Chapter 6
Machine Learning

Machine learning is a subset of artificial intelligence. This chapter presents first a machine learning tree, and then focuses on the matrix algebra methods in machine learning including single-objective optimization, feature selection, principal component analysis, and canonical correlation analysis together with supervised, unsupervised, and semi-supervised learning and active learning. More importantly, this chapter highlights selected topics and advances in machine learning: graph machine learning, reinforcement learning, Q-learning, and transfer learning.

6.1 Machine Learning Tree

Machine learning (ML) is a subset of artificial intelligence, which build a mathematical model based on sample data, known as "training data," in order to make predictions or decisions without being explicitly programmed to perform the task. Regarding learning, a good definition given by Mitchell [181] is

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

In machine learning, neural networks, support vector machines, and evolutionary computation, we are usually given a training set and a test set. By the *training set*, it will mean the union of the *labeled set* and the *unlabeled set* of examples available to machine learners. In comparison, *test set* consists of examples never seen before.

Let $(X_l, Y_l) = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ denote the labeled set, where $\mathbf{x}_i \in \mathbb{R}^D$ is the ith D-dimensional data vector and $y_i \in \mathbb{R}$ or $y_i \in \{1, \ldots, M\}$ is the corresponding label of the data vector $\mathbf{x}_i$. In regression problems, y_i is the regression or fitting of $\mathbf{x}_i$. In classification problems, y_i is the corresponding class label of $\mathbf{x}_i$ among the M classes of targets. The labeled data $\mathbf{x}_i$, $i = 1, \ldots, l$ are observed by the

X.-D. Zhang, *A Matrix Algebra Approach to Artificial Intelligence*,
https://doi.org/10.1007/978-981-15-2770-8_6

user, while $y_i, i = 1, \ldots, l$ are labeled by data labeling experts or supervisors. The unlabeled set is comprised of the data vectors and is denoted by $X_u = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\}$.

Machine learning aims to establish a regressor or classifier through learning the training set, and then to evaluate the performance of the regressor or classifier through the test set.

According to the nature of training data, we can classify machine learning as follows.

1. *Regular or Euclidean structured data learning*

   - *Supervised learning:* Given a training set consisting of labeled data (i.e., example inputs and their desired outputs) $(X_{\text{train}}, Y_{\text{train}}) = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$, supervised learning learns a general rule that maps inputs to outputs. This is like a "teacher" or a supervisor (data labeling expert) giving a student a problem (finding the mapping relationship between inputs and outputs) and its solutions (labeled output data) and telling that student to figure out how to solve other, similar problems: finding the mapping from the features of unseen samples to their correct labels or target values in the future.
   - *Unsupervised learning:* In unsupervised learning, the training set consists of the unlabeled set only $X_{\text{train}} = X_u = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\}$. The main task of the machine learner is to find the solutions on its own (i.e., patterns, structures, or knowledge in unlabeled data). This is like giving a student a set of patterns and asking him or her to figure out the underlying motifs that generated the patterns.
   - *Semi-supervised learning:* Given a training set $(X_{\text{train}}, Y_{\text{train}}) = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l y_l)\} \cup \{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}\}$ with $l \ll u$, i.e., we are given a small amount of labeled data together with a large amount of unlabeled data. Semi-supervised learning falls between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data). Depending on how the data is labeled, semi-supervised learning can be divided into the following categories:

     – *Self-training* is a semi-supervised learning using its own predictions to teach itself.
     – *Co-training* is a weakly semi-supervised learning for multi-view data using the co-training setting, and use their own predictions to teach themselves.
     – *Active learning* is a semi-supervised learning where the learner has some active or participatory role in determining which data points it will ask to be labeled by an expert or teacher.

   - *Reinforcement learning:* Training data (in the form of rewards and punishments) is given only as feedback to an artificial intelligence agent in a dynamic environment. This feedback between the learning system and the interaction experience is useful to improve performance in the task being learned. The machine learning based on data feedback is called reinforcement learning. Q-learning is a popular model-free reinforcement learning and learns a reward or punishment function (action-value functions, simply called Q-function).

- *Transfer learning:* In many real-world applications, the data distribution changes or data are outdated, and thus it is necessary to apply transfer learning for considering transfer of knowledge from the source domain to the target domain. Transfer learning includes but not limited to inductive transfer learning, transductive transfer learning, unsupervised transfer learning, multitask learning, self-taught transfer learning, domain adaptation, and EigenTransfer.

2. *Graph machine learning* is an irregular or non-Euclidean structured data learning, and learns the structure of a graph, called also graph construction, from training samples in semi-supervised and unsupervised learning cases.

The above classification of machine learning can be vividly represented by a machine learning tree, as shown in Fig. 6.1.

The machine learning tree is so-called because Fig. 6.1 looks like a tree after rotating it 90° to the left.

There are two basic tasks of machine learning: classification (for discrete data) and prediction (for continuous data).

Deep learning is to learn the internal law and representation level of sample data. The information at different hierarchy levels obtained in the learning process is very helpful to the interpretation of data such as text, image, and voice. Deep learning is a complex machine learning algorithm, which has achieved much better results in speech and image recognition than previous related technologies.

Limited to space, this chapter will not discuss deep learning, but focuses only on supervised learning, unsupervised learning, reinforcement learning, and transfer learning.

Before dealing with machine learning in detail, it is necessary to start with preparation knowledge of machine learning: its optimization problems, majorization-minimization algorithms, and how to boost a weak learning algorithm to a strong learning algorithm.

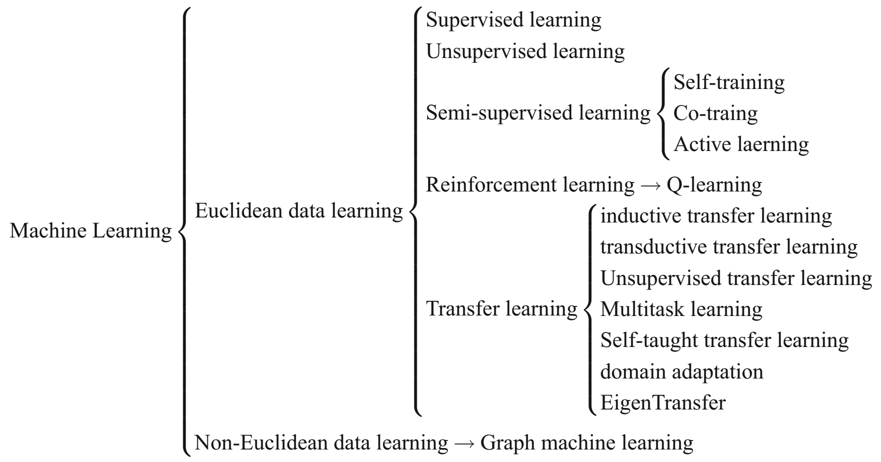

Fig. 6.1 Machine learning tree

6.2 Optimization in Machine Learning

Two of the pillars of machine learning are matrix algebra and optimization. Matrix algebra involves the matrix-vector representation and modeling of currently available data, and optimization involves the numerical computation of parameters for a system designed to make decisions on yet unseen data.

Optimization problems arise throughout machine learning. On optimization in machine learning, the following questions are naturally asked and important [31]:

1. How do optimization problems arise in machine learning applications, and what makes them challenging?
2. What have been the most successful optimization methods for large-scale machine learning, and why?
3. What recent advances have been made in the design of algorithms, and what are open questions in this research area?

This section focuses upon the most successful optimization methods for large-scale machine learning that involves very large data sets and for which the number of model parameters to be optimized is also large.

6.2.1 Single-Objective Composite Optimization

In supervised machine learning, we are given the collection of a data set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_i \in \mathbb{R}^d$ for each $i \in \{1, \ldots, N\}$ represents the feature vector of a target, and the scalar y_i is a label indicating whether a corresponding feature vector belongs ($y_i = 1$) or not ($y_i = -1$) to a particular class (i.e., topic of interest).

With a set of examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, define the *prediction function* with respect to the ith sample as

$$h(\mathbf{w}; \mathbf{x}_i) = \mathbf{w} \cdot \mathbf{x}_i = \langle \mathbf{w}, \mathbf{x}_i \rangle = \mathbf{w}^T \mathbf{x}_i, \tag{6.2.1}$$

where the predictor $\mathbf{w} = [w_1, \ldots, w_d]^T \in \mathbb{R}^d$ is a d-dimension parameter vector of machine learning model that should have the good "generalization ability."

Construct a machine learning program in which the performance of a prediction function $\hat{y}_i = h(\mathbf{w}; \mathbf{x}_i)$ is measured by counting how often the program prediction $f(\mathbf{x}_i; \mathbf{w})$ differs from the correct prediction y_i, i.e., $\hat{y}_i \neq y_i$. Therefore, we need to search for a predictor function that minimizes the frequency of observed mispredictions, called the *empirical risk*:

$$R_n(h) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}[h_i(\mathbf{x}_i) \neq y_i], \tag{6.2.2}$$

where

$$\mathbb{1}[A] = \begin{cases} 1, \text{ if } A \text{ is true;} \\ \\ 0, \text{ otherwise.} \end{cases} \tag{6.2.3}$$

The optimization in machine learning seeks to design a classifier/predictor $\mathbf{w}$ so that $\hat{y} = \langle \mathbf{w}, \mathbf{x} \rangle$ can provide the correct classification/prediction for any unknown input vector $\mathbf{x}$. The objective $f(\mathbf{w}; \mathbf{x})$ in machine learning can be represented as the following common form:

$$f(\mathbf{w}; \mathbf{x}) = l(\mathbf{w}; \mathbf{x}) + h(\mathbf{w}; \mathbf{x}), \tag{6.2.4}$$

where $l(\mathbf{w}; \mathbf{x})$ is the *loss function* and $h(\mathbf{w}; \mathbf{x})$ denotes the *regularization function*.

In this manner, the expected risk for a given $\mathbf{w}$ is the expected value of the loss function $l(\mathbf{w}; \mathbf{x})$ taken with respect to the distribution of $\mathbf{x}$ as follows [31]:

$$\text{(Expected Risk)} \qquad R(\mathbf{w}) = E\{l(\mathbf{w}; \mathbf{x})\} \tag{6.2.5}$$

since the true distribution of $\mathbf{x}$ is often unknown, the expected risk is commonly estimated by the empirical risk

$$\text{(Empirical Risk)} \qquad R_n(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} l_i(\mathbf{w}), \tag{6.2.6}$$

where $l_i(\mathbf{w}) = l(\mathbf{w}; \mathbf{x}_i)$.

If we were to try to minimize the empirical risk only, the method would be referred to as ERM (Empricial Risk Minimization). However, the empirical risk usually underestimates the true/expected risk by the *estimation error*, and thus we add a regularization term $h(w; x)$ to the composite function.

The corresponding *prediction error* is given by

$$e_i(\mathbf{w}) = e(\mathbf{w}; \mathbf{x}_i, y_i) = \mathbf{w}^T \mathbf{x}_i - y_i. \tag{6.2.7}$$

Then one can write the *empirical error* as the average of the prediction errors for all N samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$:

$$\text{(Empirical Error)} \qquad R_{\text{emp}}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} e_i(\mathbf{w}). \tag{6.2.8}$$

The loss function incurred by the parameter vector $\mathbf{w}$ can be represented as

$$l(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}|e_i(\mathbf{w})|^2 = \frac{1}{2N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2. \tag{6.2.9}$$

The above machine learning tasks such as classification and regression can be represented as solving a "*composite*" (additive) *optimization* problems

$$\min_{\mathbf{w}}\left\{f(\mathbf{w}) = l(\mathbf{w}) + h(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left(\mathbf{w}^T\mathbf{x}_i - y_i\right)^2 + h(\mathbf{w})\right\}, \tag{6.2.10}$$

where the objective function $f(\mathbf{x})$ is given by the sum of N component loss functions $f_i(\mathbf{w}) = |e_i(\mathbf{w})|^2 = (\mathbf{w}^T\mathbf{x}_i - y_i)^2$ and a possibly nonsmooth regularization function $h(\mathbf{w})$. It is assumed that each component loss function $f_i(\mathbf{w};\mathbf{x}_i) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable and the sum of N component loss functions $f(\mathbf{x})$ is strongly convex, while the regularization function $h : \mathbb{R}^d \to \mathbb{R}$ is proper, closed, and convex but not necessarily differentiable, and is used for preventing overfitting.

The composite optimization tasks exist widely in various artificial intelligence applications, e.g., image recognition, speech recognition, task allocation, text classification/processing, and so on.

To simplify notation, denote $\mathbf{g}_j(\mathbf{w}_k) = \mathbf{g}(\mathbf{w}_k;\mathbf{x}_j, y_j)$ and $f_j(\mathbf{w}_k) = f(\mathbf{w}_k;\mathbf{x}_j, y_j)$ hereafter. Optimization methods for machine learning fall into two broad categories [31]:

- *Stochastic method:* The prototypical stochastic method is the *stochastic gradient (SG) method* [214] defined by

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \alpha_k \nabla f_{i_k}(\mathbf{w}_k) \tag{6.2.11}$$

  for all $k \in \{1, 2, \ldots\}$, the index i_k, corresponding to the sample pair $(\mathbf{x}_{i_k}, y_{i_k})$, is chosen randomly from $\{1, \ldots, N\}$ and α_k is a positive stepsize. Each iteration of this method involves only the computation of the gradient $\nabla f_{i_k}(\mathbf{w}_k)$ corresponding to one sample.
- *Batch method:* The simplest form of batch method is the steepest descent algorithm:

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \alpha_k \nabla R_n(\mathbf{w}_k) = \mathbf{w}_k - \frac{\alpha_k}{N}\sum_{i=1}^{N}\nabla f_i(\mathbf{w}_k). \tag{6.2.12}$$

  The steepest descent algorithm is also referred to as the gradient, batch gradient, or full gradient method.

The gradient algorithms are usually represented as

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \mathbf{g}(\mathbf{w}_k). \tag{6.2.13}$$

Traditional gradient-based methods are effective for solving small-scale learning problems, but in the context of large-scale machine learning one of the core strategies of interest is the SG method proposed by Robbins and Monro [214]:

$$\mathbf{g}(\mathbf{w}_k) \leftarrow \frac{1}{|S_k|} \sum_{i \in S_k} \nabla f_i(\mathbf{w}_k), \tag{6.2.14}$$

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \alpha_k \mathbf{g}(\mathbf{w}_k), \tag{6.2.15}$$

for all $k \in \{1, 2, \ldots\}$ and $S_k \subseteq \{1, \ldots, N\}$ is a small subset of sample pairs $(\mathbf{x}_i, y_i)$ chosen randomly from $\{1, \ldots, N\}$ and α_k is a positive stepsize.

Algorithm 6.1 shows a generalized SG (mini-batch) method.

Algorithm 6.1 Stochastic gradient (SG) method [31]

```
1.  input: Sample pairs $(\mathbf{x}_j, y_j),\ j = 1, \ldots, N$.
2.  initialization: Choose an initial iterate $\mathbf{w}_1$.
3.  for $k = 1, 2, \ldots$ do
4.      Generate a realization of the random variables $(\mathbf{x}_i^{(k)}, y_i^{(k)})$, where $i \in S_k$ and $S_k \subseteq \{1, \ldots, N\}$
        is a small subset of samples.
5.      Compute a stochastic vector $\mathbf{g}(\mathbf{w}_k) \leftarrow \frac{1}{|S_k|} \sum_{i \in S_k} \nabla f(\mathbf{w}_k; \mathbf{x}_i^{(k)}, y_i^{(k)})$.
6.      Choose a stepsize $\alpha_k > 0$.
7.      Update the new iterate as $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \alpha_k \mathbf{g}(\mathbf{w}_k)$.
8.      exit: if $\mathbf{w}_{k+1}$ is converged.
9.  end for
10. output: $\mathbf{w} = \frac{1}{k+1} \sum_{i=1}^{k+1} \mathbf{w}_i$.
```

The SG iteration is very inexpensive, requiring only the evaluation of $\mathbf{g}(\mathbf{w}_k, \xi_k)$. However, since SG generates noisy iterate sequences that tend to oscillate around minimizers during the optimization process, a natural idea is to compute a corresponding sequence of *iterate averages*

$$\tilde{\mathbf{w}}_{k+1} \leftarrow \frac{1}{k+1} \sum_{i=1}^{k+1} \mathbf{w}_i, \tag{6.2.16}$$

where the averaged sequence $\{\tilde{\mathbf{w}}_k\}$ has no effect on the computation of the SG iterate sequence $\{\mathbf{w}_k\}$. Iterate averages would automatically possess less noisy behavior.

6.2.2 Gradient Aggregation Methods

Consider a common single-objective convex optimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^d} \{F(\mathbf{x}) = f(\mathbf{x}) + h(\mathbf{x})\}, \tag{6.2.17}$$

where the first term $f(\mathbf{x})$ is smooth and the second term $h(\mathbf{x})$ is possibly nonsmooth, which allows for the modeling of constraints.

In many applications in optimization, pattern recognition, signal processing, and machine learning, $f(\mathbf{x})$ has an additional structure:

$$f(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}) \tag{6.2.18}$$

which is the average of a number of convex functions f_i.

When N is large and when a solution with low to medium accuracy is sufficient, it is a common choice to perform classical stochastic methods, especially stochastic gradient descent (SGD) method for solving the optimization problem (6.2.17). The SGD method can be dated back to the 1951 seminal work of Robbins and Monro [214]. SGD selects an index $i \in \{1,\ldots,N\}$ uniformly at random, and then updates $\mathbf{x}$ using a stochastic estimate $\nabla f_i(\mathbf{x})$ of $\nabla f(\mathbf{x})$. Thanks to the computation of $\nabla f_i(\mathbf{x})$ that is N times cheaper than the computation of the full gradient $\nabla f(\mathbf{x})$, for optimization problems where N is very large, the per-iteration savings can be extremely large, spanning several orders of magnitude [151].

SG algorithms only use the current gradient information, while *gradient aggregation* is a method to reuse and/or revise previously computed gradient information. The goal of gradient aggregation is to achieve an improved convergence rate and a lower variance. In addition, if one maintains indexed gradient estimates in storage, then one can revise specific estimates as new information to be collected.

Three famous gradient aggregation methods are SAG, SVRG, and SAGA.

- *Stochastic average gradient* (SAG) method: If stochastic average gradient is used to replace iterate averages:

$$\text{SAG:}\quad \mathbf{g}(\mathbf{w}_k) = \frac{1}{N}\left(\nabla f_j(\mathbf{w}_k) - \nabla f_j(\mathbf{w}_{k-1}) + \sum_{i=1}^{N} \nabla f_i(\mathbf{w}_{[i]})\right), \tag{6.2.19}$$

  one gets the well-known stochastic average gradient (SAG) method [156, 222]. In iteration k of SAG, $\nabla f_j(\mathbf{w}_k) = \nabla f(\mathbf{w}_k; \mathbf{x}_j, y_j)$, $j \in \{1,\ldots,N\}$ is chosen from the current gradient at random, and $\nabla f_j(\mathbf{w}_{k-1}) = \nabla f(\mathbf{w}_{k-1}; \mathbf{x}_{[j]}, y_{[j]})$, $j \in \{1,\ldots,N\}$ is chosen from the past gradient at iteration $k-1$ randomly, while $\nabla f_i(\mathbf{w}_{[i]}) = \nabla f(\mathbf{w}_{[i]}; \mathbf{x}_{[i]}, y_{[i]})$ for all $i \in \{1,\ldots,N\}$, where $\mathbf{w}_{[i]}$ represents the latest iterate at which ∇f_i was evaluated.

- *Stochastic variance reduced gradient* (SVRG) method: Stochastic gradient descent (SGD) can be improved with the *variance reduction* (VR) technique. The SVRG method [133] adopts a constant learning rate to train parameters:

$$\text{SVRG:}\quad \tilde{\mathbf{g}}_j \leftarrow \nabla f_{i_j}(\tilde{\mathbf{w}}_j) - \left(\nabla f_{i_j}(\mathbf{w}_k) - \frac{1}{N}\sum_{i=1}^{N}\nabla f_i(\mathbf{w}_k)\right). \tag{6.2.20}$$

- *Stochastic average gradient aggregation* (SAGA) method: This method [72] is inspired both from SAG and SVRG. The stochastic gradient vector in SAGA is given by

$$\text{SAGA:}\quad \mathbf{g}_k \leftarrow \nabla f_j(\mathbf{w}_k) - \nabla f_j(\mathbf{w}_{[j]}) + \frac{1}{N}\sum_{i=1}^{N}\nabla f_i(\mathbf{w}_{[i]}), \tag{6.2.21}$$

  where $j \in \{1, \ldots, N\}$ is chosen at random and the stochastic vectors are set by

$$\nabla f_j(\mathbf{w}_k) = \frac{\partial f(\mathbf{w}_k; \mathbf{x}_j, y_j)}{\partial \mathbf{w}_k}, \tag{6.2.22}$$

  and

$$\nabla f_j(\mathbf{w}_{[j]}) = \left.\frac{\partial f(\mathbf{w}; \mathbf{x}_j, y_j)}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_{[j]}}, \tag{6.2.23}$$

$$\nabla f_i(\mathbf{w}_{[i]}) = \left.\frac{\partial f(\mathbf{w}; \mathbf{x}_i, y_i)}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_{[i]}}, \tag{6.2.24}$$

  for all $i \in \{1, \ldots, N\}$, and an integer $j \in \{1, \ldots, N\}$ is chosen at random. $\mathbf{w}_{[i]}$ and $\mathbf{w}_{[j]}$ represent the latest iterates at which ∇f_i and ∇f_j were, respectively, evaluated.

A formal description of a few variants of SVRG is presented as Algorithm 6.2. SAGA method for minimizing an empirical risk E_n is shown in Algorithm 6.3. It has been shown [72] that

- Theoretical convergence rates for SAGA in the strongly convex case are better than those for SAG and SVRG,
- SAGA is applicable to non-strongly convex problems without modification.

Algorithm 6.2 SVRG method for minimizing an empirical risk R_n [31, 133]

```
1.  input: Sample pairs $(\mathbf{x}_j, y_j)$, $j = 1, \dots, N$, update frequency $m$ and learning rate $\alpha$.
2.  initialization: Choose an initial iterate $\mathbf{w}_1 \in \mathbb{R}^d$.
3.  for $k = 1, 2, \dots$ do
4.    Compute $\nabla_i(\mathbf{w}_k) = \frac{\partial f(\mathbf{w}_k; \mathbf{x}_i, y_i)}{\partial \mathbf{w}_k}$.
5.    Compute the batch gradient $R_n(\mathbf{w}_k) = \frac{1}{N}\sum_{i=1}^N \nabla f_i(\mathbf{w}_k)$.
6.    Initialize $\tilde{\mathbf{w}}_1 \leftarrow \mathbf{w}_k$.
7.    for $j = 1, \dots, m$ do
8.      Chose $i_j$ uniformly from $\{1, \dots, N\}$.
9.      Set $\tilde{\mathbf{g}}_j \leftarrow \nabla f_{i_j}(\tilde{\mathbf{w}}_j) - (\nabla f_{i_j}(\mathbf{w}_k) - R_n(\mathbf{w}_k))$.
10.     Set $\tilde{\mathbf{w}}_{j+1} \leftarrow \tilde{\mathbf{w}}_j - \alpha \tilde{\mathbf{g}}_j$.
11.   end for
12.   Option (a): Set $\mathbf{w}_{k+1} = \tilde{\mathbf{w}}_{m+1}$.
13.   Option (b): Set $\mathbf{w}_{k+1} = \frac{1}{m}\sum_{j=1}^m \tilde{\mathbf{w}}_{j+1}$.
14.   Option (c): Choose $j$ uniformly from $\{1, \dots, m\}$ and set $\mathbf{w}_{k+1} = \tilde{\mathbf{w}}_{j+1}$.
15.   exit: If $\mathbf{w}_{k+1}$ is converged.
16. end for
17. output: $\mathbf{w} = \mathbf{w}_{k+1}$.
```

Algorithm 6.3 SAGA method for minimizing an empirical risk R_n [31, 72]

```
1.  input: Sample pairs $(\mathbf{x}_j, y_j)$, $j = 1, \dots, N$, stepsize $\alpha > 0$.
2.  initialization: Choose an initial iterate $\mathbf{w}_1 \in \mathbb{R}^d$.
3.  for $i = 1, \dots, N$ do
4.    Compute $\nabla f_i(\mathbf{w}_1) = \frac{\partial f(\mathbf{w}_1; \mathbf{x}_i, y_i)}{\partial \mathbf{w}_1}$.
5.    Store $\nabla f_i(\mathbf{w}_{[i]}) \leftarrow \nabla f_i(\mathbf{w}_1)$.
6.  end for
7.  for $k = 1, 2, \dots$ do
8.    Choose $j$ uniformly in $\{1, \dots, N\}$.
9.    Compute $\nabla f_j(\mathbf{w}_k) = \frac{\partial f(\mathbf{w}_k; \mathbf{x}_j, y_j)}{\partial \mathbf{w}_k}$.
10.   Set $\mathbf{g}_k \leftarrow \nabla f_j(\mathbf{w}_k) - \nabla f_j(\mathbf{w}_{[j]}) + \frac{1}{n}\sum_{i=1}^n \nabla f_i(\mathbf{w}_{[i]})$.
11.   Store $\nabla f_j(\mathbf{w}_{[i]}) \leftarrow \nabla f_j(\mathbf{w}_k)$.
12.   Set $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \alpha \mathbf{g}_k$.
13.   exit: If $\mathbf{w}_{k+1}$ is converged.
14. end for
15. output: $\mathbf{w} = \mathbf{w}_{k+1}$.
```

6.2.3 Coordinate Descent Methods

Coordinate descent methods (CDMs) are among the first optimization schemes suggested for solving smooth unconstrained minimization problems and large-scale regression problems (see, e.g., [12, 25, 188]).

As the name suggests, the basic operation of coordinate descent methods takes steps along coordinate directions: the objective is minimized with respect to a single variable while all others are kept fixed, then other variables are updated similarly in an iterative manner.

The coordinate descent method for minimizing $f(\mathbf{w}) : \mathbb{R}^d \to \mathbb{R}$ is given by the iteration

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \alpha_k \nabla_{i_k} f(\mathbf{w}_k)\mathbf{e}_{i_k} \tag{6.2.25}$$

with

$$\nabla f_{i_k} f(\mathbf{w}_k) = \frac{\partial f(\mathbf{w}_k)}{\partial w_{i_k}}, \tag{6.2.26}$$

where w_{i_k} denotes the i_kth element of the parameter vector $\mathbf{w}_k = [w_1, \ldots, w_d]^T \in \mathbb{R}^d$ and $\mathbf{e}_{i_k}$ represents the i_kth *unit coordinate vector* (also called *natural basis vector*) for some $i_k \in \{1, \ldots, d\}$, i.e., $\mathbf{e}_{i_k}$ is a $d \times 1$ vector whose i_kth element is equal to one and others are equal to zero.

Example 6.1 Given a vector $\mathbf{w}_k = [w_{k1}, \ldots, w_{kd}]^T \in \mathbb{R}^d$. If the function $f(\mathbf{w}_k) = \frac{1}{2}\|\mathbf{w}_k\|_2^2 = \frac{1}{2}(w_{k1}^2 + \cdots + w_{id}^2)$, then

$$\nabla_{i_k} f(\mathbf{w}_k) = w_{i_k}, \quad i_k \in \{1, \ldots, d\},$$

and the ith element of $\mathbf{w}_{k+1}$ is given by

$$w_{k+1,i} = \begin{cases} w_{k,i} - \alpha_k w_i, & i = i_k; \\ \\ w_{k,i}, & \text{otherwise}; \end{cases}$$

for $i = 1, \ldots, d$; $i_k \in \{1, \ldots, k\}$.

That is to say, the solution estimates $\mathbf{w}_{k+1}$ and $\mathbf{w}_k$ differ only in their i_kth element as a result of a move in the i_kth coordinate from $\mathbf{w}_k$.

According to the choice of i_k, the coordinate descent methods can be divided into two categories.

1. *Cyclic coordinate descent:* Cyclic coordinate search through $\{1, \ldots, d\}$;
2. *Stochastic coordinate descent:* Random coordinate descent search [31]:
   - Cyclic random coordinate search through a random reordering of these indices (with the indices reordered after each set of d steps);
   - Select simply an index randomly with replacement in each iteration.

Stochastic coordinate descent algorithms are almost identical to cyclic coordinate descent algorithms except for selecting coordinates in a random manner.

Coordinate descent methods for solving $\min_{\mathbf{x}} f(\mathbf{x})$ consist of random choice step (R) and update step (U) at each iteration [166]:

R: Randomly choose an index $i_k \in \{1, \ldots, n\}$, read $\mathbf{x}$, and evaluate $\nabla_{i_k} f(\mathbf{x})$;

U: Update component i_k of the shared $\mathbf{x}_k$ as $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k - \alpha_k \nabla_{i_k} f(\mathbf{x}_k)\mathbf{e}_{i_k}$.

In the basic coordinate descent methods, a single coordinate is only updated at each iteration. In the following we consider how to update a random subset of coordinates at each iteration,

Consider the unconstrained minimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^N} f(\mathbf{x}), \quad \mathbf{x} = [x_1, \ldots, x_N]^T \in \mathbb{R}^N, \tag{6.2.27}$$

where the objective function $f(\mathbf{x})$ is convex and differentiable on $\mathbb{R}^N$. Under the assumption that decision variables are separable, to decompose the decision vector $\mathbf{x} \in \mathbb{R}^N$ into n non-overlapping blocks in which each block contains N_i decision variables:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}^{(1)} \\ \vdots \\ \mathbf{x}^{(n)} \end{bmatrix}, \quad \text{where } \mathbf{x}^{(i)} = [x_{i_1}, \ldots, x_{i_{N_i}}]^T \in \mathbb{R}^{N_i} \tag{6.2.28}$$

such that

$$\mathbb{R}^N = \mathbb{R}^{N_1} \times \cdots \times \mathbb{R}^{N_n}, \quad N = \sum_{i=1}^{n} N_i. \tag{6.2.29}$$

Define the corresponding partition of the unit matrix $\mathbf{I}_{N\times N}$ as

$$\mathbf{I} = [\mathbf{U}_1, \ldots, \mathbf{U}_n] \in \mathbb{R}^{N\times N}, \quad \mathbf{U}_i \in \mathbb{R}^{N\times N_i},\ i = 1, \ldots, n. \tag{6.2.30}$$

If letting $f_i(\mathbf{x})$ be differentiable convex functions such that $f_i(\mathbf{x})$ depends on blocks $\mathbf{x}^{(i)}$ for $i \in S_i = \{i_1, \ldots, i_{N_i}\}$ only, then the partial derivative of function $f(\mathbf{x})$ with respect to the block-vector $\mathbf{x}^{(i)}$ is defined as

$$\nabla_i f(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}^{(i)}} = \mathbf{U}_i^T \nabla f(\mathbf{x}) \in \mathbb{R}^{N_i}. \tag{6.2.31}$$

Proposition 6.1 (Block Decomposition [212]) *Any vector $\mathbf{x} \in \mathbb{R}^N$ can be written uniquely as*

$$\mathbf{x} = \sum_{i=1}^{n} \mathbf{U}_i \mathbf{x}^{(i)}, \tag{6.2.32}$$

where $\mathbf{x}^{(i)} \in \mathbb{R}^{N_i}$. Moreover, $\mathbf{x}^{(i)} = \mathbf{U}_i^T \mathbf{x}$.

Example 6.2 When $S_1 = \{1, 2\}$, $\mathbf{U}_1 = [\mathbf{e}_1, \mathbf{e}_2]$, where $\mathbf{e}_i$ is the ith unit coordinate vector. Then we have $\mathbf{x}^{(1)} = [x_1, x_2]^T$ and $f_i(\mathbf{x}) = \mathbf{U}_{\mathbf{i}}^T f(\mathbf{x}) = [f(x_1), f(x_2)]^T$, which gives

$$\nabla_1 f(\mathbf{x}) = \nabla f_1(\mathbf{x}) = \nabla[f(x_1), f(x_2)]^T = \begin{bmatrix} f'(x_1) \\ f'(x_2) \end{bmatrix},$$

or

$$\nabla_1 f(\mathbf{x}) = [\mathbf{e}_1, \mathbf{e}_2]^T \nabla f(\mathbf{x}) = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} f'(x_1) \\ f'(x_2) \\ \vdots \\ f'(x_N) \end{bmatrix} = \begin{bmatrix} f'(x_1) \\ f'(x_2) \end{bmatrix},$$

where $f'(x_j) = \frac{\partial f(\mathbf{x})}{\partial x_j}$, $j = 1, 2$.

The *randomized coordinate descent method* (RCDM) [188] consists of the following two basic steps:

- Choose at random $i_k \in S_i = \{i_1, \ldots, i_{N_i}\}$;
- Update $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{U}_{i_k}^T \nabla_{i_k} f(\mathbf{x}_k)$.

To accelerate the randomized coordinate descent method, Nesterov [188] in 2012 proposed a computationally inefficient but theoretically interesting accelerated variant of RCDM, called *accelerated coordinate descent method (ACDM)*, for convex minimization without constraints.

Consider the regularized optimization problem with separable decision variables:

$$\min_{\mathbf{x} \in \mathbb{R}^N} \{F(\mathbf{x}) = f(\mathbf{x}) + h(\mathbf{x})\}$$
$$\text{subject to} \quad \mathbf{x} = \sum_{i=1}^{n} \mathbf{U}_i \mathbf{x}^{(i)} \in \mathbb{R}^{N_1} \times \cdots \times \mathbb{R}^{N_n} = \mathbb{R}^N, \tag{6.2.33}$$

where $f(\mathbf{x})$ is a smooth convex objective function that is separable in the blocks $\mathbf{x}^{(i)}$:

$$f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x}), \quad \text{where} \quad f_i(\mathbf{x}) = \mathbf{U}_i^T f(\mathbf{x}), \tag{6.2.34}$$

and $h(\mathbf{x})$ is a (possibly nonsmooth) convex regularizer that is also separable in the blocks $\mathbf{x}^{(i)}$:

$$h(\mathbf{x}) = \sum_{i=1}^{n} h_i(\mathbf{x}), \quad \text{where} \quad h_i(\mathbf{x}) = \mathbf{U}_i^T h(\mathbf{x}). \tag{6.2.35}$$

Fercoq and Richtárk [91] in 2015 proposed the first randomized block coordinate descent method which is simultaneously accelerated, parallel, and proxima, called simply APPROX, for solving the optimization problem (6.2.33).

Let $\hat{S}$ be a random subset of $\mathcal{N} = \{1, \ldots, N\}$. Algorithm 6.4 shows the *APPROX coordinate descent method* in a form facilitating efficient implementation. APPROX is a remarkably versatile method.

Algorithm 6.4 APPROX coordinate descent method [91]

```
1.  input:
2.  initialization: Pick $\tilde{\mathbf{z}}_0 \in \mathbb{R}^N$ and set $\theta_0 = \frac{\tau}{n}$, $\mathbf{u}_0 = \mathbf{0}$.
3.  for $k = 0, 1, \ldots$ do
4.  Generate a random set of blocks $S_k \sim \hat{S}$.
5.    $\mathbf{u}_{k+1} \leftarrow \mathbf{u}_k,\ \tilde{\mathbf{z}}_{k+1} \leftarrow \tilde{\mathbf{z}}_k$.
6.    for $i \in S_k$ do
7.      $\mathbf{t}_k^{(i)} = \arg\max_{\mathbf{t}\in\mathbb{R}^{N_i}} \left\{ \langle \nabla_i f(\theta^2\mathbf{u}_k + \tilde{\mathbf{z}}_k), \mathbf{t}\rangle + \frac{n\theta_k v_i}{2\tau}\|\mathbf{t}\|_{(i)}^2 + h_i(\tilde{\mathbf{z}}_k^{(i)}) \right\}$.
8.      $\tilde{\mathbf{z}}_{k+1}^{(i)} \leftarrow \tilde{\mathbf{z}}_k^{(i)} + \mathbf{t}_k^{(i)}$.
9.      $\mathbf{u}_{k+1}^{(i)} \leftarrow \mathbf{u}_k^{(i)} - \frac{1-\frac{n}{\tau}\theta_k}{\theta_k^2}\mathbf{t}_k^{(i)}$.
10.   end for
11.   $\theta_{k+1} = \frac{\sqrt{\theta_k^4+4\theta_k^2}-\theta_k^2}{2}$.
12. end for
13. output: $\mathbf{x} = \theta_k^2\mathbf{u}_{k+1} + \tilde{\mathbf{z}}_{k+1}$.
```

6.2.4 Benchmark Functions for Single-Objective Optimization

The test of reliability, efficiency, and validation of an optimization algorithm is frequently carried out by using a chosen set of common *benchmark* or *test functions* [129]. There have been many benchmark or test functions reported in the literature. Ideally, benchmark functions should have diverse properties when testing new algorithms in an unbiased way. On the other hand, there is a serious phenomenon called "curse of dimensionality" in many optimization methods [21]. This implies that the optimization performance deteriorates quickly as the dimensionality of the search space increases. The reasons for this phenomenon appear to be two fold [240]:

- The solution space of a problem often increases exponentially with the problem dimension [21] and more efficient search strategies are required to explore all promising regions within a given time budget.
- The characteristics of a problem may change with the scale. Because an increase in scale may result in worsening of the features of an optimization problem, a previously successful search strategy may no longer be capable of finding the optimal solution.

To solve large-scale problems, it is necessary to provide a suite of benchmark functions for large-scale numerical optimization.

Definition 6.1 (Separable Function) Given $\mathbf{x} = [x_1, \ldots, x_n]^T$. A function $f(\mathbf{x})$ is a *separable function* if and only if

$$\arg\min_{(x_1,\ldots,x_n)} f(x_1, \ldots, x_n) = \left(\arg\min_{x_1} f(x_1, \ldots), \ldots, \arg\min_{x_n} f(\ldots, x_n) \right), \tag{6.2.36}$$

where $f(\ldots, x_i, \ldots)$ denotes that its decision variable is just x_i and other variables are constants with respect to x_i.

That is to say, a function of n decision variables is separable if it can be rewritten as a sum of n functions of just one variable, i.e., $f(x_1, \ldots, x_n) = f(x_1) + \cdots + f(x_n)$. If a function $f(\mathbf{x})$ is separable, its parameters x_i are said to be independent.

A separable function can be easily optimized, as it can be decomposed into a number of subproblems, each of which involving only one decision variable while treating all others as constants.

Definition 6.2 (Nonseparable Function) A *nonseparable function* $f(\mathbf{x})$ is called m-nonseparable function if at most m (where $m < n$) of its decision variables x_i are not independent. A nonseparable function $f(\mathbf{x})$ is known as a *fully nonseparable function* if any two of its parameters x_i are not independent.

A nonseparable function has at least partly coupled or dependent decision variables.

The definitions of separability provide us a measure of the difficulty of different optimization problems based on which a spectrum of benchmark problems can be designed. In general, separable problems are considered to be easiest, while the fully nonseparable ones usually are most difficult.

A benchmark problem for large-scale optimization is to provide test functions to researchers or users for testing the optimization algorithms' performance when applied to test problems. These test functions are as same as efficient in practical scenarios.

Jamil and Yang [129] presented a collection of 175 unconstrained optimization test problems which can be used to validate the performance of optimization algorithms. Some benchmark functions are described by classification as follows.

1. *Continuous, Differentiable, Separable, Multimodal Functions*

   - Giunta Function [179]

$$\begin{aligned} f(\mathbf{x}) = 0.6 + \sum_{i=1}^{2} \Bigg[&\sin\left(\frac{16}{15}x_i - 1\right) + \sin^2\left(\frac{16}{15}x_i - 1\right) \\ &+ \frac{1}{50}\sin\left(4\left(\frac{16}{15}x_i - 1\right)\right)\Bigg] \end{aligned} \tag{6.2.37}$$

subject to $-1 \leq x_i \leq 1$. The global minimum is $f(\mathbf{x}^*) = 0.060447$ located at $\mathbf{x}^* = (0.45834282,\ 0.45834282)$.

- Parsopoulos Function

$$f(\mathbf{x}) = \cos(x_1)^2 + \sin(x_2)^2 \tag{6.2.38}$$

subject to $-5 \leq x_i \leq 5$, where $(x_1, x_2) \in \mathbb{R}^2$. This function has an infinite number of global minima in $\mathbb{R}^2$, at points $(\kappa \frac{2\pi}{2}, \lambda\pi)$, where $\kappa = \pm 1, \pm 3, \ldots$ and $\lambda = 0, \pm 1, \pm 2, \ldots$. In the given domain problem, the function has 12 global minima all equal to zero.

2. *Continuous, Differentiable, Nonseparable, Multimodal Functions*

- El-Attar-Vidyasagar-Dutta Function [85]

$$f(\mathbf{x}) = \left(x_1^2 + x_2 - 10\right)^2 + \left(x_1 + x_2^2 - 7\right)^2 + \left(x_1^2 + x_2^3 - 1\right)^2 \tag{6.2.39}$$

subject to $-500 \leq x_i \leq 500$. The global minimum is $f(\mathbf{x}^*) = 0.470427$ located at $\mathbf{x}^* = (2.842503,\ 1.920175)$.

- Egg Holder Function

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left[-(x_{i+1} + 47) \sin\sqrt{|x_{i+1} + x_i/2 + 47|} - x_i \sin\sqrt{|x_i - (x_{i+1} + 47)|} \right]$$

subject to $-512 \leq x_i \leq 512$. The global minimum for $n = 2$ is $f(\mathbf{x}^*) \approx -959.64$ located at $\mathbf{x}^* = (512,\ 404.2319)$.

- Exponential Function [209]

$$f(\mathbf{x}) = -\exp\left(-0.5 \sum_{i=1}^{D} x_i^2\right), \tag{6.2.40}$$

subject to $-1 \leq x_i \leq 1$. The global minima is $f(\mathbf{x}^*) = 1$ located at $\mathbf{x}^* = (0, \ldots, 0)$.

- Langerman-5 Function [23]

$$f(\mathbf{x}) = -\sum_{i=1}^{m} c_i \exp\left(-\frac{1}{\pi} \sum_{j=1}^{D} (x_j - a_{ij})^2\right) \cos\left(\pi \sum_{j=1}^{D} (x_j - a_{ij})^2\right) \tag{6.2.41}$$

subject to $0 \leq x_j \leq 10$, where $j \in [1, D]$ and $m = 5$. It has a global minimum value of $f(\mathbf{x}^*) = -1.4$. The matrix $\mathbf{A} = [a_{ij}]$ and column vector $\mathbf{c} = [c_i]$ are given as

$$\mathbf{A} = \begin{bmatrix} 9.681 & 0.667 & 4.783 & 9.095 & 3.517 & 9.325 & 6.544 & 0.211 & 5.122 & 2.020 \\ 9.400 & 2.041 & 3.788 & 7.931 & 2.882 & 2.672 & 3.568 & 1.284 & 7.033 & 7.374 \\ 8.025 & 9.152 & 5.114 & 7.621 & 4.564 & 4.711 & 2.996 & 6.126 & 0.734 & 4.982 \\ 2.196 & 0.415 & 5.649 & 6.979 & 9.510 & 9.166 & 6.304 & 6.054 & 9.377 & 1.426 \\ 8.074 & 8.777 & 3.467 & 1.863 & 6.708 & 6.349 & 4.534 & 0.276 & 7.633 & 1.567 \end{bmatrix}$$

and $\mathbf{c} = [0.806, 0.517, 1.500, 0.908, 0.965]^T$.

- Mishra Function 1 [180]

$$f(\mathbf{x}) = \left(1 + D - \sum_{i=1}^{n-1} 0.5(x_i + x_{i+1})\right)^{n - \sum_{i=1}^{n-1} 0.5(x_i + x_{i+1})} \tag{6.2.42}$$

subject to $0 \le x_i \le 1$. The global minimum is $f(\mathbf{x}^*) = 2$.

- Mishra Function 2 [180]

$$f(\mathbf{x}) = -\ln\left[\sin^2\left((\cos(x_1) + \cos(x_2))^2\right) + \cos^2\left((\sin(x_1) + \sin(x_2))^2\right) + x_1\right]^2 + 0.01((x_1 - 1)^2 + (x_2 - 1)^2). \tag{6.2.43}$$

The global minimum $f(\mathbf{x}^*)=-2.28395$ is located at $\mathbf{x}^*=(2.88631,\ 1.82326)$.

3. *Continuous, Non-Differentiable, Separable, Multimodal Functions*

- Price Function [208]

$$f(\mathbf{x}) = (|x_1| - 5)^2 + (|x_2| - 5)^2 \tag{6.2.44}$$

subject to $-500 \le x_i \le 500$. The global minima $f(\mathbf{x}^*) = 0$ are located at $\mathbf{x}^* = (-5, -5), (-5, 5), (5, -5), (5, 5)$.

- Schwefel Function [223]

$$f(\mathbf{x}) = -\sum_{i=1}^{n} |x_i| \tag{6.2.45}$$

subject to $-100 \le x_i \le 100$. The global minimum $f(\mathbf{x}^*) = 0$ is located at $\mathbf{x}^* = (0, \ldots, 0)$.

4. *Continuous, Non-Differentiable, Nonseparable, Multimodal Functions*

- Bartels Conn Function

$$f(\mathbf{x}) = \left|x_1^2 + x_2^2 + x_1 x_2\right| + |\sin(x_1)| + |\cos(x_2)| \tag{6.2.46}$$

subject to $-500 \leq x_i \leq 500$. The global minimum $f(\mathbf{x}^*) = 1$ is located at $\mathbf{x}^* = (0, 0)$.

- Bukin Function

$$f(\mathbf{x}) = 100\sqrt{|x_2 - 0.01x_1^2|} + 0.01|x_1 + 10| \tag{6.2.47}$$

subject to $-15 \leq x_1 \leq -5$ and $-13 \leq x_2 \leq -3$. The global minimum $F(\mathbf{x}^*) = 0$ is located at $\mathbf{x}^* = (-10, 1)$.

5. *Continuous, Differentiable, Partially Separable, Unimodal Functions*

- Chung–Reynolds Function [54]

$$f(\mathbf{x}) = \left(\sum_{i=1}^{D} x_i^2\right)^2 \tag{6.2.48}$$

subject to $-100 \leq x_i \leq 100$. The global minimum $f(\mathbf{x}^*) = 0$ is located at $\mathbf{x}^* = (0, \ldots, 0)$.

- Schwefel Function [223]

$$f(\mathbf{x}) = \left(\sum_{i=1}^{D} x_i^2\right)^\alpha \tag{6.2.49}$$

subject to $-100 \leq x_i \leq 100$, where $\alpha \geq 0$. The global minimum $f(\mathbf{x}^*) = 0$ is located at $\mathbf{x}^* = (0, \ldots, 0)$.

6. *Discontinuous, Non-Differentiable Functions*

- Corana Function [61]

$$f(\mathbf{x}) = \begin{cases} 0.15\left(z_i - 0.05\,\mathrm{sign}(z_i)^2\right) d_i, & \text{if } |v_i| < A, \\ \\ d_i x_i^2, & \text{otherwise,} \end{cases} \tag{6.2.50}$$

where

$$\begin{aligned} v_i &= |x_i - z_i|,\ A = 0.05, \\ z_i &= 0.2\left\lfloor\left|\frac{x_i}{0.2}\right| + 0.49999\right\rfloor\ \mathrm{sign}(x_i) \\ d_i &= (1, 1000, 10, 100) \end{aligned}$$

subject to $-500 \leq x_i \leq 500$. The global minimum $f(\mathbf{x}^*) = 0$ is located at $\mathbf{x}^* = (0, 0, 0, 0)$.

- Cosine Mixture Function [4]

$$f(\mathbf{x}) = -0.1\sum_{i=1}^{n}\cos(5\pi x_i) - \sum_{i=1}^{n} x_i^2 \tag{6.2.51}$$

subject to $-1 \leq x_i \leq 1$. The global minimum is located at $\mathbf{x}^* = (0, 0)$ or $\mathbf{x}^* = (0, 0, 0, 0)$, $f(\mathbf{x}^*) = 0.2$ or 0.4 for $n = 2$ and 4, respectively.

In this section we have focused on the single-objective optimization problems in machine learning. In a lot of applications in artificial intelligence, we will be faced with multi-objective optimization problems. Because these problems are closely related to evolutionary computation, we will deal with multi-objective optimization theory and methods in Chap. 9.

6.3 Majorization-Minimization Algorithms

In the era of big data, a fast development in data acquisition techniques and a growth of computing power, from an optimization perspective, can result in large-scale problems due to the tremendous amount of data and variables, which cause challenges to traditional algorithms [87]. By Hunter and Lange [125], the general principle behind *majorization-minimization (MM)* algorithms was first enunciated by the numerical analysts in 1970s [197] in the context of line search methods. MM algorithms are a set of analytic procedures for tackling difficult optimization problems. The basic idea of MM algorithms is to modify their objective functions so that solution spaces of the modified objective functions are easier to explore.

A successful MM algorithm substitutes a simple optimization problem for a difficult optimization problem. By [125], simplicity can be attained by

- avoiding large matrix inversions,
- linearizing an optimization problem,
- separating the parameters of an optimization problem,
- dealing with equality and inequality constraints gracefully, or
- turning a non-differentiable problem into a smooth problem.

MM has a long history that dates back to the 1970s [197], and is closely related to the famous expectation-maximization (EM) algorithm [278] intensively used in computational statistics.

6.3.1 MM Algorithm Framework

Consider the minimization problem

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}\in\Theta} \ g(\boldsymbol{\theta}) \tag{6.3.1}$$

for some difficult to manipulate objective function $g(\boldsymbol{\theta})$. For example, $g(\boldsymbol{\theta})$ with $\boldsymbol{\theta} \in \Theta$ is non-differential and/or non-convex, where Θ is a subset of some Euclidean space.

Let $\boldsymbol{\theta}^{(m)}$ represent a fixed value of the parameter $\boldsymbol{\theta}$ (where m is the mth iteration), and let $g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)})$ denote a real-valued function of $\boldsymbol{\theta}$ whose form depends on $\boldsymbol{\theta}^{(m)}$.

Instead of operating on $g(\boldsymbol{\theta})$, we can consider an easier *surrogate function* $h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)})$ at some point $\boldsymbol{\theta}^{(m)}$ instead.

Definition 6.3 (Majorizer [125, 191]) The surrogate function $h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)})$ is said to be a *majorizer* of objective $g(\boldsymbol{\theta})$ provided

$$h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)}) \geq g(\boldsymbol{\theta}) \quad \text{for all } \boldsymbol{\theta}, \tag{6.3.2}$$

$$h(\boldsymbol{\theta}^{(m)}|\boldsymbol{\theta}^{(m)}) = g(\boldsymbol{\theta}^{(m)}). \tag{6.3.3}$$

To find the majorizer $h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)})$ of objective $g(\boldsymbol{\theta})$ constitutes the first step of MM algorithms, called *majorization step*.

