

Introduction

This book is concerned with the challenge of mining knowledge from data. The world is full of data. Some of the oldest written records on clay tablets are dated back to 4000 BC. With the creation of paper, data had been stored in myriads of books and documents. Today, with increasing use of computers, tremendous volumes of data have filled hard disks as digitized information. In the presence of the huge amount of data, the challenge is how to truly understand, integrate, and apply various methods to discover and utilize knowledge from data. To predict future trends and to make better decisions in science, industry, and markets, people are starved for discovery of knowledge from this morass of data.

Though ‘data mining’ is a new term proposed in recent decades, the tasks of data mining, such as classification and clustering, have existed for a much longer time. With the objective to discover unknown patterns from data, methodologies of data mining are derived from machine learning, artificial intelligence, and statistics, etc. Data mining techniques have begun to serve fields outside of computer science and artificial intelligence, such as the business world and factory assembly lines. The capability of data mining has been proven in improving marketing campaigns, detecting fraud, predicting diseases based on medical records, etc.

This book introduces fuzzy neural networks (FNNs), multi-layer perceptron neural networks (MLPs), radial basis function (RBF) neural networks, genetic algorithms (GAs), and support vector machines (SVMs) for data mining. We will focus on three main data mining tasks: data dimensionality reduction (DDR), classification, and rule extraction. For more data mining topics, readers may consult other data mining text books, e.g., [129][130][346].

A data mining system usually enables one to collect, store, access, process, and ultimately describe and visualize data sets. Different aspects of data mining can be explored independently. Data collection and storage are sometimes not included in data mining tasks, though they are important for data mining. Redundant or irrelevant information exists in data sets, and inconsistent formats of collected data sets may disturb the processes of data mining, even

mislead search directions, and degrade results of data mining. This happens because data collectors and data miners are usually not from the same group, i.e., in most cases, data are not originally prepared for the purpose of data mining. Data warehouse is increasingly adopted as an efficient way to store metadata. We will not discuss data collection and storage in this book.

1.1 Data Mining Tasks

There are different ways of categorizing data mining tasks. Here we adopt the categorization which captures the processes of a data mining activity, i.e., data preprocessing, data mining modelling, and knowledge description. Data preprocessing usually includes noise elimination, feature selection, data partition, data transformation, data integration, and missing data processing, etc. This book introduces data dimensionality reduction, which is a common technique in data preprocessing. fuzzy neural networks, multi-layer neural networks, RBF neural networks, and support vector machines (SVMs) are introduced for classification and prediction. And linguistic rule extraction techniques for decoding knowledge embedded in classifiers are presented.

1.1.1 Data Dimensionality Reduction

Data dimensionality reduction (DDR) can reduce the dimensionality of the hypothesis search space, reduce data collection and storage costs, enhance data mining performance, and simplify data mining results. Attributes or features are variables of data samples and we consider the two terms interchangeable in this book.

One category of DDR is feature extraction, where new features are derived from the original features in order to increase computational efficiency and classification accuracy. Feature extraction techniques often involve non-linear transformation [60][289]. Sharma *et al.* [289] transformed features non-linearly using a neural network which is discriminatively trained on the phonetically labelled training data. Coggins [60] had explored various non-linear transformation methods, such as folding, gauge coordinate transformation, and non-linear diffusion, for feature extraction. Linear discriminant analysis (LDA) [27][168][198] and principal components analysis (PCA) [49][166] are two popular techniques for feature extraction. Non-linear transformation methods are good in approximation and robust for dealing with practical non-linear problems. However, non-linear transformation methods can produce unexpected and undesirable side effects in data. Non-linear methods are often not invertible, and knowledge learned by applying a non-linear transformation method in one feature space might not be transferable to the next feature space. Feature extraction creates new features, whose meanings are difficult to interpret.

The other category of DDR is feature selection. Given a set of original features, feature selection techniques select a feature subset that performs the

best for induction systems, such as a classification system. Searching for the optimal subset of features is usually difficult, and many problems of feature selection have been shown to be NP-hard [21]. However, feature selection techniques are widely explored because of the easy interpretability of the features selected from the original feature set compared to new features transformed from the original feature set. Lots of applications, including document classification, data mining tasks, object recognition, and image processing, require aid from feature selection for data preprocessing.

Many feature selection methods have been proposed in the literature. A number of feature selection methods include two parts: (1) a ranking criterion for ranking the importance of each feature or subsets of features, (2) a search algorithm, for example backward or forward search. Search methods in which features are iteratively added ('bottom-up') or removed ('top-down') until some termination criterion is met are referred to as *sequential* methods. For instance, sequential forward selection (SFS) [345] and sequential backward selection (SBS) [208] are typical *sequential* feature selection algorithms. Assume that d is the number of features to be selected, and n is the number of original features. SFS is a bottom-up approach where one feature which satisfies some criterion function is added to the current feature subset at a time until the number of features reaches d . SBS is a top-down approach where features are removed from the entire feature set one by one until $D - d$ features have been deleted. In both the SFS algorithm and the SBS algorithm, the number of feature subsets that have to be inspected is $n + (n - 1) + (n - 2) + \cdots + (n - d + 1)$. However, the computational burden of SBS is higher than SFS, since the dimensionality of inspected feature subsets in SBS is greater than or equal to d . For example, in SBS, all feature subsets with dimension $n - 1$ are inspected first. The dimensionality of inspected feature subsets is at most equal to d in SFS.

Many feature selection methods have been developed based on traditional SBS and SFS methods. Different criterion functions including or excluding a subset of features to the selected feature set are explored. By ranking each feature's importance level in separating classes, only n feature subsets are inspected for selecting the final feature subset. Compared to evaluating all feature combinations, ranking individual feature importance can reduce computational cost, though better feature combinations might be missed in this kind of approach. When computational cost is too heavy to stand, feature selection based on ranking individual feature importance is a preference.

Based on an entropy attribute ranking criterion, Dash *et al.* [71] removed attributes from the original feature set one by one. Thus only n feature subsets have to be inspected in order to select a feature subset, which leads to a high classification accuracy. And, there is no need to determine the number of features selected in advance. However, the class label information is not utilized in Dash *et al.*'s method. The entropy measure was used in [71] for ranking attribute importance. The class label information is critical for detecting irrelevant or redundant attributes. It motivates us to utilize the class label

information for feature selection, which may lead to better feature selection results, i.e., smaller feature subsets with higher classification accuracy.

