)))$$

Multiplying the target function by the constant -1/*m* will cause the original maximum value point to become the minimum value point, that is:

$$\underset{h}{argmin} \frac{-1}{m} \sum_{i=1}^{m} (y \ln h(x) + (1 - y) ln(1 - h(x)))$$

The loss function of logistic regression can be obtained.

$$J(w) = -\frac{1}{m} \sum (y \ln h(x) + (1 - y) ln(1 - h(x)))$$

In the formula, *w* indicates the weight parameter, *m* indicates the number of samples, *x* indicates the sample, and *y* indicates the actual value. You can also obtain the values of all the weight parameters *w* by using a gradient descent algorithm.

Softmax regression is a generalization of logistic regression that we can use for *k*-category classification. The Softmax function is used to map a *k*-dimensional vector of arbitrary actual values to another *k*-dimensional vector of actual values, to represent probability distribution of a category to which a sample belongs. The Softmax regression probability density function is as follows:

$$P(Y = c|x) = \frac{exp\{w_c^T x + b\}}{\sum_{l=1}^{k} \sum_{l}^{T} exp}$$

As shown in 0, Softmax assigns a probability value to each category in a multicategory problem. The sum of all the probabilities must be 1. Of these categories, the probability of the Apple category is 0.68, so the predicted value of the sample should be Apple.

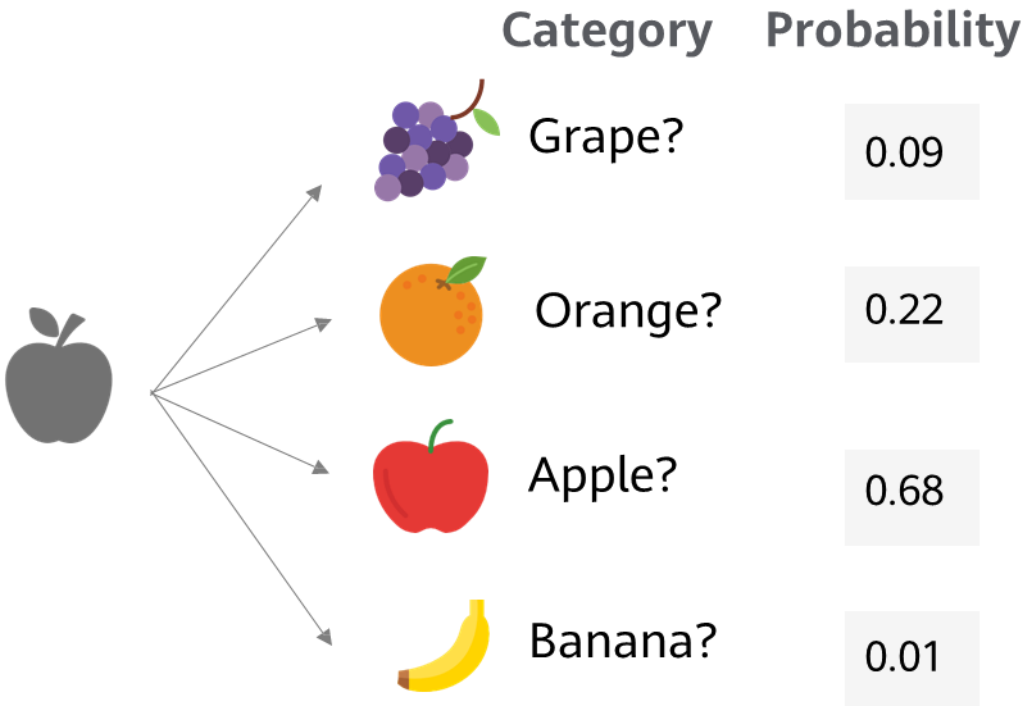


Figure 1-34 Softmax function example

1.5.4 Decision Tree

A decision tree is a binary or non-binary tree classifier, as shown in 0. Each non-leaf node represents a test on a feature attribute. Each branch represents the output of a feature attribute in a certain value range, and each leaf node stores a category. To use the decision tree, start from the root node, test the feature attributes of the items to be classified, select the output branches, and use the category stored on the leaf node as the final result.