The function $h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)})$ satisfying the conditions (6.3.2) and (6.3.3) is said to majorize a real-valued function $g(\boldsymbol{\theta})$ at the point $\boldsymbol{\theta}^{(m)}$. In other words, the surface $\boldsymbol{\theta} \to h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)})$ lies above the surface $g(\boldsymbol{\theta})$ and is tangent to it at the point $\boldsymbol{\theta} = \boldsymbol{\theta}^{(m)}$. In this sense, $h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)})$ is called the majorization function. Moreover, if $\boldsymbol{\theta}^{(m+1)}$ is a minimizer of $h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)})$, then both (6.3.2) and (6.3.3) further imply [288] that

$$g(\boldsymbol{\theta}^{(m)}) = h(\boldsymbol{\theta}^{(m)}|\boldsymbol{\theta}^{(m)}) \geq h(\boldsymbol{\theta}^{(m+1)}|\boldsymbol{\theta}^{(m)}) \geq g(\boldsymbol{\theta}^{(m+1)}). \tag{6.3.4}$$

This is an important property of the MM algorithm, which means that the sequence $\{g(\boldsymbol{\theta}^{(m)})\}$, $m = \{1, 2, \ldots\}$ is non-increasing, so the iteration procedure $\boldsymbol{\theta}^{(m)}$ pushes $g(\boldsymbol{\theta})$ toward its minimum.

The function $h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)})$ is said to minorize $g(\boldsymbol{\theta})$ at $\boldsymbol{\theta}^{(m)}$ if $-h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)})$ majorizes $-g(\boldsymbol{\theta})$ at $\boldsymbol{\theta}^{(m)}$.

The second step, known as *minimization step*, is to minimize the surrogate function, namely update $\boldsymbol{\theta}$ as

$$\boldsymbol{\theta}^{(m+1)} = \arg\min_{\boldsymbol{\theta}} \ h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)}). \tag{6.3.5}$$

The descent property (6.3.2) lends an MM algorithm remarkable numerical stability. When minimizing a difficult loss function $g(\boldsymbol{\theta})$, we relax this loss function to a surrogate function $h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)})$ that is easily minimized.

Example 6.3 Given a matrix pencil $(\mathbf{A}, \mathbf{B})$, consider its generalized eigenvalue (GEV) problem $\mathbf{Ax} = \lambda\mathbf{Bx}$. From $\lambda = \mathbf{x}^T\mathbf{Ax}/(\mathbf{x}^T\mathbf{Bx})$ it is known that the variational formulation for the GEV problem in $\mathbf{Ax} = \lambda\mathbf{Bx}$ is given by

$$\lambda_{\max}(\mathbf{A}, \mathbf{B}) = \max_{\mathbf{x}} \ \{\mathbf{x}^T\mathbf{Ax}\} \quad \text{subject to} \quad \mathbf{x}^T\mathbf{Bx} = 1, \tag{6.3.6}$$

where $\lambda_{\max}(\mathbf{A}, \mathbf{B})$ is the maximum generalized eigenvalue associated with the matrix pencil $(\mathbf{A}, \mathbf{B})$. Then, the sparse GEV problem can be written as

$$\max_{\mathbf{x}} \ \{\mathbf{x}^T\mathbf{Ax}^T\} \quad \text{subject to} \quad \mathbf{x}^T\mathbf{Bx} = 1, \ \|\mathbf{x}\|_0 \le k. \tag{6.3.7}$$

Consider the regularized (penalized) version of (6.3.7) given by [237]:

$$\max_{\mathbf{x}} \ \left\{\mathbf{x}^T\mathbf{Ax} - \rho\|\mathbf{x}\|_0\right\} \quad \text{subject to} \quad \mathbf{x}^T\mathbf{Bx} \le 1, \tag{6.3.8}$$

where $\rho > 0$ is the regularization (penalization) parameter. The constrained equality condition $\mathbf{x}^T\mathbf{Bx} = 1$ in (6.3.7) has been relaxed to the inequality constraint $\mathbf{x}^T\mathbf{Bx} \le 1$. To relax further the non-convex ℓ_0-norm $\|\mathbf{x}\|_0$, consider using

$$\|\mathbf{x}\|_0 = \sum_{i=1}^{n} \mathbb{1}\{|\mathbf{x}_i| \ne 0\} = \lim_{\epsilon\to 0} \sum_{i=1}^{n} \frac{\log(1 + |x_i|/\epsilon)}{\log(1 + 1/\epsilon)} \tag{6.3.9}$$

in (6.3.8) to rewrite as the equivalent form

$$\max_{\mathbf{x}} \ \left\{\mathbf{x}^T\mathbf{Ax} - \rho \lim_{\epsilon\to 0} \sum_{i=1}^{n} \frac{\log(1 + |x_i|/\epsilon)}{\log(1 + 1/\epsilon)}\right\} \quad \text{subject to} \quad \mathbf{x}^T\mathbf{Bx} \le 1. \tag{6.3.10}$$

The above program is approximated by the following approximate sparse GEV program by neglecting the limit in (6.3.10) and choosing $\epsilon > 0$:

$$\max_{\mathbf{x}} \ \left\{\mathbf{x}^T\mathbf{Ax} - \rho \sum_{i=1}^{n} \frac{\log(1 + |x_i|/\epsilon)}{\log(1 + 1/\epsilon)}\right\} \quad \text{subject to} \quad \mathbf{x}^T\mathbf{Bx} \le 1, \tag{6.3.11}$$

which is finally equivalent to [237]

$$\max_{\mathbf{x}} \ \left\{\mathbf{x}^T\mathbf{Ax} - \rho_\epsilon \sum_{i=1}^{n} \log(1 + |x_i|/\epsilon)\right\} \quad \text{subject to} \quad \mathbf{x}^T\mathbf{Bx} \le 1, \tag{6.3.12}$$

where $\rho_\epsilon = \rho/\log(1 + 1/\epsilon)$. Hence, instead of a non-convex sparse optimization problem (6.3.7), we can manipulate an easier surrogate convex optimization problem (6.3.12) instead by using the MM algorithm.

In addition to the typical examples above, there are a lot of applications of MM algorithms to machine learning, statistical estimation, and signal processing problems, see [191] for a list of 26 applications, which includes fully visible Boltzmann machine estimation, linear mixed model estimation, Markov random field estimation, matrix completion and imputation, quantile regression estimation, SVM estimation, and so on.

As pointed out by Nguyen [191], the MM algorithm framework is a popular tool for deriving useful algorithms for problems in machine learning, statistical estimation, and signal processing.

6.3.2 *Examples of Majorization-Minimization Algorithms*

Let $\boldsymbol{\theta}^{(0)}$ be some initial value and $\boldsymbol{\theta}^{(m)}$ be a sequence of iterates (in m) for the minimization problem (6.3.1). Definition 6.3 suggests the following scheme, referred to as an MM algorithm.

Definition 6.4 (Majorization-Minimization (MM) Algorithm [191]) Let $\boldsymbol{\theta}^{(0)}$ be some initial value and $\boldsymbol{\theta}^{(m)}$ be the mth iterate. Then, $\boldsymbol{\theta}^{(m+1)}$ is said to be the $(m+1)$th iterate of an MM algorithm if it satisfies

$$\boldsymbol{\theta}^{(m+1)} = \arg\min_{\boldsymbol{\theta}\in\Theta} h\left(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)}\right). \tag{6.3.13}$$

From Definitions 6.3 and 6.4 it can be deduced that all MM algorithms have the monotonicity property. That is, if $\boldsymbol{\theta}^{(m)}$ is a sequence of MM algorithm iterates, the objective sequence $g(\boldsymbol{\theta})$ is monotonically decreasing in m.

Proposition 6.2 (Monotonicity Decreasing [125, 191]) *If $h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)})$ is a majorizer of the objective $g(\boldsymbol{\theta})$ and $\boldsymbol{\theta}^{(m)}$ is a sequence of MM algorithm iterates, then MM algorithms have the following monotonicity decreasing:*

$$g(\boldsymbol{\theta}^{(m+1)}) \leq h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m+1)}) \leq h(\boldsymbol{\theta}^{(m)}|\boldsymbol{\theta}^{(m)}) \leq g(\boldsymbol{\theta}^{(m)}). \tag{6.3.14}$$

Remarks It is notable that an algorithm need not be an MM algorithm in the strict sense of Definition 6.4 in order for (6.3.14) to hold. In fact, any algorithm with the $(m+1)$th iterate satisfying

$$\boldsymbol{\theta}^{(m+1)} \in \left\{\boldsymbol{\theta} \in \Theta : h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)}) \leq h(\boldsymbol{\theta}^{(m)}|\boldsymbol{\theta}^{(m)})\right\} \tag{6.3.15}$$

will generate a monotonically decreasing sequence of objective evaluates. Such an algorithm can be thought of as a generalized MM algorithm.

Proposition 6.3 (Stationary Point [191]) *Starting from some initial value $\boldsymbol{\theta}^{(0)}$, if $\boldsymbol{\theta}^{(\infty)}$ is the limit point of an MM algorithm sequence of iterates $\boldsymbol{\theta}^{(m)}$ (i.e., satisfying Definition 6.4), then $\boldsymbol{\theta}^{(\infty)}$ is a stationary point of the minimization problem (6.3.1).*

Remarks Proposition 6.3 only guarantees the convergence of MM algorithm iterates to a stationary point and not a global, or even a local minimum. As such, for problems over difficult objective functions, multiple or good initial values are required in order to ensure that the obtained solution is of a high quality. Furthermore, Proposition 6.3 only guarantees convergence to a stationary point of (6.3.1) if a limit point exists for the chosen starting value. If a limit point does not exist, then the MM algorithm objective sequence may diverge.

The following result is useful when constructing the majorization function.

Lemma 6.1 ([236]) *Let $\mathbf{L}$ be an Hermitian matrix and $\mathbf{M}$ be another Hermitian matrix such that $\mathbf{M} \geq \mathbf{L}$ ($M_{ij} \geq L_{ij}$). Then for any point $\mathbf{x}_0 \in \mathbb{C}^n$, the quadratic function $\boldsymbol{\theta}^H\mathbf{L}\boldsymbol{\theta}$ is majorized by $\boldsymbol{\theta}^H\mathbf{M}\boldsymbol{\theta} + 2\,Re\big(\boldsymbol{\theta}^H(\mathbf{L}-\mathbf{M})\boldsymbol{\theta}_0\big) + \boldsymbol{\theta}_0^H(\mathbf{M}-\mathbf{L})\boldsymbol{\theta}_0$ at $\boldsymbol{\theta}_0$.*

To minimize over $g(\boldsymbol{\theta})$, the main steps of the majorization-minimization scheme are as follows [236]:

1. Find a feasible point $\boldsymbol{\theta}^{(0)}$ and set $m = 0$.
2. Construct a majorization function $h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)})$ of $g(\boldsymbol{\theta})$ at the point $\boldsymbol{\theta}^{(m)}$.
3. Solve $\boldsymbol{\theta}^{(m+1)} = \arg\min_{\boldsymbol{\theta}\in\Theta} h(\boldsymbol{\theta}|\boldsymbol{\theta}^{(m)})$.
4. If some convergence criterion is met then exit; otherwise, set $k = k + 1$ and go to Step 2.

Consider the "weighted" ℓ_0 minimization problem [43]:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{W}\mathbf{x}\|_0 \quad \text{subject to} \quad \mathbf{y} = \boldsymbol{\Phi}\mathbf{x} \tag{6.3.16}$$

and the "weighted" ℓ_1 minimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \left\{\sum_{i=1}^{p} w_i|x_i|\right\} \quad \text{subject to} \quad \mathbf{y} = \boldsymbol{\Phi}\mathbf{x}. \tag{6.3.17}$$

The weighted ℓ_1 minimization can be viewed as a relaxation of a weighted ℓ_0 minimization problem. To establish the connection between the weighted ℓ_1 minimization and the weighted ℓ_0 minimization, we consider the surrogate function

$$\frac{|x_i|}{|x_i^{(m)}| + \epsilon} \begin{cases} \approx 1, \; x_i^{(m)} = x_i; \\ = 0, \; x_i^{(m)} = x_i = 0. \end{cases} \tag{6.3.18}$$

Here ϵ is a small positive value. This shows that if updating

$$\mathbf{x}^{(m+1)} = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \left\{ \sum_{i=1}^{n} \frac{|x_i|}{x_i^{(m)} + \epsilon} \right\}, \tag{6.3.19}$$

then $\sum_{i=1}^{n} \frac{|x_i|}{x_i^{(m)}+\epsilon}$ gives approximately $\|\mathbf{x}\|_0$ when $x_i^{(m)} \to x_i$ for all $i = 1, \ldots, n$ or $\mathbf{x}^{(m)} \to \mathbf{x}$.

Interestingly, (6.3.19) can be viewed as a form of the weighted ℓ_1 minimization (6.3.17), where $w_i^{(m+1)} = \frac{1}{|x_i^{(m)}|+\epsilon}$. Therefore, the MM algorithm for solving the weighted ℓ_1 minimization problem (6.3.17) consists of the following iterative steps [43]:

1. Set the iteration count m to zero and $w_i^{(0)} = 1,\ i = 1, \ldots, n$.
2. Solve the weighted ℓ_1 minimization problem

$$\mathbf{x}^{(m)} = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \left\{ \sum_{i=1}^{n} w_i |x_i| \right\} \quad \text{subject to } \mathbf{y} = \mathbf{\Phi x}. \tag{6.3.20}$$

3. Update the weights: for each $i = 1, \ldots, n$,

$$w_i^{(m+1)} = \frac{1}{|x_i^{(m)}| + \epsilon}. \tag{6.3.21}$$

4. Terminate on convergence or when m attains a specified maximum number of iterations $m_{\max}$. Otherwise, increment m by 1 and go to Step 2.

For a two-dimensional array $(x_{i,j})$, $1 \le i, j \le n$, let $(D\mathbf{x})_{i,j}$ be the two-dimensional vector of forward differences $(D\mathbf{x})_{i,j} = [x_{i+1,j} - x_{i,j}, x_{i,j+1} - x_{i,j}]$. The total-variation (TV) norm of $D\mathbf{x}_{ij}$ is defined as

$$\|\mathbf{x}\|_{\mathrm{TV}} = \sum_{1\le i,j\le n-1} \|D\mathbf{x}_{i,j}\|. \tag{6.3.22}$$

Because many natural images have a sparse or nearly sparse gradient, it is interesting to search for the reconstruction with minimal weighted TV norm, i.e.,

$$\min \left\{ \sum_{1\le i,j\le n-1} w_{i,j}^{(m)} \|(D\mathbf{x})_{i,j}\| \right\}, \quad \text{subject to } \mathbf{y} = \mathbf{\Phi x}, \tag{6.3.23}$$

where $w_{i,j}^{(m)} = \frac{1}{\|(D\mathbf{x})_{i,j}\|+\epsilon}$ by mimicking the derivation of (6.3.21).

The MM algorithm for minimizing a sequence of weighted TV norms is as follows [43]:

1. Set $m = 0$ and $w_{i,j}^{(0)}$, $1 \leq i, j \leq n-1$.
2. Solve the weighted TV minimization problem

$$\mathbf{x}^{(m)} = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \left\{ \sum_{1\leq i,j\leq n-1} w_{i,j}^{(m)} \|(D\mathbf{x})_{i,j}\| \right\}, \quad \text{subject to } \mathbf{y} = \mathbf{\Phi}\mathbf{x} \tag{6.3.24}$$

3. Update the weights

$$w_{i,j}^{(m)} = \frac{1}{\|(D\mathbf{x})_{i,j}\| + \epsilon} \tag{6.3.25}$$

   for each (i, j), $1 \leq i, j \leq n-1$.
4. Terminate on convergence or when m attains a specified maximum number of iterations $m_{\max}$. Otherwise, increment m by 1 and go to Step 2.

6.4 Boosting and Probably Approximately Correct Learning

Valiant and Kearns [140, 254] put forward the concepts of weak learning and strong learning. A learning algorithm with error rate less than $1/2$ (i.e., its accuracy is only slightly higher than random guess) is called a *weak learning algorithm*, and the learning algorithm whose accuracy is very high and can be completed in polynomial time is called a *strong learning algorithm*. At the same time, Valiant and Kearns proposed the equivalence problem of weak learning algorithm and strong learning algorithm in probably approximately correct (PAC) learning model, that is, if any given weak learning algorithm is only slightly better than random guessing, can it be upgraded to a strong learning algorithm? In 1990, Schapire [221] first constructed a polynomial-level algorithm to prove that a weak learning algorithm which is slightly better than random guessing can be upgraded to a strong learning algorithm instead of having to find a strong learning algorithm that is too hard to get. This is the original boosting algorithm. In 1995, Freund and Schapire [96] improved Boosting algorithm and proposed AdaBoost (Adaptive Boosting) algorithm.

"*Boosting*" is a way of combining the performance of many "weak" classifiers to produce a powerful "committee." Boosting was proposed in the computational learning theory literature [96, 97, 221] and has since received much attention [100].

6.4.1 Boosting for Weak Learners

Boosting is a framework algorithm for obtaining a sample subset by an operation on the sample set. It then trains the weak classification algorithm on the sample subset to generate a series of base classifiers.

Let $\mathbf{x}_i$ be an instance and $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ be a set called a *domain* (also referred to as the *instance space*). If D_s is any fixed probability distribution over the instance space X, and D_t is a distribution over the set of training example (labeled instances) $X \times Y$, then D_s and D_t will be referred to as the *instance distribution* (or *source distribution*) and *target distribution*, respectively. In most of machine learning, it is assumed that $D_s = D_{t_X}$, i.e., the instance distribution and the target distribution are the same over the instance space.

A *concept* c over X is some subset of the instance space X with unknown labels. A concept can be equivalently defined to be a Boolean mapping $c : X \to \{0, 1\}$, with $c(\mathbf{x}) = 1$ indicating that $\mathbf{x}$ is a positive example of c and $c(\mathbf{x}) = 0$ indicating that $\mathbf{x}$ is a negative example.

The Boolean mapping or concept of an instance or domain point $\mathbf{x} \in X$ is sometimes known as a representation h of $\mathbf{x}$.

Definition 6.5 (Agree, Consistent [140]) A representation h and an example $(\mathbf{x}, y)$ *agree* if $h(\mathbf{x}) = y$; otherwise, they *disagree*. Let $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ be any labeled set of instances, where each $\mathbf{x}_i \in X$ and $y_i \in \{0, 1\}$. If c is a concept over X, and $c(\mathbf{x}_i) = y_i$ for all $1 \leq i \leq N$, then the concept or representation c is *consistent* with a sample S (or equivalently, S is consistent with c). Otherwise, they are *inconsistent*.

For M target classes where each class is represented by $C_i, \forall\, i \in \{1, \ldots, M\}$, the task of a classifier is to assign the input sample $\mathbf{x}$ to one of the $(M + 1)$ classes, with the $(M + 1)$th class denoting that the classifier rejects $\mathbf{x}$. Let $h_1, \ldots, h_K$ be K classifiers in which each gives its own classification result $h_i(\mathbf{x})$. The problem is how to produce a combined result $H(\mathbf{x}) = j,\ j \in \{1, \ldots, M, M + 1\}$ from all K predictions $h_1(\mathbf{x}), \ldots, h_K(\mathbf{x})$. The most common method to combine the outputs is the *majority voting* method.

Majority voting consists of the following three steps [281].

1. Define a binary function to represent the number of votes:

$$V_k(\mathbf{x} \in C_i) = \begin{cases} 1, \text{ if } h_k(\mathbf{x}) = i,\ i \in \{1, \ldots, M\}; \\ \\ 0, \text{ otherwise.} \end{cases} \tag{6.4.1}$$

2. Sum the votes from all K classifiers for each C_i:

$$V_H(\mathbf{x} \in C_i) = \sum_{k=1}^{K} V_k(\mathbf{x} \in C_i), \quad i = 1, \ldots, M. \tag{6.4.2}$$

3. The combined result $H(\mathbf{x})$ is determined by

$$H(\mathbf{x}) = \begin{cases} i, & \text{if } \max_{i\in\{1,\ldots,M\}} V_E(\mathbf{x} \in C_i) \text{ and } V_E(\mathbf{x} \in C_i) \geq \alpha \cdot K; \\ M+1, & \text{otherwise.} \end{cases} \tag{6.4.3}$$

Here α is a user-defined threshold that controls the confidence in the final decision.

For a learning algorithm, we are primarily interested in its computational efficiency.

Definition 6.6 (Polynomially Evaluatable [140]) Let C be a representation class over X. C is said to be *polynomially evaluatable* if there is a polynomial-time (possibly randomized) evaluation algorithm A that, on input a representation $c \in C$ and a domain point $\mathbf{x} \in X$, runs in time polynomial in $|c|$ and $|x|$ and decides if $\mathbf{x} \in c(\mathbf{x})$.

A machine learning algorithm is expected to be a strong learning algorithm that is polynomially evaluatable and has a producing representation h that is consistent with a given sample S. However, it is difficult, even impractical, to design directly a strong learning algorithm. A simple and effective approach is to use a boosting framework to a weak machine learning algorithm.

A machine learner for producing a two-class classifier is a weak learner if it is a coinflip algorithm based completely on random guessing. Schapire [221] showed that a weak learner could always improve its performance by training two additional classifiers on filtered versions of the input data stream. After learning an initial classifier h_1 on the first N training points, if two additional classifiers are produced as follows [100, 221]:

1. h_2 is learned on a new sample of N points, half of which are misclassified by h_1,
2. h_3 is learned on N points for which h_1 and h_2 disagree, and
3. the boosted classifier is $h_B = \text{Majority Vote}(h_1, h_2, h_3)$,

then a boosted weak learner can produce a two-class classifier with performance guaranteed (with high probability) to be significantly better than a coinflip.

An *additive logistic regression model* is defined as

$$\frac{\log P(Y=1|\mathbf{x})}{\log P(Y=-1|\mathbf{x})} = \beta_0 + \beta_1\mathbf{x}_1 + \cdots + \beta_p\mathbf{x}_p, \tag{6.4.4}$$

where $P(x) = \frac{1}{1+\mathrm{e}^{-x}}$ is called the *logistic function*, commonly called the sigmoid function.

Let $F(\mathbf{x})$ be an additive function of the form

$$F(\mathbf{x}) = \sum_{m=1}^{M} f_m(\mathbf{x}), \tag{6.4.5}$$

and consider a problem for minimizing the exponential criterion

$$J(F) = E\{e^{-yF(\mathbf{x})}\} \tag{6.4.6}$$

for estimation of $F(\mathbf{x})$.

Lemma 6.2 ([100]) $E\{e^{-yF(\mathbf{x})}\}$ *is minimized at*

$$F(\mathbf{x}) = \frac{1}{2}\log\frac{P(y=1|\mathbf{x})}{P(y=-1|\mathbf{x})}. \tag{6.4.7}$$

Hence

$$P(y=1|\mathbf{x}) = \frac{e^{F(\mathbf{x})}}{e^{-F(\mathbf{x})}+e^{F(\mathbf{x})}}, \tag{6.4.8}$$

$$P(y=-1|\mathbf{x}) = \frac{e^{-F(\mathbf{x})}}{e^{F(\mathbf{x})}+e^{-F(\mathbf{x})}}. \tag{6.4.9}$$

Lemma 6.2 shows that for an additive function $F(\mathbf{x})$ with the form (6.4.5), minimizing the exponential criterion $J(F)$ in (6.4.6) is an additive logistic regression problem in two classes. Friedman, Hastie, and Tibshirani [100] developed a boosting algorithm for additive logistic regression, called simply *LogitBoost*, as shown in Algorithm 6.5.

Algorithm 6.5 LogitBoost (two classes) [100]

```
1. input: weights $w_i = 1/N$, $i = 1, \ldots, N$, $F(\mathbf{x}) = 0$ and probability estimates $p(x_i) = 1/2$.
2. for $m = 1$ to $M$ do
3.    Compute the working response and weight
        $z_i = \frac{y_i^* - p(x_i)}{p(x_i)(1-p(x_i))}$,
        $w_i = p(x_i)(1 - p(x_i))$.
4.    Fit the function $f_m(\mathbf{x})$ by a weighted least squares regression of $z_i$ to $x_i$ using weights $w_i$.
5.    Update $F(\mathbf{x}) \leftarrow F(\mathbf{x}) + \frac{1}{2} f_m(\mathbf{x})$ and $p(\mathbf{x}) \leftarrow (e^{F(\mathbf{x})})/(e^{F(\mathbf{x})} + e^{-F(\mathbf{x})})$.
6. end for
7. output: the classifier $\text{sign}[F(\mathbf{x})] = \text{sign}[\sum_{m=1}^{M} f_m(\mathbf{x})]$.
```

6.4.2 *Probably Approximately Correct Learning*

We are given a set of N training examples $X = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ drawn randomly from $X \times Y$ according to distribution D_s, where Y is the label set consisting of just two possible labels $Y = \{0, 1\}$ in two-classification or C possible labels $Y \in \{1, \ldots, C\}$ in multi-classification. Machine learner is to find a hypothesis or representation h_f which is consistent with most of the sample (i.e., $h_f(\mathbf{x}_i) = y_i$ for most $1 \leq i \leq N$). In general, a hypothesis which is accurate on the training set might not be accurate on examples outside the training set; this problem is sometimes referred to as "*overfitting*" [97]. To avoid overfitting is one of the important problems that must be considered in machine learning.

Definition 6.7 (Probably Approximately Correct (PAC) Model [141]) Let $\mathcal{C}$ be a concept class over X. $\mathcal{C}$ is said to be *probably approximately correct* (PAC) *learnable* if there exists an algorithm L with the property: given an *error parameter* ϵ and a *confidence parameter* δ, for every concept $c \in \mathcal{C}$, for every distribution D_s on X, and for all $0 \le \epsilon \le 1/2$ and $0 \le \delta \le 1/2$, if L, after some amount of time, outputs a hypothesis concept $h \in \mathcal{C}$ satisfying $\text{error}(h) \le \epsilon$ with probability $1 - \delta$.

If L runs in time polynomial in $1/\epsilon$ and $1/\delta$, then c is said to be *efficiently PAC learnable*. The hypothesis $h \in \mathcal{C}$ of the PAC learning algorithm is "approximately correct" with high probability, hence is named as "*Probably Approximately Correct*" (PAC) *learning* [141].

In machine learning, it is usually difficult and even impracticable to find a learner whose concept over X, $c(\mathbf{x}_i) = y_i$ for all labels $1 \le i \le N$. Hence, the goal of PAC learning is to find a hypothesis $h : X \to \{0, 1\}$ which is consistent with *most* (rather than *all*) of the sample (i.e., $h(\mathbf{x}_i) = y_i$ for most $1 \le i \le N$). After some amount of time, the learner must output a hypothesis $h : X \to \{0, 1\}$. The value $h(\mathbf{x})$ can be interpreted as a randomized prediction of the label of $\mathbf{x}$ that is 1 with probability $h(\mathbf{x})$ and 0 with probability $1 - h(\mathbf{x})$.

Definition 6.8 (Strong PAC-Learning Algorithm [97]) A *strong PAC-learning algorithm* is an algorithm that, given an accuracy $\epsilon > 0$ and a reliability parameter $\delta > 0$ and access to random examples, outputs with probability $1 - \delta$ a hypothesis with error at most ϵ.

Definition 6.9 (Weak PAC-Learning Algorithm [97]) A *weak PAC-learning algorithm* is an algorithm that, given an accuracy $\epsilon > 0$ and access to random examples, outputs with probability $1 - \delta$ a hypothesis with error at least $1/2 - \gamma$, where $\gamma > 0$ is either a constant or decreases as $1/p$ where p is a polynomial in the relevant parameters.

In strong PAC learning, the learner is required to generate a hypothesis whose error is smaller than the required accuracy ϵ. On the other hand, in weak PAC learning, the accuracy of the hypothesis is required to be just slightly better than $1/2$, which is the accuracy of a completely random guess. When learning with respect to a given distribution over the instances, weak and strong learning are not equivalent.

For a great assortment of learning problems, the "*boosting algorithm*" can convert a "weak" PAC learning algorithm that performs just slightly better than random guessing into one with arbitrarily high accuracy [97].

There are two frameworks in which boosting can be applied: boosting by filtering and boosting by sampling [96].

Adaptive boosting (AdaBoost) of Freund and Schapire [97] is a popular boosting algorithm by sampling, which has been used in conjunction with a wide range of other machine learning algorithms to enhance their performance.

Finding multiple weak classification algorithms with low recognition rate is much easier than finding a strong classification algorithm with high recognition rate. AdaBoost aims at boosting the accuracy of a weak learner by carefully adjusting the weights of training instances and learn a classifier accordingly. After T such iterations, the final hypothesis h_f is output. The hypothesis h_f combines the outputs of the T weak hypotheses using a weighted majority vote. Algorithm 6.6 gives the adaptive boosting (AdaBoost) algorithm in [97].

Algorithm 6.6 Adaptive boosting (AdaBoost) algorithm [97]

```
1. **input:**
   1.1 sequence of $N$ labeled examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$,
   1.2 distribution $D$ over the $N$ examples,
   1.3 weak learning algorithm **WeakLearn**,
   1.4 integer $T$ specifying number of iterations.
2. **initialization:** the weight vector $w_i^1 = D(i)$ for $i = 1, \ldots, N$.
3. **for** $t = 1$ to $T$ **do**
4. Set $\mathbf{p}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^N w_i^t}$.
5. Call **WeakLearn**, providing it with the distribution $\mathbf{p}_t$; get back a hypothesis $h_t : X \to \{0, 1\}$.
6. Calculate the error of $\epsilon_t = \sum_{i=1}^N p_i^t |h_t(\mathbf{x}_i) - y_i|$.
7. Set $\beta_t = \epsilon_t/(1 - \epsilon_t)$.
8. Set the new weights vector to be $w_i^{t+1} = w_i^t \beta_t^{1-|h_t(\mathbf{x}_i)-y_i|}$.
9. **output:** the hypothesis
   $h_f(\mathbf{x}) = \begin{cases} 1, & \text{if } \sum_{t=1}^T (\log 1/\beta_t) h_t(\mathbf{x}) \geq \frac{1}{2} \sum_{t=1}^T \log(1/\epsilon_t); \\ 0, & \text{otherwise.} \end{cases}$.
```

Friedman et al. [100] analyze the AdaBoost procedures from a statistical perspective: AdaBoost can be rederived as a method for fitting an additive model $\sum_m f_m(\mathbf{x})$ in a forward stagewise manner, which largely explains why it tends to outperform a single base learner. By fitting an additive model of different and potentially simple functions, it expands the class of functions that can be approximated.

Using Newton stepping rather than exact optimization at each step, Friedman *et al.* [100] proposed a modified version of the AdaBoost algorithm, called "Gentle AdaBoost" procedure, which instead takes adaptive Newton steps much like the LogitBoost algorithm, see Algorithm 6.7.

Algorithm 6.7 Gentle AdaBoost [100]

```
1. **input:** weights $w_i = 1/N$, $i = 1, \ldots, N$, $F(\mathbf{x}) = 0$.
2. **for** $m = 1$ to $M$ **do**
3.    Fit the regression function $f_m(\mathbf{x})$ by weighted least squares of $y_i$ to $\mathbf{x}_i$ with weights $w_i$.
4.    Update $F(\mathbf{x}) \leftarrow F(\mathbf{x}) + f_m(\mathbf{x})$.
5.    Update $w_i \leftarrow w_i \exp(-y_i f_m(\mathbf{x}_i))$ and renormalize.
6. **endfor**
7. **output:** the classifier $\text{sign}[F(\mathbf{x})] = \text{sign}[\sum_{m=1}^M f_m(\mathbf{x})]$.
```

The application of adaptive boosting in transfer learning will be discussed in Sect. 6.20.

6.5 Basic Theory of Machine Learning

The pioneer of machine learning, Arthur Samuel, defined machine learning as a "field of study that gives computers the ability to learn without being explicitly programmed."

6.5.1 *Learning Machine*

Consider a learning machine whose task is to learn a mapping $\mathbf{x}_i \to y_i$. The machine is actually defined by a set of possible mappings $\mathbf{x} \to f(\mathbf{x}, \boldsymbol{\alpha})$ with the functions $f(\mathbf{x}, \boldsymbol{\alpha})$ labeled by the adjustable parameter vector $\boldsymbol{\alpha}$.

The machine is usually assumed to be deterministic: for a given input vector $\mathbf{x}$ and choice of $\boldsymbol{\alpha}$, it will always give the same output $f(\mathbf{x}, \boldsymbol{\alpha})$. A particular choice of $\boldsymbol{\alpha}$ generates a "*trained machine*," and hence a neural network with fixed architecture and containing $\boldsymbol{\alpha}$ corresponding to the weights and biases is called a *learning machine* in this sense [39].

The expectation of the test error for a trained machine is defined as

$$R(\boldsymbol{\alpha}) = \int \frac{1}{2}|y - f(\mathbf{x}, \boldsymbol{\alpha})|\mathrm{d}P(\mathbf{x}, y), \tag{6.5.1}$$

where $P(\mathbf{x}, \boldsymbol{\alpha})$ is some unknown cumulative probability distribution from which data $\mathbf{x}$ and y are drawn, i.e., the data are assumed to be independently and identically distributed (i.i.d.). When a density of cumulative probability distributions, $P(\mathbf{x}, y)$, exists, $\mathrm{d}P(\mathbf{x}, y)$ may be written as $p(\mathbf{x}, y)\mathrm{d}\mathbf{x}\mathrm{d}y$.

The quantity $R(\boldsymbol{\alpha})$ is called the expected risk, or just the risk for choice of $\boldsymbol{\alpha}$. The "*empirical risk*" $R_{\mathrm{emp}}(\boldsymbol{\alpha})$ is defined to be just the measured mean error rate on the given training set:

$$R_{\mathrm{emp}}(\boldsymbol{\alpha}) = \frac{1}{2N}\sum_{i=1}^{N}|y_i - f(\mathbf{x}_i, \boldsymbol{\alpha})|, \tag{6.5.2}$$

where no probability distribution appears.

The quantity $\frac{1}{2}|y_i - f(\mathbf{x}_i, \boldsymbol{\alpha})|$ is called the loss. $R_{\mathrm{emp}}(\boldsymbol{\alpha})$ is a fixed number for a particular choice of $\boldsymbol{\alpha}$ and for a particular training set $\{\mathbf{x}_i, y_i\}$.

Another name for a family of functions $f(\mathbf{x}, \boldsymbol{\alpha})$ is "learning machine." Designing a good learning machine depends on choice of $\boldsymbol{\alpha}$. A popular method for building $\boldsymbol{\alpha}$

is *empirical risk minimization* (ERM). By ERM, we mean $R_{\text{emp}}(\boldsymbol{\alpha})$ is minimized over all possible choices of $\boldsymbol{\alpha}$, results in a risk $R(\boldsymbol{\alpha}^*)$ which is close to its minimum.

The minimization of different loss functions results in different machine learning methods. In reality, most machine learning methods should have three phases rather than two: training, validation, and testing. After the training is complete, there maybe several models (e.g., artificial neural networks) available, and one needs to decide which one to have a good estimation of the error it will achieve on a test set. For this end, there should be a third separate data set: the validation data set.

6.5.2 Machine Learning Methods

There are many different machine learning methods for modeling the data to the underlying problem. These machine learning methods can be divided to the following types [38].

1. **Machine Learning Methods based on Network Structure.**

   - *Artificial Neural Networks (ANNs):* ANNs are inspired by the brain and composed of interconnected artificial neurons capable of certain computations on their inputs [120]. The input data activate the neurons in the first layer of the network whose output is the input to the second layer of neurons in the network. Similarly, each layer passes its output to the next layer and the last layer outputs the result. Layers in between the input and output layers are referred to as hidden layers. When an ANN is used as a classifier, the output layer generates the final classification category.
   - *Bayesian Network:* A Bayesian network is a probabilistic graphical model that represents the variables and the relationships between them [98, 120, 130]. The network is constructed with nodes as the discrete or continuous random variables and directed edges as the relationships between them, establishing a directed acyclic graph. Each node maintains the states of the random variable and the conditional probability form. Bayesian networks are built by using expert knowledge or efficient algorithms that perform inference.

2. **Machine Learning Methods based on Statistical Analysis.**

   - *Association Rules:* The concept of association rules was popularized particularly due to the article of Agrawal et al. in 1993 [3]. The goal of association rule mining is to discover previously unknown association rules from the data. An association rule describes a relationship $X \Rightarrow Y$ between an itemset X and a single item Y. Association rules have two metrics that tell how often a given relationship occurs in the data: the *support* is an indication of how frequently the itemset appears in the data set, and the *confidence* is an indication of how often the rule has been found to be true.

- *Clustering:* Clustering [126] is a set of techniques for finding patterns in high-dimensional unlabeled data. It is an unsupervised pattern discovery approach where the data are grouped together based on some similarity measure.
- *Ensemble Learning:* Supervised learning algorithms, in general, search the hypothesis space to determine the right hypothesis that will make good predictions for a given problem. Although good hypotheses might exist, it may be hard to find one. Ensemble methods combine multiple learning algorithms to obtain the better predictive performance than the constituent learning algorithms alone. Often, ensemble methods use multiple weak learners to build a strong learner [206].
- *Hidden Markov Models (HMMs):* An HMM is a statistical Markov model where the system being modeled is assumed to be a Markov process with unobserved (i.e., hidden) states [17]. The main challenge is to determine the hidden parameters from the observable parameters. The states of an HMM represent unobservable conditions being modeled. By having different output probability distributions in each state and allowing the system to change states over time, the model is capable of representing non-stationary sequences.
- *Inductive Learning:* Deduction and induction are two basic techniques of inferring information from data. Inductive learning is traditional supervised learning, and aims at learning a model from labeled examples, and trying to predict the labels of examples we have not seen or known about. In inductive learning, from specific observations one begins to detect patterns and regularities, formulates some tentative hypotheses to be explored, and lastly ends up developing some general conclusions or theories. Several ML algorithms are inductive, but by inductive learning, we usually mean "repeated incremental pruning to produce error reduction" (RIPPER) [57] and the quasi-optimal (AQ) algorithm [177].
- *Naive Bayes:* It is well known [98] that a surprisingly simple Bayesian classifier with strong assumptions of independence among features, called naive Bayes, is competitive with state-of-the-art classifiers such as C4.5. In general, the input features are assumed to be independent, whereas in practice this is seldom true. Naive Bayes classifiers [98, 270] can handle an arbitrary number of independent features whether continuous or categorical by reducing a high-dimensional density estimation task to a one-dimensional kernel density estimation, under the assumption that the features are independent. Although the Naive Bayes classifier has some limitations, it is an optimal classifier if the features are conditionally independent given the true class. One of the biggest advantages of the Naive Bayes classifier is that it is an online algorithm and its training can be completed in linear time.

3. **Machine Learning Methods based on Evolution.**

- *Evolutionary Computation:* In computer science, evolutionary computation is a family of algorithms for global optimization inspired by biological evolution, and the subfield of artificial intelligence and soft computing studying these algorithms (by Wikipedia). The term evolutionary computation encompasses genetic algorithms (GA) [107], genetic programming (GP) [152], evolution strategies [26], particle Swarm optimization [142], ant colony optimization [78], and artificial immune systems [89], which is the topics in Chap. 9 Evolutionary Computations.

6.5.3 Expected Performance of Machine Learning Algorithms

A machine learning algorithm is expected to have the following *expected performance* [145].

- *Scalability:* This parameter can be defined as the ability for an algorithm to be able to handle an increase in its scale, such as feeding more data to the system, adding more features to the input data or adding more layers in a neural network, without it limitlessly increasing its complexity [5].
- *Training Time:* This is the amount of time that a machine learning algorithm takes to be fully trained and form the ability to make its predictions.
- *Response Time:* This parameter is related to the agility of a machine learning system, and represents the time that an algorithm takes, after it has been trained, to make a prediction for the desired self-organizing networks (SON) function.
- *Training Data:* This metric of machine learning algorithm is the amount and type of training data an algorithm needs. Algorithms supported by more training data usually have better accuracy, but they also take more time to be trained.
- *Complexity:* Complexity of a system can be defined as the amount of mathematical operations that it performs in order to achieve a desired solution. This parameter can determine if certain algorithms are more suitable to be deployed at the user side.
- *Accuracy:* Future networks are expected to be much more intelligent and quicker, enabling highly different types of applications and user requirements. Deploying algorithms that have high accuracy is critical to guarantee a good operability of certain self-organizing network functions.
- *Convergence Time:* This metric of an algorithm, different from the response time, relates to how fast it agrees that the solution found for that particular problem is the optimal solution at that time.
- *Convergence Reliability:* This parameter represents the susceptibility of some algorithm to be stuck at local minima and how initial conditions can affect its performance.

6.6 Classification and Regression

In essence, machine learning is to learn the given training samples for solving one of two basic problems: regression (for continuous outputs) or classification (for discrete outputs). *Classification* is closely related to pattern recognition, its goal is to design a classifier thought learning a "training" set of input data for making recognition or classification for unknown samples. By regression, it means to design a regressor or predictor based on machine learning results of a set of training data for making a prediction for unknown continuous samples.

6.6.1 Pattern Recognition and Classification

Pattern recognition is the field devoted to the study of methods designed to categorize data into distinct classes. By Watanabe [262], a *pattern* is defined "as opposite of a chaos; it is an entity, vaguely defined, that could be given a name."

Pattern recognition is widely applied in recognitions of human biometrics (such as a person's face, fingerprint, iris) and various targets (such as aircraft, ships). In these applications, the extraction of object features is crucial. For example, when a target is considered as a linear system, the target parameter is a feature of the target signal.

Given a pattern, its recognition/classification may consist of one of the following two tasks [262]:

- *Supervised classification* (e.g., discriminant analysis) in which the input pattern is identified as a member of a predefined class;
- *Unsupervised classification* (e.g., clustering) in which the pattern is assigned to a hitherto unknown class.

The tasks of pattern recognition/classification are customarily divided into four distinct blocks:

1. *Data representation (acquisition and pre-processing).*
2. *Feature selection or extraction.*
3. *Clustering.*
4. *Classification.*

The four best known approaches for pattern recognition are [128]: (1) template matching, (2) statistical classification, (3) syntactic or structural matching, and (4) neural networks.

Data Representation

Data representation is mostly problem-specific. In matrix algebra, signal processing, pattern recognition, etc., data pre-processing most commonly involves the zero-mean normalization (i.e., *data centering*): for each data vector $\mathbf{x}_n$,

$$x_{i,j}^{\text{cent}} = x_{i,j} - \bar{x}, \quad \bar{x} = \frac{1}{N}\sum_{j=1}^{N} x_{i,j}, \quad \text{for } i = 1, \ldots, N \tag{6.6.1}$$

to remove the useless direct current (DC) components in data, where $x_{i,j}$ is the jth entry of the vector $\mathbf{x}_i$. The data centering is also referred to the data zero-meaning.

In order to prevent drastic changes in data amplitude, the input data are also required to be scaled to similar ranges:

$$x_{i,j}^{\text{scal}} = \frac{x_{i,j}}{s_i}, \quad s_i = \sqrt{\sum_{k=1}^{n} x_k^2}, \quad i = 1, \ldots, N; \; j = 1, \ldots, n. \tag{6.6.2}$$

This pre-processing is called the *data scaling*.

Feature Selection

In many applications, data vectors must become low-dimensional vectors via some transformation/processing method. These low-dimensional vectors are called the *pattern vectors* or *feature vectors* owing to the fact that they extract the features of the original data vectors, and are directly used for pattern clustering and classification. For example, colors of clouds and parameters of voice tones are pattern or feature vectors in weather forecasting and voice classification, respectively.

Due to high dimensionality, the original data vectors are not directly available for pattern recognition. Hence, one should try to find invariant features in data that describe the differences in classes as best as possible.

According to the given training set, feature selection can be divided into supervised and unsupervised methods.

- *Supervised feature selection:* Let $X_l = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ be the labeled set of data, where $\mathbf{x}_i \in \mathbb{R}^n$ is the ith data vectors and $y_i = k$ indicates the kth class to which the data vector $\mathbf{x}_i$ belongs, where $k \in \{1, \ldots, M\}$ is the label of M target classes. The pattern recognition uses supervised machine learning. Let l_p be the number of $k = p$ and $l_1 + \cdots + l_M = M$, we can construct the data matrix in the pth class of targets:

$$\mathbf{x}^{(p)} = \left[\mathbf{x}_1^{(p)}, \ldots, \mathbf{x}_{l_p}^{(p)}\right]^T \in \mathbb{R}^{N \times l_p}, \quad p = 1, \ldots, M. \tag{6.6.3}$$

  Thus we can select feature vectors in data by using matrix algebra methods such as principal component analysis, etc. This selection, as a supervised machine learning, is also known as *feature selection* or *dimensionality reduction* since the dimension of feature vector is much less than the dimension of original data vectors.
- *Unsupervised feature selection:* When data are unlabeled, i.e., only data vectors $\mathbf{x}_i, \ldots, \mathbf{x}_N$ are given, the pattern recognition is unsupervised machine learning which is more difficult than supervised patter recognition. In this case, we need to use mapping or signal processing for transforming the data into invariant features such as short time Fourier transform, bispectrum, wavelet, etc.

Feature selection is to select the most significant features of the data, and attempt to reduce the dimensionality (i.e., the number of features) for the remaining steps of the task.

We will discuss supervised and unsupervised feature extractions later in details.

Clustering
Let $\mathbf{w}$ be the weight vector of a cluster which aims at patterns within a cluster being more similar to each other rather than are patterns belonging to other clusters. Clustering methods are used in order to find the actual mapping between patterns and labels (or targets).

This categorization has two machine learning methods: supervised learning (which makes distinct labeling of the data) and unsupervised learning (division of the data into classes), or a combination of more than one of these tasks.

The above three stages (data representation, feature selection, and clustering) belong to the *training phase*. Having such a cluster, one may make a classification in the next testing phase.

Classification
Classification is one of the most frequently encountered decision making tasks of human activity. A classification problem occurs when an object needs to be assigned into a predefined group or classes related to that object after clustering a number of groups or classes based on a set of observed attributes. In other words, one needs to decide which class among a number of classes a given testing sample belongs to. This step is called the *testing phase*, compared with the previous training stage. Clearly, this is a form of supervised learning.

6.6.2 Regression

In statistical modeling and machine learning, *regression* is to fit a statistical process for estimating the relationships among variables. It focuses on the relationship between a dependent variable $\mathbf{x}$ and one or more independent variables (named as "*predictors*") $\boldsymbol{\beta}$. More specifically, regression is to analyze how the typical value of the dependent variable changes when anyone of the independent variables is varied, while the other independent variables are fixed.

Machine learning is widely used for prediction and forecasting, where its use has substantial overlap with the field of statistical regression analysis.

Consider the more general multiple regression model with p independent variables:

$$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i, \tag{6.6.4}$$

where x_{ij} is the ith observation on the jth independent variable $\mathbf{x}_j = [x_{1j}, \ldots, x_{pj}]^T$. If the first independent variable takes the value 1 for all i, $x_{i1} = 1$, then the regression model reduces to

$$y_i = \beta_1 + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i$$

and β_1 is called the *regression intercept*.

The *regression residual* is defined as

$$\varepsilon_i = y_i - \hat{\beta}_1 x_{i1} - \cdots - \hat{\beta}_p x_{ip}. \tag{6.6.5}$$

Hence, the normal equations on regression are given by

$$\sum_{i=1}^{n}\sum_{k=1}^{p} x_{ij}x_{ik}\hat{\beta}_k = \sum_{i=1}^{n} x_{ij}y_i, \quad j = 1, \ldots, p. \tag{6.6.6}$$

In matrix notation, the above normal equations can be written as

$$(\mathbf{X}^T\mathbf{X})\hat{\boldsymbol{\beta}} = \mathbf{X}^T\mathbf{y}, \tag{6.6.7}$$

where $\mathbf{X} = [x_{ij}]_{i=1,j=1}^{n,p}$ is an $n \times p$ matrix, $\mathbf{y} = [y_i]_{i=1}^{n}$ is an $n \times 1$ vector, and $\hat{\boldsymbol{\beta}} = [\hat{\beta}_j]_{j=1}^{p}$ is a $p \times 1$ vector. Finally, the solution of regression problem is given by

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{6.6.8}$$

6.7 Feature Selection

The performance of machine learning methods generally depends on the choice of data representation. The *representation choice* heavily depends on *representation learning*. Representation learning is also known as *feature learning*.

In machine learning, representation learning [22] is a set of learning techniques that allows a system to automatically discover the representations needed for feature detection or classification from raw data.

When mining large data sets, both in dimension and size, we are given a set of original variables (e.g., in gene expression) or a set of extracted features (such as pre-processing via principal or minor component analysis) in data representation. An important problem is how to select a subset of the original variables or a subset of the original features. The former is called the *variable selection*, and the latter is known as the *feature selection or representation choice*.

Feature selection is a fundamental research topic in data mining and machine learning with a long history since the 1970s. With the advance of science and technology, more and more real-world data sets in data mining and machine learning often involve a large number of features. But, not all features are essential since many of them are redundant or even irrelevant, which may significantly degrade the accuracy of learned models as well as reduce the learning speed and ability of the models. By removing irrelevant and redundant features, feature selection can reduce the dimensionality of the data, speed up the learning process, simplify the learned model, and/or improve the performance [69, 112, 284].

A smaller set of representative variables or features, retaining the optimal salient characteristics of the data, not only decreases the processing complexity and time, but also overcomes the risk of "overfitting" and leads to more compactness of the model learned and better generalization. When the class labels of the data are given, we are faced with supervised variable or feature selection, otherwise we should use unsupervised variable or feature selection.

Since the learning methods used in both the variable selection and the feature selection are essentially the same, we take the feature selection as the subject of this section.

6.7.1 Supervised Feature Selection

An important problem in machine learning is to reduce the dimensionality D of the feature space F to overcome the risk of "overfitting." Data overfitting arises when the number n of features is large and the number m of training patterns is comparatively small. In such a situation, one may easily find a decision function that separates the training data but gives a poor result for test data.

Feature selection has become the focus of much research in areas of application (e.g., text processing of internet documents, gene expression array analysis, and combinatorial chemistry) for high-dimensional data. The objective of *feature selection* is three-fold [112]:

- improving the prediction performance of the predictors,
- providing faster and more cost-effective predictors, and
- providing a better understanding of the underlying process that generated the data.

Many feature selection algorithms are based on *variable ranking*. As a pre-processing step of feature selection, variable ranking is a filter method: it is independent of the choice of the predictor. For example, in the gene selection problem, the variables are gene expression coefficients corresponding to the abundance of mRNA in a sample for a number of patients. A typical classification task is to separate healthy patients from cancer patients, based on their gene expression "profile" [112].

The feature selection methods can be divided into two categories: the exhaustive enumeration method and the greedy method.

- *Exhaustive enumeration method* selects the best subset of features satisfying a given “model selection” criterion by exhaustive enumeration of all subsets of features. However, this method is impractical for large numbers of features.
- *Greedy method* is available particularly to the feature selection in large dimensional input spaces. Among various possible methods *feature ranking* techniques are very attractive. A number of top ranked features may be selected for further analysis or to design a classifier. Alternatively, a threshold can be set on the ranking criterion. Only the features whose criterion exceeds the threshold are retained.

Consider a set of m examples $\{\mathbf{x}_k, y_k\}(k = 1,\ldots,m)$, where $\mathbf{x}_k = [x_{k,1},\ldots,x_{k,n}]^T$ and one output variable y_k. Let $\bar{x}_i = \frac{1}{m}\sum_{k=1}^m x_{k,i}$ and $\bar{y} = \frac{1}{m}\sum_{k=1}^m y_k$ stand for an average over the index k. Variable ranking makes use of a scoring function $S(i)$ computed from the values $x_{k,i}$ and y_k, $k = 1,\ldots,m$. By convention, a high score is indicative of a valuable variable (e.g., gene) and variables are sorted in decreasing order of $S(i)$.

