# Springer Handbook of Computational Intelligence

Janusz Kacprzyk, Witold Pedrycz (Eds.)

With 534 Figures and 115 Tables

Springer

*Editors*
Janusz Kacprzyk
Polish Academy of Sciences
Systems Research Inst.
ul. Newelska 6
01-447 Warsaw, Poland
kacprzyk@ibspan.waw.pl

Witold Pedrycz
University of Alberta
Dep. Electrical and Computer Engineering
116 Street 9107
T6J 2V4, Edmonton, Alberta, Canada
wpedrycz@ualberta.ca

ISBN: 978-3-662-43504-5 e-ISBN: 978-3-662-43505-2
DOI 10.1007/978-3-662-43505-2
Springer Dordrecht Heidelberg London New York

Library of Congress Control Number: 2015936335



Production and typesetting: le-tex publishing services GmbH, Leipzig
Senior Manager Springer Handbook: Dr. W. Skolaut, Heidelberg
Typography and layout: schreiberVIS, Seeheim
Illustrations: Hippmann GbR, Schwarzenbruck
Cover design: eStudio Calamar Steinen, Barcelona
Cover production: WMXDesign GmbH, Heidelberg
Printing and binding: Printer Trento s.r.l., Trento

Printed on acid free paper

Springer is part of Springer Science+Business Media (www.springer.com)

# 29. Machine Learning

James T. Kwok, Zhi-Hua Zhou, Lei Xu

This tutorial provides a brief overview of a number of important tools that form the crux of the modern machine learning toolbox. These tools can be used for supervised learning, unsupervised learning, reinforcement learning and their numerous variants developed over the years. Because of the lack of space, this survey is not intended to be comprehensive. Interested readers are referred to conference proceedings such as Neural Information Processing Systems (NIPS) and the International Conference on Machine Learning (ICML) for the most recent advances.





## 29.1 Overview

Machine learning represents one of the most prolific developments in modern artificial intelligence. It provides a new generation of computational techniques and tools that support understanding and extraction of useful knowledge from complicated data sets. So what is machine learning? *Simon* [29.1] defined machine learning as:

> *changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more effectively the next time.*

Hence, fundamentally, the emphasis of machine learning is on the system's ability to adapt or change. Typically, this is in response to some form of experience provided to the system. After learning or adaptation, the system is expected to have better future performance on the same or a related task.

Over the past decades, machine learning has grown from a few toy applications to being almost everywhere. It is now being applied to numerous real-world applications. For example, the control of autonomous robots that can navigate on their own, the filtering of spam from mailboxes, the recognition of characters

from handwriting, the recognition of speech on mobile devices, the detection of faces in digital cameras, and so on. Indeed, one can find applications of machine learning from everyday consumer products to advanced information systems in corporations. Studies of machine learning may be overviewed from either the perspective of learning intelligent systems or that of a machine learning toolbox. For the former, learning is considered as a process of an intelligent system for coordinately solving three levels of inverse problems, namely problem solving for making pattern recognition and various other tasks, parameter learning for estimating unknown parameters in the system, and model selection for shaping system configuration with an appropriate scale or complexity to describe regularities underlying a finite size of samples. Different learning approaches are featured by differences in one or more of three ingredients, namely as a learner that has an appropriate system configuration, a theory that guides learning, and an algorithm or dynamic procedure that implements learning. Examples of studies from this prospect are recently overviewed in [29.2, 3], and will not be further addressed in this chapter. Instead, this chapter aims at a tutorial on studies of machine learning from the second prospect, that is, on those important collections pooled in the machine learning toolbox for decades. Actually, the current prosperity of machine learning comes from not only further developments of the classical statistical modeling and neural network learning, but also emerging achievements of machine learning and data mining in recent decades. Due to limited space, the focus of this tutorial will be particularly placed on those advancements made in the last two decades or so.

Classically, there are three basic learning paradigms, namely, supervised learning (Sect. 29.2), unsupervised learning (Sect. 29.3), and reinforcement learning (Sect. 29.4). In supervised learning, the learner is provided with a set of inputs together with the corresponding desired outputs. This is similar to the familiar human learning process for pattern recognition, in which a teacher provides examples to teach children to recognize different objects (say, for example, animals). Such a pattern recognition task is featured by data with each input sample associated with a label, namely labeled data. In the current literature on machine learning, the term labeled data is even generally used to refer data with each input associated with an output beyond simply a label, which is also adopted in this chapter. Section 29.2 provides not only a tutorial on basic issues of supervised learning but also an overview on a number of interesting topics developed in recent years. The coverage of this section is not complete, e.g., it does not cover the supervised learning studies in the literature on neural networks. Interested readers are referred to a number of survey papers, e.g., especially those on multi-layer perceptrons and radial basis functions [29.4, 5].

Unlike supervised learning, the tasks of unsupervised learning are featured by data that consist of only inputs, namely, the data is unlabeled and there is no longer the presence of a teacher. Unsupervised learning aims at finding certain dependence structure underlying data via optimizing a learning principle. Considering different types of structures, studies include not only classic topics of data clustering, subspace, and topological maps, but also emerging topics of learning latent factor models, hidden state–space models, and hierarchical structures. Section 29.3 also consists of two parts. The first part provides a tutorial on three classic topics, while the second part makes an overview on emerging topics. Extensive studies have been made on unsupervised learning for many decades. Instead of seeking a complete coverage, this section focuses on a tutorial on fundamentals and an overview on interesting developments of recent years, mainly based on a more systematic overview [29.6]. Further, readers are referred to several recent survey papers, e.g., [29.7] for an overview on 50 years of studies beyond k-means for data clustering, [29.8, 9] for subspace and manifold learning, and [29.10] for topological maps.

The third paradigm is reinforcement learning. Upon observing the current environment and obtaining some input (if any), the learner makes an action and changes to a new environment, receiving an evaluation (award or punish) value about the action. A learning process makes a series of actions with the received total award maximized. Different to unsupervised learning, the learner gets a guidance from an external evaluation. Also unlike supervised learning in which the teacher clearly specifies the output that corresponds to an input, in reinforcement learning the learner is only provided with an evaluative value about the action made. Section 29.4 starts at giving a tutorial on basic issues of reinforcement learning, especially temporal difference TD learning and Q-learning, plus improvements on the Q-learning with the help of some unsupervised learning methods.

Besides these three basic learning paradigms, many more variants have been developed in recent years because of the advances in machine learning. Some of these will also be described in this tutorial. They are of-

a hybrid of the previous learning paradigms. A very popular variant is semi-supervised learning (Sect. 29.5), which uses both labeled data (as in supervised learning) and unlabeled data (as in unsupervised learning) for training. This is advantageous as labeled data typically are expensive and involve tedious human effort, while a large amount of unlabeled data can often be obtained in an inexpensive manner (e.g., simply downloadable from the web). Another hybrid of supervised learning and unsupervised learning is discriminative clustering. Here, one adopts a cost function originally used for supervised learning as a clustering criterion. A well-known example in this category is called maximum margin clustering [29.11–13], which tries to maximize the margin (used as a criterion in constructing the highly successful supervised learning model: support vector machine) between clusters.

Moreover, instead of just constructing one learner from training data, one can construct a set of learners and combine them to solve the problem. This approach, known as *ensemble learning* [29.14], has become very popular in recent years and will be discussed in more detail in Sect. 29.6. Finally, before learning can proceed, the data need to be appropriately represented by a set of features. In many real-world data sets, there are often a large number of features, many of which are abundant or irrelevant. Feature selection and extraction aim at automatically extracting the good features and removing the bad ones, and this will be covered in Sect. 29.7.



## 29.2 Supervised Learning

A supervised learner is provided with some labeled data (often called *training data*). This consists of a set of *training samples*, each of which is an input together with the corresponding desired outputs. Hence, the first step in machine learning is to collect these training samples. Moreover, as each training sample needs to be represented in a form amendable by the computer algorithm, one has to define a set of *features*. As an example, consider the task of recognizing handwritten characters on an envelope. To construct the training samples, obviously one first has to collect a number of envelopes with handwritten characters on. Then, the characters on each envelope have to be separated from each other. This can be performed either manually or automatically by some image segmentation algorithm. Afterwards, each character is a block of pixels (typically rectangular). A simple feature representation will be to use the intensities of these raw pixels. Each input is represented as a vector of feature values, and this vector is called the *feature vector*. Obviously, it is important to have a good set of features to work with. The presence of bad features may confuse the learning algorithm and makes learning more difficult. For example, in the context of character recognition, the color of the ink is not relevant to the identity of the character and so can be considered as a bad feature. Depending on the domain knowledge, more sophisticated features can be manually defined. It is desirable that good features can be automatically extracted and bad features automatically removed. More details on these feature selection/extraction algorithms will be covered in Sect. 29.7. Finally, each character on the envelope has to be manually labeled.

In practice, as the real-world data are often dirty, a significant amount of time may have to be spent on data pre-processing in order to create the training data. There are many forms of *dirty* data. For example, it can be incomplete in that certain attribute values (e.g., occupation) may be lacking; it can contain outliers or errors (e.g., the *salary* is negative); parts of it may be inconsistent (e.g., the customer's age is 42 but his/her birthday is *03/07/2012*); it may also be redundant in that there are duplicate records or unnecessary attributes. All these problems may be due to faulty or careless data collection, human/hardware/software problems, errors in data transmission; or that the data may have come from a number of different data sources. In all cases, data preprocessing can have a significant impact on the resultant machine learning system, as no quality data implies no quality learning results!

### 29.2.1 Classification and Regression

The two main goals in supervised learning are (i) classification, which aims at assigning the input pattern to different categories (also called classes or labels); and (ii) regression, which aims at predicting a real value or vector associated with the input. The basic idea and the training/testing procedures in regression are similar to those in classification. Hence, we will mainly focus on the classification problem in the sequel.

The simplest case for classification is binary classification, in which there are only two classes. Examples include classifying an email as spam or non-spam; and classifying an image as face or non-face. For each sample, the supervised learner examines the feature values in that sample and predicts the class that the sample belongs to. Essentially, the supervised learner partitions the whole feature space (the space of all possible feature value combinations) into two regions, one for each class. The boundary is called the *decision boundary*. A wide variety of models can be used to construct this decision boundary. A simple example is the linear classifier, which creates a linear boundary. Depending on the task, the linear classifier may be too simple to differentiate the two classes. Then, one can also use a more complicated decision boundary, such as a quadratic surface, leading to a quadratic classifier. In machine learning, a large number of various models that are capable of producing nonlinear decision boundaries have been proposed. The most popular ones include the decision tree classifier, nearest neighbor classifiers, neural network classifiers, Bayesian classifiers, and support vector machines. Each of these models has some parameters that have to be adapted to the particular data set. For example, the parameters of a linear classifier include the weight on each feature (which controls the slope of the linear boundary) and a bias (which controls the offset). To estimate or train these parameters, one has to provide a training set, where the $i$-th training pattern $(x_i, y_i)$ consists of an input $x_i$ and the corresponding target output label $y_i$ (for regression problems, this $y_i$ is a real value or vector). The greater the amount of training data, intuitively the more accurate the learned model. However, since the training data in supervised learning are labeled, obtaining these output labels typically involve expensive and tedious human effort. Hence, recent machine learning algorithms also try to utilize data that are unlabeled, leading to the development of semi-supervised learning algorithms in Sect. 29.5.

Given the model, different strategies can be used to learn the model parameters so that it fits the training set (i. e., train the model). Parameter estimation and feature selection (Sect. 29.7) can sometimes be performed together. However, note that there is the danger of *overfitting*, which occurs when the model performs better than other models on the training data, but worse on the entire data distribution as it has captured the trends of the noise underlying the data. Often this happens when the model is excessively complex, such as when it has a lot more parameters than can be reliably estimated from the limited number of training patterns. To combat overfitting, one can constrain the model's freedom during training by adding a regularizer or Bayesian to the parameters or model beforehand. Alternatively, one can stop the learning procedure before convergence (*early stopping*) or remove part of the model when training is complete (*pruning*). If there are noisy training samples that significantly deviate from the underlying input–output trend, one can also perform outlier detection to first remove these outlying samples.

There are two general approaches to train the model parameters. The first approach treats the model as a generative model that defines how the data are generated (typically by using a probabilistic model). One can then maximize the likelihood by varying the parameters, or to maximize the posterior probability of the parameters given the training data. Alternatively, one can take a discriminative approach that directly considers how the output is related to the input. The parameters can be obtained by *empirical risk minimization*, which seeks the parameters which best fit the training data. The risk is dependent on the loss function, which measures the difference between the prediction and the target output. Let $y_i$ be the target output for sample $i$, and $\tilde{y}_i$ be the predicted output from the supervised learner. For classification problems, commonly used loss functions include the logistic loss $\ln(1+\exp(-y_i\tilde{y}_i))$ and the hinge loss $\max(0, 1-y_i\tilde{y}_i)$; and for regression problems, the most common loss function is the square loss $(y_i-\tilde{y}_i)^2$. However, in order to combat overfitting, it is better to perform *regularized risk minimization* instead of empirical risk minimization. Regularized risk consists of two components. The first component is the loss as in empirical risk minimization. The second component is a regularizer, which helps to control the model complexity and prevents overfitting. Various regularizers have been proposed. Let $w = [w_1, w_2, \ldots, w_d]^T$ be the vector of parameters. A popular regularizer is the $\ell_2$-norm of $w$, i. e.,

$$\|w\|_2^2 = \sum_{i=1}^{d} w_i^2 \,.$$

This leads to ridge regression when the linear model is used, and is commonly called *weight decay* in the neural networks literature. Instead of using the $\ell_2$-norm, one can use the $\ell_0$-norm $\|w\|_0$, which counts the number of nonzero $w_i$ in the model. However, this is nonconvex and the associated optimization is more difficult. A common way to alleviate this problem is by using the

$$= \sum_{i=1}^{n} |w_i| ,$$

... still convex (as for the $\ell_2$-norm) but can still ... sparse parameter solution (as for the $\ell_0$-norm). ... with the square loss on the linear model, this ... well-known lasso model.

... method, the classifier can be used to predict ... an unseen test sample. The underlying as- ... that this test sample comes from the same ... as that of the training samples. In this case, ... the trained classifier to be able to generalize ... new sample. This can also be formally de- ... generalization error bounds in computational ... theory.

... are multiple ways to measure the perfor- ... of a trained classifier. An obvious performance ... criterion is classification accuracy, which is ... fraction of test samples that are correctly classi- ... (by comparing the prediction obtained from the ... and the true class output of the test sample). ... mentioned above, because of the issue of overfit- ... it can be misleading to simply gauge classification ... on the training set. Instead, one can mea- ... classification accuracy on a separate validation set ... is used as a proxy for the underlying data distri- ..., or use cross-validation. Moreover, sometimes, ... the sample sizes of the two classes differ sig- ..., this accuracy may again be misleading, as ... may attain an apparently high accuracy by sim- ... predicting the test sample to belong to the majority ... In these cases, other measures such as precision, ... and F-measure may be more useful. Moreover, ... the classifier's accuracy is often an important cri- ..., other aspects may also be important, such as the ... and testing of computational complexities (in- ... with time and space), user-friendliness (e.g., is ... model considered as a black-box or can it be ... conveyed and explained to the users), etc.

... binary classification assumes the presence of ... output classes, many real-world applications ... more than two (say, $K$), leading to a multi-class ... problem. There are two common ap- ... to reduce a multi-class classification problem ... classification problems, namely, the one-vs- ... (also called one-vs-all) approach and the one-vs- ... approach. In the one-vs-rest approach, $K$ binary ... are constructed, each one separating the sam- ... belonging to the $i$-th class from those that do not. On prediction, the test sample is sent to all the $K$ classifiers, and its label corresponds to the classifier with the highest output. In the one-vs-one approach, one binary classifier is built for each pair of outputs (e.g., outputs $i$ and $j$), and each classifier tries to discriminate samples belonging to the $i$ class from those belonging to the $j$ class. Thus, there are a total of $\frac{1}{2}K(K-1)$ classifiers. On prediction, the test sample is again sent to all the binary classifiers, and the class that receives the largest number of votes is output.

