Introduction to Supervised Machine Learning for Data Science

Mohammad Samy BALADRAM1,‡, Atsushi KOIKE1,2,‡ and Kazunori D YAMADA1,3,*

1Graduate School of Information Sciences, Tohoku University, Sendai 980-8579, Japan
2Ichinoseki College, National Institute of Technology, Ichinoseki 021-8511, Japan
3Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology, Tokyo 135-0064, Japan

We present an introduction to supervised machine learning methods with emphasis on neural networks, kernel support vector machines, and decision trees. These methods are representative methods of supervised learning. Recently, there has been a boom in artificial intelligence research. Neural networks are a key concept of deep learning and are the origin of the current boom in artificial intelligence research. Support vector machines are one of the most sophisticated learning methods from the perspective of prediction performance. Its high performance is primarily owing to the use of the kernel method, which is an important concept not only for support vector machines but also for other machine learning methods. Although these methods are the so-called black-box methods, the decision tree is a white-box method, where the judgment criteria of prediction by the predictor can be easily interpreted. Decision trees are used as the base method of ensemble learning, which is a refined learning technique to improve prediction performance. We review the theory of supervised machine learning methods and illustrate their applications. We also discuss nonlinear optimization methods for the machine to learn the training dataset.

KEYWORDS: supervised learning, nonlinear optimization, neural network, kernel method, ensemble learning

1. Overview of Machine Learning and Supervised Learning Methods

1.1 Classification of machine learning

Machine learning can be separated into supervised and unsupervised learning methods. Supervised learning attempts to determine a function or relationship based on labeled training data and uses the function to map new unlabeled data. It includes multilayer perceptron (MLP), logistic regression, decision tree, support vector machine (SVM), etc. In these methods, a model is developed from a training dataset where the input and output values are previously known. Supervised learning methods require a sufficient number of labeled records to create a good model. These methods uncover hidden patterns in unlabeled data.

In unsupervised learning, there are no output variables to be predicted. The objective of unsupervised learning methods is to identify patterns in data based on relationships among data points themselves. Unsupervised learning methods include hierarchical clustering, k-means clustering, autoencoder, principal component analysis. A choice between supervised or unsupervised learning methods is based on the problems that are to be solved.

1.2 Overview of machine learning

In the era of big data, the importance of machine learning methods is rapidly growing as increasing amounts of data are available owing to significant progress in information technologies. When analyzing data using machine learning methods, we read the data, extract features from it, and identify rules of the data using a machine, which is usually a computer. Generally, the objective of analysis with machine learning is to elucidate the relationship between instances (data points) in the given data or to predict certain properties of unknown data related to the given data. The problems to be solved by supervised learning methods are traditionally categorized into two types: regression and classification problems [1].

In the regression problem, we predict a numerical value when input data are given. For example, the problem of predicting the sales of a shop from other data is a regression problem. In the classification problem, we predict a class (or flag) of the input data. A well-known classification problem is a binary classification problem in which we predict the class of input data as 0 or 1.
Figure 1 shows an illustration of the classification problem. Let us assume that we are given four instances, each of which consists of a two-dimensional input vector and one-dimensional target vector (also called teacher vector), as listed in Table 1.

The input vectors are plotted in the $xy$-coordinate plane and are separated by their target vectors. The aim is to judge whether a new input vector, which is represented by a star, has a value of 0 or 1. In this two-dimensional problem, machine learning methods attempt to determine a classification curve. To address this problem, we determine a classification curve $y = f(x) = wx + b$ in Fig. 1(b), where $w$ and $b$ are called the parameters of the equation that must be determined. In fact, we can obtain optimal values of the parameters using machine learning methods. In this example, the class of a new input vector is predicted to be zero because it lies in the lower half of the plane, separated by the classification curve. Several variations of machine learning methods have been introduced. The primary difference between these methods is based on the techniques used to determine the classification curves. Most machine learning methods are useful for both regression and classification problems.

Now, let us discuss the supervised learning methods. In general, a supervised learning method produces a predictor, which accepts input data and outputs a prediction result against the input data. Machine learning with supervised learning methods includes two phases: learning and prediction phases. In the learning phase, the machines read the input and target vectors and improve the rules based on them. The machines repeat this procedure until the quality of the rules becomes sufficiently good. A learning process is also called an inference process. After the learning process, the machines are called predictors. The goal of the learning process is to achieve good prediction performance for as long as possible. Finally, in the prediction phase, the properties or features of new data are predicted by the predictor.

### 1.3 Organization of this paper

Because the aforementioned basic concept of supervised learning methods is common, in this paper, we introduce a few supervised learning methods together with mathematical optimization methods, which are utilized to optimize the parameters of machine learning methods. In Sect. 2, we introduce gradient descent and Newton’s methods, which are well known as nonlinear programming methods. Currently, gradient descent methods are utilized for parameter optimization of neural networks; moreover, tuning of the algorithm itself is sometimes required for good learning. In Sect. 3, we introduce the principle of a neural network and their applications. The recent boom in artificial intelligence research was sparked by the rediscovery of the importance of neural networks from the perspective of deep learning. In Sect. 4, we introduce kernel methods and their application methods. In particular, kernel SVM is one of the best learning methods from the perspective of classification performance. In Sect. 5, we introduce the principle and applications of the decision tree and ensemble method. Unlike neural networks and SVM, it is easy to interpret judgment criteria using the predictor constructed from the decision tree. From this perspective, the decision tree is an important learning method.

In this article, we denote a scalar in normal font, a vector in bold font, and matrices as capitalized in bold font. If we temporarily change this rule for simplicity of explanation, we present a new definition each time.

### Table 1. Input vectors and corresponding target vectors, as shown in Fig. 1.

| Input Vectors | Target Vectors |
|---------------|----------------|
| (1,1)         | (0)            |
| (1,3)         | (1)            |
| (2,4)         | (1)            |
| (4,1)         | (0)            |
2. Nonlinear Optimization Methods in Machine Learning

2.1 Overview

Mathematical optimization is essential for machine learning of input data. The word “optimize” indicates an attempt to determine “optimal solutions” of given problems. In the context of machine learning, the given problem is generally a function and it is called “objective function.” In the machine learning field, a nonlinear optimization or programming method is more commonly used. Nonlinear programming studies the general case in which objective functions, constraints, or both contain nonlinear parts. Nonlinear programming deals with minimizing a real-valued objective function \( f \) subject to certain constraint functions \( g_i, i = 1, \ldots, l \) for certain positive integers \( l \). Conversely, optimization problems can also occur without constraint functions. This is called an “unconstrained optimization” problem. For example, for a function \( f \) defined as \( f(x) = (x - 1)^2 \) for \( x \in \mathbb{R} \), the optimal solution is \( x = 1 \), as this minimizes the objective function \( f \). In this section, we focus on the unconstrained optimization problem and then introduce the most representative methods to solve this problem, which includes the gradient descent and Newton’s methods.

2.2 Gradient descent method

2.2.1 Definition of gradient

Before we discuss the gradient descent method, it is important to understand the basics of the gradient itself. The following definition describes the mathematical formula for the gradient.

**Definition 2.1.** Let \( f \) be a function defined in the \( n \)-dimensional field \( \mathbb{R}^n \). The gradient function, denoted by \( \nabla f \), is a vector function where its components are the partial derivatives of \( f \), that is, for \( x = (x_1, \ldots, x_n) \in \mathbb{R}^n \),

\[
\nabla f(x) = \left( \frac{\partial}{\partial x_1} f(x), \frac{\partial}{\partial x_2} f(x), \ldots, \frac{\partial}{\partial x_n} f(x) \right)^\top.
\]

Here, Fig. 2 shows the curve \( y = f(x) = x^2 \) and the gradients are represented by the colored arrows. For any point \( x \), the gradient function can be computed as \( \nabla f(x) = 2x \). Thus, the gradients at \( x = 4 \) and \( -2 \) are 8 and \( -4 \), respectively. As mentioned above, the gradient is a vector and it possesses direction and magnitude, which corresponds to the slope of the function.

![Fig. 2. Illustration of gradient of the curve \( y = x^2 \). The gradients at \( x = 4 \) and \( -2 \) are 8 and \( -4 \), respectively. The arrows represent the gradient vectors at these points.](image)

When we consider a function, the goal of the gradient descent method is to determine a value \( x^* \), which minimizes \( f(x^*) \). In a function with one variable, it is well known that the value of \( x \) for which the derivative of \( f(x) \) with respect to \( x \) is equal to zero corresponds to a local maximum, a local minimum, or an inflection point of the function \( f(x) \). For \( n \)-dimensional cases, we can extend this as follows.

**Theorem 2.2.** Let \( f \) be a differentiable function and \( x^* \) be an extremum on \( f \). Then,

\[
\nabla f(x^*) = 0,
\]

where \( 0 \) denotes a null vector, whose dimension is identical to that of the gradient vector.
2.2.2 Principle of the gradient descent method

The gradient descent method is a mathematical optimization method that attempts to determine an extreme value of a given function. This method is the most popular optimization method for nonlinear programming. Currently, one of the most important applications of the gradient descent method is the neural network. The advantage of using the gradient descent method is that its algorithm is relatively simple and implementation is easy; moreover, it has global convergence and the computational space complexity is good. In the gradient descent method, the final solution, namely the optimal value, is obtained by iterating and updating randomly chosen initial values (= initial parameters) to the optimal values gradually. The steps of this method are shown in the following algorithm:

Algorithm 1 Update rule of the gradient descent method

Require: Objective function \( f(\theta) \), Initial parameter \( \theta_0 \), Learning rate \( \alpha \)
1: \( t = 0, \ \mathbf{g}_0 = \nabla f(\theta_0) \)
2: while \( \mathbf{g}_t \neq \mathbf{0} \) do
3: \( \mathbf{g}_t \leftarrow \nabla f(\theta_t) \)
4: \( \theta_{t+1} \leftarrow \theta_t - \alpha \mathbf{g}_t \)
5: \( t \leftarrow t + 1 \)
6: end while

Thus, in the algorithm, first, an initial parameter, \( \theta_0 \), is randomly set. Generally, the initial parameter is derived from a uniform distribution. Second, if \( \theta_t \) is the optimal value or is extremely close to it, the iterative calculation is completed. As explained in Theorem 2.2, the iterative calculation is terminated if the gradient is \( \mathbf{0} \). Third, the parameter is updated using the following update rule:

\[
\theta_{t+1} = \theta_t - \alpha \mathbf{g}_t, \tag{2.3}
\]

where \( \alpha \) and \( \mathbf{g}_t \) are the learning rate and gradient at timestep \( t \), respectively. In this step, new parameters are updated by the former parameters and gradient. Here, the learning rate is normally a positive small value. The learning rate is described in the following subsections. Finally, this step is repeated until the gradient is \( \mathbf{0} \).

As an example, consider the optimization problem of the function \( f \) in the previous subsection. Let us consider that \( x = 4 \); then, the current timestep is \( t \). We can write this as \( x_t = 4 \). The gradient vector on this point has a positive value (the arrow pointing to the right in Fig. 2); therefore, according to (2.3), the parameter at timestep \( t + 1 \) must be updated to a smaller number. Conversely, if the parameter is \( x_t = -2 \), the gradient vector has a negative value (the arrow pointing to the left in Fig. 2); thus, the parameter at timestep \( t + 1 \) is updated to a larger number. In both cases, the sequence of the update leads to \( x = 0 \), where the function reaches its optimum value.

2.2.3 Adjusted gradient descent method

In the actual implementation of the method, we rarely obtain the true optimum when implementing Algorithm 1, that is, the gradient vector never reaches \( \mathbf{0} \). To prevent this from happening, we can start by introducing a new constant \( \epsilon \). This value can be a predetermined arbitrary small number or the so-called machine epsilon, which is the difference between the minimum number that is greater than 1 and 1 in the computer. The updated algorithm of the gradient descent method is shown in the following algorithm.

Algorithm 2 Actual implementation of the gradient descent method in a computer

Require: Objective function \( f(\theta) \), Initial parameter \( \theta_0 \), Learning rate \( \alpha \), Constant value \( \epsilon \)
1: \( t = 0, \ \mathbf{g}_0 = \nabla f(\theta_0) \)
2: while \( \| \mathbf{g}_t \| > \epsilon \) do
3: \( \mathbf{g}_t \leftarrow \nabla f(\theta_t) \)
4: \( \theta_{t+1} \leftarrow \theta_t - \alpha \mathbf{g}_t \)
5: \( t \leftarrow t + 1 \)
6: end while

Here, \( \| \mathbf{g}_t \| \) is the \( L^2 \)-norm of gradient at timestep \( t \) and \( L^2 \)-norm \( \| \mathbf{x} \| \) of \( n \)-dimensional vector \( \mathbf{x} \) is calculated as follows:

\[
\| \mathbf{x} \| = \sqrt{n \sum_{i=1}^{n} x_i^2}. \tag{2.4}
\]

Thus, if the \( L^2 \)-norm of the gradient is sufficiently small, the iteration process is terminated.