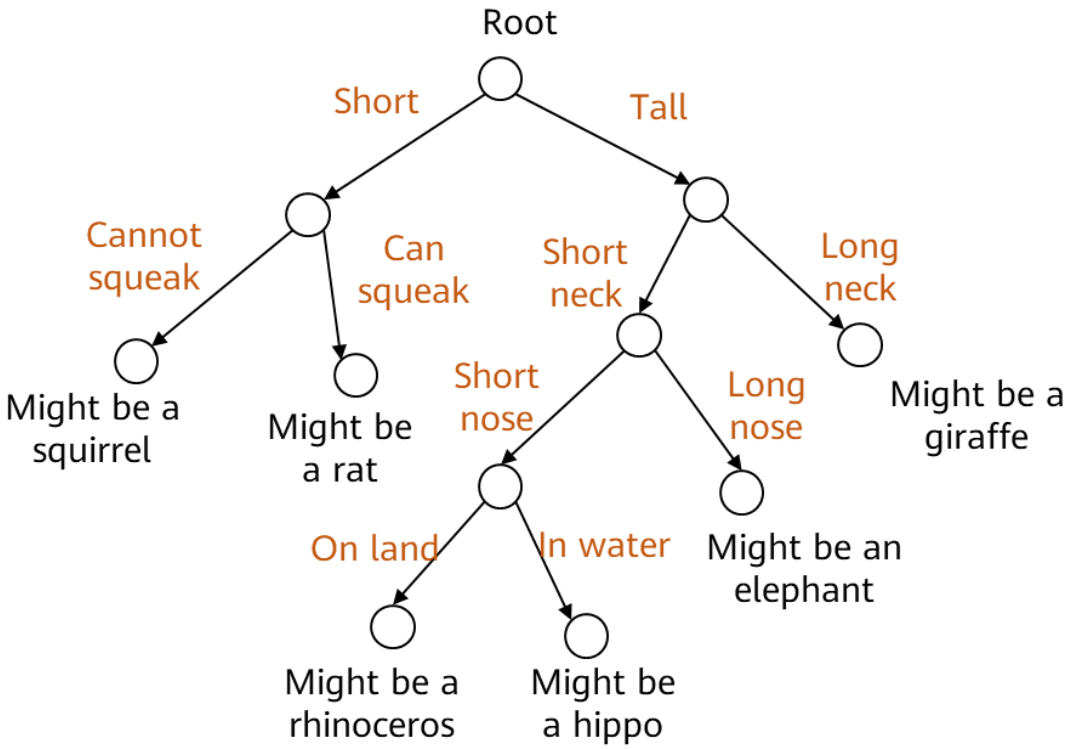


Figure 1-35 Decision tree example

Tree construction is the most important part of the decision tree model. To construct a decision tree, we need to select attributes and determine the topology structure between feature attributes. The key step of constructing a decision tree is to divide data of all feature attributes, compare the result sets in terms of purity, and select the attribute with the highest purity as the data point for dataset division. The learning algorithm of a decision tree is a decision tree construction algorithm. The common algorithms include ID3, C4.5, and CART. The differences between these algorithms lie in quantitative indicators of purity, such as information entropy and Gini coefficient.

$$H(X) = -\sum_{k=1}^{K} p_k \log_2 p_k$$

$$Gini = 1 - \sum_{k=1}^{K} p_k^2$$

In the formula, p_k indicates the probability that the sample belongs to category k, and K indicates the total number of categories. The larger the difference between the purity before and after the segmentation, the better the model accuracy is to be improved by judging a certain feature. Therefore, the feature should be added to the decision tree model.

Generally, the decision tree construction process can be divided into the following three phases:

(1) Feature selection: Select a feature from the features of the training data as the split standard of the current node. (Different standards generate different decision tree algorithms.)

(2) Decision tree generation: Generate subnodes from top down based on the selected feature and stop until the dataset can no longer be split.

(3) Pruning: Reduce the tree size and optimize its node structure to restraint overfitting of the model. Pruning can be classified into pre-pruning and post-pruning.

0 shows an example of classification using a decision tree model. The classification result is affected by the Refund, Marital Status, and Taxable Income attributes. From this example, we can see that the decision tree model can handle not only the case where the attribute has two values, but also the case where the attribute has multiple values or even consecutive values. In addition, a decision tree model is interpretable. We can intuitively analyze the importance relationship between attributes based on the structure diagram on the right in 0.

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125,000	No
2	No	Married	100,000	No
3	No	Single	70,000	No
4	Yes	Married	120,000	No
5	No	Divorced	95,000	Yes
6	No	Married	60,000	No
7	Yes	Divorced	220,000	No
8	No	Single	85,000	Yes
9	No	Married	75,000	No
10	No	Single	90,000	Yes

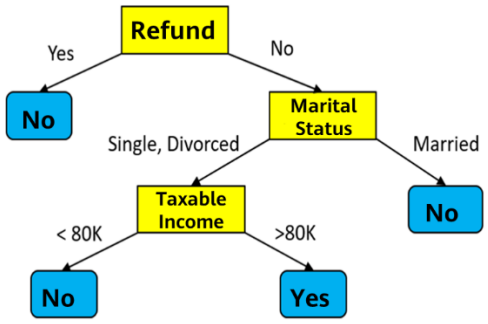


Figure 1-36 Building a decision tree

1.5.5 SVMs

An SVM is a linear classifier defined in the eigenspace with the largest interval. By means of kernel tricks, SVMs can be made into nonlinear classifiers in essence. The SVM learning algorithm is the optimal solution to convex quadratic linear programming. In general, the main ideas of SVM include two points:

(1) Based on the structural risk minimization principle, an optimal hyperplane is constructed in the eigenspace, so that the learner is optimized globally, and the expectation of the whole sample space satisfies an upper boundary with a certain probability.

(2) In the case of linear inseparability, non-linear mapping algorithms are used to convert the linearly inseparable samples of low-dimensional input space into high-dimensional eigenspace. In this way, samples are linearly separable. Then the linear algorithm can be used to analyze the non-linear features of samples.

Straight lines are used to divide data into different categories. Actually, we can use multiple straight lines to divide data, as shown in 0. The core idea of SVM is to find a line that meets the preceding conditions and keep the point closest to the line away from the

line as far as possible. This gives the model a strong generalization capability. These points closest to the straight line are called support vectors.

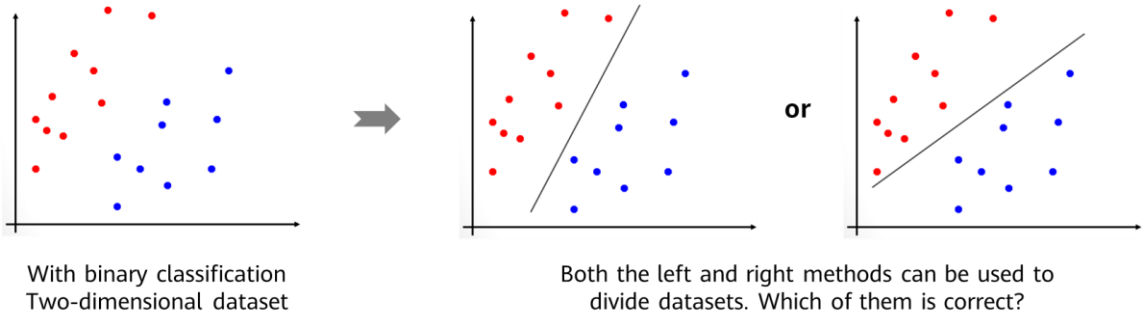


Figure 1-37 Performance of a linear classifier

The use of linear SVM can work well on linear separable datasets, but we cannot segment nonlinear datasets with straight lines. In this case, you need to use kernel functions to construct nonlinear SVMs. Kernel functions allow algorithms to fit the largest hyperplane in a transformed high-dimensional feature space, as shown in 0. Common kernel functions include the linear kernel function, polynomial kernel function, Sigmoid kernel function, and Gaussian kernel function. The Gaussian kernel function can map samples to infinite dimension space, so the effect is also good. It is one of the most commonly used kernel functions.