Genetic algorithms (GAs) are used widely in feature selection [44][322][351]. In a GA feature selection method, a feature subset is represented by a binary string with length n . A zero or one in position i indicates the absence or presence of feature i in the feature subset. In the literature, most feature selection algorithms select a general feature subset (class-independent features) [44][123][322] for all classes. Actually, a feature may have different discriminatory capability for distinguishing different classes from other classes. For discriminating patterns of a certain class from other patterns, a multi-class data set can be considered as a two-class data set, in which all the other classes are treated as one class against the current processed class. For example, there is a data set containing the information of ostriches, parrots, and ducks. The information of the three kinds of birds includes weight, feather color (colorful or not), shape of mouth, swimming capability (whether it can swim or not), flying capability (whether it can fly or not), etc. According to the characteristics of each bird, the feature ‘weight’ is sufficient for separating ostriches from the other birds, the feature ‘feather color’ can be used to distinguish parrots from the other birds, and the feature ‘swimming capability’ can separate ducks from the other birds.

Thus, it is desirable to obtain individual feature subsets for the three kinds of birds by class-dependent feature selection, which separates each one from others better than using a general feature subset. The individual characteristics of each class can be highlighted by class-dependent features. Class-dependent feature selection can also facilitate rule extraction, since lower dimensionality leads to more compact rules.

1.1.2 Classification and Clustering

Classification and clustering are two data mining tasks with close relationships. A class is a set of data samples with some similarity or relationship and all samples in this class are assigned the same class label to distinguish them from samples in other classes. A cluster is a collection of objects which are similar locally. Clusters are usually generated in order to further classify objects into relatively larger and meaningful categories.

Given a data set with class labels, data analysts build classifiers as predictors for future unknown objects. A classification model is formed first based on available data. Future trends are predicted using the learned model. For example, in banks, individuals’ personal information and historical credit records are collected to build a model which can be used to classify new credit applicants into categories of low, medium, or high credit risks. In other cases, with only personal information of potential customers, for example, age, education levels, and range of salary, data miners employ clustering techniques to group the clusters according to some similarities and further label the customers into low, medium, or high levels for later targeted sales.

In general, clustering can be employed for dealing with data without class labels. Some classification methods cluster data into small groups first before proceeding to classification, e.g. in the RBF neural network. This will be further discussed in Chap. 4.

1.1.3 Rule Extraction

Rule extraction [28][150][154][200] seeks to present data in such a way that interpretations are actionable and decisions can be made based on the knowledge gained from the data. For data mining clients, they expect a simple explanation of why there are certain classification results: what is going on in a high-dimensional database, and which feature affects data mining results significantly, etc. For example, a succinct description of a market behavior is useful for making decisions in investment. A classifier learns from training data and stores learned knowledge into the classifier parameters, such as the weights of a neural network classifier. However, it is difficult to interpret the knowledge in an understandable format by the classifier parameters. Hence, it is desirable to extract IF-THEN rules to represent valuable information in data.

Rule extraction can be categorized into two major types. One is concerned with the relationship between input attributes and output class labels in labelled data sets. The other is association rule mining, which extracts relationships between attributes in data sets which may not have class labels. Association rule extraction techniques are usually used to discover relationships between items in transaction data. An association rule is expressed as ' $X \Rightarrow Z$ ', where X and Z are two sets of items. ' $X \Rightarrow Z$ ' represents that if a transaction $T \in D$ contains X , then the transaction also contains Z , where D is the transaction data set. A confidence parameter, which is the conditional probability $p(Z \in T \mid X \in T)$ [137], is used to evaluate the rule accuracy. The association rule mining can be applied for analyzing supermarket transactions. For example, 'A customer who buys butter will also buy bread with a certain probability'. Thus, the two associated items can be arranged in close proximity to improve sales according to this discovered association rule. In the rule extraction part of this book, we focus on the first type of rule extraction, i.e., rule extraction based on classification models. Usually, association rule extraction can be treated as the first category of rule extraction, which is based on classification. For example, if an association rule task is to inspect what items are apt to be bought together with a particular item set X , the item set X can be used as class labels. The other items in a transaction T are treated as attributes. If X occurs in T , the class label is 1, otherwise it is labelled 0. Then, we could discover the items associated with the occurrence of X , and also the non-occurrence of X . The association rules can be equally extracted based on classification. The classification accuracy can be considered as the rule confidence.

RBF neural networks are functionally equivalent to fuzzy inference systems under some restrictions [160]. Each hidden neuron could be considered as a fuzzy rule. In addition, fuzzy rules could be obtained by combining fuzzy logic with our crisp rule extraction system. In Chap. 3, fuzzy rules are presented. For crisp rules, there are three kinds of rule decision boundaries found in the literature [150][154][200][214]: hyper-plane, hyper-ellipse, and hyper-rectangular. Compared to the other two rule decision boundaries, a hyper-rectangular decision boundary is simpler and easier to understand. Take a simple example; when judging whether a patient gets a high fever, his body temperature is measured and a given temperature range is preferred to a complex function of the body temperature. Rules with a hyper-rectangular decision boundary are more understandable for data mining clients. In the RBF neural network classifier, the input data space is separated into hyper-ellipses, which facilitates the extraction of rules with hyper-rectangular decision boundaries. We also describe crisp rules in Chap. 7 and Chap. 10 of this book.

1.2 Computational Intelligence Methods for Data Mining

1.2.1 Multi-layer Perceptron Neural Networks

Neural network classifiers are very important tools for data mining. Neural interconnections in the brain are abstracted and implemented on digital computers as neural network models. New applications and new architectures of neural networks (NNs) are being used and further investigated in companies and research institutes for controlling costs and deriving revenue in the market. The resurgence of interest in neural networks has been fuelled by the success in theory and applications.

A typical multi-layer perceptron (MLP) neural network shown in Fig. 1.1 is most popular in classification. A hidden layer is required for MLPs to classify linearly inseparable data sets. A hidden neuron in the hidden layer is shown in Fig. 1.2.