## 29.2.2 Other Variants of Supervised Learning

### Multi-Label Classification

While an instance can only belong to one and only one class in multi-class classification, an instance in multi-label classification can belong to multiple classes. Many real-world applications involve multi-label classification. For example, in text categorization, a document can belong to more than one category, such as government and health; in bioinformatics, a gene may be associated with more than one function, such as metabolism, transcription, and protein synthesis; and in image classification, an image may belong to multiple semantic categories, such as beach and urban. Note that the number of labels associated with an unseen instance is unknown and can also vary from instance to instance. Hence, this makes the multi-label classification problem more complicated than the multi-class classification problem. In the special case where the number of labels associated with each instance is always equal to one, obviously multi-label classification reduces to multi-class classification.

In general, multi-label classification algorithms can be divided into two categories: *problem transformation* and *algorithm adaptation* [29.15]. Problem transformation methods transform a multi-label classification problem into one or more single-label classification problems. The basic approach (called *binary relevance*) simply decomposes a multi-label problem with $K$ labels into $K$ binary classification problems, one for each label. In other words, the $i$-th classifier is a binary classifier that tries to decide whether the sample belongs to the $i$-th class. However, since this considers the labels independently, any possible correlations among labels will be ignored, leading to inferior performance in problems with highly correlated labels. More refined variants thus take the label correlation into account during training, a similar idea that is also exploited in multi-task learning (Sect. 29.2.2). On the

other hand, algorithm adaptation methods extend a specific learning algorithm for multi-label classification. The specific extension is thus tailor-made for each individual learning algorithm and less general. Example learning algorithms that have been extended in this way include boosting, decision trees, ensemble methods, neural networks, support vector machines, genetic algorithms, and the nearest-neighbor classifier. Recent surveys on the progress of multi-label classification and its use in different applications can be found in [29.15, 16].

In many applications, the labels are often organized in a hierarchy, either in the form of a tree (such as documents in Wikipedia) or as a directed acyclic graph (such as gene ontology). An instance is associated with a label only if it is also associated with the label's parent(s) in the hierarchy. Recently, progress has also been made in multi-label classification in these structured label hierarchies [29.17–19].

### Multi-Instance Learning

In multi-instance learning (MIL), the training set is composed of many *bags* each containing multiple instances, where a positive bag contains at least one positive instance, whereas a negative bag contains only negative instances; labels of the training bags are known, but labels of the instances are unknown. The task is to make predictions for labels of unseen bags. The multi-instance learning framework is illustrated in Fig. 29.1. Notice that the instances are described by the same feature set, rather than different feature sets.

The MIL learning framework originated from the study of drug activity [29.20], where a molecule with multiple low-energy shapes is known to be useful to make a drug, whereas it is unknown which shape is crucial. Later, many real tasks are found to be natural multi-instance learning problems. For example, in image retrieval if we regard each image patch as an instance, then the fact that an user is interested (or not interested) in an image implies that there are at least one patch (or no patch) that contains his/her interesting objects.

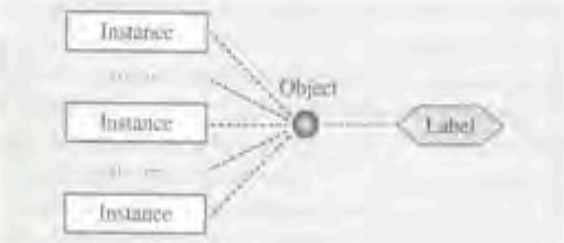


**Fig. 29.1** Illustration of multi-instance learning

Most MIL methods attempt to adapt single-instance supervised learning algorithms to the multi-instance representation by shifting their focus from discrimination on instances to discrimination on bags; there are also methods that try to adapt the multi-instance representation to single-instance algorithms by representation transformation [29.21]. Recently, it has been recognized that the instances in the bags should not be treated independently [29.22]; otherwise MIL is a special case of semi-supervised learning [29.23].

In addition to classification, multi-instance regression and clustering have been studied, and different versions of generalized multi-instance learning have been defined [29.24, 25]. To deal with complicated data objects that are associated with multiple labels simultaneously, a new framework, multi-instance, multi-label learning (MIML) [29.26], was developed recently.

Notice that the original MIL assumption implies that there exists a *key instance* in a positive bag; later, some other assumptions were introduced [29.27]. For example, some methods assumed that there is no key instance and every instance contributes to the bag label.

### Multi-View Learning

In many real tasks there is more than one feature set. For example, a video film can be described by audio features, image features, etc.; a web page can be described by features characterizing its own content, features characterizing its linked pages, etc. A classical routine is to take these features together and represent each instance using a concatenated feature vector. The different feature sets, however, usually convey information from different channels, and therefore, it may be better to consider the difference explicitly. This motivates multi-view learning, where each feature set is called a *view*.

Each instance in multi-view learning is represented by multiple feature vectors each in a different, usually non-overlapping feature set. Multi-view learning methods in supervised learning setting are closely related to studies of *information fusion, combining classifiers* [29.28–30], and *ensemble methods* [29.14]. A popular representative is to construct a model from each view, and then combine their predictions using *voting* or *averaging*. The models are often assigned with different weights, reflecting their different strength, reliability, and/or importance.

Multi-views make great sense when unlabeled data are considered. For example, it has been proved that when there are *sufficient and redundant views* (that is,

each view contains sufficient information for constructing a good model, and the two views are conditionally independent given the class label), *co-training* is able to boost the performance of any initial weak learner to an arbitrary performance using unlabeled data [29.31]. Later, it was found that such a process is beneficial even when the two views satisfy weaker assumptions, such as weak dependence, expansion, or large diversity [29.32–34], and when there are really sufficient and redundant views, even semi-supervised learning with a single labeled example is possible [29.35]. Moreover, in *active learning* where the learner actively selects some unlabeled instances to query their labels from an *oracle* (such as a human expert), it has been proved that multi-view learning enables exponential improvement of sample complexity in a setting close to real tasks [29.36], whereas previously it was believed that only polynomial improvement is possible.

### Multi-Task Learning

Many real-world problems involve the learning of a number of similar asks. Consider the simple example of learning to recognize the numeric digits 0–9. One can build ten separate classifiers, one for each digit. However, apparently these ten classifiers share some common features, e.g., many of the digits consist of loops and strokes. Hence, the ability to detect these higher latent features is of common interest to all these classifiers, and learning all these tasks together will allow them to borrow strength from each other. Moreover, when the number of training examples is rare for each task, most single-task learning methods may fail. By learning them together, better generalization performance can be obtained by harnessing the intrinsic task relationships. Consequently, this leads to the development of *multi-task learning* (MTL) [29.37]. These different tasks have different output spaces and can also have different input features. But it is also quite often that these different tasks share the same set of input features. In this case, the problem is similar to multi-label classification (Sect. 29.2.2).

A popular MTL approach is regularized multi-task learning (RMTL) [29.38, 39]. It assumes that the tasks are highly related, and encourages the parameters of all the tasks to be close. More specifically, let there be $T$ tasks and denote the parameter associated with the $t$-th task by $w_t$. RMTL assumes that all the $w_t$'s are close to some shared task $\bar{w}$, and that the $w_t$'s differ by each other only in a term $\Delta w_t$'s as $w_t = \bar{w} + \Delta w_t$. Hence, $\bar{w}$ represents the component that is shared by all the tasks, and thus can benefit from learning all the tasks together; while $\Delta w_t$ is the component that is specific to each individual task, and can be used to capture the individual variations. Alternatively, other MTL methods, such as multi-task feature learning (MTFL) [29.40], assumes that all the tasks lie in a shared low-dimensional space.

Moreover, tasks are supposed to form several clusters rather than from the same group. If such a task clustering structure is known, then a simple remedy is to constrain task sharing to be just within the same cluster [29.39, 41]. More generally, all the tasks are related in different degrees, which can be represented by a network of task relationships [29.42]. In this case, MTL can also be performed. In practice, however, such an explicit knowledge of task clusters/network may not be readily available.

A number of efforts have made towards identifying task relationships simultaneously during parameter learning, e.g., learning a low-dimensional subspace shared by most of the tasks [29.43], finding the correlations between tasks [29.44], and inferring the clustering structure [29.45, 46], as well as integrating low-rank and group-sparse structures for robust multi-task learning [29.47].

### Transfer Learning

As discussed in Sect. 29.1, traditionally, *machine learning* is defined as:

> *changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more effectively the next time.*

However, recently, there has been increasing interest in adapting a classifier/regressor trained in one task for use in another. This so-called *transfer learning* is particularly crucial when the target application is in short supply of labeled data. For example, it is very expensive to calibrate a WiFi localization model in a large-scale environment. To reduce re-calibration effort, we might want to adapt the localization model trained in one time period (source domain) for a new time period (target domain), or to adapt the localization model trained on a mobile device (source domain) for a new mobile device (target domain). However, the WiFi signal strength is a function of time, device, and other dynamic factors. Thus, transfer learning is used to adapt the distributions of WiFi data collected over time or across devices.

In general, transfer learning addresses the problem of how to utilize plentiful labeled data in a source do-

main to solve related but different problems in a target domain, even when the training and testing problems have different distributions or features. The success to transfer learning from one context to another context depends on how similar the learning task is to the transferred task. There are two main approaches to transfer learning. The first approach tries to learn a common set of features from both domains, which can then be used for knowledge transfer [29.48–50]. Intuitively, a good feature representation should be able to reduce the difference in distributions between domains as much as possible, while at the same time preserving important (geometric or statistical) properties of the original data. With a good feature representation, we can apply standard machine learning methods to train classifiers or regression models in the source domain for use in the target domain. The second approach to transfer learning is based on instances [29.51–53]. It tries to learn different weights on the source examples for better adaptation in the target domain. For example, in the *kernel mean matching* algorithm [29.52], instances in a reproducing kernel Hilbert space are re-weighted based on the theory of *maximum mean discrepancy*.



### Cost-Sensitive Learning

In many real tasks, the costs of making different types of mistakes are usually unequal. In such situations, maximizing the *accuracy* (or equivalently, minimizing the number of mistakes) may not provide the optimal decision. For example, two instances that each cost 10 dollars are less important than one instance that costs 50 dollars. Cost-sensitive learning methods attempt to minimize the *total cost* by reducing serious mistakes through sacrificing minor mistakes.

There are two types of misclassification costs, i.e., *example-dependent* or *class-dependent* cost. The former assumes that every example has its own misclassification cost, whereas the latter assumes that every class has its own misclassification cost. To obtain example-dependent cost is usually much more difficult in real practice, and therefore, most studies focus on class-dependent cost.

The essence of most cost-sensitive learning methods is *rescaling* (or *rebalance*), which tries to rebalance the classes such that the influence of each class in the learning process is in proportion to its costs. Suppose the cost of misclassifying the $i$-th class to the $j$-th class is $cost_{ij}$. For binary classification, it can be derived from the Bayes risk theory that the optimal rescaling ratio of the $i$-th class against the $j$-th class is $\tau_{ij} = \frac{cost_{ij}}{cost_{ji}}$ [29.54]. For multi-class problems, however, there is no direct solution to obtain the optimal rescaling ratios [29.55], and one may want to decompose a multi-class problem to a series of binary problems to solve.

*Rescaling* can be implemented in different ways, e.g., re-weighting or re-sampling the training examples of different classes, or even moving the decision threshold directly towards the cheaper class. It can be easily incorporated into existing supervised learning algorithms. For example, for support vector machines, the corresponding optimization problem can be written as

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|_{\mathcal{H}}^2 + C\sum_{i=1}^{m} cost(x_i)\xi_i \\ \text{s.t.} \quad & y_i(w^T\phi(x_i)+b) \geq 1-\xi_i \\ & \xi_i \geq 0 \quad i = 1,\ldots,m\,, \end{aligned} \tag{29.1}$$

where $\phi$ is the feature induced from a kernel function and $cost(x_i)$ is the cost for misclassifying $x_i$. It can be found that the only difference with the classical support vector machine is the insertion of $cost(x_i)$.

It is often difficult to know precise costs in real practice, and some recent studies have tried to address this issue [29.56]. Notice that a learning process may involve various costs such as the *test cost*, *teacher cost*, *intervention cost*, etc. [29.57], and these costs can also be considered in cost-sensitive learning. Last but not least, it should be noted that the variants introduced in this section already go beyond the classic paradigm of supervised learning. Many of them are integrated with unsupervised learning. Some further issues will be also addressed in the following sections.

## 29.3 Unsupervised Learning

Given a set $X_N = \{x_i\}_{i=1}^N$ of unlabeled data samples, unsupervised learning aims at finding a certain dependence structure underlying data $X_N$ with help of a learning principle. The simplest one is the structure of merely a point $\mu$ in a vector space as illustrated in Fig. 29.2a(A(2)). It represents each sample $x_i$ with an error measure $\varepsilon_i = \|x_i - \mu\|^2$. The best $\mu$ may be obtained under a learning principle, e.g., minimizing the

lowing error

$$E_2 = \sum_{i=1}^{N} \varepsilon_{\ell\,i} \tag{29.2}$$

which results in that $\mu$ is simply the mean of the samples.

Efforts made have been far from a simple point structure. As illustrated in Fig. 29.2, these efforts are roughly grouped into several closely related streams. One consists of those listed in Fig. 29.2a(A), featured by increasing the dimensionality of the modeling structure from a single point to a line, plane, and subspace. The second stream consists of those listed in Fig. 29.2a(B), with multiple structures replacing its counterparts listed in Fig. 29.2a(A). The third stream consists of those listed in Fig. 29.2b(C), based on matrix/graph representation of underlying dependence structures. Moreover, another stream is featured with underlying dependencies in tree structures, such as temporal modeling, hierarchical learning, and causal tree structuring, as illustrated in Fig. 29.2b(D). This section will provide a tutorial on the basic structures listed in Fig. 29.2. Also, an overview will be made of a number of emerging topics, mainly coming from a recent systematic overview [29.6].

Additionally, there is also a stream of studies that not only consist of unsupervised learning as a major ingredient but also include features of supervised learning and reinforcement learning, some of which are referred to under the term semi-supervised learning, while others are referred to under the names of semi-unsupervised learning, hybrid learning, mixture of experts, etc. Among them, semi-supervised learning has become a well-adopted name in the literature of machine learning and will be further introduced in Sect. 29.5. Moreover, readers are further referred to Sect. 4.3 of [29.58] and [29.59] for a general formulation called semi-blind learning.