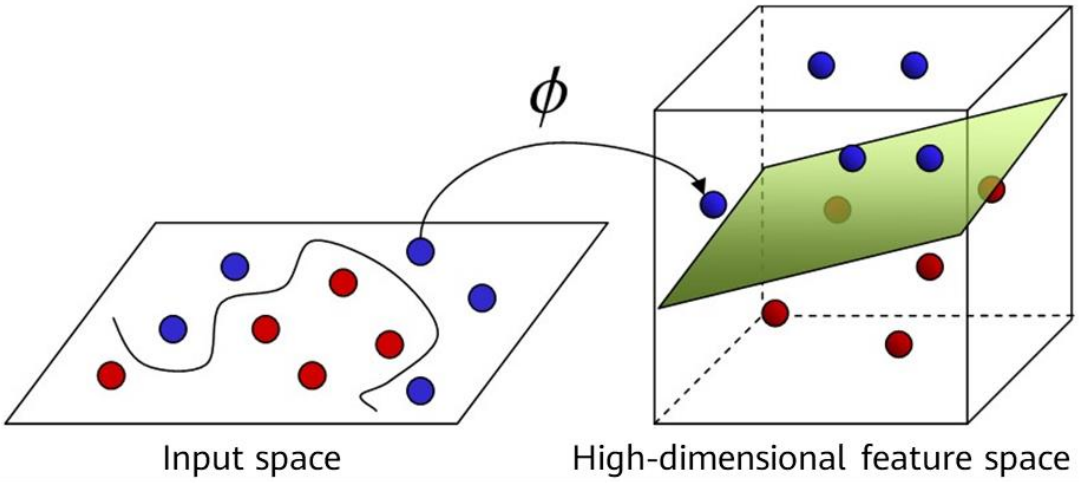


Figure 1-38 Kernel method

1.5.6 KNN

The K-nearest neighbor (KNN) classification algorithm is a theoretically mature method and one of the simplest machine learning algorithms. KNN is a non-parametric method, which usually works well in datasets with irregular decision boundaries. According to this method, if the majority of *K* samples most similar to one sample (nearest neighbors in the eigenspace) belong to a specific category, this sample also belongs to this category.

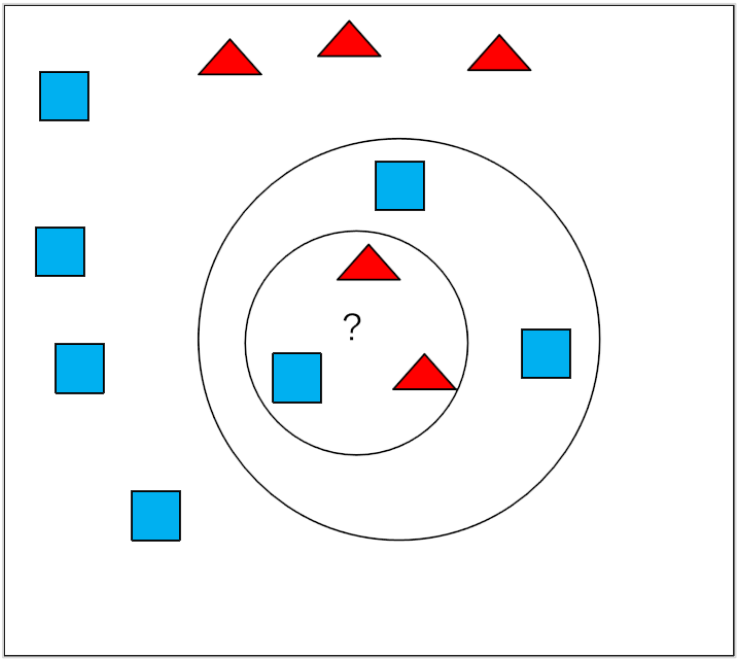


Figure 1-39 KNN example

As the prediction result is determined based on the number and weights of neighbors in the training set, the KNN algorithm has a simple logic. However, like *k*-fold cross validation, *K* in KNN is also a hyperparameter. This means that it is difficult to properly select the *K* value. As shown in 0, when the value of *K* is **3**, the prediction result at the question mark is red triangles. When the value of *K* is **5**, the prediction result at the question mark is blue squares. 0 shows the decision boundaries corresponding to different *K* values. It can be found that the boundary becomes smoother as the value of *K* increases. If the *K* value continues to increase to exceed the number of training samples, the entire plane will eventually become all blue. Generally, a larger *K* value reduces the impact of noise on classification, but obfuscates the boundary between categories. A larger *K* value means a higher probability of underfitting because the decision boundary is too rough. A smaller *K* value means a higher probability of overfitting because the decision boundary is too refined.