2.2.4 Learning rate

In Algorithms 1 and 2, we use the notation \( \alpha \) to represent the learning rate; it is also sometimes called the step size. The learning rate is a hyperparameter that determines the value of the updated parameter computed with values in the
gradient vector. The value of this parameter is set by the experimenters before the iteration process begins. The magnitude of the learning rate is determined empirically according to the given problems. Figure 3 shows the differences in the progress of the gradient descent depending on the magnitude of the learning rate. We observe that this curve takes an extremum at \( x = 0 \). If the learning rate is too small, as shown in Fig. 3(a), the magnitude of progress becomes too small as well, and thus it requires a much longer time to reach the optimal value. Similarly, if the learning rate is too large, as illustrated in Fig. 3(b), the parameter perturbs and consequently it requires a longer time to reach the optimum. In contrast, if a proper learning rate is set as shown in Fig. 3(c), the optimum is reached in a shorter period of time.

Adjusting or determining the most suitable learning rate is an important topic in the machine learning field. Various studies were conducted to decide or adjust the best way of setting the value of the learning rate. The simplest method divides the learning rate by two with the progress in the iteration. This procedure is shown in Algorithm 3.

**Algorithm 3** Gradient descent method with the adjusted learning rate

| Require: Objective function \( f(\theta) \), Initial parameter \( \theta_0 \), Learning rate \( \alpha_t \), Constant value \( \epsilon \) |
|---|
| 1: \( t = 0, \ g_0 = \nabla f(\theta_0) \) |
| 2: while \( \| g_t \| > \epsilon \) do |
| 3: \( \ g_t \leftarrow \nabla f(\theta_t) \) |
| 4: \( \theta_{t+1} \leftarrow \theta_t - \alpha_t g_t \) |
| 5: \( \alpha_{t+1} \leftarrow \alpha_t / 2 \) |
| 6: \( t \leftarrow t + 1 \) |
| 7: end while |

Here, \( \epsilon \) is a small constant value, which is used as a threshold for convergence. While this method is relatively straightforward, many improved versions of algorithms have been developed to optimize the gradient descent method. The report by Ruder [2] includes several state-of-the-art methods.

### 2.2.5 Convergence rate of the gradient descent method

One of the important features of an optimization method is the convergence rate, namely the speed at which the extreme value is reached. If an optimization method satisfies the following expression:

\[
\epsilon_{t+1} \leq k \epsilon_t, \tag{2.5}
\]

the convergence rate of the method is considered to be a linear convergence. Here, \( \epsilon_t \) is the difference between an output obtained using an optimization method and the true value, while \( k \) is some positive number \((0 < k < 1)\). Conversely, if an optimization method satisfies the following expression:

\[
\epsilon_{t+1} \leq k \epsilon_t^2, \tag{2.6}
\]

the convergence rate of the method is considered to be a quadratic convergence. Here, \( k \) is some positive number \((k > 0)\), and the inequality implies that the error between the true value and the output value reduces in a quadratic manner by one iteration of the update rule. Evidently, the convergence rate of the method with quadratic convergence is faster than that with linear convergence. Although the gradient descent method is the most popular optimization method for nonlinear programming, its convergence rate is not faster than other methods and it is of linear convergence.
2.3 Newton’s method

2.3.1 Definition of Hessian

Before we explain this optimization method, it is important to understand the concept of Hessian, in addition to the aforementioned gradient descent method. With respect to a function $f$, the Hessian is a square matrix that includes second-order partial derivatives of a scalar function or gradient of the gradient vector function $\nabla f$. The Hessian is denoted by $H$ or $\nabla^2 f$ and can be expressed as follows:

$$
H_f(x) = 
\begin{bmatrix}
\frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\
\frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2}
\end{bmatrix}.
$$

(2.7)

For a smooth $f(x)$ (or for a twice continuously differentiable function $f(x)$) the Hessian becomes a symmetric matrix.

2.3.2 Principle of the Newton’s method

The procedure of the Newton’s method is shown in Algorithm 4, where $t$, $g_t$, and $H_t^{-1}$ indicate the timestep, the gradient at timestep $t$, and an inverse matrix of Hessian at timestep $t$, respectively. First, an initial parameter $\theta_0$ is set randomly. This step is identical to that of the gradient descent method. Next, if $\theta_t$ is equal to or close to the optimal value, the iteration process is terminated. Next, the parameter is updated using the following update rule:

$$
\theta_{t+1} \leftarrow \theta_t - H_t^{-1} g_t.
$$

(2.8)

This update rule is similar to that of the gradient descent. The difference is that this update rule does not have the learning rate. Instead in the algorithm of the Newton’s method, an inverse matrix of Hessian is utilized. Finally, this update rule is repeated until the objective function takes the optimum value or its closest value, which is determined by the constant $\epsilon$, which is a small value close enough to zero.

**Algorithm 4 Update rule of Newton’s method**

**Require**: Objective function $f(\theta)$, Initial parameter $\theta_0$, Constant value $\epsilon$

1: $t = 0$, $g_0 = \nabla f(\theta_0)$, $H_0 = \nabla^2 f(\theta_0)$
2: while $\|g_t\| > \epsilon$ do
3: $g_t \leftarrow \nabla f(\theta_t)$
4: $H_t \leftarrow \nabla^2 f(\theta_t)$
5: $\theta_{t+1} \leftarrow \theta_t - H_t^{-1} g_t$
6: $t \leftarrow t + 1$
7: end while


The advantage of using Newton’s method is that it has a relatively fast computational speed in reaching the optimal value, while the limitation is that it requires large space complexity. As shown in Algorithm 4, the Newton’s method updates the parameter by $-H_t^{-1} g_t$. This vector is called Newton’s direction. This value can be obtained as follows. First, we consider the objective function $f$. Next, to determine a point $x$ on $f$, we define $d$ as finite increments. We can calculate the Taylor series around $x$ using the following equation:

$$
f(d + x) = f(x) + g^\top d + \frac{1}{2} d^\top H d,
$$

(2.9)

where $g$ and $H$ are the gradient vector and Hessian matrix, respectively. In this process, we can perform a quadratic approximation by ignoring the terms of the third and higher orders. Another way of writing this equation is as follows:

$$
f(d + x) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} H_{ij} d_i d_j + \sum_{i=1}^{n} g_i d_i + f(x),
$$

(2.10)

where $H_{ij}$ is the $(i,j)$-element of the Hessian $H$ and $d_i$ is the $i$-th element of the gradient vector $g$. Next, we calculate the extreme value of the function. First, we apply the partial derivatives to both sides of (2.10). As the function takes extremum when the gradient of the function is 0, as shown in Theorem 2.2, to determine it, we calculate the gradient of the function as follows:
Based on Theorem 2.2, the extremum of the function is calculated by the following equation

\[ \nabla f(d + x) = 0 \]  \hspace{1cm} (2.13)

Hence, we can write (2.11) as

\[ d = -H^{-1}g. \]  \hspace{1cm} (2.14)

As we are focusing on the point around the increment, we define a variable \( x^+ \) as follows:

\[ x^+ = x + d. \]  \hspace{1cm} (2.15)

Thus, the quadratic-approximated function \( f \) takes an extreme value at \( x^+ \). This implies that we can determine an extreme value by updating \( x \) by the update rule of (2.8).

### 2.3.3 Convergence rate of the Newton’s method

The following theorem is used to determine the convergence rate of this method.

**Theorem 2.3** (Convergence rate of Newton’s method). Let \( \epsilon_t \) be the difference between the output value of timestep \( t \) and the true value, and \( k \) be some positive number \( (k > 0) \). At this time, Newton’s method satisfies the following expression:

\[ \epsilon_{t+1} \leq k\epsilon_t^2. \]  \hspace{1cm} (2.16)

As mentioned above, the convergence performance of Newton’s method follows quadratic convergence; consequently, this method is superior to the gradient descent method, whose convergence rate is a linear convergence. Conversely, the calculation cost of Newton’s method is not good because of the computation of the inverse matrix of Hessian. The time complexity for calculating an inverse matrix is normally \( O(n^3) \), where \( n \) is the number of dimensions of an objective function.

Neural networks use several parameters, which can sometimes be greater than 10 million. In these cases, it is impossible to implement Newton’s method for the learning process. Consequently, the gradient descent method is generally used for the learning process of a neural network instead of the Newton’s method. However, the fact that the Newton’s method converges to a second order is significantly attractive. Thus, to overcome the disadvantage of Newton’s method, various quasi-Newton methods were developed [3, 4]. In addition to the gradient descent method and Newton’s method, these methods can be an option for better optimization.

### 2.4 Summary

In summary, we first described the two representatives of nonlinear optimization methods, that is, the gradient descent method and Newton’s method. Currently, one of the most important applications of the gradient descent method is in deep learning. Thus, various improved algorithms for the gradient descent method have been developed recently. Conversely, in the era of big data, Newton’s method is not utilized for neural network optimization because of its complexity in computing the inverse matrix of Hessian. However, its superior convergence rate is considerably attractive. In fact, several algorithms with improved gradient descent techniques attempt to incorporate the merits of the Newton’s method [5]. As developing optimizers is a trending topic in the field of artificial intelligence research, it will be useful to learn algorithms of various optimization methods related to these two methods.

### 3. Neural Networks and Deep Learning

#### 3.1 Overview

##### 3.1.1 Various neural networks

In this section, we introduce the principle of neural networks, deep learning, and their applications. In the animal brain, neurons communicate information from the brain to other neurons. The communication steps include inputting data (signals), activating neural cells and synapses (receivers of the signals), and outputting signals. This complex system achieves the high-quality performance of the animal brain, such as cognition, thinking, and emotions. The neural network is a machine learning method that mimics the behaviors of the animal brain and nervous system. This method was developed more than sixty years ago. Before we elaborate more about neural networks, we must familiarize ourselves with certain key terms, that is, deep learning and artificial intelligence.

A term closely related to neural networks is deep learning. Practically, this is a neural network. To understand this
term, we must understand that the neural network uses “layers” in its algorithm. This network that contains multiple layers is called deep learning. In addition, artificial intelligence is the intelligence demonstrated by machines or more specifically by a computer. It can be achieved using machine learning methods, including neural networks. Thus, machine learning includes the neural network, and the neural network includes deep learning.

There are several architectures of deep learning methods, including MLP, convolution neural network (CNN), recurrent neural network (RNN), and generative adversarial network (GAN). These methods can achieve artificial intelligence.

3.1.2 Brief history of neural network

The Turing Award is an annual award presented by the Association for Computing Machinery to an individual selected for contributions “of lasting and major technical importance to the computer field.” The Turing Award is generally recognized as the highest distinction in computer science and is considered as the “Nobel Prize of computing.” The recipients of the Turing Award in 2018 are Yoshua Bengio, Geoffrey Hinton, and Yann LeCun. They won the award for the following reason: “For conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.” Neural networks, including deep learning studies, are now booming globally. Several researchers have used neural networks as a tool for analyzing data and several researchers are developing novel neural network architectures.

This boom in neural networks has not occurred for the first time, but for the third time. The first boom occurred in the 1950s. The boom was induced by the development of a perceptron. The perceptron was developed by Rosenblatt in 1958 [6]. It is a one-layer neural network, and the layer has only one neuron. The perceptron can converge to an optimal value only if the input and target data can be linearly separated. However, the perceptron has several defects. First, it cannot learn nonlinear relationships in the given data. This is a big problem because most real-world data cannot be linearly separated. Second, it is not robust against noise in data and requires considerable time for convergence. The first boom in artificial intelligence was overdue for these reasons. The second boom in artificial intelligence occurred with the development of the MLP and universal approximation. This boom lasted from the 1980s to the 1990s.

The most important findings regarding the second boom were backpropagation [7] and the universal approximation theory [8]. Backpropagation is a method that allows a predictor to efficiently learn input data. The universal approximation theory is a theory that clarified that an MLP can mimic any nonlinear function. This theory is considerably important because it ensures the use of neural network methods for any problem globally. However, the MLP has several defects. First, the biggest challenge is the vanishing gradient problem, where the gradient vanishes depending on the learning iteration. Another problem is that the MLP requires a large amount of time for optimization. Further, the overfitting problem is a serious issue. Overfitting is a phenomenon in which the predictor learns only the features from the training dataset and loses generalization performance for other data. The second boom in artificial intelligence ended owing to these reasons. After this period till the current period, the development of neural network was stagnant. The third artificial intelligence boom occurred with the finding of the effectiveness of the deep neural network. This boom started in the 2010s.