A good feature selection method selects first a few of features that individually classify best the training data, and then increases the number of features. Classical feature selection methods include correlation methods and expression ratio methods.

A typical *correlation method* is based on the *Pearson correlation coefficients* defined as

$$R(i) = \frac{\sum_{k=1}^m (x_{k,i}-\bar{x}_i)(y_k-\bar{y})}{\sqrt{\sum_{k=1}^m (x_{k,i}-\bar{x}_i)^2 \sum_{k=1}^m (y_k-\bar{y})^2}}, \tag{6.7.1}$$

where the numerator denotes the *between-class variance* and the denominator represents the *within-class variance*. In linear regression, using the square of $R(i)$ as a variable ranking criterion enforces a ranking according to goodness of linear fit of individual variables.

For two-class data $\{\mathbf{x}_k, y_k\}(k = 1,\ldots,m)$, where $\mathbf{x}_k = [x_{k,1},\ldots,x_{k,n}]^T$ consists of n input variables $x_{k,i}(i = 1,\ldots,n)$ and $y_k \in \{+1,-1\}$ (e.g., certain disease vs. normal), let

$$\mu_i^+ = \frac{1}{m^+}\sum_{k=1}^{m^+} x_{k,i}, \quad \mu_i^- = \frac{1}{m^-}\sum_{k=1}^{m^-} x_{k,i}, \tag{6.7.2}$$

$$\sigma_i^+ = \sum_{k=1}^{m^+}(x_{k,i}-\mu_i^+)^2, \quad \sigma_i^- = \sum_{k=1}^{m^-}(x_{k,i}-\mu_i^-)^2, \tag{6.7.3}$$

where m^+ and m^- are the numbers of the ith variable $x_{k,i}$ corresponding to $y_i = +1$ and $y_i = -1$, respectively; μ_i^+ and μ_i^- are the means of $x_{k,i}$ over the index k associated with $y_i = +1$ and $y_i = -1$, respectively; and σ_i^+ and σ_i^- are, respectively, the standard deviations of $x_{k,i}$ over the index k associated with $y_i = +1$ and $y_i = -1$.

A well-known *expression ratio method* uses the ratio [103, 109]:

$$F(x_i) = \left| \frac{\mu_i^+ - \mu_i^-}{\sigma_i^+ + \sigma_i^-} \right|. \tag{6.7.4}$$

This method selects highest ratios $F(x_i)$ differing most on average in the two classes and having small deviations in the scores in the respective classes as top features.

The *expression ratio criterion* (6.7.4) is similar to Fisher's discriminant criterion [83].

It should be noted that the variable dependencies cannot be ignored in feature selection as follows [112].

- Noise reduction and consequently better class separation may be obtained by adding variables that are presumably redundant.
- Perfectly correlated variables are truly redundant in the sense that no additional information is gained by adding them.
- Very high variable correlation (or anti-correlation) does not mean absence of variable complementarity.
- A variable that is completely useless by itself can provide a significant performance improvement when taken with others.
- Two variables that are useless by themselves can be useful together.

One possible use of feature ranking is the design of a class predictor (or classifier) based on a pre-selected subset of features. The weighted voting scheme yields a particular linear discriminant classifier:

$$D(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} - \boldsymbol{\mu} \rangle = \mathbf{w}^T (\mathbf{x} - \boldsymbol{\mu}), \tag{6.7.5}$$

where $\boldsymbol{\mu} = \frac{1}{2}(\boldsymbol{\mu}^+ + \boldsymbol{\mu}^-)$ with

$$\boldsymbol{\mu}^+ = \frac{1}{n^+} \sum_{\mathbf{x}_i \in X^+} \mathbf{x}_i, \tag{6.7.6}$$

$$\boldsymbol{\mu}^- = \frac{1}{n^-} \sum_{\mathbf{x}_i \in X^-} \mathbf{x}_i, \tag{6.7.7}$$

where $X^+ = \{(\mathbf{x}_i, y_i = +1)\}$, $X^- = \{(\mathbf{x}_i, y_i = -1)\}$, and n^+ and n^- are the numbers of the training data vectors $\mathbf{x}_i$ belonging to the classes (+) and (−), respectively.

Define the $n \times n$ within-class scatter matrix

$$\mathbf{S}_w = \sum_{\mathbf{x}_i \in X^+} (\mathbf{x}_i - \boldsymbol{\mu}^+)(\mathbf{x}_i - \boldsymbol{\mu}^+)^T + \sum_{\mathbf{x}_i \in X^-} (\mathbf{x}_i - \boldsymbol{\mu}^-)(\mathbf{x}_i - \boldsymbol{\mu}^-)^T. \tag{6.7.8}$$

By Fisher's linear discriminant, the classifier is given by [113]

$$\mathbf{w} = \mathbf{S}_w^{-1}(\boldsymbol{\mu}^+ - \boldsymbol{\mu}^-). \tag{6.7.9}$$

Once the classifier $\mathbf{w}$ is designed using the training data $(\mathbf{x}_i, y_i), i = 1, \ldots, n$ (where $y_i \in \{+1, -1\}$), for any given data vector $\mathbf{x}$, the new pattern can be classified according to the sign of the decision function [113]:

$$D(\mathbf{x}) = \mathbf{w}^T\mathbf{x} > 0 \;\Rightarrow\; \mathbf{x} \in \text{class } (+),$$
$$D(\mathbf{x}) = \mathbf{w}^T\mathbf{x} < 0 \;\Rightarrow\; \mathbf{x} \in \text{class } (-),$$
$$D(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = 0, \quad \text{decision boundary}.$$

6.7.2 Unsupervised Feature Selection

Consider the unsupervised feature selection for given data vectors $\mathbf{x}_i, i = 1, \ldots, n$ without labeled class index $y_i, i = 1, \ldots, n$. In this case, there are two common similarity measures between two random vectors $\mathbf{x}$ and $\mathbf{y}$, as stated below [182].

1. *Correlation coefficient:* The correlation coefficient $\rho(\mathbf{x}, \mathbf{y})$ between two random vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$\rho(\mathbf{x}, \mathbf{y}) = \frac{\text{cov}(\mathbf{x}, \mathbf{y})}{\sqrt{\text{var}(\mathbf{x})\text{var}(\mathbf{y})}}, \tag{6.7.10}$$

   where $\text{var}(\mathbf{x})$ denotes the variance of the random vector $\mathbf{x}$ and $\text{cov}(\mathbf{x}, \mathbf{y})$ denotes the covariance between $\mathbf{x}$ and $\mathbf{y}$. The correlation coefficient has the following properties.

   (a) $0 \leq 1 - |\rho(\mathbf{x}, \mathbf{y})| \leq 1$.
   (b) $1 - |\rho(\mathbf{x}, \mathbf{y})| = 0$ if and only if $\mathbf{x}$ and $\mathbf{y}$ are linearly dependent.
   (c) *Symmetric:* $1 - |\rho(\mathbf{x}, \mathbf{y})| = 1 - |\rho(\mathbf{y}, \mathbf{x})|$.
   (d) *Invariant to scaling and translation:* if $\mathbf{u} = \frac{\mathbf{x}-\mathbf{a}}{c}$ and $\mathbf{v} = \frac{\mathbf{y}-\mathbf{b}}{d}$ for some constant vectors $\mathbf{a}$,$\mathbf{b}$ and constants c, d, then $1 - |\rho(\mathbf{x}, \mathbf{y})| = 1 - |\rho(\mathbf{u}, \mathbf{v})|$.
   (e) *Sensitive to rotation:* If $(\mathbf{u}, \mathbf{v})$ is some rotation of $(\mathbf{x}, \mathbf{y})$, then $|\rho(\mathbf{x}, \mathbf{y})| \neq |\rho(\mathbf{u}, \mathbf{v})|$.

2. *Least square regression error:* Let $\mathbf{y} = a\mathbf{1} + b\mathbf{x}$ be the linear regression of $\mathbf{x}$ with the regression coefficients a and b. Least square regression error is defined as

$$e^2(\mathbf{x}, \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n}(e(\mathbf{x}, \mathbf{y})_i)^2, \tag{6.7.11}$$

where $e(\mathbf{x}, \mathbf{y})_i = y_i - a - bx_i$. The regression coefficients are given by $a = \frac{1}{n}\sum_{i=1}^n y_i$, $b = \frac{\text{cov}(\mathbf{x},\mathbf{y})}{\text{var}(\mathbf{x})}$ and the mean square error $e(\mathbf{x}, \mathbf{y}) = \text{var}(\mathbf{y})(1-\rho(\mathbf{x}, \mathbf{y})^2)$. If $\mathbf{x}$ and $\mathbf{y}$ are linearly correlated, then $e(\mathbf{x}, \mathbf{y}) = 0$, and if $\mathbf{x}$ and $\mathbf{y}$ are completely uncorrelated, then $e(\mathbf{x}, \mathbf{y}) = \text{var}(\mathbf{y})$. The measure e^2 is also called the *residual variance*. The least square regression error measure has the following properties.

(a) $0 \leq e(\mathbf{x}, \mathbf{y}) \leq \text{var}(\mathbf{y})$.
(b) $e(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x}$ and $\mathbf{y}$ are linearly correlated.
(c) *Unsymmetric:* $e(\mathbf{x}, \mathbf{y}) \neq e(\mathbf{y}, \mathbf{x})$.
(d) *Sensitive to scaling:* If $\mathbf{u} = \mathbf{x}/c$ and $\mathbf{v} = \mathbf{y}/d$ for some constants c and d, then $e(\mathbf{x}, \mathbf{y}) = d^2 e(\mathbf{u}, \mathbf{v})$. However, e is invariant to translation of the variables.
(e) The measure e is sensitive to the rotation of the scatter diagram in x–y plane.

A feature similarity measure is expected to be symmetric, sensitive to scaling and translation of variables, and invariant to their rotation. Hence, the above two similarity measures are not expected, because the correlation coefficient has the unexpected properties (d) and (e), and the least square regression is not symmetric (property (c)) and is sensitive to the rotation (property (d)).

In order to have all the desired properties, Mitra et al. [182] proposed a *maximal information compression index* λ_2 defined as

$$2\lambda_2(\mathbf{x}, \mathbf{y}) = \text{var}(\mathbf{x}) + \text{var}(\mathbf{y}) - \sqrt{(\text{var}(\mathbf{x}) + \text{var}(\mathbf{y}))^2 - 4\text{var}(\mathbf{x})\text{var}(\mathbf{y})(1 - \rho(\mathbf{x}, \mathbf{y})^2)}. \tag{6.7.12}$$

Here, $\lambda_2(\mathbf{x}, \mathbf{y})$ denotes the smallest eigenvalue of the covariance matrix $\boldsymbol{\Sigma}$ of the random variables $\mathbf{x}$ and $\mathbf{y}$.

The feature similarity measure $\lambda_2(\mathbf{x}, \mathbf{y})$ has the following properties [182].

1. $0 \leq \lambda_2(\mathbf{x}, \mathbf{y}) \leq 0.5(\text{var}(\mathbf{x}) + \text{var}(\mathbf{y}))$.
2. $\lambda_2(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x}$ and $\mathbf{y}$ are linearly correlated.
3. *Symmetric:* $\lambda_2(\mathbf{x}, \mathbf{y}) = \lambda_2(\mathbf{y}, \mathbf{x})$.
4. *Sensitive to scaling:* If $\mathbf{u} = \mathbf{x}/c$ and $\mathbf{v} = \mathbf{y}/d$ for some nonzero constants c and d, then $\lambda_2(\mathbf{x}, \mathbf{y}) \neq \lambda_2(\mathbf{u}, \mathbf{v})$. Since the expression of $\lambda_2(\mathbf{x}, \mathbf{y})$ does not contain the mean but only the variance and covariance terms, it is invariant to translation of $\mathbf{x}$ and $\mathbf{y}$.
5. λ_2 is *invariant to rotation* of the variables about the origin.

Let the original number of features be D, and the original feature set be $O = \{F_i, i = 1, \ldots, D\}$. Apply the measures of linear dependency $\big($e.g., $\rho(F_i, F_j)$, $e(F_i, F_j)$, and/or $\lambda_2(F_i, F_j)\big)$ to compute the dissimilarity between F_i and F_j, denoted by $d(F_i, F_j)$. Smaller the value of $d(F_i, F_j)$ is, the more similar are the features F_i and F_j. Let r_i^k denote the dissimilarity between feature F_i and its nearest neighbor feature in R, and $\inf_{F_i \in R} r_i^k$ represent the infimum of r_i^k for $F_i \in R$.

Algorithm 6.8 is the feature clustering algorithm of Mitra et al. [182].

Algorithm 6.8 Feature clustering algorithm [182]

```
1.  **input:** the original feature set $O = \{F_1, \ldots, F_D\}$.
2.  **initialization:** Choose an initial value of $k \leq d_1$ and set $R \leftarrow O$.
3.  For each $F_i \in R$ compute $r_i^k$.
4.  Find feature $F_{i'}$ for which $r_{i'}^k$ is minimum. Retain this feature in $R$ and discard $k$ nearest
    features of $F_{i'}$.
5.  Let $\epsilon = r_{i'}^k$.
6.  If $k > |R| - 1$ then $k = |R| - 1$.
7.  If $k = 1$ then goto Step 14.
8.  **While** $r_{i'}^k > \epsilon$ **do**
9.      (a) $k = k - 1$.
10.         $r_{i'}^k = \inf_{F_i \in R} r_i^k$.
            ("$k$ is decremented by 1, until the "$k$th nearest neighbor of at least one of features in $R$
             is less than $\epsilon$-dissimilar with the feature.)
11.     (b) If $k = 1$ then goto Step 14.
            (if no feature in $R$ has less than $\epsilon$-similar the "nearest neighbor" select all the remaining
             feature in $R$.)
12. **end while**
13. Goto Step 3.
14. **output:** return feature set $R$ as the reduced feature set.
```

6.7.3 Nonlinear Joint Unsupervised Feature Selection

Let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ be the original feature matrix of n data samples, where $\mathbf{x}_i \in \mathbb{R}^D$ and x_{ip} denote the value of the pth ($p = 1, \ldots, D$) feature of $\mathbf{x}_i$. Consider selecting d ($d \leq D$) high-quality features. To this end, use $\mathbf{s} \in \{0, 1\}^D$ as the selection indicator vector, where $s_p = 1$ indicates the pth feature is selected and $s_p = 0$ indicates not selected.

Given a feature mapping $\boldsymbol{\phi}(\mathbf{x}) : X \to F$, $\boldsymbol{\phi}(\mathbf{x}) - E\{\boldsymbol{\phi}(\mathbf{x})\}$ is called the centered feature mapping, where $E\{\boldsymbol{\phi}(\mathbf{x})\}$ is its expectation.

Definition 6.10 (Centered Kernel Function [267]) The kernel function $K \in \mathbb{R}^{n\times n}$ after centering is given by

$$K_c(\mathbf{x}_i, \mathbf{x}_j) = (\boldsymbol{\phi}(\mathbf{x}_i) - E\{\boldsymbol{\phi}(\mathbf{x}_i)\})^T (\boldsymbol{\phi}(\mathbf{x}_i) - E\{\boldsymbol{\phi}(\mathbf{x}_i)\}). \tag{6.7.13}$$

For a finite number of samples, a feature vector is centered by subtracting its empirical expectation, i.e., $\boldsymbol{\phi}(\mathbf{x}_i) - \bar{\boldsymbol{\phi}}$, where $\bar{\boldsymbol{\phi}} = \frac{1}{n}\sum_{j=1}^n \boldsymbol{\phi}(\mathbf{x}_j)$.

Definition 6.11 (Centered Kernel Matrix [267]) The kernel matrix $\mathbf{K} \in \mathbb{R}^{n\times n}$ after centering is defined by its entry as follows:

$$(K_c)_{ij} = K_{ij} - \frac{1}{n}\sum_{i=1}^n K_{ij} - \frac{1}{n}\sum_{j=1}^n K_{ij} + \frac{1}{n^2}\sum_{i=1}^n\sum_{j=1}^n K_{ij}. \tag{6.7.14}$$

Denoting $\mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T$, then the kernel matrix $\mathbf{K} \in \mathbb{R}^{n\times n}$ after centering can be expressed as $\mathbf{K}_c = \mathbf{HKH}$. It is easy to know that $\mathbf{H}$ is an idempotent matrix, i.e., $\mathbf{HH} = \mathbf{H}$.

Definition 6.12 (Kernel Alignment [65, 267]) For two kernel matrices $\mathbf{K}_1 \in \mathbb{R}^{n\times n}$ and $\mathbf{K}_2 \in \mathbb{R}^{n\times n}$ (assume $\|\mathbf{K}_1\|_F > 0$ and $\|\mathbf{K}_2\|_F > 0$), the alignment between $\mathbf{K}_1$ and $\mathbf{K}_2$ is defined as the form of matrix trace: $\rho(\mathbf{K}_1, \mathbf{K}_2) = \mathrm{tr}(\mathbf{K}_1\mathbf{K}_2)$.

Assume $\mathbf{L} \in \mathbb{R}^{n\times n}$ is a kernel matrix computed from original feature matrix $\mathbf{X}$, $\mathbf{K} \in \mathbb{R}^{n\times n}$ is a kernel matrix computed from selected features.

Denoting

$$\boldsymbol{\Phi}(\mathbf{X}) = [\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_n)] \in \mathbb{R}^{D\times n}, \tag{6.7.15}$$

$$\boldsymbol{\Phi}(\mathbf{X}_s) = [\boldsymbol{\phi}(\mathbf{Diag}(\mathbf{s})\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{Diag}(\mathbf{s})\mathbf{x}_n)], \tag{6.7.16}$$

then

$$\mathbf{L} = (\boldsymbol{\Phi}(\mathbf{X}))^T\boldsymbol{\Phi}(\mathbf{X}) \in \mathbb{R}^{n\times n}, \tag{6.7.17}$$

$$\mathbf{K} = (\boldsymbol{\Phi}(\mathbf{X}_s))^T\boldsymbol{\Phi}(\mathbf{X}_s) \in \mathbb{R}^{n\times n}. \tag{6.7.18}$$

By Definition 6.12 and using $\mathrm{tr}(\mathbf{AB}) = \mathrm{tr}(\mathbf{BA})$ and $\mathbf{HH} = \mathbf{H}$, the kernel alignment between $\mathbf{L}_c$ and $\mathbf{K}_c$ can be represented as

$$\rho(\mathbf{L}_c, \mathbf{K}_c) = \mathrm{tr}(\mathbf{HLHHKH}) = \mathrm{tr}(\mathbf{HHLHHK}) = \mathrm{tr}(\mathbf{HLHK}). \tag{6.7.19}$$

The core issue with nonlinear joint unsupervised feature selection is to search for an optimal selection vector $\mathbf{s}$ so that the selected d high-quality features from D original features can be the solution to the following constrained optimization problem:

$$\mathbf{s} = \arg\min_{\mathbf{s}} \ \{f(\mathbf{K}) = -\mathrm{tr}(\mathbf{HLHK})\} \tag{6.7.20}$$

$$\text{subject to} \sum_{p=1}^{D} s_p = d, \quad s_p \in \{1, 0\},\ \forall\, p = 1, \ldots, D, \tag{6.7.21}$$

or

$$\mathbf{s} = \arg\min_{\mathbf{s}} \ \{f(\mathbf{K}) = -\mathrm{tr}(\mathbf{HLHK}) + \lambda\|\mathbf{s}\|_1\} \tag{6.7.22}$$

$$\text{subject to}\ s_p \in \{1, 0\},\ \forall\, p = 1, \ldots, D. \tag{6.7.23}$$

For Gaussian kernel $K_{ij} \sim N(0, \sigma^2)$, its gradient with respect to the selection vector $\mathbf{s}_p$ is given by

$$\frac{\partial K_{ij}}{\partial \mathbf{s}_p} = -K_{ij} \frac{2(x_{iP} - x_{jp})^2 \mathbf{s}_p}{\sigma^2}. \tag{6.7.24}$$

Hence, the gradient of the objective $f(\mathbf{K})$ with respect to the selection vector $\mathbf{s}_p$ is given by [267]

$$\begin{aligned} \frac{\partial f(\mathbf{K})}{\partial \mathbf{s}_p} &= -\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\mathbf{HLH}_{ij} \cdot \frac{\partial K_{ij}}{\partial \mathbf{s}_p}\right) + \lambda \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}\left(\left((\mathbf{HLH})_{ij} \odot \mathbf{K}\right)_{ij}(x_{ip} - x_{jp})^2\right)\frac{2\mathbf{s}_p}{\sigma^2} + \lambda. \end{aligned} \tag{6.7.25}$$

This problem can be solved by using spectral projected gradient (SPG), see Algorithm 6.9.

Algorithm 6.9 Spectral projected gradient (SPG) algorithm for nonlinear joint unsupervised feature selection [267]

```
1.  input: $\mathbf{s}_0 = \mathbf{1}$.
2.  initialization: Set step length $\alpha_0$, step length bound $\alpha_{\min} = 10^{-10}$, $\alpha_{\max} = 10^{10}$, history
    $h = 10, t = 0$.
3.  while not converged do
4.    $\alpha_t = \min\{\alpha_{\max}, \max(\alpha_{\min}, \alpha_0)\}$;
5.    $\mathbf{d}_t = \mathrm{Proj}_{[0,1]}(\mathbf{s}_t - \alpha_t \Delta f(\mathbf{s}_t)) - \mathbf{s}_t$;
6.    bound $f_b = \max\{f(\mathbf{s}_t), f(\mathbf{s}_{t-1}), \ldots, f(\mathbf{s}_{t-h})\}$.
7.    set $\alpha = 1$;
8.    if $f(\mathbf{s}_t + \alpha \mathbf{d}_t) > f_b + \eta\alpha(\Delta f(\mathbf{s}_t))^T \mathbf{d}_t$ then
9.      Choose $\alpha \in (0, \alpha)$;
10.   end if
11.   $\mathbf{s}_{t+1} = \mathbf{s}_t + \alpha \mathbf{d}_t$;
12.   $\mathbf{u}_t = \mathbf{s}_{t+1} - \mathbf{s}_t$;
13.   $\mathbf{y}_t = \Delta f(\mathbf{s}_{t+1}) - \Delta f(\mathbf{s}_t)$;
14.   $\alpha_0 = \mathbf{y}_t^T \mathbf{y}_t / \mathbf{u}_t^T \mathbf{y}_t$;
15.   $t = t + 1$;
16. end while
17. output: the selected features with corresponding entry in $\mathbf{s}$ equal to 1.
```

6.8 Principal Component Analysis

Principal component analysis (PCA) is a powerful data processing and dimension reduction technique for extracting structure from possibly high-dimensional data set [134]. PCA has several important variants or generalizations: robust principal component analysis, kernel principal component analysis, sparse principal component analysis, etc. The PCA technique is widely applied in engineering, biology, and social science, such as in face recognition, handwritten zip code classification, gene expression data analysis, etc.

6.8.1 Principal Component Analysis Basis

Given a high-dimensional data vector $\mathbf{x} = [x_1, \ldots, x_N]^T \in \mathbb{R}^N$, where N is large. We use an N-dimensional feature extraction vector $\mathbf{a}_i = [a_{i1}, \ldots, a_{iN}]^T$ to extract the ith feature of the data vector $\mathbf{x}$ as $\tilde{x}_i = \mathbf{a}_i^T \mathbf{x} = \sum_{j=1}^N a_{ij} x_j$. If we hope to use an $K \times N$ feature extract matrix $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_K] \in \mathbb{R}^{N\times K}$ for extracting the total K features of $\mathbf{x}$, then the K-dimensional feature vector $\tilde{\mathbf{x}} \in \mathbb{R}^K$ is given by

$$\tilde{\mathbf{x}} = \mathbf{A}^T\mathbf{x} = \begin{bmatrix} \mathbf{a}_1^T\mathbf{x} \\ \vdots \\ \mathbf{a}_K^T\mathbf{x} \end{bmatrix} = [\tilde{x}_1, \ldots, \tilde{x}_K]^T \in \mathbb{R}^K. \tag{6.8.1}$$

For a data matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_P] \in \mathbb{R}^{N\times P}$, its feature matrix $\tilde{\mathbf{X}}$ is given by

$$\tilde{\mathbf{X}} = \mathbf{A}^T\mathbf{X} = \mathbf{A}^T[\mathbf{x}_1, \ldots, \mathbf{x}_P] = \begin{bmatrix} \mathbf{a}_1^T\mathbf{x}_1 & \cdots & \mathbf{a}_1^T\mathbf{x}_P \\ \vdots & \ddots & \vdots \\ \mathbf{a}_K^T\mathbf{x}_1 & \cdots & \mathbf{a}_K^T\mathbf{x}_P \end{bmatrix} = [\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_P] \in \mathbb{R}^{K\times P}, \tag{6.8.2}$$

where

$$\tilde{\mathbf{x}}_i = \mathbf{A}^T\mathbf{x}_i = [\tilde{x}_{i1}, \ldots, \tilde{x}_{iK}]^T \in \mathbb{R}^K,\ i = 1, \ldots, P. \tag{6.8.3}$$

The feature vector $\tilde{\mathbf{x}}_i,\ i = 1, \ldots, P$ should satisfy the following requirements.

1. *Dimensionality reduction:* Due to correlation of data vectors, there is redundant information among the N components of each data vector $\mathbf{x}_i = [x_{i1}, \ldots, x_{iN}]^T$ for $i = 1, \ldots, P$. We hope to construct a new set of lower-dimensional feature vectors $\tilde{\mathbf{x}}_i = [\tilde{x}_{i1}, \ldots, \tilde{x}_{iK}]^T \in \mathbb{R}^K$, where $K \ll N$. Such a process is known as *feature extraction* or *dimension reduction*.
2. *Orthogonalization:* In order to avoid redundant information, P feature vectors should be orthogonal to each other, i.e., $\tilde{\mathbf{x}}_i^T\tilde{\mathbf{x}}_j = \delta_{ij}$ for all $i, j \in \{1, \ldots, P\}$.

3. *Power maximization:* Under the assumption of $P < N$, let $\sigma_1^2, \geq \cdots \geq \sigma_K^2$ be K leading eigenvalues of the $P \times P$ covariance matrix $\mathbf{X}^T\mathbf{X}$. Then we have

$$E_{\tilde{x}_i} = E\{|\tilde{x}_i|_2^2\} = \mathbf{v}_i^T(\mathbf{X}^T\mathbf{X})\mathbf{v}_i = \sigma_i^2,$$

where σ_i are the i leading singular values of $\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$ or $\lambda_i = \sigma_i^2$ are the ith leading eigenvalues of $\mathbf{X}^T\mathbf{X} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T$ with $\boldsymbol{\Sigma} = \mathbf{Diag}(\sigma_1, \ldots, \sigma_P)$ and $\boldsymbol{\Lambda} = \mathbf{Diag}(\lambda_1, \ldots, \lambda_P)$. Because the eigenvalues σ_i^2 are arranged in nondescending order and thus $E_{\tilde{\mathbf{x}}_1} \geq E_{\tilde{\mathbf{x}}_2} \geq \cdots \geq E_{\tilde{\mathbf{x}}_P}$. Therefore, $\tilde{\mathbf{x}}_1$ is often referred to as the first principal component feature of $\mathbf{X}$, $\tilde{\mathbf{x}}_2$ as the second principal component feature of $\mathbf{X}$, and so on. Hence, we have

$$E\{\tilde{\mathbf{x}}_1\} + \cdots + E\{\tilde{\mathbf{x}}_P\} = \sigma_1^2 + \cdots + \sigma_K^2 \approx E\left\{|\mathbf{x}_1|_2^2\right\} + \cdots + E\left\{|\mathbf{x}_P|_2^2\right\}. \qquad (6.8.4)$$

A direct choice satisfying the above three requirements is to take $\mathbf{A} = \mathbf{U}_1$ in (6.8.2), giving the principal component feature matrix

$$\tilde{\mathbf{X}}_1 = [\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_P] = \mathbf{U}_1^T\mathbf{X} = \boldsymbol{\Sigma}_1\mathbf{V}_1^T = \begin{bmatrix} \sigma_1\mathbf{v}_1^T \\ \vdots \\ \sigma_K\mathbf{v}_K^T \end{bmatrix} \in \mathbb{R}^{K\times P} \qquad (6.8.5)$$

because

$$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T = [\mathbf{U}_1, \mathbf{U}_2]\begin{bmatrix} \boldsymbol{\Sigma}_1 & \mathbf{O}_{K\times(P-K)} \\ \mathbf{O}_{(P-K)\times K} & \boldsymbol{\Sigma}_2 \end{bmatrix}\begin{bmatrix} \mathbf{V}_1^T \\ \mathbf{V}_2 \end{bmatrix} = \mathbf{U}_1\boldsymbol{\Sigma}_1\mathbf{V}_1^T + \mathbf{U}_2\boldsymbol{\Sigma}_2\mathbf{V}_2^T$$

with $\mathbf{U}_1 = [\mathbf{u}_1, \ldots, \mathbf{u}_K] \in \mathbb{R}^{N\times K}$, $\mathbf{U}_2 = [\mathbf{u}_{K+1}, \ldots, \mathbf{u}_P] \in \mathbb{R}^{N\times(P-K)}$, $\mathbf{V}_1 = [\mathbf{v}_1, \ldots, \mathbf{v}_K] \in \mathbb{R}^{P\times K}$, $\mathbf{V}_2 = [\mathbf{v}_{K+1}, \ldots, \mathbf{v}_P] \in \mathbb{R}^{P\times(P-K)}$, while $\boldsymbol{\Sigma}_1 = \mathbf{Diag}(\sigma_1, \ldots, \sigma_K)$ and $\boldsymbol{\Sigma}_2 = \mathbf{Diag}(\sigma_{K+1}, \ldots, \sigma_P)$.

The feature extraction method with the choice $\mathbf{A}_1 = \mathbf{U}_1$ is known as the *principal component analysis* (PCA). PCA has the following important properties:

- Principal components sequentially capture the maximum variability among the columns of $\mathbf{X}$, thus guaranteeing minimal information loss;
- Principal components are uncorrelated, so we can talk about one principal component without referring to others.

6.8.2 *Minor Component Analysis*

Definition 6.13 (Minor Components [169, 282]) Let $\mathbf{X}^T\mathbf{X}$ be the covariance matrix of an $N \times P$ data matrix $\mathbf{X}$ with K principal eigenvalues and $P - K$ minor eigenvalues (i.e., small eigenvalues). The $P - K$ eigenvectors $\mathbf{v}_{K+1}, \ldots, \mathbf{v}_P$ corresponding to these minor eigenvalues are called the *minor components* of the data matrix $\mathbf{X}$.

Data or signal analysis based on the $(P - K)$ minor components is known as *minor component analysis* (MCA). Unlike the PCA, the choice for the MCA is to take $\mathbf{A} = \mathbf{V}_2$ in (6.8.2), so that the $(P - K) \times P$ minor component feature matrix is determined by

$$\tilde{\mathbf{X}}_2 = \mathbf{U}_2\mathbf{X} = \mathbf{\Sigma}_2\mathbf{V}_2^T = \begin{bmatrix} \sigma_{K+1}\mathbf{v}_{K+1}^T \\ \vdots \\ \sigma_P\mathbf{v}_P^T \end{bmatrix} \in \mathbb{R}^{(P-K)\times P}. \tag{6.8.6}$$

Interestingly, the *contribution ratio* of the ith principal singular vector $\mathbf{v}_i$ in PCA is $\sigma_i/(\sigma_1^2 + \cdots + \sigma_K^2)^{1/2}, i = 1, \ldots, K$, while this ratio of the jth minor singular vector $\mathbf{v}_j$ in MCA is $\sigma_j/(\sigma_{K+1}^2 + \cdots + \sigma_P^2)^{1/2}, j = K + 1, \ldots, P$.

In signal/data processing, pattern recognition (such as face, fingerprints, irises recognition, etc.), image processing, principal components correspond to the key body of a signal or image background, since they describe the global energy of a signal or an image. In contrast, minor components correspond to details of the signal or image, which are usually dictated by the moving objects in the image background.

6.8.3 Principal Subspace Analysis

Principal subspace analysis (PSA) and *minor subspace analysis* (MSA), in which PCA and MCA are special cases, are important for many applications. The goal of PSA or MSA is to extract, from a stationary random process $\mathbf{x}(k) \in \mathbb{R}^N$ with covariance $\mathbf{C} = E\{\mathbf{x}(k)\mathbf{x}^T(k)\}$, the subspaces spanned by the m principal or minor eigenvectors, respectively [79].

For the matrix $\mathbf{X} \in \mathbb{R}^{N\times P}$ with K principal component vectors $\mathbf{V}_s = [\mathbf{v}_1, \ldots, \mathbf{v}_K]$, the *principal subspace* (also known as *signal subspace*) of $\mathbf{X}$ is defined as

$$\mathcal{S} = \text{span}(\mathbf{V}_s) = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_K\}, \tag{6.8.7}$$

and the orthogonal projector (i.e., *signal projection matrix*) is given by

$$\mathbf{P}_{\mathcal{S}} = \mathbf{V}_s(\mathbf{V}_s^T\mathbf{V}_s)^{-1}\mathbf{V}_s^T = \mathbf{V}_s\mathbf{V}_s^T, \tag{6.8.8}$$

where $\mathbf{V}_s = [\mathbf{v}_1, \ldots, \mathbf{v}_K]$ is a $P \times K$ semi-orthogonal matrix such that $\mathbf{V}_s^T\mathbf{V}_s = \mathbf{I}_{K\times K}$.

Similarly, the *minor subspace* (also known as *noise subspace*) of $\mathbf{X}$ is defined as

$$\mathcal{N} = \text{span}(\mathbf{V}_n) = \text{span}\{\mathbf{v}_{K+1}, \ldots, \mathbf{v}_P\}, \tag{6.8.9}$$

and the orthogonal projector (i.e., *noise projection matrix*) is given by

$$\mathbf{P}_{\mathcal{N}} = \mathbf{V}_n(\mathbf{V}_n^T\mathbf{V}_n)^{-1}\mathbf{V}_n^T = \mathbf{V}_n\mathbf{V}_n^T, \tag{6.8.10}$$

where $\mathbf{V}_n = [\mathbf{v}_{K+1}, \ldots, \mathbf{v}_P]$ is a $P \times (P - K)$ semi-orthogonal matrix such that $\mathbf{V}_n^T\mathbf{V}_n = \mathbf{I}_{(P-K)\times(P-K)}$.

It is noticed that the principal subspace and the minor subspace are orthogonal, i.e., $\mathcal{S} \perp \mathcal{N}$, because

$$\mathbf{P}_{\mathcal{S}}^T\mathbf{P}_{\mathcal{N}} = \mathbf{V}_s\mathbf{V}_s^T\mathbf{V}_n\mathbf{V}_n^T = \mathbf{O}_{P\times P} \quad \text{(null matrix)}.$$

Moreover, as

$$\mathbf{P}_{\mathcal{S}} + \mathbf{P}_{\mathcal{N}} = \mathbf{V}_s, \mathbf{V}_s^T + \mathbf{V}_n\mathbf{V}_n^T = [\mathbf{V}_s, \mathbf{V}_n]\begin{bmatrix}\mathbf{V}_s^T\\ \mathbf{V}_n^T\end{bmatrix} = \mathbf{V}_{P\times P}\mathbf{V}_{P\times P}^T = \mathbf{I}_{P\times P},$$

there is the following relationship between the principal and minor subspaces:

$$\mathbf{P}_{\mathcal{S}} = \mathbf{I} - \mathbf{P}_{\mathcal{N}} \quad \text{or} \quad \mathbf{V}_s\mathbf{V}_s = \mathbf{I} - \mathbf{V}_n\mathbf{V}_n^T. \tag{6.8.11}$$

Therefore, the PSA seeks to find the principal subspace $\mathbf{V}_s\mathbf{V}_s^T$ instead of the principal components $\mathbf{V}_s = [\mathbf{v}_1, \ldots, \mathbf{v}_K]$ in the PCA, while the MSA finds the minor subspace $\mathbf{V}_n\mathbf{V}_n^T$ instead of the minor components $\mathbf{V}_n = [\mathbf{v}_{K+1}, \ldots, \mathbf{v}_P]$ in the MCA.

In order to deduce the PSA and MSA updating algorithms, we consider the minimization of the objective function $J(\mathbf{W})$, where $\mathbf{W}$ is an $n \times r$ matrix. The common constraints on $\mathbf{W}$ are of two types, as follows.

- *Orthogonality constraint:* $\mathbf{W}$ is required to satisfy an orthogonality condition, either $\mathbf{W}^H\mathbf{W} = \mathbf{I}_r$ $(n \geq r)$ or $\mathbf{W}\mathbf{W}^H = \mathbf{I}_n$ $(n < r)$. The matrix $\mathbf{W}$ satisfying such a condition is called *semi-orthogonal matrix.*
- *Homogeneity constraint:* It is required that $J(\mathbf{W}) = J(\mathbf{WQ})$, where $\mathbf{Q}$ is an $r \times r$ orthogonal matrix.

For an unconstrained optimization problem $\min J(\mathbf{W}_{n\times r})$ with $n > r$, its solution is a single matrix $\mathbf{W}$. However, for an optimization problem subject to the semi-orthogonality constraint $\mathbf{W}^H\mathbf{W} = \mathbf{I}_r$ and the homogeneity constraint $J(\mathbf{W}) = J(\mathbf{WQ})$, where $\mathbf{Q}$ is an $r \times r$ orthogonal matrix, i.e.,

$$\min J(\mathbf{W}) \quad \text{subject to} \quad \mathbf{W}^H\mathbf{W} = \mathbf{I}_r,\ J(\mathbf{W}) = J(\mathbf{WQ}), \tag{6.8.12}$$

its solution is any member of the set of semi-orthogonal matrices defined by

$$Gr(n, r) = \left\{\mathbf{W} \in \mathbb{C}^{n\times r} \,\middle|\, \mathbf{W}^H\mathbf{W} = \mathbf{I}_r,\ \mathbf{W}_1\mathbf{W}_1^H = \mathbf{W}_2\mathbf{W}_2^H\right\}. \tag{6.8.13}$$

This set is known as the *Grassmann manifold* of the semi-orthogonal matrices $\mathbf{W}_{n\times r}$ such that $\mathbf{W}^T\mathbf{W} = \mathbf{I}$ but $\mathbf{W}\mathbf{W}^T$ is an $n \times n$ singular matrix.

For two given semi-orthogonal matrices $\mathbf{W}_1$ and $\mathbf{W}_2$, their subspaces are called *equivalent subspaces* if $\mathbf{W}_1\mathbf{W}_1^T = \mathbf{W_2}\mathbf{W}_2^T$. The feature extraction methods based on equivalent subspaces are *subspace analysis methods*.

Therefore, the goals of the PSA and MSA are to find the equivalent principal and minor subspaces, respectively.

Subspace analysis methods have the following characteristics [287].

- The signal (or principal) subspace method and the noise (or minor) subspace method need only a few singular vectors or eigenvectors.
- In many applications, it is not necessary to know singular values or eigenvalues; it is sufficient to know the rank of the matrix and its singular vectors or eigenvectors.
- In most cases, it is not necessary to know the singular vectors or eigenvectors exactly; it is sufficient to know the basis vectors spanning the principal subspace or minor subspace.
- Conversion between the principal subspace $\mathbf{V}_s\mathbf{V}_s^H$ and the minor subspace $\mathbf{V}_n\mathbf{V}_n^H$ can be made by using $\mathbf{V}_n\mathbf{V}_n^H = \mathbf{I} - \mathbf{V}_s\mathbf{V}_s^H$.

On the more basic theory and applications of subspace analysis, the reader can refer to [294].

In the following we deduce the PSA algorithm. Let $\mathbf{C} = E\{\mathbf{x}\mathbf{x}^T\}$ denote the covariance matrix of an $N \times 1$ random vector $\mathbf{x} \in \mathbb{R}^N$, and consider the following constrained optimization problem:

$$\min_{\mathbf{W}} E\left\{\|\mathbf{x} - \mathbf{P}_W\mathbf{x}\|_2^2\right\} \quad \text{subject to} \quad \mathbf{W}^T\mathbf{W} = \mathbf{I}, \tag{6.8.14}$$

where $\mathbf{P}_W = \mathbf{W}(\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T$ is the projector of the weighting matrix $\mathbf{W}_{N\times P}$ with $N > P$, while $\mathbf{P}_W\mathbf{x}$ denotes the feature vector of $\mathbf{x}$ extracted by $\mathbf{W}$. Clearly, $\mathbf{x} - \mathbf{P}_W\mathbf{x}$ denotes the error between the original input data vector and its extracted feature vector, and thus (6.8.14) is a minimum squares error (MSE) criterion.

Interestingly, the constrained optimization problem (6.8.14) can be equivalently written as the following unconstrained optimization problem:

$$\min_{\mathbf{W}} \quad E\left\{\left\|\mathbf{x} - \mathbf{W}\mathbf{W}^T\mathbf{x}\right\|_2^2\right\} \tag{6.8.15}$$

whose objective function is given by [287]:

$$\begin{aligned} J(\mathbf{W}) &= E\left\{\left\|\mathbf{x} - \mathbf{W}\mathbf{W}^T\mathbf{x}\right\|_2^2\right\} \\ &= E\left\{\left(\mathbf{x} - \mathbf{W}\mathbf{W}^T\mathbf{x}\right)^T\left(\mathbf{x} - \mathbf{W}\mathbf{W}^T\mathbf{x}\right)\right\} \\ &= E\left\{\mathbf{x}^T\mathbf{x}\right\} - 2E\left\{\mathbf{x}^T\mathbf{W}\mathbf{W}^T\mathbf{x}\right\} + E\left\{\mathbf{x}^T\mathbf{W}\mathbf{W}^T\mathbf{W}\mathbf{W}^T\mathbf{x}\right\}. \end{aligned} \tag{6.8.16}$$

Because

$$E\{\mathbf{x}^T\mathbf{x}\} = \sum_{i=1}^{n} E\{|x_i|^2\} = \mathrm{tr}\left(E\{\mathbf{x}\mathbf{x}^T\}\right) = \mathrm{tr}(\mathbf{C}),$$

$$E\{\mathbf{x}^T\mathbf{W}\mathbf{W}^T\mathbf{x}\} = \mathrm{tr}\Big(E\{\mathbf{W}^T\mathbf{x}\mathbf{x}^T\mathbf{W}\}\Big) = \mathrm{tr}\big(\mathbf{W}^T\mathbf{C}\mathbf{W}\big),$$

$$\begin{aligned} E\{\mathbf{x}^T\mathbf{W}\mathbf{W}^T\mathbf{W}\mathbf{W}^T\mathbf{x}\} &= \mathrm{tr}\left(E\{\mathbf{W}^T\mathbf{x}\mathbf{x}^T\mathbf{W}\mathbf{W}^T\mathbf{W}\}\right) \\ &= \mathrm{tr}\big(\mathbf{W}^T\mathbf{C}\mathbf{W}\mathbf{W}^T\mathbf{W}\big), \end{aligned}$$

the objective function in (6.8.16) can be represented as the trace form

$$J(\mathbf{W}) = \mathrm{tr}(\mathbf{C}) - 2\mathrm{tr}(\mathbf{W}^T\mathbf{C}\mathbf{W}) + \mathrm{tr}(\mathbf{W}^T\mathbf{C}\mathbf{W}\mathbf{W}^T\mathbf{W}), \tag{6.8.17}$$

where $\mathbf{W}$ is an $N \times P$ matrix whose rank is assumed to be K.

From Eq. (6.8.17) it can be seen that in the time-varying case the matrix differential of the objective function $J(\mathbf{W}(t))$ is given by

$$\begin{aligned} \mathrm{d}J(\mathbf{W}(t)) = &-2\mathrm{tr}\Big(\mathbf{W}^T(t)\mathbf{C}(t)\mathrm{d}\mathbf{W}(t) + (\mathbf{C}(t)\mathbf{W}(t))^T\mathrm{d}\mathbf{W}^*(t)\Big) \\ &+ \mathrm{tr}\Big((\mathbf{W}^T(t)\mathbf{W}(t)\mathbf{W}^T(t)\mathbf{C}(t) + \mathbf{W}(t)\mathbf{C}(t)\mathbf{W}(t)\mathbf{W}^T(t))\mathrm{d}\mathbf{W}(t) \\ &+ (\mathbf{C}(t)\mathbf{W}(t)\mathbf{W}^T(t)\mathbf{W}(t) + \mathbf{W}(t)\mathbf{W}^T(t)\mathbf{C}(t)\mathbf{W}(t))^T\,\mathrm{d}\mathbf{W}^*(t)\Big), \end{aligned}$$

which yields the gradient matrix

$$\begin{aligned} \nabla_W J(\mathbf{W}(t)) &= -2\mathbf{C}(t)\mathbf{W}(t) + \mathbf{C}(t)\mathbf{W}(t)\mathbf{W}^T(t)\mathbf{W}(t) + \mathbf{W}(t)\mathbf{W}^T(t)\mathbf{C}(t)\mathbf{W}(t) \\ &= \mathbf{W}(t)\mathbf{W}^T(t)\mathbf{C}(t)\mathbf{W}(t) - \mathbf{C}(t)\mathbf{W}(t), \end{aligned}$$

where the semi-orthogonality constraint $\mathbf{W}^T(t)\mathbf{W}(t) = \mathbf{I}$ has been used.

Substituting $\mathbf{C}(t) = \mathbf{x}(t)\mathbf{x}^T(t)$ into the above gradient matrix formula, one obtains a gradient descent algorithm for solving the minimization problem $\mathbf{W}(t) = \mathbf{W}(t-1) - \mu\nabla_W J(\mathbf{W}(t))$, as follows:

$$\mathbf{y}(t) = \mathbf{W}^T(t)\mathbf{x}(t), \tag{6.8.18}$$

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \beta(t)\big(\mathbf{y}^T(t)\mathbf{x}(t) - \mathbf{W}(t)\mathbf{y}(t)\mathbf{y}^T(t)\big). \tag{6.8.19}$$

Here, $\beta(t) > 0$ is a learning parameter. This algorithm is called the *projection approximation subspace tracking* (PAST) that was developed by Yang in 1995 [287] from the viewpoint of subspace.

On the PAST algorithm, the following two theorems are true.

Theorem 6.1 ([287]) *The matrix* $\mathbf{W}$ *is a stationary point of* $J(\mathbf{W})$ *if and only if* $\mathbf{W} = \mathbf{V}_r\mathbf{Q}$*, where* $\mathbf{V}_r \in \mathbb{R}^{P\times r}$ *consists of r eigenvectors of the covariance matrix* $\mathbf{C} = \mathbf{V}_r\mathbf{D}_r\mathbf{V}_r^T$ *and* $\mathbf{Q} \in \mathbb{R}^{r\times r}$ *is any orthogonal matrix. At every stationary point, the value of the objective function* $J(\mathbf{W})$ *is equal to the sum of the eigenvalues whose corresponding eigenvectors are not in* $\mathbf{V}_r$*.*

Theorem 6.2 ([287]) *All its stationary points are saddle points of the objective function* $J(\mathbf{W})$ *unless* $r = K$ *(rank of* $\mathbf{C}$*), i.e.,* $\mathbf{V}_r = \mathbf{V}_s$ *consists of K principal eigenvectors of the covariance matrix* $\mathbf{C}$*. In this particular case* $J(\mathbf{W})$ *reaches a global minimum.*

Theorems 6.1 and 6.2 show the following facts.

1. In the unconstrained minimization of the objective function $J(\mathbf{W})$ in (6.8.17), it is not required that the columns of $\mathbf{W}$ are orthogonal, but the two theorems show that minimization of the objective function $J(\mathbf{W})$ in (6.8.17) will automatically result in a semi-orthogonal matrix $\mathbf{W}$ such that $\mathbf{W}^T\mathbf{W} = \mathbf{I}$.
2. Theorem 6.2 shows that when the column space of $\mathbf{W}$ is equal to the signal or principal subspace: $\mathrm{Col}(\mathbf{W}) = \mathrm{Span}(\mathbf{V}_s)$, the objective function $J(\mathbf{W})$ reaches a global minimum, and it has no other local minimum.
3. From the definition formula (6.8.17) it is easy to see that $J(\mathbf{W}) = J(\mathbf{WQ})$ holds for all $K \times K$ unitary matrices $\mathbf{Q}$, i.e., the objective function automatically satisfies the homogeneity constraint.
4. Because the objective function defined by (6.8.17) automatically satisfies the homogeneity constraint, and its minimization automatically has the result that $\mathbf{W}$ satisfies the orthogonality constraint $\mathbf{W}^H\mathbf{W} = \mathbf{I}$, the solution $\mathbf{W}$ is not uniquely determined but is a point on the Grassmann manifold. This greatly relaxes the solution for the optimization problem (6.8.17).
5. The projection matrix $\mathbf{P} = \mathbf{W}(\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T = \mathbf{W}\mathbf{W}^T = \mathbf{V}_s\mathbf{V}_s^T$ is uniquely determined. That is to say, the different solutions $\mathbf{W}$ span the same column space.
6. When $r = 1$, i.e., the objective function is a real function of a vector $\mathbf{w}$, the solution $\mathbf{w}$ of the minimization of $J(\mathbf{w})$ is the eigenvector corresponding to the largest eigenvalue of the covariance matrix $\mathbf{C}$.

Therefore, Eq. (6.8.19) gives the global solution to the optimization problem (6.8.16). It is worth emphasizing that (6.8.19) is just the PSA rule of Oja [194].

On the other hand, the two MSA algorithms are [49]

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \beta(t)\left(\mathbf{W}(t)\mathbf{W}^T(t)\mathbf{y}(t)\mathbf{x}(t) - \mathbf{y}(t)\mathbf{y}^T(t)\mathbf{W}(t)\right), \tag{6.8.20}$$

and [79]

$$\begin{aligned}\mathbf{W}(t+1) = \mathbf{W}(t) - \beta(t)\Big(&\mathbf{W}(t)\mathbf{W}^T(t)\mathbf{W}(t)\mathbf{W}^T(t)\mathbf{y}(t)\mathbf{x}^T(t)\\ &- \mathbf{y}(t)\mathbf{y}^T(t)\mathbf{W}(t)\Big).\end{aligned} \tag{6.8.21}$$

The Douglas algorithm (6.8.21) is a self-stabilizing in that the vectors do not need to be periodically normalized to unit modulus.

6.8.4 Robust Principal Component Analysis

Given an observation or data matrix $\mathbf{D} = \mathbf{A} + \mathbf{E}$, where $\mathbf{A}$ and $\mathbf{E}$ are unknown, but $\mathbf{A}$ is known to be a low- rank true data matrix and $\mathbf{E}$ is known to be an additive sparse noise matrix, the aim is to recover $\mathbf{A}$. Because the low-rank matrix $\mathbf{A}$ can be regarded as the principal component of the data matrix $\mathbf{D}$, and the sparse matrix $\mathbf{E}$ may have a few gross errors or outlying observations, the SVD-based principal component analysis (PCA) usually breaks down. Hence, this problem is a *robust principal component analysis* problem [164, 272].