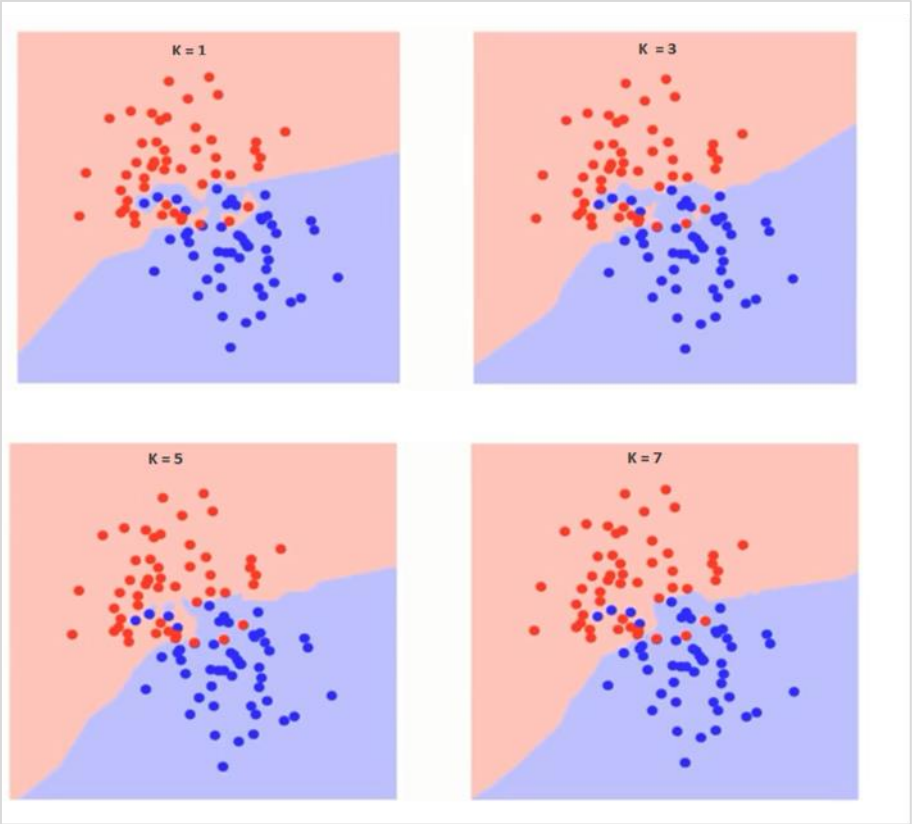


Figure 1-40 Influence of the *K* value on the decision boundary

The KNN algorithm can be used not only for classification prediction, but also for regression prediction. The majority voting method is generally used for classification prediction, and the average value method is generally used for regression prediction. Although these methods seem to be related only to the *K* samples of the nearest neighbor, KNN requires a very large amount of computation. This is because KNN needs to traverse all samples to determine which *K* samples are adjacent to the sample to be tested.

1.5.7 Naive Bayes

Naive Bayes is a simple multicategory algorithm based on the Bayes theorem and assumes that features are independent of each other. For a given sample feature *X*, the probability that a sample belongs to a category *c* is as follows:

$$P(C = c|X) = \frac{P(X|C = c)P(C = c)}{P(X)}$$

In the formula, *P(C=c|X)* indicates the posterior probability, *P(C=c)* indicates the prior probability of the target, and *P(X)* indicates the prior probability of the feature. Generally, we do not consider *P(X)* because it can be regarded as a fixed value during classification, that is:

$$P(C = c|X) \propto P(X|C = c)P(C = c)$$

P(C=c) is irrelevant to *X* and needs to be determined before model training. Generally, *P(C=c)* is calculated based on the ratio of samples whose dataset type is *c*. Therefore, the

core of classification is to calculate *P(X|C=c)*. Assume that feature *X* consists of multiple elements.

$$X = (X_1, X_2, \cdots, X_n)$$

It can be easily calculated as follows:

$$\prod_{i=1}^{n} P(X_i|C = c)$$

By combining the independent hypothesis of features, we can prove that:

$$P(X|C = c) = \prod_{i=1}^{n} P(X_i|C = c)$$

The content of the independent hypothesis of features is that the distribution of each attribute value is independent of the distribution of other attribute values when a given sample classification is used as a condition. Naive Bayes is "naive" precisely because of the use of independent hypothesis of features in its model. Making this hypothesis effectively simplifies computation, and makes the Naive Bayes classifier have higher accuracy and training speed on large databases.

For example, we want to determine a person's gender *C* based on the height X_1 and weight X_2. Suppose that the probabilities of men with a height of 180 centimeters and 150 centimeters are 80% and 20% respectively, and the probabilities of men with a weight of 80 kilograms and 50 kilograms are 70% and 30% respectively. According to the Naive Bayesian model, the probability that a person with a height of 180 centimeters and a weight of 50 kilograms is male is $0.8 \times 0.3 = 0.24$, while the probability that a person with a height of 150 centimeters and a weight of 80 kilograms is male is only $0.7 \times 0.2 = 0.14$. It can be assumed that the two features of height and weight independently contribute to the probability that a person is male.

The performance of the Naive Bayesian model usually depends on the degree to which the independent hypothesis of features is satisfied. In the preceding example, the two features of height and weight are not completely independent. This correlation inevitably affects the accuracy of the model. However, as long as the correlation is not high, we can continue to use the Naive Bayesian model. In actual applications, different features are seldom completely independent of each other.

1.5.8 Ensemble Learning

Ensemble learning is a machine learning paradigm, in which multiple learners are trained and combined to solve the same problem, as shown in 0. When multiple learners are used, the integrated generalization capability can be much stronger than that of a single learner. If you ask a complex question to thousands of person at random and then summarize their answers, the summarized answer is better than an expert's answer in most cases. This is wisdom of the masses.