### 29.3.1 Principal Subspaces and Independent Factor Analysis

When a point $\mu$ is replaced with a line structure as illustrated in Fig. 29.2a(A(3)), e.g., represented by a vector $w_\mu = w - \mu$ of a unit length, we consider the error $\varepsilon_i$ as the shortest distance from $x_i$ to the line. Then, minimizing $E$ by (29.2) results in that $w_\mu$ is the principal component direction of the sample set $X_N$, that is, we

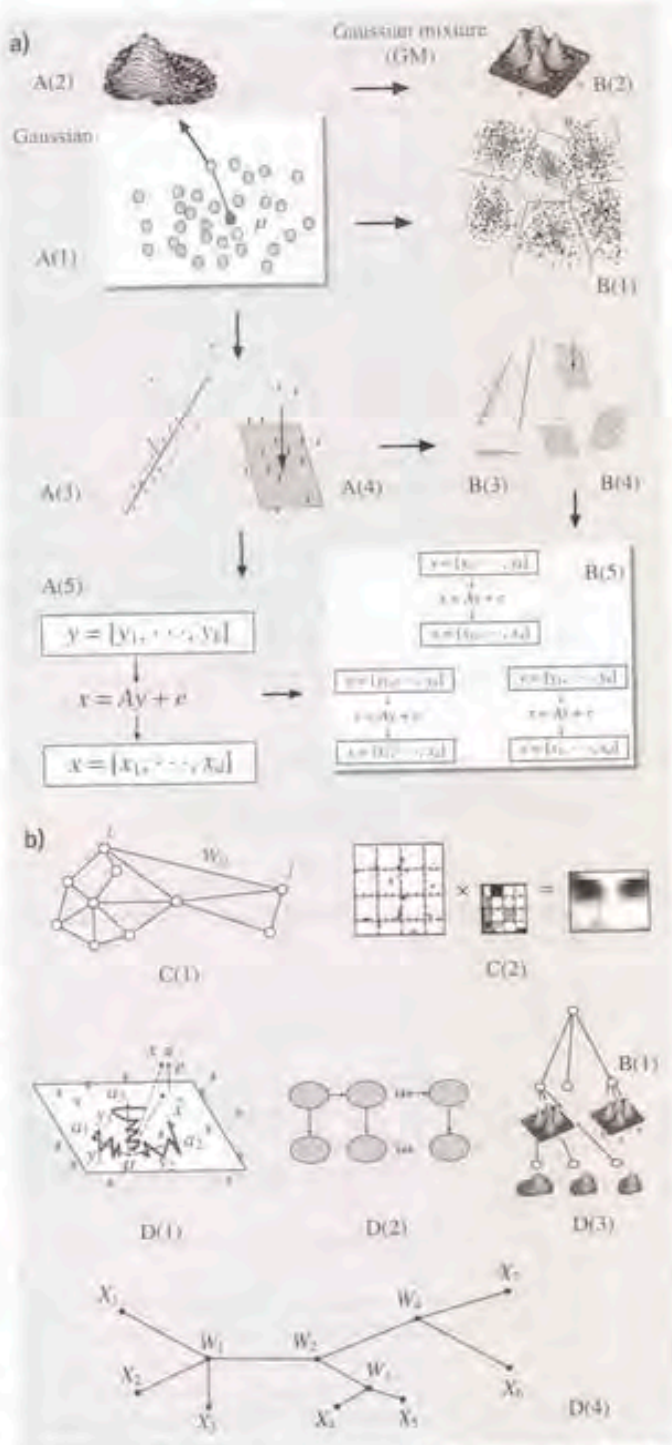


**Fig. 29.2a,b** Four streams of unsupervised learning studies featured by types of underlying dependence structures

have

$$Sw_\mu = \lambda w_\mu,\ S = \frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)(x_i-\mu)^T\ . \quad (29.3)$$

where $\lambda$ is the largest eigenvalue of the sample covariance matrix $S$, and $w_\mu$ is the corresponding eigenvector.

Moreover, we consider a plane or subspace illustrated in Fig. 29.2a(A(4)), resulting in a principal plane or subspace, i.e., a subspace spanned by $m$ eigenvectors that correspond to the first $m$ largest eigenvalue of $S$. Usually, the tasks in Fig. 29.2a(A(3)) and Fig. 29.2a(A(4)) are called *principal component analysis (PCA)* and *principal subspace analysis (PSA)*.

Considering a subspace spanned by the column vectors of $A$, each sample $x_t$ is represented by $\hat{x}_t = Ay_t$ from a vector $y_t = [y_t^{(1)}, \ldots, y_t^{(m)}]^T$ in the subspace with the mutually independent elements of $y_t$ being the coordinates along these column vectors, subject to an error $e_t = x_t - \hat{x}_t$ that is uncorrelated to or independent of $y_t$. Thus, as illustrated in Fig. 29.2a(A(5)), $x_t$ comes from an underlying subspace as follows

$$\begin{aligned} &x_t = \hat{x}_t + e_t = Ay_t + e_t\ , \\ &Ee_t y_t^T = 0 \text{ or } p(e_t|y_t) = p(e_t)\ . \end{aligned} \quad (29.4)$$

featured with the following independence

$$p(y_t) = p(y_t^{(1)}) \cdots p(y_t^{(m)})\ . \quad (29.5)$$

Particularly, it is called *factor analysis (FA)* if we consider

$$\begin{aligned} &p(y_t) = G(y_t|0, I)\ , \\ &p(e_t) = G(e_t|0, D) \quad \text{for a diagonal } D\ . \end{aligned} \quad (29.6)$$

where $G(x|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ denotes a Gaussian density with the mean vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$.

In general, the matrix $\mathbf{A}$ and other unknowns in (29.4) are estimated by the maximum likelihood learning on

$$q(x|\theta) = G(x|0, \mathbf{A}\mathbf{A}^T + D)\ . \quad (29.7)$$

with help of the expectation maximization (EM) algorithm. With the special case $D = \sigma^2 I$, the space spanned by the column vectors of $A$ is the same subspace spanned by $m$ eigenvectors that correspond to the first $m$ largest eigenvalue of $S$ in (29.3), which has been a well-known fact since Anderson's work in 1959. In the last two decades, there has been a renewed interest in the machine learning literature under the new name of probabilistic PCA.

Classically, the principal subspace is obtained via computing the eigenvectors of the sample covariance $\boldsymbol{\Sigma}$. However, $\boldsymbol{\Sigma}$ is usually poor in accuracy when the sample size $N$ is small while the dimensionality of $x_t$ is high. Alternatively, Oja's rule and variants thereof have been proposed to learn the eigenvectors adaptively per sample without directly computing $\boldsymbol{\Sigma}$. Also, extensions have been made on adaptive robust PCA learning on data with outliers and on the adaptive principle curve as the line in Fig. 29.2a(A(3)) extended to a curve.

A hyperplane has two dual representations. One is spanned by several one-dimensional unit vectors, while the other is represented by a unit length normal vector $w$ that is orthogonal to this subspace. In the latter case, minimizing $E$ results in that $w_\mu$ is still a solution of (29.3) but with $\lambda$ becoming the smallest eigenvalue instead of the largest one. Accordingly, the problem is called *minor component analysis* (MCA). In general, an $m$-dimensional subspace in $R^d$ may also be represented by the spanning vectors of a $d-m$ complementary subspace, for which minimizing $E$ results in $d-m$ eigenvectors that correspond to the first $m$ smallest eigenvalues of $S$. The problem is called *minor subspace analysis* (MSA). When $m > d/2$, the minor subspace needs fewer free parameters than the principal subspace does. In a dual subspace pattern classification, each class is represented by either a principal subspace or a minor subspace. Because they are different from PCA and PSA, MCA and MSA are more prone to noises. For further details about these topics the interested reader is referred to Sect. 3.2.1 of [29.58] and a recent overview [29.60]. In the following, we add brief summaries on three typical methods.

### PCA versus ICA

Independent component analysis (ICA) has been widely studied in the past two decades. The key point is to seek a linear mapping $y_t = Wx_t$ such that the components $y_t^{(1)}, \ldots, y_t^{(m)}$ of $y_t$ become mutually independent, as shown in (29.5), as an extension of PCA by which the components $[y_t^{(1)}, \ldots, y_t^{(m)}]$ of $y_t = Wx_t$ become mutually de-correlated when the rows of $W$ consist of the eigenvectors of the first $m$ largest eigenvalue of $S$ in (29.3). Strictly speaking, this is inexact since the counterpart of ICA should be called de-correlated component analysis (DCA), with independence among $y_t^{(1)}, \ldots, y_t^{(m)}$ in the second order of statistics. PCA is one extreme case of DCA that

...ooses those de-correlated components with the first ... largest variances/eigenvalues, while MCA is an...ther extreme case that chooses those with the first ... smallest variances/eigenvalues. Correspondingly, the ...xtended counterpart of PCA/MCA should be the prin...cipal/minor ICA that chooses the independent compo...nents with the first $m$ largest/smallest variances. Read...ers are referred to Sect. 2.4 of [29.61] for further details. Several adaptive learning algorithms have been devel...oped for implementing ICA, but their implementation cannot be guaranteed (29.5). Theoretically, theorems have been proved that such a guarantee can be reached as long as one bit of information is provided to each component $y_t^{(j)}$ [29.62].

#### FA-a, FA-b, and Model Selection

In the literature of statistics and machine learning, the model (29.4) with (29.6) is conventionally referred to as FA. Actually, we have $Ay = \bar{A}\bar{y}$ with $\bar{A} = A\phi^{-1}$, $\bar{y} = \phi y$ for any unknown nonsingular matrix. Among the different choices to handle this indeterminacy, the standard one, shortly denoted by FA-a, imposes (29.6) on $y$, which reduces an indeterminacy of a general nonsingular $\phi$ to an orthonormal matrix. One other choice is given by (29.4) with

$$A^T A = I,\ p(y_t) = G(y_t|0, \Lambda) \text{ for a diagonal } \Lambda\ . \tag{29.8}$$

shortly denoted by FA-b. We have $\hat{x}_t = Ay = A\Lambda\Lambda^{-1}y = A\Lambda\phi^T\phi\Lambda^{-1}y = \bar{A}\bar{y}$, with $\bar{A} = A\Lambda\phi^T$, $\bar{y} = \phi\Lambda^{-1}y$, and $\phi^T\phi = I$, i.e., its $\hat{x}_t$ is equivalent to the one by FA-a for a given $m$ with an invertible $\Lambda^{-1}$. In other words, FA-a and FA-b are equivalent for a learning principle based on $e_t = x_t - \hat{x}_t$, e.g., minimizing $E$ by (29.2) or maximizing the likelihood on $x_t$. Moreover, FA-a and FA-b are still equivalent when model selection is used for determining an appropriate value $m$ by one of the classic model selection criteria to be introduced later in (29.14). However, FA-a and FA-b become considerably different to the Bayesian Yin-Yang (BYY) harmony learning in Sect. 3.2.1 of [29.58] and also to automatic model selection in general. Empirically, experiments show that not only BYY harmony learning but also the variational Bayes method perform considerably better on FA-b than on FA-a [29.63].

#### Non-Gaussian FA

Both FA-a and FA-b still suffer an indeterminacy of an $m \times m$ orthonormal matrix $\phi$, which can be further removed when at most one of the components $y_t^{(1)}, \ldots, y_t^{(m)}$ is Gaussian. Accordingly, (29.4) with non-Gaussian components $p(y_t^{(j)})$ in (29.5) is called *non-Gaussian FA* (NFA). It is also referred to as *independent FA* (IFA), although NFA sounds better, since the concept of IFA covers not only NFA but also FA-a and FA-b. One useful special case of NFA is called binary FA when $y_t$ is a binary vector. Moreover, in the degenerated case $e_t = 0$, obtaining $A$ of $x_t = Ay_t$ subject to (29.5) is equivalent to getting $W = A^{-1}$ such that $Wx_t = y_t$ to satisfy (29.5). For this reason, NFA with $e_t \neq 0$ is also sometimes referred to as noisy ICA. Strictly speaking, the map $x_t \to y_t$ towards (29.5), being an inverse of NFA, should be nonlinear instead of a linear $y_t = Wx_t$. Maximum likelihood learning is implemented with help of the EM algorithm, which was developed in the middle to end of the 1990s for BFA/NFA, respectively. Also, learning algorithms have been proposed for implementing BYY harmony learning with automatic model selection on $m$. Recently, both BFA and NFA were used for transcription regulatory networks in gene analysis; for further details the reader is referred to the overview [29.60] and especially its *Roadmap*.

### 29.3.2 Multi-Model-Based Learning: Data Clustering, Object Detection, and Local Regression

The task of data clustering is partitioning a set of samples into several clusters such that samples within a sample cluster are similar while samples from different clusters should be as different as possible. An indicator matrix $\mathbf{P} = [p_{\ell,t}]$ with $PP^T = I$ is used to represent one possible partition of a sample set $X_N = \{x_t\}_{t=1}^N$ into one of $\ell = 1, \ldots, k$ clusters, i.e., $p_{\ell,t} = 1$ if $x_t$ belongs to the $\ell$-th cluster, otherwise $p_{\ell,t} = 1$. For multi-model-based clustering, each cluster is modeled by one structure, with $p_{\ell,t}$ obtained by a competition of using the structure of each cluster to represent a sample $x_t$. The structure of each cluster could be one of the ones listed on the left-hand side of Fig. 29.2; multiple clusters are thus represented by multiple structures listed on the right-hand side of Fig. 29.2, which feature the basic topics of the second stream of studies.

We still start from the simplest point structure illustrated in Fig. 29.2a(A(1)), extending to the structure of multi-points illustrated in Fig. 29.2a(B(1)). With data already divided into $k$ clusters, it is easy to obtain the mean $\mu_j$ of each cluster. Given $\{\mu_j\}_{j=1}^k$ fixed, it is also

easy to divide $X_N$ into $k$ clusters by

$$p_{\ell,t} = \begin{cases} 1, & \ell = \arg\min_j \varepsilon_{j,t} \\ 0, & \text{otherwise} \end{cases}, \tag{29.9}$$

where $\varepsilon_{j,t} = \|x_t - \mu_j\|^2$, i.e., $x_t$ is assigned to the $\ell$-th cluster if $p_{\ell,t} = 1$. The key idea of the k-means algorithm is alternatively getting $p_{\ell,t}$ and computing $\mu_j$ from an initialization. Although it aims at minimizing $E_2$ by 29.2 with $\varepsilon_t = \sum_{\ell=1}^{k} p_{\ell,t}\varepsilon_{j,t}$, k-means typically results in a local minimum of $E_2$, depending on the initialization.

Merely using the mean $\mu_j$ is not good for describing a cluster beyond a ball shape. Instead, it is extended to considering the Gaussian illustrated in Fig. 29.2a(A(2)) and thus its counterpart in Fig. 29.2a(B(2)), i.e., the following Gaussian mixture

$$q(x|\theta) = \sum_{j=1}^{k} \alpha_j G(x|\mu_j, \Sigma_j), \tag{29.10}$$

K-means can be extended to getting $p_{\ell,t}$ by (29.11) with

$$\varepsilon_{j,t} = -\ln[\alpha_j G(x|\mu_j, \Sigma_j)]. \tag{29.11}$$

and computing each Gaussian by

$$\begin{aligned} \alpha_\ell^* &= \frac{\sum_t p_{\ell,t}}{N}, \\ \mu_\ell^* &= \frac{1}{N\alpha_\ell^*} \sum_t p_{\ell,t} x_t, \\ \Sigma_\ell^* &= \frac{1}{N\alpha_\ell^*} \sum_t p_{\ell,t}(x_t - \mu_\ell^*)(x_t - \mu_\ell^*)^T, \end{aligned} \tag{29.12}$$

which actually performs a type of elliptic clustering.

Instead of getting $p_{\ell,t}$ by (29.9), we compute

$$p_{\ell,t} = q(\ell|x_t, \theta^*), \; q(\ell|x_t, \theta) = e^{-\varepsilon_{\ell,t}} / \sum_{j=1}^{k} e^{-\varepsilon_{j,t}}, \tag{29.13}$$

Actually, alternatively iterating (29.13) and (29.12) is the well-known EM algorithm for carrying out maximum likelihood learning on the Gaussian mixture.

Another important topic is to determine an appropriate $k$ (model selection), i.e. how many clusters are needed. Classic model selection seeks a best $k^* = \arg\min_k J(k)$ with a criterion $J(k)$ in a format as follows

$$J(k) = -L(k, \theta^*) + \omega(k, N), \; \theta^* = \arg\max_{\theta} L(k, \theta), \tag{29.14}$$

where $L(k, \theta)$ is the likelihood function of $q(x|\theta)$, and $\omega(k, N) > 0$ increases with $k$ and decreases with $N$. One typical example is called the Bayesian information criterion (BIC) or minimum description length (MDL). To obtain $k^*$ one needs to enumerate a set of $k$ values and estimate $\theta^*$ for each $k$ value, which incurs an extensive computation and is thus difficult to scale up to a large number of clusters.