So-called robust PCA is able to correctly recover underlying low-rank structure in the data matrix, even in the presence of gross errors or outlying observations. This problem is also called the *principal component pursuit* (PCP) problem [46] since it tracks down the principal components of the data matrix $\mathbf{D}$ by minimizing a combination of the nuclear norm $\|\mathbf{A}\|_*$ and the weighted ℓ_1-norm $\mu\|\mathbf{E}\|_1$:

$$\min_{\mathbf{A},\mathbf{E}} \left\{\|\mathbf{A}\|_* + \mu\|\mathbf{E}\|_1\right\} \quad \text{subject to} \quad \mathbf{D} = \mathbf{A} + \mathbf{E}. \tag{6.8.22}$$

Here, $\|\mathbf{A}\|_* = \sum_i^{\min\{m,n\}} \sigma_i(\mathbf{A})$ represents the nuclear norm of $\mathbf{A}$, i.e., the sum of all singular values, which reflects the cost of the low-rank matrix $\mathbf{A}$, while $\|\mathbf{E}\|_1 = \sum_{i=1}^m \sum_{j=1}^n |E_{ij}|$ is the sum of the absolute values of all entries of the additive error matrix $\mathbf{E}$, where some of its entries may be arbitrarily large. The role of the constant $\mu > 0$ is to balance the contradictory requirements of low rank and sparsity.

By using the minimization in (6.8.22), an initially unknown low-rank matrix $\mathbf{A}$ and unknown sparse matrix $\mathbf{E}$ can be recovered from the data matrix $\mathbf{D} \in \mathbb{R}^{m\times n}$. That is to say, the unconstrained minimization of the robust PCA or the PCP problem can be expressed as

$$\min_{\mathbf{A},\mathbf{E}} \left\{\|\mathbf{A}\|_* + \mu\|\mathbf{E}\|_1 + \frac{1}{2}(\|\mathbf{A} + \mathbf{E} - \mathbf{D})\|_F^2\right\}. \tag{6.8.23}$$

It can be shown [44] that, under rather weak assumptions, the PCP estimate can exactly recover the low-rank matrix $\mathbf{A}$ from the data matrix $\mathbf{D} = \mathbf{A} + \mathbf{E}$ with gross but sparse error matrix $\mathbf{E}$.

In order to solve the robust PCA or the PCP problem (6.8.23), consider a more general family of optimization problems of the form

$$F(\mathbf{X}) = f(\mathbf{X}) + \mu h(\mathbf{X}), \tag{6.8.24}$$

where $f(\mathbf{X})$ is a convex, smooth (i.e., differentiable), and L-Lipschitz function and $h(\mathbf{X})$ is a convex but nonsmooth function (such as $\|\mathbf{X}\|_1$, $\|\mathbf{X}\|_*$, and so on).

Instead of directly minimizing the composite function $F(\mathbf{X})$, we minimize its separable quadratic approximation $Q(\mathbf{X}, \mathbf{Y})$, formed at specially chosen points $\mathbf{Y}$:

$$\begin{aligned} Q(\mathbf{X},\mathbf{Y}) &= f(\mathbf{Y}) + \langle \nabla f(\mathbf{Y}), \mathbf{X}-\mathbf{Y}\rangle + \frac{1}{2}(\mathbf{X}-\mathbf{Y})^T\nabla^2 f(\mathbf{Y})(\mathbf{X}-\mathbf{Y}) + \mu h(\mathbf{X}) \\ &= f(\mathbf{Y}) + \langle \nabla f(\mathbf{Y}), \mathbf{X}-\mathbf{Y}\rangle + \frac{L}{2}\|\mathbf{X}-\mathbf{Y}\|_F^2 + \mu h(\mathbf{X}), \end{aligned} \tag{6.8.25}$$

where $\nabla^2 f(\mathbf{Y})$ is approximated by $L\mathbf{I}$.

When minimizing $Q(\mathbf{X}, \mathbf{Y})$ with respect to $\mathbf{X}$, the function term $f(\mathbf{Y})$ may be regarded as a constant term that is negligible. Hence, we have

$$\begin{aligned} \mathbf{X}_{k+1} &= \arg\min_{\mathbf{X}} \left\{ \mu h(\mathbf{X}) + \frac{L}{2}\left\| \mathbf{X} - \mathbf{Y}_k + \frac{1}{L}\nabla f(\mathbf{Y}_k)\right\|_F^2 \right\} \\ &= \mathbf{prox}_{\mu L^{-1}h}\left(\mathbf{Y}_k - \frac{1}{L}\nabla f(\mathbf{Y}_k)\right). \end{aligned} \tag{6.8.26}$$

If we let $f(\mathbf{Y}) = \frac{1}{2}\|\mathbf{A}+\mathbf{E}-\mathbf{D}\|_F^2$ and $h(\mathbf{X}) = \|\mathbf{A}\|_* + \lambda\|\mathbf{E}\|_1$, then the Lipschitz constant $L = 2$ and $\nabla f(\mathbf{Y}) = \mathbf{A}+\mathbf{E}-\mathbf{D}$. Hence

$$Q(\mathbf{X},\mathbf{Y}) = \mu h(\mathbf{X}) + f(\mathbf{Y}) = (\mu\|\mathbf{A}\|_* + \mu\lambda\|\mathbf{E}\|_1) + \frac{1}{2}\|\mathbf{A}+\mathbf{E}-\mathbf{D}\|_F^2$$

reduces to the quadratic approximation of the robust PCA problem.

By Theorem 4.5, we have

$$\begin{aligned} \mathbf{A}_{k+1} &= \mathbf{prox}_{\mu/2\|\cdot\|_*}\left(\mathbf{Y}_k^A - \frac{1}{2}(\mathbf{A}_k + \mathbf{E}_k - \mathbf{D})\right) = \mathbf{U}\text{soft}(\mathbf{\Sigma}, \lambda)\mathbf{V}^T, \\ \mathbf{E}_{k+1} &= \mathbf{prox}_{\mu\lambda/2\|\cdot\|_1}\left(\mathbf{Y}_k^E - \frac{1}{2}(\mathbf{A}_k + \mathbf{E}_k - \mathbf{D})\right) = \text{soft}\left(\mathbf{W}_k^E, \frac{\mu\lambda}{2}\right), \end{aligned}$$

where $\mathbf{U\Sigma V}^T$ is the SVD of $\mathbf{W}_k^A$ and

$$\mathbf{W}_k^A = \mathbf{Y}_k^A - \frac{1}{2}(\mathbf{A}_k + \mathbf{E}_k - \mathbf{D}), \tag{6.8.27}$$

$$\mathbf{W}_k^E = \mathbf{Y}_k^E - \frac{1}{2}(\mathbf{A}_k + \mathbf{E}_k - \mathbf{D}). \tag{6.8.28}$$

The robust PCA shown in Algorithm 6.10 has the following convergence.

Algorithm 6.10 Robust PCA via accelerated proximal gradient [164, 272]

```
1. input: Data matrix $\mathbf{D} \in \mathbb{R}^{m\times n}$, $\lambda$, allowed tolerance $\epsilon$.
2. initialization: $\mathbf{A}_0, \mathbf{A}_{-1} \leftarrow \mathbf{O}$; $\mathbf{E}_0, \mathbf{E}_{-1} \leftarrow \mathbf{O}$; $t_0, t_{-1} \leftarrow 1$; $\bar{\mu} \leftarrow \delta\mu_0$.
3. repeat
4.    $\mathbf{Y}_k^A \leftarrow \mathbf{A}_k + \frac{t_{k-1}-1}{t_k}(\mathbf{A}_k - \mathbf{A}_{k-1})$.
5.    $\mathbf{Y}_k^E \leftarrow \mathbf{E}_k + \frac{t_{k-1}-1}{t_k}(\mathbf{E}_k - \mathbf{E}_{k-1})$.
6.    $\mathbf{W}_k^A \leftarrow \mathbf{Y}_k^A - \frac{1}{2}(\mathbf{A}_k + \mathbf{E}_k - \mathbf{D})$.
7.    $(\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}) = \mathrm{svd}(\mathbf{W}_k^A)$.
8.    $r = \max\left\{j : \sigma_j > \frac{\mu_k}{2}\right\}$.
9.    $\mathbf{A}_{k+1} = \sum_{i=1}^r \left(\sigma_i - \frac{\mu_k}{2}\mathbf{u}_i\mathbf{v}_i^T\right)$.
10.   $\mathbf{W}_k^E \leftarrow \mathbf{Y}_k^E - \frac{1}{2}(\mathbf{A}_k + \mathbf{E}_k - \mathbf{D})$.
11.   $\mathbf{E}_{k+1} = \mathrm{soft}\left(\mathbf{W}_k^E, \frac{\lambda\mu_k}{2}\right)$.
12.   $t_{k+1} \leftarrow \frac{1+\sqrt{4t_k^2+1}}{2}$.
13.   $\mu_{k+1} \leftarrow \max(\eta\mu_k, \bar{\mu})$.
14.   $\mathbf{S}_{k+1}^A = 2(\mathbf{Y}_k^A - \mathbf{A}_{k+1}) + (\mathbf{A}_{k+1} + \mathbf{E}_{k+1} - \mathbf{Y}_k^A - \mathbf{Y}_k^E)$.
15.   $\mathbf{S}_{k+1}^E = 2(\mathbf{Y}_k^E - \mathbf{E}_{k+1}) + (\mathbf{A}_{k+1} + \mathbf{E}_{k+1} - \mathbf{Y}_k^A - \mathbf{Y}_k^E)$.
16.   exit if $\|\mathbf{S}_{k+1}\|_F^2 = \|\mathbf{S}_{k+1}^A\|_F^2 + \|\mathbf{S}_{k+1}^E\|_F^2 \le \epsilon$.
17. return $k \leftarrow k+1$.
18. output: $\mathbf{A} \leftarrow \mathbf{A}_k$, $\mathbf{E} \leftarrow \mathbf{E}_k$.
```

Theorem 6.3 ([164]) *Let $F(\mathbf{X}) = F(\mathbf{A}, \mathbf{E}) = \mu\|\mathbf{A}\|_* + \mu\lambda\|\mathbf{E}\|_1 + \frac{1}{2}\|\mathbf{A}+\mathbf{E}-\mathbf{D}\|_F^2$. Then, for all $k > k_0 = C_1/\log(1/n)$ with $C_1 = \log(\mu_0/\mu)$, one has*

$$F(\mathbf{X}) - F(\mathbf{X}^*) \le \frac{4\|\mathbf{X}_{k_0} - \mathbf{X}^*\|_F^2}{(k-k_0+1)^2}, \tag{6.8.29}$$

where $\mathbf{X}^*$ is a solution to the robust PCA problem $\min_{\mathbf{X}} F(\mathbf{X})$.

6.8.5 Sparse Principal Component Analysis

Consider a *linear regression* problem: given an observed response vector $\mathbf{y} = [y_1, \ldots, y_n]^T \in \mathbb{R}^n$ and an observed data matrix (or predictors) $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_p] \in \mathbb{R}^{n\times p}$, find a fitting coefficient vector $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_p]^T \in \mathbb{R}^p$ such that

$$\hat{\beta} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 = \left\|\mathbf{y} - \sum_{i=1}^p \mathbf{x}_i\beta_i\right\|_2^2, \tag{6.8.30}$$

where $\mathbf{y}$ and $\mathbf{x}_i, i = 1, \ldots, p$ are assumed to be centered, respectively.

Minimizing the above squared error usually leads to sensitive solutions. Many regularization methods have been proposed to decrease this sensitivity. Among them, Tikhonov regularization [247] and Lasso [84, 244] are two widely known and cited algorithms, as pointed out in [283].

The Lasso (*least absolute shrinkage and selection operator*) is a penalized least squares method:

$$\hat{\beta}_{\text{Lasso}} = \arg\min_{\boldsymbol{\beta}} \left\| \mathbf{y} - \sum_{i=1}^{p} \mathbf{x}_i \beta_i \right\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1, \tag{6.8.31}$$

where $\|\boldsymbol{\beta}\|_1 = \sum_{i=1}^{p} |\beta_i|$ is the ℓ_1-norm and λ is nonnegative.

The Lasso can provide a sparse prediction coefficient solution vector due to the nature of the ℓ_1-norm penalty through continuously shrinking the prediction coefficients toward zero, and achieving its prediction accuracy via the bias variance trade-off.

Although the Lasso has shown success in many situations, it has some limitations [307].

- If $p > n$, then the Lasso selects at most n variables before it saturates due to the nature of the convex optimization problem. This seems to be a limiting feature for a variable selection method. Moreover, the Lasso is not well defined unless the bound on the ℓ_1-norm of $\boldsymbol{\beta}$ is smaller than a certain value.
- If there is a group of variables among which the pairwise correlations are very high, then the Lasso tends to select only one variable from the group and does not care which one is selected.
- For usual $n > p$ cases, if there are high correlations between predictors $\mathbf{x}_i$, it has been empirically observed that the prediction performance of the Lasso is dominated by ridge regression [244].

To overcome these drawbacks, an *elastic net* was proposed by Zou and Hastie [307] through generalizing the Lasso:

$$\hat{\beta}_{\text{en}} = (1 + \beta_2) \left(\arg\min_{\boldsymbol{\beta}} \left\| \mathbf{y} - \sum_{i=1}^{p} \mathbf{x}_i \beta_i \right\|_2^2 + \lambda_1 \sum_{i=1}^{p} |\beta_i| + \lambda_2 \sum_{i=1}^{p} |\lambda_i|^2 \right) \tag{6.8.32}$$

for any nonnegative λ_1 and λ_2. The elastic net penalty is a convex combination of the ridge and Lasso penalties. If $\lambda_2 = 0$, then the elastic net reduces to the Lasso.

Theorem 6.4 ([308]) *Let* $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$ *be the rank-*K *SVD of* $\mathbf{X}$. *For each* i, *denote by* $\mathbf{z}_i = d_{ii}\mathbf{u}_i$ *the* i*th principal component. Consider a positive* λ *and the ridge* *estimates* $\hat{\boldsymbol{\beta}}_{\text{ridge}}$ *given by*

$$\hat{\boldsymbol{\beta}}_{\text{ridge}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{z}_i - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_2^2. \tag{6.8.33}$$

Letting $\hat{\mathbf{v}} = \frac{\hat{\boldsymbol{\beta}}_{\text{ridge}}}{\|\hat{\boldsymbol{\beta}}_{\text{ridge}}\|}$, *then* $\hat{\mathbf{v}} = \mathbf{v}_i$.

This theorem establishes the connection between PCA and a regression method.

Theorem 6.5 ([308]) *Let* $\mathbf{A}_{p\times k} = [\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_k]$ *and* $\mathbf{B}_{p\times k} = [\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_k]$. *For any* $\lambda > 0$, *consider the constrained optimization problem*

$$(\hat{\mathbf{A}}, \hat{\mathbf{B}}) = \arg\min_{\mathbf{A},\mathbf{B}} \left\|\mathbf{X} - \mathbf{X}\mathbf{B}\mathbf{A}^T\right\|_F^2 + \lambda \sum_{j=1}^{k} \|\boldsymbol{\beta}_j\|_2^2 \tag{6.8.34}$$

subject to $\mathbf{A}^T\mathbf{A} = \mathbf{I}_{k\times k}$. *Then,* $\hat{\boldsymbol{\beta}}_j \propto \mathbf{v}_j$ *for* $j = 1, \ldots, k$.

If adding the Lasso penalty, then the criterion (6.8.34) becomes the following optimization problem:

$$(\hat{\mathbf{A}}, \hat{\mathbf{B}}) = \arg\min_{\mathbf{A},\mathbf{B}} \left\|\mathbf{X} - \mathbf{X}\mathbf{A}\mathbf{B}^T\right\|_F^2 + \lambda \sum_{j=1}^{k} \|\boldsymbol{\beta}_j\|_2^2 + \sum_{j=1}^{k} \lambda_{1,j} \|\boldsymbol{\beta}_j\|_1 \tag{6.8.35}$$

subject to $\mathbf{A}^T\mathbf{A} = \mathbf{I}_{k\times k}$. Due to $\|\mathbf{A}\|_F^2 = \|\mathbf{A}^T\mathbf{A}\|_2 = 1$, one has

$$\|\mathbf{X} - \mathbf{X}\mathbf{B}\mathbf{A}^T\|_F^2 = \|\mathbf{X}\mathbf{A} - \mathbf{X}\mathbf{B}\|_F^2 = (\mathbf{A} - \mathbf{B})^T\mathbf{X}^T\mathbf{X}(\mathbf{A} - \mathbf{B}).$$

Substitute this result into (6.8.35) to get

$$(\hat{\boldsymbol{\alpha}}_j, \hat{\boldsymbol{\beta}}_j) = \arg\min_{\boldsymbol{\alpha}_j,\boldsymbol{\beta}_j} (\boldsymbol{\alpha}_j - \boldsymbol{\beta}_j)^T\mathbf{X}^T\mathbf{X}(\boldsymbol{\alpha}_j - \boldsymbol{\beta}_j) + \lambda\|\boldsymbol{\beta}_j\|_2^2 + \lambda_{1,j}\|\boldsymbol{\beta}_j\|_1. \tag{6.8.36}$$

This coupled optimization problem can be solved by using the well-known alternating method.

By summarizing the above discussion, the *sparse principal component analysis (SPCA) algorithm* developed by Zou et al. [308] can be described by Algorithm 6.11.

Algorithm 6.11 Sparse principal component analysis (SPCA) algorithm [308]

```
1. input: The data matrix $\mathbf{X} \in \mathbb{R}^{n\times p}$ and the response vector $\mathbf{y} \in \mathbb{R}^n$.
2. initialization:
   2.1 Make the truncated SVD $\mathbf{X} = \sum_{i=1}^{k} d_i \mathbf{u}_i \mathbf{v}_i^T$.
   2.2 Let $\boldsymbol{\alpha}_j = \mathbf{v}_j,\ j = 1,\ldots,k$.
   2.3 Select $\lambda > 0$ and $\lambda_{1,i} > 0,\ i = 1,\ldots,k$.
3. Given a fixed $\mathbf{A} = [\boldsymbol{\alpha}_1,\ldots,\boldsymbol{\alpha}_k]$, solve the following elastic net problem for $j = 1,\ldots,k$:
       $\boldsymbol{\beta}_j = \arg\min_{\boldsymbol{\beta}}\ (\boldsymbol{\alpha}_j - \boldsymbol{\beta})^T\mathbf{X}^T\mathbf{X}(\boldsymbol{\alpha}_j - \boldsymbol{\beta}) + \lambda\|\boldsymbol{\beta}\|_2^2 + \lambda_{1,j}\|\boldsymbol{\beta}\|_1$.
4. For a fixed $\mathbf{B} = [\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_k]$, compute the SVD of $\mathbf{X}^T\mathbf{X}\mathbf{B} = \mathbf{U}\mathbf{D}\mathbf{V}^T$, then update $\mathbf{A} = \mathbf{U}\mathbf{V}^T$.
5. Repeat Steps 3-4, until convergence.
6. Normalization: $\boldsymbol{\beta}_j \leftarrow \frac{\boldsymbol{\beta}_j}{\|\boldsymbol{\beta}_j\|_2}$ for $j = 1,\ldots,k$.
7. output: normalized fitting coefficient vectors $\boldsymbol{\beta}_j,\ j = 1,\ldots,k$.
```

6.9 Supervised Learning Regression

Supervised learning in the form of regression (for continuous outputs) and classification (for discrete outputs) is an important field of statistics and machine learning.

The aim of machine learning is to establish a mapping relationship between inputs and outputs via learning the correspondence between the sample and the label. As the name implies, supervised learning is based on a supervisor (data labeling expert): a type of machine learning of input (or training) data with labels.

In supervised learning, we are given a "training" set of input vectors $\{\mathbf{x}_n\}_{n=1}^N$ along with corresponding targets $\{y_n\}_{n=1}^N$. The data set $\{\mathbf{x}_n, y_n\}_{n=1}^N$ is known as the labeled data set.

Let the domain of input instances be X, and the domain of labels be Y, and let $P(x, y)$ be an (unknown) joint probability distribution on instances and labels $X \times Y$. Given an independent identically distributed (i.i.d.) training data $\{\mathbf{x}_i\}_{i=1}^N$ sampled from $P(\mathbf{x})$, denoted $\mathbf{x}_i \overset{\text{i.i.d.}}{\sim} P(\mathbf{x})$, supervised learning trains a function $f : X \to Y$ in some function family F in order to make $f(\mathbf{x})$ predict the true label y on future data $\mathbf{x}$, where $\mathbf{x} \overset{\text{i.i.d.}}{\sim} P(\mathbf{x})$ as well.

Depending on the target y, supervised learning problems are further divided into two of the most common supervised learning forms: classification and regression.

- *Classification* is the supervised learning problem with discrete class labels y, for example, $y = \{+1, -1\}$ for two classes. The function f is called a classifier.
- *Regression* is the supervised learning problem with real values y. The function f is called a regression function.

From this training set we wish to learn a model of the dependency of the targets on the inputs with the objective of making accurate predictions of y for previously unseen values of $\mathbf{x}$.

Goals in model selection include [116]:

- accurate predictions,
- *interpretable models:* determining which predictors are meaningful,

- *stability:* small changes in the data should not result in large changes in either the subset of predictors used, the associated coefficients, or the predictions,
- avoiding bias in hypothesis tests during or after variable selection.

6.9.1 Principle Component Regression

A task to approximate real-valued functions from a training sample is variously referred to as prediction, regression, interpolation, or function approximation.

In statistical modeling, regression analysis is a set of statistical processes for estimating the relationships between a dependent variable (called also "criterion variable") X and one or more independent variables (or predictors) Y. More specifically, regression analysis helps us understand how the typical value of the dependent variable changes when any one of the independent variables is varied, while the other independent variables are held fixed.

Most commonly, regression analysis estimates the conditional expectation of the dependent variable given the independent variables.

In all cases, a function of the independent variables called the regression function is to be estimated.

Given a training set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ with $\mathbf{x}_i = [x_{i1}, \ldots, x_{ip}]^T$ for all $i \in [n] \overset{\text{def}}{=} \{1, \cdots, n\}$. Denote $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$, and consider the supervised learning problem: to learn a function $f(\mathbf{X}) : \mathbf{X} \to \mathbf{y}$ so that $f(\mathbf{X})$ is a good predictor for the corresponding value of $\mathbf{y}$, namely minimize$\frac{1}{2}\|\mathbf{y} - f(\mathbf{X})\|_2^2$. For historical reasons, this function f is called a *hypothesis*.

The regression models involve the following parameters and variables:

- The *unknown parameters*, denoted as $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_p]^T$, which is usually a vector.
- The independent variable matrix $\mathbf{X} = [x_{ij}]_{i=1,j=2}^{n,p}$.
- The dependent variable vector, denoted as $\mathbf{y} = [y_1, \ldots, y_n]^T$.

A regression model uses the function of $\mathbf{X}$ and $\boldsymbol{\beta}$

$$\mathbf{y} \approx f(\mathbf{X}, \boldsymbol{\beta}) \tag{6.9.1}$$

as an approximation of $E(\mathbf{y}|\mathbf{X}) = f(\mathbf{X}, \boldsymbol{\beta})$.

Standard Linear Regression Method
Consider the more general multiple regression

$$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i. \tag{6.9.2}$$

As data pre-processing, it is usually assumed that $\mathbf{y}$ and each of the p columns of $\mathbf{X}$ have already been centered so that all of them have zero empirical means.

After centering, the standard Gauss–Markov linear regression model for $\mathbf{y}$ on $\mathbf{X}$ can be represented as:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{6.9.3}$$

where $\boldsymbol{\beta} \in \mathbb{R}^p$ denotes the unknown parameter vector of regression coefficients and $\boldsymbol{\varepsilon}$ denotes the vector of random errors with $E(\boldsymbol{\varepsilon}) = \mathbf{0}$ and $\mathrm{var}(\boldsymbol{\varepsilon}) = \sigma^2 \mathbf{I}_{n\times n}$ for some unknown variance parameter $\sigma^2 > 0$.

The LS estimate of regression parameter vector $\boldsymbol{\beta}$ is given by

$$\hat{\boldsymbol{\beta}}_{\mathrm{LS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{6.9.4}$$

for the data matrix $\mathbf{X}$ with full column rank.

Put the regularized cost function

$$L(\boldsymbol{\beta}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2, \tag{6.9.5}$$

then the Tikhonov regularization LS solution is given by

$$\hat{\boldsymbol{\beta}}_{\mathrm{Tik}} = (\mathbf{X}^H\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^H\mathbf{y} \tag{6.9.6}$$

for the data matrix $\mathbf{X}$ with deficit column rank.

The total least squares (TLS) estimate of regression parameter vector $\boldsymbol{\beta}$ is given by

$$\hat{\boldsymbol{\beta}}_{\mathrm{TLS}} = (\mathbf{X}^H\mathbf{X} - \lambda\mathbf{I})^{-1}\mathbf{X}^H\mathbf{y} \tag{6.9.7}$$

for the data matrix $\mathbf{X}$ with full column rank and observation errors.

Principle Component Regression Method

Principle component regression (PCR) [175] is another technique for estimating $\boldsymbol{\beta}$, based on principle component analysis (PCA).

Let $\mathbf{R} = \mathbf{X}^T\mathbf{X}$ be the sample autocorrelation matrix and its SVD be $\mathbf{R} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T$, where $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_p]$ is the principal loading matrix. Then, $\mathbf{XV} = [\mathbf{Xv}_1, \ldots, \mathbf{Xv}_p]$ is the principal components of $\mathbf{X}$.

The idea of PCR is simple [175]: use principal components $\mathbf{W} = \mathbf{XV}$ of $\mathbf{X}$ instead of the original data matrix $\mathbf{X}$ in standard linear regression $\mathbf{y} = \mathbf{X}\boldsymbol{\beta}$ to form the regression equation:

$$\mathbf{y} = \mathbf{W}\boldsymbol{\gamma} = \mathbf{XV}\boldsymbol{\gamma}. \tag{6.9.8}$$

Comparing the PCR $\mathbf{y} = \mathbf{XV}\boldsymbol{\gamma}$ with the linear regression $\mathbf{y} = \mathbf{X}\boldsymbol{\beta}$, we immediately have $\boldsymbol{\beta} = \mathbf{V}\boldsymbol{\gamma}$.

Then, the procedure of PCR consists of the following steps.

- *Principal Components Extraction:* PCR starts by performing an SVD on the autocorrelation matrix of centered data matrix $\mathbf{X}$: $\mathbf{R} = \mathbf{X}^T\mathbf{X} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T$, where

$\mathbf{\Lambda}_{p\times p} = \mathbf{Diag}\left(\lambda_1^2, \ldots, \lambda_p^2\right)$ with $\lambda_1 \geq \cdots \geq \lambda_p \geq 0$ denoting the nonnegative eigenvalues (also called the principal eigenvalues) of $\mathbf{X}^T\mathbf{X}$. Then, $\mathbf{v}_j$ denotes the jth principal component direction (or known as PCA loading) corresponding to the jth largest principal value λ_j for each $j \in \{1, \ldots, p\}$. Hence, $\mathbf{V}$ is called the PCA loading matrix of $\mathbf{X}^T\mathbf{X}$.

- *PCR model:* For any $k \in \{1, \ldots, p\}$, let $\mathbf{V}_k$ denote the $p \times k$ matrix with orthonormal columns consisting of the first k columns of $\mathbf{V}$. Let $\mathbf{W}_k = \mathbf{X}\mathbf{V}_k = [\mathbf{X}\mathbf{v}_1, \ldots, \mathbf{X}\mathbf{v}_k]$ denote the $n \times k$ matrix having the first k principal components as its columns. Then, $\mathbf{W}$ may be viewed as the data matrix obtained by using the transformed covariants $\mathbf{x}_i^k = \mathbf{V}_k^T\mathbf{x}_i \in \mathbb{R}^k$ instead of using the original covariants $\mathbf{x}_i \in \mathbb{R}^p, \forall i = 1, \ldots, n$. Hence, if the standard Gauss–Markov linear regression model $\mathbf{y} = \mathbf{X}\boldsymbol{\beta}$ is viewed as a full component regression, then by using the principal component matrix $\mathbf{W}_k = \mathbf{X}\mathbf{V}_k$ instead of the original data matrix $\mathbf{X}$, we have the PCR model

$$\mathbf{y}_k = \mathbf{W}_k\boldsymbol{\gamma}_k. \tag{6.9.9}$$

- *PCR Estimator $\hat{\boldsymbol{\gamma}}_k$:* The solution of the above PCR equation is given by $\hat{\boldsymbol{\gamma}}_k = \left(\mathbf{W}_k^T\mathbf{W}_k\right)^{-1}\mathbf{W}_k^T\mathbf{y}_k \in \mathbb{R}^k$, which denotes the vector of estimated regression coefficients obtained by ordinary least squares regression of the response vector $\mathbf{y}$ on the data matrix $\mathbf{W}_k$.
- *PCR Estimator $\hat{\boldsymbol{\beta}}_k$:* Substitute $\mathbf{W}_k = \mathbf{X}\mathbf{V}_k$ into Eq. (6.9.9) to get $\mathbf{y}_k = \mathbf{X}\mathbf{V}_k\boldsymbol{\gamma}_k$. By comparing $\mathbf{y}_k = \mathbf{X}\mathbf{V}_k\boldsymbol{\gamma}_k$ with $\mathbf{y}_k = \mathbf{X}\boldsymbol{\beta}_k$, it is easily known that for any $k \in \{1, \ldots, p\}$, the final PCR estimator of $\boldsymbol{\beta}$ based on using the first k principal components is given by $\hat{\boldsymbol{\beta}}_k = \mathbf{V}_k\hat{\boldsymbol{\gamma}}_k \in \mathbb{R}^p, \quad k \in \{1, \ldots, p\}$.

Algorithm 6.12 shows the principle component regression algorithm.

Algorithm 6.12 Principal component regression algorithm [175]

```
1. input:
   1.1 The data matrix $\mathbf{X} \in \mathbb{R}^{n\times p}$ of observed covariants;
   1.2 The vector $\mathbf{y} \in \mathbb{R}^n$ of observed outcomes, where $n \geq p$.
2. initialization: $x_{ij} \leftarrow x_{ij} - \bar{x}_j$ with $\bar{x}_j = \frac{1}{n}\sum_{i=1}^n x_{ij}, j = 1, \ldots, p$, and $y_i \leftarrow y_i - \bar{y}$,
   $\bar{y} = \frac{1}{n}\sum_{i=1}^n y_i$.
3. Compute the SVD: $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, and save the singular values $\sigma_1, \ldots, \sigma_p$ and the right
   singular-vector matrix $\mathbf{V}$.
4. Determine $k$ largest principal eigenvalues $\lambda_i = \sigma_i^2, i = 1, \ldots, k$ with $k \in \{1, \ldots, p\}$, and
   denote $\mathbf{V}_k = [\mathbf{v}_1, \ldots, \mathbf{v}_k]$.
5. Construct the transformed data matrix $\mathbf{W}_k = [\mathbf{X}\mathbf{v}_1, \ldots, \mathbf{X}\mathbf{v}_k], k \in \{1, \ldots, p\}$.
6. Compute the PCR estimate $\hat{\boldsymbol{\gamma}}_k = \left(\mathbf{W}_k^T\mathbf{W}_k\right)^{-1}\mathbf{W}_k^T\mathbf{y} \in \mathbb{R}^k, k \in \{1, \ldots, p\}$.
7. The regression parameter estimate based on $k$ principal components is given by
      $\hat{\boldsymbol{\beta}}_k = \mathbf{V}_k\hat{\boldsymbol{\gamma}}_k \in \mathbb{R}^p, k \in \{1, \ldots, p\}$.
8. output: $\hat{\boldsymbol{\beta}}_k, k \in \{1, \ldots, p\}$.
```

6.9.2 *Partial Least Squares Regression*

Let $\mathcal{X} \subset \mathbb{R}^N$ be an N-dimensional space of variables representing the first block and $\mathcal{Y} \subset \mathbb{R}^M$ be a space representing the second block of variables. The general setting of a linear *partial least squares* (PLS) algorithm is to model the relations between these two data sets (blocks of variables) by means of score vectors.

The PLS model can be considered as consisting of *outer relations* ($\mathcal{X}$- and $\mathcal{Y}$-blocks individually) and an *inner relation* (linking both blocks).

Let p denote the number of components in model, N be the number of objects. The main symbols in PLS model are given as follows:

$\mathbf{X}$: $N \times K$ matrix of predictor variables,
$\mathbf{Y}$: $N \times M$ matrix of response variables,
$\mathbf{E}$: $N \times K$ residual matrix for $\mathbf{X}$,
$\mathbf{F}^*$: $N \times M$ residual matrix for $\mathbf{Y}$,
$\mathbf{B}$: $K \times M$ matrix of PLS regression coefficients,
$\mathbf{W}$: $K \times p$ matrix of the PLS weights for $\mathbf{X}$,
$\mathbf{C}$: $M \times p$ matrix of the PLS weights for $\mathbf{Y}$,
$\mathbf{P}$: $K \times p$ matrix of the PLS loadings for $\mathbf{X}$,
$\mathbf{Q}$: $M \times p$ matrix of the PLS loadings for $\mathbf{Y}$,
$\mathbf{T}$: $N \times p$ matrix of the PLS scores for $\mathbf{X}$,
$\mathbf{U}$: $N \times p$ matrix of the PLS scores for $\mathbf{Y}$.

The outer relations for the $\mathcal{X}$- and $\mathcal{Y}$-blocks are given by [105]

$$\mathbf{X} = \mathbf{T}\mathbf{P}^T + \mathbf{E} = \sum_{h=1}^{p} \mathbf{t}_h \mathbf{p}_h^T + \mathbf{E}, \tag{6.9.10}$$

$$\mathbf{Y} = \mathbf{U}\mathbf{Q}^T + \mathbf{F}^* = \sum_{h=1}^{p} \mathbf{u}_h \mathbf{q}_h^T + \mathbf{F}^*, \tag{6.9.11}$$

where $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_p] \in \mathbb{R}^{K\times p}$ and $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_p] \in \mathbb{R}^{M\times p}$ with p score vectors (components, latent vectors) extracted from $\mathbf{X}$ and $\mathbf{Y}$, respectively; the $K \times p$ matrix $\mathbf{P} = [\mathbf{p}_1, \ldots, \mathbf{p}_p]$ and the $M \times p$ matrix $\mathbf{Q} = [\mathbf{q}_1, \ldots, \mathbf{q}_p]$ represent matrices of loadings, and both $\mathbf{E} \in \mathbb{R}^{N\times K}$ and $\mathbf{F}^* \in \mathbb{R}^{N\times M}$ are the matrices of residuals.

Let $\mathbf{t} = \mathbf{X}\mathbf{w}$ be the $\mathcal{X}$-space score vector (simply $\mathcal{X}$-score) and $\mathbf{u} = \mathbf{Y}\mathbf{c}$ be the $\mathcal{Y}$-space score vector (simply $\mathcal{Y}$-score). For the user of PLS, it is necessary to know the following main properties of the PLS factors [105, 121].

1. The PLS weight vectors $\mathbf{w}_i$ for $\mathbf{X}$ are mutually orthogonal:

$$\langle \mathbf{w}_i, \mathbf{w}_j \rangle = \mathbf{w}_i^T \mathbf{w}_j = 0, \ \text{for } i \neq j. \tag{6.9.12}$$

2. The PLS score vectors $\mathbf{t}_i$ are mutually orthogonal:

$$\langle \mathbf{t}_i, \mathbf{t}_j \rangle = \mathbf{t}_i^T \mathbf{t}_j = 0, \text{ for } i \neq j. \tag{6.9.13}$$

3. The PLS weight vectors $\mathbf{w}_i$ are orthogonal to the PLS loading vectors $\mathbf{p}_j$ for $\mathbf{X}$:

$$\langle \mathbf{w}_i, \mathbf{p}_j \rangle = \mathbf{w}_i^T \mathbf{p}_j = 0, \quad \text{for } i < j. \tag{6.9.14}$$

4. The PLS loading vectors $\mathbf{p}_i$ are orthogonal in the kernel space of $\mathbf{X}$:

$$\langle \mathbf{p}_i, \mathbf{p}_j \rangle_X = \mathbf{p}_i^T (\mathbf{X}^T \mathbf{X})^\dagger \mathbf{p}_j = 0, \quad \text{for } i \neq j. \tag{6.9.15}$$

5. Both the PLS loading vectors $\mathbf{p}_i$ for $\mathbf{X}$ and the PLS loading vectors $\mathbf{q}_i$ for $\mathbf{Y}$ have unit length for each i: $\|\mathbf{p}_i\| = \|\mathbf{q}_i\| = 1$.
6. Both the PLS score vector $\mathbf{t}_h$ for $\mathbf{X}$ and the PLS score vector $\mathbf{u}_h$ for $\mathbf{Y}$ have the zero means for each h: $\sum_{i=1}^K t_{hi} = 0$ and $\sum_{i=1}^M u_{hi} = 0$.

The classical form of RLS is to find weight vectors $\mathbf{w}$ and $\mathbf{c}$ such that the inner relation linking $\mathcal{X}$-block and $\mathcal{Y}$-block satisfies [121]:

$$[\text{cov}(\mathbf{t}, \mathbf{u})]^2 = [\text{cov}(\mathbf{Xw}, \mathbf{Yc})]^2 = \max_{\|\mathbf{r}\|=\|\mathbf{s}\|=1} [\text{cov}(\mathbf{Xr}, \mathbf{Ys})]^2, \tag{6.9.16}$$

where $\text{cov}(\mathbf{t}, \mathbf{u}) = \mathbf{t}^T \mathbf{u}/N$ denotes the sample covariance between the score vectors $\mathbf{t}$ and $\mathbf{u}$. This classical RLS algorithm is called the *nonlinear iterative partial least squares* (NIPALS) *algorithm*, proposed by Wold in 1975 [274], as shown in Algorithm 6.13.

Algorithm 6.13 Nonlinear iterative partial least squares (NIPALS) algorithm [274]

```
1.  input: Data blocks $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_K]$ and $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_M]$.
2.  initialization: randomly generate $\mathcal{Y}$-space score vector $\mathbf{u}$.
3.  repeat
      // $\mathcal{X}$-space score vector update:
4.      $\mathbf{w} = \mathbf{X}^T \mathbf{u}/(\mathbf{u}^T \mathbf{u})$,
5.      $\|\mathbf{w}\| \to 1$,
6.      $\mathbf{t} = \mathbf{Xw}$.
      // $\mathcal{Y}$-space score vector update:
7.      $\mathbf{c} = \mathbf{Y}^T \mathbf{t}/(\mathbf{t}^T \mathbf{t})$,
8.      $\|\mathbf{c}\| \to 1$,
9.      $\mathbf{u} = \mathbf{Yc}$.
10.     exit if both $\mathbf{t}$ and $\mathbf{u}$ converge.
11. return
12. output: $\mathcal{X}$-space score vector $\mathbf{t}$ and $\mathcal{Y}$-space score vector $\mathbf{u}$.
```

Here are the characteristics of PLS [105].

- There are outer relations of the form $\mathbf{X} = \mathbf{TP}^T + \mathbf{E}$ and $\mathbf{Y} = \mathbf{UQ}^T + \mathbf{F}^*$.

- There is an inner relation $\mathbf{u}_h = b_h \mathbf{t}_h$.
- The mixed relation is $\mathbf{Y} = \mathbf{TBQ}^T + \mathbf{F}^*$, where $\|\mathbf{F}^*\|$ is to be minimized.
- In the iterative algorithm, the blocks get each other's scores, this gives a better inner relation.
- In order to obtain orthogonal $\mathcal{X}$-scores, as in the PCA, it is necessary to introduce weights.

There are the interesting connections between PLS, principal component analysis (PCA), and canonical correlation analysis (CCA) as follows [215].

1. The optimization criterion of PCA can be written as
$$\max_{\|\mathbf{r}\|=1} \left\{ \text{var}(\mathbf{Xr}) \right\}, \tag{6.9.17}$$
where $\text{var}(\mathbf{t}) = \mathbf{t}^T \mathbf{t}/N$ denotes the sample variance.
2. CCA finds the direction of maximal correlation by solving the optimization problem
$$\max_{\|\mathbf{r}\|=\|\mathbf{s}\|=1} \left\{ [\text{corr}(\mathbf{Xr}, \mathbf{Ys})]^2 \right\}, \tag{6.9.18}$$
where $[\text{corr}(\mathbf{t}, \mathbf{u})]^2 = [\text{cov}(\mathbf{t}, \mathbf{u})]^2/(\text{var}(\mathbf{t})\text{var}(\mathbf{u}))$ denotes the sample squared correlation.
3. The PLS optimization criterion (6.9.18) can be rewritten as
$$\max_{\|\mathbf{r}\|=\|\mathbf{s}\|=1} \left\{ [\text{cov}(\mathbf{Xr}, \mathbf{Ys})]^2 \right\} = \max_{\|\mathbf{r}\|=\|\mathbf{s}\|=1} \left\{ \text{var}(\mathbf{Xr})[\text{corr}(\mathbf{Xr}, \mathbf{Ys})]^2 \text{var}(\mathbf{Ys}) \right\}. \tag{6.9.19}$$
This represents that PLS is a form of CCA where the criterion of maximal correlation is balanced with the requirement to explain as much variance as possible in both $\mathcal{X}$- and $\mathcal{Y}$-spaces. Note that only the $\mathcal{X}$-space variance is involved in the case of a one-dimensional $\mathcal{Y}$-space.

The regression based on PLS-model equations (6.9.10) and (6.9.11) is called *partial least squares* (PLS) *regression*. To combine (6.9.10) and (6.9.11) into the standard model of PLS regression, two assumptions are made:

- the score vectors $\{\mathbf{t}_i\}_{i=1}^p$ are good predictors of $\mathbf{Y}$, where p denotes the number of extracted $\mathcal{X}$-score vectors;
- a linear inner relation between the score's vectors $\mathbf{t}$ and $\mathbf{u}$ exists; that is, $\mathbf{u}_i = \alpha_i \mathbf{t}_i + \mathbf{h}_i$, where $\mathbf{h}_i$ denotes a residual vector. Therefore, $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_p]$ and the $K \times M$ matrix of PLS regression coefficients $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_p]$ are related by
$$\mathbf{U} = \mathbf{TD} + \mathbf{H}, \tag{6.9.20}$$

where $\mathbf{D} = \mathbf{Diag}(\alpha_1, \ldots, \alpha_p)$ is $p \times p$ diagonal matrix and $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_p]$ is the matrix of residuals.

Letting $\mathbf{W}$ be a weighting matrix such that $\mathbf{EW} = \mathbf{O}$ (zero matrix), and post-multiplying (6.9.10) by $\mathbf{W}$, then

$$\mathbf{XW} = \mathbf{TP}^T\mathbf{W} \quad \Rightarrow \quad \mathbf{T} = \mathbf{XW}(\mathbf{P}^T\mathbf{W})^{-1}. \tag{6.9.21}$$

On the other hand, substitute (6.9.20) into (6.9.11) to give

$$\mathbf{Y} = \mathbf{TDQ}^T + \mathbf{HQ}^T + \mathbf{F}^* \quad \Rightarrow \quad \mathbf{Y} = \mathbf{TC}^T + \mathbf{F}, \tag{6.9.22}$$

where $\mathbf{C} = \mathbf{QD}$ denotes the $N \times M$ matrix of regression coefficients and $\mathbf{F} = \mathbf{HQ}^T + \mathbf{F}^*$ is the residual matrix. Then, substituting (6.9.21) into (6.9.22), we have $\mathbf{Y} = \mathbf{XW}(\mathbf{P}^T\mathbf{W})^{-1}\mathbf{C}^T + \mathbf{F}^*$ or can write the PLS regression model as

$$\mathbf{Y} = \mathbf{XB}_{\text{PLS}} + \mathbf{F} \quad \text{and} \quad \mathbf{B}_{\text{PLS}} = \mathbf{W}(\mathbf{P}^T\mathbf{W})^{-1}\mathbf{C}^T \tag{6.9.23}$$

represents the matrix of PLS regression coefficients.

In NIPALS algorithm, $\mathbf{w} = \mathbf{X}^T\mathbf{u}/(\mathbf{u}^T\mathbf{u})$ and $\mathbf{c} = \mathbf{Y}^T\mathbf{t}/(\mathbf{t}^T\mathbf{t})$ are generated, which imply that

$$\mathbf{W} = \mathbf{X}^T\mathbf{U} \quad \text{and} \quad \mathbf{C} = \mathbf{Y}^T\mathbf{T}. \tag{6.9.24}$$

Moreover, pre-multiplying (6.9.21) by $\mathbf{T}^T$ and using $\mathbf{T}^T\mathbf{T} = \mathbf{I}$, then

$$(\mathbf{P}^T\mathbf{W})^{-1} = (\mathbf{T}^T\mathbf{XX}^T\mathbf{U})^{-1}. \tag{6.9.25}$$

Substitute (6.9.24) and (6.9.25) into (6.9.23) to give

$$\mathbf{B}_{\text{PLS}} = \mathbf{X}^T\mathbf{U}(\mathbf{T}^T\mathbf{XX}^T\mathbf{U})^{-1}\mathbf{T}^T\mathbf{Y}. \tag{6.9.26}$$

The NIPALS regression is an iterative process; i.e., after extraction of one component $\mathbf{t}_1$ the algorithm starts again using the deflated matrices $\mathbf{X}$ and $\mathbf{Y}$ to extract the next component $\mathbf{t}_2$. This process is repeated until the deflated matrix $\mathbf{X}$ is a null matrix.

Algorithm 6.14 shows a simple nonlinear iterative partial least squares (NIPALS) regression algorithm developed by Wold et al. [275].

The basic idea of the NIPALS regression can be summarized below.

- Find the first $\mathcal{X}$-score $\mathbf{t}_1 = \mathbf{Xw}$ and the first $\mathcal{Y}$-score $\mathbf{u}_1 = \mathbf{Yc}/(\mathbf{c}^T\mathbf{c})$ from the original data blocks $\mathbf{X}$ and $\mathbf{Y}$.
- Use contractive mapping to construct the residual matrices $\mathbf{X} = \mathbf{X} - \mathbf{tp}^T$ and $\mathbf{Y} = \mathbf{Y} - b\mathbf{tc}^T$.

Algorithm 6.14 Simple nonlinear iterative partial least squares regression algorithm [275]

```
1.  input: Optionally transformed, scaled, and centered data, $\mathbf{X} = \mathbf{X}_0$ and $\mathbf{Y} = \mathbf{Y}_0$.
2.  initialization: A starting vector of $\mathbf{u}$, usually one of columns of $\mathbf{Y}$. With a single $\mathbf{y}$, take
    $\mathbf{u} = \mathbf{y}$. Put $k = 0$
3.  repeat
4.    Compute the $\mathcal{X}$-weights: $\mathbf{w} = \mathbf{X}^T\mathbf{u}/(\mathbf{u}^T\mathbf{u})$.
5.    Calculate the $\mathcal{X}$-scores $\mathbf{t} = \mathbf{Xw}$.
6.    Calculate the $\mathcal{Y}$-weights $\mathbf{c} = \mathbf{Y}^T\mathbf{t}/(\mathbf{t}^T\mathbf{t})$.
7.    Update a set of $\mathcal{Y}$-scores $\mathbf{u}$ as $\mathbf{u} = \mathbf{Yc}/(\mathbf{c}^T\mathbf{c})$.
8.    exit if the convergence $\|\mathbf{t}_{\text{old}} - \mathbf{t}_{\text{new}}\|/\|\mathbf{t}_{\text{new}}\| < \varepsilon$ has been reached, where $\varepsilon$ is "small" (e.g.,
      $10^{-6}$ or $10^{-8}$).
9.  return
10. Remove (deflate, peel off) the present component from $\mathbf{X}$ and $\mathbf{Y}$, and use these deflated
    matrices as $\mathbf{X}$ and $\mathbf{Y}$ in the next component. Here the deflation of $\mathbf{Y}$ is optional; the results
    are equivalent whether $\mathbf{Y}$ is deflated or not.
11. Compute $\mathcal{X}$-loadings: $\mathbf{p} = \mathbf{X}^T\mathbf{t}/(\mathbf{t}^T\mathbf{t})$.
12. Compute $\mathcal{Y}$-loadings: $\mathbf{q} = \mathbf{Y}^T\mathbf{u}/(\mathbf{u}^T\mathbf{u})$.
13. Regression ($\mathbf{u}$ on $\mathbf{t}$): $b = \mathbf{u}^T\mathbf{t}/(\mathbf{t}^T\mathbf{t})$.
14. Deflate $\mathbf{X}$, $\mathbf{Y}$ matrices: $\mathbf{X} \leftarrow \mathbf{X} - \mathbf{tp}^T$ and $\mathbf{Y} \leftarrow \mathbf{Y} - b\mathbf{tc}^T$.
15. $k \leftarrow k + 1$.
16. $\mathbf{u}_k = \mathbf{u}$ and $\mathbf{t}_k = \mathbf{t}$.
17. The next set of iterations starts with the new $\mathbf{X}$ and $\mathbf{Y}$ matrices as the residual matrices from
    the previous iteration. The iterations continue until a stopping criteria is used or $\mathbf{X}$ becomes
    the zero matrix.
18. $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_p]$ and $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_p]$.
19. $\mathbf{B}_{\text{PLS}} = \mathbf{X}^T\mathbf{U}(\mathbf{T}^T\mathbf{X}\mathbf{X}^T\mathbf{U})^{-1}\mathbf{T}^T\mathbf{Y}$.
20. output: the matrix of PLS regression coefficients $\mathbf{B}_{\text{PLS}}$.
```

- Find the second $\mathcal{X}$-score $\mathbf{t}_2$ and $\mathcal{Y}$-score $\mathbf{u}_2$ from the residual matrices $\mathbf{X} = \mathbf{X} - \mathbf{tp}^T$ and $\mathbf{Y} = \mathbf{Y} - b\mathbf{tc}^T$.
- Construct the new residual matrices $\mathbf{X}$ and $\mathbf{Y}$ and find the third $\mathcal{X}$- and $\mathcal{Y}$-scores $\mathbf{t}_3$ and $\mathbf{u}_3$. Repeat this process until the residual matrix $\mathbf{X}$ becomes a null matrix.
- Use (6.9.26) to compute the matrix of regression coefficients $\mathbf{B}$, where $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_p]$ and $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_p]$ consist of p $\mathcal{X}$- and $\mathcal{Y}$-scores, respectively.

Once the matrix of PLS regression coefficients $\mathbf{B}_{\text{PLS}}$ is found by using Algorithm 6.14, for a given new data block $\mathbf{X}_{\text{new}}$, then unknown $\mathcal{Y}$-values can be predicted by Eq. (6.9.23) as

$$\hat{\mathbf{Y}}_{\text{new}} = \mathbf{X}_{\text{new}}\mathbf{B}_{\text{PLS}}. \tag{6.9.27}$$

6.9.3 Penalized Regression

For linear regression problems $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} \in \mathbb{R}^n$ with $\mathbf{X} \in \mathbb{R}^{n\times p}$, $\boldsymbol{\beta} \in \mathbb{R}^p$, ordinary least squares (OLS) regression minimizes squared regression error $\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 =$

$(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})$, and yields a $p \times 1$ unbiased estimator $\hat{\boldsymbol{\beta}}^{\text{OLS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$. Although the OLS estimator is simple and unbiased, if the design matrix $\mathbf{X}$ is not of full-rank, then $\hat{\boldsymbol{\beta}}$ is not unique and its variance $\text{var}(\hat{\boldsymbol{\beta}}) = (\mathbf{X}^T\mathbf{X})^{-1}\sigma^2$ (σ^2 is the variance of i.i.d. regression error) may be very large.