3.1.3 Boom in deep learning

The term deep learning refers to the act of learning something in the input data using a deep neural network. Deep neural networks have the ability to achieve more complex activities than shallow networks such as the perceptron. This includes processing images, understanding conversations, and writing sentences. The prosperity of deep learning was achieved owing to various factors. For example, recent increases in available data were one factor. In general, allowing deep neural networks to learn correctly is a difficult task. Deep networks have several parameters that must be optimized; moreover, the larger parameters require various types of data, that is, diverse data. However, the current big data era has enabled us to easily obtain diverse data. In addition to the amount of available data, improvements in the performance of semiconductors, especially graphic processing units (GPUs) and environments such as Compute Unified Device Architecture (CUDA) and CUDA Deep Neural Network library are important factors. As deep neural networks have large parameters that must be optimized, powerful computing devices are required for the optimization process. Improvements in the performance of the GPU has provided powerful computational performance. Finally, improvements in neural network algorithms such as optimized activation functions such as rectified linear unit (ReLU) [9] and optimizers of backpropagation, such as Adam [10], were also one factor. With improvements in neural network algorithms, various related problems have been solved so far.

Owing to the boom in deep learning and artificial intelligence, it is essential that we are both users and developers of artificial intelligence. Consequently, learning both the basic principles of neural networks and the state-of-the-art topics pertaining to neural networks is vital.

3.2 Multilayer perceptron

Neural networks such as MLP can be interpreted as an extension of the simple regression method, as shown in Fig. 1. We will start by providing a short introduction to simple and multiple regression.
3.2.1 Simple, multiple, and extended regressions

The simple regression method can be observed as a type of machine learning. Consider a one-dimensional input vector \((x)\) and one-dimensional target vector \((y)\), whose values must be fit on an optimal line to the data. Further, let \(y\) be the output prediction value of the novel input vector. Using a simple regression analysis, the following equation is determined according to the given data:

\[
y = f(x) = wx + b.
\]  

(3.1)

A conceptual regression line is shown in Fig. 4(a). If this is not a regression problem but a binary classification problem, the target vector can be expressed as follows:

\[
y = \phi(f(x)),
\]  

(3.2)

where \(\phi\) is a step function defined by:

\[
\phi(z) = \begin{cases} 
1 & \text{if } z \geq 0, \\
0 & \text{otherwise}. 
\end{cases}
\]  

(3.3)

As shown in the example in Fig. 1, the number of variables in this simple regression is one. If the number of variables is more than two, the regression method is called a multiple regression method. We can say that the multiple regression method is basically an extension of the simple regression method. As an example, let us consider \(x = (x_1, x_2)^T\) as the input vector and \((y)\) as a target vector. We want to determine an optimal regression plane to predict the value \(y\) of a novel input vector. In this case, the equation of the multiple regression method is expressed as follows:

\[
y = f(x_1, x_2) = w_1x_1 + w_2x_2 + b.
\]  

(3.4)

\[
= (w_1, w_2)(x_1, x_2)^T + b.
\]  

\[
= w^T x + b = f(x),
\]

where \(w = (w_1, w_2)^T\) and \(b\) are the weight parameter and bias term, respectively.

In the multiple regression analysis, we attempt to determine optimal values of \(w\) and \(b\) by repeatedly reading the input data and learning the features of the given data. The conceptual regression plane is shown in Fig. 4(b). As shown, the difference between simple regression and multiple regression is only the number of explanatory variables and their corresponding weight parameters. Although the dimension of the input vector can be increased depending on the given problems, this relationship remains unchanged.

![Fig. 4. Difference between planes of single and multiple regressions. The regression planes are colored in cyan. (a) Regression line by simple regression method in a two-dimensional space: \(y = f(x) = wx + b\). (b) Regression plane by multiple regression method in a three-dimensional space: \(y = f(x) = wx + b\).](image-url)

To generalize the multiple regression method into any arbitrary number of dimensions, we introduce the following expression:

\[
Wx + b = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix},
\]  

(3.4)

\[
= w^T x + b = f(x),
\]
where $W$ is an $m \times n$ matrix and $b$ is an $m \times 1$ matrix ($m$-dimension vector). Using this definition, the regression equation can be expressed as follows:

$$y = f(x) = w^T(Wx + b) + b,$$

where $w$ and $b$ are $m \times 1$ and $1 \times 1$ parameter matrices, respectively. As shown in the equation, regardless of the number of parameters, the dimensions of the input and output vectors do not change as that of the multiple regression. The difference between multiple regression and this extended regression is only in the number of parameters.

### 3.2.2 Principle of neural network

The neural network has almost the same structure as the extended regression method. The only difference between the extended regression and the neural network is that the neural network introduces an "activation function" to the extended regression. Thus, the equation of the regression plane in the neural network is expressed by the following equation:

$$y = f(x) = w^T\phi(Wx + b) + b.$$  

(3.6)

The extended regression method cannot separate nonlinear data. This is true for simple and multiple regression methods. However, real-world data are normally not separated linearly. Activation functions are required for neural networks to mimic nonlinear functions. Activation functions are one of the most important components of neural networks. There are several known activation functions. Some examples include the sigmoid function

$$\phi(u) = \frac{1}{1 + e^{-u}},$$

(3.7)

and ReLU function

$$\phi(u) = \max(0, u).$$

(3.8)

The sigmoid function was the most popular activation function before the development of ReLU, which has overcome a limitation of the sigmoid function and is the most extensively utilized activation function currently. The activation function works as an activator of neural cells in the animal brain, and this function decides whether the previous signals should be communicated to the next neurons or not. Figure 5 shows a conceptual image of the neural network. This figure describes the process of a feedforward neural network, specifically the MLP. The MLP has three layers: the input, middle, and output layers. Here is a brief explanation of this process. The elements of input vectors, $x_i$, are transformed to some element $u_i$. Subsequently, $u_i$ is activated by an activation function $\phi$. Finally, the activated elements $\phi(u_i)$ are finally transformed to the output scalar value $y$. The final output vectors can be the product of an activation function, such as the softmax function defined by

$$\phi(u)_i = \frac{e^{u_i}}{\sum_{j=1}^{N} e^{u_j}}.$$  

(3.9)

where $\phi(u)_i$ denotes the $i$-th element of the output vector of the softmax function with respect to the vector $u$, $u_i$ is the $i$-th element of the vector, and $N$ is the dimension of the vector. This function is frequently used for classification problems. The element of the output vector of the softmax function can be interpreted as a probability, that is,

$$\sum_{i=1}^{N} \phi(u)_i = 1.$$  

(3.10)

The circles in Fig. 5 are called neurons, and the connecting lines between the neurons are weight parameters that must be optimized.

Thus, as shown here, neural networks such as MLP can be interpreted as an extension of the extended regression method. The difference between these methods is the existence of the activation function. Further, we observe that the extended regression technique is an extension of the multiple regression technique, while the multiple regression technique is an extension of the simple regression technique. The regression planes of these methods are listed in Table 2.

### 3.3 Objective function

In the learning process of a neural network, the machine reads the input data repeatedly and attempts to determine the optimal value of parameters in a trial-and-error manner. To optimize the parameter of networks, an objective function is required. Here, the objective function is a function, usually denoted by $E$, for which the parameters are optimized. Conversely, a loss function is a strictly defined scientific term that calculates the difference between the input and target data. Further, a cost function is a function similar to the loss function. This function includes a loss function term and a
There are several loss functions, and a suitable one has to be chosen, depending on the problems. For example, the mean square error (MSE) is generally used as a loss function in regression analysis. The function is expressed as follows:

$$E(\theta) = \frac{1}{2} \sum_{i=1}^{N} \| t_i - y_i \|^2,$$

where $E$ is the loss function, $\theta$ indicates the parameters to be optimized, $N$ is the number of instances, $t_i$ is the $i$-th target vector, $y_i$ is the $i$-th output vector from the predictor, and $\| *, * \|$ is the $L^2$-norm. In this equation, note that if the difference between the true value and the predicted value is large, the value of the loss function will be large, and vice versa. In addition to MSE, cross-entropy (CE) is utilized as a loss function. The CE is defined by the following expression:

$$E(\theta) = -\sum_{i=1}^{N} t_i^T \log y_i,$$

where $E$ is the loss function, $\theta$ indicates the parameters that must be optimized, $N$ indicates the number of instances, $t_i$ is the $i$-th target vector, and $y_i$ is the $i$-th output vector. Similar to the MSE, if the difference between the true and predicted distributions is large, the value of the loss function will be large, and vice versa. The optimization method of networks attempts to minimize this difference to the smallest possible value. Thus, these two functions are the most utilized loss functions; moreover, MSE is primarily used for regression problems and CE with softmax function is primarily used for classification problems.

### 3.4 Backpropagation

Generally, neural networks have multiple parameters, whose gradient, $\nabla E(\theta)$, must be calculated. If the gradient can be calculated, we can apply an optimization method such as the gradient descent method, as shown in the previous section. To calculate the gradient, a backpropagation method is generally utilized.

#### 3.4.1 Backpropagation on a two-layer network

In this section, we consider a multilayer network to understand the principle of backpropagation. For simplicity, we consider a network as illustrated in Fig. 6, which is a two-layer network consisting of $n$- and $(n+1)$-th layers extracted.
from a multilayer network. In this figure, the network has two layers. The first layer is the $n$-th layer, which is an input layer. The second layer is the $(n+1)$-th layer, which is an output layer. In the first layer, there are $I$ neurons, while the layer after it has $J$ neurons. Inside the neurons, $s^n_i$ is the inactivated value of the $i$-th neuron on the $n$-th layer, and $u^n_i$ is the activated $s^n_i$ by the activation function $\phi$, that is,

$$u^n_i = \phi(s^n_i).$$

(3.13)

The neurons on the first and second layers are connected to each other by the parameter matrix $w^{n,n+1}$ of size $I \times J$. The line represents the element of the weight parameter matrix. Thus, $s^{n+1}_j$ can be calculated using the following expression:

$$s^{n+1}_j = \sum_{i=1}^{I} u^n_i w^{n,n+1}_{ij}.$$ 

(3.14)

Note that the inactivated value in the second layer is calculated by the summation of all products between neurons in the previous layer and its corresponding weight parameters. Normally, the initial weight parameters, in this case $w^{n,n+1}$, are randomly selected. Using the random value and input vectors in the dataset, we can perform forward calculation. Thus, the second layer produces the final vector. Finally, by comparing the output and target vectors, the weight parameters are updated iteratively. As mentioned in the previous section, to update the parameters, the gradient descent method is used. This method uses the gradient of the cost function $E$ with respect to the parameter $w^{n,n+1}$. We denote the matrix containing the gradient as $Z$, where

$$Z_{ij} = \frac{\partial E}{\partial u^{n+1}_j}.$$ 

(3.15)

![Fig. 6. Two-layer neural network. The circles represent neurons and the lines between the circles are weight parameters. The weight parameter to be calculated is colored in magenta.](image)

In the two-layer network, the cost function is not a function of the variable $u^{n,n+1}_j$ but a function of the variable $u^{n+1}_j$. Thus, to calculate $Z_{ij}$, we must use the chain rule of derivation. Using (3.13) and (3.14), the value of $Z_{ij}$ can be calculated as follows:

$$Z_{ij} = \frac{\partial E}{\partial u^{n+1}_j} \frac{\partial u^{n+1}_j}{\partial s^{n+1}_j} \frac{\partial s^{n+1}_j}{\partial u^{n,n+1}_j}$$

$$= \frac{\partial E}{\partial u^{n+1}_j} \frac{\partial}{\partial s^{n+1}_j} \phi(s^{n+1}_j) \frac{\partial}{\partial u^{n,n+1}_j} \left( \sum_{i=1}^{I} u^n_i w^{n,n+1}_{ij} \right)$$

$$= \frac{\partial E}{\partial u^{n+1}_j} \phi'(s^{n+1}_j) u^n_i.$$ 

(3.16)

Note that the resultant terms can be calculated from output values in a learning process; and thus, we can obtain the exact value of $Z_{ij}$. Further, in the final result, we have a derivative of the activation function $\phi$. This requires the activation function to be differentiable.

### 3.4.2 Backpropagation on three-or-more-layered network

Here, we calculate $Z_{ij}$ in the three networks shown in Fig. 7, where the $(n+2)$-th layer is the output layer. In this case, instead of $u^{n+1}$, the cost function is $u^{n+2}$. This requires another method of computing $Z_{ij}$.
To work with the three-layer neural network, we require the following extra equation:

$$\frac{\partial E}{\partial u_{n+1}^{j}} = \sum_{t=1}^{K} \frac{\partial E}{\partial u_{n+2}^{jt}} \frac{\partial u_{n+2}^{jt}}{\partial u_{n+1}^{j}}.$$  \hspace{1cm} \text{(3.17)}$$

Using this and a computation similar to that of the two-layer network, the value of $Z_{ij}$ in the three-layer network can be calculated as follows:

$$Z_{ij} = \frac{\partial E}{\partial u_{n+1}^{j}} = \sum_{s=1}^{L} \frac{\partial E}{\partial u_{n+2}^{is}} \phi'(s_{j}^{n+2})u_{n+1,n+2}^{i} \phi'(s_{j}^{n+1})u_{n}^{j}.$$  \hspace{1cm} \text{(3.18)}$$

Further, we consider a four-layer network, as shown in Fig. 8, where the $(n+3)$-th layer is the output layer.

$$Z_{ij} = \frac{\partial E}{\partial u_{n+3}^{ij}} = \sum_{s=1}^{L} \sum_{t=1}^{K} \frac{\partial E}{\partial u_{n+4}^{st}} \phi'(s_{j}^{n+3})u_{n+2,n+3}^{i} \phi'(s_{j}^{n+2})u_{n+1,n+2}^{i} \phi'(s_{j}^{n+1})u_{n}^{j}.$$  \hspace{1cm} \text{(3.19)}$$

Again, note that all terms in the equation are calculated through the forward calculation. From these results, the exact value of $Z_{ij}$ can be obtained. We can observe from the equation of the backpropagation that to calculate the derivative of the cost function with respect to the objective parameter $u_{n+1}^{j}$, the backpropagation method requires the exact values of all cyan-colored parameters in Fig. 8. These values must be memorized in a learning process so that this
parameter can be updated. As the number of parameters increases based on the number of layers, a deeper neural network will require a large amount of memory. Finally, by using $Z_{ij}$ calculated by the backpropagation and gradient descent method, the parameter $w_{ij}^{n,n+1}$ is updated according to the following update rule:

$$w_{ij}^{n,n+1} = w_{ij}^{n,n+1} - \alpha \frac{\partial E}{\partial w_{ij}^{n,n+1}},$$

(3.20)

where $t$ is the learning timestep and $\alpha$ is the learning rate.