The j th output of a feedforward MLP neural network is:

$$y_j = f\left(\sum_{i=1}^K W_{ij}^{(2)} \phi_i(\mathbf{x}) + b_j^{(2)}\right), \quad (1.1)$$

where $W_{ij}^{(2)}$ is the weight connecting hidden neuron i with output neuron j . K is the number of hidden neurons. $b_j^{(2)}$ is the bias of output neuron j . $\phi_i(\mathbf{x})$ is the output of hidden neuron i . \mathbf{x} is the input vector.

$$\phi_i(\mathbf{x}) = f(\mathbf{W}_i^{(1)} \cdot \mathbf{x} + b_i^{(1)}), \quad (1.2)$$

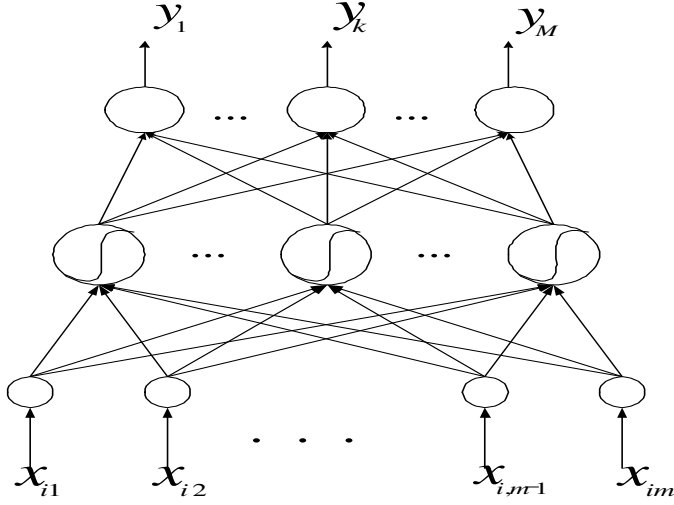


Fig. 1.1. A two-layer MLP neural network with a hidden layer and an output layer. The input nodes do not carry out any processing.

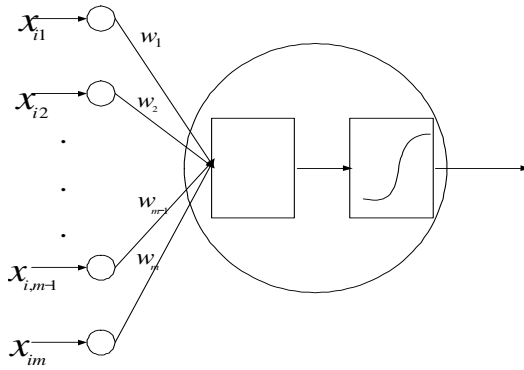


Fig. 1.2. A hidden neuron of the MLP.

where $\mathbf{W}_i^{(1)}$ is the weight vector connecting the input vector with hidden neuron i . $b_i^{(1)}$ is the bias of hidden neuron i .

A common activation function f is a sigmoid function. The most common of the sigmoid functions is the logistic function:

$$f(z) = \frac{1}{1 + e^{-\beta z}}. \quad (1.3)$$

where β is the gain.

Another sigmoid function often used in MLP neural networks is the hyperbolic tangent function that takes on values between -1 and 1 :

$$f(z) = \frac{e^{\beta z} - e^{-\beta z}}{e^{\beta z} + e^{-\beta z}}, \quad (1.4)$$

There are many training algorithms for MLP neural networks. As summarized in [63][133], the training algorithms include: (1) gradient descent error back-propagation, (2) gradient descent with adaptive learning rate back-propagation, (3) gradient descent with momentum and adaptive learning rate back-propagation, (4) Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton back-propagation, (5) bayesian regularization back-propagation, (6) conjugate gradient back-propagation with Powell-Beale restarts, (7) conjugate gradient back-propagation with Fletcher-Reeves updates, (8) conjugate gradient back-propagation with Polak-Ribiere updates, (9) scaled conjugate gradient back-propagation, (10) the Levenberg-Marquardt algorithm, and (11) one-step secant back-propagation.

1.2.2 Fuzzy Neural Networks

Symbolic techniques and crisp (non-fuzzy) neural networks have been widely used for data mining. Symbolic models are represented as either sets of 'IF-THEN' rules or decision trees generated through symbolic inductive algorithms [30][251]. A crisp neural model is represented as an architecture of threshold elements connected by adaptive weights. There have been extensive research results on extracting rules from trained crisp neural networks [110][116][200][297][313][356]. For most noisy data, crisp neural networks lead to more accurate classification results.

Fuzzy neural networks (FNNs) combine the learning and computational power of crisp neural networks with human-like descriptions and reasoning of fuzzy systems [174][218][235][268][336][338]. Since fuzzy logic has an affinity with human knowledge representation, it should become a key component of data mining systems. A clear advantage of using fuzzy logic is that we can express knowledge about a database in a manner that is natural for people to comprehend. Recently, there has been much research attention devoted to rule generation using various FNNs. Rather than attempting an exhaustive literature survey in this area, we will concentrate below on some work directly related to ours, and refer readers to a recent review by Mitra and Hayashi [218] for more references.

In the literature, crisp neural networks often have a fixed architecture, i.e., a predetermined number of layers with predetermined numbers of neurons. The weights are usually initialized to small random values. Knowledge-based networks [109][314] use crude domain knowledge to generate the initial network architecture. This helps in reducing the search space and time required for the network to find an optimal solution. There have also been mechanisms to generate crisp neural networks from scratch, i.e., initially there are no neurons or weights, which are generated and then refined during training. For example, Mezard and Nadal's tiling algorithm [216], Fahlman and Lebiere's

cascade correlation [88], and Giles *et al.*'s constructive learning of recurrent networks [118] are very useful.

For FNNs, it is also desirable to shift from the traditional fixed architecture design methodology [143][151][171] to self-generating approaches. Higgins and Goodman [135] proposed an algorithm to create a FNN according to input data. New membership functions are added at the point of maximum error on an as-needed basis, which will be adopted in this book. They then used an information-theoretic approach to simplify the rules. In contrast, we will combine rules using a computationally more efficient approach, i.e., a fuzzy similarity measure.

Juang and Lin [165] also proposed a self-constructing FNN with online learning. New membership functions are added based on input-output space partitioning using a self-organizing clustering algorithm. This membership creation mechanism is not directly aimed at minimizing the output error as in Higgins and Goodman [135]. A back-propagation-type learning procedure was used to train network parameters. There were no rule combination, rule pruning, or eliminations of irrelevant inputs.