To achieve better prediction, Hoerl and Kennard [117, 118] introduced *ridge regression*

$$\hat{\boldsymbol{\beta}} = \arg\max_{\boldsymbol{\beta}} \left\{\frac{1}{2}\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2\right\} \quad \text{subject to} \sum_{j=1}^{p} |\beta_j|^\gamma \le t, \tag{6.9.28}$$

where $\gamma \ge 1$ and $t \ge 0$. The equivalent form of ridge regression is the following *penalized regression* [101]:

$$\hat{\boldsymbol{\beta}} = \arg\max_{\boldsymbol{\beta}} \left\{\frac{1}{2}\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\sum_{j=1}^{p} |\beta_j|^\gamma\right\}, \tag{6.9.29}$$

where $\gamma \ge 1$ and $\lambda \ge 0$ are a positive scalar; $\lambda = 0$ corresponds to ordinary least squares regression.

When γ takes different values, the penalized regression has different forms. The most well-known penalized regressions are ℓ_2 penalized regression with $\gamma = 2$ (often called the ridge regression):

$$\hat{\boldsymbol{\beta}}^{\text{ridge}} = \arg\min_{\boldsymbol{\beta}} \left\{\frac{1}{2}\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2\right\}, \tag{6.9.30}$$

and the ℓ_1 penalized regression with $\gamma = 1$, called *least absolute shrinkage and selection operator* (Lasso), given by

$$\hat{\boldsymbol{\beta}}^{\text{Lasso}} = \arg\min_{\boldsymbol{\beta}} \left\{\frac{1}{2}\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1\right\}, \tag{6.9.31}$$

where $\|\boldsymbol{\beta}\|_1 = \sum_{i=1}^{p} |\beta_i|$ is ℓ_1-norm.

The solution of the ridge regression is given by

$$\hat{\boldsymbol{\beta}}^{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{6.9.32}$$

that is just the Tikhonov regularized solution [246, 247].

The Lasso solution is, in a soft-thresholded version [77], given by [99]

$$\hat{\beta}_i^{\text{Lasso}}(\gamma) = S(\hat{\beta}_i, \gamma) = \text{sign}(\hat{\beta}_i)(|\hat{\beta}_i| - \gamma)_+ \tag{6.9.33}$$

$$= \begin{cases} \hat{\beta}_i - \gamma, & \text{if } \hat{\beta}_i > 0 \text{ and } \gamma < |\hat{\beta}_i|; \\ \hat{\beta}_i + \gamma, & \text{if } \hat{\beta}_i < 0 \text{ and } \gamma < |\hat{\beta}_i|; \\ 0, & \text{if } \gamma \geq |\hat{\beta}_i|; \end{cases} \tag{6.9.34}$$

with

$$x_+ = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{if } x \leq 0. \end{cases} \tag{6.9.35}$$

Important differences between the ridge regression and the Lasso regression are twofold.

- The ℓ_1 penalty in Lasso results in variable selection, as variables with coefficients of zero are effectively omitted from the model, but the ℓ_2 penalty in ridge regression does not perform variable selection.
- An ℓ_2 penalty $\lambda \sum_{j=1}^n \beta_j^2$ pushes β_j toward zero with a force proportional to the value of the coefficient, whereas an ℓ_1 penalty $\lambda \sum_{j=1}^n |\beta_j|$ exerts the same force on all nonzero coefficients. Hence for variables that are most valuable in the model and where shrinkage toward zero is less desirable, an ℓ_1 penalty shrinks less [116].

Typically, the optimization problems with inequality constraints are carried out using a standard quadratic programming algorithm. An alternative is to explore "one-at-a-time" *coordinate-wise descent algorithms* for these problems [99].

For large optimization problems, coordinate descent can deliver a path of solutions efficiently, and can be applied to many other convex statistical problems such as the large Lasso, the garotte, elastic net, and so on [99].

The following are a few of typical versions of the Lasso and their respective solutions [99].

- *Nonnegative garotte:* This method, developed by Breiman [35], is a precursor to the Lasso, and solves

$$\min_c \left\{ \frac{1}{2} \sum_{i=1}^n \left(y_i - \sum_{j=1}^p x_{ij} c_j \hat{\beta}_j \right)^2 + \gamma \sum_{j=1}^p c_j \right\} \quad \text{subject to } c_j \geq 0, \tag{6.9.36}$$

where $\hat{\beta}_j$ is the usual least squares estimates ($p \le n$ is assumed). The coordinate-wise update is given by

$$c_j \leftarrow \left(\frac{\tilde{\beta}_j \hat{\beta}_j - \lambda}{\hat{\beta}_j^2} \right)_+, \tag{6.9.37}$$

where $\tilde{\beta}_j = \sum_{i=1}^n x_{ij} \left(y_i - \tilde{y}_i^{(j)} \right)$ and $\tilde{y}_i^{(j)} = \sum_{k \ne j} x_{ik} c_k \hat{\beta}_k$.

- *Elastic net:* This method [307] adds another constraint $\lambda_2 \sum_{j=1}^p \beta_j^2/2$ to Lasso to solve

$$\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2} \sum_{i=1}^n \left(y_i - \sum_{j=1}^p x_{ij} \beta_j \right) + \lambda_1 \sum_{j=1}^p |\beta_j| + \lambda_2 \sum_{j=1}^p \beta_j^2/2 \right\}. \tag{6.9.38}$$

The coordinate-wise update has the form

$$\tilde{\beta}_j \leftarrow \frac{S\left(\sum_{i=1}^n x_{ij} \left(y_i - \tilde{y}_i^{(j)} \right), \lambda_1 \right)_+}{1 + \lambda_2}. \tag{6.9.39}$$

- *Grouped Lasso:* Let $\mathbf{X}_j$ be an $N \times p_j$ orthonormal matrix that represents the jth group of p_j variables, $j = 1, \ldots, m$, and β_j the corresponding coefficient vector. The grouped Lasso [292] solves

$$\min_{\boldsymbol{\beta}} \left\{ \left\| \mathbf{y} - \sum_{j=1}^m \mathbf{X}_j \boldsymbol{\beta}_j \right\|_2^2 + \sum_{j=1}^m \lambda_j \|\boldsymbol{\beta}_j\|_2 \right\}, \tag{6.9.40}$$

where $\lambda_j = \lambda \sqrt{p_j}$. The coordinate-wise update is

$$\tilde{\beta}_j \leftarrow (\|\mathbf{s}_j\|_2 - \lambda_j)_+ \frac{\mathbf{s}_j}{\|\mathbf{s}_j\|_2}, \tag{6.9.41}$$

here $\mathbf{s}_j = \mathbf{X}_j^T \left(\mathbf{y} - \tilde{\mathbf{y}}^{(j)} \right)$ with $\tilde{\mathbf{y}}^{(j)} = \sum_{k \ne j} \mathbf{X}_k \tilde{\boldsymbol{\beta}}_k$.

- *"Berhu" penalty:* This method [198] is a robust hybrid of Lasso and ridge regression, and its Lagrange form is

$$\begin{aligned} \min_{\boldsymbol{\beta}} \Bigg\{ & \frac{1}{2} \sum_{i=1}^n \left(y_i - \sum_{j=1}^p x_{ij} \beta_j \right) \\ & + \lambda \sum_{j=1}^p \left(|\beta_j| \cdot I(|\beta_j| < \delta) + \frac{(\beta_j^2 + \delta^2)}{2\delta} \cdot I(|\beta_j| \ge \delta) \right) \Bigg\}. \end{aligned} \tag{6.9.42}$$

Berhu penalty is the reverse of a “Huber” function. The coordinate-wise update has the form

$$\tilde{\beta}_j \leftarrow \begin{cases} S\left(\sum_{i=1}^n x_{ij}(y_i - \tilde{y}^{(j)}), \lambda\right), & \text{if } |\beta_j| < \delta; \\ \sum_{i=1}^n x_{ij}(y_i - \tilde{y}^{(j)})/(1+\lambda/\delta), & \text{if } |\beta_j| \geq \delta. \end{cases} \tag{6.9.43}$$

This is a robust hybrid consisting of Lasso-style soft-thresholding for values less than δ and ridge-style beyond δ.

6.9.4 Gradient Projection for Sparse Reconstruction

As the general formulation of Lasso problem, we consider the unconstrained convex optimization problem

$$\min_{\mathbf{x}} \left\{ \frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \tau\|\mathbf{x}\|_1 \right\}, \tag{6.9.44}$$

where $\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^k, \mathbf{A} \in \mathbb{R}^{k\times n}$, and τ is a nonnegative parameter. When the variable $\mathbf{x}$ is a sparse vector, the above optimization is also called the *sparse reconstruction* of $\mathbf{x}$.

The optimization problem (6.9.44) is closely related to the following constrained convex optimization problems: *quadratically constrained linear program* (QCLP)

$$\min_{\mathbf{x}} \|\mathbf{x}\|_1 \quad \text{subject to} \quad \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 \leq \epsilon \tag{6.9.45}$$

and *quadratic program* (QP)

$$\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 \quad \text{subject to} \quad \|\mathbf{x}\|_1 \leq \alpha, \tag{6.9.46}$$

where ϵ and α are nonnegative real parameters.

By various enhancements to the basic gradient projection (GP) algorithm applied to a quadratic programming formulation of (6.9.44), Figueiredo et al. [92] proposed a GPSR (*gradient projection for sparse reconstruction*). This GPSR approach, as in [102], splits the variable $\mathbf{x}$ into its positive and negative parts:

$$\mathbf{x} = \mathbf{u} - \mathbf{v}, \quad \mathbf{u} \geq \mathbf{0}, \ \mathbf{v} \geq \mathbf{0}, \tag{6.9.47}$$

where $u_i = (x_i)_+$ and $v_i = (-x_i)_+$ for all $i = 1, \ldots, n$ with $(x)_+ = \max\{0, x\}$. Then, (6.9.44) can be rewritten as the following *bound-constrained quadratic* *program* (BCQP):

$$\min_{\mathbf{u},\mathbf{v}} \left\{ \frac{1}{2}\|\mathbf{y}-\mathbf{A}(\mathbf{u}-\mathbf{v})\|_2^2 + \tau \mathbf{1}_n^T \mathbf{u} + \tau \mathbf{1}_n^T \mathbf{v} \right\},$$

$$\text{subject to} \quad \mathbf{u} \geq \mathbf{0},\ \mathbf{v} \geq \mathbf{0}. \tag{6.9.48}$$

Equation (6.9.48) can be written in more standard BCQP form

$$\min_{\mathbf{z}} \left\{ F(\mathbf{z}) = \mathbf{c}^T\mathbf{z} + \frac{1}{2}\mathbf{z}^T\mathbf{B}\mathbf{z} \right\} \quad \text{subject to} \quad \mathbf{z} \geq \mathbf{0}, \tag{6.9.49}$$

where

$$\mathbf{z} = \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}, \quad \mathbf{b} = \mathbf{A}^T\mathbf{y}, \quad \mathbf{c} = \tau \mathbf{1}_{2n} + \begin{bmatrix} -\mathbf{b} \\ \mathbf{b} \end{bmatrix}, \tag{6.9.50}$$

and

$$\mathbf{B} = \begin{bmatrix} \mathbf{A}^T\mathbf{A} & -\mathbf{A}^T\mathbf{A} \\ -\mathbf{A}^T\mathbf{A} & \mathbf{A}^T\mathbf{A} \end{bmatrix}. \tag{6.9.51}$$

When the solution of (6.9.44) is known in advance to be nonnegative, we have

$$\frac{1}{2}\|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2 = \frac{1}{2}\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{A}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x},$$

since $\mathbf{x}^T\mathbf{A}^T\mathbf{y} = (\mathbf{x}^T\mathbf{A}^T\mathbf{y})^T = \mathbf{y}^T\mathbf{A}\mathbf{x}$. It is noted that $\mathbf{y}^T\mathbf{y}$ is a constant independent of the variable $\mathbf{x}$ and $\tau\|\mathbf{x}\|_1 = \tau\mathbf{1}_n^T\mathbf{x}$ under the assumption that $\mathbf{x}$ is nonnegative, thus Eq. (6.9.44) can be equivalently written as

$$\min_{\mathbf{x}} \left\{ \left(\tau\mathbf{1}_n - \mathbf{A}^T\mathbf{y}\right)^T \mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} \right\} \quad \text{subject to} \quad \mathbf{x} \geq \mathbf{0} \tag{6.9.52}$$

which has the same form as (6.9.49).

Gradient projection is an efficient method for solving general minimization problems over a convex set C in unconstrained minimization form

$$\min_{\mathbf{x}} \{f(\mathbf{x}) + h(\mathbf{x})\}, \tag{6.9.53}$$

where $f(\mathbf{x})$ is continuously differentiable on C and $h(\mathbf{x})$ is non-smoothing on C. Following the first-order optimality condition, we have

$$\mathbf{x}^* \in \arg\min_{\mathbf{x}\in C} \{f(\mathbf{x}) + h(\mathbf{x})\} \quad \Leftrightarrow \quad \mathbf{0} \in \nabla f(\mathbf{x}^*) + \partial h(\mathbf{x}^*). \tag{6.9.54}$$

where $\nabla f(\mathbf{x})$ is the gradient of the continuously differentiable function f and $\partial h(\mathbf{x})$ is the subdifferential of the non-smoothing function h.

Especially, when $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}-\mathbf{z}\|_2^2$, we have $\nabla f(\mathbf{x}) = \mathbf{x}-\mathbf{z}$, and thus the above first-order condition becomes

$$\mathbf{0} \in (\mathbf{x} - \mathbf{z}) + \partial h(\mathbf{x}) \quad \Leftrightarrow \quad \mathbf{z} \in \mathbf{x} + \partial h(\mathbf{x}). \tag{6.9.55}$$

If taking $\partial h(\mathbf{x}) = -\nabla f(\mathbf{x})$ in Eq. (6.9.54), then Eq. (6.9.55) gives the following result:

$$\mathbf{z} = \mathbf{x} - \nabla f(\mathbf{x}). \tag{6.9.56}$$

This is just the *gradient projection* of $\mathbf{x}$ onto the convex set C. Then, the basic gradient projection updating formula is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mu_k \nabla f(\mathbf{x}_k), \tag{6.9.57}$$

where μ_k is the step length at the kth update.

The basic gradient projection approach for solving (6.9.49) consists of the following two steps [92].

- First, choose some scalar parameter $\alpha_k > 0$ and set

$$\mathbf{w}_k = (\mathbf{z}_k - \alpha_k \nabla F(\mathbf{z}_k))_+ . \tag{6.9.58}$$

- Then, choose the second scalar $\lambda_k \in [0, 1]$ and set

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \lambda_k(\mathbf{w}_k - \mathbf{z}_k). \tag{6.9.59}$$

In the basic approach, each iterate $\mathbf{z}_k$ is searched along the negative gradient $-\nabla F(\mathbf{z}_k)$, projecting onto the nonnegative orthant, and performing a *backtracking line search* until a sufficient decrease is attained in F [92].

Figueiredo et al. [92] proposed two gradient projection algorithms for sparse reconstruction: a basic gradient projection (GPSR-Basic) algorithm and a Barzilai–Borwein gradient projection (GPSR-BB) algorithm.

The GPSR-BB algorithm, as shown in Algorithm 6.15, is based on the Barzilai–Borwein (BB) method, using $\mathbf{H}_k = \eta^{(k)}\mathbf{I}$ as an approximation of the Hessian matrix (MATLAB implementations are available at http://www.lx.it.pt/~mtf/GPSR).

Algorithm 6.15 GPSR-BB algorithm [92]

```
1.  input: The data vector $\mathbf{y} \in \mathbb{R}^m$ and the input matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$.
2.  initialization: Randomly generate a nonnegative vector $\mathbf{z}^{(0)} \in \mathbb{R}_+^{2n}$.
3.  Choose nonnegative parameters $\tau, \beta \in (0, 1), \mu \in (0, 1/2)$ and $\alpha_{\min}, \alpha_{\max}, \alpha_0 \in [\alpha_{\min}, \alpha_{\max}]$.
4.  Compute $\mathbf{b} = \mathbf{A}^T\mathbf{y},\ \mathbf{c} = \tau\mathbf{1}_{2n} + \begin{bmatrix} -\mathbf{b} \\ \mathbf{b} \end{bmatrix}$ and $\mathbf{B} = \begin{bmatrix} \mathbf{A}^T\mathbf{A} & -\mathbf{A}^T\mathbf{A} \\ -\mathbf{A}^T\mathbf{A} & \mathbf{A}^T\mathbf{A} \end{bmatrix}$.
5.  Set $k = 0$.
6.  repeat
7.     Calculate the gradient $\nabla F(\mathbf{z}^{(k)}) = \mathbf{c} + \mathbf{B}\mathbf{z}^{(k)}$.
8.     Compute step: $\boldsymbol{\delta}^{(k)} = \left(\mathbf{z}^{(k)} - \alpha^{(k)}\nabla F(\mathbf{z}^{(k)})\right)_+ - \mathbf{z}^{(k)}$.
9.     (line search): Find the scalar $\lambda^{(k)}$ that minimizes $F(\mathbf{z}^{(k)} + \lambda^{(k)}\boldsymbol{\delta}^{(k)})$ on the interval $\lambda^{(k)} \in$
       $[0, 1]$.
10.    Set $\mathbf{z}^{(k+1)} = \mathbf{z}^{(k)} + \lambda^{(k)}\boldsymbol{\delta}^{(k)}$.
11.    (update $\alpha$): compute
          $\gamma^{(k)} = \left(\boldsymbol{\delta}^{(k)}\right)^T \mathbf{B}\boldsymbol{\delta}^{(k)};$
       if $\gamma^{(k)} = 0$ then let $\alpha^{(k)} = \alpha_{\max}$, otherwise
          $\alpha^{(k)} = \mathrm{mid}\left\{\alpha_{\min}, \frac{\|\boldsymbol{\delta}^{(k)}\|_2^2}{\gamma^{(k)}}, \alpha_{\max}\right\}.$
12.    If $\left\|\mathbf{z}^{(k)} - \left(\mathbf{z}^{(k)} - \bar{\alpha}\nabla F(\mathbf{z})\right)_+\right\| \le$ tolP, where tolP is a small parameter and $\bar{\alpha}$ is a positive
       constant, then terminate; otherwise set $k \leftarrow k + 1$ return to Step 6.
13. output: $\mathbf{z}^{(k+1)}$.
```

6.10 Supervised Learning Classification

Linear classification is a useful tool in machine learning and data mining. Unlike nonlinear classifiers (such as kernel methods) mapping data to a higher-dimensional space, linear classifiers directly work on data in the original input space. For some data in a rich dimensional space, the performance (i.e., testing accuracy) of linear classifiers has shown to be close to that of nonlinear classifiers [293].

6.10.1 Binary Linear Classifiers

Definition 6.14 (Instance [305]) An *instance* $\mathbf{x}$ represents a specific object. The instance is often represented by a D-dimensional feature vector $\mathbf{x} = [x_1, \ldots, x_D]^T \in \mathbb{R}^D$, where each dimension is called a feature. The length D of the feature vector is known as the dimensionality of the feature vector.

It is assumed that these instances are sampled independently from an underlying distribution $P(\mathbf{x})$, which is unknown to us and is denoted by $\{\mathbf{x}_i\}_{i=1}^N \overset{\text{i.i.d.}}{\sim} P(\mathbf{x})$, where i.i.d. stands for *independent and identically distributed*.

Definition 6.15 (Training Sample [305]) A *training sample* is a collection of instances $\{\mathbf{x}_i\}_{i=1}^N = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, which acts as the input to the learning process.

Definition 6.16 (Label [305]) The desired prediction y on an instance $\mathbf{x}$ is known as its *label.*

Definition 6.17 (Supervised Learning Classification) Let $X = \{\mathbf{x}_i\}$ be the domain of instances and $Y = \{y_i\}$ be the domain of labels. Let $P(\mathbf{x}, y)$ be an (unknown) joint probability distribution on instances and labels $X \times Y$. Given a set of training samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N \overset{\text{i.i.d.}}{\sim} P(\mathbf{x}, y)$, *supervised learning classification* trains a function $f : X \to Y$ in some function family F in order to make $f(\mathbf{x})$ predict the true label y on future testing data $\mathbf{x}$, where $(\mathbf{x}, y) \overset{\text{i.i.d.}}{\sim} P(\mathbf{x}, y)$ as well.

In binary classification, we are given training data $\{\mathbf{x}_i, y_i\} \in \mathbb{R}^D \times \{-1, +1\}$ for $i = 1, \ldots, N$. Linear classification methods construct the following decision function:

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = \mathbf{w} \cdot \mathbf{x} + b = \mathbf{w}^T \mathbf{x} + b, \tag{6.10.1}$$

where $\mathbf{w}$ is called the weight vector and b is an *intercept*, or called the *bias*.

To generate a decision function $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$, linear classification involves the following risk minimization problem [293]:

$$\min_{\mathbf{w}} \left\{ f(\mathbf{w}) = R(\mathbf{w}) + C \sum_{i=1}^N \xi(\mathbf{w}^T \mathbf{x}_i, y_i) \right\}, \tag{6.10.2}$$

where

- $\mathbf{w}$ is a weight vector consisting of linear classifier parameters;
- $R(\mathbf{w})$ is a regularization function that prevents the overfitting parameters;
- $\xi(\mathbf{w}^T \mathbf{x}; y_i)$ is a loss function measuring the discrepancy between the classifier's prediction and the true output y_i for the ith training example;
- $C > 0$ is a scalar constant (pre-specified by the user of the learning algorithm) that controls the balance between the regularization $R(\mathbf{w})$ and the loss function $\xi(\mathbf{w}^T \mathbf{x}; y_i)$.

The output of linear classifier, $d(\mathbf{w}) = \mathbf{w}^T \mathbf{x}$, is called the *decision function* of the classifier, since the classification decision (i.e., the label y) is made by $\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x})$.

The aim of the regularization function $R(\mathbf{w})$ is to prevent our model from overfitting the observations found in the training set. The following regularization items are commonly applied:

$$R_1(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{k=1}^N |w_k|, \tag{6.10.3}$$

or

$$R_2(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|_2^2 = \frac{1}{2} \sum_{k=1}^N w_k^2. \tag{6.10.4}$$

The regularization $R_1(\mathbf{w}) = \|\mathbf{w}\|_1$ has the following limitations:

- It is not strictly convex, so the solution may not be unique.
- For two highly correlated features, the solution obtained by R_1 regularization may select only one of these features. Consequently, R_1 regularization may discard the group effect of variables with high correlation [307].

To overcome the above two limitations of R_1 regularization, a convex combination of R_1 and R_2 regularizations forms the following elastic net [307]:

$$R_e(\mathbf{w}) = \lambda\|\mathbf{w}\|_2^2 + (1-\lambda)\|\mathbf{w}\|_1, \tag{6.10.5}$$

where $\lambda \in (0, 1)$.

Popular loss functions $\xi(\mathbf{w}^T\mathbf{x}_i, y_i)$ include the hinge loss (i.e., interior penalty function)

$$\xi_1\Big(\mathbf{w}^T\mathbf{x}_i, y_i\Big) = \max\Big(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\Big), \tag{6.10.6}$$

$$\xi_2\Big(\mathbf{w}^T\mathbf{x}_i, y_i\Big) = \max\Big(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\Big)^2, \tag{6.10.7}$$

for linear support vector machines and the log loss (or log penalty term)

$$\xi_3\Big(\mathbf{w}^T\mathbf{x}_i, y_i\Big) = \log\Big(1 + \mathrm{e}^{-y_i\mathbf{w}^T\mathbf{x}_i}\Big) \tag{6.10.8}$$

for linear logistic regression [293].

Clearly, if all of the classifier's outputs $\mathbf{w}^T\mathbf{x}_i$ and the corresponding labels y_i have the same sign (i.e., correct decision making), then all terms $y_i\mathbf{w}^T\mathbf{x}_i > 0$ for $i = 1, \ldots, N$, thus resulting in any of the above loss functions evaluating to zero; otherwise, there is at least one of $\xi(\mathbf{w}^T\mathbf{x}_i, y_i) > 0$ for some i, so the loss function is punished.

6.10.2 Multiclass Linear Classifiers

In multiclass classification it is necessary to determine which of a finite set of classes the test input $\mathbf{x}$ belongs to.

Let $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ be a set of N training examples, where each example $\mathbf{x}_i$ is drawn from a domain $\mathcal{X} \subseteq \mathbb{R}^D$, and each label y_i is an integer from the set $\mathcal{Y} = \{1, \ldots, k\}$.

As a concrete example of a supervised learning method, *k-nearest neighbor* (*k*NN) is a simple classification algorithm, as shown in Algorithm 6.16.

Being a D-dimensional feature vector, the test instance $\mathbf{x}$ can be viewed as a point in D-dimensional feature space. A classifier assigns a label to each point in

Algorithm 6.16 k-nearest neighbor (kNN)

```
1. input: Training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{N}, {}_N)$, distance function $d(\mathbf{a}, \mathbf{b})$, number of neighbors $k$, testing instance $\mathbf{x}$.
2. initialization: let the $i$th neighbor's data subset $L_i = \{\mathbf{x}_1^{(i)}, \ldots, \mathbf{x}_{n_i}^{(i)}\}$ for $y_1, \ldots, y_N \in \{i\}$, where $n_1 + \cdots + n_k = N$.
3. Compute centers of $k$ neighbors $\mathbf{m}^{(i)} = \frac{1}{N_i}\sum_{j=1}^{n_i} \mathbf{x}_j^{(i)}$ for $i = 1, \ldots, k$.
4. Find the distances between $\mathbf{x}$ and the center $\mathbf{m}_i$ of the $i$th neighbor, denoted $d(\mathbf{x}, \mathbf{m}_i)$, for $i = 1, \ldots, k$.
5. Find the nearest neighbor of $\mathbf{x}$ using label of $y \leftarrow \min_i \{d(\mathbf{x}, \mathbf{m}^{(1)}), \ldots, d(\mathbf{x}, \mathbf{m}^{(k)})\}$.
6. output: label of $y$ as the majority class of $y_{i_1}, \ldots, y_{i_k}$.
```

the feature space. This divides the feature space into decision regions within which points have the same label. The boundary separating these regions is called the *decision boundary* induced by the classifier.

A *multiclass classifier* is a function $H : \mathcal{X} \to \mathcal{Y}$ that maps an instance $\mathbf{x} \in \mathcal{X}$ to an element y of $\mathcal{Y}$.

Consider k-class classifiers of the form [64]

$$H_{\mathbf{W}}(\mathbf{x}) = \underset{m=1,\ldots,k}{\arg\max}\ \mathbf{w}_m^T \mathbf{x}, \tag{6.10.9}$$

where $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_k]$ is an $N \times k$ weight matrix, while the value of the inner-product of the mth column of $\mathbf{W}$ with the instance $\mathbf{x}$ is called the *confidence* or the *similarity score* for the mth class. By this definition, the predicted label of the new instance $\mathbf{x}$ is the index of the column attaining the highest similarity score with $\mathbf{x}$.

For the case of $k = 2$, linear binary classifiers predict that the label of an instance $\mathbf{x}$ is 1 if $\mathbf{w}_1^T \mathbf{x} > 0$ and -1 (or 2) if $\mathbf{w}_2^T \mathbf{x} \le 0$. In this case, the weight matrix $\mathbf{W} = [\mathbf{w}, -\mathbf{w}]$ is an $N \times 2$ matrix. By using the parsimonious model (6.10.9) when $k \ge 3$, Crammer and Singer [64] set the label of a new input instance by choosing the index of the most similar column of $\mathbf{W}$.

The multiclass classifier of Crammer and Singer [64] is to solve the following primal optimization problem with "soft" constraints:

$$\min_{\mathbf{W},\xi} \left\{ \frac{1}{2}\beta \|\mathbf{W}\|_2^2 + \sum_{i=1}^{N} \xi_i \right\} \tag{6.10.10}$$

$$\text{subject to} \quad \mathbf{w}_{y_i}^T \mathbf{x}_i + \delta_{y_i,m} - \mathbf{w}_m^T \mathbf{x}_i \ge 1 - \xi_i,\ \forall i = 1, \ldots, N;\ m = 1, \ldots, k,$$

where $\beta > 0$ is a regularization constant and for $m = y_i$ the above inequality constraints become $\xi_i \ge 0$, while $\delta_{p,q}$ is equal to 1 if $p = q$ and 0 otherwise.

By the Lagrange multiplier method, the dual optimization problem of (6.10.10) is given by

$$\min_{\mathbf{W},\xi,\eta} \Bigg\{\mathcal{L}(\mathbf{W},\xi,\eta) = \frac{1}{2}B\sum_{m=1}^{k}\|\mathbf{w}_m\|_2^2 + \sum_{i=1}^{N}\xi_i + \sum_{i=1}^{N}\sum_{m=1}^{k}\eta_{i,m}\Big[(\mathbf{w}_m - \mathbf{w}_{y_i})^T\mathbf{x}_i - \delta_{y_i,m} + 1 - \xi_i\Big]\Bigg\} \tag{6.10.11}$$

$$\text{subject to} \quad \eta_{i,m} \geq 0, \quad \forall i = 1,\ldots,N;\ m = 1,\ldots,k,$$

where $\eta_{i,m}$ are the Lagrange multipliers.

From $\frac{\partial \mathcal{L}}{\partial \xi_i} = 0$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{w}_m} = \mathbf{0}$ one has, respectively,

$$\sum_{m=1}^{k}\eta_{i,m} = 1, \quad \forall\, i = 1,\ldots,N \tag{6.10.12}$$

and

$$\mathbf{w}_m = \beta^{-1}\left(\sum_{i=1}^{N}(\delta_{y_i,m} - \eta_{i,m})\mathbf{x}_i\right), \quad m = 1,\ldots,r, \tag{6.10.13}$$

which can be rewritten as

$$\mathbf{w}_m = \beta^{-1}\Bigg(\sum_{i:y_i=m}(1 - \eta_{i,m}\mathbf{x}_i) + \sum_{i:y_i\neq m}(-\eta_{i,m})\mathbf{x}_i\Bigg) \tag{6.10.14}$$

for $m = 1,\ldots,r$.

Equations (6.10.12) and (6.10.14) show the following important performance of the linear multiclass classifier of Crammer and Singer [64]:

- Since the set of Lagrange multipliers, $\{\eta_{i,1},\ldots,\eta_{i,k}\}$, satisfies the constraints $\eta_{i,1}\ldots,\eta_{i,k} \geq 0$ and $\sum_m \eta_{i,m} = 1$ in (6.10.12) for each pattern $\mathbf{x}_i$, each set can be viewed as a probability distribution over the labels $\{1,\ldots,k\}$. Under this probabilistic interpretation an example $\mathbf{x}_i$ is a support pattern if and only if its corresponding distribution is not concentrated on the correct label y_i. Therefore, the classifier is constructed using patterns whose labels are uncertain; the rest of the input patterns are ignored.
- The first sum in (6.10.14) is over all patterns that belong to the mth class. This shows that an example $\mathbf{x}_i$ labeled $y_i = m$ is a support pattern only if $\eta_{i,m} = \eta_{i,y_i} < 1$.
- The second sum in (6.10.14) is over the rest of the patterns whose labels are different from m. In this case, an example $\mathbf{x}_i$ is a support pattern only if $\eta_{i,m} > 0$.

6.11 Supervised Tensor Learning (STL)

In many disciplines, more and more problems need three or more subscripts for describing the data. Data with three or more subscripts are referred to as multi-channel data, whose representation is the *tensor*.

With development of modern applications, multi-way higher-order data models have been successfully applied across many fields, which include, among others, social network analysis/Web mining (e.g., [1, 150, 238]), computer vision (e.g., [111, 242, 258]), hyperspectral image [47, 157, 204], medicine and neuroscience [86], multilinear image analysis of face recognition (tensor face) [256], epilepsy tensors [2], Chemistry [36], and so on.

In this section, we focus on supervised tensor learning and tensor learning for regression.

6.11.1 Tensor Algebra Basics

Tensors will be represented by mathcal symbols, such as $\mathcal{T}, \mathcal{A}, \mathcal{X}$, and so on. A tensor represented in an N-way array is called an Nth-order tensor, and is defined as a multilinear function on an N-dimensional Cartesian product vector space, denoted $\mathcal{T} \in \mathbb{K}^{I_1 \times I_2 \times \cdots \times I_N}$, where $\mathbb{K}$ denotes either the real field $\mathbb{R}$ or the complex field $\mathbb{C}$ and where I_n is the number of entries or the dimension in the nth "direction" or mode. For example, for a third-order tensor $\mathcal{T} \in \mathbb{K}^{4\times 2\times 5}$, the first, second, and third modes have dimensions, respectively, equal to 4, 2, and 5. A scalar is a zero-order tensor, a vector is a first-order tensor, and a matrix is a second-order tensor. An Nth-order tensor is a multilinear mapping

$$\mathcal{T}: \mathbb{K}^{I_1} \times \mathbb{K}^{I_2} \times \cdots \times \mathbb{K}^{I_N} \to \mathbb{K}^{I_1 \times I_2 \times \cdots \times I_N}. \tag{6.11.1}$$

Because $\mathcal{T} \in \mathbb{K}^{I_1 \times \cdots \times I_N}$ can be regarded as an Nth-order matrix, *tensor algebra* is essentially *higher-order matrix algebra*.

The following figure shows, respectively, an example of third-order and firth-order tensors (Fig. 6.2).

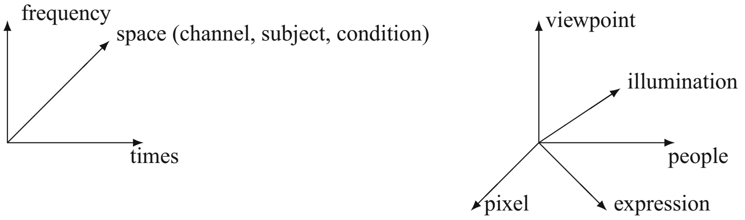

Fig. 6.2 Three-way array (left) and a tensor modeling a face (right)

In linear algebra (i.e., matrix algebra in a finite-dimensional vector space) a linear operator is defined in a finite-dimensional vector space. Because a matrix expresses a second-order tensor, linear algebra can also be regarded as the algebra of second-order tensors. In multiple linear algebra (i.e., higher-order tensor algebra) a multiple linear operator is also defined in a finite-dimensional vector space.

A matrix $\mathbf{A} \in \mathbb{K}^{m\times n}$ is represented by its entries and the symbol $[\cdot]$ as $\mathbf{A} = [a_{ij}]_{i,j=1}^{m,n}$. Similarly, an nth-order tensor $\mathcal{A} \in \mathbb{K}^{I_1\times\cdots\times I_n}$ is, using a dual matrix symbol $[\![\cdot]\!]$, represented as $\mathcal{A} = [\![a_{i_1\cdots i_n}]\!]_{i_1,\ldots,i_n=1}^{I_1,\ldots,I_n}$, where $a_{i_1\cdots i_n}$ is the $(i_1,\ldots,i_n)$th entry of the tensor. An nth-order tensor is sometimes known as an *n-dimensional hypermatrix* [60]. The set of all $(I_1\times\cdots\times I_n)$-dimensional tensors is denoted as $\mathcal{T}(I_1,\ldots,I_n)$.

The most common tensor is a *third-order tensor*, $\mathcal{A} = [\![a_{ijk}]\!]_{i,j,k}^{I,J,K} \in \mathbb{K}^{I\times J\times K}$. A third-order tensor is sometimes called a three-dimensional matrix. A square third-order tensor $\mathcal{X} \in \mathbb{K}^{I\times I\times I}$ is said to be *cubical*. In particular, a cubical tensor is called *supersymmetric tensor* [60], if its entries have the following symmetry:

$$x_{ijk} = x_{ikj} = x_{jik} = x_{jki} = x_{kij} = x_{kji}, \quad \forall\, i,j,k = 1,\ldots,I.$$

For a tensor $\mathcal{A} = [\![a_{i_1\cdots i_m}]\!] \in \mathbb{R}^{N\times\cdots\times N}$, the line connecting $a_{11\cdots 1}$ to $a_{NN\cdots N}$ is called the *superdiagonal line* of the tensor $\mathcal{A}$. A tensor $\mathcal{A}$ is known as *unit tensor*, if all its entries on the superdiagonal line are equal to 1 and all other entries are equal to zero, i.e.,

$$a_{i_1\cdots i_m} = \delta_{i_1\cdots i_m} = \begin{cases} 1, & \text{if } i_1 = \cdots = i_m; \\ 0, & \text{otherwise.} \end{cases} \tag{6.11.2}$$

A third-order unit tensor is denoted as $\mathcal{I} \in \mathbb{R}^{N\times N\times N}$.

In tensor algebra, it is convenient to view a third-order tensor as a set of vectors or matrices. Suppose that the ith entry of a vector $\mathbf{a}$ is denoted by a_i and that a matrix $\mathbf{A} = [a_{ij}] \in \mathbb{K}^{I\times J}$ has I row vectors $\mathbf{a}_{i:}$, $i = 1,\ldots,I$ and J column vectors $\mathbf{a}_{:j}$, $j = 1,\ldots,J$; the ith row vector is denoted by $\mathbf{a}_{i:} = [a_{i1},\ldots,a_{iJ}]$, and the jth column vector is denoted by $\mathbf{a}_{:j} = [a_{1j},\ldots,a_{Ij}]^T$. However, the concepts of row vectors and column vectors are no longer directly applicable for a higher-order tensor.

The three-way arrays of a third-order tensor are not known as row vectors and column vectors, but are renamed *tensor fibers*. A fiber is a one-way array with one subscript that is variable and the other subscripts fixed. The fibers of a third-order tensor are *vertical fibers*, *horizontal fibers*, and *"depth" fiber*. The vertical fibers of the third-order tensor $\mathcal{A} \in \mathbb{K}^{I\times J\times K}$ are also called *column fibers*, denoted $\mathbf{a}_{:jk}$; the horizontal fibers are also known as the *row fibers*, denoted $\mathbf{a}_{i:k}$; the depth fibers are also called the *tube fiber*, denoted $\mathbf{a}_{ij:}$.

Definition 6.18 (Tensor Vectorization) The *tensor vectorization* of an Nth-order tensor $\mathcal{A} = [\![\mathcal{A}_{i_1,\ldots,i_N}]\!] \in \mathbb{K}^{I_1\times\cdots\times I_N}$ is denoted by $\mathbf{a} = \text{vec}(\mathcal{A})$ whose entries are given by

$$a_l = \mathcal{A}_{i_1,i_2,\ldots,i_N}, \quad l = i_1 + \sum_{n=2}^{N}\left((i_n - 1)\prod_{k=1}^{n-1} I_k\right). \tag{6.11.3}$$

For example, $a_1 = \mathcal{A}_{1,1,\ldots,1}, a_{I_1} = \mathcal{A}_{I_1,1,\ldots,1}, a_{I_2} = \mathcal{A}_{1,I_2,1,\ldots,1}, a_{I_1I_2\cdots I_N} = \mathcal{A}_{I_1,I_2,\ldots,I_N}$.

The operation for transforming a tensor into a matrix is known as *tensor matrixing* or *tensor unfolding*.

Definition 6.19 (Tensor Unfolding) Given an Nth-order tensor $\mathcal{A} \in \mathbb{K}^{I_1\times\cdots\times I_N}$, the mappings $\mathcal{A} \to \mathbf{A}_{(n)}$ and $\mathcal{A} \to \mathbf{A}^{(n)}$ are, respectively, said to be the *horizontal unfolding* and *longitudinal unfolding* of the tensor $\mathcal{A}$, where the matrix $\mathbf{A}_{(n)} \in \mathbb{K}^{I_n\times(I_1\cdots I_{n-1}I_{n+1}\cdots I_N)}$ is known as the (mode-n) *horizontal unfolding matrix* of the tensor $\mathcal{A}$; and the matrix $\mathbf{A}^{(n)} = \mathbb{K}^{(I_1\cdots I_{n-1}I_{n+1}\cdots I_N)\times I_n}$ is called the (mode-n) *longitudinal unfolding matrix* of $\mathcal{A}$.

There are three horizontal unfolding methods.

1. *Kiers horizontal unfolding method:* Given an Nth-order tensor $\mathcal{A} \in \mathbb{K}^{I_1\times\cdots\times I_N}$, the Kiers horizontal unfolding method, presented by Kiers [143] in 2000, maps its entry $a_{i_1i_2\cdots i_N}$ to the (i_n, j)th entry of the matrix $\mathbf{A}_{(n)}^{\text{Kiers}} \in \mathbb{K}^{I_n\times(I_1\cdots I_{n-1}I_{n+1}\cdots I_N)}$, i.e.,

$$\mathbf{A}_{(n)}^{\text{Kiers}}(i_n, j) = a_{i_n,j}^{\text{Kiers}} = a_{i_1,i_2,\ldots,i_N}, \tag{6.11.4}$$

where $i_n = 1, \ldots, I_n$ and

$$j = \sum_{p=1}^{N-2}\left(\left(i_{N+n-p} - 1\right)\prod_{q=n+1}^{N+n-p-1} I_q\right) + i_{n+1}, \quad n = 1, \ldots, N \tag{6.11.5}$$

with $I_{N+m} = I_m$ and $i_{N+m} = i_m$ $(m > 0)$.

2. *LMV horizontal unfolding method:* This method, introduced by Lathauwer, Moor, and Vanderwalle [154] in 2000, maps the entry $a_{i_1,\ldots,i_N}$ of an Nth-order tensor $\mathcal{A} \in \mathbb{K}^{I_1\times\cdots\times I_N}$ to the entry $a_{i_n,j}^{\text{LMV}}$ of the matrix $\mathbf{A}_{(n)}^{\text{LMV}} \in \mathbb{K}^{I_n\times(I_1\cdots I_{n-1}I_{n+1}\cdots I_N)}$, where

$$j = i_{n-1} + \sum_{k=1}^{n-2}\left((i_k - 1)\prod_{m=k+1}^{n-1} I_m\right) + \prod_{p=1}^{n-1} I_p\left(\sum_{k=n+1}^{N}\left((i_k - 1)\prod_{q=k}^{N} I_{q+1}\right)\right), \tag{6.11.6}$$

where $I_q = 1$ if $q > N$.

3. *Kolda horizontal unfolding method:* This unfolding, introduced by Kolda [149] in 2006, maps the entry $a_{i_1,\ldots,i_N}$ of an Nth-order tensor to the entry $a_{i_n,j}^{\text{Kolda}}$ of the matrix $\mathbf{A}_{(n)}^{\text{Kolda}} \in \mathbb{K}^{I_n \times (I_1 \cdots I_{n-1} I_{n+1} \cdots I_N)}$, where

$$j = 1 + \sum_{k=1, k\neq n}^{N} \left((i_k - 1) \prod_{m=1, m\neq n}^{k-1} I_m \right). \tag{6.11.7}$$

Similar to the above three horizontal unfolding methods, there are three longitudinal unfolding methods for $\mathbf{A}^{(n)} = \mathbb{K}^{(I_1 \cdots I_{n-1} I_{n+1} \cdots I_N) \times I_n}$. The relationships between the longitudinal unfolding and the horizontal unfolding of a third-order tensor are

$$\mathbf{A}_{\text{Kiers}}^{(n)} = \left(\mathbf{A}_{(n)}^{\text{Kiers}} \right)^T, \quad \mathbf{A}_{\text{LMV}}^{(n)} = \left(\mathbf{A}_{(n)}^{\text{LMV}} \right)^T, \quad \mathbf{A}_{\text{Kolda}}^{(n)} = \left(\mathbf{A}_{(n)}^{\text{Kolda}} \right)^T. \tag{6.11.8}$$

Definition 6.20 (Tensor Inner Product) For two tensors $\mathcal{A}, \mathcal{B} \in \mathbb{K}^{I_1 \times \cdots \times I_N}$, their *tensor inner product* $\langle \mathcal{A}, \mathcal{B} \rangle$ is a scalar and is defined as the inner product of the column vectorizations of the two tensors, i.e.,

$$\begin{aligned} \langle \mathcal{A}, \mathcal{B} \rangle &\overset{\text{def}}{=} \langle \text{vec}(\mathcal{A}), \text{vec}(\mathcal{B}) \rangle = (\text{vec}(\mathcal{A}))^H \text{vec}(\mathcal{B}) \\ &= \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_n=1}^{I_N} a_{i_1 i_2 \cdots i_n}^* b_{i_1 i_2 \cdots i_n}. \end{aligned} \tag{6.11.9}$$

The tensor inner product is also called the *tensor dot product*, denoted by $\mathcal{A} \cdot \mathcal{B}$, namely $\mathcal{A} \cdot \mathcal{B} = \langle \mathcal{A}, \mathcal{B} \rangle$.

Definition 6.21 (Tensor Outer Product) Given two tensors $\mathcal{A} \in \mathbb{K}^{I_1 \times \cdots \times I_P}$ and $\mathcal{B} \in \mathbb{K}^{J_1 \times \cdots \times J_Q}$, their *tensor outer product* is denoted $\mathcal{A} \circ \mathcal{B} \in \mathbb{K}^{I_1 \times \cdots \times I_P \times J_1 \times \cdots \times J_Q}$, and is defined as

$$(\mathcal{A} \circ \mathcal{B})_{i_1 \cdots i_P j_1 \cdots j_Q} = a_{i_1 \cdots i_P} b_{j_1 \cdots j_Q}, \quad \forall i_1, \ldots, i_P, j_1, \ldots, j_Q. \tag{6.11.10}$$

Definition 6.22 (Tensor Frobenius Norm) The *Frobenius norm* of a tensor $\mathcal{A}$ is defined as

$$\|\mathcal{A}\|_F = \sqrt{\langle \mathcal{A}, \mathcal{A} \rangle} \overset{\text{def}}{=} \left(\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_n=1}^{I_N} |a_{i_1 i_2 \cdots i_n}|^2 \right)^{1/2}. \tag{6.11.11}$$

Definition 6.23 (Tucker Operator) Given an Nth-order tensor $\mathcal{G} \in \mathbb{K}^{J_1 \times J_2 \times \cdots \times J_N}$ and matrices $\mathbf{U}^{(n)} \in \mathbb{K}^{I_n \times J_n}$, where $n \in \{1, \ldots, N\}$, the *Tucker operator* is defined as [149]:

$$[\![\mathcal{G}; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!] \overset{\text{def}}{=} \mathcal{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \cdots \times_N \mathbf{U}^{(N)}. \tag{6.11.12}$$

The result is an Nth-order $I_1 \times I_2 \times \cdots \times I_N$ tensor.

The mode-n product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ with a matrix $\mathbf{U} \in \mathbb{R}^{J \times I_n}$, denoted as $\mathcal{X} \times_n \mathbf{U}$, is a tensor of size $I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$, its elements are defined as

$$(\mathcal{X} \times_n \mathbf{U})_{i_1,\ldots,i_{n-1},j,i_{n+1},\ldots,i_N} = \sum_{i_n=1}^{I_n} x_{i_1,\ldots,i_N} u_{j i_n}. \tag{6.11.13}$$

The equivalent unfolded expression is

$$\mathcal{Y} = \mathcal{X} \times_n \mathbf{U} \Leftrightarrow \mathbf{Y}_{(n)} = \mathbf{U}\mathbf{X}_{(n)}. \tag{6.11.14}$$

Especially, for a three-order tensor $\mathcal{X} \in \mathbb{K}^{I_1 \times I_2 \times I_3}$ and the matrices $\mathbf{A} \in \mathbb{K}^{J_1 \times I_1}$, $\mathbf{B} \in \mathbb{K}^{J_2 \times I_2}$, $\mathbf{C} \in \mathbb{K}^{J_3 \times I_3}$, the *Tucker mode-1 product* $\mathcal{X} \times_1 \mathbf{A}$, the *Tucker mode-2 product* $\mathcal{X} \times_2 \mathbf{B}$, and the *Tucker mode-3 product* $\mathcal{X} \times_3 \mathbf{C}$ are, respectively, defined as [155]:

$$(\mathcal{X} \times_1 \mathbf{A})_{j_1 i_2 i_3} = \sum_{i_1=1}^{I_1} x_{i_1 i_2 i_3} a_{j_1 i_1}, \ \forall\, j_1, i_2, i_3, \tag{6.11.15}$$

$$(\mathcal{X} \times_2 \mathbf{B})_{i_1 j_2 i_3} = \sum_{i_2=1}^{I_2} x_{i_1 i_2 i_3} b_{j_2 i_2}, \ \forall\, i_1, j_2, i_3, \tag{6.11.16}$$

$$(\mathcal{X} \times_3 \mathbf{C})_{i_1 i_2 j_3} = \sum_{i_3=1}^{I_3} x_{i_1 i_2 i_3} c_{j_3 i_3}, \ \forall\, i_1, i_2, j_3. \tag{6.11.17}$$

Decompositions of a third-order tensor have two basic forms.

1. *Tucker decomposition:*

$$\mathcal{X} = \mathcal{G} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \ \Leftrightarrow \ x_{ijk} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr}\, a_{ip}\, b_{jq}\, c_{kr}. \tag{6.11.18}$$

Here, $\mathcal{G}$ is called the *core tensor* of tensor decomposition.

2. *Canonical decomposition* (CANDECOM)/*Parallel factors decomposition* (PARFAC), simply known as CP decomposition:

$$\mathcal{X} = \mathcal{I} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \;\Leftrightarrow\; x_{ijk} = \sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{r=1}^{R} a_{ip}\, b_{jq}\, c_{kr}, \tag{6.11.19}$$

where $\mathcal{I}$ is a unit tensor.

It can be easily seen that there are the following differences between the Tucker and CP decompositions:

- In the Tucker decomposition, the entry g_{pqr} of the core tensor $\mathcal{G}$ shows that there are interaction forms involving the entry a_{ip} of the mode-A vector $\mathbf{a}_i = [a_{i1}, \ldots, a_{iP}]^T$, the entry b_{jq} of the mode-B vector $\mathbf{b}_j = [b_{j1}, \ldots, b_{jQ}]^T$, and the entry c_{kr} of the mode-C vector $\mathbf{c}_k = [c_{k1}, \ldots, c_{kR}]^T$.
- In the CP decomposition, the core tensor is a unit tensor. Because the entries on the superdiagonal $p = q = r \in \{1, \ldots, R\}$ are equal to 1, and all the other entries are zero, there are interactions only between the rth factor a_{ir} of the mode-A vector $\mathbf{a}_i$, the rth factor b_{jr} of the mode-B vector $\mathbf{b}_j$, and the rth factor c_{kr} of the mode-C vector $\mathbf{c}_k$. This means that the mode-A, mode-B and mode-C vectors have the same number R of factors, i.e., all modes extract the same number of factors.

Therefore, the CP decomposition can be understood as a canonical Tucker decomposition.