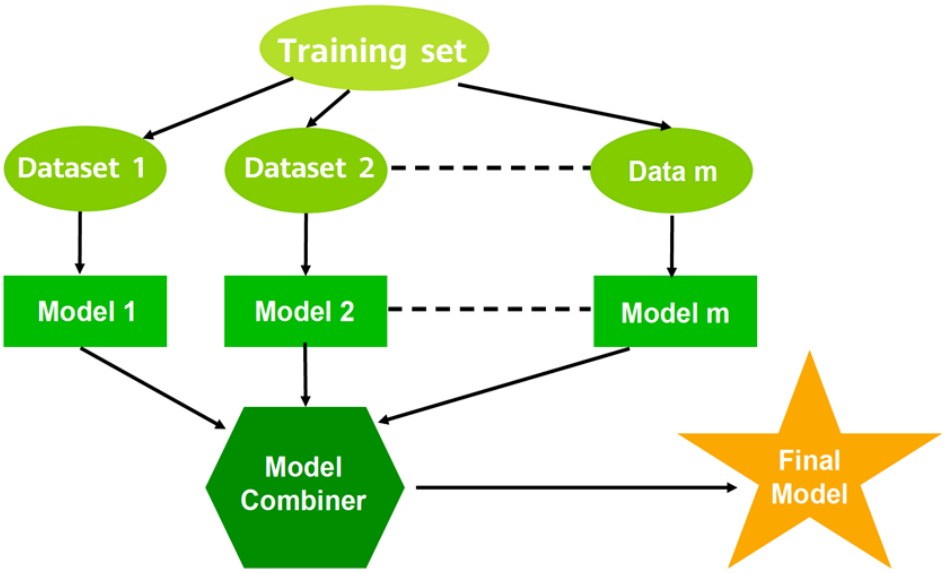


Figure 1-41 Ensemble learning

Ensemble learning can be divided into Bagging and Boosting. The Bagging method independently builds several basic learners and then averages their predictions. Typical models include the random forest. On average, a composite learner is usually better than a single-base learner because of a smaller variance. The Boosting method constructs basic learners in sequence to gradually reduce the bias of a composite learner. Typical models include Adaboost, GBDT, and XGboost. In general, the Bagging method can reduce the variance, thus restraining overfitting. The Boosting method focuses on reducing the bias, thereby improving the capacity of the model, but may cause overfitting.

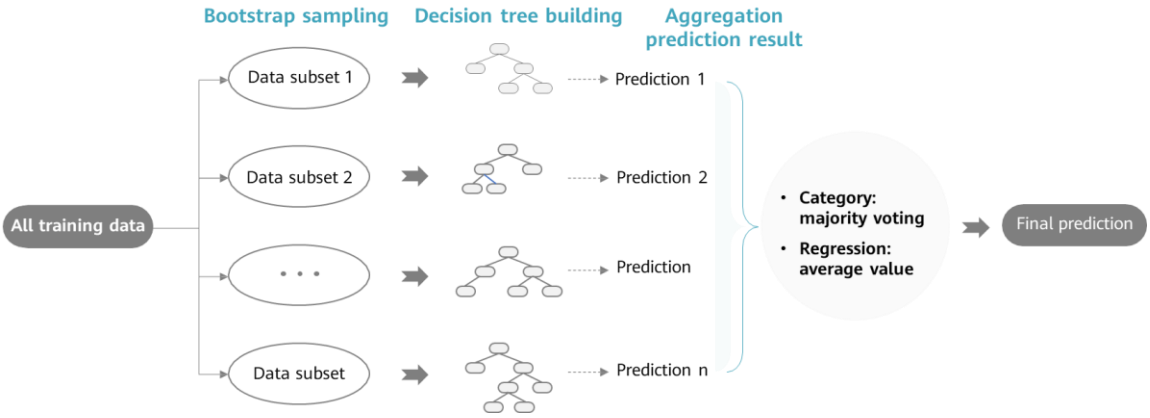


Figure 1-42 Random forest algorithm

Random forest is a combination of the Bagging method and CART decision tree. 0 shows the overall process of the random forest algorithm. The random forest algorithm can be used for classification and regression problems. The basic principle is to build multiple decision trees and merge them together to make prediction more accurate and stable. In

the training process of decision trees, sampling is performed on both the sample level and feature level. At the sample level, the sample subsets used for decision tree training are determined by Bootstrap sampling (repeatable sampling). At the feature level, some features are randomly selected to calculate the information gain before each node of a decision tree is split. By synthesizing the prediction results of multiple decision trees, the random forest model can reduce the variance of a single decision tree model, but cannot effectively correct the bias. Therefore, the random forest model requires that each decision tree cannot be underfitting, even if this requirement may lead to overfitting of some decision trees. In addition, each decision tree model in the random forest is independent of each other. Therefore, the training and prediction processes can be executed concurrently.