Wang and Langari [335] and Cai and Kwan [41] used self-organizing clustering approaches [267] to partition the input/output space, in order to determine the number of rules and their membership functions in a FNN through batch training. A back-propagation-type error-minimizing algorithm is often used to train network parameters in various FNNs with batch training [160], [151].

Liu and Li [197] applied back-propagation and conjugate gradient methods for the learning of a three-layer regular feedforward FNN [37]. They developed a theory for differentiating the input-output relationship of the regular FNN and approximately realized a family of fuzzy inference rules and some given fuzzy functions.

Frayman and Wang [95][96] proposed a FNN based on the Higgins-Goodman model [135]. This FNN has been successfully applied to a variety of data mining [97] and control problems [94][98][99]. We will describe this FNN in detail later in this book.

1.2.3 RBF Neural Networks

The RBF neural network [91][219] is widely used for function approximation, interpolation, density estimation, classification, etc. For detailed theory and applications of other types of neural networks, readers may consult various textbooks on neural networks, e.g., [133][339].

RBF neural networks were first proposed in [33][245]. RBF neural networks [22] are a special class of neural networks in which the activation of a hidden neuron (hidden unit) is determined by the *distance* between the input vector and a prototype vector. Prototype vectors refer to centers of clusters obtained during RBF training. Usually, three kinds of distance metrics can be used in

RBF neural networks, such as Euclidean, Manhattan, and Mahalanobis distances. Euclidean distance is used in this book. In comparison, the activation of an MLP neuron is determined by a dot-product between the input pattern and the weight vector of the neuron. The dot-product is equivalent to the Euclidean distance only when the weight vector and all input vectors are normalized, which is not the case in most applications.

Usually, the RBF neural network consists of three layers, i.e., the input layer, the hidden layer with Gaussian activation functions, and the output layer. The architecture of the RBF neural network is shown in Fig. 1.3. The RBF neural network provides a function $Y : R^n \rightarrow R^M$, which maps n -dimensional input patterns to M -dimensional outputs ($\{(X_i, Y_i) \in R^n \times R^M, i = 1, 2, \dots, N\}$). Assume that there are M classes in the data set. The m th output of the network is as follows:

$$y_m(X) = \sum_{j=1}^K w_{mj} \phi_j(X) + w_{m0} b_m. \quad (1.5)$$

Here X is the n -dimensional input pattern vector, $m = 1, 2, \dots, M$, and K is the number of hidden units. M is the number of classes (outputs). w_{mj} is the weight connecting the j th hidden unit to the m th output node. b_m is the bias. w_{m0} is the weight connecting the bias and the m th output node.

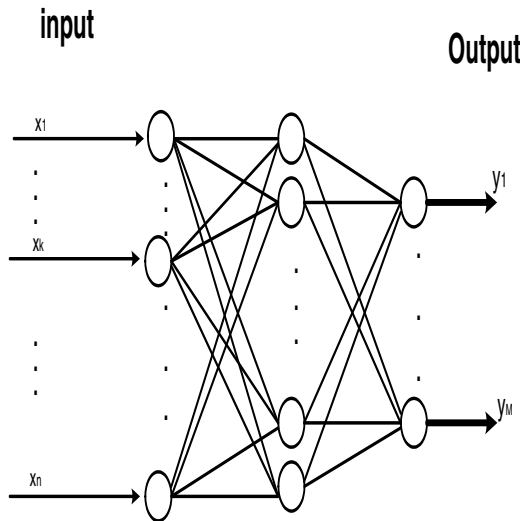


Fig. 1.3. Architecture of an RBF neural network. (© 2005 IEEE) We thank the IEEE for allowing the reproduction of this figure, first appeared in [104].

The radial basis activation function $\phi(x)$ of the RBF neural network distinguishes it from other types of neural networks. Several forms of activation functions have been used in applications:

1.

$$\phi(x) = e^{-x^2/2\sigma^2}, \quad (1.6)$$

2.

$$\phi(x) = (x^2 + \sigma^2)^{-\beta}, \quad \beta > 0, \quad (1.7)$$

3.

$$\phi(x) = (x^2 + \sigma^2)^\beta, \quad \beta > 0, \quad (1.8)$$

4.

$$\phi(x) = x^2 \ln(x); \quad (1.9)$$

here σ is a parameter that determines the smoothness properties of the interpolating function.

The Gaussian kernel function and the function (Eq. (1.7)) are localized functions with the property that $\phi \rightarrow 0$ as $|x| \rightarrow \infty$. One-dimensional Gaussian function is shown in Fig. 1.4. The other two functions (Eq. (1.8), Eq. (1.9)) have the property that $\phi \rightarrow \infty$ as $|x| \rightarrow \infty$.

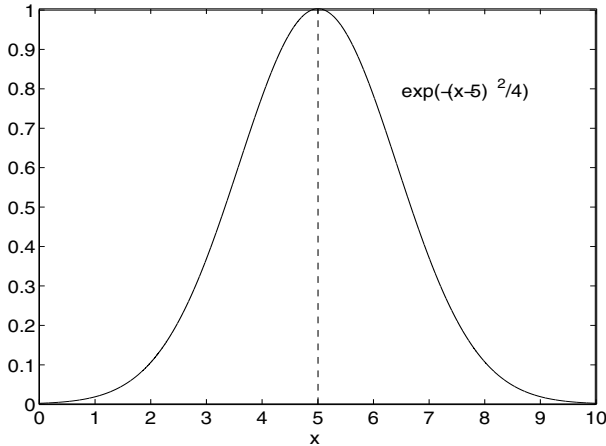


Fig. 1.4. Bell-shaped Gaussian Profile: The kernel possesses the highest response at the center $x = 5$ and degrades to zero quickly

In this book, the activation function of RBF neural networks is the Gaussian kernel function. $\phi_j(X)$ is the activation function of the j th hidden unit:

$$\phi_j(X) = e^{-\|\mathbf{X} - \mathbf{C}_j\|^2 / 2\sigma_j^2}, \quad (1.10)$$

where \mathbf{C}_j and σ_j are the center and the width for the j th hidden unit, respectively, which are adjusted during learning. When calculating the distance between input patterns and centers of hidden units, Euclidean distance measure is employed in most RBF neural networks.