The generalization of the Tucker decomposition to higher-order tensors is called the *higher-order singular value decomposition.*

Theorem 6.6 (Higher-Order SVD [154]) *Every $I_1 \times I_2 \times \cdots \times I_N$ real tensor $\mathcal{X}$ can be decomposed into the mode-n product*

$$\mathcal{X} = \mathcal{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \cdots \times_N \mathbf{U}^{(N)} = [\![\mathcal{G}; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!] \tag{6.11.20}$$

with entries

$$x_{i_1 i_2 \cdots i_N} = \sum_{j_1=1}^{J_1}\sum_{j_2=1}^{J_2}\cdots\sum_{j_N=1}^{J_N} g_{i_1 i_2 \cdots i_N} u^{(1)}_{i_1 j_1} u^{(2)}_{i_2 j_2} \cdots u^{(N)}_{i_N j_N}, \tag{6.11.21}$$

where $\mathbf{U}^{(n)} = [\mathbf{u}_1^{(n)}, \ldots, \mathbf{u}_{J_n}^{(n)}]$ is an $I_n \times J_n$ semi-orthogonal matrix: $(\mathbf{U}^{(n)})^T \mathbf{U}^{(n)} = \mathbf{I}_{J_n}$ with $J_n \le I_n$, the core tensor $\mathcal{G}$ is a $J_1 \times J_2 \times \cdots \times J_N$ tensor and the subtensor $\mathcal{G}_{j_n=\alpha}$ is the tensor $\mathcal{X}$ with the fixed index $j_n = \alpha$. The subtensors have the following properties.

- *All-orthogonality: two subtensors $\mathcal{G}_{j_n=\alpha}$ and $\mathcal{G}_{j_n=\beta}$, for $\alpha \neq \beta$, are orthogonal:*

$$\langle \mathcal{G}_{j_n=\alpha}, \mathcal{G}_{j_n=\beta} \rangle = 0, \quad \forall \alpha \neq \beta,\ n = 1, \ldots, N. \tag{6.11.22}$$

- *Ordering:*

$$\|\mathcal{G}_{i_n=1}\|_F \geq \|\mathcal{G}_{i_n=2}\|_F \geq \cdots \geq \|\mathcal{G}_{i_n=N}\|_F. \tag{6.11.23}$$

The mode-n product is simply denoted as

$$\mathcal{X} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \cdots \times_N \mathbf{U}^{(N)} = \mathcal{X} \prod_{n=1}^{N} \times_n \mathbf{U}^{(n)}. \tag{6.11.24}$$

Note that the typical CP decomposition factorizes usually an N-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ into a linear combination of the R rank-one tensors:

$$\begin{aligned}\mathcal{X} &\approx \sum_{r=1}^{R} \mathbf{u}_r^{(1)} \circ \mathbf{u}_r^{(2)} \circ \cdots \circ \mathbf{u}_r^{(N)} \\ &= [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!].\end{aligned} \tag{6.11.25}$$

This operation is called the *rank-one decomposition* of tensor. Here, the operator "$\circ$" denotes the outer product of vectors of the factor matrices $\mathbf{U}^{(n)} = [\mathbf{u}_1^{(n)}, \ldots, \mathbf{u}_R^{(n)}] \in \mathbb{R}^{I_n \times R}, n = 1, \ldots, N$.

On tensor analysis, readers can further refer to [294].

6.11.2 Supervised Tensor Learning Problems

Given N training tensor-scalar data $(\mathcal{X}_i, y_i), i = 1, \ldots, N$, where the tensor $\mathcal{X}_i \in \mathbb{R}^{I_1, \ldots, I_M}$. The *supervised tensor learning* is to train a parameter/weight tensor $\mathcal{W}$ with rank-one decomposition $\mathcal{W} = \mathbf{w}_1 \circ \mathbf{w}_2 \circ \cdots \circ \mathbf{w}_M$ such that

$$\begin{aligned}(\mathbf{w}_1, \ldots, \mathbf{w}_M; b, \mathbf{x}) &= \underset{\mathbf{w}_1, \ldots, \mathbf{w}_M; b, \mathbf{x}}{\arg\min}\ f(\mathbf{w}_1, \ldots, \mathbf{w}_M; b, \mathbf{x}) \\ \text{subject to }\ y_i c_i &\left(\mathcal{X}_i \prod_{k=1}^{M} \times_k \mathbf{w}_k + b\right) \geq \xi_i, \quad i = 1, \ldots, N.\end{aligned} \tag{6.11.26}$$

If $y_i \in \{+1, -1\}$, then the weight tensor $\mathcal{W}$ is called the *tensor classifier* and if $y_i \in \mathbb{R}$, then $\mathcal{W}$ is known as the *tensor predictor*.

The Lagrangian for supervised tensor learning defined in (6.11.26) is given by

$$\begin{aligned} & L(\mathbf{w}_1, \ldots, \mathbf{w}_M; b, \mathbf{x}) \\ &= f(\mathbf{w}_1, \ldots, \mathbf{w}_M; b, \mathbf{x}) - \sum_{i=1}^{N} \lambda_i \left[y_i c_i \left(\mathcal{X}_i \prod_{k=1}^{M} \times_k \mathbf{w}_k + b \right) - \xi_i \right] \\ &= f(\mathbf{w}_1, \ldots, \mathbf{w}_M; b, \mathbf{x}) - \sum_{i=1}^{N} \lambda_i y_i c_i \left(\mathcal{X}_i \prod_{k=1}^{M} \times_k \mathbf{w}_k + b \right) - \boldsymbol{\lambda}^T \mathbf{x}, \end{aligned} \tag{6.11.27}$$

where $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_N]^T \geq 0$ is a nonnegative Lagrangian multiplier vector and $\mathbf{x} = [\xi_1, \ldots, \xi_N]^T$ is a slack variable vector.

The partial derivative of $L(\mathbf{w}_1, \ldots, \mathbf{w}_M; b, \mathbf{x})$ with respect to $\mathbf{w}_j$ is given by

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}_j} &= \frac{\partial f}{\partial \mathbf{w}_j} - \sum_{i=1}^{N} \lambda_i y_i \frac{\partial c_i}{\partial \mathbf{w}_j} \frac{\partial}{\partial \mathbf{w}_j} \left(\mathcal{X}_i \prod_{k=1}^{M} \times_k \mathbf{w}_k + b \right) \\ &= \frac{\partial f}{\partial \mathbf{w}_j} - \sum_{i=1}^{N} \lambda_i y_i \frac{\mathrm{d} c_i}{\mathrm{d} \mathbf{z}} (\mathcal{X}_i \bar{\times}_j \mathbf{w}_j), \end{aligned} \tag{6.11.28}$$

where $\mathbf{z} = \mathcal{X}_i \prod_{k=1}^{M} \times_k \mathbf{w}_k + b$ and

$$\mathcal{X}_i \bar{\times}_j \mathbf{w}_j = \mathcal{X}_i \circ \mathbf{w}_1 \circ \cdots \circ \mathbf{w}_{j-1} \circ \mathbf{w}_{j+1} \circ \cdots \circ \mathbf{w}_M. \tag{6.11.29}$$

Similarly, the partial derivative of $L(\mathbf{w}_1, \ldots, \mathbf{w}_M; b, \mathbf{x})$ with respect to the bias b is

$$\begin{aligned} \frac{\partial L}{\partial b} &= \frac{\partial f}{\partial b} - \sum_{i=1}^{N} \lambda_i y_i \frac{\partial c_i}{\partial b} \frac{\partial}{\partial b} \left(\mathcal{X}_i \prod_{k=1}^{M} \times_k \mathbf{w}_k + b \right) \\ &= \frac{\partial f}{\partial b} - \sum_{i=1}^{N} \lambda_i y_i \frac{\mathrm{d} c_i}{\mathrm{d} \mathbf{z}} \frac{\partial \mathbf{z}}{\partial b} \frac{\partial}{\partial b} \left(\mathcal{X}_i \prod_{k=1}^{M} \times_k \mathbf{w}_k + b \right) \\ &= \frac{\partial f}{\partial b} - \sum_{i=1}^{N} \lambda_i y_i \frac{\mathrm{d} c_i}{\mathrm{d} \mathbf{z}}. \end{aligned} \tag{6.11.30}$$

Let $\frac{\partial L}{\partial \mathbf{w}_j} = \mathbf{0}$ and $\frac{\partial L}{\partial b} = \mathbf{0}$ then from (6.11.28) and (6.11.30) it is known that

$$\frac{\partial f}{\partial \mathbf{w}_j} = \sum_{i=1}^{N} \lambda_i y_i \frac{\mathrm{d} c_i}{\mathrm{d} \mathbf{z}} (\mathcal{X}_i \bar{\times}_j \mathbf{w}_j), \tag{6.11.31}$$

$$\frac{\partial f}{\partial b} = \sum_{i=1}^{N} \lambda_i y_i \frac{\mathrm{d}c_i}{\mathrm{d}\mathbf{z}}. \tag{6.11.32}$$

Hence, one has the following update formulae [242]:

$$w_{j,t} \leftarrow \mathbf{w}_{j,(t-1)} - \eta_{1,t} \sum_{i=1}^{N} \lambda_i y_i \frac{\mathrm{d}c_i}{\partial \mathbf{z}} (\mathcal{X}_i \bar{\times}_j \mathbf{w}_{j,t-1}), \tag{6.11.33}$$

$$b_t \leftarrow b_{t-1} - \eta_{2,t} \sum_{i=1}^{N} \lambda_i y_i \frac{\mathrm{d}c_i}{\mathrm{d}\mathbf{z}}. \tag{6.11.34}$$

Algorithm 6.17 shows an alternating projection algorithm for the supervised tensor learning [242].

Algorithm 6.17 Alternating projection algorithm for the supervised tensor learning [242]

```
1.  input: Training data $(\mathcal{X}_i, y_i)$, where $\mathcal{X}_i \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $y_i \in \mathbb{R}$ for $i = 1, \ldots, N$.
2.  initialization: Set $\mathbf{w}_k$ equal to random unit vector in $\mathbb{R}^{I_k}$ for $k = 1, \ldots, M$.
3.  repeat
4.    for $j = 1$ to $M$
5.      $w_j^{(t)} \leftarrow \mathbf{w}_j^{(t-1)} - \eta_1 \sum_{i=1}^{N} \lambda_i y_i \frac{\mathrm{d}c_i}{\mathrm{d}\mathbf{z}} (\mathcal{X}_i \bar{\times}_j \mathbf{w}_j)$
6.      $b^{(t)} \leftarrow b^{(t-1)} - \eta_2 \sum_{i=1}^{N} \lambda_i y_i \frac{\mathrm{d}c_i}{\mathrm{d}\mathbf{z}}$
7.    end for
8.    if $\sum_{i=k}^{M} \left( |\mathbf{w}_{k,t}^T \mathbf{w}_{k,t-1}| / \|\mathbf{w}_{k,t}\|_F^2 - 1 \right) \leq \epsilon$, then goto Step 11.
9.    $t \leftarrow t + 1$
10. return
11. output: the parameters in classification tensorplane $\mathbf{w}_1, \ldots, \mathbf{w}_M$ and $b$.
```

6.11.3 Tensor Fisher Discriminant analysis

Fisher discriminant analysis (FDA) [94] is a widely applied method for classification. Suppose there are N training data ($\mathbf{x}_i \in \mathbb{R}^I$, $1 \leq i \leq N$) associated with the class labels $y_i \in \{+1, -1\}$.

Let N_+ and N_- be, respectively, the numbers of the positive training measurements ($\mathbf{x}_i$, $y_i = +1$) and the negative training measurements ($\mathbf{x}_i$, $y_i = -1$). Denote

$$\mathbb{1}(y_i = +1) = \begin{cases} 1, & y_i = +1; \\ 0, & \text{otherwise}; \end{cases} \tag{6.11.35}$$

$$\mathbb{1}(y_i = -1) = \begin{cases} 1, \; y_i = -1; \\ \\ 0, \; \text{otherwise}. \end{cases} \tag{6.11.36}$$

For N_+ positive training measurements $(\mathbf{x}_i, y_i = +1)$, their mean vector is $\mathbf{m}_+ = (1/N_+)\mathbb{1}(y_i = +1)\mathbf{x}_i$; and for N_- negative training measurements $(\mathbf{x}_i, y_i = -1)$, their mean vector can be calculated as $\mathbf{m}_- = (1/N_-)\mathbb{1}(y_i = -1)\mathbf{x}_i$, while the mean vector and the covariance matrix of all training measurements are $\mathbf{m} = (1/N)\sum_{i=1}^N \mathbf{x}_i$ and $\mathbf{\Sigma} = \sum_{i=1}^N (\mathbf{x}_i - \mathbf{m})(\mathbf{x}_i - \mathbf{m})^T$.

The *between-class scatter matrix* $\mathbf{S}_b$ and *within-class scatter matrix* $\mathbf{S}_w$ of two classes of targets $(\mathbf{x}_i, y_i = +1)$ and $(\mathbf{x}_i, y_i = -1)$ are, respectively, defined as

$$\mathbf{S}_b = (\mathbf{m}_+ - \mathbf{m}_-)(\mathbf{m}_+ - \mathbf{m}_-)^T, \tag{6.11.37}$$

$$\mathbf{S}_w = \sum_{i=1}^N (\mathbf{x}_i - \mathbf{m})(\mathbf{x}_i - \mathbf{m})^T = N\mathbf{\Sigma}. \tag{6.11.38}$$

The FDA criterion is to design the classifier $\mathbf{w}$ such that

$$\mathbf{w} = \arg\max_{\mathbf{w}} \left\{ J_{\text{FDA}} = \frac{\mathbf{w}^T \mathbf{S}_b \mathbf{w}}{\mathbf{w}^T \mathbf{S}_w \mathbf{w}} \right\}, \tag{6.11.39}$$

or equivalently

$$\mathbf{w} = \arg\max_{\mathbf{w}} \left\{ J_{\text{FDA}} = \frac{\|\mathbf{m}_+ - \mathbf{m}_-\|}{\sqrt{\mathbf{w}^T \mathbf{\Sigma} \mathbf{w}}} \right\}. \tag{6.11.40}$$

The tensor extension of FDA, called *tensor Fisher discriminant analysis* (TFDA), was proposed by Tao et al. [242], and is a combination of Fisher discriminant analysis and supervised tensor learning.

Given the training tensor measurements $\mathcal{X}_i \in \mathbb{R}^{I_1 \times \cdots \times I_M} (1 = 1, \ldots, N)$ and their corresponding class labels $y_i \in \{+1, -1\}$. The *mean tensor* of the training positive measurements is $\mathcal{M}_+ = (1/N_+)\sum_{i=1}^N \mathbb{1}(y_i = +1)\mathcal{X}_i$; the mean tensor of the training negative measurements is given by $\mathcal{M}_- = (1/N_-)\sum_{i=1}^N \mathbb{1}(y_i = -1)\mathcal{X}_i$; the mean tensor of all training measurements is $\mathcal{M} = (1/N)\sum_{i=1}^N \mathcal{X}_i$.

The TFDA criterion is to design a tensor classifier $\mathcal{W}$ such that

$$(\mathbf{w}_1, \ldots, \mathbf{w}_M) = \arg\max_{\mathbf{w}_1, \ldots, \mathbf{w}_M} \left\{ J_{\text{TFDA}} = \frac{\left\| (\mathcal{M}_+ - \mathcal{M}_-) \prod_{k=1}^M \times_k \mathbf{w}_k \right\|_2^2}{\sum_{i=1}^N \left\| (\mathcal{X}_i - \mathcal{M}) \prod_{k=1}^M \times_k \mathbf{w}_k \right\|_2^2} \right\}. \tag{6.11.41}$$

On more applications of supervised tensor learning, see [242].

6.11.4 Tensor Learning for Regression

Given a set of labeled training set $\{\mathcal{X}_i, y_i\}_{i=1}^N$, where $\mathcal{X}_i \in \mathbb{R}^{I_1\times\cdots\times I_M}$ is an M-mode tensor and y_i are the associated scalar targets.

A classic linear predictor in the vector space, $y = \langle \mathbf{x}, \mathbf{w}\rangle + b$, can be extended from the vector space to the tensor space as

$$\hat{y}_i = \langle \mathcal{X}_i, \mathcal{W}\rangle + b, \tag{6.11.42}$$

where $\mathcal{W} \in \mathbb{R}^{I_1\times\cdots\times I_M}$ is the weight tensor and the scalar b is the bias.

The *tensor regression* seeks to design a weight tensor $\mathcal{W}$ to give the regression output $\hat{y}_i = \langle \mathcal{X}_i, \mathcal{W}\rangle + b$. To this end, let the weight tensor $\mathcal{W}$ is a sum of R rank-one tensors:

$$\mathcal{W} = \sum_{r=1}^R \mathbf{u}_r^{(1)} \circ \mathbf{u}_r^{(2)} \circ \cdots \circ \mathbf{u}_r^{(M)} \triangleq [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(M)}]\!], \tag{6.11.43}$$

where $\mathbf{U}^{(m)} = [\mathbf{u}_1^{(m)}, \ldots, \mathbf{u}_R^{(m)}],\ m = 1, \ldots, M$. Then, the tensor regression can be rewritten as

$$\hat{y}_i = \langle \mathcal{X}_i, \mathcal{W}\rangle + b = \langle \mathcal{X}_i, [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(M)}]\!]\rangle + b. \tag{6.11.44}$$

Therefore, the *higher rank tensor ridge regression* (hrTRR) is to minimize the loss function [111]:

$$\begin{aligned} L(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(M)}; b) = &\frac{1}{2}\sum_{i=1}^N \left(y_i - \left\langle \mathcal{X}_i, [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(M)}]\!]\right\rangle - b\right)^2 \\ &+ \frac{\lambda}{2}\left\| [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(M)}]\!]\right\|_F^2. \end{aligned} \tag{6.11.45}$$

The closed form solution of the optimization $\arg\min_{\mathbf{U}^{(1)},\ldots,\mathbf{U}^{(M)};b} L(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(M)}; b)$ can be derived as [111]:

$$b = \sum_{i=1}^N \left(y_i - \left\langle \mathcal{X}_i, [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(M)}]\!]\right\rangle\right), \tag{6.11.46}$$

and

$$\hat{\mathbf{u}}^{(m)} = \left(\mathbf{\Phi}_{(m)}^T \mathbf{\Phi}_{(m)} + \lambda \mathbf{I}\right)^{-1} \mathbf{\Phi}_{(m)}^T \mathbf{y}, \quad m = 1, \ldots, M, \tag{6.11.47}$$

where $\hat{\mathbf{u}}^{(m)} = [\text{vec}(\hat{\mathbf{U}}^{(m)})^T, b]^T$ is the vector of unknowns, $\mathbf{y} = [y_1, \ldots, y_N]^T$ are targets, and the ith row of the matrix $\boldsymbol{\Phi}_{(m)}$ is $[\text{vec}(\tilde{\mathbf{X}}_{i(m)}), 1]$. Here, the matrix $\tilde{\mathbf{X}}_{i(m)}$ can be constructed as follows:

$$\mathbf{U}^{(-m)} = \mathbf{U}^{(M)} \odot \cdots \odot \mathbf{U}^{(m+1)} \odot \mathbf{U}^{(m-1)} \odot \cdots \odot \mathbf{U}^{(1)}, \tag{6.11.48}$$

$$\mathbf{B}_{(m)} = \mathbf{U}^{(-m)T}\mathbf{U}^{(-m)}, \tag{6.11.49}$$

$$\tilde{\mathbf{X}}_{i(m)} = \mathbf{X}_{i(m)}\mathbf{U}^{(-m)}\mathbf{B}_{(m)}^{1/2}, \tag{6.11.50}$$

where $\mathbf{X}_{i(m)} \in \mathbb{R}^{I_m \times (I_1 \cdots I_{m-1} I_{m+1} \cdot I_M)}$ is the horizontal unfolding matrix based on any of the Kiers method, the LMV method, and the Kolda method.

In the above hrTRR, the Frobenius norm regularization is used, requiring the a priori selection of the tensor rank R. Instead of the Frobenius norm regularization, consider the group sparsity norm regularization [111]:

$$\psi(\mathbf{W}) = \sum_{r=1}^{R} \left(\sum_{m=1}^{M} \|\mathbf{U}_{:,r}^{(m)}\|_2^2 \right)^{1/2}, \tag{6.11.51}$$

where $\mathbf{U}_{:,r}^{(m)}, m = 1, \ldots, M$ denote the rth column of the matrix $\mathbf{U}^{(m)}$.

Lemma 6.3 ([111]) *The group sparsity norm regularization can be equivalently written as*

$$\begin{aligned}\psi(\mathbf{W}) &= \sum_{r=1}^{R} \left(\sum_{m=1}^{M} \|\mathbf{U}^{(m)}\|_2^2 \right)^{1/2} \\ &= \min_{\eta \in \mathbb{R}} \left\{ \frac{1}{2} \sum_{r=1}^{R} \frac{\sum_{m=1}^{M} \|\mathbf{U}_{:,r}^{(m)}\|_2^2}{\eta_r} + \frac{1}{2}\|\boldsymbol{\eta}\|_1 \right\}.\end{aligned} \tag{6.11.52}$$

The minimum is obtained for

$$\eta_r = \left(\sum_{m=1}^{M} \|\mathbf{U}_{:,r}^{(m)}\|_2^2 \right)^{1/2}, \ \forall r = 1, \ldots, R. \tag{6.11.53}$$

Since minimizing $\psi(\mathbf{W})$ contains the minimization of $\|\boldsymbol{\eta}\|_1$, the optimal rank R can be obtained from the sparsity of $\boldsymbol{\eta}$. Then, the *optimal rank tensor ridge regression* (orTRR) $(\hat{\mathbf{U}}^{(m)}, b) = \arg\min_{\mathbf{U}^{(m)}, b} L_m(\mathbf{U}^{(m)}, b)$ can be represented as

$$(\hat{\mathbf{U}}^{(m)}, b) = \arg\min_{\mathbf{U}^{(m)}, b} \left\{ \frac{1}{2} \left(\sum_{i=1}^{N} y_i - \text{tr}\left(\mathbf{U}^{(m)}\mathbf{U}^{(-m)T}\mathbf{X}_{i(m)}^T\right) - b \right)^2 + \frac{\lambda}{2}\text{tr}\left(\mathbf{U}^{(m)}\boldsymbol{\Lambda}\mathbf{U}^{(m)T}\right) \right\}, \tag{6.11.54}$$

where $\mathbf{\Lambda} = \mathbf{Diag}\left(\frac{1}{\eta_1}, \dots, \frac{1}{\eta_R}\right)$.

The closed form solution of the orTRR is given by

$$\hat{b} = \sum_{i=1}^{N} y_i - \text{tr}\left(\mathbf{U}^{(m)}\mathbf{U}^{(-m)T}\mathbf{X}_{i(m)}^T\right), \tag{6.11.55}$$

$$\hat{\mathbf{u}}^{(m)} = \left(\mathbf{\Phi}^T\mathbf{\Phi} + \frac{\lambda}{2}\tilde{\mathbf{\Lambda}}\right)^{-1}\mathbf{\Phi}^T\mathbf{y}, \tag{6.11.56}$$

where $\tilde{\mathbf{\Lambda}} = \mathbf{\Lambda} \otimes \mathbf{I}_{I_m \times I_m}$.

Guo et al. [111] developed a tensor learning algorithm for regression, as shown in Algorithm 6.18.

Algorithm 6.18 Tensor learning algorithm for regression [111]

```
1. input: The set of training tensors and their corresponding targets, that is $\{\mathcal{X}_i, y_i\}_{i=1}^N$.
2. initialization: Construct randomly $\{\mathbf{U}_0^{(1)}, \dots, \mathbf{U}_0^{(M)}\}$, unfolding $\mathcal{X}_i$ to the matrix $\mathbf{X}_{i(m)}$, and
   let $t = 0$.
3. repeat
4.   for $k = 1$ to $M$ do
5.     for high rank tensor ridge regression (hrTRR), calculate
6.       $\mathbf{U}_t^{(-j)} = \mathbf{U}_t^{(M)} \odot \cdots \odot \mathbf{U}_t^{(j+1)} \odot \mathbf{U}_t^{(j-1)} \odot \cdots \odot \mathbf{U}_t^{(1)}$,
7.       $\mathbf{B}_t = \mathbf{U}_t^{(-j)T}\mathbf{U}_t^{(-j)}$,
8.       $\tilde{\mathbf{X}}_{i(j)} = \mathbf{X}_{i(j)}\mathbf{U}_t^{(-j)}\mathbf{B}_t^{1/2}$,
9.       The $i$th row of the matrix $\mathbf{\Phi}_{(m)}$ is given by $[\text{vec}(\tilde{\mathbf{X}}_{i(m)}), 1]$.
10.      Calculate $b = \sum_{i=1}^N \left(y_i - \langle \mathcal{X}_i, [\![\mathbf{U}^{(1)}, \dots, \mathbf{U}^{(M)}]\!]\rangle\right)$.
11.      Calculate $\hat{\mathbf{u}}^{(j)} = (\mathbf{\Phi}_{(m)}^T\mathbf{\Phi}_{(m)} + \lambda\mathbf{I})^{-1}\mathbf{\Phi}_{(m)}^T\mathbf{y}$;
12.    for optimal rank tensor ridge regression (orTRR), calculate
13.      $\eta_r = \left(\sum_{m=1}^M \|\mathbf{U}_{:,r}^{(m)}\|_2^2\right)^{1/2}$, $\forall r = 1, \dots, R$.
14.      $\mathbf{\Lambda} = \mathbf{Diag}\left(\frac{1}{\eta_1}, \dots, \frac{1}{\eta_R}\right)$ and $\tilde{\mathbf{\Lambda}} = \mathbf{\Lambda} \otimes \mathbf{I}_{I_j \times I_j}$.
15.      Calculate $b = \sum_{i=1}^N y_i - \text{tr}\left(\mathbf{U}^{(m)}\mathbf{U}^{(-m)T}\mathbf{X}_{i(m)}^T\right)$.
16.      Calculate $\hat{\mathbf{u}}^{(j)} = (\mathbf{\Phi}^T\mathbf{\Phi} + \frac{\lambda}{2}\tilde{\mathbf{\Lambda}})^{-1}\mathbf{\Phi}^T\mathbf{y}$,
17.  end for
18.  Update the parameter $\eta$ given by $\eta_r = (\sum_{m=1}^M \|\mathbf{U}_{:,r}^{(m)}\|_2^2)^{1/2}$, $r = 1, \dots, R$.
19.  Prune the columns $\mathbf{U}_{:,r}^{(m)}$ of factor matrices for $m \in \{1, \dots, M\}$ and $r \in \{j | \eta_j \leq \epsilon, j =$
     $1, \dots, R\}$.
20.  $\mathcal{W}^t \leftarrow [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \dots, \mathbf{U}^{(M)}]\!]$.
21.  $t \leftarrow t + 1$
22.  until $\|\mathcal{W}^{(t)} - \mathcal{W}^{(t-1)}\| / \|\mathcal{W}^{(t-1)}\| \leq \epsilon$ or $t \geq T_{\max}$.
23. end repeat
24. output: the weights $\{\mathbf{U}^{(1)}, \dots, \mathbf{U}^{(M)}\}$ and the bias term $b \in \mathbb{R}$ that minimize the objective
          function.
```

6.11.5 Tensor K-Means Clustering

For one multi-modal object $\mathbf{x}$ with three modalities, i.e., image modality, text modality, and audio modality, there are three feature vectors: image feature vector $\mathbf{a}$, text feature vector $\mathbf{b}$, and audio feature vector $\mathbf{c}$, after feature learning. Hence, we can build a three-order feature tensor $\mathcal{T} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$ for the object $\mathbf{x}$.

One feature tensor is built for each heterogeneous object after feature fusion. However, the conventional K-means algorithm could cluster the objects represented by vectors, and hence it is necessary to extend the K-means clustering algorithm from feature vectors to feature tensors.

Given two feature tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times I_1 \times \cdots \times I_n}$, the *tensor distance* (TD) between $\mathcal{X}$ and $\mathcal{Y}$, denoted as $d_{\text{TD}}(\mathcal{X}, \mathcal{Y})$, is defined as [37]:

$$d_{\text{TD}}(\mathcal{X}, \mathcal{Y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \mathbf{G} (\mathbf{x} - \mathbf{y})}, \tag{6.11.57}$$

where $\mathbf{x} = \text{vec}(\mathcal{X})$ and $\mathbf{y} = \text{vec}(\mathcal{Y})$ are, respectively, the vectorizations of $\mathcal{X}$ and $\mathcal{Y}$, and $\mathbf{G}$ denotes the coefficient matrix used to reveal the location correlation between $\mathcal{X}$ and $\mathcal{Y}$ in the tensor space.

Therefore, the *tensor K-means clustering* algorithm based on the tensor distance is outlined in the following four steps [37].

1. Randomly select K objects as the clustering centers.
2. Use Eq. (6.11.57) to compute the tensor distance between each object and every clustering center, and assign each object to the nearest clustering center.
3. Recompute each clustering center.
4. If convergent, stop the algorithm, otherwise, return to Step 2.

The key step of the tensor K-means algorithm is to compute the tensor distance between each object and every clustering center. Hence the tensor K-means algorithm has a computational complexity of $O(tnk)$, where t, n, k denote the numbers of iterations, objects, and clustering centers, respectively.

6.12 Unsupervised Clustering

The main purpose of unsupervised learning methods is to extract generally useful features from unlabeled data, to detect and remove input redundancies, and to preserve only essential aspects of the data in robust and discriminative representations [174].

Since no or very little information about the underlying distribution is available, the exploration of complex data sets fundamentally relies on the identification of "natural" group structures in the data. The analysis based on natural group structures in the data is called "*cluster analysis*."

Clustering is one of the most widely used techniques for exploratory data analysis, with applications ranging from statistics, computer science, information science and engineering, biology, medicine to social sciences or psychology.

Clustering criteria can be broadly divided into three fundamental categories [114]:

1. *Compactness:* This concept is generally implemented by keeping the intra-cluster variation small. The resulting methods tend to be very effective for spherical or well-separated clusters, but they may fail to detect more complicated cluster structures.
2. *Connectedness:* Its basic idea is that neighboring data items should share the same cluster. They are well-suited for the detection of arbitrarily shaped clusters, but can lack robustness when there is little spatial separation between the clusters.
3. *Spatial separation:* This is a criterion that gives little guidance during the clustering process and can easily lead to trivial solutions. It is therefore usually combined with other objectives, most notably measures of compactness or balance of cluster sizes.

In every scientific and engineering field dealing with empirical data, one attempts to get a first impression on their data by trying to identify groups of "similar behavior" in their data.

Unsupervised learning includes clustering algorithms, which partition input vectors (a data set covering various dimensions) into clusters satisfying certain criteria. The most popular modern clustering algorithms are hierarchical clustering, spectral clustering, K-means clustering, etc.

In unsupervised learning, we only have the objects and do not have their labels. That is to say, given a set of inputs, an unsupervised learning algorithm is to correctly infer the outputs without having a supervisor providing the correct answers or the degree of error for each observation.

A validation step is needed due to the following two issues that arise when using clustering algorithms [114]:

- Bias of clustering algorithms towards particular cluster properties is inevitable due to their own clustering criterion.
- Unsupervised classification relies on the existence of a distinct structure within the data. However, most clustering algorithms return a clustering even in the absence of actual structure, leaving it to the user to detect the lack of significance of the results returned.

Data clustering has been used for the following three main purposes [127].

- *Underlying structure:* It gains insight into data for generating hypotheses, detecting anomalies, and identifying salient features.
- *Natural classification:* It identifies the degree of similarity among forms or organisms (phylogenetic relationship).
- *Compression:* It is a method for organizing the data and summarizing it through cluster prototypes.

6.12.1 Similarity Measures

One of the main mathematical tools for unsupervised clustering and classification is the distance measure.

The distance between two vectors $\mathbf{p}$ and $\mathbf{g}$, denoted by $D(\mathbf{p}\|\mathbf{g})$, is a *measure* if it has the following properties.

- *Nonnegativity and positiveness:* $D(\mathbf{p}\|\mathbf{g}) \geq 0$, and equality holds if and only if $\mathbf{p} = \mathbf{g}$.
- *Symmetry:* $D(\mathbf{p}\|\mathbf{g}) = D(\mathbf{g}\|\mathbf{p})$.
- *Triangle inequality:* $D(\mathbf{p}\|\mathbf{z}) \leq D(\mathbf{p}\|\mathbf{g}) + D(\mathbf{g}\|\mathbf{z})$.

The basic rule of unsupervised clustering and classification is to adopt some distance metric to measure the similarity of two feature vectors. As the name suggests, this *similarity* is a measure of the degree of similarity between vectors.

Consider a clustering problem in pattern recognition (e.g., template matching). For simplicity, suppose that there are N pattern vectors $\mathbf{s}_1, \ldots, \mathbf{s}_N$, but the number of classes is unknown. Our problem is to cluster N pattern vectors into a number of classes. For this purpose, we need to compare these pattern vectors in order to be clustered into a number of classes such that pattern vectors belonging to the same class are more similar than those belonging to other classes. On the basis of a similarity comparison, we can obtain the unsupervised pattern clustering.

A quantity known as the *dissimilarity* is used to make a reverse measurement of the similarity between vectors: two vectors with a smaller dissimilarity are more similar.

Let $D(\mathbf{s}_i, \mathbf{s}_j)$ $(i, j = 1, \ldots, N$, but $j \neq i)$ be the dissimilarities between the ith and jth pattern vectors $\mathbf{s}_i$ and $\mathbf{s}_j$. If

$$D(\mathbf{s}_i, \mathbf{s}_{j_1}) < D(\mathbf{s}_i, \mathbf{s}_{j_2}), \tag{6.12.1}$$

then we say the pattern vector $\mathbf{s}_{j_1}$ is more similar to $\mathbf{s}_i$ than $\mathbf{s}_{j_2}$.

The simplest and most intuitive dissimilarity parameter is the Euclidean distance between vectors. The regularized Euclidean distance between the ith and the jth known pattern vectors $(\mathbf{s}_i, \mathbf{s}_j)$, denoted $D_E(\mathbf{s}_i, \mathbf{s}_j)$, is defined as

$$D_E(\mathbf{s}_i, \mathbf{s}_j) = \frac{\|\mathbf{s}_i - \mathbf{s}_j\|_2}{\sqrt{\|\mathbf{s}_i\|^2 + \|\mathbf{s}_j\|^2}} = \frac{\sqrt{(\mathbf{s}_i - \mathbf{s}_j)^T(\mathbf{s}_i - \mathbf{s}_j)}}{\sqrt{\|\mathbf{s}_i\|^2 + \|\mathbf{s}_j\|^2}}. \tag{6.12.2}$$

Two extreme values imply that two vectors are completely similar and completely dissimilar, respectively, namely

$$D_E(\mathbf{x}, \mathbf{y}) = \begin{cases} 0 & \Leftrightarrow \quad \mathbf{x} \text{ and } \mathbf{y} \text{ are completely similar,} \\ 1 & \Leftrightarrow \quad \mathbf{x} \text{ and } \mathbf{y} \text{ are completely dissimilar.} \end{cases} \tag{6.12.3}$$

If

$$D_E(\mathbf{s}_1, \mathbf{s}_i) = \min_k \; D_E(\mathbf{s}_1, \mathbf{s}_k), \quad k = 1, \ldots, N, \; i \neq 1, \tag{6.12.4}$$

then $\mathbf{s}_i$ is said to be a *nearest neighbor* to $\mathbf{s}_1$.

A widely used classification method is *nearest neighbor classification*, which judges $\mathbf{s}_1$ and $\mathbf{s}_i$ to belong in the same model type.

All pattern vectors satisfying

$$D_E(\mathbf{s}_1, \mathbf{s}_j) \approx D_E(\mathbf{s}_1, \mathbf{s}_i), \quad j = 2, \ldots, N, \text{ but } j \neq i, \tag{6.12.5}$$

are said to be the nearest neighbors to $\mathbf{s}_1$ as well and thus these pattern vectors are judged to belong to Class 1. Mimicking this process for other unclustered pattern vectors until all vectors are clustered, we can cluster all other possible classes (Class 2, Class 3, and so on) and determine the number of classes. This is the basic idea of the unsupervised clustering approach.

Another frequently used distance function is the *Mahalanobis distance*, proposed by Mahalanobis in 1936 [172]. The Mahalanobis distance from the vector $\mathbf{x}$ to its mean $\boldsymbol{\mu}$ is given by

$$D_M(\mathbf{x}, \boldsymbol{\mu}) = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^T \mathbf{C}_x^{-1} (\mathbf{x} - \boldsymbol{\mu})}, \tag{6.12.6}$$

where $\mathbf{C}_x = \text{cov}(\mathbf{x}, \mathbf{x}) = E\{(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T\}$ is the autocovariance matrix of the vector $\mathbf{x}$.

The Mahalanobis distance between vectors $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^n$ is denoted by $D_M(\mathbf{x}, \mathbf{y})$, and is defined as [172]

$$D_M(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \mathbf{C}_{xy}^{-1} (\mathbf{x} - \mathbf{y})}, \tag{6.12.7}$$

where $\mathbf{C}_{xy} = \text{cov}(\mathbf{x}, \mathbf{y}) = E\{(\mathbf{x} - \boldsymbol{\mu}_x)(\mathbf{y} - \boldsymbol{\mu}_y)^T\}$ is the cross-covariance matrix of $\mathbf{x}$ and $\mathbf{y}$, while $\boldsymbol{\mu}_x$ and $\boldsymbol{\mu}_y$ are the means of $\mathbf{x}$ and $\mathbf{y}$, respectively.

Clearly, if the covariance matrix is the identity matrix, i.e., $\mathbf{C} = \mathbf{I}$, then the Mahalanobis distance reduces to the Euclidean distance. If the covariance matrix takes a diagonal form, then the corresponding Mahalanobis distance is called the *normalized Euclidean distance* and is given by

$$D_M(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} \frac{(x_i - y_i)^2}{\sigma_i^2}}, \tag{6.12.8}$$

in which σ_i is the standard deviation of x_i and y_i in the whole sample set.

Let

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{s}_i, \quad \mathbf{C} = \sum_{i=1}^{N}\sum_{j=1}^{N}(\mathbf{s}_i - \boldsymbol{\mu})(\mathbf{s}_j - \boldsymbol{\mu})^T \tag{6.12.9}$$

be the sample mean vector of N known pattern vectors $\mathbf{s}_i$ and the sample cross-covariance matrix. Then the Mahalanobis distance from the ith pattern vector $\mathbf{s}_i$ to the jth pattern vector $\mathbf{s}_j$ is defined as

$$D_M(\mathbf{s}_i, \mathbf{s}_j) = \sqrt{(\mathbf{s}_i - \mathbf{s}_j)^T\mathbf{C}^{-1}(\mathbf{s}_i - \mathbf{s}_j)}. \tag{6.12.10}$$

By the nearest neighbor classification method, if

$$D_M(\mathbf{s}_1, \mathbf{s}_i) = \min_k \, D_M(\mathbf{s}_1, \mathbf{s}_k), \quad k = 1, \ldots, M, \tag{6.12.11}$$

then the pattern vector $\mathbf{s}_i$ is recognized as being in the pattern type to which $\mathbf{s}_1$ belongs.

The measure of dissimilarity between vectors is not necessarily limited to distance functions. The cosine function of the acute angle between two vectors,

$$D(\mathbf{x}, \mathbf{s}_i) = \cos\theta_i = \frac{\mathbf{x}^T\mathbf{s}_i}{\|\mathbf{x}\|_2\|\mathbf{s}_i\|_2}, \tag{6.12.12}$$

is an effective measure of dissimilarities as well.

If $\cos\theta_i < \cos\theta_j, \forall\, j \neq i$, holds, then the unknown pattern vector $\mathbf{x}$ is said to be most similar to the known pattern vector $\mathbf{s}_i$. The variant of Eq. (6.12.12)

$$D(\mathbf{x}, \mathbf{s}_i) = \frac{\mathbf{x}^T\mathbf{s}_i}{\mathbf{x}^T\mathbf{x} + \mathbf{s}_i^T\mathbf{s}_i + \mathbf{x}^T\mathbf{s}_i} \tag{6.12.13}$$

is referred to as the *Tanimoto measure* [251] and is widely used in information retrieval, the classification of diseases, animal and plant classifications, etc.

Given a testing pattern vector $\mathbf{x}$ and the number m of clustered classes. Define the mean vector of the ith class as

$$\bar{\mathbf{s}}^{(i)} = \frac{1}{N_i}\sum_{k=1}^{N_i}\mathbf{s}_k^{(i)}, \tag{6.12.14}$$

where N_i is the number of pattern vectors in the ith class such that $N_1 + \cdots + N_m = N$ and $\mathbf{s}_k^{(i)}$ is the kth pattern vector in the ith class.

By the nearest neighbor classification, we decide that $\mathbf{x}$ belongs to Class i, if

$$D\left(\mathbf{x}, \bar{\mathbf{s}}^{(i)}\right) = \min_{k=1,\dots,m} D\left(\mathbf{x}, \bar{\mathbf{s}}^{(k)}\right). \tag{6.12.15}$$

The similarity measures used in clustering depend on application problems. In addition to the Euclidean distance, the Mahalanobis distance, and the Tanimoto measure described above, there are various measures. For two objects $\mathbf{x}$ and $\mathbf{y}$, the sum of frequencies with which the object $\mathbf{x}$ was understood as the object $\mathbf{y}$ and $\mathbf{y}$ was understood as $\mathbf{x}$ is defined as [132, 178]

$$s(\mathbf{x}, \mathbf{y}) = \frac{f(\mathbf{x}, \mathbf{y})}{f(\mathbf{x}, \mathbf{x})} + \frac{f(\mathbf{y}, \mathbf{x})}{f(\mathbf{y}, \mathbf{y})}. \tag{6.12.16}$$

This is a useful similarity measure in various applications such as speech analysis, fault diagnosis, etc., where $f(\mathbf{x}, \mathbf{y})$ and $f(\mathbf{y}, \mathbf{x})$, for example, are the frequencies with which the consonant $\mathbf{x}$ was heard as the consonant $\mathbf{y}$ and the consonant $\mathbf{y}$ was heard as the consonant $\mathbf{x}$, respectively.

A similarity value is also called *similarity strength*.

Given N objects $\mathbf{x}_1, \dots, \mathbf{x}_N$. The matrix whose entries are pairwise similarities s_{ij} is called the *similarity matrix*, and is denoted by $\mathbf{S} = [s_{ij}]_{i=1,j=1}^{N,N}$, where

$$s_{ij} = s(\mathbf{x}_i, \mathbf{x}_j), \quad i = 1, \dots, N;\ j = 1, \dots, N, \tag{6.12.17}$$

with $s_{ii} = s(\mathbf{x}_i, \mathbf{x}_i) = 0$.

Similarly, when using the pairwise distances d_{ij} in clustering, the matrix $\mathbf{D}$ with entries

$$d_{ij} = d(\mathbf{x}_i, \mathbf{x}_j), \quad i = 1, \dots, N;\ j = 1, \dots, N, \tag{6.12.18}$$

is known as the *distance matrix*, where $d_{ii} = d(\mathbf{x}_i, \mathbf{x}_i) = 0$.

The similarity matrix is a symmetric and nonnegative matrix:

$$\mathbf{S}^T = \mathbf{S} \quad \text{and} \quad \mathbf{S} \geq 0. \tag{6.12.19}$$

This is because of the similarity symmetry $s(\mathbf{x}_i, \mathbf{x}_j) = s(\mathbf{x}_j, \mathbf{x}_i)$ and the nonnegativity of similarity, $s(\mathbf{x}_i, \mathbf{x}_j) \geq 0,\ \forall\, i, j$.

Similarly, the distance matrix $\mathbf{D}$ is also symmetric and nonnegative.

Similarity assessment usually depends on practical applications. For example, there are the following similarity assessment methods in clinical text recognition [50]:

- *Word similarity:* A vector of words weighted by the term frequency-inverse document frequency (TF-IDF) weighting scheme is used to represent each sentence. Then the cosine similarity between two vectors is calculated as the similarity between the two sentences.

- *Syntax similarity:* Each sentence is parsed by the Stanford parser and the dependency relations derived from the parse tree are used to form the vector. Each dependency relation in the vector is weighted by using the TF-IDF weight scheme based on their counts in the sentence and the corpus. Finally, cosine similarity is computed for each pair of sentences, similar to the method of word similarity.
- *Semantic similarity:* This method calculates semantic similarity between two sentences based on concept similarity: (1) extraction of clinical concepts in each sentence so that each sentence can be represented using a vector of union concepts from the two sentences; and (2) calculation of the similarity between the two sentence vectors of concepts, by measuring similarity scores between any two concepts and computing the cosine similarity of two sentence vectors.
- *Combined similarity:* This method combines all word, syntactic, and semantic information for similarity calculation. First combine words and dependency relations for the same sentence into one vector, and then compute the cosine similarity for each pair of sentences based on the new vectors. The final combined similarity between the two sentences is the average similarity for both the newly computed cosine similarity between word/dependency vectors and the semantic similarity, which can be extracted from task-specific vocabularies such as the Unified Medical Language System (UMLS).

6.12.2 Hierarchical Clustering

In unsupervised clustering based on similarity measures, if the number of objects is large, since there is one similarity value for each pair of objects, the resulting array of similarity measures can be so enormous that the underlying pattern or structure is not evident.

If similarity measures are arranged into different rows according to their "values" or "strength," we have an array of similarity measures, and thus construct a hierarchical system of clustering representations, ranging from the first clustering (top row) to the last clustering (bottom line). The first clustering is the "weak" clustering (i.e., each of N objects is represented as a separate cluster), and the last clustering is the "strong" clustering (i.e., all N objects are grouped together as a single cluster). This is the basic idea of the *hierarchical clustering scheme* (HCS).

To introduce the general notion of a hierarchical clustering scheme, we are given N objects, represented by the integers 1 through N. Moreover, we are also given a sequence of $m + 1$ clusterings, $c_0, c_1, \ldots, c_m$ such that $C = c_1 \cup c_2 \cup \cdots \cup c_m$ and $|C| = m + 1$, and each clustering c_i corresponds to its value of similarity measure, $\alpha_i \in [0, 1]$. Let c_0 be the weak clustering of the N objects, with dissimilarity measure $\alpha_0 = D(\mathbf{x}_i, \mathbf{x}_j) = 0$ for any $i \neq j$ (i.e., any pair of N objects cannot be completely similar), and let c_m be the strong clustering with the smallest dissimilarity value. We require also that the numbers α_i increase, i.e., $\alpha_{i+1} \geq \alpha_i$ for

$i = 1, 2, \ldots, m$, and the clusters "increase" also, where $c_{i+1} > c_i$ means that every cluster in c_{i+1} is the merging (or union) of clusters in c_i. This general arrangement is referred to as a hierarchical clustering scheme (HCS) [132].

The steps of hierarchical clustering methodology are given below [185]:

1. Define decision variables, feasible set, and objective functions.
2. Choose and apply a Pareto optimization algorithm, e.g., NSGA-II.
3. Clustering analysis:
   - *Clustering tendency:* By visual inspection or data projections verify that a hierarchical cluster structure is a reasonable model for the data.
   - *Data scaling:* Remove implicit variable weightings due to relative scales using range scaling.
   - *Proximity:* Select and apply an appropriate similarity measure for the data, here, Euclidean distance.
   - *Choice of algorithm(s):* Consider the assumptions and characteristics of clustering algorithms and select the most suitable algorithm for the application, here, group average linkage.
   - *Application of algorithm:* Apply the selected algorithm and obtain a dendrogram.
   - *Validation:* Examine the results based on application subject matter knowledge, assess the fit to the input data and stability of the cluster structure, and compare the results of multiple algorithms, if used.
4. Represent and use the clusters and structure: if the clustering is reasonable and valid, then examine the divisions in the hierarchy for trade-offs and other information to aid decision making.

Example 6.4 Six objects have the following similarity values:

$$0.04: \quad (3,6) \tag{6.12.20}$$

$$0.08: \quad (3,5),\ (5,6) \tag{6.12.21}$$

$$0.22: \quad (1,3),\ (1,5),\ (1,6),\ (2,4) \tag{6.12.22}$$

$$0.35: \quad (1,2),\ (1,4) \tag{6.12.23}$$

$$> 0.35: \quad (2,3), (2,4), (2,5), (3,4), (4,5), (4,6). \tag{6.12.24}$$

The hierarchical clustering results are

$$0.00: \quad [1],\ [3],\ [5],\ [6],\ [4],\ [2] \tag{6.12.25}$$

$$0.04: \quad [1],\ [3,6],\ [2],\ [4],\ [5] \tag{6.12.26}$$

$$0.08: \quad [3,5,6],\ [1],\ [2],\ [4] \tag{6.12.27}$$

$$0.22: \quad [1,3,5,6],\ [2,4] \tag{6.12.28}$$

$$0.35: \quad [1,2,3,4,5,6]. \tag{6.12.29}$$

Table 6.1 Similarity values and clustering results

Values	Object number					
	1	3	6	5	2	4
0.00	●	●	●	●	●	●
0.04	●	XXXXXXXX		●	●	●
0.08	●	XXXXXXXXXXXXXXX			●	●
0.22	XXXXXXXXXXXXXXXXXXXXXX				●	●
0.35	XXXXXXXXXXXXXXXXXXXXXX				XXXXXXXX	

Table 6.1 shows similarity values and corresponding hierarchical clustering results.

In hierarchical clustering, similarity measures have the following properties [132]:

- $D(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$.
- $D(\mathbf{x}, \mathbf{y}) = D(\mathbf{y}, \mathbf{x})$ for all objects $\mathbf{x}$ and $\mathbf{y}$.
- The ultrametric inequality:

$$D(\mathbf{x}, \mathbf{z}) \leq \max\{D(\mathbf{x}, \mathbf{y}), D(\mathbf{y}, \mathbf{z})\}. \tag{6.12.30}$$

- The triangle inequality:

$$D(\mathbf{x}, \mathbf{z}) \leq D(\mathbf{x}, \mathbf{y}) + D(\mathbf{y}, \mathbf{z}). \tag{6.12.31}$$

Due to the symmetry $D(\mathbf{x}, \mathbf{y}) = D(\mathbf{y}, \mathbf{x})$, so for N objects the hierarchical clustering scheme method needs only computing $N(N-1)/2$ dissimilarities.

The (i, j)th entry of the similarity (or distance) matrix $\mathbf{S}$ is defined by the similarity between the ith and jth objects as follows:

$$s_{ij} \overset{\text{def}}{=} D(\mathbf{x}_i, \mathbf{x}_j). \tag{6.12.32}$$

It should be noted that if all objects are clustered into one class at the mth hierarchical clustering with similarity $\alpha^{(m)}$, then the (i, j)th entries of similarity matrix are given by

$$s_{ij} = \begin{cases} s_{ij}, & s_{ij} \leq \alpha^{(m)}; \\ \alpha^{(m)}, & s_{ij} > \alpha^{(m)}. \end{cases} \tag{6.12.33}$$

In hierarchical clustering, there are different distance matrices corresponding to different clusterings.

Table 6.2 gives the distance matrix before clustering in Example 6.4.

The distance matrix after the first clustering [1], [2], [3, 6], [4], [5] is given in Table 6.3.

Table 6.2 Distance matrix in Example 6.4

d_{ij}	1	2	3	4	5	6
1	0.00	0.35	0.22	0.35	0.22	0.22
2	0.35	0.00	0.35	0.22	0.35	0.35
3	0.22	0.35	0.00	0.35	0.08	0.04
4	0.35	0.22	0.35	0.00	0.35	0.35
5	0.22	0.35	0.08	0.35	0.00	0.08
6	0.22	0.35	0.04	0.35	0.08	0.00

Table 6.3 Distance matrix after first clustering in Example 6.4

d_{ij}	1	2	[3, 6]	4	5
1	0.00	0.35	0.22	0.35	0.22
2	0.35	0.00	0.35	0.22	0.35
[3, 6]	0.22	0.35	0.00	0.35	0.08
4	0.35	0.22	0.35	0.00	0.35
5	0.22	0.35	0.08	0.35	0.00

Table 6.4 Distance matrix after second clustering in Example 6.4

d_{ij}	1	2	[3, 5, 6]	4
1	0.00	0.35	0.22	0.35
2	0.35	0.00	0.35	0.22
[3, 5, 6]	0.22	0.35	0.00	0.35
4	0.35	0.22	0.35	0.00

Table 6.4 shows the distance matrix after the second clustering [1], [2], [3, 5, 6], [4] in Example 6.4.