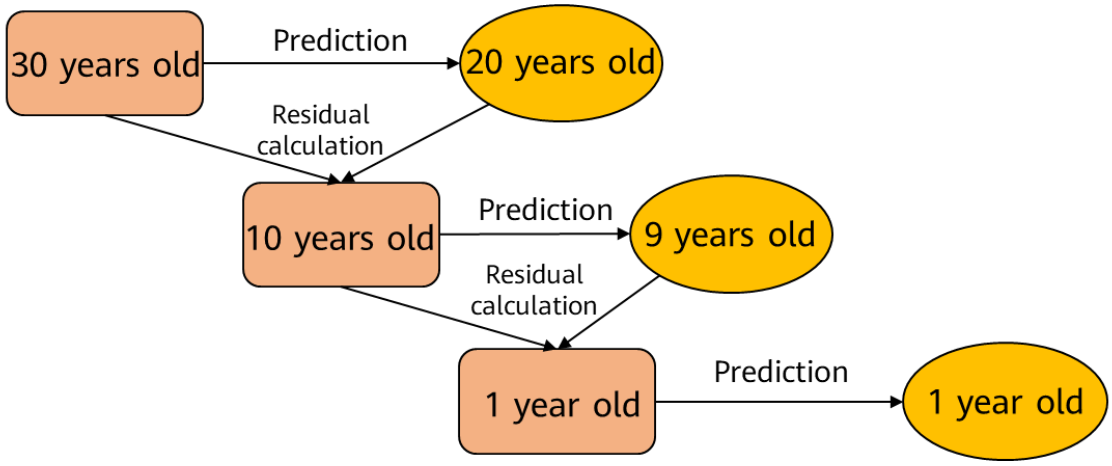


Figure 1-43 GBDT algorithm

Gradient boosting decision tree (GBDT) is one of the Boosting algorithms. The prediction value of the model is the sum of the results of all decision trees. The essence of GBDT is to continuously use new decision trees and learn the residuals of all previous decision trees, that is, the errors between predicted values and actual values. As shown in 0, the prediction result of a sample in the first decision tree is 20 years old, but the actual age of the sample is 30 years old. The difference between the predicted value and the actual value is 10 years. If we use another decision tree to predict this difference, we can improve the 20-year-old prediction result to bring it closer to 30. Based on this idea, we introduce a second decision tree to learn the error of the first decision tree, and so on. Finally, the actual value of 30 year old is obtained by adding the prediction results of the three learners. GBDT improves the accuracy by continuously correcting the bias of decision trees. Therefore, some underfitting is allowed for decision trees. However, GBDT cannot correct the variance. Therefore, overfitting is not allowed for decision trees. This is one of the biggest differences between the Boosting and Bagging algorithms. In addition, the training data of each decision tree in GBDT depends on the output of the previous decision tree. Therefore, concurrent model training is not supported.

1.5.9 Clustering Algorithm

K-means clustering aims to partition *n* observations into *K* clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. As shown in 0, the obtained clusters meets the following condition: The similarity of objects is high in the same cluster but low in different clusters.

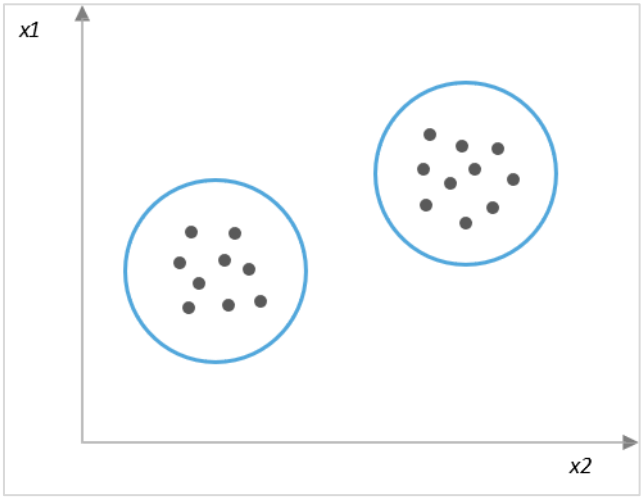


Figure 1-44 Clustering algorithm

Compared with the *K*-means algorithm, the hierarchical clustering algorithm not only outputs the clustering result, but also outputs the tree-like relationship between samples. As shown in 0, the hierarchical clustering algorithm divides a dataset at different layers and forms a tree-like clustering structure. The dataset division can use a "bottom-up" aggregation policy, or a "top-down" splitting policy. The hierarchy of clustering is represented in a tree graph. The root is the cluster of all samples, and the leaves are the cluster of only a sample.

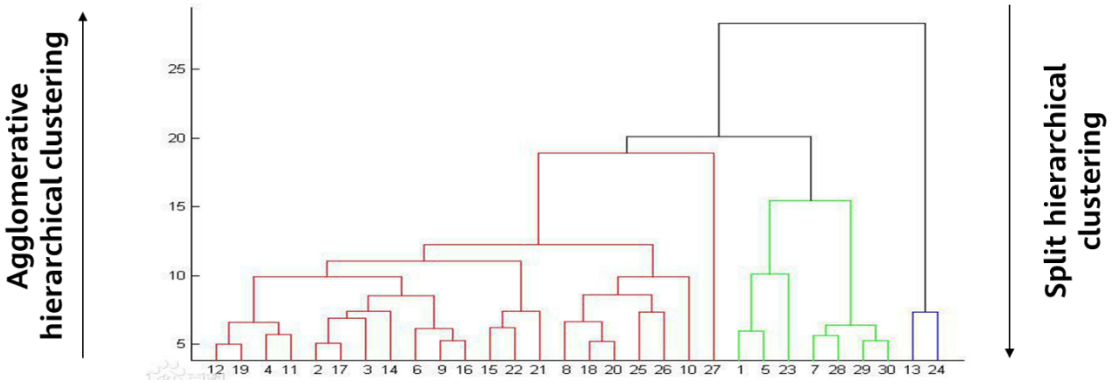


Figure 1-45 Hierarchical clustering

1.6 Case Study

This section reviews the overall process of machine learning by using a specific case. Assume that a dataset contains the house areas and prices of 21,613 housing units in a city, as shown in 0. Based on the data, we can train a model to predict the prices of other housing units in the city.

x	y
House Area	Price
1,180	221,900
2,570	538,000
770	180,000
1,960	604,000
1,680	510,000
5,420	1,225,000
1,715	257,500
1,060	291,850
1,160	468,000
1,430	310,000
1,370	400,000
1,810	530,000
...	...

Dataset

Figure 1-46 House price dataset

By analyzing the data, we can find that the input (house area) and output (price) in the data are continuous values. Therefore, we can use the regression model of supervised learning. The project aims to build a model function $h(x)$ that infinitely approaches the function that expresses the true distribution of the dataset. 0 shows the scatter chart of the data and a possible model function.