Alternatively, *automatic model selection* aims at obtaining $k^*$ during learning $\theta^*$ by a mechanism of principle that is different from the maximum likelihood. This learning drives away an extra cluster via a certain indicator $\rho_j \to 0$, e.g., $\rho_j = \alpha_j$ or $\rho_j = \alpha_j Tr[\Sigma_j]$. One early effort is rival penalized competitive learning (RPCL). RPCL learning does not implement (29.12) by either (29.9) or (29.13), with $p_{\ell,t}$ given as follows

$$p_{\ell,t} = \begin{cases} 1, & \ell^* = \arg\min_j \varepsilon_{j,t}, \\ -\gamma, & \ell = \arg\min_{\ell \neq \ell^*} \varepsilon_{j,t}, \\ 0, & \text{otherwise}, \end{cases} \tag{29.15}$$

by which learning is made on a cluster when $p_{\ell,t} = 1$, and penalizing or de-learning is made on a cluster when $p_{\ell,t} = -\gamma$, with a heuristic penalizing strength of roughly $\gamma \approx 0.005 \approx 0.05$.

The BYY harmony learning gets rid of the difficulty of finding an appropriate penalizing strength, with both parameter learning and model selection made under the *Ying Yang* best harmony principle. The algorithm obtained still implements (29.12) and replaces $p_{\ell,t}$ by (29.12) with

$$\begin{aligned} p_{\ell,t} &= q(\ell|x_t, \theta^*)(1 + \Delta\pi_{\ell,t}), \\ \Delta\pi_{\ell,t} &= \sum_j q(j|x_t, \theta^*)\varepsilon_{j,t} - \varepsilon_{\ell,t}, \end{aligned} \tag{29.16}$$

where $\Delta\pi_{\ell,t} > 0$ means that the $j$-th component is better than the average of all the components for describing the sample $x_t$. We further update the $j$-th component in (29.12) to enhance the description. If $0 > \Delta\pi_{\ell,t} > -1$, i.e., the fitness by the $j$-th component to $x_t$ is below the average but still not too far away, updating of the $j$-th component remains the same trend as in (29.12) but with reduced strength. Moreover, when $-1 > \Delta\pi_{\ell,t}$, the updating on the $j$-th component

reverses direction to become de-learning, similar to updating the rival in RPCL learning.

RPCL learning, which was proposed in 1992, and BYY harmony learning, which was developed in 1995, are similar in nature to the popular sparse learning method, which was developed in 1995 [29.64, 65], and prior-based automatic model selection approaches [29.66–68], that is, extra parts in a model are removed as some parameters are pushed towards zero. Without any priors on the parameters, these prior-based approaches degenerate to maximum likelihood learning, while RPCL learning and further improved by incorporating appropriate priors.

For further details about automatic model selection, prior-aided learning, and model selection criteria the reader is referred to Sect. 2.2 of [29.58, 69] and [29.70] for recent overviews. Also, readers are referred to Sect. 2.1 and Table 1 of [29.58] for a tutorial on several algorithms for learning Gaussian mixture, including the ones introduced above. In the following, only three typical ones are briefly summarized.

#### Local Subspaces and Local Factor Analysis

As illustrated in Fig. 29.2a(B(3)–(5)), the structure of multi-points illustrated in Fig. 29.2a(B(1)) can be extended into multiple subspaces and FA models. Still, we can obtain $p_{\ell,t}$ by (29.9), (29.15), and (29.13) with $\varepsilon_{j,t}$ given by either (29.11) with $\Sigma_j = A_j A_j^T + D_j$ or simply the shortest square distance from $x_t$ to the $j$-th subspace. Given data divided into $k$ clusters, we may estimate the subspace or FA of each cluster as introduced in Sect. 29.3.1, which leads to extensions of the k-means algorithm, the EM algorithm, and the BYY harmony learning algorithm for learning FAs or subspaces that locate at different $\{\mu_j\}_{j=1}^k$. Moreover, readers are referred to [29.71] and [29.2] for learning local FAs with both the number $k$ and the dimensions $\{m_j\}$ determined automatically during BYY harmony learning.

#### Object Detection and Pattern-Based Clustering

The structures in Fig. 29.3a(B(3),(4)) are applicable to the tasks of detecting lines and subspaces among image data, which are topics that are widely studied in the literature of pattern recognition and handled by the well-known Hough transform (HT) and randomized HT (RHT) [29.70]. Extensions can be made to detect multiple objects such as circles, ellipses, lines and other shapes, as well as so-called pattern based clustering, still obtaining $p_{\ell,t}$ by (29.9), (29.15), and (29.13) but with $\varepsilon_{j,t}$ being the shortest square distance from $x_t$ to each shape. However, it is no longer possible to use (29.12) for updating the parameters $\theta_j$ of each shape. Instead, learning is done by

$$\theta_\ell^{\text{new}} = \theta_\ell^{\text{old}} + \eta p_{\ell,t} \nabla_{\theta_\ell} \varepsilon_{\ell,t} , \tag{29.17}$$

where $\eta > 0$ is a learning step size; for further details the reader is referred to [29.69, 70] and [29.8].

#### Mixture of Experts, RBF Networks, SBF Functions

Let each Gaussian to be associated with a function $f(x|\phi_j)$ for a mapping $x \to z$, we consider the task of learning

$$q(z_t|x_t, \psi) = \sum_{j=1}^{k} q(\ell|x_t, \theta) G[z_t|f_j(x_t, \phi_j), \Gamma_j] , \tag{29.18}$$

from a set $D_N = \{x_t, z_t\}_{t=1}^N$ of labeled data. The above $q(z_t|x_t, \psi)$ is actually the alternative mixture of experts [29.72], featured by a combination of unsupervised learning for the Gaussian mixture by (29.10) and supervised learning for every $f(x|\phi_j)$. For a regression task, typically we consider $f(x|\phi_j) = w_j^T x + c_j$ with $E[z|x] = \sum_{j=1}^k q(j|x, \theta) f_j(x, \phi_j)$ implementing a type of piecewise linear regression. In implementation, we still obtain $p_{\ell,t}$ by (29.9), (29.15), and (29.13) but with the following $\varepsilon_{j,t}$

$$\varepsilon_{j,t} = -\ln[\alpha_j G(x|\mu_j, \Sigma_j) G(z_t|f_j(x_t, \phi_j))] , \tag{29.19}$$

and then compute each Gaussian by (29.12), as well as update $G(z_t|f_j(x_t, \phi_j), \Gamma_j)$. When $\alpha_j = |\Sigma_j| / \sum_{i=1}^k |\Sigma_i|$, it becomes equivalent to an extended normalized radial basis function (RBF) network and a normalized RBF network simply with $w_j = 0$. Moreover, letting each subspace be associated with $f(x|\phi_j)$ will lead to subspace-based functions (SBFs). For further details readers are referred to [29.5] and Sect. 7 of [29.69].

### 29.3.3 Matrix-Based Learning: Similarity Graph, Nonnegative Matrix Factorization, and Manifold Learning

We proceed to the third stream, featured with graph/matrix structures. We start with the sample similarity graph, with each node for a sample and each edge

attached with a similarity measure between two samples, as illustrated in Fig. 29.2b(C(1)). Such a graph is also equivalently represented by a symmetric matrix $\mathbf{W} = [w_{ij}]$.

One similarity measure is simply the inner product $w_{ij} = x_i^T x_j$ of two samples. Given a data matrix $\mathbf{X} = [x_1, \ldots, x_N]$, we simply have $\mathbf{W} = \mathbf{X}^T\mathbf{X}$. We seek an indicator matrix $\mathbf{P} = [p_{\ell,i}]$ that divides $X_N = \{x_t\}_{t=1}^N$ into $k$ clusters, with help of the following maximization

$$\max_{HH^T=I, H\geq 0} \mathrm{Tr}[\mathbf{H}^T\mathbf{W}\mathbf{H}] .$$
$$\mathbf{H} = \mathrm{diag}[n_1^{-0.5}, \ldots, n_k^{-0.5}]P , \qquad (29.20)$$

where $\mathbf{H} \geq 0$ is a nonnegative matrix with each element $h_{ij} \geq 0$, and $n_\ell$ is the number of samples in the $\ell$-th cluster. It can be shown that this problem is equivalent to minimizing $E_2$ by (29.2) with $\varepsilon_t = \sum_{\ell=1}^k p_{\ell,t}\|x_t - \mu_\ell\|^2$, i. e., the same target that k-means aims at.

Computationally, (29.20) is a typical intractable binary quadratic programming problem, for which various approximate methods are proposed. The most simple one is dropping the constraint $\mathbf{H} \geq 0$ to do a PCA analysis about the matrix $\mathbf{W}$. That is, the columns of $\mathbf{H}$ consist of the $k$ eigenvectors of $\mathbf{W}$ that correspond to the first $k$ largest eigenvalues. Then, each element of the matrix $\mathrm{diag}[n_1^{0.5}, \ldots, n_k^{0.5}]\mathbf{H}$ is chopped into 1 or 0 by a rule of thumb.

Another similarity measure is $w_{ij} = \exp(-\|x_i - x_j\|^2)$, based on which we consider dividing the nodes of a graph into balanced two sets $A, B$ such the total sum of $w_{ij}$ associated with edges connecting the two sets becomes as small as possible. Using a vector $f = [f_1, \ldots, f_N]^T$ with $f_t = 1$ if $x_t \in A$ and $f_t = -1$ if $x_t \in B$, the problem is formulated as follows

$$\min_f f^T L f , \ s.t. \ [1, \ldots, 1]f = 0 ,$$
$$L = D - W, \ D = \mathrm{diag}[w_{11}, \ldots, w_{NN}] , \qquad (29.21)$$

where $L$ is the graph Laplacian. Again, it is an intractable combinatorial problem and needs to consider some approximation. A typical one is given as follows

$$\min_f \frac{f^T L f}{f^T f} , s.t. \ [1, \ldots, 1]f = 0 . \qquad (29.22)$$

Its solution $f$ is the eigenvector of $L$ that corresponds the second smallest eigenvalue. Moreover, this idea has been extended to cutting a graph into multiple clusters, which leads to approximately finding $\mathbf{H}$ with its columns being the eigenvectors of $\tilde{\mathbf{W}} = D^{-0.5}WD^{-0.5}$, corresponding to the first $k$ largest eigenvalues.

Moreover, the above studies are closely related to nonnegative matrix factorization (NMF) problems [29.73]. For example, the above problem can be equivalently expressed as factorization $\tilde{\mathbf{W}} \approx \mathbf{H}^T\mathbf{H}$ by

$$\min_{HH^T=I, H\geq 0} \|\tilde{\mathbf{W}} - \mathbf{H}^T\mathbf{H}\|^2 . \qquad (29.23)$$

More generally, the NMF problem considers that

$$\mathbf{X} \approx \mathbf{F}\mathbf{H}, \ \mathbf{X} \geq 0, \ \mathbf{F} \geq 0, \mathbf{H} \geq 0 , \qquad (29.24)$$

as illustrated in Fig. 29.2b(C(1)). One typical method is to iterate the following multiplicative update rule

$$\mathbf{H}_{ij}^{\mathrm{new}} = \mathbf{H}_{ij}^{\mathrm{old}} \frac{(\mathbf{F}^T\mathbf{X})_{ij}}{(\mathbf{F}^T F\mathbf{H})_{ij}} , \ \mathbf{F}_{ij}^{\mathrm{new}} = \mathbf{F}_{ij}^{\mathrm{old}} \frac{(\mathbf{X}\mathbf{H}^T)_{ij}}{(\mathbf{F}\mathbf{H}\mathbf{H}^T)_{ij}} , \qquad (29.25)$$

which guarantees nonnegativity and is supposed to converge to a local solution of the following minimization

$$\min_{HH^T=I, H\geq 0, F\geq 0} \|\mathbf{X} - \mathbf{F}\mathbf{H}\|^2 . \qquad (29.26)$$

Particularly, if we also impose the constraint $\mathbf{F}^T\mathbf{F} = \mathbf{I}$, the resulted $\mathbf{H}$ divides the columns of $\mathbf{X}$ into $k$ clusters, while the resulted $\mathbf{F}$ also divides the rows of $\mathbf{X}$ into $k$ clusters, and is thus called *bi-clustering* [29.74].

Several NMF learning algorithms have been developed in the literature. In [29.75], a binary matrix factorization (BMF) algorithm was developed under BYY harmony learning for clustering proteins that share similar interactions, featured with the nature of automatically determining the cluster number, while this number has to be pre-given for most existing BMF algorithms.

In the past decade, the similarity graph and especially the graph Laplacian $L$ have also taken important roles in another popular topic called *manifold learning* [29.76, 77]. Considering a mapping $Y \approx WX$, a locality preserving projection is made to minimize the sum of each distance between two mapped points on the graph, subject to a unity $L_2$ norm of this projection $WX$.

Alternatively, we may also regard that $X$ is generated via $X = AY + E$ such that the topological dependence among $Y$ is preserved by considering

$$q(Y) \propto e^{-\frac{1}{2}\mathrm{Tr}[YLY^T\Lambda^{-1}]} , \qquad (29.27)$$

where $\Lambda$ is a positive diagonal matrix. Learning is implemented by BYY harmony learning, during which automatic model selection is made via updating $q(Y)$ to drive some diagonal elements of $\Lambda$ towards zeros. For further details readers are referred to the end part of Sect. 5 in [29.58].

### 29.3.4 Tree-Based Learning: Temporal Ladder, Hierarchical Mixture, and Causal Tree

Unsupervised learning also includes learning temporal and hierarchical underlying dependence structures, as illustrated in Fig. 29.2b(D). Instead of directly modeling temporal dependence underlying data $X = [x_1, \ldots, x_N]$, its structure is typically represented in a hidden space, while non-temporal or spatial dependence is represented by a relation from the hidden space to the space where $X$ is observed, in the ladder structure illustrated in Fig. 29.2b(D(1)).

One typical example is the classic hidden Markov model (HMM). Its hidden space is featured by a discrete variable that jumps between a set of discrete values or states $\{s_j\}$, with temporal dependence described by the jumping probabilities between the states, typically considering $p(s_j|s_i)$ of jumping from one state $s_i$ to another $s_j$. The relation from the hidden space to the space of $X$ is described by $p(x_t|s_i)$ for the probability that the value of $x_t$ is emitted from the state $s_i$. Classically, the values of $x_t$ are also a set of labels. The task is learning from $X = [x_1, \ldots, x_N]$ two probability matrices $\mathbf{Q} = [p(s_j|s_i)]$ and $\mathbf{E} = [p(x_t|s_i)]$. Given the number of states, learning is typically implemented to maximize the likelihood $p(X|\mathbf{Q}, \mathbf{E})$ by the well-known Baum–Welch algorithm.

Another example is the classic state–space model (SSM), which has been widely studied in the literature of control theory and signal processing since the 1960s; this has also been called a linear dynamical system with considered with renewed interest since the beginning of the 2000s. As illustrated in Fig. 29.2b(D(2)), its hidden space is featured by an $m$-dimensional subspace and temporal dependence is described by one first-order vector autoregressive model as follows

$$y_t = By_{t-1} + \varepsilon_t,\ Ey_{t-1}\varepsilon_t^T = 0\,,$$
$$\varepsilon_t \approx G(\varepsilon_t|0, \Lambda),\ \Lambda \text{ is diagonal}\,, \qquad (29.28)$$

while the spatial dependence is represented by a relation between the coordinates of the state–space and the coordinates of the space of $X$, e.g., typically by (29.4). Though the EM algorithm has also been suggested for learning the SSM parameters, the performance is usually unsatisfactory because an SSM is generally not identifiable due to an indeterminacy of any unknown nonsingular matrix, similar to what was discussed previously with respect to the FA in (29.5). Favorably, it has been shown that the indeterminacy of not only any unknown nonsingular matrix but also an unknown orthonormal matrix is usually removed by additionally requiring a diagonal matrix $B$, which leads to temporal factor analysis (TFA).