RBF neural networks are able to make an exact interpolation by passing through every data point $\{X_i, Y_i\}$. In practice, noise is often present in data sets and an exact interpolation may not be desirable. Proomhead and Lowe [33] proposed a new RBF neural network model to reduce computational complexity, i.e., the number of radial basis functions. In [219], a smooth interpolating function is generated by the RBF network with a reduced number of radial basis functions.

Consider the following two major function approximation problems:

(a) target functions are known. The task is to approximate the known function by simpler functions, such as Gaussian functions,

(b) target functions are unknown but a set of samples $\{x, y(x)\}$ are given. The task is to approximate the function y .

RBF neural networks with free adjustable radial basis functions or prototype vectors are universal approximators, which can approximate any continuous function with arbitrary precision if there are sufficient hidden neurons [237][282]. The domain of y can be a finite set or an infinite set. If the domain of y is a finite set, RBF neural networks deal with classification problems [241].

The RBF neural network as a classifier differs from the RBF neural network as an interpolation tool in the following aspects [282]:

1. The number of kernel functions in an RBF classifier model is usually much fewer than the number of input patterns. The kernel functions are located in the centers of clusters of RBF classifiers. The clusters separate the input space into subspaces with hyper-ellipse boundaries.
2. In the approximation task, a global scaling parameter σ is used for all kernel functions. However, in the classification task, different σ 's are employed for different radial basis kernel functions.
3. In RBF network classifier models, three types of distances are often used. The Euclidean distance is usually employed in function approximation.

Generalization and the learning abilities are important issues in both function approximation and classification tasks. An RBF neural network can attain no errors for a given training data set if the RBF network has as many hidden neurons as the training patterns. However, the size of the network may be too large when tackling large data sets and the generalization ability of such a large RBF network may be poor. Smaller RBF networks may have better generalization ability; however, too small a RBF neural network will perform poorly on both training and test data sets. It is desirable to determine a training method which takes the learning ability and the generalization ability into consideration at the same time.

Three training schemes for RBF networks [282] are as follows:

- One-stage training

In this training procedure, only the weights connecting the hidden layer and the output layer are adjusted through some kind of supervised methods, e.g., minimizing the squared difference between the RBF neural network's output and the target output. The centers of hidden neurons are subsampled from the set of input vectors (or all data points are used as centers) and, typically, all scaling parameters of hidden neurons are fixed at a predefined real value [282] typically.

- Two-stage training

Two-stage training [17][22][36][264] is often used for constructing RBF neural networks. At the first stage, the hidden layer is constructed by selecting the center and the width for each hidden neuron using various clustering algorithms. At the second stage, the weights between hidden neurons and output neurons are determined, for example by using the linear least square (LLS) method [22]. For example, in [177][280], Kohonen's learning vector quantization (LVQ) was used to determine the centers of hidden units. In [219][281], the k -means clustering algorithm with the selected data points as seeds was used to incrementally generate centers for RBF neural networks. Kubat [183] used C.4.5 to determine the centers of RBF neural networks. The width of a kernel function can be chosen as the standard deviation of the samples in a cluster. Murata *et al.* [221] started with a sufficient number of hidden units and then merged them to reduce the size of an RBF neural network. Chen *et al.* [48][49] proposed a constructive method in which new RBF kernel functions were added gradually using an orthogonal least square learning algorithm (OLS). The weight matrix is solved subsequently [48][49].

- Three-stage training

In a three-stage training procedure [282], RBF neural networks are adjusted through a further optimization after being trained using a two-stage learning scheme. In [73], the conventional learning method was used to generate the initial RBF architecture, and then the conjugate gradient method was used to tune the architecture based on the quadratic loss function.

An RBF neural network with more than one hidden layer is also presented in the literature. It is called the multi-layer RBF neural network [45]. However, an RBF neural network with multiple layers offers little improvement over the RBF neural network with one hidden layer. The inputs pass through an RBF neural network and form subspaces of a local nature. Putting a second hidden layer after the first hidden layer will lead to the increase of the localization and the decrease of the valid input signal paths accordingly [138]. Hirasawa *et al.* [138] showed that it was better to use the one-hidden-layer RBF neural network than using the multi-layer RBF neural network.

Given N patterns as a training data set, the RBF neural network classifier may obtain 100% accuracy by forming a network with N hidden units, each of

which corresponds to a training pattern. However, the 100% accuracy in the training set usually cannot lead to a high classification accuracy in the test data set (the unknown data set). This is called the generalization problem. An important question is: ‘how do we generate an RBF neural network classifier for a data set with the fewest possible number of hidden units and with the highest possible generalization ability?’.

The number of radial basis kernel functions (hidden units), the centers of the kernel functions, the widths of the kernel functions, and the weights connecting the hidden layer and the output layer constitute the key parameters of an RBF classifier. The question mentioned above is equivalent to how to optimally determine the key parameters. Prior knowledge is required for determining the so-called ‘sufficient number of hidden units’. Though the number of the training patterns is known in advance, it is not the only element which affects the number of hidden units. The data distribution is another element affecting the architecture of an RBF neural network. We explore how to construct a compact RBF neural network in the latter part of this book.

1.2.4 Support Vector Machines

Support vector machines (SVMs) [62][326][327] have been widely applied to pattern classification problems [46][79][148][184][294] and non-linear regressions [230][325]. SVMs are usually employed in pattern classification problems. After SVM classifiers are trained, they can be used to predict future trends. We note that the meaning of the term *prediction* is different from that in some other disciplines, e.g., in time-series prediction where prediction means guessing future trends from past information. Here, ‘prediction’ means supervised classification that involves two steps. In the first step, an SVM is trained as a classifier with a part of the data in a specific data set. In the second step (i.e., prediction), we use the classifier trained in the first step to classify the rest of the data in the data set.

The SVM is a statistical learning algorithm pioneered by Vapnik [326][327]. The basic idea of the SVM algorithm [29][62] is to find an optimal hyper-plane that can maximize the margin (a precise definition of margin will be given later) between two groups of samples. The vectors that are nearest to the optimal hyper-plane are called support vectors (vectors with a circle in Fig. 1.5) and this algorithm is called a support vector machine. Compared with other algorithms, SVMs have shown outstanding capabilities in dealing with classification problems. This section briefly describes the SVM.