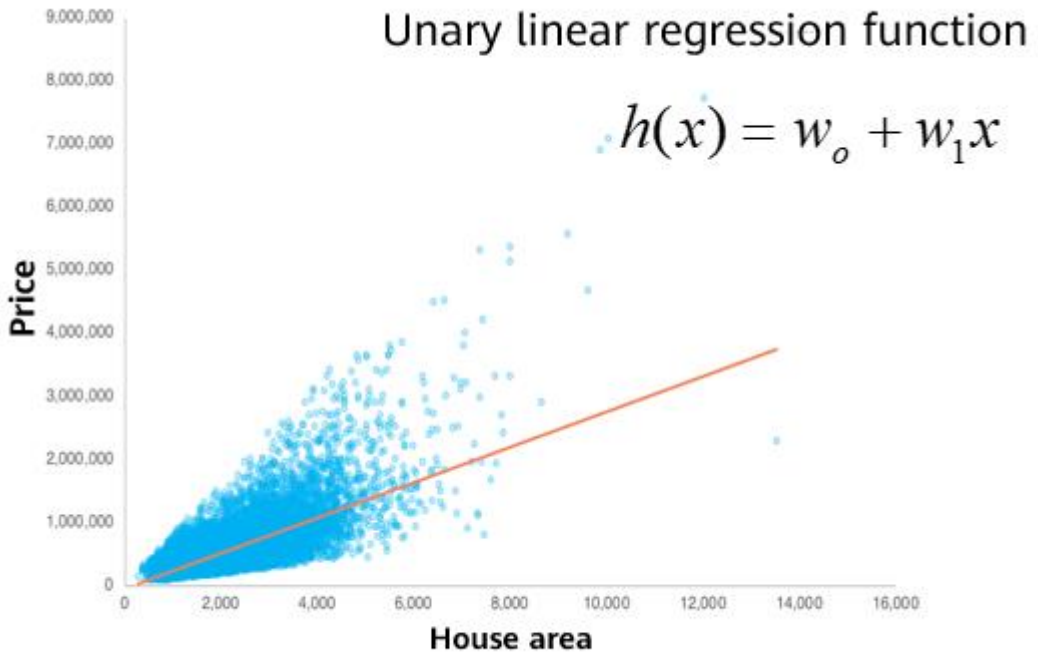


Figure 1-47 Model hypothesis

Linear regression aims to find a straight line that best fits the dataset, that is, determine the parameter *w* in the model. To find the optimal parameter, construct a loss function and find the parameter values when the loss function becomes the minimum.

$$J(w) = \frac{1}{2m}\sum(h(x) - y)^2$$

In the preceding formula, *m* indicates the number of samples, *h(x)* indicates the predicted value, and *y* indicates the actual value. Intuitively, the loss function represents the sum of squares of errors between all samples and the model function, as shown in 0. In normal cases, when the loss is minimized, all samples should be evenly distributed on both sides of the fitting straight line. In this case, the fitting straight line is the required model function.

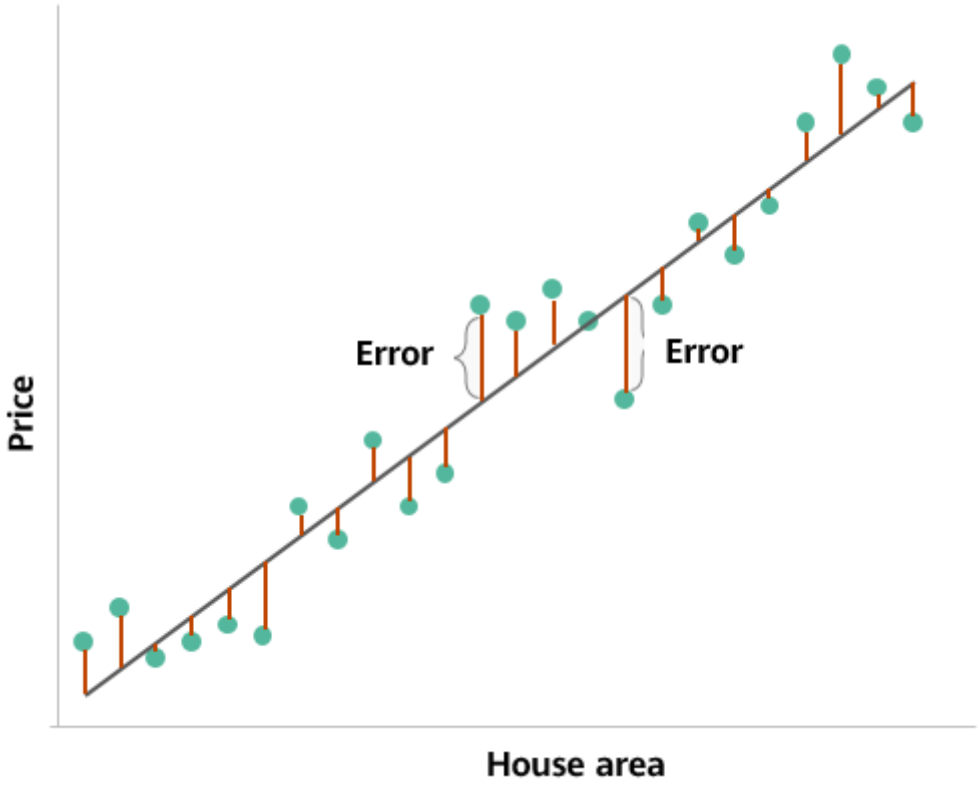


Figure 1-48 Geometric meaning of errors

As described above, a gradient descent algorithm finds the minimum value of a function by using the iteration method. As shown in Figure 1-49, a gradient descent algorithm randomly selects an initial point on the loss function, and then finds the global minimum value based on the negative gradient direction. This parameter value is the optimal parameter value.