TFA is an extension of the FA by (29.4) with (29.6) replaced by (29.28). As introduced in Sect. 29.3.1, the FA is generalized into NFA when (29.6) becomes (29.5) with each $p(y_t^{(j)})$ being non-Gaussian. The NFA with a real vector $y_t$ can be further extended into a temporal NFA when (29.5) is also extended by (29.28) with

$$p(\varepsilon_t) = p(\varepsilon_t^{(1)}) \cdots p(\varepsilon_t^{(m)})\,.$$

Moreover, the BFA (i.e., NFA with a binary vector $y_t$) can be extended into a temporal BFA. Also, TFA has been extended into an integration of several TFA models coordinated by an HMM. For further details readers are referred to Sect. 5.2 of [29.58] for a recent overview on TFA and its extensions.

A ladder is merely a special type of tree structure. Hierarchical modeling is one other type of tree structure, as illustrated in Fig. 29.2b(D(3)). Again, the EM algorithm has been extended to implement learning on a hierarchical or tree mixture of Gaussians [29.78]. Also, a learning algorithm is available for implementing BYY harmony learning with tree configuration determined during learning. A learning algorithm for a three-level hierarchical mixture of Gaussians is shown in Fig. 12 of [29.3], featured by a hierarchical learning flow circling from bottom up as one step and then top down as the other step. Similar to (29.16), where there is a term of $\Delta$ featuring the difference of BYY harmony learning from EM learning, there is also such a $\Delta$ term on each level of hierarchy. If these $\Delta$ terms are set to be zero, the algorithm degenerates back to the EM algorithm. For further details readers are referred to Sect. 5.1 in [29.3] and especially equation (55) therein.

Many applications consider several sets of samples. Each set is known to come from one model or pattern class. Typically, one does unsupervised learning on each set of samples by a hierarchical mixture of Gaussians, and then integrates individual hierarchical models in a supervised way to form a classifier.

Alternatively, we may put together all the individual hierarchical mixtures with each as a branch of one higher level root of a tree, and then do learning as shown in Fig. 12 of [29.3]. The BYY harmony learning algorithm (including the EM algorithm as its degenerated case) for learning a two-level hierarchical mixture of Gaussians is shown in Sect. 5.3 of [29.58], and especially Fig. 11 therein. This type of learning can be regarded as semi-supervised learning in the sense that each sample has two teaching labels. One is known, indicating which individual hierarchy $x_l$ comes from, while the other is unknown to be determined, indicating which Gaussian component $x_l$ comes from. Even generally, this type of learning provides a general formulation that involves the multi-label classification of Sect. 29.2.2 (especially labels with a hierarchy).

There are also real applications that consider a combination of ladder structures and hierarchical structures. For example, what is widely used in speech processing is an HMM model with each hidden state associated with a two-level hierarchical Gaussian mixture as illustrated in Fig. 11 of [29.58]. Also, extensions are made with each Gaussian mixture replaced by a mixture of local subspaces or FA or NFA models. For further details readers are referred to Sect. 5.3 and Fig. 14 of [29.3]. Another example is considering a two-level hierarchical model with both HMM for modeling nonstationary temporal dependence and TFA for modeling stationary temporal dependence. For further details readers are referred to Sect. 5.2.2 of [29.58].

Another typical tree structure, as illustrated in Fig.29.2b(D(4)), is a learning probabilistic tree, i.e., a joint distribution of a set of variables on a tree with one node per variable. The most well-known study is structuring such tree models for a given set of bi-valued variables, as done by *Pearl* in 1986 [29.79]. Following this, one study in 1987 [29.80] extends this to construct tree representations of continuous variables. It has been proved that the tree can be structured from the correlations observed between pairs of variables if the visible variables are governed by a tree decomposable joint normal distribution. Moreover, the conditions for tree decomposable normal distribution are less restrictive than those of bi-valued variables.

Nowadays, many advances have been made along this line. Some of the basic results, e.g., (29.15) and (29.17) in [29.80], has become a widely used technique in network construction for detecting whether an edge describes a direct link or a duplicated indirect link. For example, considering the association between two nodes $i, j$ linked to a third node $w$ with the correlation coefficients $\rho_{iw}$ and $\rho_{jw}$, we can remove the link $i, j$ if its correlation coefficient $\rho_{ij}$ fails to satisfy, i.e.,

$$\rho_{ij} > \rho_{iw}\rho_{wj} \,. \tag{29.29}$$

Otherwise we may either choose to keep the link $i, j$, or let three nodes to be linked to a newly added node and then remove all the original links among the three nodes.

## 29.4 Reinforcement Learning

Differently to unsupervised learning, reinforcement learning gets guidance from external evaluation. Also, unlike supervised learning in which the teacher clearly specifies the output that corresponds to an input, reinforcement learning is only provided with an evaluative value about the action made. Furthermore, reinforcement learning is featured by a dynamic process in discrete time steps. At each step, upon observing the current environment and getting some input (if any), the learner makes an action and moves to a new state, receiving an award or punish value about the action. The aim is to maximize the total award received.

This section provides a brief tutorial on the basic issues of reinforcement learning, especially TD learning and Q-learning. Then, improvements on Q-learning are proposed by replacing its built-in winner-take-all competition mechanism with some unsupervised learning methods. For further reading readers are referred to tutorials and reviews in [29.6, 81, 82].

### 29.4.1 Markov Decision Processes

Reinforcement learning is closely related to Markov decision processes (MDP), which consist of a series of states $s_0, s_1, \ldots, s_t, s_{t+1}, \ldots$. At a state $s_t$, an action $a_t = \pi(s_t)$ is selected from the set $A$ of actions according to a policy $\pi$, which makes the environment move to a new state $s_{t+1}$, and the reward $r_{t+1}$ associated with the transition $(s_t, a_t, s_{t+1})$ is received. The goal is to collect as much reward as possible, that is, to maximize the total reward or return

$$R = \sum_{t=0}^{N-1} r_{t+1} \,,$$

re $N$ denotes the random time when a terminal state eached. In the case of nonepisodic problems the rn $R = \sum_{t=0}^{\infty} \gamma^t r_{t+1}$ is considered by a discount- tor $0 \leq \gamma \leq 1$.

Given an initial distribution based on which the itial state is sampled at random, we can assign the pected return $E[R|\pi]$ to policy $\pi$. Since the actions e selected according to $\pi$, the task is to specify algorithm that can be used to find a policy $\pi$ to aximize $E[R]$. Suppose we know the state transition obability $p_a(s'|s) = P(s_{t+1} = s'|s_t = s)$ and the cor- sponding reward $r_{t+1} = R_a(s'|s)$, the standard family algorithms to calculate this optimal policy is featured by iterating the following two steps

$$\begin{aligned} &(1) \quad \pi(s) = \arg\max_a \; V^a(s) \,, \\ &(2) \quad V^{\pi}(s) = \sum_{s'} p_{\pi(s)}(s'|s)[R_{\pi(s)}(s'|s) + \gamma V(s')] \,. \end{aligned} \tag{29.30}$$

with $V^{\pi}(s)$ estimating $E[R|s, \pi]$. The iteration can be made in one of several variants as follows:

- Doing step (1) once and then repeating step (2) several times or until it converges. Then step (1) is done once again, and so on.
- Doing step (2) by solving a set of linear equations.
- Substituting the calculation of $\pi(s)$ into the calculation of $V^*(s) = \max_{\pi} V^{\pi}(s)$, resulting in a combined step

$$V^*(s) = \max_a \left\{ \sum_{s'} p_a(s'|s)[R_a(s'|s) + \gamma V^*(s')] \right\} \,, \tag{29.31}$$

  which is called backward induction and is iterated for all states until it converges to what is called the Bellman equation.
- Preferentially applying the steps to states that are in some way of importance.

Under some mild regularity conditions, all the implementations will reach a policy that achieves these optimal values of $V^*(s) = \max_{\pi} V^{\pi}(s)$ and thus also maximizes the expected return $E[V^{\pi}(s)]$, where $s$ is a state that is randomly sampled from the underlying distribution.

In the implementation of MDPs we need to know the probability $p_a(s'|s)$ per action $a$. Reinforcement learning avoids obtaining this $p_a(s'|s)$ with the help of stochastic approximation. The two most popular examples are temporal difference (TD) learning and Q-learning, respectively. The name TD derives from its use of differences in predictions over successive time steps to drive learning, while the name Q comes from its use of a function that calculates the quality of a state-action combination.

### 29.4.2 TD Learning and Q-Learning

TD learning aims at predicting a measure of the total amount of reward expected over the future. At time $t$, we seek an estimate $\hat{r}_t$ of $R_t = \sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}$ with $0 \leq \gamma < 1$. Each estimate is a prediction because it involves future values of $r$. We can write $\hat{r}_t = \Pi(s_t)$, where $\Pi$ is a prediction function. The prediction at any given time step is updated to bring it closer to the prediction of the same quantity at the next step, based on the error correction $\delta_{t+1} = R_t - \Pi_t(s_t)$. To obtain $R_t$ exactly requires waiting for the arrival of all the future values of $r$. Instead, we use $R_t = r_{t+1} + \gamma R_{t+1}$ with $\Pi_t(s_{t+1})$ as an estimate of $R_{t+1}$ available at step $t$, that is, we have

$$\delta_{t+1} = r_{t+1} + \gamma \Pi_t(s_{t+1}) - \Pi_t(s_t) \,, \tag{29.32}$$

which is termed the temporal difference error (or TD error).

The simplest TD algorithm updates the prediction function $\Pi_t$ at step $t$ into to a new prediction function $\Pi_{t+1}$ as follows

$$\Pi_{t+1}(x) = \begin{cases} \Pi_t(x) + \eta \delta_{t+1} & \text{if } x = s_t \\ \Pi_t(x) & \text{otherwise} \end{cases} \,, \tag{29.33}$$

where $\eta$ is a learning step size and $x$ denotes any possible input signal. The simplest format is a prediction function implemented as a lookup table. Suppose that $s_t$ takes only a finite number of values and that there is an entry in a lookup table to store a prediction for each of these values. At step $t$, the state $s_t$ moves to the next $s_{t+1}$ based on the current status of the table, e.g., the table entry for $s_{t+1}$ is the largest across the table, or $s_{t+1}$ is selected according to a fixed policy. When $r_{t+1}$ is observed, only the table entry for $s_t$ changes from its current value of $\hat{r}_t = \Pi_t(s_t)$ to $\Pi_t(s_t) + \eta \delta_{t+1}$.

The algorithm uses a prediction of a later quantity $\Pi_t(s_{t+1})$ to update a prediction of an earlier quantity $\Pi_t(s_t)$. As learning proceeds, later predictions tend to become accurate sooner than earlier ones, resulting in an overall error reduction. This depends on whether an

input sequence has sufficient regularity to make predicting possible. When $s_t$ comes from the states of a Markov chain, on which the $r$ values are given by a function of these states, a prediction function may exist that accurately gives the expected value of the quantity $R_t$ for each $t$.

Another view of the TD algorithm is that it operates to maintain the following consistency condition

$$\Pi(s_t) = r_{t+1} + \gamma \Pi(s_{t+1}) \,. \tag{29.34}$$

which must be satisfied by correct predictions. By the theory of *Markov* decision processes, any function that satisfies $R_t = r_{t+1} + \gamma R_{t+1}$ for all $t$ must actually give the correct predictions. The TD error indicates how far the current prediction function deviates from this condition, and the algorithm acts to reduce this error towards this condition. Actually, $\Pi_t(s_t) + \eta\delta_{t+1} = (1 - \eta)\Pi_t(s_t) + \eta[r_{t+1} + \gamma\Pi_t(s_{t+1})]$ is a type of stochastic approximation to the value function in (29.30), without directly requiring to know the probability $p_a(s'|s)$.

Alternatively, Q-learning calculates the quality of a state-action combination, i. e., estimating $Q(s_t, a_t)$ of $R_t$ conditionally on the action $a_t$ at $s_t$. The implementation of Q-learning consists of

$$a_t = \arg\max_{a \in A} Q_t(s_t, a) \,.$$

$$Q_{t+1}(x, a) = \begin{cases} Q_t(x, a) + \eta\delta_{t+1}(a) & \text{if } x = s_t \; a = a_t \\ Q_t(x, a) & \text{otherwise.} \end{cases}$$

$$\delta_{t+1}(a) = r(s_t, a) + \gamma \max_{\ell} Q_t(s_{t+1}, \ell) - Q_t(s_t, a) \,. \tag{29.35}$$

At $s_t$, an action $a_t \in A$ is obtained in an easy computation, and then makes a move to a new state $s_{t+1}$. Receiving the reward $r_{t+1} = r(s_t, a_t)$ associated with the transition $(s_t, a_t, s_{t+1})$, only the table entry for $s_t$ and $a_t$ is updated.

The format of $\delta_{t+1}$ is similar to the one in (29.32) with the prediction $\Pi_t(s_t)$ replaced by $Q(s_t, a_t)$ and $\Pi_t(s_{t+1})$ replaced by $\max_a \; Q_t(s_{t+1}, a)$. Alternatively, we may select $a_{t+1}$ by a fixed policy and then obtain $\delta_{t+1}$ with $\max_a \; Q_t(s_{t+1}, a)$ replaced by $Q_t(s_{t+1}, a_{t+1})$, which leads to a variant of the Q-learning rule called state-action-reward-state-action (SARSA). Under some mild regularity conditions, similarly to TD learning, both Q-learning and SARSA converge to prediction functions that make optimal action choices.

Both TD learning and Q-learning have variants and extensions. In the following, we briefly summarize two typical streams:

- In (29.33) and (29.35), only the table entry for $s_t$ is modified, though $r_{t+1}$ provides useful information for learning earlier predictions as well. Under the name of eligibility traces, an exponentially decaying memory trace is provided on a number of previous input signals so that each new observation can update the parameters related to these signals.
- In addition to a lookup table, the prediction function can be replaced by a more advanced prediction function. It could be a linear or nonlinear regression function $F_t(\omega_\tau, \theta)$ with input signals $\omega_\tau = \{x_\tau^{(1)}, \ldots, x_\tau^{(m)}\}$. Each $x_\tau^{(j)}$ could be either a state or an action or even one additional feature around a state in one eligibility trace, where $\tau$ can be different from $t$. Then, learning adjusts $\theta$ to reduce the error $\delta_{t+1}$ or $\delta_{t+1}(a_t)$.

### 29.4.3 Improving Q-Learning by Unsupervised Methods

Examining the Q-learning by (29.35), we observe that it shares some common features with the multi-model-based learning introduced in Sect. 29.3. For a set $A$ of finite many actions, we use the index $\ell = 1, \ldots, k$ to denote each action. Obtaining $a_t$ in (29.35) is equivalent to obtaining $p_{\ell,t}$ in (29.9) with $\varepsilon_{j,t} = -Q_t(s_t, j)$, that is, a selection is made by winner-take-all (WTA) competition. Then, updating $Q_t(s_t, a)$ in (29.35) can be rewritten as follows

$$\begin{aligned} Q_{t+1}(s_t, \ell) &= Q_t(s_t, \ell) + \eta \, p_{\ell,t}\delta_{t+1}(\ell), \\ Q_{t+1}(s, \ell) &= Q_t(s, \ell), \; \textit{for } s \neq s_t \,. \end{aligned} \tag{29.36}$$

which is similar to the general updating rule by (29.17), with $p_{\ell,t}$ selecting which column of $Q$ table to update. This motivates the following improvements on Q-learning, motivated by the multi-model-based learning methods in Sect. 29.3.2.

First, the WTA selection of the above $p_{\ell,t}$ can be replaced by an estimation of the posteriori probabilities as follows

$$p_{\ell,t} = q(\ell|s_t), \; q(\ell|s) = e^{Q_t(s,\ell)} \Big/ \sum_{j=1}^{k} e^{Q_t(s,j)} \,. \tag{29.37}$$

Putting this into (29.38), we improve the weak points incurred from a WTA competition by updating all the columns of the $Q$ table with the weights by $p_{\ell,t}$, as

unterpart of (29.12) of the well-known EM algo-m for learning a finite mixture.