Basically, the concept of backpropagation is a chain rule of derivation of the cost function. As presented in this section, the calculation of the backpropagation proceeds from the output layer to the input layer; thus, the method is called backpropagation.

### 3.4.3 Vanishing gradient problem

The backpropagation method is a powerful method for learning the parameters in the neural network. However, it involves the risk of a vanishing gradient. Here, the vanishing gradient implies that the gradient required for the gradient descent method vanishes during the learning process. As shown in the update rule of the gradient descent method (3.20), when the gradient value is zero, the update of parameters will stop even though the parameter is not sufficiently optimized. Thus, the vanishing gradient problem is one of the most serious problems related to neural network inference. One of the causes of the vanishing gradient phenomenon is the activation function terms in the backpropagation algorithm and the choice of the activation function. For example, as shown in (3.19), for the four-layer MLP, there are three derivatives of the activation function $\phi(s)$. This implies that we are required to at least calculate the cube of the derivative of the activation function in the backpropagation process. A four-layer neural network is generally not considered as a deep network. However, regardless of the shallowness of the network, we must calculate the activation function to the power of three. The calculation of power is the primary cause of the vanishing gradient problem.

As an example, we will demonstrate this problem by using the sigmoid function as the activation function. This function, shown in (3.7), was often used as the activation function for a general neural network in the past. The derivative of the function is calculated as follows:

$$\phi'(x) = \phi(x)(1 - \phi(x)).$$

(3.21)

The graph in Fig. 9 shows two curves: the actual output values of the derivative of the activation function and its cube. In the graph, the maximum value of $\phi'(x)$ is 0.25, while the maximum value of $[\phi(x)]^3$ is 0.015625. This shows that as the power of the derivative of the activation functions is repeated or as the number of layers increases, the value of the gradient decreases. This will subsequently make it close to zero as the value of $\phi(x)$ reaches closer to zero when $n$ is large. This is the true nature of the vanishing gradient problem.

### 3.5 Rectified linear unit (ReLU)

#### 3.5.1 Principle of ReLU

While the vanishing gradient problem has been causing a serious problem in neural network studies for a long time, there is a solution to this problem. It is well known that an effective way to overcome this problem is by changing the activation function, such as the sigmoid function, to another function, such as ReLU [9]. Mathematically, this function can be defined as $y = \max(0, x)$, as shown in Fig. 10.

ReLU is the most commonly used activation function in neural networks, especially in CNNs, which we describe in a subsequent subsection. This function is one of the better choices when people are unsure on which activation function to use in the network. ReLU is linear for all positive values and zero for all negative values. This implies that the
The computation cost is low as there is no complicated calculation. Therefore, the model requires less time to train or run. This function also converges faster and does not have the vanishing gradient problem, which is a limitation of other activation functions such as sigmoid. The drawback of being zero for all negative values is a problem called “dying ReLU.” A ReLU neuron is “dead” if the value persists on the negative side and the output is always zero. Because the slope of the ReLU in the negative range is also zero, once a neuron becomes negative, its recovery ability is diminished. Such neurons do not perform any role in discriminating the input and are essentially useless. Over time, a large part of the network may end up doing nothing. This dying problem is likely to occur when the learning rate is too high or there is a large negative bias. To fix the dying problem, there are certain variants of ReLU, such as leaky ReLU, parametric ReLU (PReLU), and exponential linear unit (ELU) or scaled ELU (SELU).

3.5.2 Leaky ReLU and parametric ReLU

Note that in ReLU, the function has zero values for all negative values. Leaky ReLU has a small slope for negative values, instead of it being completely zero. For example, leaky ReLU may have $y = 0.1x$ when $x < 0$. Instead of having a predetermined slope such as 0.1, PReLU is a type of leaky ReLU with the form $y = ax$ when $x < 0$, where the value of $a$ is determined by the neural network itself (Fig. 11).

Leaky ReLU has two benefits: it fixes the “dying ReLU” problem as it does not have zero-slope parts and speeds up training. There is evidence that having a “mean activation” close to zero results in a faster training. Unlike ReLU, leaky ReLU is more “balanced,” and may therefore learn faster. It is good to remember that the result is not always consistent. This function is not always better than plain ReLU and should be considered only as an alternative.

3.5.3 Exponential linear unit (ELU), scaled ELU

Similar to the leaky ReLU, ELU has a small slope for negative values. Instead of a straight line, it uses a log curve as shown in Fig. 12.

ELU is designed to combine the benefits of ReLU and leaky ReLU, that is, while evading the dying ReLU problem, it saturates for large negative values, allowing them to be essentially inactive. ELU was first proposed in [11]. It is sometimes called SELU because of the constant factor $a$. The plot of this function is shown in Fig. 12.
3.5.4 Concatenated ReLU

The concatenated ReLU (CReLU) has two outputs, a ReLU and negative ReLU, concatenated together. In other words, for positive $x$, it produces $\frac{1}{2}x$; $0$, and for negative $x$, it produces $0$; $\frac{1}{2}x$. Because it has two outputs, CReLU doubles the output dimension. CReLU was first proposed in [12]. There is also a function called ReLU-6 in certain libraries, which is ReLU capped at a maximum size of six (Fig. 13). It was first used in [13] for the CIFAR-10 dataset from the Canadian Institute For Advanced Research, and the value of six was an arbitrary choice that worked well.

3.6 Convolutional neural network

3.6.1 Impact on object recognition contest

The success of object recognition using CNNs has had a huge impact on artificial intelligence researchers. The first success of CNN in the context of an image recognition competition was at the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The ILSVRC evaluates algorithms for object detection and image classification on a large scale. In this contest, researchers compare their developing methods pertaining to object detection of various types of images in the large image database, ImageNet, and measure the progress of computer vision research. Figure 14 illustrates the final results of the contest for each year from 2010 to 2017. In the bar graph, a smaller value
indicates that the prediction performance is better. As shown in the figure, the final result of the contest was improved radically in 2012. Conversely, the performance in 2010 and 2011 appeared to reach a plateau; thus, many artificial intelligence researchers were surprised at this improvement. The algorithm used by the winner in the contest was CNN [14]. Since then, CNN has been widely used and further improvement has been achieved in the past years.

3.6.2 Overview of convolutional neural network

In the recent boom in deep learning, the development of CNNs can be considered as one of the most important events. CNN is a more sophisticated neural network than MLP, as illustrated in the previous section. When CNNs are considered, it can be said that “machines have eyes,” as it is primarily utilized for image analysis, such as object recognition. In this section, we consider the two images shown in Fig. 15(a). The image consists of 12 rows and 12 columns (matrix); hence, there are a total of 144 squares in an image. Both images include the numeral 1, and a task in the section is classifying these two images as the same class.

In this problem, one of the challenges for machines is to identify a similar pattern defined by the contrast between the background and the number from both images. In addition to pattern recognition, another challenge for machines is to neglect the gap in the location of the number. The number in the first image is located almost at the center of the image, while the number in the other image is located on the left side. If one wants to learn these images using the MLP, the $12 \times 12$ image matrix is flattened to a 144-dimension vector, as shown in Fig. 15(b), where the left and right images in Fig. 15(a) correspond to the upper and lower vectors, respectively. This new vector is used as input data to the MLP because MLP can only accept vector-type data as the input data. However, in this case, the flattened vectors are different from each other, even though these two images include the same number, i.e., 1. Recognizing this is a difficult task for MLP because there is a gap between the two images. Even though the MLP can learn these types of images accurately, this method is not good at learning these data efficiently.

3.6.3 Distinctive features of convolutional neural network

To overcome the difficulties related to image processing, such as pattern recognition in images and the existence of a gap, CNN has two distinctive features: convolution calculation and pooling processes. First, the convolution emphasizes the contrast between the original images. This makes it easier for the machine to detect similar patterns in images. To perform this operation, a filter, which is also called a kernel, is utilized. Figure 16 shows a conceptual image of the filter.

![Conceptual image of a filter utilized in a convolution process.](image-url)
As shown, the filter is a matrix, and its elements consist of numerical values. The initial value of the filter is normally determined at random. Using the filter and original input image, convolution is conducted using the following equation:

\[ u_{ij} = \sum_{p=1}^{F} \sum_{q=1}^{F} x_{i+p,j+q} f_{pq}, \]

(3.22)

where \( u_{ij} \) is the value of the filtered cell \((i,j)\), \( f_{pq} \) is the value of cells of the filter, \( F \) is the size of the filter, and \( x_{ij} \) is the value of the cell \((i,j)\) of the original image. As shown in Fig. 15, the filter itself has a contrast. In this case, according to (3.22), if the contrast is similar to that of a portion of the original image, the cell value of a filtered image, which is called a map, is large and vice versa. Specifically, using the filtering process, a portion that is similar to the filter is emphasized and a portion that is not similar to the filter is blurred. By this mechanism, the convolution process can extract distinctive contrast patterns from images. Figure 17 shows the calculation of the convolution process.

First, the convolution calculation is performed on the upper panel on the image on the left. Next, the filter is shifted to read the other regions. This action is called a stride. Although the filter is shifted by a column in the stride, the stride size can be changed; thus, it is a hyperparameter. The stride size is generally decided in a trial-and-error manner during the learning process. After the convolution process, the resultant image, which is called a map, is obtained as shown in the lower panel of Fig. 17. Although the map in the conceptual image still holds the symbolic pattern “1,” a resultant map by actual convolution calculation will no longer hold a clear pattern, that is, a human normally cannot recognize it. Although the size of the original image and the map in Fig. 17 are the same, the size of the map becomes smaller than the original image. To prevent this, zero-padding method, where the outer portion of the original image matrix is filled with zeroes, is generally utilized as a heuristic technique.

Figure 18 illustrates the process of the pooling method, which is able to cancel the inferior effect caused by the existence of a gap in the input images. The upper red images show the maps generated by the convolution process. The pooling calculation is processed from the image on the left to that on the right. The green range that is overlapped on the map is the range of pooling calculation. From the range, a number is extracted based on some criteria, which includes extraction of the maximum value in the range (max pooling) or mean value of the values in the range (average pooling). The numbers extracted by the process are used for components of a novel matrix, as shown in the image in the bottom panel of Fig. 18. The operation is repeated for all regions of the map and finally, a matrix is obtained. As the pooling process sums up the information in a range, it can absorb the inferior effect of the existence of gaps in images.

Thus, the convolution and pooling processes are the most distinctive features of the CNN. The CNN confers “eyes” to artificial intelligence because it has a mechanism of detecting patterns in images and absorbing the gap of images through the features. The eye is an important concept for artificial intelligence. For example, artificial intelligence for an automatic driving system must recognize road signs correctly. Disaster rescue robots must possess the ability to find humans and animals from collapsed buildings, and artificial intelligence for disease diagnosis has to properly understand diagnostic images. The fact that artificial intelligence obtains eyes was a significantly revolutionary event for artificial intelligence researchers; thus, the development of CNN has been directly linked to the third boom of artificial intelligence.
3.7 Recurrent neural network

3.7.1 Various recurrent neural networks and context data

The RNN is a general term of a neural network that has recurrent loop structures in its architecture. Several RNNs have been developed, including the Elman network (EN), echo state network (ESN), Boltzmann machine, long short-term memory (LSTM), gated recurrent unit (GRU). The most basic RNN is EN, which has only one simple recurrent loop in its architecture. The ESN has a unique and unusual feature. It does not change the values of the weight parameter from the initial random values. Instead, it changes the connections between neurons in the learning process. LSTM and GRU are popular architectures. As one of the prominent features of the RNN, several architectures such as LSTM and GRU memorize the context of input data using the recurrent loop. Here, the word context refers to the relationship between information in sequential data. The types of RNN are important for understanding the generally used artificial intelligence technologies because there are various types of context in the real-world data. For example, it includes text data written in natural languages, biological sequence information such as DNA and proteins, the notes of music, fluctuations in stock price depending on time. The principle of reading and memorizing the context is relatively simple. Although RNN reads a context using the recurrent loop as mentioned above, the method of actual implementation is as shown in Fig. 19.