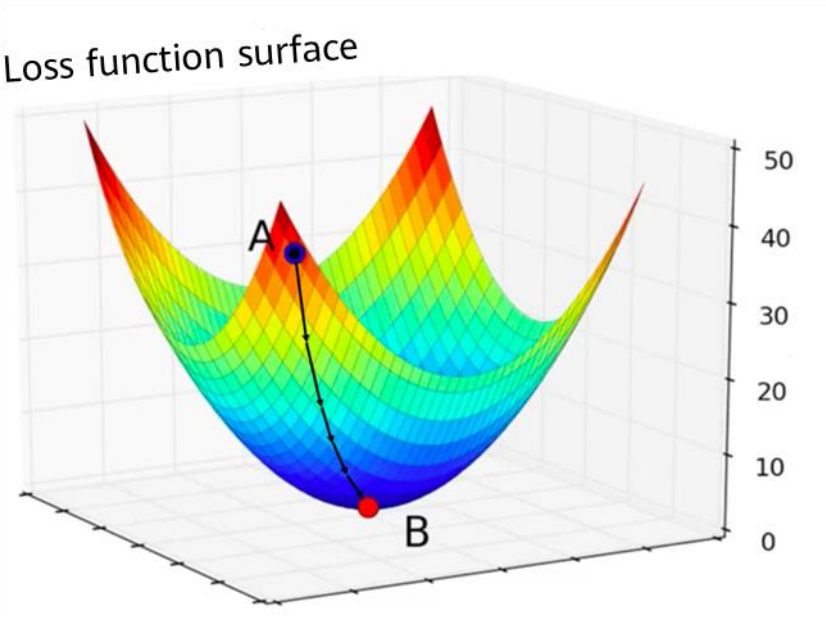


Figure 1-49 Loss function surface

Point A in Figure 1-49 indicates the position of parameter w after random initialization. Point B indicates the global minimum value of the loss function, that is, the final parameter value. The A-B connection line indicates the formed based on descents in the negative gradient direction. The value of parameter w changes in each iteration. As a result, the regression line changes continuously. 0 is an example of gradient descent iteration. As observed, red points on the loss function surface gradually approach the lowest point, and fitting of the red line of linear regression with data becomes better. Finally, we can obtain the optimal model function $h(x) = 280.62x - 43581$.

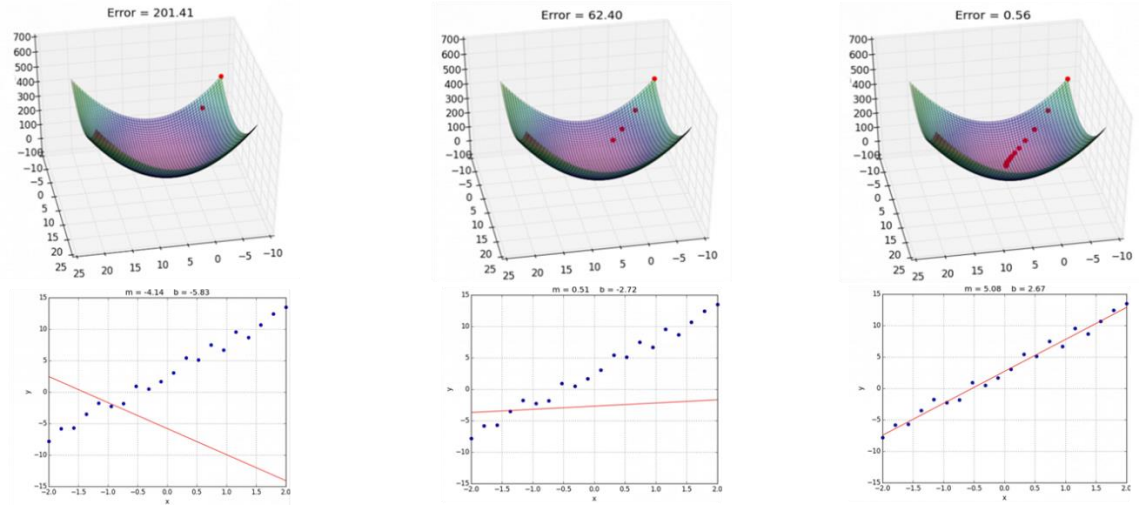


Figure 1-50 Visualized gradient descent process

After model training is complete, we need to use the test set for testing to ensure that the model has a strong generalization capability. If overfitting occurs during testing, add a regular term to the loss function and adjust hyperparameters. If underfitting occurs during testing, use a more complex regression model, such as GBDT. Afterwards, we need to retrain the model and test it again using the test set until the generalization capability of the model meets expectations. Note that data cleansing and feature engineering cannot be ignored because real data is used in the project.

1.7 Summary

This chapter first describes the definition and classification of machine learning, as well as problems machine learning solves. Then, it introduces key knowledge points of machine learning, including the overall procedure (data collection, data cleansing, feature selection, model training, model evaluation, and model deployment), common algorithms (including linear regression, logistic regression, decision tree, SVM, Naive Bayes, KNN, ensemble learning, and K-means), gradient descent algorithms, and hyperparameters. Finally, a complete machine learning process is presented by the case of using linear regression to predict house prices.

1.8 Quiz

1. Machine learning is the core technology of AI. Please define machine learning.
2. The generalization error of a model can be divided into variance, bias, and irreducible error. What is the difference between variance and bias? What are the characteristics of variance and bias of an overfitting model?
3. Please calculate the value of F_1 for the confusion matrix shown in 0.
4. In machine learning, a dataset is generally divided into the training set, validation set, and test set. What is the difference between the validation set and test set? Why do we need to introduce the validation set?
5. Linear regression models use linear functions to fit data. How does a linear regression model process non-linear data?
6. Many classification models can only deal with binary-classification problems. Try to provide a method, using SVM as an example, to deal with multiclass classification problems.
7. How does the Gaussian kernel function in the SVM map a feature to an infinite dimensional space?
8. Is gradient descent the only way to train a model? What are the limitations of this algorithm?