Linearly Separable Patterns

Given l input vectors $\{\mathbf{x}_i \in R^n, i = 1, \dots, l\}$ that belong to two classes, with desired output $y_i \in \{-1, 1\}$, if there exists a hyper-plane

$$\mathbf{w}^T \mathbf{x} + b = 0 \quad (1.11)$$

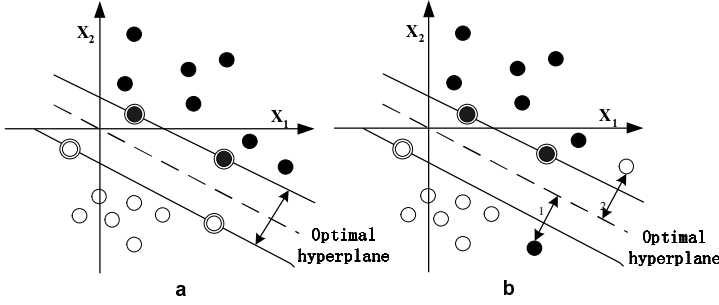


Fig. 1.5. An optimal hyper-plane for classification in a two-dimensional case, for (a) linearly separable patterns and (b) linearly non-separable patterns.

that separates the two classes, that is,

$$\mathbf{w}^T \mathbf{x}_i + b \geq 0, \text{ for all } i \text{ with } y_i = +1, \quad (1.12)$$

$$\mathbf{w}^T \mathbf{x}_i + b < 0, \text{ for all } i \text{ with } y_i = -1, \quad (1.13)$$

then we say that these patterns are *linearly separable*. Here \mathbf{w} is a weight vector and b is a bias. By rescaling \mathbf{w} and b properly, we can change the two inequalities above to:

$$\mathbf{w}^T \mathbf{x}_i + b \geq 1, \text{ for all } i \text{ with } y_i = +1, \quad (1.14)$$

$$\mathbf{w}^T \mathbf{x}_i + b \leq -1, \text{ for all } i \text{ with } y_i = -1. \quad (1.15)$$

Or,

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq -1. \quad (1.16)$$

There are two parallel hyper-planes:

$$H1: \mathbf{w}^T \mathbf{x} + b = 1, \quad (1.17)$$

$$H2: \mathbf{w}^T \mathbf{x} + b = -1. \quad (1.18)$$

The distance ρ between $H1$ and $H2$ is defined as the *margin* between the two classes (Fig. 1.5a). According to the standard result of the distance between the origin and a hyper-plane, we can figure out that the distances between the origin and $H1$ and $H2$ are $|b - 1|/\|\mathbf{w}\|$ and $|b + 1|/\|\mathbf{w}\|$, respectively. The sum of these two distances is ρ , because $H1$ and $H2$ are parallel. Therefore,

$$\rho = 2/\|\mathbf{w}\|. \quad (1.19)$$

The objective is to maximize the margin between the two classes, i.e., to minimize $\|\mathbf{w}\|$. This objective is equivalent to minimizing the cost function:

$$\psi = \frac{1}{2} \|\mathbf{w}\|^2. \quad (1.20)$$

Then, this optimization problem subject to the constraint (1.16) can be solved using Lagrange multipliers. The Lagrange function is

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^l \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1], \quad (1.21)$$

where $\alpha_i, i = 1, 2, \dots, l$ are the Lagrange multipliers. Differentiating this Lagrange function, we obtain

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = \mathbf{0}, \quad (1.22)$$

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \alpha} = 0. \quad (1.23)$$

Considering the Wolfe's dual [89], we can obtain a dual problem of the primal one:

$$\text{maximize: } Q(\alpha) = \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j, \quad (1.24)$$

subject to:

$$\sum_{i=1}^l \alpha_i y_i = 0, \quad (1.25)$$

$$\alpha_i \geq 0. \quad (1.26)$$

From this dual problem, the optimal weight vector, i.e., \mathbf{w}_o and the optimal Lagrange multipliers, i.e., $\alpha_{o,i}$ of the optimal hyper-plane can be obtained:

$$\mathbf{w}_o = \sum_{i=1}^l \alpha_{o,i} y_i \mathbf{x}_i. \quad (1.27)$$

Linearly Non-separable Patterns

If the vectors $\{\mathbf{x}_i \in R^n, i = 1, \dots, l\}$ cannot be linearly separated, we would like to slacken the constraints described by (1.16). Here we introduce a group of slack variables, i.e., ξ_i :

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad (1.28)$$

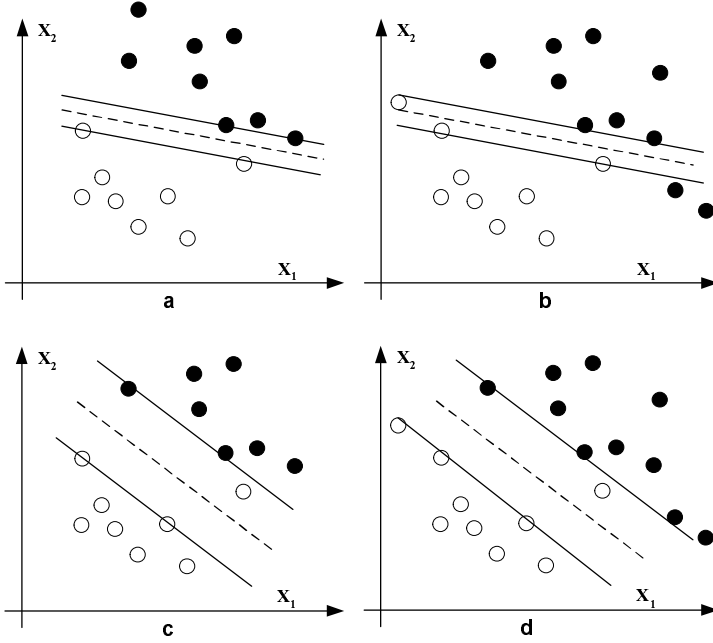


Fig. 1.6. The influence of C on the performance of the classifier. (a) a classifier with a large C (small margin); (b) an overfitting classifier; (c) a classifier with a small C (large margin); (d) a classifier with a proper C .

$$\xi_i \geq 0. \quad (1.29)$$

In fact, ξ_i is the distance between the training example \mathbf{x}_i and the optimal hyper-plane (Fig. 1.5b). For $0 \leq \xi_i \leq 1$, \mathbf{x}_i falls in the region between the two hyper-planes, i.e., $H1$ and $H2$, but on the correct side of the optimal hyper-plane. However, for $\xi_i > 1$, \mathbf{x}_i falls on the wrong side of the optimal hyper-plane.