![Fig. 18. Conceptual image of computation of the pooling process. The upper red images illustrate the map generated by the convolution process. The green range overlapped on the map is the range of the pooling calculation. The small image on the bottom panel is the resultant matrix after the pooling process.](image)

Fig. 19. Expanding the structure of the recurrent neural network (RNN) to feedforward neural networks. In the equation, the left side includes the original RNN with the loop structure and the right side includes expanded feedforward neural networks. The number of feedforward neural networks corresponds to the dimension of input sequential data, \( x \). On the right side, the horizontal arrows indicate the transmission of the memory data to the next network of the timestep.

Here, we have sequential input data, \( x \), and its corresponding sequential output data is \( y \). The \( t \)-th elements of the sequence data are \( x_t \) and \( y_t \), which can be either a scalar value or a vector. To read the type of sequential data, the RNN expands the loop structure to normal feedforward neural networks for the length of the data and repeatedly learns elements of the input sequence data one by one and outputs its corresponding data. At this time, a vector, whose size is the same as the output, is maintained as a memory throughout the recurrent computation, and it is also used as additional input data for the middle layer of the neural network of the next timestep. The memory vector memorizes the previous situation of data, and thus the neural network of the next timestep can utilize the previous information, using which RNN can read a context of input data.

3.7.2 Long short-term memory and gated recurrent unit

As described in the previous subsection, LSTM and GRU are some of the representative methods of RNN. LSTM is
one of the RNNs with the highest performance, which was originally developed in 1997 [15]. Unlike the simple RNN such as EN, LSTM has complex links between parameters in its architecture, called the LSTM cell. The complex links of parameters in the LSTM are defined by the following six equations:

\[
\begin{align*}
v_1 &= \tau(W_{1a}u + W_{1b}h_{t-1} + b_1) \\
v_2 &= \sigma(W_{2a}u + W_{2b}h_{t-1} + b_2) \\
v_3 &= \sigma(W_{3a}u + W_{3b}h_{t-1} + b_3) \\
v_4 &= \sigma(W_{4a}u + W_{4b}h_{t-1} + b_4) \\
v_5 &= v_1 \odot v_2 + v_3 \odot s_{t-1} \\
h &= v_4 \odot \tau(s_t)
\end{align*}
\]

where \( u \) is an input vector to the LSTM cell, \( W \)s are weight parameters, \( b \)s are bias parameters, \( \sigma \) is a sigmoid function, \( \tau \) is a hyperbolic tangent, \( v \)s are vectors that are temporally generated in the LSTM cell, \( s \) is a vector of the constant error carousel at timestep \( t \), which is a mechanism to prevent the gradient from vanishing or exploding, \( h \) is an output vector of timestep \( t \), which is saved in the LSTM cell and used for the calculation of the next timestep, and \( \odot \) is an operator of the Hadamard product, which is an element-wise product of two matrices. Here, the LSTM is not the original model developed in 1997, but is one of the most popular implementations, improved in 1999 [16]. The LSTM diagram is shown in Fig. 20, which includes links between parameters in the LSTM cell.

Fig. 20. Diagram of a long short-term memory cell. Here, \( \times, c, \) and \( z \) indicate an operator of the matrix product, an operator for decomposing a vector to four vectors of the same size, and the output vector, which is the same as \( h \).

The steps in this diagram can be described as follows:

1. The matrix product between the input vector to the LSTM cell and weight parameter is calculated, where all weight and bias parameters used in (3.23)–(3.26) are concatenated in tandem to matrices \( W_a, W_b, \) and \( b \).
2. The bias parameter and product between the previous output vector \( (h) \) and other weight parameter to the first product is added.
3. The resulting vector is decomposed into four same-sized vectors by the operator \( c \); then, the vectors are processed by the hyperbolic tangent and sigmoid function.
4. The sigmoid functions work as a gate similar to the human brain. The range of output of the sigmoid function is \( (0, 1) \). If the output value is almost zero, the gate works to stop the information flow and vice versa. For example, the first sigmoid function from the top one is an input gate, which determines whether LSTM memorizes the input data by calculating the Hadamard product between the gated vector \( v_2 \) and the standardized input vector \( v_1 \), as shown in (3.27), as \( v_1 \odot v_2 \).
5. The other two sigmoid functions work as a type of gate, where the second sigmoid function is a forget gate and the third function is an output gate.
6. The vectors \( h \) and \( s \) indicate memories and in the memory, all of the past information of input context data is saved. This mechanism enables LSTM to memorize the previous output to learn the context of input sequential data.

The LSTM demonstrates superior performance and the memory power of LSTM can theoretically last indefinitely; in comparison, the EN can last only for 20 or 30 timesteps.

The GRU is another famous RNN capable of learning the context of input data. This architecture was originally developed in 2014 [17] and is a comparably new method. The links between parameters in the GRU are defined by the following expressions:

\[
\begin{align*}
v_1 &= \sigma(W_{1a}u + W_{1b}h_{t-1} + b_1) \\
v_2 &= \sigma(W_{2a}u + W_{2b}h_{t-1} + b_2)
\end{align*}
\]
\[ v_3 = \tau(W_{3a}u + W_{3b}(v_2 \odot h_{t-1})) + b_3 \]  
(3.31)

\[ h_t = (1 - v_1) \odot h_{t-1} + v_1 \odot v_3, \]  
(3.32)

where almost all notations are the same as that of LSTM. Here, \( \mathbf{I} \) is an all-one vector. The conceptual diagram of the GRU is shown in Fig. 21.

Fig. 21. Diagram of gated recurrent unit. Here, \( \mathbf{I} \)-stands for element-wise subtraction from all-one matrix.

As shown in the equations and diagram, GRU is considerably similar to LSTM, which has gate structures in its architecture. In the GRU, the sigmoid functions in (3.29) and (3.30) work as gates. One of the prominent differences with LSTM is the number of parameters. The LSTM has twelve parameters, including eight weight parameters and four bias parameters. Conversely, GRU has only nine parameters, including six weight parameters and three bias parameters. The number of parameters of the GRU is smaller than that of the LSTM, and thus the GRU sometimes has better convergent performance than LSTM.

### 3.7.3 Application of recurrent neural network

RNNs can be applied in various research fields such as natural language processing, bioinformatics, and acoustical engineering. For example, the encoder-decoder model, which is one of the sequences of the sequence model, is one of the most famous applications of RNN in the field of natural language processing. The encoder-decoder model, which was originally published in 2014 [18], can accept sequence data as an input and output a sequence, as shown in Fig. 22, where “ABC” is an input sequence (sentence), “WXYZ” is an output sequence, and EOS indicates the end-of-sentence.

Fig. 22. Encoder-decoder model, as shown in a study published in 2014 [18]. “ABC” is an input sentence and “WXYZ” is an output sentence. The white boxes represent layers of RNN.

Because there are several types of sequence data in the real world, this architecture can be utilized for various purposes. For example, using the encoder-decoder model, machines can learn a sentence and output a sentence. As an example, artificial intelligence can accept an input conversation and return a response to it. Conversation with other people is one of the most important abilities of human beings; thus, RNN, which has the ability to achieve this, is a fundamental technology to create artificial general intelligence. Thus, an RNN capable of reading contexts could be one of the most promising architectures.

### 3.8 Summary

Neural networks have performed a significant part in the recent boom in artificial intelligence as one of the most popular learning methods. In the inference process of a neural network, the backpropagation method can be used.
Backpropagation is a method used to calculate a derivative of a cost function based on a simple chain rule of derivation for the parameters to be learned. There are various types of neural networks, including MLP, CNN, and RNN. The MLP is the most basic neural network architecture. CNNs, which started the boom in artificial intelligence, are applied primarily in image processing studies. RNN is one of the most promising architectures for achieving artificial general intelligence. In addition to these neural networks, a GAN is one of the most popular and promising architectures. As the progress in neural network development is rapid, continuous update of knowledge with the latest information is required.

4. Kernel Method and Support Vector Machine

4.1 Overview

The kernel method is a method that maps input data into a high-dimensional feature space with the purpose of elucidating nonlinear relationships of input data. This data is not elucidated in the original low-dimensional feature space. Thus, the kernel method is a method to handle nonlinear data that can be separated in only a high-dimensional feature space. To map the data into a high-dimensional space, the kernel methods do not explicitly assume a type of high-dimensional space, but it explicitly handles the data by omitting heavy calculations. This method of processing data using the kernel method is called kernel trick, which is performed by kernel functions. The various machine learning methods using this kernel trick are also classified as kernel methods. With the kernel methods, we can easily modify a linear machine learning method to a nonlinear learning method. Because almost all data in the real world consist of nonlinear data, this method is considerably useful in modifying existing analysis methods with kernel functions. For example, normal and linear SVMs are transformed to nonlinear kernel SVMs by introducing the kernel method. SVM is a machine learning method that uses both linear classification and regression methods. The SVMs for classification and regression are called support vector classification (SVC) and support vector regression, respectively. Although SVM is a nonlinear analysis method, it is normally used with the kernel method to handle nonlinear data. The SVM was the most popular machine learning method in the past until it was replaced with deep learning methods. Here, we compare the SVM with the neural network.

1. There is a difference in expressing the nonlinear relationship between the two methods. Although neural networks express nonlinear relationships through activation functions, SVM expresses it through kernel functions.
2. The computation speed of SVM is not superior to that of the neural network. The SVM requires a time complexity of $O(n^2)$ for the inference, where $n$ is the number of instances in the input data. This is one of the biggest disadvantages of SVM and one of the reasons why SVM was replaced with deep learning.
3. The convergence performance of the SVM is better than that of the neural network. The optimization problem solved in SVM calculation is a convex optimization problem, and basically, it is not difficult to reach an optimal value in the problem.
4. The expression power of SVM is not superior to that of neural networks. For example, deep learning methods can explicitly read image data using CNN and can understand the context of input data using RNN. Further, with neural networks, the size or dimension of the output vector is easily and freely designed. As artificial general intelligence has to handle complicated real-world problems, this expression of power is a strong advantage of neural networks.

As previously discussed, different machine learning methods have various features; thus, analyzers have to choose the proper method depending on problems. The merit of using SVM is its good convergence performance, a feature that can modify a linear analysis method to a nonlinear analysis method, and demonstrates high classification and regression performance. In this section, we introduce the principle of the kernel method and its application to kernel SVM.

4.2 Kernel method

4.2.1 Linear separation

In this subsection, we introduce the principles of the kernel method by considering the problem of separating instances in data as an example. Figure 23(a) includes two types of data belonging to different classes, represented by black and white filled circles on a two-dimensional plane with axes $x_1$ and $x_2$. The objective is to identify a separator of the black and white instances, as shown in Fig. 23(b).

As shown, the circles are separated by a linear function. This type of linear problem can be solved using normal machine learning methods of linear discriminant. Next, we consider the separation problem shown in Fig. 24(a). In the plane, the instances are scattered similar to a circular case; thus, it is impossible to separate the data using linear functions. In this case, if we use a nonlinear function (a dashed circle), we can separate the two types of data, as shown in Fig. 24(b). These problems can also be solved without using nonlinear functions. This is performed by plotting the data on a higher dimensional space where the nonlinear data are expected to be separated linearly.

To perform this operation, the data are mapped to a higher dimensional space by adding an additional axis $\phi(x_1, x_2)$ to the original two-dimensional space, as shown in Fig. 24(c). For example, this function $\phi$ can be of the form $x_1x_2, x_1^2, x_2^2, x_1^2x_2$, or other equations. These data can then be separated by linear functions, as shown in Fig. 24(d).
In the example, the two-dimensional space is extended to a three-dimensional space. However, we can increase the number of dimensions arbitrarily. The original axes consist of $x_1$ and $x_2$, and in this case, the following extension can be considered:

$$(x_1, x_2, x_1^2, x_2^2, x_1x_2, \ldots).$$

(4.1)

There are numerous ways to increase the dimensionality of data. If we extend the coordinate to the $d$-dimensional space with the nonlinear function $\phi$, the feature vector can be expressed by the following expression:

$$(\phi_1(x_1, x_2), \phi_2(x_1, x_2), \ldots, \phi_d(x_1, x_2)).$$

(4.2)

Although the dimension of the original data is extended to higher dimensions by the extension of axes, the values in the axes can be expressed by the following vector:

$$\phi(x) = (\phi_1(x), \phi_2(x), \ldots, \phi_d(x)),$$

(4.3)

where $\phi(x)$ is the extended vector value projected on the $d$-dimensional space from the original instance vector $x$, where $\phi$ is an extension of the original axes, as in (4.1). In this case, the separator $f(x)$ is expressed by the following expression:

$$f(x) = w^T \phi(x) = \sum_{m=1}^{d} w_m \phi_m(x).$$

(4.4)

In the example given in this section, the separator corresponds to the plane in Fig. 24(d). By this process, the original data are mapped into a higher $d$-dimensional space and the transformed data can be analyzed using a linear model as represented by (4.4).