Second, $\delta_{t+1}(a)$ in (29.35) uses $\max_a Q_t(s_{t+1}, a)$ prediction of the Q-value at $s_{t+1}$, also by a WTA petition that gives an optimistic choice. Alterna-ly, we can use the following more reliable one

$$\begin{aligned} {}_{-t}(s_t, a_t) &= r(s_t, a_t) + \gamma \Delta\pi_{\ell,t}(s_{t+1}), \\ \Delta\pi_{\ell,t}(s) &= \sum_j q(j|s) Q_t(s, j) - Q_t(s_t, a_t) \,. \end{aligned} \quad (29.38)$$

ere $q(j|s_{t+1})$ is given by (29.37) with $s = s_{t+1}$ in-d of $s = s_t$, to obtain $\Delta\pi_{\ell,t}(s_{t+1})$ with $q(\ell|s)$.

Third, instead of $p_{\ell,t}$ given by (29.37), we may also a counterpart of (29.16) to implement Q-learning h help of BYY harmony learning. That is, we con-ider

$$p_{\ell,t} = q(\ell|s_t)[1 + \Delta\pi_{\ell,t}(s_t)] \,. \quad (29.39)$$

which an action is encouraged when its value is gher than the average of all actions, while an action discouraged when its value is below the average but ll not too far away, and then is repelled when its value far below this average.

Moreover, we may simplify the above $p_{\ell,t}$ by focus-ing on a few of major actions, e.g., the winning action $a_t = \arg\max_{a \in \mathcal{A}} Q_t(s_t, a)$ and its rival action similar to rival penalized competitive learning (RPCL) by $p_{\ell,t}$ given as follows

$$p_{\ell,t} = \begin{cases} 1, & \ell^* = \arg\max_j Q_t(s_t, j) \,, \\ -\gamma, & \ell = \arg\max_{\ell \neq \ell^*} Q_t(s_t, j) \,, \\ 0, & \text{otherwise} \,, \end{cases} \quad (29.40)$$

i. e., the winning action is encouraged while its rival is repelled.

BYY harmony learning and RPCL learning lead to discriminative Q-learning by which actions at each state become more discriminative and thus easier to be selected. As a result, confusing branches in a searching tree will be pruned away. Moreover, we may discard one extra action if we observe that its corresponding

$$\alpha_\ell^{\text{new}} = (1 - \eta)\alpha_\ell^{\text{new}} + \eta q(\ell|s_t) \,, \quad (29.41)$$

is pushed to zero. Actually, this is the nature of automatic model selection, which controls the complexity of function $Q(s, j)$.

## 29.5 Semi-Supervised Learning

In many real tasks it is easy to obtain a large amount of unlabeled training data but labeling them is expensive because of the requirement of great human effort and expertise or high execution cost. Semi-supervised learning [29.83–86] attempts to exploit unlabeled data to help improve the learning performance without assuming human intervention. In situations where the unlabeled data are exactly the test data, it is also called *transductive learning* [29.87].

Figure 29.3 illustrates why unlabeled data (*gray points*) can be helpful. It can be seen that although both classification boundaries are consistent with labeled data, the boundary obtained by considering unlabeled data is better in generalization. One reason is that the unlabeled data can disclose some information about data distribution which is helpful for model construction.

There are two popular assumptions connecting the distribution information disclosed by unlabeled data with label information. The *cluster assumption* assumes that data with similar inputs have similar class labels; the *manifold assumption* assumes that data live in a low-dimensional manifold, whereas unlabeled data can help to identify that manifold. The latter can be regarded as a generalization of the former because it is usually assumed that the cluster structure of the data will be more easily found in the lower-dimensional

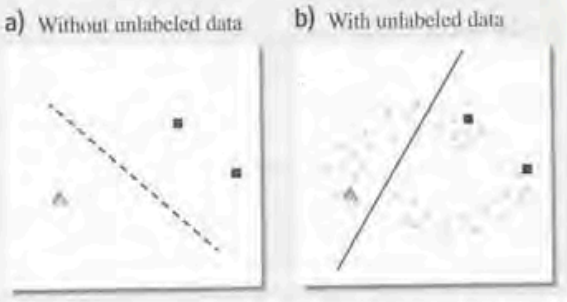


**Fig. 29.3a,b** Illustration of the usefulness of unlabeled data

manifold. These assumptions are closely related to *low-density separation*, which specifies that the boundary should not go across high-density regions in the instance space.

Many semi-supervised learning methods have been developed. Roughly speaking, they can be categorized into four categories. In *generative methods*, both labeled and unlabeled data are assumed to be generated by the same model, and thus, the unlabeled data can be exploited to model the label estimation or parameter estimation process. For example, if we assume the data come from a mixture model with $T$ components, i. e.,

$$f(x|\theta) = \sum_{t=1}^{T} \alpha_t f(x|\theta_t) \,, \tag{29.42}$$

where $\alpha_t$ is mixing coefficient and $\theta = \{\theta_t\}$ are the model parameters, then label $c_i$ can be determined by the mixture component $m_i$ and the instance $x_i$ according to the *maximum* a posteriori criterion

$$\arg\max_k \sum_j P(c_i = k|m_i = j, x_i) P(m_i = j|x_i) \,. \tag{29.43}$$

where estimating $P(c_i = k|m_i = j, x_i)$ requires label information, but unlabeled data can be used to help estimate $P(m_i = j|x_i)$, and hence improve the learning performance. Actually, the posteriori probability is equivalently given by a *mixture of experts* that will be further addressed next in Sect. 29.6.1.

In *semi-supervised support vector machines* (S3VM), unlabeled data are used directly to help adjust the decision boundary, as illustrated in Fig. 29.3. Given $l$ labeled examples and $u$ unlabeled instances, the goal is usually accomplished by minimizing an objective

$$\frac{1}{2}\|w\|_{\mathcal{H}}^2 + C_1 \sum_{i=1}^{l} \ell[y_i, f(x_i)] + C_2 \sum_{j=1}^{u} \ell[\hat{y}_j, f(x_j)] \,, \tag{29.44}$$

where the first term is structural risk, the second term is empirical risk on the labeled data $(x_i, y_i)$, the third term is empirical risk on the unlabeled instances $x_j$ $(j = 1, \ldots, u)$ and the estimated outputs $\hat{y}_j$, whereas $C_1/C_2$ balance the contribution of labeled/unlabeled data.

*Graph-based methods* construct a graph whose nodes are the training instances (both labeled and unlabeled), and the edges between nodes reflect a certain relation, such as similarity, between the corresponding examples. Then, the learning process is accomplished by propagating label information on the graph.

*Disagreement-based methods* generate multiple learners and exploit the disagreements among the learners, where unlabeled data serve as a kind of platform for information exchange; if one learner is much more confident on a disagreed unlabeled instance than other learner(s), then it will *teach* other(s) by assigning a predicted *pseudo-label* to the instance. A representative of this category is *co-training* [29.31], which constructs two learners from two different views, and thus is closely related to multi-view learning.

In addition to classification, semi-supervised regression, dimension reduction, clustering, etc., have also been well studied. It is worth mentioning that exploiting unlabeled data does not always improve the performance, and sometimes the performance may be even worse than using only the labeled data. Some recent studies have tried to address this issue under the name of *safe* semi-supervised learning [29.88].

## 29.6 Ensemble Methods

### 29.6.1 Basic Concepts

Ordinary learning methods try to construct one learner from training data, whereas ensemble methods [29.14] try to construct a set of learners and combine them to solve the problem. Such kinds of learning methods are also called *committee-based learning*, *meta-learning*, or *multiple classifier systems*, although ensemble methods have also been found to be helpful in clustering [29.14, 89, 90] and various tasks other than classification.

Figure 29.4 shows a common ensemble architecture. An ensemble contains a number of *base learners*, or *individual learners*, *component learners*, or *weak learners* because the main purpose of ensemble methods is to generate strong learners by combining learners whose generalization performances are not strong. Base learners can be generated by a *base learning*

*algorithm*, such as a decision tree algorithm, a neural network algorithm, etc., and such ensembles are called *homogeneous* ensembles because they contain homogeneous base learners. An ensemble can also be *heterogeneous* if multiple types of base learners are included.

The generalization ability of an ensemble is often much stronger than that of base learners. Roughly, there are three threads of studies that lead to the state-of-the art of ensemble methods. The *combining classifiers* thread was mostly studied in the pattern recognition community, where researchers usually focused on the design of powerful combining rules to obtain a strong combined classifier [29.28, 29]. The *mixture of experts* thread generally considered a divide-and-conquer strategy, trying to learn a mixture of parametric models jointly [29.91]. Equation (29.43) is actually a mixture of experts for classification, with $P(c_i = k|m_i = j, x_i)$ being the individual expert and $P(m_i = j|x_i)$ the gating net, especially for the one given in [29.72] where the gating net is given by the posteriori of $\alpha_i f(x|\theta_i)$ in (29.42). The *ensembles of weak learners* thread often works on weak learners and tries to design powerful algorithms to boost performance from weak to strong. Readers are referred to [29.30] for a recent survey on *combining classifiers* and *mixture-of-experts* as well as their relations, and to Sect. 5 of [29.86] for a brief overview on all the three threads.

Generally, an ensemble is built in two steps: that is, generating the base learners and then combining them. It is worth noting that the computational cost of constructing an ensemble is often not much larger than creating a single learner. This is because when using a single learner, one usually has to generate multiple versions of the learner for model selection or parameter tuning; this is comparable to generating multiple base learners in ensembles, whereas the computational cost for combining base learners is often small because most combining rules are simple.

The term *boosting* refers to a family of algorithms originated in [29.92], with AdaBoost [29.93] as its representative. This kind of algorithm is usually provably able to convert weak learners that are just slightly better than random guess to strong learners that have nearly perfect performance.

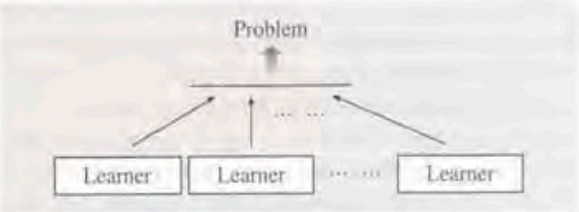


**Fig. 29.4** A common ensemble architecture

Algorithm 29.1 shows the pseudo-code of AdaBoost. Roughly speaking, the basic idea of boosting is to let later learners try to correct the mistakes made by earlier learners, and this is accomplished by deriving in each round a new data distribution which makes the earlier mistakes more evident. The base learners should be able to learn with specific distributions; this is usually accomplished by re-weighting or re-sampling the training examples according to the data distribution in each round. Such a learning process is very similar to residual minimization, and it has a close relation to additive models, inspiring an interpretation that AdaBoost is a stagewise estimation procedure for fitting an additive logistic regression model with an exponential loss [29.94]. Notice that AdaBoost was designed for binary classification, but it has many variants for multi-class problems [29.93, 95, 96].

It has been proved [29.93] that the generalization error of AdaBoost is upper bounded by

$$\epsilon \leq \epsilon_D + \tilde{O}\left(\sqrt{\frac{dT}{m}}\right) . \tag{29.45}$$

with probability at least $1 - \delta$, where $\epsilon_D$ is the error on the training sample $D$, $d$ is the VC-dimension of base learners, $m$ is the number of training samples, and $\tilde{O}(\cdot)$ is used instead of $O(\cdot)$ to hide logarithmic terms and constant factors. This generalization bound implies that the complexity $d$ of base learners and the number $T$ of learning rounds need to be constrained; otherwise AdaBoost will overfit. Empirical studies, however, show that AdaBoost often seems resistant to overfitting; that is, the test error often tends to decrease even after the training error reaches zero.

*Algorithm 29.1 The AdaBoost Algorithm*

**Input:** data set $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$;
Base learning algorithm $\mathfrak{L}$; number of learning rounds $T$.

**Process:**

1: $\mathcal{D}_1(x) = 1/m$. % initialize the weight distribution
2: **for** $t = 1, \ldots, T$:
3: $h_t = \mathfrak{L}(D, \mathcal{D}_t)$; % train a classifier $h_t$ from $D$ under distribution $\mathcal{D}_t$
4: $\epsilon_t = P_{x \approx \mathcal{D}_t}(h_t(x) \neq f(x))$; % evaluate the error of $h_t$
5: **if** $\epsilon_t > 0.5$ **then break**

6: $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$; % determine the weight of $h_t$

7: $\mathcal{D}_{t+1}(x) = \frac{\mathcal{D}_t(x)\exp(-\alpha_t f(x)h_t(x))}{Z_t}$ % update the distribution, where $Z_t$ is % a normalization factor which enables $\mathcal{D}_{t+1}$ to be a distribution

8: **end**

**Output:** $H(x) = \text{sign}\left(\sum_{t=1}^{T}\alpha_t h_t(x)\right)$

### 29.6.2 Boosting

For binary classification, formally, $f(x) \in \{-1, +1\}$, the margin of the classifier $h$ on the instance $x$ is defined as $f(x)h(x)$, and similarly, the margin of the ensemble $H(x) = \sum_{t=1}^{T}\alpha_t h_t(x)$ is $f(x)H(x) = \sum_{t=1}^{T}\alpha_t f(x)h_t(x)$, whereas the *normalized margin* is

$$f(x)H(x) = \frac{\sum_{t=1}^{T}\alpha_t f(x)h_t(x)}{\sum_{t=1}^{T}\alpha_t}. \tag{29.46}$$

where $\alpha_t$ are the weights of base learners. Given any threshold $\theta > 0$ of margin over the training sample $D$, it was proved in [29.97] that the generalization error of the ensemble is bounded with probability at least $1-\delta$ by

$$\epsilon \le 2^T \prod_{t=1}^{T}\sqrt{\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}} + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2} + \ln\frac{1}{\delta}}\right). \tag{29.47}$$

where $\epsilon_t$ is the training error of the base learner $h_t$. This bound implies that when other variables are fixed, the larger the margin over the training set, the smaller the generalization error. Thus, [29.97] argued that AdaBoost tends to be resistant to overfitting because it can increase the ensemble margin even after the training error reaches zero.

This margin-based explanation seems reasonable; however, it was later questioned [29.98] by the fact that (29.47) depends heavily on the minimum margin, whereas there are counterexamples where an algorithm is able to produce uniformly larger minimum margins than AdaBoost, but the generalization performance drastically decreases. From then on, there was much debate about whether the margin-based explanation holds; more details can be found in [29.14].

One drawback of AdaBoost lies in the fact that it is very sensitive to noise. Great efforts have been devoted to address this issue [29.99, 100]. For example, RobustBoost [29.101] tries to improve the noise tolerance ability by boosting the normalized classification margin, which was believed to be closely related to the generalization error.

### 29.6.3 Bagging

In contrast to *sequential ensemble methods* such as boosting where the base learners are generated in a sequential style to exploit the *dependence* between the base learners, bagging [29.99] is a kind of *parallel ensemble method* where the base learners are generated in parallel, attempting to exploit the *independence* between the base learners.

The name bagging came from the abbreviation of *Bootstrap AGGregatING*. Algorithm 29.2 shows its pseudo-code, where $\mathbb{I}$ is the indicator function. Bagging applies *bootstrap sampling* [29.102] to obtain multiple different data subsets for training the base learners. Given $m$ training examples, a set of $m$ training examples is generated by sampling with replacement; some original examples appear more than once, whereas some do not present. By applying the process $T$ times, $T$ such sets are obtained, and then each set is used for training a base learner. Bagging can deal with binary as well as multi-class problems by using *majority voting* to combine base learners; it can also be applied to regression by using *averaging* for combination.

*Algorithm 29.2 The Bagging Algorithm*

**Input:** Data set $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$;
Base learning algorithm $\mathfrak{L}$;
Number of base learners $T$.