Mimicking the above processes, we can find the distance matrices after third clustering etc.

Given N objects and a similarity metric $d = D(\mathbf{x}, \mathbf{y})$ satisfying the ultrametric inequality (6.12.30), the minimum method for hierarchical clustering is as follows [132].

1. Make the weak clustering C_0 with $d = 0$.
2. For the given clustering C_{i-1} with the distance matrix defined for all objects or clusters in C_{i-1}, let α_i be the smallest nonzero entry in the distance matrix. Merge the pair of objects and/or clusters with distance α_i, to create C_i of value α_i.
3. Create a new similarity function for C_i in the following manner: if $\mathbf{x}$ and $\mathbf{y}$ are clustered in C_i and not in C_{i-1} (i.e., $d(\mathbf{x}, \mathbf{y}) = \alpha_i$), then define the distance from the cluster $[\mathbf{x}, \mathbf{y}]$ to any third object or cluster, $\mathbf{z}$, by $d([\mathbf{x}, \mathbf{y}], \mathbf{z}) = \min\{d(\mathbf{x}, \mathbf{z}), d(\mathbf{y}, \mathbf{z})\}$. If $\mathbf{x}$ and $\mathbf{y}$ are objects and/or clusters in C_{i-1} not clustered in C_i, then $d(\mathbf{x}, \mathbf{y})$ remains the same. Thus, we obtain a new similarity function d for C_i in this way.
4. Repeat Steps 2 and 3 above until the strong clustering is finally obtained.

If $d([\mathbf{x}, \mathbf{y}], \mathbf{z}) = \min\{d(\mathbf{x}, \mathbf{z}), d(\mathbf{y}, \mathbf{z})\}$ in Step 3 above is replaced by $d([\mathbf{x}, \mathbf{y}], \mathbf{z}) = \max\{d(\mathbf{x}, \mathbf{z}), d(\mathbf{y}, \mathbf{z})\}$, the resulting method is called the maximum method for hierarchical clustering.

6.12.3 Fisher Discriminant Analysis (FDA)

In Sect. 6.11.3, we have presented Fisher discriminant analysis (FDA) for supervised binary classification and its extension to tensor data. In this subsection, we deal with the FDA for unsupervised multiple clustering.

The divergence is a measure of the distance or dissimilarity between two signals and is often used in feature discrimination and evaluation of the effectiveness of class discrimination.

Let Q denote the common dimension of the signal-feature vectors extracted by various methods. Assume that there are c classes of signals; the Fisher class separability measure, called simply the *Fisher measure*, between the ith and the jth classes is used to determine the ranking of feature vectors. Consider the class discrimination of c classes of signals. Let $\mathbf{v}_k^{(l)}$ denote the kth sample feature vector of the lth class of signals, where $l = 1, \ldots, c$ and $k = 1, \ldots, l_K$, where l_K is the number of sample feature vectors in the lth signal class. Under the assumption that the prior probabilities of the random vectors $\mathbf{v}^{(l)} = \mathbf{v}_k^{(l)}$ are the same for $k = 1, \ldots, l_K$ (i.e., equal probabilities), the Fisher measure is defined as

$$m^{(i,j)} = \frac{\sum_{l=i,j}\left[\text{mean}_k\left(\mathbf{v}_k^{(l)}\right) - \text{mean}_l\left(\text{mean}\left(\mathbf{v}_k^{(l)}\right)\right)\right]^2}{\sum_{l=i,j}\text{var}_k\left(\mathbf{v}_k^{(l)}\right)}, \tag{6.12.34}$$

where

- $\text{mean}_k\big(\mathbf{v}_k^{(l)}\big)$ is the mean (centroid) of all signal-feature vectors of the lth class;
- $\text{var}\big(\mathbf{v}_k^{(l)}\big)$ is the variance of all signal-feature vectors of the lth class;
- $\text{mean}_l\big(\text{mean}_k\big(\mathbf{v}_k^{(l)}\big)\big)$ is the total sample centroid of all classes.

As an extension of the Fisher measure, consider the projection of all $Q \times 1$ feature vectors onto the $(c-1)$th dimension of class discrimination space.

Put $N = N_1 + \cdots + N_c$, where N_i represents the number of feature vectors of the ith class signal extracted in the training phase. Suppose that

$$\mathbf{s}_{i,k} = [s_{i,k}(1), \ldots, s_{i,k}(Q)]^T$$

denotes the ith ($Q \times 1$) feature vector obtained by the kth group of observation data of the ith signal in the training phase, while

$$\mathbf{m}_i = [m_i(1), \ldots, m_i(Q)]^T$$

is the sample mean of the ith signal-feature vector, where

$$m_i(q) = \frac{1}{N_i} \sum_{k=1}^{N_i} s_{i,k}(q), \quad i = 1, \ldots, c, \; q = 1, \ldots, Q.$$

Similarly, let

$$\mathbf{m} = [m(1), \ldots, m(Q)]^T$$

denote the ensemble mean of all the signal-feature vectors obtained from all the observation data, where

$$m(q) = \frac{1}{c} \sum_{i=1}^{c} m_i(q), \quad q = 1, \ldots, Q.$$

Using the above vectors, one can define a $Q \times Q$ within-class scatter matrix [83]

$$\mathbf{S}_w \overset{\text{def}}{=} \frac{1}{c} \sum_{i=1}^{c} \left(\frac{1}{N_i} \sum_{k=1}^{N_i} (\mathbf{s}_{i,k} - \mathbf{m}_i)(\mathbf{s}_{i,k} - \mathbf{m}_i)^T \right) \tag{6.12.35}$$

and a $Q \times Q$ between-class scatter matrix [83]

$$\mathbf{S}_b \overset{\text{def}}{=} \frac{1}{c} \sum_{i=1}^{c} (\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^T. \tag{6.12.36}$$

For the optimal class discrimination matrix $\mathbf{U}$ to be determined, define a criterion function

$$J(\mathbf{U}) \overset{\text{def}}{=} \frac{\prod_{\text{diag}} \mathbf{U}^T \mathbf{S}_b \mathbf{U}}{\prod_{\text{diag}} \mathbf{U}^T \mathbf{S}_w \mathbf{U}}, \tag{6.12.37}$$

where $\prod_{\text{diag}} \mathbf{A}$ denotes the product of diagonal entries of the matrix $\mathbf{A}$. As it is a measure of class discrimination ability, J should be maximized. We refer to Span($\mathbf{U}$) as the *class discrimination space*, if

$$\mathbf{U} = \underset{\mathbf{U}\in\mathbb{R}^{Q\times Q}}{\arg\max}\, J(\mathbf{U}) = \frac{\prod\limits_{\text{diag}}\mathbf{U}^T\mathbf{S}_b\mathbf{U}}{\prod\limits_{\text{diag}}\mathbf{U}^T\mathbf{S}_w\mathbf{U}}. \tag{6.12.38}$$

This optimization problem can be equivalently written as

$$[\mathbf{u}_1,\ldots,\mathbf{u}_Q] = \underset{\mathbf{u}_i\in\mathbb{R}^Q}{\arg\max}\left\{\frac{\prod\limits_{i=1}^{Q}\mathbf{u}_i^T\mathbf{S}_b\mathbf{u}_i}{\prod\limits_{i=1}^{Q}\mathbf{u}_i^T\mathbf{S}_w\mathbf{u}_i} = \prod_{i=1}^{Q}\frac{\mathbf{u}_i^T\mathbf{S}_b\mathbf{u}_i}{\mathbf{u}_i^T\mathbf{S}_w\mathbf{u}_i}\right\} \tag{6.12.39}$$

whose solution is given by

$$\mathbf{u}_i = \underset{\mathbf{u}_i\in\mathbb{R}^Q}{\arg\max}\,\frac{\mathbf{u}_i^T\mathbf{S}_b\mathbf{u}_i}{\mathbf{u}_i^T\mathbf{S}_w\mathbf{u}_i}, \quad i=1,\ldots,Q. \tag{6.12.40}$$

This is just the maximization of the generalized Rayleigh quotient. The above equation has a clear physical meaning: the column vectors of the matrix $\mathbf{U}$ constituting the optimal class discrimination subspace should maximize the between-class scatter and minimize the within-class scatter, namely these vectors should maximize the generalized Rayleigh quotient.

For c classes of signals, the optimal class discrimination subspace is $(c-1)$-dimensional. Therefore Eq. (6.12.40) is maximized only for $c-1$ generalized Rayleigh quotients. In other words, it is necessary to solve the generalized eigenvalue problem

$$\mathbf{S}_b\mathbf{u}_i = \lambda_i\mathbf{S}_w\mathbf{u}_i, \quad i=1,2,\ldots,c-1, \tag{6.12.41}$$

for $c-1$ generalized eigenvectors $\mathbf{u}_1,\ldots,\mathbf{u}_{c-1}$. These generalized eigenvectors constitute the $Q\times(c-1)$ matrix

$$\mathbf{U}_{c-1} = [\mathbf{u}_1,\ldots,\mathbf{u}_{c-1}] \tag{6.12.42}$$

whose columns span the optimal class discrimination subspace.

After obtaining the $Q\times(c-1)$ matrix $\mathbf{U}_{c-1}$, we can find its projection onto the optimal class discrimination subspace,

$$\mathbf{y}_{i,k} = \mathbf{U}_{c-1}^T\mathbf{s}_{i,k}, \quad i=1,\ldots,c,\ k=1,\ldots,N_i, \tag{6.12.43}$$

for every signal-feature vector obtained in the training phase.

When there are only three classes of signals ($c = 3$), the optimal class discrimination subspace is a plane, and the projection of every feature vector onto the optimal class discrimination subspace is a point. These projection points directly reflect the discrimination ability of different feature vectors in signal classification.

6.12.4 K-Means Clustering

Clustering technique may broadly be divided into two categories: hierarchical and non-hierarchical [6]. As one of the more widely used non-hierarchical clustering techniques, the K-means [251] aims to minimize the sum of squared Euclidean distance between patters and cluster centers via an iterative hill climbing algorithm.

Given a set of N data points in d-dimensional vector space $\mathbb{R}^d$, consider partitioning these points into a number (say $K \leq N$) of groups (or clusters) based on some similarity/dissimilarity metric which establishes a rule for assigning patterns to the domain of a particular cluster center.

Let the set of N data be $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and the K clusters be $C_1, \ldots, C_K$. Then the clustering is to make K clusters satisfy the following conditions [186]:

$$\begin{aligned} &C_i \notin \emptyset, \quad \text{for } i = 1, \ldots, K; \\ &C_i \cap C_j = \emptyset, \quad \text{for } i = 1, \ldots, K;\ j = 1, \ldots, K \text{ and } i \neq j; \\ &\textstyle\bigcup_{i=1}^K C_i = S. \end{aligned}$$

Here each observation vector belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

Given an open set $S \in \mathbb{R}^d$, the set $\{V_i\}_{i=1}^K$ is called a *tessellation* of S if $V_i \cap V_j = \emptyset$ for $i \neq j$ and $\bigcup_{i=1}^K V_i = S$. When a set of points $\{\mathbf{z}_i\}_{i=1}^K$ is given, the sets $\hat{V}_i$ corresponding to the point $\mathbf{z}_i$ are defined by

$$\hat{V}_i = \left\{\mathbf{x} \in S \,\middle|\, \|\mathbf{x} - \mathbf{z}_i\| \leq \|\mathbf{x} - \mathbf{z}_j\| \text{ for } j = 1, \ldots, K,\ j \neq i\right\}. \tag{6.12.44}$$

The points $\{\mathbf{z}_i\}_{i=1}^K$ are called *generators*. The set $\{\hat{V}_i\}_{i=1}^K$ is a *Voronoi tessellation* or *Voronoi diagram* of S, and each $\hat{V}_i$ is known as the *Voronoi set* corresponding to the point $\mathbf{z}_i$.

The K-means algorithm requires three user-specified parameters: number of clusters K, cluster initialization, and distance metric [127, 245].

- *Number of clusters K:* The most critical parameter is K. As there are no perfect mathematical criterion available, the K-means algorithm is usually run independently for different values of K. The value of K is then selected using the partition that appears the most meaningful to the domain expert.

- *Cluster initialization:* To overcome the local minima of K-means clustering, the K-means algorithm selects K initial cluster centers $\mathbf{m}_1, \ldots, \mathbf{m}_K$ randomly from the N data points $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$.
- *Distance metric:* Let d_{ik} denote the distance between data $\mathbf{x}_i = [x_{i1}, \ldots, x_{iN}]^T$ and $\mathbf{x}_k = [x_{k1}, \ldots, x_{kN}]^T$. K-means is typically used with the Euclidean distance $d_{ik} = \sum_j (x_{ik} - x_{jk})^2$ or Mahalanobis distance metric for computing the distance between points $\mathbf{x}_i, \mathbf{x}_k$ and cluster centers.

The most common algorithm uses an iterative refinement technique for finding a locally minimal solution without computing the Voronoi set $\hat{V}_i$. Due to its ubiquity it is often called the *K-means algorithm*. However, the K-means algorithm, as a greedy algorithm, can only converge to a local minimum.

Given any set Z of K centers, for each center $\mathbf{z} \in Z$, let $V(\mathbf{z})$ denote its neighborhood, namely the set of data points for which $\mathbf{z}$ is the nearest neighbor. Based on an elegant random sequential sampling method, the random K-means algorithm, or simply the K-means algorithm, does not require the calculation of Voronoi sets.

Algorithm 6.19 shows the *random K-means algorithm* [15, 81].

Algorithm 6.19 Random K-means algorithm [15, 81]

```
1. input: Dataset $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with $\mathbf{x}_i \in \mathbb{R}^d$, a positive integer $K$.
2. initialization:
   2.1 Select an initial set of $K$ centers $Z = \{\mathbf{z}_1, \ldots, \mathbf{z}_K\}$, e.g., by using a Monte Carlo method;
   2.2 Set $j_i = 1$ for $i = 1, \ldots, K$.
3. repeat
4.    Select a data point $\mathbf{x} \in S$ at random, according to the probability density function $\rho(x)$.
5.    Find the $\mathbf{z}_i \in \mathbb{R}^d$ that is closest to $\mathbf{x}$, for example, use
          $\mathbf{z}_i = \arg\min_{\mathbf{z}_1,\ldots,\mathbf{z}_N} \{\|\mathbf{x} - \mathbf{z}_1\|^2, \ldots, \|\mathbf{x} - \mathbf{z}_K\|^2\}$,
      denote the index of that $\mathbf{z}_i$ by $i^*$. Ties are resolved arbitrarily.
6.    Set
          $\mathbf{z}_{i^*} \leftarrow \frac{j_{i^*}\mathbf{z}_{i^*} + \mathbf{x}}{j_{i^*} + 1}$   and   $j_{i^*} \leftarrow j_{i^*} + 1$,
      this new $\mathbf{z}_{i^*}$, along with the unchanged $\mathbf{z}_i, i \neq i^*$, forms the new set of cluster centers
      $\mathbf{z}_1, \ldots, \mathbf{z}_K$.
7.    Assign point $\mathbf{x}_i, i = 1, \ldots, n$ to cluster $C_j,\ j \in \{1, \ldots, K\}$ if and only if
          $\|\mathbf{x}_i - \mathbf{z}_j\| \leq \|\mathbf{x}_i - \mathbf{z}_p\|, \quad p = 1, \ldots, K;$ and $j \neq p$.
8.    exit if $K$ cluster centers $\mathbf{z}_1, \ldots, \mathbf{z}_K$ are converged.
9. return
10. output: $K$ cluster centroids $\mathbf{z}_1, \ldots, \mathbf{z}_K$.
```

Once the K cluster centers $\mathbf{z}_1, \ldots, \mathbf{z}_K$ are obtained, we can perform the classification of the new testing data point $\mathbf{y} \in \mathbb{R}^d$ by

$$\text{class of } \mathbf{y} = \text{order number of } \mathbf{z}_k = \min\{d(\mathbf{y}, \mathbf{z}_1), \ldots, d(\mathbf{y}, \mathbf{z}_K)\} \tag{6.12.45}$$

where $d(\mathbf{y}, \mathbf{z})$ is the distance metric.

The classic K-means clustering algorithm finds cluster centers that minimize the distance between data points and the nearest center. K-means can be viewed as a way of constructing a "dictionary" $\mathbf{D} \in \mathbb{R}^{n\times k}$ of k vectors so that a data vector $\mathbf{x}_i \in \mathbb{R}^n, i = 1, \ldots, m$ can be mapped to a code vector $\mathbf{s}_i$ that minimizes the error in reconstruction [56].

It is known [56] that initialization and pre-processing of the inputs are useful and necessary for K-means clustering.

1. Though it is common to initialize K-means to randomly chosen examples drawn from the data, this has been found to be a poor choice. Instead, it is better to randomly initialize the centroids from a normal distribution and then normalize them to unit length.
2. The centroids tend to be biased by the correlated data, and thought whitening input data, obtained centroids are more orthogonal.

After initializing and whitening inputs, a modified version of K-means, called *gain shape vector quantization* [290] or *spherical K-means* [75], is applied to find the dictionary $\mathbf{D}$.

Including the initialization and pre-processing, the full K-means training are as follows.

1. Normalize inputs:

$$\mathbf{x}_i \equiv \frac{\mathbf{x}_i - \text{mean}(\mathbf{x}_i)}{\text{var}(\mathbf{x}_i) + \epsilon_{\text{norm}}}, \quad \forall\, i, \tag{6.12.46}$$

   where mean$(\mathbf{x}_i)$ and var$(\mathbf{x}_i)$ are the mean and variance of the elements of $\mathbf{x}_i$, respectively.
2. Whiten inputs:

$$\text{cov}(\mathbf{x}) = \mathbf{V}\mathbf{D}\mathbf{V}^T \tag{6.12.47}$$

$$\mathbf{x}_i = \mathbf{V}(\mathbf{D} + \epsilon)^{-1/2}\mathbf{V}^T\mathbf{x}_i, \quad \forall\, i. \tag{6.12.48}$$

3. Loop until convergence (typically 10 iterations is enough):

$$s_i(j) = \begin{cases} \mathbf{d}_j^T\mathbf{x}_i, & \text{if } j = \arg\max\limits_{l} |\mathbf{d}_l^T\mathbf{x}_i|; \\ 0, & \text{otherwise.} \end{cases} \quad (\forall\, j, i) \tag{6.12.49}$$

$$\mathbf{D} = \mathbf{X}\mathbf{S}^T + \mathbf{D}, \tag{6.12.50}$$

$$\mathbf{d}_j = \mathbf{d}_j / \|\mathbf{d}_j\|_2, \quad \forall\, j. \tag{6.12.51}$$

The columns $\mathbf{d}_j$ of the dictionary $\mathbf{D}$ give the centroids, and K-means tends to learn low-frequency edge-like centroids.

This procedure of normalization, whitening, and K-means clustering is an effective "off the shelf" unsupervised learning module that can serve in many feature-learning roles [56].

6.13 Spectral Clustering

Spectral clustering is one of the most popular modern clustering algorithms, and has been extensively studied in the image processing, data mining, and machine learning communities, see, e.g., [190, 230, 257].

In multivariate statistics and the clustering of data, spectral clustering techniques make use of the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions. The similarity matrix is provided as an input and consists of a quantitative assessment of the relative similarity of each pair of points in the data set.

6.13.1 Spectral Clustering Algorithms

It is assumed that data consists of N "points" $\mathbf{x}_1, \ldots, \mathbf{x}_N$ which can be arbitrary objects. Let their pairwise similarities $s_{ij} = s(\mathbf{x}_i, \mathbf{x}_j)$ be measured by using some similarity function which is symmetric and nonnegative, and denote the corresponding similarity matrix by $\mathbf{S} = [s_{ij}]_{i,j=1}^{N,N}$.

Two popular spectral clustering algorithms [190, 230] are based on spectral graph partitioning. A promising alternative is to use matrix algebra approach.

Let $\mathbf{h}_k$ be a cluster that partitions the mapped data $\mathbf{\Phi} = [\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_N)]$, producing the output $\mathbf{\Phi h}_k$. Maximize the output energy to get

$$\max \left\{ \|\mathbf{\Phi h}_k\|_2^2 = \mathbf{h}_k^T \mathbf{\Phi}^T \mathbf{\Phi h}_k = \mathbf{h}_k^T \mathbf{W h}_k \right\},$$

where

$$\mathbf{W} = \mathbf{\Phi}^T \mathbf{\Phi} = [K(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^{N,N} = \left[\boldsymbol{\phi}^T(\mathbf{x}_i)\boldsymbol{\phi}(\mathbf{x}_j)\right]_{i,j=1}^{N,N}. \tag{6.13.1}$$

In spectral clustering, due to $K(\mathbf{x}_i, \mathbf{x}_j) = s(\mathbf{x}_i, \mathbf{x}_j)$, the affinity matrix $\mathbf{W}$ becomes the similarity matrix $\mathbf{S}$, i.e., $\mathbf{W} = \mathbf{S}$.

For K clusters we maximize the total output energy

$$\max_{\mathbf{h}_1, \ldots, \mathbf{h}_K} \left\{ \sum_{k=1}^{K} \mathbf{h}_k^T \mathbf{W h}_k \right\} \tag{6.13.2}$$

$$\text{subject to} \quad \mathbf{h}_k^T \mathbf{h}_k = 1, \ \forall k = 1, \ldots, K, \tag{6.13.3}$$

to get the primal constrained optimization problem:

$$\max_{\mathbf{H}=[\mathbf{h}_1,\ldots,\mathbf{h}_K]} \quad \mathrm{tr}(\mathbf{H}^T\mathbf{W}\mathbf{H}) \tag{6.13.4}$$

$$\text{subject to} \quad \mathbf{H}^T\mathbf{H}=\mathbf{I}. \tag{6.13.5}$$

By using the Lagrange multiplier method, the above constrained optimization can be rewritten as the dual unconstrained optimization:

$$\max_{\mathbf{H}} \left\{ \mathcal{L}(\mathbf{H}) = \mathrm{tr}(\mathbf{H}^T\mathbf{W}\mathbf{H}) + \lambda\left(1 - \|\mathbf{H}\|_2^2\right)\right\}. \tag{6.13.6}$$

From the first-order optimality condition we have

$$\frac{\partial \mathcal{L}}{\partial \mathbf{H}} = \mathbf{O} \quad \Rightarrow \quad 2\mathbf{W}\mathbf{H} - 2\lambda\mathbf{H} = \mathbf{O} \quad \Rightarrow \quad \mathbf{W}\mathbf{H} = \lambda\mathbf{H} \tag{6.13.7}$$

due to the symmetry of the affinity matrix $\mathbf{W} = \mathbf{S}$. That is, $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_K]$ consists of the K leading eigenvectors (with the largest eigenvalues) of $\mathbf{W}$ as columns.

If the affinity matrix $\mathbf{W}$ is replaced by the normalized Laplacian matrix $\mathbf{L}_{\text{sys}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$, then the K spectral clusters $\mathbf{h}_1, \ldots, \mathbf{h}_K$ are given directly by the K leading eigenvectors. This is just the key point of the normalized spectral clustering algorithm developed by Ng et al. [190], see Algorithm 6.20.

Algorithm 6.20 Normalized spectral clustering algorithm of Ng et al. [190]

```
1. input: Dataset $V = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with $\mathbf{x}_i \in \mathbb{R}^n$, number $k$ of clusters, kernel function
   $K(\mathbf{x}_i, \mathbf{x}_j) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$.
2. Construct the $N \times N$ weighted adjacency matrix $\mathbf{W} = [\delta_{ij} K(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^{N,N}$.
3. Compute the $N \times N$ diagonal matrix $[\mathbf{D}]_{ij} = \delta_{ij} \sum_{j=1}^N w_{ij}$.
4. Compute the $N \times N$ normalized Laplacian matrix $\mathbf{L}_{\text{sys}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$.
5. Compute the first $k$ eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_k$ of the eigenvalue problem $\mathbf{L}_{\text{sys}}\mathbf{u} = \lambda\mathbf{u}$.
6. Let the matrix $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_k] \in \mathbb{R}^{N\times k}$.
7. Compute the $(i, j)$th entries of the $N \times k$ matrix $\mathbf{T} = [t_{ij}]_{i,j=1}^{N,k}$ as $t_{ij} = u_{ij}/(\sum_{m=1}^k u_{im}^2)^{1/2}$.
   Let $\mathbf{y}_i = [t_{i1}, \ldots, t_{ik}]^T \in \mathbb{R}^k$, $i = 1, \ldots, N$.
8. Cluster the points $\mathbf{y}_1, \ldots, \mathbf{y}_N$ via the K-means algorithm into clusters $C_1, \ldots, C_k$.
9. output: clusters $A_1, \ldots, A_k$ with $A_i = \{j | \mathbf{y}_j \in C_i\}$.
```

Interestingly, if letting $\tilde{\mathbf{H}} = \mathbf{D}^{1/2}\mathbf{H}$ and $\tilde{\mathbf{W}} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$, then the solution to the optimization problem

$$\max_{\mathbf{H}} \left\{ \mathcal{L}(\tilde{\mathbf{H}}) = \mathrm{tr}(\tilde{\mathbf{H}}^T\tilde{\mathbf{W}}\tilde{\mathbf{H}}) + \lambda\left(1 - \|\tilde{\mathbf{H}}\|_2^2\right)\right\} \tag{6.13.8}$$

is given by the eigenproblem $\tilde{\mathbf{W}}\tilde{\mathbf{H}} = \lambda\tilde{\mathbf{H}}$ which, by using $\tilde{\mathbf{H}} = \mathbf{D}^{1/2}\mathbf{H}$ and $\tilde{\mathbf{W}} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$, becomes the generalized eigenproblem $\mathbf{W}\mathbf{H} = \lambda\mathbf{D}\mathbf{H}$. Therefore, if the affinity matrix $\mathbf{W}$ is replaced by the normalized Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{W}$, then the K spectral clusters $\mathbf{h}_1, \ldots, \mathbf{h}_K$ consist of the K leading generalized eigenvectors (corresponding to the largest generalized eigenvalues) of the matrix pencil $(\mathbf{L}_{\text{sys}}, \mathbf{D})$. Therefore, we get the normalized spectral clustering algorithm of Shi and Malik [230], as shown in Algorithm 6.21.

Algorithm 6.21 Normalized spectral clustering algorithm of Shi and Malik [230]

```
1. input: Dataset $V = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with $\mathbf{x}_i \in \mathbb{R}^n$, number $k$ of clusters, kernel function
   $K(\mathbf{x}_i, \mathbf{x}_j) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$.
2. Construct the $N \times N$ weighted adjacency matrix $\mathbf{W} = [\delta_{ij} K(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^{N,N}$.
3. Compute the $N \times N$ diagonal matrix $[\mathbf{D}]_{ij} = \delta_{ij} \sum_{j=1}^{N} w_{ij}$.
4. Compute the $N \times N$ normalized Laplacian matrix $\mathbf{L}_{\text{sys}} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-1/2}$.
5. Compute the first $k$ generalized eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_k$ of the generalized eigenproblem
   $\mathbf{L}_{\text{sys}}\mathbf{u} = \lambda\mathbf{D}\mathbf{u}$.
6. Let the matrix $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_k] \in \mathbb{R}^{N \times k}$.
7. Compute the $(i, j)$th entries of the $N \times k$ matrix $\mathbf{T} = [t_{ij}]_{i,j=1}^{N,k}$ as $t_{ij} = u_{ij}/(\sum_{m=1}^{k} u_{im}^2)^{1/2}$.
   Let $\mathbf{y}_i = [t_{i1}, \ldots, t_{ik}]^T \in \mathbb{R}^k$, $i = 1, \ldots, N$.
8. Cluster the points $\mathbf{y}_1, \ldots, \mathbf{y}_N$ via the K-means algorithm into clusters $C_1, \ldots, C_k$.
9. output: clusters $A_1, \ldots, A_k$ with $A_i = \{j | \mathbf{y}_j \in C_i\}$.
```

Remark 1 Both Algorithms 6.20 and 6.21 are normalized spectral clustering algorithms, but different normalized Laplacian matrices are used: $\mathbf{L}_{\text{sys}} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-1/2}$ in Algorithm 6.21, while $\mathbf{L}_{\text{sys}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ in Algorithm 6.20.

Remark 2 If using the unnormalized Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{W}$ instead of the normalized Laplacian matrices $\mathbf{L}_{\text{sys}}$ in Step 3 in the algorithms above described, then we get the unnormalized spectral clustering algorithms.

Remark 3 In practical computation of the "ideal" weighted adjacency matrix $\mathbf{W}$, there is a perturbation to $\mathbf{A}$ that results in estimation errors of either generalized eigenvectors in Algorithm 6.21 or eigenvectors in Algorithm 6.20, i.e., there is a perturbed version $\hat{\mathbf{T}} = \mathbf{T} + \mathbf{E}$, where $\mathbf{E}$ is a perturbation matrix to the "ideal" principal (generalized) eigenvector matrix $\mathbf{T}$.

Spectral clustering techniques are widely used, due to their simplicity and empirical performance advantages compared to other clustering methods, such as hierarchical clustering and K-means clustering and so on.

In recent years, spectral clustering has gained new promotion and development. In the next two subsections, we introduce the constrained spectral clustering and the fast spectral clustering, respectively.

6.13.2 Constrained Spectral Clustering

Both K-means and hierarchical clustering are constrained clustering, but an intractable problem is how to satisfy many constraints in these algorithmic settings. Alternative to encode many constraints in clustering is to use spectral clustering.

To help spectral clustering recover from an undesirable partition, side information in various forms can be introduced, in either small or large amounts [259]:

- *Pairwise constraints:* In some cases a pair of instances must be in the same cluster (*Must-Link*), while a pair of instances cannot be in the same cluster (*Cannot-Link*).
- *Partial labeling:* There can be labels on some of the instances, which are neither complete nor exhaustive.
- *Alternative weak distance metrics:* In some situations there may be more than one distance metrics available.
- *Transfer of knowledge:* When the graph Laplacian is treated as the target domain, it can be viewed as the source domain to transfer knowledge from a different but related graph.

Consider encoding side information, in k-class clustering, with an $N \times N$ constraint matrix $\mathbf{Q}$ defined as [259]:

$$Q_{ij} = Q_{ji} = \begin{cases} +1, & \text{if nodes } \mathbf{x}_i, \mathbf{x}_j \text{ belong to same cluster;} \\ -1, & \text{if nodes } \mathbf{x}_i, \mathbf{x}_j \text{ belong to different clusters;} \\ 0, & \text{no side information available.} \end{cases} \tag{6.13.9}$$

Define the *cluster indicator vector* $\mathbf{u} = [u_1, \ldots, u_N]^T$ as

$$u_i = \begin{cases} +1, & \text{if } \mathbf{x}_i \text{ belongs to cluster } i; \\ -1, & \text{if } \mathbf{x}_i \text{ does not belong to cluster } i; \end{cases} \tag{6.13.10}$$

for $i = 1, \ldots, N$. Therefore,

$$\mathbf{u}^T \mathbf{Q} \mathbf{u} = \sum_{i=1}^{N} \sum_{j=1}^{N} u_i u_j Q_{ij} \tag{6.13.11}$$

becomes a real-valued measure of how well the constraints in $\mathbf{Q}$ are satisfied in the relaxed sense. The larger $\mathbf{u}^T \mathbf{Q} \mathbf{u}$ is, the better the cluster assignment $\mathbf{u}$ conforms to the given constraints in $\mathbf{Q}$.

Let constant $\alpha \in \mathbb{R}$ be a lower-bound of $\mathbf{u}^T\mathbf{Q}\mathbf{u}$, i.e., $\mathbf{u}^T\mathbf{Q}\mathbf{u} \geq \alpha$. If substituting $\mathbf{u}$ with $\mathbf{D}^{-1/2}\mathbf{v}$ (where $\mathbf{D}$ is a degree matrix), then this inequality constraint becomes

$$\mathbf{v}^T\bar{\mathbf{Q}}\mathbf{v} \geq \alpha, \tag{6.13.12}$$

where $\bar{\mathbf{Q}} = \mathbf{D}^{-1/2}\mathbf{Q}\mathbf{D}^{-1/2}$ is the *normalized constraint matrix*.

Therefore, the *constrained spectral clustering* can be written as a constrained optimization problem [259]:

$$\min_{\mathbf{v}\in\mathbb{R}^N} \ \mathbf{v}^T\bar{\mathbf{L}}\mathbf{v} \quad \text{subject to } \mathbf{v}^T\bar{\mathbf{Q}}\mathbf{v} \geq \alpha,\ \mathbf{v}^T\mathbf{v} = \text{vol},\ \mathbf{v} \neq \mathbf{D}^{1/2}\mathbf{1}, \tag{6.13.13}$$

where $\bar{\mathbf{L}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$ and $\text{vol} = \sum_{i=1}^N D_{ii}$ is the volume of graph $G(V, E)$. The second constraint $\mathbf{v}^T\mathbf{v} = \text{vol}$ normalizes $\mathbf{v}$; the third constraint $\mathbf{v} \neq \mathbf{D}^{1/2}\mathbf{1}$ rules out the trivial solution $\mathbf{D}^{1/2}\mathbf{1}$. By the Lagrange multiplier method, one gets the dual spectral clustering problem:

$$\min_{\mathbf{v}\in\mathbb{R}^N} \left\{\mathcal{L}_{\text{D}} = \mathbf{v}^T\bar{\mathbf{L}}\mathbf{v} + \lambda\big(\alpha - \mathbf{v}^T\bar{\mathbf{Q}}\mathbf{v}\big) + \mu\big(\text{vol} - \mathbf{v}^T\mathbf{v}\big)\right\}. \tag{6.13.14}$$

From the stationary point condition, we have

$$\frac{\partial \mathcal{L}_{\text{D}}}{\partial \mathbf{v}} = \mathbf{0} \quad \Rightarrow \quad \bar{\mathbf{L}}\mathbf{v} - \lambda\bar{\mathbf{Q}}\mathbf{v} - \mu\mathbf{v} = \mathbf{0}. \tag{6.13.15}$$

Letting the auxiliary variable $\beta = -\frac{\mu}{\lambda}\text{vol}$ then the stationary point condition is

$$\bar{\mathbf{L}}\mathbf{v} - \lambda\bar{\mathbf{Q}}\mathbf{v} + \frac{\lambda\beta}{\text{vol}}\mathbf{v} = \mathbf{0} \quad \text{or} \quad \bar{\mathbf{L}}\mathbf{v} = \lambda\left(\bar{\mathbf{Q}} - \frac{\beta}{\text{vol}}\mathbf{I}\right)\mathbf{v}. \tag{6.13.16}$$

That is, $\mathbf{v}$ is the generalized eigenvector of the matrix pencil $\big(\bar{\mathbf{L}}, \bar{\mathbf{Q}} - \frac{\lambda\beta}{\text{vol}}\mathbf{I}\big)$ with the largest generalized eigenvalue $\lambda_{\max}$.

If there is one largest generalized eigenvalue $\lambda_{\max}$, then the associated generalized eigenvector $\mathbf{u}$ is the unique solution of the generalized eigenproblem. However, if there are multiple (say, p) largest generalized eigenvalues, then p associated generalized eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_p$ are solutions of the generalized eigenproblem. In order to find the unique solution, let $\mathbf{V}_1 = [\mathbf{v}_1, \ldots, \mathbf{v}_p] \in \mathbb{R}^{N\times p}$. If using the Household transform matrix $\mathbf{H} \in \mathbb{R}^{p\times p}$ such that

$$\mathbf{V}_1\mathbf{H} = \left[\begin{array}{c:ccc} \gamma & 0 & \cdots & 0 \\ \hdashline \mathbf{z} & & \times & \end{array}\right], \tag{6.13.17}$$

where $\gamma \neq 0$ and $\times$ may be any values that are not interesting, then the unique solution of the generalized eigenproblem is given by $\mathbf{v} = \begin{bmatrix} \gamma \\ \mathbf{z} \end{bmatrix}$. Then, make $\mathbf{v} \leftarrow \frac{\mathbf{v}}{\|\mathbf{v}\|}\sqrt{\mathrm{vol}}$.

Algorithm 6.22 shows the constrained spectral clustering method.

Algorithm 6.22 Constrained spectral clustering for two-way partition [259]

```
1.  input: Affinity matrix $\mathbf{A}$, constraint matrix $\mathbf{Q}$, constant $\beta$.
2.  $\mathrm{vol} \leftarrow \sum_{i=1}^N \sum_{j=1}^N A_{ij}$,  $\mathbf{D} \leftarrow \mathrm{Diag}\big(\sum_{j=1}^N A_{1j}, \ldots, \sum_{j=1}^N A_{Nj}\big)$.
3.  $\bar{\mathbf{L}} \leftarrow \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$,  $\bar{\mathbf{Q}} \leftarrow \mathbf{D}^{-1/2}\mathbf{Q}\mathbf{D}^{-1/2}$.
4.  $\lambda_{\max} \leftarrow$ the largest eigenvalue of $\bar{\mathbf{Q}}$.
5.  if $\beta \geq \lambda_{\max} \cdot \mathrm{vol}$, then
6.    return $\mathbf{u} = \mathbf{0}$;
7.    else
8.    Solve the generalized eigenproblem $\bar{\mathbf{L}}\mathbf{v} = \lambda\left(\bar{\mathbf{Q}} - \frac{\beta}{\mathrm{vol}}\mathbf{I}\right)\mathbf{v}$ for $(\lambda_i, \mathbf{v}_i)$
9.    if $\lambda_{\max} = \lambda_1$ is unique, then
10.     return $\mathbf{v} \leftarrow \frac{\mathbf{v}}{\|\mathbf{v}\|}\sqrt{\mathrm{vol}}$.
11.     else
12.     $\mathbf{V}_1 \leftarrow [\mathbf{v}_1, \ldots, \mathbf{v}_p]$ for $\lambda_{\max} = \lambda_1 \approx \cdots \approx \lambda_p$.
13.     Perform Household transform $\mathbf{V}_1\mathbf{H}$ in (6.13.17) to get $\mathbf{v} = \begin{bmatrix} \gamma \\ \mathbf{z} \end{bmatrix}$.
14.     return $\mathbf{v} \leftarrow \frac{\mathbf{v}}{\|\mathbf{v}\|}\sqrt{\mathrm{vol}}$.
15.     $\mathbf{v}^* \leftarrow \arg\min_{\mathbf{v}} \mathbf{v}^T\bar{\mathbf{L}}\mathbf{v}$, where $\mathbf{v}$ is among the feasible eigenvectors generated in the
        previous step;
16.   end if
17. end if
18. output: the optimal (relaxed) cluster indicator $\mathbf{u}^* \leftarrow \mathbf{D}^{-1/2}\mathbf{v}^*$.
```

For K-way partition, given all the feasible eigenvectors, the top $K - 1$ eigenvectors are picked in terms of minimizing $\mathbf{v}^T\bar{\mathbf{L}}\mathbf{v}$. Instead of only using the optimal feasible eigenvector $\mathbf{u}^*$ in Algorithm 6.22, one needs to preserve top $(K - 1)$ eigenvectors associated with positive eigenvalues, and perform K-means algorithm based on that embedding. Letting the $K - 1$ eigenvectors form the columns of $\mathbf{V} \in \mathbb{R}^{N\times(K-1)}$, then K-means clustering is performed on the rows of $\mathbf{V}$ to get the final clustering. See Algorithm 6.23.

6.13.3 *Fast Spectral Clustering*

For large data sets, a serious computational challenge of spectral clustering is computation of an affinity matrix between pairs of data points for large data sets.

By combining the spectral clustering algorithm with the Nyström approximation to the graph Laplacian, a fast spectral clustering was developed by Choromanska et al. in 2013 [52]. The Nyström r-rank approximation for any symmetric positive semi-definite matrix $\mathbf{L} \in \mathbb{R}^{N\times N}$ with large N is below.

Algorithm 6.23 Constrained spectral clustering for K-way partition [259]

```
1. input: Affinity matrix $\mathbf{A}$, constraint matrix $\mathbf{Q}$, constant $\beta$, number of classes $K$.
2. $\mathrm{vol} \leftarrow \sum_{i=1}^N \sum_{j=1}^N A_{ij}$, $\mathbf{D} \leftarrow \mathrm{Diag}\big(\sum_{j=1}^N A_{1j}, \ldots, \sum_{j=1}^N A_{Nj}\big)$.
3. $\bar{\mathbf{L}} \leftarrow \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$, $\bar{\mathbf{Q}} \leftarrow \mathbf{D}^{-1/2}\mathbf{Q}\mathbf{D}^{-1/2}$.
4. $\lambda_{\max} \leftarrow$ the largest eigenvalue of $\bar{\mathbf{Q}}$.
5. if $\beta \geq \lambda_{K-1}\mathrm{vol}$, then
6.     return $\mathbf{u} = \mathbf{0}$
7. else
8.     Solve $\bar{\mathbf{L}}\mathbf{v}\left(\bar{\mathbf{L}} - \frac{\beta}{\mathrm{vol}}\mathbf{I}\right)\mathbf{v}$ for $\mathbf{v}_1, \ldots, \mathbf{v}_{K-1}$.
9.     $\mathbf{v} \leftarrow \frac{\mathbf{v}}{\|\mathbf{v}\|}\sqrt{\mathrm{vol}}$.
10.    $\mathbf{V}^* \leftarrow \arg\min_{\mathbf{V}\in\mathbb{R}^{N\times(K-1)}} \mathrm{tr}(\mathbf{V}^T\bar{\mathbf{L}}\mathbf{V})$, where the columns of $\mathbf{V}$ are a subset of the feasible
       eigenvectors generated in the previous step;.
11.    return $\mathbf{u}^* \leftarrow \mathrm{kmeans}(\mathbf{D}^{-1/2}\mathbf{V}^*, K)$.
12. end if
```

1. Perform uniform sampling of $\mathbf{L}$ (without replacement schemes) to create matrix $\mathbf{C} \in \mathbb{R}^{N\times l}$ from the sampled columns, and let $\mathcal{L}$ be indices of l columns sampled.
2. Form matrix $\mathbf{W} \in \mathbb{R}^{l\times l}$ from $\mathbf{C}$ by sampling its l rows via the indices in $\mathcal{L}$.
3. Compute the eigenvalue decomposition (EVD) $\mathbf{W} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^T$, where $\mathbf{U}$ is orthonormal and $\boldsymbol{\Sigma} = \mathbf{Diag}(\sigma_1, \ldots, \sigma_l)$ is an $l \times l$ real diagonal matrix with the diagonal sorted in decreasing order.
4. Calculate the best rank-r approximation to $\mathbf{W}$, denoted as $\mathbf{W}_r = \sum_{p=1}^r \sigma_p \mathbf{u}_p \mathbf{u}^p$, and its Moore–Penrose inverse $\mathbf{W}_r^\dagger = \sum_{p=1}^r \sigma_p^{-1}\mathbf{u}_p\mathbf{u}^p$, where $\mathbf{u}_p$ and $\mathbf{u}^p$ are the pth column and row of $\mathbf{U}$, respectively.
5. The Nyström r-rank approximation $\mathbf{L}_r$ of $\mathbf{L}$ can be obtained as $\mathbf{L}_r = \mathbf{C}\mathbf{W}_r^\dagger\mathbf{C}$.
6. Perform the EVD $\mathbf{W}_r = \mathbf{U}_{W_r}\boldsymbol{\Sigma}_{W_r}\mathbf{U}_{W_r}$.
7. The EVD of the Nyström r-rank approximation $\mathbf{L}_r$ is given by $\mathbf{L}_r = \mathbf{U}_{L_r}\boldsymbol{\Sigma}_{L_r}\mathbf{U}_{L_r}^T$, where $\boldsymbol{\Sigma}_{L_r} = \frac{N}{l}\boldsymbol{\Sigma}_{W_r}$ and $\mathbf{U}_{L_r} = N^{-1/2}\mathbf{C}\mathbf{U}_{W_r}\boldsymbol{\Sigma}_{W_r}^{-1}$.

Algorithm 6.24 shows the Nyström method for matrix approximation.

Algorithm 6.24 Nyström method for matrix approximation [52]

```
1. input: $N \times N$ symmetric matrix $\mathbf{L}$, number $l$ of columns sampled, rank $r$ of matrix approxi-
   mation ($r \leq l \ll N$).
2. $\mathcal{L} \leftarrow$ indices of $l$ columns sampled.
3. $\mathbf{C} \leftarrow \mathbf{L}(:, \mathcal{L})$.
4. $\mathbf{W} \leftarrow \mathbf{C}(\mathcal{L}, :)$.
5. $\mathbf{W} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^T$ and $\mathbf{W}_r = \sum_{p=1}^r \sigma_p\mathbf{u}_p\mathbf{u}^p$, where $\mathbf{u}_p$ and $\mathbf{u}^p$ are the $p$th column and row of $\mathbf{U}$.
6. $\mathbf{W}_r = \mathbf{U}_{W_r}\boldsymbol{\Sigma}_{W_r}\mathbf{U}_{W_r}$.
7. $\boldsymbol{\Sigma}_{L_r} = \frac{N}{l}\boldsymbol{\Sigma}_{W_r}$ and $\mathbf{U}_{L_r} = \sqrt{\frac{l}{N}}\mathbf{C}\mathbf{U}_{W_r}\boldsymbol{\Sigma}_{W_r}^{-1}$.
8. output: Nyström approximation of $\mathbf{L}$ is given by $\mathbf{L}_r = \mathbf{U}_{L_r}\boldsymbol{\Sigma}_{L_r}\mathbf{U}_{L_r}^T$.
```

Once the Nyström approximation of $\mathbf{L}$ is obtained, one can get the fast spectral clustering algorithm of Choromanska et al. [52]. See Algorithm 6.25.

Algorithm 6.25 Fast spectral clustering algorithm [52]

```
1. input: $N \times d$ dataset $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \in \mathbb{R}^d$, number $k$ of clusters, number $l$ of columns
   sampled, rank $r$ of approximation ($k \le r \le l \ll N$).
2. initialization: $\mathbf{A} \in \mathbb{R}^{N \times N}$ with $a_{ij} = [\mathbf{A}]_{ij} = \delta_{ij} K(\mathbf{x}_i, \mathbf{x}_j)$ if $i \neq j$ and 0 otherwise.
3. $\mathcal{L} \leftarrow$ indices of $l$ columns sampled (uniformly without replacement).
4. $\hat{\mathbf{A}} = [\hat{a}_{ij}]_{i,j=1}^{N,l} \leftarrow \mathbf{A}(:, \mathcal{L})$.
5. $\mathbf{D} \in \mathbb{R}^{N \times N}$ with $d_{ij} = \delta_{ij} 1/\sqrt{\sum_{j=1}^{l} \hat{a}_{ij}}$.
6. $\mathbf{\Delta} \in \mathbb{R}^{l \times l}$ with $\Delta_{ij} = \delta_{ij} 1/\sqrt{\sum_{i=1}^{N} \hat{a}_{ij}}$.
7. $\mathbf{C} \leftarrow \hat{\mathbf{I}} - \sqrt{\frac{l}{N}} \mathbf{D}\hat{\mathbf{A}}\mathbf{\Delta}$, where $\hat{\mathbf{I}}$ is an $N \times l$ matrix consisting of columns of the identity matrix
   $\mathbf{I}_{N \times N}$ indexed by $\mathcal{L}$.
8. $\mathbf{W} \leftarrow \mathbf{C}(\mathcal{L}, :)$.
9. $\mathbf{W} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^T$ and $\mathbf{W}_r = \sum_{p=1}^{r} \sigma_p \mathbf{u}_p \mathbf{u}^p$, where $\mathbf{u}_p$ and $\mathbf{u}^p$ are the $p$th column and row of $\mathbf{U}$.
10. $\mathbf{W}_r = \mathbf{U}_{W_r} \mathbf{\Sigma}_{W_r} \mathbf{U}_{W_r}$.
11. $\mathbf{\Sigma}_{L_r} = \frac{N}{l} \mathbf{\Sigma}_{W_r}$ and $\mathbf{U}_{L_r} = \sqrt{\frac{l}{N}} \mathbf{C}\mathbf{U}_{W_r} \mathbf{\Sigma}_{W_r}^{-1}$.
12. $\mathbf{Z} = [z_{im}] \in \mathbb{R}^{N \times k} \leftarrow k$ eigenvectors of $\mathbf{U}_{L_r}$ with $k$ smallest eigenvalues.
13. $[\mathbf{Y}]_{im} = y_{im} = z_{im}/(\sum_m z_{im}^2)^{1/2}$ for $i = 1, \ldots, N;\ j = 1, \ldots, k$.
14. Use any $k$-clustering algorithm (e.g., K-means) to $N$ rows of $\mathbf{Y} \in \mathbb{R}^{N \times k}$ to get $k$ clusters
    $S_i = \{\mathbf{y}_{i_1}^r, \ldots, y_{i_{m_i}}^r\}$, $i = 1, \ldots, k$ with $m_1 + \cdots + m_k = N$, where $y_p^r$ is the $p$th row of $\mathbf{Y}$.
15. output: $k$ spectral clustering results of $S$ are given by $S_i = \{\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_{m_i}}\}$, $i = 1, \ldots, k$.
```

6.14 Semi-Supervised Learning Algorithms

The significance of *semi-supervised learning* (SSL) is mainly reflected in the following three aspects [20].

- From an engineering standpoint, due to the difficulty of collecting labeled data, an approach to pattern recognition that is able to make better use of unlabeled data to improve recognition performance is of potentially great practical significance.
- Arguably, most natural (human or animal) learning occurs in the semi-supervised regime. In many cases, a small amount of feedback is sufficient to allow the child to master the acoustic-to-phonetic mapping of any language.
- In most pattern recognition tasks, humans have access only to a small number of labeled examples. The ability of humans to learn unsupervised concepts (e.g., learning clusters and categories of objects) suggests that the success of human learning in this small sample regime is plausibly due to effective utilization of the large amounts of unlabeled data to extract information that is useful for generalization.

The semi-supervised learning problem has recently drawn a large amount of attention in the machine learning community, mainly due to considerable improvement in learning accuracy when unlabeled data is used in conjunction with a small amount of labeled data.

6.14.1 Semi-Supervised Inductive/Transductive Learning

Typically, semi-supervised learning uses a small amount of labeled data with a large amount of unlabeled data. Hence, as the name suggests, semi-supervised learning falls between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data).

In supervised classification, one is always interested in classifying future test data by using fully labeled training data. In semi-supervised classification, however, the training samples contain both labeled and unlabeled data.

Given a training set $D_{\text{train}} = (X_{\text{train}}, Y_{\text{train}})$ that consists of the labeled set $(X_{\text{labeled}}, Y_{\text{labeled}}) = \{(\mathbf{x}_1, y_1), \ldots (\mathbf{x}_l, y_l)\}$ and the disjoint unlabeled set $X_{\text{unlabeled}} = \{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}\}$. Depending on different training sets and goals, there are two slightly different types of semi-supervised learning: semi-supervised inductive learning and semi-supervised transductive learning.