### 4.2.2 Principles of the kernel method

Next, we introduce the principles of the kernel method. As illustrated in the previous paragraph, the linear separator can be determined in some $d$-dimensional space. However, we do not know the size of $d$, and this number could be significantly large. Determining the right size of $d$ requires a large computation time, and sometimes, it can be impossible to determine. The kernel method can be used to address this problem. The kernel method can reduce the time complexity of the computation to the size of instances of dataset $n$, instead of the dimension $d$. In the previous subsection, we obtained the separator $f(x)$ in (4.4) with the time complexity of $O(n)$. First, we introduce the representer theorem.

**Theorem 4.1** (Representer theorem). Let the loss function $L$ be a function of $w^T \phi(x_i)$, where $x_i$ is the $i$-th input vector data. Moreover, let the regularization term be $\lambda w^T w$. The optimal solution for this is a linear combination of $\phi(x_i)$.

In this theorem, regularization is the process of adding information to solve an ill-posed problem, which is a problem where the final result is sensitively affected by changes in the input data.

The cost function is expressed by the loss function and regularization term as follows:

$$J(w) = L(w^T \phi(x_i)) + \lambda w^T w,$$

(4.5)

where $J$ is the cost function to be optimized and $\lambda$ is a hyperparameter to determine the effect of regularization. According to the representer theorem, the optimal solution is expressed by the following expression:
\[ w_{\text{constant}} = \frac{1}{\sum_{i=1}^{n} C_{11} x_i}. \]

Thus, the separator \( f(x) \) is expressed by the following expression:

\[ f(x) = \sum_{i=1}^{n} \alpha_i \phi(x_i) \phi(x). \]

The most distinct feature of the representer theorem is that the separator is expressed by the number of instances in the dataset, \( n \), rather than the dimension of the higher dimensional feature space, \( d \). If the value of \( n \) is smaller than \( d \), the representer theorem provides values with more efficient computation time. Next, we define the kernel function \( k(a, b) \) as follows:

\[ k(a, b) = \phi(a)^T \phi(b) = \sum_{m=1}^{d} \phi_m(a) \phi_m(b). \]

where \( \phi \) is the vector function used to embed the input data into the high-dimensional feature space. The function \( k \) is used to generate an inner product of the transformed data in the high-dimensional space. Using (4.7) and (4.8), the separator \( f(x) \) can be expressed as follows:

\[ f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x). \]

With the kernel function, the calculation of the summation of inner products in \( d \) times is no longer required. Moreover, knowing the form of the mapping function \( \phi \) is also not required. Instead, the following types of kernel functions are utilized:

\[ k(a, b) = \exp(-\gamma \|a - b\|^2), \]
where $\gamma$ is a hyperparameter determined in the learning process. This enables the omission of the calculation of the mapping function. This kernel function is called radial basis function (RBF) kernel or Gaussian kernel. The RBF kernel is the most popular type and this kernel should be the first choice. As shown, the kernel function $k$ uses the inner product $\langle \Phi(a), \Phi(b) \rangle$ in the feature space. Thus, the kernel function must correspond to an inner product $\langle \Phi(a), \Phi(b) \rangle$ in a feature space. Conversely, if $k$ only corresponds to an inner product $\langle \Phi(a), \Phi(b) \rangle$, it implies that $k$ is a kernel function. Mercer’s theorem explains why the kernel function can be used to achieve a kernel trick.

Before introducing Mercer’s theorem, we define a positive semidefinite kernel. For $n$ points $(x_1, x_2, \ldots, x_n)$ in a feature space, a function that satisfies the following conditions is a positive semidefinite kernel:

1. The functions are symmetrical to each other:
   \[
   k(x_i, x_j) = k(x_j, x_i).
   \] (4.12)

2. For $(a_1, \ldots, a_n) \in \mathbb{R}^n$, the following expression is satisfied:
   \[
   \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j k(x_i, x_j) \geq 0.
   \] (4.13)

With a positive semidefinite kernel, Mercer’s theorem is expressed as subsequently presented.

**Theorem 4.2** (Mercer’s theorem). If a function $k$ is a positive semidefinite kernel, then
\[
k(x_i, x_j) = \sum_{m=1}^{\infty} \lambda_m \phi_m(x_i) \phi_m(x_j), \quad \lambda_m \geq 0.
\] (4.14)

If a function of $x_i$ and $x_j$ has a positive semidefinite solution, it can be used as a kernel function to express the summation of the inner product on an infinite feature space. The theorem indicates that the inner products of the input vectors in the feature space correspond to the kernel function.

In conclusion, to calculate the nonlinear separator using the kernel method, we introduce a $d$-dimensional feature space to handle nonlinear relationships in the dataset, where the mapping function to the space is $\phi$. Next, using the representer theorem, the time complexity of the computation of the separator can be changed from $O(n)$ to $O(d)$. The equation of the separator is expressed by the kernel function without using the mapping function. Then, the kernel trick can be validated using Mercer’s theorem.

### 4.3 Kernel support vector machine

In this section, we introduce the principle of SVC. SVC is a linear discriminator and can be classified as an SVM. For example, let us consider $n$ instances of the dataset $(x_i, t_i), i = 1, 2, \ldots, n$, where $x_i \in \mathbb{R}^d$ and $t_i \in [-1, 1]$. Here, $x_i$ is a $d$-dimensional input vector, and $t_i$ is a target vector. The instances in the dataset are plotted on the two-dimensional plane in Fig. 25(a), where each $t_i$ is expressed by a circle if its value is 1 and a square if its value is −1. The objective of the analysis with SVC is to determine a separator that separates the circles and squares.

![Fig. 25](image)

(a) Instances in a two-dimensional space where each $t_i$ is expressed by a circle if its value is 1 and a square if its value is −1. (b) The separator function $f$ is represented by a solid line. The dashed lines are marginal lines. The open circles and squares are support vectors.

As shown in Fig. 25(b), the equation of the separator is as follow:
\[
f(x) = \tilde{w}^T x + \tilde{b},
\] (4.15)

where $\tilde{w}$ and $\tilde{b}$ are the weight and bias parameters, respectively. These parameters are optimized by the SVC process.

In addition to the separator, SVC also calculates marginal lines. This is represented by the dashed lines in Fig. 25(b). The region defined by this marginal line is called the margin. SVC attempts to maximize the size of the margin to classify instances with as large a separation as possible. The margin is the distance from the separator to its closest
instances. These instances are called support vectors. In Fig. 25(b), the support vectors are the open squares and circles that lie on the line of \( f(x) = M \) or \( f(x) = -M \). As mentioned, SVC attempts to maximize the distance from the separator to instances. These distances can be expressed using the following expression:

\[
D = \frac{|\mathbf{w}^T \mathbf{x}_i + \mathbf{b}|}{\|\mathbf{w}\|},
\]

where \( \mathbf{x}_i \) is a support vector and satisfies the following:

\[
f(\mathbf{x}_i) = \begin{cases} M & \text{if } t_i = 1, \\ -M & \text{if } t_i = -1. \end{cases}
\]

Thus, the distance from the separator to the support vector \( D_{sv} \) can be expressed by the following expression:

\[
D_{sv} = \frac{|\mathbf{w}^T \mathbf{x}_i + \mathbf{b}|}{\|\mathbf{w}\|} = \frac{M}{\|\mathbf{w}\|}.
\]

In this case, because the filled circles and squares in Fig. 25 correspond to \( t_i = 1 \) and \( t_i = -1 \), respectively, the following relation is always satisfied:

\[
t_i f(\mathbf{x}_i) \geq M, \quad i = 1, 2, \ldots, n.
\]

Thus, the value to be optimized in SVC can be expressed as

\[
\max_{\mathbf{w}} \frac{M}{\|\mathbf{w}\|} \quad \text{s.t.} \quad t_i (\mathbf{w}^T \mathbf{x}_i + \mathbf{b}) \geq M, \quad i = 1, 2, \ldots, n.
\]

This is because we want SVC to maximize the margin formed by the marginal lines. Next, we divide all terms in the equation by \( M \) and perform the transformation \( \mathbf{w} = \mathbf{w}/M \) and \( b = b/M \). Then, (4.20) can be written as

\[
\max_{\mathbf{w}} \frac{1}{\|\mathbf{w}\|} \quad \text{s.t.} \quad t_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad i = 1, 2, \ldots, n.
\]

As maximizing \( 1/\|\mathbf{w}\| \) is equivalent to minimizing the squared norm of \( \mathbf{w} \), the final equations are expressed in the following form:

\[
\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{s.t.} \quad t_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad i = 1, 2, \ldots, n.
\]

This is the optimization equation of SVC, specifically, the hard-margin SVC. The hard-margin SVC assumes that there is a separator that classifies the training data perfectly. This implies that there are no instances left within the margin. Unfortunately, the assumption in hard margin is too idealistic. In real-world problems, soft-margin SVC is normally used. The soft-margin SVC allows instances to exist out of the margin. These instances range from the separator to the marginal line, such as open symbols, as shown in Fig. 26.

![Fig. 26. Open symbols are instances that do not exist in the range from the separator to the marginal line.](image)

To achieve this, the soft-margin SVC introduces the use of a non-negative value \( \xi \). Using this value, the constraint condition of SVC is transformed to the following form:

\[
t_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, 2, \ldots, n.
\]

The optimization problem of SVC is then transformed to the form shown below:

\[
\min_{\mathbf{w}, \xi} g(\mathbf{w}, \xi) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \quad \text{s.t.} \quad - (t_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i) \leq 0 \quad \text{and} \quad \xi_i \leq 0, \quad i = 1, 2, \ldots, n.
\]
where \( C \) is a hyperparameter determined in a trial-and-error manner during the learning process. Note that there is a summation of \( \xi \) on the right-hand side of the formula. The existence of this term implies that the soft-margin SVC attempts to minimize the number of instances that exist in the inner region of the margin. This is a problem to be solved by the soft-margin SVC. If \( \xi \) is zero, the problem is identical to that of a hard-margin SVC. The form of SVC as a problem is a primal problem. By changing primal problems to dual problems, the number of variables and difficulties of problems can be reduced. To achieve this, we use the Lagrangian system with Lagrange multipliers \( \alpha \) and \( \beta \) as follows:

\[
L(w, b, \xi, \alpha, \beta) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i (t_i (w^\top x_i + b - 1 + \xi_i)) - \sum_{i=1}^{n} \beta_i \xi_i, \tag{4.25}
\]

where \( w, b, \) and \( \xi \) are called primal variables, and \( \alpha \) and \( \beta \) are called dual variables.

Next, we consider the problem of maximizing \( L(w, b, \xi, \alpha, \beta) \) with regard to \( \alpha \) and \( \beta \) as follows:

\[
\max_{\alpha, \beta} L(w, b, \xi, \alpha, \beta) \text{ s.t. } \alpha_i, \beta_i \geq 0, \quad i = 1, 2, \ldots, n. \tag{4.26}
\]

Here, if the two constraints in (4.24) are not satisfied, \( L(w, b, \xi, \alpha, \beta) \) increases infinitely, and thus the problem does not have any solution. Conversely, if the two constraints are satisfied, the maximum value of \( L(w, b, \xi, \alpha, \beta) \) is \( g(w, \xi) \).

Thus, the following expression is satisfied:

\[
\max_{\alpha, \beta} L(w, b, \xi, \alpha, \beta) = g(w, \xi). \tag{4.27}
\]

Next, we consider the following inequality:

\[
\min_{w, b, \xi} \max_{\alpha, \beta} L(w, b, \xi, \alpha, \beta) \geq \max_{\alpha, \beta} \min_{w, b, \xi} L(w, b, \xi, \alpha, \beta). \tag{4.28}
\]

As the following inequality is always satisfied:

\[
\max_{\alpha, \beta} L(w, b, \xi, \alpha, \beta) \geq \max_{\alpha, \beta} \min_{w, b, \xi} L(w, b, \xi, \alpha, \beta). \tag{4.29}
\]

Therefore, from (4.27) and (4.28),

\[
\min_{w, b, \xi} g(w, \xi) \geq \max_{\alpha, \beta} \min_{w, b, \xi} L(w, b, \xi, \alpha, \beta). \tag{4.30}
\]

The left-hand side of this equation is called the primal problem, which minimizes the objective function of the original SVC. The right-hand side is called the Lagrange dual problem. The inequality implies that the solution of the primal problem is the same or larger than that of the dual problem. This is a weak duality of the SVC problem. Therefore, we introduce the strong duality theorem.

**Theorem 4.3 (Strong duality of SVC [19]).** If a primal problem is a convex problem and the Slater condition is satisfied, the optimal solution of the primal problem is the same as that of the dual problem.