**Process:**

1: **for** $t = 1, \ldots, T$:

2: $h_t = \mathfrak{L}(D, \mathcal{D}_{bs})$ % $\mathcal{D}_{bs}$ is bootstrap distribution

3: **end**

**Output:** $H(x) = \arg\max_{y\in\mathcal{Y}}\sum_{t=1}^{T}\mathbb{I}(h_t(x) = y)$

Theoretical analysis [29.99, 103, 104] shows that Bagging is particularly effective with *unstable* base learners (whose performance will change significantly with even slight variation of the training sample), such as decision trees, because it has a tremendous variance reduction effect, whereas it is not wise to apply Bagging to stable learners, such as nearest neighbor classifiers.

A prominent extension of Bagging is the random forest method [29.105], which has been successfully deployed in many real tasks. Random forest incorporates a randomized feature selection process in constructing the individual decision trees. For each individual decision tree, at each step of split selection, it randomly selects a feature subset, and then executes conventional split selection within the feature subset. The recommended size of feature subsets is the logarithm of the number of all features [29.105].

### 29.6.4 Stacking

Stacking [29.106–108] trains a *meta-learner* (or *second-level learner*), to combine the individual learners (or *first-level learners*). First-level learners are often generated by different learning algorithms, and therefore, stacked ensembles are often heterogeneous, although it is also possible to construct homogeneous stacked ensembles. Also, one similar approach was proposed in IJCNN1991 [29.109], with *meta-learner* referred to by a different name called *associative switch*, which is learned from examples for combining multiple classifiers.

Stacking can be viewed as a generalized framework of many ensemble methods, and can also be regarded as a specific combination method, i. e., *combining by learning*. It uses the original training examples to construct the first-level learners, and then generates a new data set to train the meta-learner, where the first-level learners' outputs are used as input features whereas the original labels are used as labels. Notice that there will be a high risk of overfitting if the exact data that are used to train the first-level learners are also used to generate the new data set for the meta-learner. Hence, it is recommended to exclude the training examples for the first-level learners from the data that are used for the meta-learner, and a cross-validation procedure is usually used.

It is crucial to consider the types of features for the new training data, and the types of learning algorithms for the meta-learner [29.106]. It has been suggested [29.110] to use class probabilities instead of crisp class labels as features for the new data, and to use multi-response linear regression (MLR) for the meta-learner. It has also been suggested [29.111] to use different sets of features for the linear regression problems in MLR.

If stacking (and many other ensemble methods) is simply viewed as assigning weights to combine different models, then it is closely related to *Bayes model averaging* (BMA), which assigns weights to models based on posterior probabilities. In theory, if the correct data generation model is in consideration and if the noise level is low, BMA is never worse and often better than stacking. In practice, however, BMA rarely performs better than stacking, because the correct data generalization model is usually unknown, whereas BMA is quite sensitive to model approximation error [29.112].

### 29.6.5 Diversity

If the base learners are independent, an amazing combination effect will occur. Taking binary classification, for an example, suppose each base learner has an independent generalization error $\epsilon$ and $T$ learners are combined via majority voting. Then, the ensemble makes an error only when at least half of its base learners make errors. Thus, by *Hoeffding inequality*, the generalization error of the ensemble is

$$\sum_{k=0}^{\lfloor T/2 \rfloor} \binom{T}{k} (1-\epsilon)^k \epsilon^{T-k} \leq \exp\left(-\frac{1}{2} T (2\epsilon - 1)^2\right) , \tag{29.48}$$

which implies that the generalization error decreases exponentially to the ensemble size $T$, and ultimately approaches zero as $T$ approaches infinity.

It is practically impossible to obtain really independent base learners, but it is generally accepted that to construct a good ensemble, the base learners should be as accurate as possible, and as *diverse* as possible. This has also been confirmed by *error-ambiguity decomposition* and *bias-variance-covariance decomposition* [29.113–115]. Generating diverse base learners, however, is not easy, because these learners are generated from the same training data for the same learning problem, and thus they are usually highly correlated. Actually, we need to require that the base learners must not be very poor; otherwise their combination may even worsen the performance.

Usually, combining only accurate learners is often worse than combining some accurate ones together with some relatively weak ones, because the complementarity is more important than pure accuracy. Notice that it is possible to do some selection to construct a smaller but stronger ensemble after obtaining all base learners [29.116], possibly because this way makes it easier to trade off between individual performance and diversity.

Unfortunately, there is not yet a clear understanding about diversity although it is crucial for ensemble methods. Many efforts have been devoted to designing diversity measures, however, none of them is well-accepted [29.14, 117]. In practice, heuristics are usually employed to generate diversity, and popular strategies include manipulating data samples, input features, learning parameters, and output representations [29.14].

## 29.7 Feature Selection and Extraction

Real-world data are often high-dimensional and contain many spurious features. For example, in face recognition, an image of size $m \times n$ is often represented as a vector in $R^{mn}$, which can be very high-dimensional for typical values of $m$ and $n$. Similarly, biological databases such as microarray data can have thousands or even tens of thousands of genes as features. Such a large number of features can easily lead to the curse of dimensionality and severe overfitting. A simple approach is to manually remove irrelevant features from the data. However, this may not be feasible in practice. Hence, automatic dimensionality reduction techniques, in the form of either feature selection or feature extraction, play a fundamental role in many machine learning problems.

Feature selection selects only a relevant subset of features for use with the model. In feature selection, the features may be scored either individually or as a subset. Not only can feature selection improve the generalization performance of the resultant classifier, the use of fewer features is also less computationally expensive and thus implies faster testing. Moreover, it can eliminate the need to collect a large number of irrelevant and redundant features, and thus reduces cost. The discovery of a small set of highly predictive variables also enhances our understanding of the underlying physical, biological, or natural processes, beyond just the building of accurate *black-box* predictors.

Feature selection and extraction has been a classic topic in the literature of pattern recognition for several decades; many results obtained before the 1980s are systematically summarized in [29.118]. Reviews on further studies in the recent three decadse are referred to [29.119–121]. Roughly, feature selection methods can be classified into three main paradigms: filters, wrappers, and the embedded approach [29.120]. Filters score the usefulness of the feature subset obtained as a pre-processing step. Commonly used scores include mutual information and the inter/intra class distance. This filtering step is performed independently of the classifier and is typically least computationally expensive among the three paradigms. Wrappers, on the other hand, score the feature subsets according to their prediction performance when used with the classifier. In other words, the classifier is trained on each of the candidate feature subsets, and the one with the best score is then selected. However, as the number of candidate feature subsets can be very large, this approach is computationally expensive, though it is also expected to perform better than filters. Both filters and wrappers rely on search strategies to guide the search for the *best* feature subset. While a large number of search strategies can be used, one is often limited to the computationally simple greedy strategies: (i) forward, in which features are added to the candidate set one by one; or (ii) backward, in which one starts with the full feature set and deletes features one by one. Finally, embedded methods combine feature selection with the classifier to create a sparse model. For example, one can use the $\ell_1$ regularizer which shrinks the coefficients of the useless features to zero, essentially removing them from the model. Another popular algorithm is called recursive feature elimination [29.122] for use with support vector machines. It repeatedly constructs a model and then removes those features with low weights. Empirically, embedded methods are often more efficient than filters and wrappers [29.120].

While most feature selection methods are supervised, there are also recent works on feature selection in the unsupervised learning setting. However, unsupervised feature selection is much more difficult due to the lack of label information to guide the search for relevant features. Most unsupervised feature selection methods are based on the filter approach [29.123–125], though there are also some studies on wrappers [29.126] and embedded approaches [29.124, 127–129].

Recently, feature selection in multi-task learning has been receiving increasing attention. Recall that the $\ell_1$ regularizer is commonly used to induce feature selection in single-task learning; this is extended to the mixed norms in MTL. Specifically, let $W = [w_1, w_2, \ldots, w_T]$, where $w_t \in R^d$ is the parameter associated with the $t$-th task. To enforce joint sparsity across the $T$ tasks, the $\ell_{\infty,1}$ norm of $W$ is used as the regular-

izer, i. e., $\|W\|_{\infty,1} = \sum_{j=1}^{d} \max_{1\le i\le T} |W_{ji}|$ [29.130]. In other words, one uses an $\ell_\infty$ norm on the rows of the $W$ to combine the contributions of each row (feature) from all the tasks, and then combine the features by using the $\ell_1$ norm, which, because of its sparsity-encouraging property, leads to only a few nonzero rows of $W$.

Instead of only selecting a subset from the existing set of features, feature extraction aims at extracting a set of new features from the original features. This can be viewed as performing dimensionality reduction that maps the original features to a new lower-dimensional feature space, while ensuring that the overall structure of the data points remains intact. The unsupervised methods previously introduced in Sect. 29.3.1 can all be used for feature extraction. The classic ones consist of *principal component analysis* (PSA) and *principal subspace analysis* (PSA), and their complementary counterparts *minor component analysis* (MCA) and *minor subspace analysis* (MSA), as well as the closely related *factor analysis* (FA), while independent component analysis (ICA) and *non-Gaussian factor analysis* (NFA) are further developments of PCA and FA, respectively. Another popular further development of PCA is kernel principal component analysis (KPCA) [29.131].

Moreover, feature extraction and unsupervised learning are coordinately conducted in many learning tasks, such as local factor analysis (LFA) in Sect. 29.3.2, nonnegative matrix factorization (NMF) and manifold learning in Sect. 29.3.3, temporal and hierarchical learning in Sect. 29.3.4, as well as other latent factor featured methods. Furthermore, the use of supervised information can lead to even better discriminative features for classification problems. Linear discriminant analysis (LDA) is the most classic example, which results in the Bayes optimal transform direction in the special case that the two classes are normally distributed with the same covariance. Learning multiple layer perceptron or neural networks can be regarded as nonlinear extensions of LDA, with hidden units extracting optimal features for supervised classification and regression.



## References


29.1 H. Simon: Why should machines learn? In: *Machine Learning. An Artificial Intelligence Approach*, ed. by I.R. Anderson, R.S. Michalski, J.G. Carbonell, T.M. Mitchell (Tioga Publ., Palo Alto 1983)

29.2 L. Xu: Bayesian Ying Yang learning, Scholarpedia **2**(3), 1809 (2007)

29.3 L. Xu: Bayesian Ying-Yang system, best harmony learning, and five action circling, Front. Electr. Electr. Eng. China **5**(3), 281–328 (2010)

29.4 L. Xu, S. Klasa, A. Yuille: Recent advances on techniques static feed-forward networks with supervised learning, Int. J. Neural Syst. **3**(3), 253–290 (1992)

29.5 L. Xu: Learning algorithms for RBF functions and subspace based functions. In: *Handbook of Research on Machine Learning, Applications and Trends: Algorithms, Methods and Techniques*, ed. by E. Olivas, J.D.M. Guerrero, M.M. Sober, J.R.M. Benedito, A.J.S. López (Inform. Sci. Ref., Hershey 2009) pp. 60–94

29.6 L. Xu: Several streams of progresses on unsupervised learning: A tutorial overview, Appl. Inf. **1** (2013)

29.7 A. Jain: Data clustering: 50 years beyond k-means, Pattern Recognit. Lett. **31**, 651–666 (2010)

29.8 H. Kriegel, P. Kroger, A. Zimek: Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering, ACM Trans. Knowl. Discov. Data **3**(1), 1 (2009)

29.9 H. Yin: Advances in adaptive nonlinear manifolds and dimensionality reduction, Front. Electr. Electr. Eng. China **6**(1), 72–85 (2011)

29.10 T.T. Kohonen Honkela: Kohonen network, Scholarpedia **2**(1), 1568 (2007)

29.11 L. Xu, J. Neufeld, B. Larson, D. Schuurmans: Maximum margin clustering, Adv. Neural Inf. Process. Syst. (2004) pp. 1537–1544

29.12 K. Zhang, I. Tsang, J. Kwok: Maximum margin clustering made practical, IEEE Trans. Neural Netw. **20**(4), 583–596 (2009)

29.13 Y.-F. Li, I. Tsang, J. Kwok, Z.-H. Zhou: Tighter and convex maximum margin clustering, Proc. 12th Int. Conf. Artif. Intell. Stat. (2009)

29.14 Z.-H. Zhou: *Ensemble Methods: Foundations and Algorithms* (Taylor Francis, Boca Raton 2012)

29.15 G. Tsoumakas, I. Katakis, I. Vlahavas: Mining multi-label data. In: *Data Mining and Knowledge Discovery Handbook*, 2nd edn., ed. by O. Maimon, L. Rokach (Springer, Berlin, Heidelberg 2010)

29.16 C. Silla, A. Freitas: A survey of hierarchical classification across different application domains, Data Min. Knowl. Discov. **22**(1/2), 31–72 (2010)

29.17 W. Bi, J. Kwok: Multi-label classification on tree- and DAG-structured hierarchies, Proc. 28th Int. Conf. Mach. Learn. (2011)

29.18 W. Bi, J. Kwok: Hierarchical multilabel classification with minimum Bayes risk, Proc. Int. Conf. Data Min. (2012)

29.19 W. Bi, J. Kwok: Mandatory leaf node prediction in hierarchical multilabel classification, Adv. Neural Inf. Process. Syst. (2012)

29.20 T.G. Dietterich, R.H. Lathrop, T. Lozano-Pérez: Solving the multiple-instance problem with axis-parallel rectangles, Artif. Intell. **89**(1-2), 31–71 (1997)

29.21 Z.-H.M.-L. Zhou Zhang: Solving multi-instance problems with classifier ensemble based on constructive clustering, Knowl. Inf. Syst. **11**(2), 155–170 (2007)

29.22 Z.-H. Zhou, Y.-Y. Sun, Y.-F. Li: Multi-instance learning by treating instances as non-i.i.d. samples, Proc. 26th Int. Conf. Mach. Learn. (2009) pp. 1249–1256

29.23 Z.-H. Zhou, J.-M. Xu: On the relation between multi-instance learning and semi-supervised learning, Proc. 24th Int. Conf. Mach. Learn. (2007) pp. 1167–1174

29.24 N. Weidmann, E. Frank, B. Pfahringer: A two-level learning method for generalized multi-instance problem, Proc. 14th Eur. Conf. Mach. Learn. (2003) pp. 468–479

29.25 S.D. Scott, J. Zhang, J. Brown: On generalized multiple-instance learning, Int. J. Comput. Intell. Appl. **5**(1), 21–35 (2005)

29.26 Z.-H. Zhou, M.-L. Zhang, S.-J. Huang, Y.-F. Li: Multi-instance multi-label learning, Artif. Intell. **176**(1), 2291–2320 (2012)

29.27 J. Foulds, E. Frank: A review of multi-instance learning assumptions, Knowl. Eng. Rev. **25**(1), 1–25 (2010)

29.28 L. Xu, A. Krzyzak, C. Suen: Several methods for combining multiple classifiers and their applications in handwritten character recognition, IEEE Trans. Syst. Man Cybern. SMC **22**(3), 418–435 (1992)

29.29 J. Kittler, M. Hatef, R. Duin, J. Matas: On combining classifiers, IEEE Trans. Pattern Anal. Mach. Intell. **20**(3), 226–239 (1998)

29.30 L. Xu, S.I. Amari: Combining classifiers and learning mixture-of-experts. In: *Encyclopedia of Artificial Intelligence*, ed. by J. Dopioco, J. Dorado, A. Pazos (Inform. Sci. Ref., Hershey 2008) pp. 318–326

29.31 A. Blum, T. Mitchell: Combining labeled and unlabeled data with co-training, Proc. 11th Annu. Conf. Comput. Learn. Theory (1998) pp. 92–100

29.32 S. Abney: Bootstrapping, Proc. 40th Annu. Meet. Assoc. Comput. Linguist. (2002) pp. 360–367

29.33 M.-F. Balcan, A. Blum, K. Yang: Co-training and expansion: Towards bridging theory and practice, Adv. Neural Inf. Process. Syst. (2005) pp. 89–96