Since it is expected that the optimal hyper-plane can maximize the margin between the two classes and minimize the errors, the cost function from Eq. (1.20) is rewritten:

$$\psi = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^l \xi_i, \quad (1.30)$$

where C is a positive factor. This cost function must satisfy the constraints Eq. (1.28) and Eq. (1.29). There is also a dual problem:

$$\text{maximize: } Q(\alpha) = \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j, \quad (1.31)$$

subject to:

$$\sum_{i=1}^l \alpha_i y_i = 0, \quad (1.32)$$

$$C \geq \alpha_i \geq 0. \quad (1.33)$$

From this dual problem, the optimal weight vector, i.e., \mathbf{w}_o and the optimal Lagrange multipliers, i.e., $\alpha_{o,i}$ of the optimal hyper-plane can be obtained. They are the same as their counterparts in Eq. (1.27), except that the constraints change to Eq. (1.32) and (1.33).

In general, C controls the trade-off between the two goals of the binary SVM, i.e., to maximize the margin between the two classes and to separate the two classes well. When C is small, the margin between the two classes is large, but it may make more mistakes in training patterns. Or, alternatively, when C is large, the SVM is likely to make fewer mistakes in training patterns; however, the small margin makes the network vulnerable for overfitting. Figure 1.6 depicts the functionality of the parameter C , which has a relatively large impact on the performance of the SVM. Usually, it is determined experimentally for a given problem.

A Binary Non-linear SVM Classifier

According to [65], if a non-linear transformation can map the input feature space into a new feature space whose dimension is high enough, the classification problem is more likely to be linearly solved in this new high-dimensional space. In view of this theorem, the *non-linear* SVM algorithm performs such a transformation to map the input feature space to a new space with much higher dimension. Actually, other kernel learning algorithms, such as radial basis function (RBF) neural networks, also perform such a transformation for the same reason. After the transformation, the features in the new space are classified using the optimal hyper-plane we constructed in the previous sections. Therefore, using this non-linear SVM to perform classification includes the following two steps:

1. Mapping the input space into a much higher dimensional space with a non-linear kernel function.
2. Performing classification in the new high-dimensional space by constructing an optimal hyper-plane that is able to maximize the margin between the two classes.

Combining the transformation and the linear optimal hyper-plane, we formulate the mathematical descriptions of this non-linear SVM as follows.

It is supposed to find the optimal values of weight vector \mathbf{w} and bias b such that they satisfy the constraint:

$$y_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \quad (1.34)$$

$$\xi_i \geq 0. \quad (1.35)$$

where $\phi(\mathbf{x}_i)$ is the function mapping the i th pattern vector to a potentially much higher dimensional feature space. The weight vector \mathbf{w} and the slack variables ξ_i should minimize the cost function:

$$\psi = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^l \xi_i, \quad (1.36)$$

This optimization problem is very similar to the problem we have dealt with using a linear optimal hyper-plane. The only difference is that the input vectors \mathbf{x}_i have been replaced by $\phi(\mathbf{x}_i)$.

To solve this optimization problem, a similar procedure is followed as before. Through constructing the Lagrange function and differentiating it, a dual problem is obtained as below:

$$\text{maximize: } Q(\alpha) = \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j), \quad (1.37)$$

subject to:

$$\sum_{i=1}^l \alpha_i y_i = 0, \quad (1.38)$$

$$C \geq \alpha_i \geq 0, \quad (1.39)$$

where $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel function:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j). \quad (1.40)$$

From this dual problem, the optimal weight vector i.e., \mathbf{w}_o and the optimal Lagrange multipliers, i.e., $\alpha_{o,i}$ of the optimal hyper-plane can be obtained:

$$\mathbf{w}_o = \sum_{i=1}^l \alpha_{o,i} y_i \mathbf{x}_i. \quad (1.41)$$

The optimal hyper-plane that discriminates different classes is:

$$\mathbf{w}_o^T \phi(\mathbf{x}) + b = 0. \quad (1.42)$$

One of the most commonly used kernel functions is the polynomial kernel:

$$K(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}^T \mathbf{x}_i + 1)^p, \quad (1.43)$$

where p is a constant specified by users. Another kind of widely used kernel function is the radial basis function:

$$K(\mathbf{x}, \mathbf{x}_i) = e^{-\gamma \|\mathbf{x} - \mathbf{x}_i\|^2}, \quad (1.44)$$

where γ is also a constant specified by users. According to its mathematical description, the structure of an SVM is shown in Fig. 1.7.

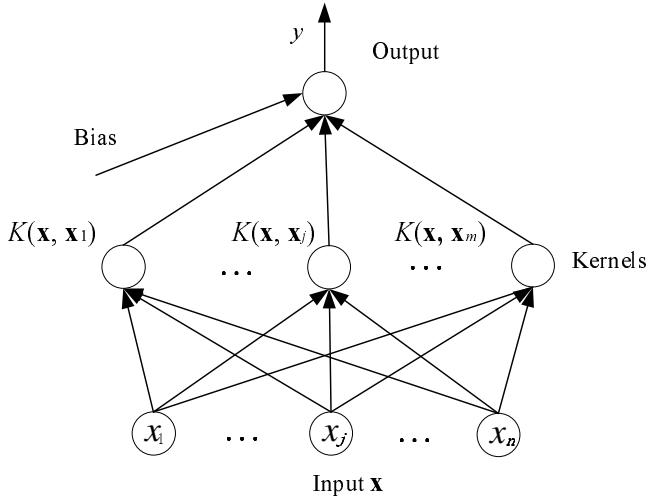


Fig. 1.7. The structure of an SVM.