By this theorem, the inequality in (4.30) can be removed for the SVC problem, and thus the following equation is satisfied:

\[
\min_{w, b, \xi} g(w, \xi) = \max_{\alpha, \beta} \min_{w, b, \xi} L(w, b, \xi, \alpha, \beta). \tag{4.31}
\]

To calculate the minimal solution of \( L(w, b, \xi, \alpha, \beta) \) with regard to \( w, b, \) and \( \xi \), that is,

\[
\min_{w, b, \xi} L(w, b, \xi, \alpha, \beta), \tag{4.32}
\]

the gradient of \( L(w, b, \xi, \alpha, \beta) \) must be 0 as shown below:

\[
\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i t_i x_i = 0, \tag{4.33}
\]

\[
\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i t_i = 0, \tag{4.34}
\]

\[
\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \beta_i = 0. \tag{4.35}
\]

Here, from (4.24) and (4.26), the following conditions are required:

\[
-t_i (w^\top x_i + b - 1 + \xi_i) \leq 0, \tag{4.36}
\]

\[
-\xi_i \leq 0, \tag{4.37}
\]

\[
\alpha_i \geq 0, \tag{4.38}
\]

\[
\beta_i \geq 0. \tag{4.39}
\]
Further, from the inequalities presented by (4.36) and (4.37), the following equations must be satisfied such that $L(w, \beta, \xi, \alpha, \beta)$ has an optimum value:

$$-\alpha_i(t_i(w^\top x_i + b) - 1 + \xi_i) = 0, \quad (4.40)$$

$$-\beta_i \xi_i = 0. \quad (4.41)$$

The equations and inequalities represented from (4.33) to (4.41) indicate the Karush–Kuhn–Tucker condition of the current problem. Using the relations from (4.33) to (4.35) and (4.25), the problem represented by (4.32) is solved as follows:

$$\min_{w, b, \xi, \alpha, \beta} L(w, b, \xi, \alpha, \beta) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j t_i t_j x_i^\top x_j. \quad (4.42)$$

In addition, because $C - \alpha_i = \beta_i \geq 0$ is satisfied based on (4.35) and (4.39), the following expression is also satisfied:

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, \ldots, n. \quad (4.43)$$

Finally, the following expressions are obtained as the optimization problem of SVC:

$$\min_{w, b, \xi} g(w, \xi) = \max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j t_i t_j x_i^\top x_j$$

s.t. $\sum_{i=1}^{n} \alpha_i t_i = 0$, $0 \leq \alpha_i \leq C, \quad i = 1, 2, \ldots, n. \quad (4.44)$

Note that as (4.42) does not include $\beta$, the right-hand side of (4.44) becomes a maximizing problem that depends primarily only on $\alpha$. Thus, the problem of minimizing the original objective function of SVC is transformed to the problem of maximizing the function of a dual variable. To calculate the optimal $\alpha$, we can use the gradient descent or other optimization methods. Finally, the equation of the separator is calculated as follows:

$$f(x) = \sum_{i=1}^{n} \alpha_i t_i x_i^\top x + b. \quad (4.45)$$

As shown in Fig. 25(b), the support vectors are based on the separator, and the following equation is satisfied:

$$t_a f(x_a) = t_a \left( \sum_{i=1}^{n} \alpha_i t_i x_i^\top x_a + b \right) = 1, \quad (4.46)$$

where $x_a$ and $t_a$ are the input and target vectors of a support vector, respectively. From this equation and $t_a^2 = 1$, the bias parameter is calculated as follows:

$$b = t_a - \sum_{i=1}^{n} \alpha_i t_i x_i^\top x_a. \quad (4.47)$$

By substituting this value of $b$ in (4.45), we have

$$f(x) = \sum_{i=1}^{n} \alpha_i t_i x_i^\top x + t_a - \sum_{i=1}^{n} \alpha_i t_i x_i^\top x_a. \quad (4.48)$$

Although the SVC is originally a linear discriminator, it can be modified to a nonlinear discriminator using the kernel method. Here, using the same notations as used in the previous subsection, we introduce a projection function $\phi(x)$. This function projects an original input vector $x$ onto a high-dimensional feature space where the projected data are separated linearly. The equation for the separator is transformed as follows:

$$f(x) = w^\top \phi(x) + b. \quad (4.49)$$

Thus, (4.48) can be transformed as follows:

$$f(x) = \sum_{i=1}^{n} \alpha_i t_i \phi^\top(x_i) \phi(x) + t_a - \sum_{i=1}^{n} \alpha_i t_i \phi^\top(x_i) \phi(x_a). \quad (4.50)$$

Note that, in this equation, we have an inner product of projection functions. This inner product can be replaced by the kernel function mentioned in the previous subsection [refer to (4.8)]. Therefore, the separator is expressed in the following form using the kernel function:

$$f(x) = \sum_{i=1}^{n} \alpha_i t_i k(x_i, x) + t_a - \sum_{i=1}^{n} \alpha_i t_i k(x_i, x_a). \quad (4.51)$$
This is the final form of the kernel SVC that has the ability to classify nonlinear data. As shown in the equation, the SVC is required to calculate the kernel of all-against-all pairs of instances during a learning process. For calculating the Gram matrix that includes the all-against-all kernel information, a time complexity of $O(n^2)$ is required. This computation time can be considerably large when the size of the dataset is large.

In conclusion, we introduced the principles of kernel SVC. To separate the input data ideally, the SVC attempts to maximize the margin. Moreover, to reduce the difficulty of the optimization problem, a Lagrange multiplier is used. The primal problem is transformed into a dual problem. Because it is difficult to solve the problem analytically, the target variable $\alpha$ is solved computationally. Finally, to analyze the nonlinear relationship in the dataset, we introduced the concept of the kernel function. This kernel SVC, as shown, has the ability to classify nonlinear data.

4.4 Summary

In this section, we introduced the kernel method using a simple regression problem. With the kernel function, a linear method can be modified to a nonlinear method. In addition, we introduced the principle of SVC. Although SVC is originally considered as a linear method, it can be modified into a nonlinear method with the kernel function. Because the calculation of the Gram matrix is required in this process, the time complexity of SVC is not good. The time complexity of SVM is $O(n^2)$, where $n$ is the number of instances in the dataset. This time complexity is smaller than calculating the $d$-dimensional projection with the mapping function $\phi$ explicitly. However, in this era of big data, the time complexity of $O(n^2)$ is not fast in practice. For example, if we have one million input data, we require one trillion computations to calculate the Gram matrix. Neural networks generally do not require such a large computation time to reach the optimal solution. Consequently, SVM became outdated in the boom of artificial intelligence. To improve the computation speed, various heuristic methods such as randomized Fourier features [20] are being developed. The convergence performance of SVM is excellent and the theory is well organized. Future success in improving the computation time revive the boom of SVM one more time.

5. Decision Trees and Ensemble Learning Methods

5.1 Overview

A decision tree is a supervised learning method. This method is significantly intuitive and has been studied for a long time. The early contributors are Breiman, Friedman, Stone, and Olsen [21]. In their paper, they developed the classification and regression tree (CART) algorithm, which is still being used in the present study. Iterative Dichotomiser 3 (ID3) [22] is also a well-known method, which was originally developed by Quinlan. The decision tree uses a rooted tree, as shown in Fig. 27 for prediction. This tree helps us make a decide regarding a certain problem. In the example, the problem is “should we go on a picnic or not?” Here, the decision tree helps us resolve this problem by using different information such as weather, humidity, and wind.

![Decision Tree Diagram](image)

Fig. 27. Example of decision trees. The problem considered here is whether one should go on a picnic or not.
includes options of events, namely question, and each leaf node represents a class label. The paths from the root to the leaves represent classification rules. The leaf nodes have classification labels. For example, the leftmost node in the bottom layer has the label of "Yes" and the rightmost node in the bottom layer has the label of "No."

As described, decision trees can be used for classification problems. In addition, it can be used for regression problems. Using a decision tree, we can solve both regression and classification problems. If labels belong to a discrete set of values, the tree is called a classification tree. Otherwise, it is called a regression tree. It can be noted that in addition to prediction, machine learning methods are used to study the characteristics of data. Machine learning methods not only have to be of high performance but also have the capability to extract domain-specific knowledge from analysis results. In this sense, the important thing is that machine learning methods have high interpretability. However, it is difficult to interpret why and how the predictor judges the results when we are using basic machine learning methods such as neural networks, deep learning methods, and SVMs. These methods are called "black-box" methods. Conversely, the criteria of the decision tree are relatively clear. When we obtain a label from an instance using a decision tree, we can easily understand why the label is generated. Thus, decision trees are interpretable. The interpretability of a decision tree is a significantly strong advantage when compared to other machine learning methods.

Unfortunately, there are several limitations in decision trees. For example, the prediction performance of a decision tree is not higher than that of more sophisticated methods such as neural networks and SVMs. Various methods were developed to overcome this disadvantage. The most famous method is the ensemble learning method, although it can actually be utilized not only for decision trees but also for various machine learning methods. The ensemble learning method is a type of learning technique in which several weak predictors are generated and the final prediction results are obtained by using these predictors. In this section, the most famous methods, such as bagging, boosting, and stacking, are illustrated.

5.2 Principle of decision trees

5.2.1 Algorithm

In a standard decision tree, the tree takes an instance of binary attributes and returns a binary value called a label. Each internal node stores a question, and each branch stores an answer to the question written in the parent node. In addition, each leaf node stores a label that can be obtained as follows: starting from the root, we read the question and select the answer from the branches, then move to the corresponding child node. Then, we repeatedly answer questions until we obtain a label (i.e., until we reach a leaf).

Before we generate a decision tree, it is helpful to formalize the problem first. Let $X_k$ ($k = 1, 2, \ldots, m$) be the set of attributes and $Y$ be the set of labels. For the sake of simplicity, we assume that the attributes and labels are binary sets, namely, $X_1 = X_2 = \cdots = X_m = Y = \{0, 1\}$. The input of this decision tree is a set of instances $[(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})]$ where $x^{(1)}, \ldots, x^{(n)} \in X_1 \times X_2 \times \cdots \times X_m$ and $y^{(1)}, \ldots, y^{(n)} \in Y$.

When the label of an instance is one, the instance is called a positive instance. Conversely, when the label of an instance is zero, the instance is called a negative instance. Our goal is to generate a decision tree that predicts $y^{(i)}$ from $x^{(i)}$ ($i = 1, \ldots, m$). Note that the tree should work for instances with unknown labels. If the tree overfits the given instances, it would not predict the labels of unknown instances with high accuracy. The first goal here is to divide all instances into positive and negative instances using a series of questions in the tree. To generate a tree, we can use well-known algorithms such as CART [21], ID3 [22], and C4.5 [23]. These algorithms assume that each question consists of a single attribute. Each internal node asks the value of an attribute, while the corresponding branches consist of two attributes. The division of instances can be understood by using the simple example shown in Table 3. There are 26 students, and 10 of them are sick with the flu, which we call positive instances. We know their height and body temperature and want to select an attribute that is useful to pick up the students who are sick with the flu. Intuitively, "body temperature $\geq 38^\circ C$" is a good condition because one divided set consists of many positive instances and the other set consists of many negative instances. Conversely, "height $\geq 165$" is not a good condition. Intuitively, good attributes divide instances into positive and negative instances with high accuracy.

| Height (in cm) | Body Temperature (in °C) | Total |
|---------------|--------------------------|-------|
| < 165         |                          |       |
| Not Flu:      | 7                        | 9     |
| Flu:          | 0                        | 5     |
| ≥ 165         |                          |       |
| Not Flu:      | 6                        | 7     |
| Flu:          | 1                        | 5     |
| Total         | 13                       | 9     |
| Not Flu:      | 3                        | 3     |
The basic strategy is to create two divided sets that are as pure as possible. First, we must define the impurity of the set in mathematical terms. Let \( \text{Imp}(S) \) denote the impurity of a set \( S \). When we divide a set \( D \) into \( D_1 \) and \( D_2 \) using an attribute, the impurity of \( D_1 \) and \( D_2 \) is defined as follows:

\[
\text{Imp}(D_1, D_2) = \frac{|D_1|}{|D|} \cdot \text{Imp}(D_1) + \frac{|D_2|}{|D|} \cdot \text{Imp}(D_2).
\]

In summary, these algorithms repeatedly select an attribute in a given set \( D \) to minimize \( \text{Imp}(D_1, D_2) \). These algorithms repeatedly select a good attribute in a greedy manner. First, they select an attribute that minimizes the impurity of the two divided sets. Then, they select an attribute from each divided set. They repeat it until the impurity of a divided set becomes smaller than a certain threshold. We can use additional termination conditions, such as maximum tree height. The time complexity of the algorithms is \( O(nm^3) \), where \( m \) is the number of attributes and \( n \) is the number of instances. The worst case occurs when the height of the tree is \( O(m) \) because attribute selection on the nodes at height \( h \) requires \( O(nm) \) time in total. In several cases, these algorithms run significantly faster than the worst case because the height of the tree tends to be smaller than \( m \).