29.34 W. Wang, Z.-H. Zhou: A new analysis of co-training, Proc. 27th Int. Conf. Mach. Learn. (2010) pp. 1135–1142

29.35 Z.-H. Zhou, D.-C. Zhan, Q. Yang: Semi-supervised learning with very few labeled training examples, Proc. 22nd AAAI Conf. Artif. Intell. (2007) pp. 675–680

29.36 W. Wang, Z.-H. Zhou: Multi-view active learning in the non-realizable case, Adv. Neural Inf. Process. Syst. (2010) pp. 2388–2396

29.37 R. Caruana: Multitask learning, Mach. Learn. **28**(1), 41–75 (1997)

29.38 T. Evgeniou, M. Pontil: Regularized multi-task learning, Proc. 10th Int. Conf. Know. Discov. Data Min. (2004) pp. 109–117

29.39 T. Evgeniou, C.A. Micchelli, M. Pontil: Learning multiple tasks with kernel methods, J. Mach. Learn. Res. **6**, 615–637 (2005)

29.40 A. Argyriou, T. Evgeniou, M. Pontil: Multi-task feature learning, Adv. Neural Inf. Process. Syst. (2007) pp. 41–48

29.41 A. Argyriou, T. Evgeniou, M. Pontil: Convex multi-task feature learning, Mach. Learn. **73**(3), 243–272 (2008)

29.42 T. Kato, H. Kashima, M. Sugiyama, K. Asai: Multi-task learning via conic programming, Adv. Neural Inf. Process. Syst. (2007) pp. 737–744

29.43 R. Ando, T. Zhang: A framework for learning predictive structures from multiple tasks and unlabeled data, J. Mach. Learn. Res. **6**, 1817–1853 (2005)

29.44 Y. Zhang, D.-Y. Yeung: A convex formulation for learning task relationships in multi-task learning, Proc. 24th Conf. Uncertain. Artif. Intell. (2010) pp. 733–742

29.45 L. Jacob, F. Bach, J. Vert: Clustered multi-task learning: A convex formulation, Adv. Neural Inf. Process. Syst. (2008) pp. 745–752

29.46 L.J. Zhong Kwok: Convex multitask learning with flexible task clusters, Proc. 29th Int. Conf. Mach. Learn. (2012)

29.47 J. Chen, J. Zhou, J. Ye: Integrating low-rank and group-sparse structures for robust multi-task learning, Proc. 17th Int. Conf. Knowl. Discov. Data Min. (2011) pp. 42–50

29.48 S. Pan, J. Kwok, Q. Yang, J. Pan: Adaptive localization in A dynamic WiFi environment through multi-view learning, Proc. 22nd AAAI Conf. Artif. Intell. (2007) pp. 1108–1113

29.49 S. Pan, J. Kwok, Q. Yang: Transfer learning via dimensionality reduction, Proc. 23rd AAAI Conf. Artif. Intell. (2008)

29.50 S. Pan, I. Tsang, J. Kwok, Q. Yang: Domain adaptation via transfer component analysis, IEEE Trans. Neural Netw. **22**(2), 199–210 (2011)

29.51 W. Dai, Q. Yang, G. Xue, Y. Yu: Boosting for transfer learning, Proc. 24th Int. Conf. Mach. Learn. (2007) pp. 193–200

29.52 J. Huang, A. Smola, A. Gretton, K. Borgwardt, B. Schölkopf: Correcting sample selection bias by unlabeled data, Adv. Neural Inf. Process. Syst. (2007) pp. 601–608

29.53 M. Sugiyama, S. Nakajima, H. Kashima, P.V. Buenau, M. Kawanabe: Direct importance estimation with model selection and its application to covariate shift adaptation, Adv. Neural Inf. Process. Syst. (2008)

29.54 C. Elkan: The foundations of cost-sensitive learning, Proc. 17th Int. Jt. Conf. Artif. Intell. (2001) pp. 973–978

29.55 Z.-H. Zhou, X.-Y. Liu: On multi-class cost-sensitive learning, Proc. 21st Natl. Conf. Artif. Intell. (2006) pp. 567–572

29.56 X.-Y. Liu, Z.-H. Zhou: Learning with cost intervals, Proc. 16th Int. Conf. Knowl. Discov. Data Min. (2010) pp. 403–412

29.57 P.D. Turney: Types of cost in inductive concept learning, Proc. 17th Int. Conf. Mach. Learn. (2000) pp. 15–21

29.58 L. Xu: On essential topics of BYY harmony learning: Current status, challenging issues, and gene analysis applications, Front. Electr. Elect. Eng. China **7**(1), 147–196 (2012)

29.59 L. Xu: Semi-blind bilinear matrix system, BYY harmony learning, and gene analysis applications, Proc. 6th Int. Conf. New Trends Inf. Sci. Serv. Sci. Data Min. (2012) pp. 661–666

29.60 L. Xu: Independent subspaces. In: *Encyclopedia of Artificial Intelligence*, ed. by J. Dopioco, J. Dorado, A. Pazos (Inform. Sci. Ref., Hershey 2008) pp. 903–912

29.61 L. Xu: Independent component analysis and extensions with noise and time: A Bayesian Ying-Yang learning perspective, Neural Inf. Process. Lett. Rev. **1**(1), 1–52 (2003)

29.62 L. Xu: One-bit-matching ICA theorem, convex-concave programming, and distribution approximation for combinatorics, Neural Comput. **19**, 546–569 (2007)

29.63 S. Tu, L. Xu: Parameterizations make different model selections: Empirical findings from factor analysis, Front. Electr. Electr. Eng. China **6**(2), 256–274 (2011)

29.64 P. Williams: Bayesian regularization and pruning using A Laplace prior, Neural Comput. **7**(1), 117–143 (1995)

29.65 R. Tibshirani: Regression shrinkage and selection via the lasso, J. R. Stat. Soc. Ser. B: Methodol. **58**(1), 267–288 (1996)

29.66 M. Figueiredo, A. Jain: Unsupervised learning of finite mixture models, IEEE Trans. Pattern Anal. Mach. Intell. **24**(3), 381–396 (2002)

29.67 C. McGrory, D. Titterington: Variational approximations in Bayesian model selection for finite mixture distributions, Comput. Stat. Data Anal. **51**(11), 5352–5367 (2007)

29.68 A. Corduneanu, C. Bishop: Variational Bayesian model selection for mixture distributions, Proc. 8th Int. Conf. Artif. Intell. Stat. (2001) pp. 27–34

29.69 L. Xu: Rival penalized competitive learning, Scholarpedia **2**(8), 1810 (2007)

29.70 L. Xu: A unified perspective and new results on RHT computing, mixture based learning, and multi-learner based problem solving, Pattern Recognit. **40**(8), 2129–2153 (2007)

29.71 L. Xu: BYY harmony learning, structural RPCL, and topological self-organizing on mixture models, Neural Netw. **8–9**, 1125–1151 (2002)

29.72 L. Xu, M. Jordan, G. Hinton: An alternative model for mixtures of experts, Adv. Neural Inf. Process. Syst. (1995) pp. 633–640

29.73 D. Lee, H. Seung: Learning the parts of objects by non-negative matrix factorization, Nature **401**(6755), 788–791 (1999)

29.74 S. Madeira: A. Oliveira, Biclustering algorithms for biological data analysis: A survey, IEEE Trans. Comput. Biol. Bioinform. **1**(1), 25–45 (2004)

29.75 S. Tu, R. Chen, L. Xu: A binary matrix factorization algorithm for protein complex prediction, Proteome Sci. **9**(Suppl 1), S18 (2011)

29.76 X. He, P. Niyogi: Locality preserving projections, Adv. Neural Inf. Process. Syst. (2003) pp. 152–160

29.77 X. He, B. Lin: Tangent space learning and generalization, Front. Electr. Electr. Eng. China **6**(1), 27–42 (2011)

29.78 M.M. Meila Jordan: Learning with mixtures of trees, J. Mach. Learn. Res. **1**, 1–48 (2000)

29.79 J. Pearl: Fusion, propagation and structuring in belief networks, Artif. Intell. **29**(3), 241–288 (1986), Sep.

29.80 L. Xu, J. Pearl: Structuring causal tree models with continuous variables, Proc. 3rd Annu. Conf. Uncertain. Artif. Intell. (1987) pp. 170–179

29.81 A. Barto: Temporal difference learning, Scholarpedia **2**(11), 1604 (2007)

29.82 F. Woergoetter, B. Porr: Reinforcement learning, Scholarpedia **3**(3), 1448 (2008)

29.83 O. Chapelle, B. Schölkopf, A. Zien: Semi-Supervised Learning (MIT, Cambridge 2006)

29.84 X. Zhu: *Semi-supervised learning literature survey* (Univ. of Wisconsin, Madison 2008)

29.85 Z.-H. Zhou, M. Li: Semi-supervised learning by disagreement, Knowl. Inform. Syst. **24**(3), 415–439 (2010)

29.86 Z.-H. Zhou: When semi-supervised learning meets ensemble learning, Front. Electr. Electr. Eng. China **6**(1), 6–16 (2011)

29.87 V.N. Vapnik: *Statistical Learning Theory* (Wiley, New York 1998)

29.88 Y.-F. Li, Z.-H. Zhou: Towards making unlabeled data never hurt, Proc. 28th Int. Conf. Mach. Learn. (2011) pp. 1081–1088

29.89 A. Fred, A.K. Jain: Data clustering using evidence accumulation, Proc. 16th Int. Conf. Pattern Recognit. (2002) pp. 276–280

29.90 A. Strehl, J. Ghosh: Cluster ensembles – A knowledge reuse framework for combining multiple partitions, J. Mach. Learn. Res. **3**, 583–617 (2002)

29.91 R. Jacobs, M. Jordan, S. Nowlan, G. Hinton: Adaptive mixtures of local experts, Neural Comput. **3**, 79–87 (1991)

29.92 R.E. Schapire: The strength of weak learnability, Mach. Learn. **5**(2), 197–227 (1990)

29.93 Y. Freund, R.E. Schapire: A decision-theoretic generalization of on-line learning and an application to boosting, J. Comput. Syst. Sci. **55**(1), 119–139 (1997)

29.94 J. Friedman, T. Hastie, R. Tibshirani: Additive logistic regression: A statistical view of boosting (with discussions), Ann. Stat. **28**(2), 337–407 (2000)

29.95 R.E. Schapire, Y. Singer: Improved boosting algorithms using confidence-rated predictions, Mach. Learn. **37**(3), 297–336 (1999)

29.96 J. Zhu, S. Rosset, H. Zou, T. Hastie: Multi-class AdaBoost, Stat. Interface **2**, 349–360 (2009)

29.97 R.E. Schapire, Y. Freund, P. Bartlett, W.S. Lee: Boosting the margin: A new explanation for the effectiveness of voting methods, Ann. Stat. **26**(5), 1651–1686 (1998)

29.98 L. Breiman: Prediction games and arcing algorithms, Neural Comput. **11**(7), 1493–1517 (1999)

29.99 L. Breiman: Bagging predictors, Mach. Learn. **24**(2), 123–140 (1996)

29.100 C. Domingo, O. Watanabe: Madaboost: A modification of AdaBoost, Proc. 13th Annu. Conf. Comput. Learn. Theory (2000) pp. 180–189

29.101 Y. Freund: An adaptive version of the boost by majority algorithm, Mach. Learn. **43**(3), 293–318 (2001)

29.102 B. Efron, R. Tibshirani: *An Introduction to the Bootstrap* (Chapman Hall, New York 1993)

29.103 A. Buja, W. Stuetzle: Observations on bagging, Stat. Sin. **16**(2), 323–351 (2006)

29.104 J.H.P. Friedman Hall: On bagging and nonlinear estimation, J. Stat. Plan. Inference **137**(3), 669–683 (2007)

29.105 L. Breiman: Random forests, Mach. Learn. **45**(1), 5–32 (2001)

29.106 D.H. Wolpert: Stacked generalization, Neural Netw. **5**(2), 241–260 (1992)

29.107 L. Breiman: Stacked regressions, Mach. Learn. **24**(1), 49–64 (1996)

29.108 P. Smyth, D. Wolpert: Stacked density estimation, Adv. Neural Inf. Process. Syst. (1998) pp. 668–674

29.109 L. Xu, A. Krzyzak, C. Sun: Associative switch for combining multiple classifiers, Int. Jt. Conf. Neural Netw. (1991) pp. 43–48

29.110 K.M. Ting, I.H. Witten: Issues in stacked generalization, J. Artif. Intell. Res. **10**, 271–289 (1999)

29.111 A.K. Seewald: How to make stacking better and faster while also taking care of an unknown weakness, Proc. 19th Int. Conf. Mach. Learn. (2002) pp. 554–561

29.112 B. Clarke: Comparing Bayes model averaging and stacking when model approximation error cannot be ignored, J. Mach. Learn. Res. **4**, 683–712 (2003)

29.113 A. Krogh, J. Vedelsby: Neural network ensembles, cross validation, and active learning, Adv. Neural Inf. Process. Syst. (1995) pp. 231–238

29.114 N.R. Ueda Nakano: Generalization error of ensemble estimators, Proc. IEEE Int. Conf. Neural Netw. (1996) pp. 90–95

29.115 G. Brown, J.L. Wyatt, P. Tino: Managing diversity in regression ensembles, J. Mach. Learn. Res. **6**, 1621–1650 (2005)

29.116 Z.-H. Zhou, J. Wu, W. Tang: Ensembling neural networks: Many could be better than all, Artif. Intell. **137**(1-2), 239–263 (2002)

29.117 L.I. Kuncheva, C.J. Whitaker: Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy, Mach. Learn. **51**(2), 181–207 (2003)

29.118 P. Devijver, J. Kittler: *Pattern Recognition: A Statistical Approach* (Prentice Hall, New York 1982)

29.119 Y. Saeys, I. Inza, P. Larraaga: A review of feature selection techniques in bioinformatics, Bioinformatics **19**(23), 2507–2517 (2007)

29.120 I. Guyon, A. Elisseeff: An introduction to variable and feature selection, J. Mach. Learn. Res. **3**, 1157–1182 (2003)

29.121 A. Jain, R. Duin, J. Mao: Statistical pattern recognition: A review, IEEE Trans. Pattern Anal. Mach. Intell. **22**, 1 (2000)

29.122 I. Guyon, J. Weston, S. Barnhill, V. Vapnik: Gene selection for cancer classification using support vector machines, Mach. Learn. **46**(1-3), 389–422 (2002)

29.123 M. Dash, K. Choi, P. Scheuermann, H. Liu: Feature selection for clustering – A filter solution, Proc. 2nd Int. Conf. Data Min. (2002) pp. 115–122, Dec.

29.124 M. Law, M. Figueiredo, A. Jain: Simultaneous feature selection and clustering using mixture models, IEEE Trans. Pattern Anal. Mach. Intell. **26**(9), 1154–1166 (2004)

29.125 P. Mitra, C. Murthy, S.K. Pal: Unsupervised feature selection using feature similarity, IEEE Trans. Pattern Anal. Mach. Intell. **24**(3), 301–312 (2002)

29.126 V. Roth: The generalized LASSO, IEEE Trans. Neural Netw. **15**(1), 16–28 (2004)

29.127 C. Constantinopoulos, M. Titsias, A. Likas: Bayesian feature and model selection for Gaussian mixture models, IEEE Trans. Pattern Anal. Mach. Intell. **28**(6), 1013–1018 (2006)

29.128 J. Dy, C. Brodley: Feature selection for unsupervised learning, J. Mach. Learn. Res. **5**, 845–889 (2004)

29.129 B. Zhao, J. Kwok, F. Wang, C. Zhang: Unsupervised maximum margin feature selection with manifold regularization, Proc. Int. Conf. Comput. Vis. Pattern Recognit. (2009)

29.130 B. Turlach, W. Venables, S. Wright: Simultaneous variable selection, Technometrics **27**, 349–363 (2005)

29.131 B. Schölkopf, A. Smola: *Learning with Kernels* (MIT Press, Cambridge 2002)