1.2.5 Genetic Algorithms

Genetic algorithms (GAs) are motivated by the natural evolutionary process. The basic concepts in GAs are as follows. Solutions of the problem at hand are encoded in chromosomes or individuals. An initial population of individuals is generated at random or heuristically. The operators in GAs include selection, crossover, and mutation. To generate a new generation, chromosomes are selected according to their fitness scores, i.e., a predefined quality criterion used for evaluating solutions of a problem. The selection operator gives preference to better individuals as parents for the next generation. The crossover operator and the mutation operator are used to generate offspring from the parents. A crossover site is randomly chosen in the parents. The two bit strings in the two individuals are exchanged up to the crossover site. For example, suppose that parents $I_1 = 0001100$ and $I_2 = 1110000$ are selected for generating new offspring. After applying the crossover operator, we obtain $I'_1 = 0010100$ and $I'_2 = 1101000$ with two crossover points at the third

and fourth bits. By exchanging portions of good individuals, crossover may produce even better individuals. The mutation operator is used to prevent premature convergence to local optima. It is implemented by flipping bits at random with a mutation probability.

GAs are specially useful under the following circumstances:

- the problem space is large, complex;
- prior knowledge is scarce;
- it is difficult to determine a machine learning model to solve the problem due to complexities in constraints and objectives;
- traditional search methods perform badly.

The steps to apply the basic GA as a problem-solving model are as follows:

1. figure out a way to encode solutions of the problem according to domain knowledge and required solution quality;
2. randomly generate an initial population of chromosomes which corresponds to solutions of the problem;
3. calculate the fitness of each chromosome in the population pool;
4. select two parental chromosomes from the population pool to produce offspring by crossover and mutation operators;
5. go to step 3, and iterate until an optimal solution is found.

The basic genetic algorithm is simple but powerful in solving problems in various areas. In addition, the basic GA could be modified to meet requirements of diverse problems by tuning the basic operators. For a detailed discussion of variations of the basic GA, as well as other techniques in a broader category called evolutionary computation, see text books, such as [10][86].

1.3 How This Book is Organized

In Chap. 1, data mining tasks and conventional data mining methods are introduced. Classification and clustering tasks are explained, with emphasis on the classification task. An introduction to data mining methods is presented.

In Chap. 2, a wavelet multi-layer perceptron neural network is described for predicting temporal sequences. The multi-layer perceptron neural network has its input signal decomposed to various resolutions using a wavelet transformation. The time frequency information which is normally hidden is exposed by the wavelet transformation. Based on the wavelet transformation, less important wavelets are eliminated. Compared with the conventional MLP network, the wavelet MLP neural network has less performance swing sensitivity to weight initialization. In addition, we describe a cost-sensitive MLP in which errors in prediction are biased towards ‘important’ classes. Since different prediction errors in different classes usually lead to different costs, it is worthwhile discussing the cost-sensitive problem. In experimental results,

it is shown that the recognition rates for the ‘important’ classes (with higher cost) are higher than the recognition rates for the ‘less important’ classes.

In Chap. 3, the FNN is described. This FNN that we proposed earlier combines the powerful features of initial fuzzy model self-generation, fast input selection, partition validation, parameter optimization, and rule-base simplification. The structure and learning procedure are introduced first. Then, we describe the implementation and functionality of the FNN. Synthetic databases and microarray data are used to demonstrate the fuzzy neural network proposed earlier [59][349]. Experimental results are compared with the pruned feedforward crisp neural network and decision tree approaches.

Chapter 4 describes how to construct an RBF neural network that allows for large overlaps between clusters with the same class label, which reduces the number of hidden units without degrading the accuracy of the RBF neural network. In addition, we describe a new method dealing with unbalanced data. The method is based on the modified RBF neural network. Weights inversely proportional to the number of patterns of classes are given to each class in the mean squared error (MSE) function.

In Chap. 5, DDR methods, including feature selection and feature extraction techniques, are reviewed first. A novel algorithm for attribute importance ranking, i.e., the separability and correlation measure (SCM), is then presented. Class-separability measure and attribute-correlation measure are weighted to produce a combined evaluation for relative attribute importance. The top-down search and the bottom-up search are explored, and their difference in attribute ranking is presented. The attribute ranking algorithm with class information is compared with other attribute ranking methods. Data dimensionality is reduced based on attribute ranking results.

Data dimensionality reduction is then performed by combining the SCM method and RBF classifiers. In the DDR method, there are a fewer number of candidate feature subsets to be inspected compared with other methods, since attribute importance is ranked first by the SCM method. The size of a data set is reduced and the architecture of the RBF classifier is simplified. Experimental results show the advantages of the DDR method.

In Chap. 6, reviews of existing class-dependent feature selection techniques are presented first. The fact that different features might have different discrimination capabilities for separating one class from the other classes is adopted. For a multi-class classification problem, each class has its own specific feature subset as inputs of the RBF neural network classifier. The novel class-dependent feature selection algorithm is based on RBF neural networks and the genetic algorithm (GA).

In Chap. 7, reviews of rule extraction work in the literature are presented first. Several new rule extraction methods are described based on the simplified RBF neural network classifier in which large overlaps between clusters of the same class are allowed. In the first algorithm, A GA combined with an RBF neural network is used to extract rules. The GA is used to determine the intervals of each attribute as the premise of the rules. In the second algorithm,

rules are extracted directly based on simplified RBF neural networks using gradient descent. In the third algorithm, the DDR technique is combined with rule extraction. Rules with a fewer number of premises (attributes) and higher rule accuracy are obtained. In the fourth algorithm, class-dependent feature selection is used as a preprocessing procedure of rule extraction. The results from the four algorithms are compared with other algorithms.

In Chap. 8, a hybrid neural network predictor is described for protein secondary structure prediction (PSSP). The hybrid network is composed of the RBF neural network and the MLP neural network. Experiments show that the performance of the hybrid network has reached a comparable performance with the existing leading method.

In Chap. 9, support vector machine classifiers are used to deal with two bioinformatics problems, i.e., cancer diagnosis based on gene expression data and protein secondary structure prediction.

Chapter 10 describes a rule extraction algorithm RuleXSVM that we proposed earlier [108]. Decisions made by a non-linear SVM classifier are decoded into linguistic rules based on the support vectors and decision functions according to a geometrical relationship.



<http://www.springer.com/978-3-540-24522-3>

Data Mining with Computational Intelligence

Wang, L.; Fu, X.

2005, XI, 276 p., Hardcover

ISBN: 978-3-540-24522-3