The Gini index and entropy are sometimes discussed when talking about the decision tree. Given a set of instances \( S \) containing \( n_p \) positive instances, we define \( \hat{p} \) as the proportion of the positive instances, namely, \( \hat{p} = n_p/|S| \). Thus, we can define the Gini index and entropy of this problem. The Gini index represents the value of the expected error rate when we randomly select an instance and predict that it is positive with probability \( \hat{p} \). In detail, we select a positive instance with probability \( \hat{p} \). In such a case, the error rate is \( 1 - \hat{p} \). Similarly, we select a negative instance with probability \( 1 - \hat{p} \). In such a case, the error rate is \( \hat{p} \). In the result, the total error rate, that is, the Gini index, is \( \hat{p}(1 - \hat{p}) + (1 - \hat{p})\hat{p} = 2\hat{p}(1 - \hat{p}) \). Entropy represents the value of the expected amount of information we obtain when we know that an instance is positive or negative. The value in this case is \( -\hat{p} \log_2 \hat{p} - (1 - \hat{p}) \log_2 (1 - \hat{p}) \).

Let us consider certain variations. If the attributes have more than two values, the number of branches exiting the corresponding node is more than two. The impurity of multiple sets is defined as a weighted average impurity, as is the case with two sets. If the label has more than two values, we can also define the impurity similarly. If attributes have continuous values, we split values into two categories using a threshold. One category contains values less than the threshold, and the other contains the rest. For each attribute, we calculated the threshold value to minimize the impurity of divided sets.

### 5.2.2 Techniques to improve performance

There are several techniques to improve the performance of decision trees. One of the most important methods is regularization methods related to Occam’s razor. Occam’s razor is a concept that states, “to explain a concept, the most basic assumption is appropriate.” This concept is related to the overfitting problems of machine learning analysis. Generally, unnecessarily complex models, namely models with too many parameters, tend to induce overfitting. In decision tree analysis, Occam’s razor specifies that trees should be compact for performance. Several techniques are used to reduce overfitting. While analyzing decision trees, the most standard methods are “pruning” and “setting constraints on the depth of the final tree.” Both methods attempt to achieve a compact tree based on the concept of Occam’s razor. The first one is pruning, which deletes redundant nodes from the final tree. As described in the previous section, the object function of decision trees is the reduction of impurity. The reduction is calculated based on the scores of the branches or scores of the questions. The total score can be calculated for all questions in the final tree. Splits with a higher score are regarded as good. The pruning sets a threshold value on the score and prevents establishing a split with less score. A question with a lower score implies that the question is less effective in reducing the impurity of whole models. Nevertheless, complex models with more questions are at risk of overfitting. Therefore, pruning attempts to delete the less effective split from the final model. The second method to improve the performance of decision tree analysis is setting the maximum depth to the final tree. By setting the maximum depth, the elongation of a tree is restricted, and thus the complexity of the model can be reduced. The threshold value of depth is a hyperparameter, and we have to decide the value in the learning process. Both methods attempt to generate less-complex models, and the objective of the regularization is to make a model as compact as possible.

### 5.3 Ensemble learning methods

#### 5.3.1 Merit of the ensemble learning method

Although the decision tree is a white-box method and useful for data analysis, the prediction performance is not superior to the other sophisticated methods, such as neural networks and SVMs. To overcome this limitation, we can use another method called the ensemble learning method. The ensemble method modifies the decision tree and further improves the performance. Ensemble learning is a way to learn and predict something by combining more than two predictors. We will explain three types of ensemble learning methods: “bagging,” “boosting,” and “stacking.”
5.3.2 Bagging

Bagging is a method in which several weak predictors are created for randomly chosen samples, and the final output is calculated by averaging the results of outputs by the weak predictors. As shown in Fig. 28, a novel sampled dataset is generated from the original dataset. The generated dataset is used for creating weak predictors. The final output is calculated by averaging the output from the weak predictors. To generate the sample dataset, the so-called bootstrap method is necessary. Bootstrap is a traditional sampling method that is often used in the field of statistics [24]. If \( N \) is the number of instances in the original dataset, the bootstrap randomly extracts instances from the dataset by \( N \) times, allowing the overlap of sampling instances. Consequently, the newly sampled dataset will have \( N \) instances.

![Computation scheme of bagging.](image)

The averaging methods include voting, calculating the mean value, and various other implementations. In the classification problem, the final output is calculated using the principle of majority voting. In the regression problem, the final output is determined by calculating the mean value of the outputs from weak predictors. Bagging is effective because it has the ability to reduce the variance of models by averaging weak predictors. One of the machine learning methods, called random forest, is considered as the most representative machine learning method with bagging. Random forest utilizes several decision trees as weak predictors. In several situations, the random forest can produce a satisfactory prediction performance.

5.3.3 Boosting

Boosting is a method in which weak predictors are serially updated based on previously obtained information. The basic idea of boosting is to minimize errors between the predicted results and the correct answers. While this concept naturally happens in the context of machine learning, boosting explicitly focuses on this idea. As shown in Fig. 29, boosting first creates a weak predictor using the original dataset. Next, based on the prediction results of the weak predictor, the original dataset is modified to the weighted dataset. Then, another weak predictor is generated using the newly generated weighted dataset. Finally, the final output is calculated by averaging the outputs from the weak predictors.

![Computation scheme of boosting.](image)
In the newly generated weighted dataset, the instances when the weak predictor cannot output the correct answers are considered to be more important than the other instances, namely those instances when the weak predictor succeeded in outputting the correct answer. The instances then continued to be weighted. Boosting focuses more on difficult problems rather than on easy problems. The basic idea of boosting is overcoming difficult problems leading to higher performance of a predictor. Boosting attempts to ensure that the weak predictors overcome the weak points. The difficult problems, of course, generate two things: a difference between predicted results and correct answers, and the difference in errors between outputs by predictors and true distribution. The concept of boosting is relatively simple and attempts to reduce the errors. The most popular learning method using boosting is AdaBoost, which is the method that we discussed here. In addition, gradient boosting is also a popular method. The concept of this method is similar to that of AdaBoost. Gradient boosting regards the errors in the form of residuals between predictions by weak predictors and the true distribution as the gradient. In gradient boosting, after the first step of learning, residuals are used as a novel dataset to be learned. The most famous implementation of gradient boosting is XGBoost, and it is frequently adopted by many winning solutions in machine learning challenges hosted at Kaggle (https://www.kaggle.com/).

5.3.4 Stacking

Stacking is considered currently as the strongest ensemble method. Stacking has distinctive features: it generates novel datasets by adding the results of weak predictors into the original data as new attributes. The other feature in stacking is that it has a blender that accepts results from weak predictors and outputs the final result by using the information from the weak predictors.

As shown in Fig. 30, a new dataset is generated based on the original data and prediction results. In the final phase, the blender decides which information should be used for the final results. Note that stacking can reduce bias derived from machine learning methods because it may include various machine learning models as weak predictors.

5.4 Summary

Unlike black-box models such as neural networks and SVMs, the decision tree is classified as a white-box model. The feature of the white-box model is that it has high interpretability. Thus, the method is useful for data analysis in addition to its use as a prediction model. The basic idea of the decision tree algorithm is to reduce the impurity of nodes. The well-known metrics of impurity are Gini impurity and entropy. One of the weaknesses of the decision tree is that it tends to overfit the learning dataset. To prevent overfitting, decision tree models are generated based on the concept of Occam’s razor. The most popular methods to prevent overfitting problems include pruning the less effective branches and setting a maximum depth for the final tree. Because the prediction performance of the decision tree is poor, ensemble learning methods are often used to enhance this. There are several ensemble learning methods that include bagging, boosting, and stacking. Ensemble methods are considerably useful in improving the performance of a prediction and they are frequently used in various machine learning competitions.

6. Importance of Preparing Datasets in Machine Learning Analysis

Preparing datasets is one of the most important aspects of machine learning. As mentioned previously, machine learning analysis normally includes two phases: the learning and prediction phases. In the learning phase, the machines learns from the data of interest and determines the rules from it. These machines after performing the learning process are known as predictors. The learning process is also called the inference process. The later phase is the prediction phase. In this phase, we let the predictors predict the properties or features of the new data. This phase is also called the test phase.

For these phases, we have to prepare a proper dataset. When we start analyzing the dataset (original dataset), we first split it into two datasets, which are used for the learning and test phases. The first one is the “learning dataset” and the
second one is “test dataset,” as shown in Fig. 31. Further, the learning dataset must be divided into training and validation datasets.

Using the learning dataset, the parameters of machine learning algorithms are updated to construct predictors. The test dataset is used to evaluate the performance of the predictor, which is the final product of the machine learning analysis. This dataset must be independent of the learning dataset, although it follows the same probability distribution as the learning dataset.

The learning dataset is further divided into training and validation datasets. The training dataset is used to update parameters to produce predictors. During the learning phase, the performance of the predictors is monitored and evaluated based on the validation dataset. Through this process, we confirm whether parameter update is performed appropriately and determine whether overfitting occurs. Here, the word “independent” implies that the data are expected to follow the same probability distribution, but the process of generating data is different from each other. If the dataset is not divided into these datasets properly, it is illogical to perform machine learning analysis. The most important thing is that predictors have a generalization performance for any data. If we evaluate a predictor on the training or validation dataset, it is meaningless. In this case, we cannot determine whether predictors are overfitted to the learning dataset. In addition, if the test dataset is dependent on the learning dataset, it is also meaningless. In this case, the predictors are possibly overfitted to the learning dataset.

Acknowledgments

This work was supported in part by the International Priority Graduate Programs (Data Science Program) and the Top Global University Project (Graduate Program in Data Science) from the Ministry of Education, Culture, Sports, Science and Technology of Japan (MEXT).

REFERENCES

[1] Alpaydin, E., Introduction to Machine Learning, 3rd ed., Adaptive Computation and Machine Learning, MIT Press, Cambridge, MA (2014).
[2] Ruder, S., An Overview of Gradient Descent Optimization Algorithms, arXiv preprint arXiv:1609.04747 (2016).
[3] Nocedal, J., “Updating quasi-Newton matrices with limited storage,” *Mathematics of Computation*, 35(151): 773–782 (1980).
[4] Liu, D. C., and Nocedal, J., “On the limited memory BFGS method for large scale optimization,” *Mathematical Programming*, 45(1–3): 503–528 (1989).
[5] Zeiler, M. D., ADADELTA: An Adaptive Learning Rate Method. arXiv 2012, arXiv preprint arXiv:1212.5701, 1212 (2012).
[6] Rosenblatt, F., “A probabilistic model for information storage and organization in the brain,” *Artificial Intelligence: Critical Concepts*, 2(6): 398 (2000).
[7] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., “Learning representations by back-propagating errors,” *Cognitive Modeling*, 5(3): 1 (1988).
[8] Cybenko, G., “Approximations by superpositions of a sigmoidal function,” *Mathematics of Control, Signals and Systems*, 2: 183–192 (1989).
[9] Nair, V., and Hinton, G. E., Rectified Linear Units Improve Restricted Boltzmann Machines, Proceedings of the 27th International Conference on Machine Learning (ICML-10) (2010) pp. 807–814.
[10] Kingma, D. P., and Ba, J., Adam: A Method for Stochastic Optimization, arXiv preprint arXiv:1412.6980 (2014).
[11] Clevert, D.-A., Unterthiner, T., and Hochreiter, S., Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs), arXiv preprint arXiv:1511.07289 (2016).
121

Concatenated Rectified Linear Units, arXiv preprint arXiv:1603.05201 (2016).
[13] Krishevsky, A., Convolutional Deep Belief Networks on CIFAR-10 (2010).
[14] Krizhevsky, A., Sutskever, I., and Hinton, G. E., Imagenet Classification with Deep Convolutional Neural Networks, Advances in Neural Information Processing Systems (2012) pp. 1097–1105.
[15] Hochreiter, S., and Schmidhuber, J., “Long short-term memory,” Neural Computation, 9(8): 1735–1780 (1997).
[16] Gers, F. A., Schmidhuber, J., and Cummins, F., “Learning to forget: Continual prediction with LSTM,” Neural Computation, 12(10): 2451–2471 (2000).
[17] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y., Learning Phrase Representations Using RNN Encoder-decoder for Statistical Machine Translation, arXiv preprint arXiv:1406.1078 (2014).
[18] Sutskever, I., Vinyals, O., and Le, Q. V., Sequence to Sequence Learning with Neural Networks, Advances in Neural Information Processing Systems (2014).
[19] Boyd, S., and Vandenberghe, L., Convex Optimization, Cambridge University Press (2004).
[20] Rahimi, A., and Recht, B., Random Features for Large-scale Kernel Machines, Advances in Neural Information Processing Systems (2008) pp. 1177–1184.
[21] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J., Classification and Regression Trees, Wadsworth, Belmont, CA (1984).
[22] Quinlan, R. J., “Induction of decision trees,” Machine Learning, 1(1): 81–106 (1986).
[23] Quinlan, R. J., C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA (1993).
[24] Efron, B., “Bootstrap methods: Another look at the jackknife,” Annals of Statistics, 7(1): 1–26 (1979).