Definition 6.24 (Inductive Learning) If training set is $\mathcal{D}_{\text{train}} = (X_{\text{train}}, Y_{\text{train}}) = (X_{1\text{abeled}}, Y_{\text{labeled}}) = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$, and test set is $\mathcal{D}_{\text{test}} = X_{\text{test}}$ (unlabeled) that is not appeared in training set, then this setting is called *inductive learning*.

Definition 6.25 (Semi-Supervised Inductive Learning [110, 234]) Let training set consist of a large amount of unlabeled data $X_{\text{unlabeled}} = \{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}\}$ and a small amount of the auxiliary labeled samples $X_{\text{auxiliary}} = (X_{\text{labeled}}, Y_{\text{labeled}}) = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ with $l \ll u$. *Semi-supervised inductive learning* builds a function f so that f is expected to be a good predictor on future data set X_{test}, beyond $\{\mathbf{x}_j\}_{j=l+1}^{l+u}$.

If we do not care about X_{test}, but only want to know the effect of machine learning on $X_{\text{unlabeled}}$, this setting is known as semi-supervised transductive learning, because the unlabeled set $X_{\text{unlabeled}}$ was seen in training.

Definition 6.26 (Semi-Supervised Transductive Learning) Let training set consist of a large amount of unlabeled data $X_{\text{unlabeled}} = \{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}\}$ and a small amount of the auxiliary labeled samples $(X_{\text{labeled}}, Y_{\text{labeled}}) = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ with $l \ll u$. *Semi-supervised transductive learning* uses all labeled data $(X_{\text{labeled}}, Y_{\text{labeled}})$ and unlabeled data $X_{\text{unlabeled}}$ to give the prediction or classification labels of $X_{\text{unlabeled}}$ or its some interesting subset without building the function f.

To put it simply, there are the following differences between inductive learning, semi-supervised learning, semi-supervised inductive learning, and semi-supervised transductive learning.

- Differences in training set: Inductive learning includes only all labeled data $(X_{\text{labled}}, Y_{\text{labled}})$, while semi-supervised learning, semi-super inductive learning, and semi-supervised transductive learning use the same training set including both labeled data $(X_{\text{labeled}}, Y_{\text{labeled}}) = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ and unlabeled data $X_{\text{unlabeled}} = \{\mathbf{x}_{l+1}, \ldots, x_{l+u}\}$.
- Differences in testing data: Unlabeled data X_{unlabled} is not the testing data in inductive learning, semi-supervised learning, and semi-supervised inductive learning, i.e., the samples we want to predict are not seen (or used) in training, whereas unlabeled data X_{unlabled} is the testing data in transductive learning, i.e., the samples we want to predict have been seen (or used) in training: $X_{\text{test}} = X_{\text{unlabeled}}$.
- Differences in prediction functions: In inductive learning, semi-supervised learning, and semi-supervised inductive learning, one needs to build the prediction function on unseen (or used) data (i.e., outside the training data), $f : D_{\text{train}} \to D_{\text{test}}$. However, this function does not need to be built in semi-supervised transductive learning, which only needs to give the predictions for some subset, rather than all, of the unlabeled data seen in training.

The semi-supervised transductive learning is simply called the transductive learning. In semi-supervised learning, semi-supervised inductive learning, and semi-supervised transductive learning, a small amount of labeled data set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ is usually viewed as an *auxiliary set*, denoted as $D_{\text{auxiliary}} = (X_{\text{labeled}}, Y_{\text{labeled}})$. It has been noted that a small amount of auxiliary labeled data $X_{\text{auxiliary}}$ typically helps boost the performance of the classifier or predictor significantly.

In fact, most semi-supervised learning strategies are based on extending either unsupervised or supervised learning to include additional information typical of the other learning paradigm. Specifically, semi-supervised learning encompasses the following different settings [305]:

- *Semi-supervised classification:* It is an extension to the supervised classification, and its goal is to train a classifier f from both the labeled and unlabeled data, such that it is better than the supervised classifier trained on a small amount of labeled data alone.
- *Constrained clustering:* This is an extension to unsupervised clustering. Some "supervised information" can be so-called must-link constraints: two instances $\mathbf{x}_i, \mathbf{x}_j$ must be in the same cluster; and cannot-link constraints: $\mathbf{x}_i, \mathbf{x}_j$ cannot be in the same cluster. One can also constrain the size of the clusters. The goal of constrained clustering is to obtain better clustering than the clustering from unlabeled data alone.

The acquisition of labeled data for a learning problem often is expensive since it requires a skilled human agent or a physical experiment. The cost associated with the labeling process thus may render a fully labeled training set infeasible, while acquisition of unlabeled data is relatively inexpensive. In such situations, semi-supervised learning can be particularly useful.

For example, in the process industry, it is a difficult task to assign the type of detected faulty data samples. This may require process knowledge and experiences of process engineers, and will be costly and may need a lot of time. As a result, there are only a small number of faulty samples that are assigned to their corresponding types, while most faulty samples are unlabeled. Since these unlabeled samples still contain important information of the process, they can be used to improve greatly the efficiency of the fault classification system. In this case, as compared to the model that only depends on a small part of labeled data samples, semi-supervised learning, with both labeled samples and unlabeled samples, will provide an improved fault classification model [104].

To achieve higher accuracy of machine learning with fewer training labels and a large pool of unlabeled data, several approaches have been proposed for semi-supervised learning. In the following, we summarize the common semi-supervised learning algorithms: self-training, cluster-then-label, and co-training.

6.14.2 Self-Training

Self-training is a commonly used technique for semi-supervised learning, and is the learning process that uses its own predictions to teach itself. For this reason, it is also known as *self-teaching* or *bootstrapping*.

In self-training a classifier is first found with the small amount of labeled data. Then, the classifier is used to classify the unlabeled data. Typically the most confident unlabeled points, together with their predicted labels, are added to the training set. The classifier is retrained and the procedure repeated until all of unlabeled data are classified. See Algorithm 6.26 for self-training.

Algorithm 6.26 Self-training algorithm [305]

```
1. input: labeled data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$, unlabeled data $\{\mathbf{x}_j\}_{j=l+1}^{l+u}$.
2. initialization: $L = \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$ and $U = \{\mathbf{x}_j\}_{j=l+1}^{l+u}$. Let $k = l + 1$.
3. repeat
4.   Train $f$ from $L$ using supervised learning.
5.   Apply $f$ to the unlabeled instance $\mathbf{x}_k$ in $U$.
6.   Remove a subset $S$ of learning success from $U$; and add $\{(\mathbf{x}_k, f(\mathbf{x}_k))|\mathbf{x}_k \in S\}$ to $L$.
7.   exist if $k = l + u$.
8.   $k \leftarrow k + 1$.
9. return
10. output: $f$.
```

On self-training, one has the following comments and considerations [305].

Remark 1 For self-training, the algorithm's own predictions, assuming high confidence, tend to be correct. This is likely to be the case when the classes form well-separated clusters.

Remark 2 Implementation for self-training is simple. Importantly, it is also a wrapper method, which means that the choice of learner for f in Step 4 is left completely open: the learner can be a simple kNN algorithm, or a very complicated classifier. The self-training procedure "wraps" around the learner without changing its inner workings. This is important for many real applications like natural language processing, where the learners can be complicated black boxes not amenable to changes.

Self-training is available for a larger labeled data set L. If L is small, an early mistake made by f can reinforce itself by generating incorrectly labeled data. Retraining with this data will lead to an even worse f in the next iteration.

The heuristics for *semi-supervised clustering* can alleviate the above problem, see Algorithm 6.27.

Algorithm 6.27 Propagating 1-nearest neighbor clustering algorithm [305]

```
1. input: labeled data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$, unlabeled data $\{\mathbf{x}_j\}_{j=l+1}^{l+u}$, distance function $d(\cdot,\cdot)$.
2. initialization: $L = \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$ and $U = \{\mathbf{x}_j\}_{j=l+1}^{l+u} = \{\mathbf{z}_j\}_{j=l+1}^{l+u}$.
3. repeat
4.   Select $\mathbf{z} = \arg\min_{\mathbf{z}\in U, \mathbf{x}\in L} d(\mathbf{z}, \mathbf{x})$.
5.   Let $k$ be the label of $\mathbf{x} \in L$ nearest to $\mathbf{z}$, and add the new labeled data $(\mathbf{z}, y_k)$ to $L$.
6.   Remove $\mathbf{z}$ from $U$.
7.   exit if $U$ is empty.
8. return
9. output: semi-supervised learning dataset $L = \{(\mathbf{x}_i, y_i)\}_{i=1}^{l+u}$.
```

6.14.3 Co-training

The traditional machine learning algorithms including self-training and the propagating 1-nearest neighbor clustering are based on the assumption that the instance has one attribute or one kind of information. However, in many real-world applications there are *two-views data* and *co-occurring data*. These data can be described using two different attributes or two different "kinds" of information. For example, a web-page contains two different kinds of information [30]:

- One kind of information about a web-page is the text appearing on the document itself.
- Another kind of information is the anchor text attached to hyperlinks pointing to this page, from other pages on the web.

Therefore, analyzing two views and co-occurring data is an important challenge in machine learning, pattern recognition, and signal processing communities, e.g., automatic annotation of text, audio, music, image, and video (see, e.g., [30, 80, 144, 193]) and sensor data mining [62].

More generally, the *co-training setting* [30] applies when a data set has a natural division of its features. Co-training setting assumes usually:

- features can be split into two sets;
- each sub-feature set is sufficient to train a good classifier;
- the two sets are conditionally independent given the class.

Traditional machine learning algorithms that learn over these domains ignore this division and pool all features together.

Definition 6.27 (Co-training Algorithm [30, 193]) *Co-training algorithm* is a multi-view weakly supervised algorithm using the co-training setting, and may learn separate classifiers over each of the feature sets, and combine their predictions to decrease classification error.

Given a small set of labeled two-view data $\mathcal{L} = \{(\mathbf{x}_i, \mathbf{y}_i), z_i\}_{i=1}^{l}$ and a large set of unlabeled two-view data $\mathcal{U} = \{(\mathbf{x}_j, \mathbf{y}_j)\}_{j=l+1}^{l+u}$, where $(\mathbf{x}_i, \mathbf{y}_i)$ is called the data bag and z_i is the class label associated labeled two-view data bag $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1, \ldots, l$.

Two classifiers $\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$ work in co-training fashion: $\mathbf{h}^{(1)}$ is a view-1 classifier that is only based the first view $\mathbf{x}^{(1)}$ (say $\mathbf{x}_i$) and ignores the second view $\mathbf{x}^{(2)}$ (say $\mathbf{y}_i$), while $\mathbf{h}^{(2)}$ is a view-2 classifier based on the second view $\mathbf{x}^{(2)}$ and ignores the first view $\mathbf{x}^{(1)}$.

The working process of co-training is [189, 304]: two separate classifiers are initially trained with the labeled data, on the two sub-feature sets, respectively. Then, each classifier uses one view of the data and selects most confident predictions from the pool and adds the corresponding instances with their predicted labels to the labeled data while maintaining the class distribution in the labeled data. One classifier "teaches" the other classifier with the few unlabeled examples and provides their feeling most confident unlabeled-data predictions as the training data for the other. Each classifier is then retrained with the additional training examples given by the other classifier, and the process repeats. In co-training process, the unlabeled data is eventually exhausted.

Co-training is a wrapper method. That is, it does not matter what the learning algorithms are for the two classifiers $\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$. The only requirement is that the two classifiers can assign a confidence score to their predictions. The confidence score is used to select which unlabeled instances to turn into additional training data for the other view. Being a wrapper method, co-training is widely applicable to many tasks.

As semi-supervised learning for two-view data and co-occurring data, the most popular algorithm is the standard co-training developed by Blum and Mitchell [30], as shown in Algorithm 6.28.

The following are two variants of co-training [304].

Algorithm 6.28 Co-training algorithm [30]

```
1. input: labeled data set $L = \{\mathbf{x}_i, y_i\}_{i=1}^{l}$, unlabeled data set $U = \{\mathbf{x}_j\}_{j=l+1}^{l+u}$. Each instance has
   two views $\mathbf{x}_i = [\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}]$.
2. initialization: create a pool $U'$ of examples by choosing $u$ examples at random from $U$.
3. repeat until unlabeled data is used up:
4.    Train a view-1 classifier $\mathbf{h}^{(1)}$ from $\{\mathbf{x}_i^{(1)}, y_i\}$ in $L$.
5.    Train a view-2 classifier $\mathbf{h}^{(2)}$ from $\{\mathbf{x}_i^{(2)}, y_i\}$ in $L$.
6.    Allow $\mathbf{h}^{(1)}$ to label $p$ positive and $n$ negative examples from $U'$.
7.    Allow $\mathbf{h}^{(2)}$ to label $p$ positive and $n$ negative examples from $U'$.
8.    Add these self-labeled examples to $L$, and remove these from the unlabeled data $U$.
9.    Randomly choose $2p + 2n$ examples from $U$ to replenish $U'$.
10. return
11. output: $\{\mathbf{x}_i, y_i\}_{i=1}^{l+u}$.
```

1. *Democratic co-learning* [299]. Let $\mathcal{L}$ be the set of labeled data and $\mathcal{U}$ be the set of unlabeled data, and $A_1, \ldots, A_n$ (for $n \geq 3$) the provided supervised learning algorithms. Democratic co-learning begins by training all learners on the original labeled data set $\mathcal{L}$. For every example $\mathbf{x}_u \in \mathcal{U}$ in the unlabeled data set $\mathcal{U}$, each learner predicts a label. If a majority of learners confidently agree on the class of an unlabeled point $\mathbf{x}_u$, that classification is used as the label of $\mathbf{x}_u$. Then $\mathbf{x}_u$ and its label are added to the training data. All learners are retrained on the updated training set. The final prediction is made with a variant of a weighted majority vote among the n learners.
2. *Tri-training* [300]. If two of tri-training learners agree on the classification of an unlabeled point, the classification is used to teach the third classifier. This approach thus avoids the need of explicitly measuring label confidence of any learner. It can be applied to data sets without different views, or different types of classifiers.

6.15 Canonical Correlation Analysis

As the previous section illustrated, for a process described by two sets of variables corresponding to two different aspects or views, analyzing the relations between these two views helps improve the understanding of the underlying system. An alternative to two-view data analysis is canonical correlation analysis.

Canonical correlation analysis (CCA) [7, 122] is a standard two-view multivariate statistical method of correlating linear relationships between two random vectors such that they are maximally correlated, and hence is a powerful tool for analyzing multi-dimensional paired sample data. Paired samples are chosen in a random manner, and thus are random samples.

6.15.1 Canonical Correlation Analysis Algorithm

In the case of CCA, the variables of an observation can be partitioned into two sets that can be seen as the two views of the data. Let the two-view matrices $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N] \in \mathbb{R}^{N\times p}$ and $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N] \in \mathbb{R}^{N\times q}$ have covariance matrices $(\mathbf{C}_{xx}, \mathbf{C}_{yy})$ and cross-covariance matrices $(\mathbf{C}_{xy}, \mathbf{C}_{yx})$, respectively. The row vectors $\mathbf{x}_n \in \mathbb{R}^{1\times p}$ and $\mathbf{y}_n \in \mathbb{R}^{1\times q}$ $(n = 1, \ldots, N)$ denote the sets of empirical multivariate observations in $\mathbf{X}$ and $\mathbf{Y}$, respectively.

The aim of CCA is to extract the linear relations between the variables of $\mathbf{X}$ and $\mathbf{Y}$. For this end, consider the following transformations:

$$\mathbf{X}\mathbf{w}_x = \mathbf{z}_x \quad \text{and} \quad \mathbf{Y}\mathbf{w}_y = \mathbf{z}_y, \tag{6.15.1}$$

where $\mathbf{w}_x \in \mathbb{R}^p$ and $\mathbf{w}_y \in \mathbb{R}^q$ are, respectively, the projection vectors associated with the data matrices $\mathbf{X}$ and $\mathbf{Y}$, while $\mathbf{z}_x \in \mathbb{R}^N$ and $\mathbf{z}_y \in \mathbb{R}^N$ are, respectively, the images of the positions $\mathbf{w}_x$ and $\mathbf{w}_y$. The positions $\mathbf{w}_x$ and $\mathbf{w}_y$ are often referred to as *canonical weight vectors*, and the images $\mathbf{z}_x$ and $\mathbf{z}_y$ are termed as *canonical variants* or *score variants* [253, 265]. Hence, the data matrices $\mathbf{X}$ and $\mathbf{Y}$ represent linear transformations of the positions $\mathbf{w}_x$ and $\mathbf{w}_y$ onto the images $\mathbf{z}_x$ and $\mathbf{z}_y$ in the space $\mathbb{R}^N$, respectively.

Define the cosine of the angle between the images $\mathbf{z}_x$ and $\mathbf{z}_y$:

$$\cos(\mathbf{z}_x, \mathbf{z}_y) = \frac{\langle \mathbf{z}_x, \mathbf{z}_y\rangle}{\|\mathbf{z}_x\|\,\|\mathbf{z}_y\|}, \tag{6.15.2}$$

which is referred to as the *canonical correlation*.

The constraints of CCA on the mappings are given below:

- the position vectors of the images $\mathbf{z}_x$ and $\mathbf{z}_y$ are unit norm vectors;
- the enclosing angle $\theta \in [0, \frac{\pi}{2}]$ [108], between $\mathbf{z}_x$ and $\mathbf{z}_y$, is minimized.

The principle behind CCA is to find two positions in the two data spaces, respectively, that have images on a unit ball such that the angle between them is minimized and consequently the canonical correlation is maximized:

$$\cos\theta = \max_{\mathbf{z}_x, \mathbf{z}_y \in \mathbb{R}^N} \langle \mathbf{z}_x, \mathbf{z}_y\rangle \tag{6.15.3}$$

subject to $\|\mathbf{z}_x\|_2 = 1$ and $\|\mathbf{z}_y\|_2 = 1$.

For solving the above optimization problem, define the sample cross-covariance matrix

$$\mathbf{C}_{xy} = \mathbf{X}^T\mathbf{Y}, \quad \mathbf{C}_{yx} = \mathbf{Y}^T\mathbf{X} = \mathbf{C}_{xy}^T \tag{6.15.4}$$

and the empirical variance matrices

$$\mathbf{C}_{xx} = \mathbf{X}^T\mathbf{X}, \quad \mathbf{C}_{yy} = \mathbf{Y}^T\mathbf{Y}. \tag{6.15.5}$$

Hence, the constraints $\|\mathbf{z}_x\|_2 = 1$ and $\|\mathbf{z}_y\|_2 = 1$ can be rewritten as follows:

$$\|\mathbf{z}_x\|_2^2 = \mathbf{z}_x^T \mathbf{z}_x = \mathbf{w}_x^T \mathbf{C}_{xx} \mathbf{w}_x = 1, \tag{6.15.6}$$

$$\|\mathbf{z}_y\|_2^2 = \mathbf{z}_y^T \mathbf{z}_y = \mathbf{w}_y^T \mathbf{C}_{yy} \mathbf{w}_y = 1. \tag{6.15.7}$$

In this case, the covariance matrix between $\mathbf{z}_x$ and $\mathbf{z}_y$ can be also written as

$$\mathbf{z}_x^T \mathbf{z}_y = \mathbf{w}_x^T \mathbf{X}^T \mathbf{Y} \mathbf{w}_y = w_x^T \mathbf{C}_{xy} \mathbf{w}_y. \tag{6.15.8}$$

Then, the optimization problem (6.15.3) becomes

$$\cos\theta = \max_{\mathbf{z}_x, \mathbf{z}_y \in \mathbb{R}^N} \langle \mathbf{z}_x, \mathbf{z}_y \rangle = \max_{w_x \in \mathbb{R}^p, \mathbf{w}_y \in \mathbb{R}^q} \mathbf{w}_x^T \mathbf{C}_{xy} \mathbf{w}_y \tag{6.15.9}$$

subject to $\|\mathbf{z}_x\|_2^2 = \mathbf{w}_x^T \mathbf{C}_{xx} \mathbf{w}_x = 1$ and $\|\mathbf{z}_y\|_2^2 = \mathbf{w}_y^T \mathbf{C}_{yy} \mathbf{w}_y = 1$.

In order to solve this constrained optimization problem, define the Lagrange objective function

$$\mathcal{L}(\mathbf{w}_x, \mathbf{w}_y) = w_x^T \mathbf{C}_{xy} \mathbf{w}_y - \lambda_1 \left(\mathbf{w}_x^T \mathbf{C}_{xx} \mathbf{w}_x - 1 \right) - \lambda_2 \left(\mathbf{w}_y^T \mathbf{C}_{yy} \mathbf{w}_y - 1 \right), \tag{6.15.10}$$

where λ_1 and λ_2 are the Lagrange multipliers. We then have

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_x} = \mathbf{C}_{xy} w_y - \lambda_1 \mathbf{C}_{xx} \mathbf{w}_x = \mathbf{0} \Rightarrow \mathbf{w}_x^T \mathbf{C}_{xy} w_y - \lambda_1 \mathbf{w}_x^T \mathbf{C}_{xx} \mathbf{w}_x = 0, \tag{6.15.11}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_y} = \mathbf{C}_{yx} w_x - \lambda_2 \mathbf{C}_{yy} \mathbf{w}_y = \mathbf{0} \Rightarrow \mathbf{w}_y^T \mathbf{C}_{yx} w_x - \lambda_2 \mathbf{w}_y^T \mathbf{C}_{yy} \mathbf{w}_y = 0. \tag{6.15.12}$$

Since $\mathbf{w}_x^T \mathbf{C}_{xx} \mathbf{w}_x = 1, \mathbf{w}_y^T \mathbf{C}_{yy} \mathbf{w}_y = 1$ and $\cos(\mathbf{z}_x, \mathbf{z}_y) = \mathbf{w}_x^T \mathbf{C}_{xy} w_y = \cos(\mathbf{z}_y, \mathbf{z}_x) = \mathbf{w}_y^T \mathbf{C}_{yx} w_x$, we get

$$\lambda_1 = \lambda_2 = \lambda. \tag{6.15.13}$$

Substitute (6.15.13) into (6.15.11) and (6.15.12), respectively, we obtain

$$\mathbf{C}_{xy} \mathbf{w}_y = \lambda \mathbf{C}_{xx} \mathbf{w}_x, \tag{6.15.14}$$

$$\mathbf{C}_{yx} \mathbf{w}_x = \lambda \mathbf{C}_{yy} \mathbf{w}_y, \tag{6.15.15}$$

which can be combined into the generalized eigenvalue decomposition problem [13, 115]:

$$\begin{bmatrix} \mathbf{O} & \mathbf{C}_{xy} \\ \mathbf{C}_{yx} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \mathbf{w}_x \\ \mathbf{w}_y \end{bmatrix} = \lambda \begin{bmatrix} \mathbf{C}_{xx} & \mathbf{O} \\ \mathbf{O} & \mathbf{C}_{yy} \end{bmatrix} \begin{bmatrix} \mathbf{w}_x \\ \mathbf{w}_y \end{bmatrix}, \tag{6.15.16}$$

where $\mathbf{O}$ denotes the null matrix.

Equation (6.15.16) shows that λ and $\begin{bmatrix} \mathbf{w}_x \\ \mathbf{w}_y \end{bmatrix}$ are, respectively, the generalized eigenvalue and the corresponding generalized eigenvector of the matrix pencil

$$\left(\begin{bmatrix} \mathbf{O} & \mathbf{C}_{xy} \\ \mathbf{C}_{yx} & \mathbf{O} \end{bmatrix}, \begin{bmatrix} \mathbf{C}_{xx} & \mathbf{O} \\ \mathbf{O} & \mathbf{C}_{yy} \end{bmatrix} \right). \tag{6.15.17}$$

The generalized eigenvalue decomposition problem (6.15.16) can also solved by using the SVDs of the data matrices $\mathbf{X}$ and $\mathbf{Y}$.

Let $\mathbf{X} = \mathbf{U}_x \mathbf{D}_x \mathbf{V}_x^T$ and $\mathbf{Y} = \mathbf{U}_y \mathbf{D}_y \mathbf{V}_y^T$ be the SVDs of $\mathbf{X}$ and $\mathbf{Y}$. Then, we have

$$\mathbf{C}_{xx} = \mathbf{X}^T \mathbf{X} = \mathbf{V}_x \mathbf{D}_x^2 \mathbf{V}_x^T, \tag{6.15.18}$$

$$\mathbf{C}_{yy} = \mathbf{Y}^T \mathbf{Y} = \mathbf{V}_y \mathbf{D}_y^2 \mathbf{V}_y^T, \tag{6.15.19}$$

$$\mathbf{C}_{xy} = \mathbf{X}^T \mathbf{Y} = \mathbf{V}_x \mathbf{D}_x \mathbf{U}_x^T \mathbf{U}_y \mathbf{D}_y \mathbf{V}_y^T = C_{yx}^T. \tag{6.15.20}$$

From (6.15.14) it is known that $\mathbf{w}_x = \frac{1}{\lambda} \mathbf{C}_{xx}^{-1} \mathbf{C}_{xy} \mathbf{w}_y$. Substitute this result into (6.15.15) to get

$$\left(\mathbf{C}_{yx} \mathbf{C}_{xx}^{-1} \mathbf{C}_{xy} - \lambda^2 \mathbf{C}_{yy} \right) \mathbf{w}_y = \mathbf{0}. \tag{6.15.21}$$

Substituting $\mathbf{C}_{xx}^{-1} = \mathbf{V}_x \mathbf{D}_x^{-2} \mathbf{V}_x^T$ into the above equation, we obtain

$$\left(\mathbf{V}_y \mathbf{D}_y \mathbf{U}_y^T \mathbf{U}_x \mathbf{V}_x^T \mathbf{V}_x \mathbf{D}_x \mathbf{U}_x^T \mathbf{U}_y \mathbf{D}_y \mathbf{V}_y^T - \lambda^2 \mathbf{V}_y \mathbf{D}_y^2 \mathbf{V}_y^T \right) = \mathbf{0}$$

By using $\mathbf{V}_x \mathbf{V}_x = \mathbf{I}$, the above equation can be rewritten as

$$\left(\mathbf{V}_y \mathbf{D}_y \mathbf{U}_y^T \mathbf{U}_x \mathbf{U}_y \mathbf{D}_y \mathbf{V}_y^T - \lambda^2 \mathbf{V}_y \mathbf{D}_y^2 \mathbf{V}_y^T \right) = \mathbf{0}. \tag{6.15.22}$$

Premultiplying $\mathbf{D}_y^{-1} \mathbf{V}_y^T$, then the above equation reduces to

$$\left(\mathbf{U}_y^T \mathbf{U}_x \mathbf{U}_x^T \mathbf{U}_y \mathbf{V}_y^T - \lambda^2 \right) \mathbf{D}_y \mathbf{V}_y^T \mathbf{w}_y = \mathbf{0}. \tag{6.15.23}$$

Let $\mathbf{U}_x^T\mathbf{U}_y = \mathbf{U}\mathbf{D}\mathbf{V}^T$ be the SVD of $\mathbf{U}_x^T\mathbf{U}_y$, then

$$\left(\mathbf{V}\mathbf{D}^2\mathbf{V}^T - \lambda^2\mathbf{I}\right)\mathbf{D}_y\mathbf{V}_y^T\mathbf{w}_y = \mathbf{0}. \tag{6.15.24}$$

or

$$\mathbf{V}\left(\mathbf{D}^2 - \lambda^2\mathbf{I}\right)\mathbf{V}^T\mathbf{D}_y\mathbf{V}_y^T\mathbf{w}_y = \mathbf{0}, \tag{6.15.25}$$

Premultiplying $\mathbf{V}^T$ and noting that $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, the above equation can be simplified to [265]:

$$\left(\mathbf{D}^2 - \lambda^2\mathbf{I}\right)\mathbf{V}^T\mathbf{D}_y\mathbf{V}_y^T\mathbf{w}_y = \mathbf{0}. \tag{6.15.26}$$

Clearly, when

$$\mathbf{w}_y = \mathbf{V}_y\mathbf{D}_y^{-1}\mathbf{V}, \tag{6.15.27}$$

Equation (6.15.26) yields the result

$$\left(\mathbf{D}^2 - \lambda^2\mathbf{I}\right) = \mathbf{0} \quad \text{or} \quad |\mathbf{D}^2 - \lambda^2\mathbf{I}| = 0. \tag{6.15.28}$$

By mimicking this process, we immediately have that when

$$\mathbf{w}_x = \mathbf{V}_x\mathbf{D}_x^{-1}\mathbf{V}, \tag{6.15.29}$$

then

$$\left(\mathbf{D}^2 - \lambda^2\mathbf{I}\right)\mathbf{V}^T\mathbf{D}_x\mathbf{V}_x^T\mathbf{w}_x = \mathbf{0}. \tag{6.15.30}$$

The above analysis shows that the generalized eigenvalues λ in the generalized eigenvalue problem (6.15.16) is given by the singular value matrix $\mathbf{D}$ of the matrix $\mathbf{U}_x^T\mathbf{U}_y$, as shown in (6.15.28), and the corresponding generalized eigenvectors (i.e., canonical variants) are given by $\mathbf{w}_x = \mathbf{V}_x\mathbf{D}_x^{-1}\mathbf{V}$ in (6.15.29) and $\mathbf{w}_y = \mathbf{V}_y\mathbf{D}_y^{-1}\mathbf{V}$ in (6.15.27), respectively.

Algorithm 6.29 summarizes the CCA algorithm.

6.15.2 Kernel Canonical Correlation Analysis

Let X and Y denote two views (i.e., two attribute sets describing the data), and $(\mathbf{x}, \mathbf{y}, c)$ be a labeled example where $\mathbf{x} \in X$ and $\mathbf{y} \in Y$ are the two portions of the example, and c is their label. For simplicity, assume that $c \in \{0, 1\}$ where 0 and 1 denote negative and positive classes, respectively. Assume that there exist

Algorithm 6.29 Canonical correlation analysis (CCA) algorithm [13, 115]

```
1. input: The two-view data vectors $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ with $\mathbf{x}_n \in \mathbb{R}^{1\times p}, \mathbf{y}_n \in \mathbb{R}^{1\times q}$.
2. initialization: Normalize $\mathbf{x}_n \leftarrow \mathbf{x}_n - \frac{1}{p}\sum_{i=1}^p x_n(i)\mathbf{1}_{p\times 1}^T$ and $\mathbf{y}_n \leftarrow \mathbf{y}_n - \frac{1}{q}\sum_{i=1}^q y_n(i)\mathbf{1}_{q\times 1}^T$.
3. Let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$ and $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N]$.
4. Make the SVD $\mathbf{X} = \mathbf{U}_x\mathbf{D}_x\mathbf{V}_x^T$, $\mathbf{Y} = \mathbf{U}_y\mathbf{D}_y\mathbf{V}_y^T$.
5. Calculate the SVD $\mathbf{U}_x^T\mathbf{U}_y = \mathbf{U}\mathbf{D}\mathbf{V}^T$.
6. Calculate $\mathbf{w}_x = \mathbf{V}_x\mathbf{D}_x^{-1}\mathbf{V}$ and $\mathbf{w}_y = \mathbf{V}_y\mathbf{D}_y^{-1}\mathbf{V}$.
7. output: the canonical variants $(\mathbf{w}_x, \mathbf{w}_y)$ of $(\mathbf{X}, \mathbf{Y})$.
```

two decision functions f_X and f_Y over X and Y, respectively, such that $f_X(\mathbf{x}) = f_Y(\mathbf{y}) = c$. Intuitively, this means that every example is associated with two views each contains sufficient information for determining the label of the example.

Given one labeled example $((\mathbf{x}_0, \mathbf{y}_0), 1)$ and a large number of unlabeled examples $U = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N-1}$, the task of semi-supervised learning is to train a classifier for classifying unlabeled examples $U = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N-1}$, i.e., determining unknown labels $c_i, i = 1, \ldots, N-1$.

For the data described by two sufficient views, some projections in these two views should have strong correlation.

Let $\mathbf{X} = [\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{N-1}]$ and $\mathbf{Y} = [\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_{N-1}]$ be two-view data matrices consisting of one labeled data $((\mathbf{x}_0, \mathbf{y}_0), 1)$ and $N-1$ unlabeled data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N-1}$. CCA finds two projector vectors $\mathbf{w}_x$ and $\mathbf{w}_y$ such that the projections $\mathbf{X}^T\mathbf{w}_x$ and $\mathbf{Y}^T\mathbf{w}_y$ are strongly correlated as soon as possible, namely their correlation coefficient is maximized:

$$(\mathbf{w}_x, \mathbf{w}_y) = \arg\max_{\mathbf{w}_x, \mathbf{w}_y} \frac{\langle \mathbf{X}^T\mathbf{w}_x, \mathbf{Y}^T\mathbf{w}_y\rangle}{\|\mathbf{X}^T\mathbf{w}_x\| \cdot \|\mathbf{Y}^T\mathbf{w}_y\|}$$
$$= \arg\max_{\mathbf{w}_x, \mathbf{w}_y} \left(\frac{\mathbf{w}_x^T\mathbf{C}_{xy}\mathbf{w}_y}{\sqrt{\mathbf{w}_x^T\mathbf{C}_{xx}\mathbf{w}_x \cdot \mathbf{w}_y^T\mathbf{C}_{yy}\mathbf{w}_y}}\right) \tag{6.15.31}$$

$$\text{subject to} \begin{cases} \mathbf{w}_x^T\mathbf{C}_{xx}\mathbf{w}_x = 1, \\ \mathbf{w}_y^T\mathbf{C}_{yy}\mathbf{w}_y = 1, \end{cases} \tag{6.15.32}$$

where $C_{xy} = \frac{1}{N}\mathbf{X}\mathbf{Y}^T$ is the between-sets covariance matrix of $\mathbf{X}$ and $\mathbf{Y}$, while $\mathbf{C}_{xx} = \frac{1}{N}\mathbf{X}\mathbf{X}^T$ and $\mathbf{C}_{yy} = \frac{1}{N}\mathbf{Y}\mathbf{Y}^T$ are the within-sets covariance matrices of $\mathbf{X}$ and $\mathbf{Y}$, respectively.

By the Lagrange multiplier method, the objective of the above primal constrained optimization can be rewritten as the objective of the following dual unconstrained optimization:

$$\mathcal{L}_{\mathrm{D}}(\mathbf{w}_x, \mathbf{w}_y) = \mathbf{w}_x^T \mathbf{C}_{xy} \mathbf{w}_y - \frac{\lambda_x}{2}\left(\mathbf{w}_x^T \mathbf{C}_{xx} \mathbf{w}_x - 1\right) + \frac{\lambda_y}{2}\left(\mathbf{w}_y^T \mathbf{C}_{yy} \mathbf{w}_y - 1\right). \tag{6.15.33}$$

From the first-order optimality conditions we have

$$\frac{\partial \mathcal{L}_{\mathrm{D}}}{\partial \mathbf{w}_x} = \mathbf{0} \quad \Rightarrow \quad \mathbf{C}_{xy}\mathbf{w}_y = \lambda_x \mathbf{C}_{xx}\mathbf{w}_x, \tag{6.15.34}$$

$$\frac{\partial \mathcal{L}_{\mathrm{D}}}{\partial \mathbf{w}_y} = \mathbf{0} \quad \Rightarrow \quad \mathbf{C}_{yx}\mathbf{w}_x = \lambda_y \mathbf{C}_{yy}\mathbf{w}_y, \tag{6.15.35}$$

which can be combined into

$$\begin{aligned} 0 &= \mathbf{w}_x^T \mathbf{C}_{xy}\mathbf{w}_y - \lambda_x \mathbf{w}_x^T \mathbf{C}_{xx}\mathbf{w}_x - \mathbf{w}_y^T \mathbf{C}_{yx}\mathbf{w}_x + \lambda_y \mathbf{w}_y^T \mathbf{C}_{yy}\mathbf{w}_y \\ &= \lambda_y \mathbf{w}_y^T \mathbf{C}_{yy}\mathbf{w}_y - \lambda_x \mathbf{w}_x^T \mathbf{C}_{xx}\mathbf{w}_x, \\ &= \lambda_y - \lambda_x. \end{aligned}$$

Letting $\lambda = \lambda_x = \lambda_y$ and assuming that $\mathbf{C}_{yy}$ is invertible, then (6.15.35) becomes

$$\mathbf{w}_y = \frac{1}{\lambda}\mathbf{C}_{yy}^{-1}\mathbf{C}_{yx}\mathbf{w}_x. \tag{6.15.36}$$

Substituting (6.15.36) into (6.15.34) yields

$$\mathbf{C}_{xy}\mathbf{C}_{yy}^{-1}\mathbf{C}_{yx}\mathbf{w}_x = \lambda^2 \mathbf{C}_{xx}\mathbf{w}_x. \tag{6.15.37}$$

This implies that the projection vector $\mathbf{w}_x$ is the generalized eigenvector corresponding to the largest generalized eigenvalue $\lambda_{\max}^2$ of the matrix pencil $(\mathbf{C}_{xy}\mathbf{C}_{yy}^{-1}\mathbf{C}_{yx}, \mathbf{C}_{xx})$.

In order to identify nonlinearly correlated projections between the two views, we can apply the kernel extensions of canonical correlation analysis, called simply *Kernel canonical correlation analysis* (kernel CCD) [115]. The kernel CCA maps the instances $\mathbf{x}$ and $\mathbf{y}$ to the higher-dimensional kernel instances $\boldsymbol{\phi}_x(\mathbf{x})$ and $\boldsymbol{\phi}_y(\mathbf{y})$, respectively.

Letting

$$\mathbf{S}_x = [\boldsymbol{\phi}_x(\mathbf{x}_0), \boldsymbol{\phi}_x(\mathbf{x}_1), \ldots, \boldsymbol{\phi}_x(\mathbf{x}_{N-1})], \tag{6.15.38}$$

$$\mathbf{S}_y = [\boldsymbol{\phi}_y(\mathbf{y}_0), \boldsymbol{\phi}_y(\mathbf{y}_1), \ldots, \boldsymbol{\phi}_y(\mathbf{y}_{N-1})], \tag{6.15.39}$$

then the projection vectors $\boldsymbol{\phi}_x(\mathbf{x}_i)$ and $\boldsymbol{\phi}_y(\mathbf{y}_i)$ in higher-dimensional kernel space can be rewritten as $\mathbf{w}_x^{\phi} = \mathbf{S}_x\boldsymbol{\alpha}$ and $\mathbf{w}_y^{\phi} = \mathbf{S}_y\boldsymbol{\beta}$, where $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{R}^N$.

Therefore, the objective function (6.15.33) for linearly correlated projections becomes the objective function for nonlinearly correlated projections below [303]:

$$L(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{\left\langle \mathbf{S}_x^T\mathbf{w}_x^{\phi}, \mathbf{S}_y^T\mathbf{w}_y^{\phi}\right\rangle}{\|\mathbf{S}_x^T\mathbf{w}_x^{\phi}\| \cdot \|\mathbf{S}_y^T\mathbf{w}_y^{\phi}\|} = \frac{\boldsymbol{\alpha}^T\mathbf{S}_x^T\mathbf{S}_x\mathbf{S}_y^T\mathbf{S}_y\boldsymbol{\beta}}{\|\boldsymbol{\alpha}^T\mathbf{S}_x^T\mathbf{S}_x\| \cdot \|\boldsymbol{\beta}^T\mathbf{S}_y^T\mathbf{S}_y\|}. \tag{6.15.40}$$

Let

$$K_x(\mathbf{x}_i, \mathbf{x}_j) = \langle \boldsymbol{\phi}_x(\mathbf{x}_i), \boldsymbol{\phi}_x(\mathbf{x}_j)\rangle = \boldsymbol{\phi}_x^T(\mathbf{x}_i)\boldsymbol{\phi}_x(\mathbf{x}_j),\ i, j = 0, 1, \ldots, N-1, \tag{6.15.41}$$

$$K_y(\mathbf{y}_i, \mathbf{y}_j) = \langle \boldsymbol{\phi}_y(\mathbf{y}_i), \boldsymbol{\phi}_y(\mathbf{y}_j)\rangle = \boldsymbol{\phi}_y^T(\mathbf{y}_i)\boldsymbol{\phi}_y(\mathbf{y}_j),\ i, j = 0, 1, \ldots, N-1 \tag{6.15.42}$$

be the kernel functions on the two views, respectively.

Defining the kernel matrices

$$\begin{aligned}\mathbf{K}_x = \mathbf{S}_x^T\mathbf{S}_x &= \begin{bmatrix} \boldsymbol{\phi}_x^T(\mathbf{x}_0) \\ \boldsymbol{\phi}_x^T(\mathbf{x}_1) \\ \vdots \\ \boldsymbol{\phi}_x^T(\mathbf{x}_{N-1}) \end{bmatrix} \left[\boldsymbol{\phi}_x(\mathbf{x}_0), \boldsymbol{\phi}_x(\mathbf{x}_1), \ldots, \boldsymbol{\phi}_x(\mathbf{x}_{N-1})\right] \\ &= [K_x(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=0}^{N-1,N-1},\end{aligned} \tag{6.15.43}$$

and

$$\begin{aligned}\mathbf{K}_y = \mathbf{S}_y^T\mathbf{S}_y &= \begin{bmatrix} \boldsymbol{\phi}_y^T(\mathbf{y}_0) \\ \boldsymbol{\phi}_y^T(\mathbf{y}_1) \\ \vdots \\ \boldsymbol{\phi}_y^T(\mathbf{y}_{N-1}) \end{bmatrix} \left[\boldsymbol{\phi}_y(\mathbf{y}_0), \boldsymbol{\phi}_y(\mathbf{y}_1), \ldots, \boldsymbol{\phi}_y(\mathbf{y}_{N-1})\right] \\ &= [K_y(\mathbf{y}_i, \mathbf{y}_j)]_{i,j=0}^{N-1,N-1},\end{aligned} \tag{6.15.44}$$

then the objective function in (6.15.40) can be simplified to

$$L(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{\langle \mathbf{K}_x\boldsymbol{\alpha}, \mathbf{K}_y\boldsymbol{\beta}\rangle}{\|\mathbf{K}_x\boldsymbol{\alpha}\| \cdot \|\mathbf{K}_y\boldsymbol{\beta}\|} = \frac{\boldsymbol{\alpha}^T\mathbf{K}_x^T\mathbf{K}_y\boldsymbol{\beta}}{\sqrt{\boldsymbol{\alpha}^T\mathbf{K}_x^T\mathbf{K}_x\boldsymbol{\alpha} \cdot \boldsymbol{\beta}^T\mathbf{K}_y^T\mathbf{K}_y\boldsymbol{\beta}}}, \tag{6.15.45}$$

$$\text{subject to} \begin{cases} \boldsymbol{\alpha}^T \mathbf{K}_x^T \mathbf{K}_x \boldsymbol{\alpha} = 1, \\ \boldsymbol{\beta}^T \mathbf{K}_y^T \mathbf{K}_y \boldsymbol{\beta} = 1. \end{cases} \tag{6.15.46}$$

By the Lagrange multiplier method and the regularization method, we have the following objective function for dual optimization:

$$\begin{aligned} \mathcal{L}_{\rm D}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = {}& \boldsymbol{\alpha}^T \mathbf{K}_x^T \mathbf{K}_y \boldsymbol{\beta} - \frac{\lambda_x}{2}\left(\boldsymbol{\alpha}^T \mathbf{K}_x^T \mathbf{K}_x \boldsymbol{\alpha} - 1\right) - \frac{\nu_x}{2}\|\mathbf{S}_x^T \boldsymbol{\alpha}\|^2 \\ & - \frac{\lambda_y}{2}\left(\boldsymbol{\beta}^T \mathbf{K}_y^T \mathbf{K}_y \boldsymbol{\beta} - 1\right) - \frac{\nu_y}{2}\|\mathbf{S}_y^T \boldsymbol{\beta}\|^2. \end{aligned} \tag{6.15.47}$$

The first-order optimality conditions give the results:

$$\begin{aligned} \frac{\partial \mathcal{L}_{\rm D}}{\partial \boldsymbol{\alpha}} = \mathbf{0} \ &\Rightarrow \ \mathbf{K}_x^T \mathbf{K}_y \boldsymbol{\beta} = \lambda_x \mathbf{K}_x^T \mathbf{K}_x \boldsymbol{\alpha} + \nu_x \mathbf{K}_x^T \boldsymbol{\alpha} \\ &\Rightarrow \ \mathbf{K}_y \boldsymbol{\beta} = \lambda_x \mathbf{K}_x \boldsymbol{\alpha} + \nu_x \boldsymbol{\alpha}, \end{aligned} \tag{6.15.48}$$

$$\begin{aligned} \frac{\partial \mathcal{L}_{\rm D}}{\partial \boldsymbol{\beta}} = \mathbf{0} \ &\Rightarrow \ \mathbf{K}_y^T \mathbf{K}_x \boldsymbol{\alpha} = \lambda_y \mathbf{K}_y^T \mathbf{K}_y \boldsymbol{\beta} + \nu_y \mathbf{K}_y^T \boldsymbol{\beta} \\ &\Rightarrow \ \mathbf{K}_x \boldsymbol{\alpha} = \lambda_y \mathbf{K}_y \boldsymbol{\beta} + \nu_y \boldsymbol{\beta}. \end{aligned} \tag{6.15.49}$$

For simplicity, let $\lambda = \lambda_1 = \lambda_2$, $\nu = \nu_1 = \nu_2$, and $\kappa = \frac{\nu}{\lambda}$. Then, from (6.15.49) we get

$$\boldsymbol{\beta} = \frac{1}{\lambda}(\mathbf{K}_y + \kappa \mathbf{I})^{-1} \mathbf{K}_x \boldsymbol{\alpha}. \tag{6.15.50}$$

Substituting this result into (6.15.48), we obtain

$$(\mathbf{K}_x + \kappa \mathbf{I})^{-1} \mathbf{K}_y (\mathbf{K}_y + \kappa \mathbf{I})^{-1} \mathbf{K}_x \boldsymbol{\alpha} = \lambda^2 \boldsymbol{\alpha}, \tag{6.15.51}$$

or

$$\mathbf{K}_y (\mathbf{K}_y + \kappa \mathbf{I})^{-1} \mathbf{K}_x \boldsymbol{\alpha} = \lambda^2 (\mathbf{K}_x + \kappa \mathbf{I}) \boldsymbol{\alpha}. \tag{6.15.52}$$

This implies that a number of $\boldsymbol{\alpha}$ (and corresponding λ) can be found by solving the eigenvalue problem (6.15.51) or the generalized eigenvalue problem (6.15.52).

Once $\boldsymbol{\alpha}$ is found, Eq. (6.15.50) can be used to find the unique $\boldsymbol{\beta}$ for each $\boldsymbol{\alpha}$. That is to say, in addition to the most strongly correlated pair of projections, one can also identify the correlations between other pairs of projections and the strength of the correlations can be measured by the values of λ.

In the jth projection, the similarity between an original unlabeled instance $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1, 2, \ldots, N-1$ and the original labeled instance $(\mathbf{x}_0, \mathbf{y}_0)$ can be measured by [303]

$$\begin{aligned}\text{sim}_{i,j} &= \exp(-d^2(P_j(\mathbf{x}_i), P_j(\mathbf{x}_0))) + \exp(-d^2(P_j(\mathbf{y}_i), P_j(\mathbf{y}_0))),\\ &\quad i = 1, \ldots, N-1;\ j = 1, \ldots, m,\end{aligned} \tag{6.15.53}$$

where $d(\mathbf{a}, \mathbf{b})$ is the Euclidean distance between $\mathbf{a}$ and $\mathbf{b}$, m is the number of unlabeled instances $(\mathbf{x}_i, \mathbf{y}_i)$ correlating with the labeled instance $(\mathbf{x}_0, \mathbf{y}_0)$, while $(P_j(\mathbf{x}_i), P_j(\mathbf{y}_i))$ is the projection of the instance $(\mathbf{x}_i, \mathbf{y}_i)$ on higher-dimensional kernel instances, $\mathbf{K}_x\boldsymbol{\alpha}$ and $\mathbf{K}_y\boldsymbol{\beta}$.

6.15.3 Penalized Canonical Correlation Analysis

Given two-view data matrices $(\mathbf{X}, \mathbf{Y})$, the standard CCA seeks to find $(\mathbf{u}, \mathbf{v})$ that maximize $\text{cor}(\mathbf{Xu}, \mathbf{Yv}) = \mathbf{u}^T\mathbf{XYv}$:

$$\max_{\mathbf{u},\mathbf{v}}\ \mathbf{u}^T\mathbf{XYv} \quad \text{subject to } \|\mathbf{Xu}\|_2^2 \le 1 \text{ and } \|\mathbf{Yv}\|_2^2 \le 1. \tag{6.15.54}$$

If including penalties $p_1(\mathbf{u}) \le c_1$ and $p_2(\mathbf{v}) \le c_2$ in the CCA then one obtains a *penalized canonical correlation analysis* (penalized CCA) in [271]:

$$\begin{aligned}&\max_{\mathbf{u},\mathbf{v}}\ \mathbf{u}^T\mathbf{XYv}\\ &\text{subject to } \|\mathbf{Xu}\|_2^2 \le 1,\ \|\mathbf{Yv}\|_2^2 \le 1,\ p_1(\mathbf{u}) \le c_1,\ p_2(\mathbf{v}) \le c_2.\end{aligned} \tag{6.15.55}$$

If pre-normalizing the column vectors of the data matrices $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{N\times P}$ such that they become the semi-orthogonal, i.e., $\mathbf{X}^T\mathbf{X} = \mathbf{I}_{P\times P}$ and $\mathbf{Y}^T\mathbf{Y} = \mathbf{I}_{P\times P}$, then the penalized CCA is simplified to [271]

$$\begin{aligned}&\max_{\mathbf{u},\mathbf{v}}\ \mathbf{u}^T\mathbf{XYv}\\ &\text{subject to } \|\mathbf{u}\|_2^2 \le 1,\ \|\mathbf{v}\|_2^2 \le 1,\ p_1(\mathbf{u}) \le c_1,\ p_2(\mathbf{v}) \le c_2\end{aligned} \tag{6.15.56}$$

because $\|\mathbf{Xu}\|_2^2 = \mathbf{u}^T\mathbf{X}^T\mathbf{Xu} = \|\mathbf{u}\|_2^2$ and $\|\mathbf{Yv}\|_2^2 = \|\mathbf{v}\|_2^2$.

Equation (6.15.56) is called "*diagonal penalized CCA*." If penalties $p_1(\mathbf{u}) = \|\mathbf{u}\|_1$ and $p_2(\mathbf{v}) = \|\mathbf{v}\|_1$, then the diagonal penalized CCA become the *sparse CCA* that gives the *sparse canonical variants* $\mathbf{u}$ and $\mathbf{v}$.

Let soft denote the soft thresholding operator defined by

$$\text{soft}(a, c) = \text{sign}(a)(|a| - c)_+, \tag{6.15.57}$$

where $c > 0$ is a constant and

$$x_+ = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{if } x \leq 0. \end{cases} \tag{6.15.58}$$

Lemma 6.4 ([271]) *For the optimization problem*

$$\max_{\mathbf{u}} \ \mathbf{u}^T \mathbf{a} \quad \text{subject to } \|\mathbf{u}\|_2^2 \leq 1, \ \|\mathbf{u}\|_1 \leq c, \tag{6.15.59}$$

its solution is given by

$$\mathbf{u} = \frac{\mathbf{s}(\mathbf{a}, \Delta)}{\|\mathbf{s}(\mathbf{a}, \Delta)\|_2}, \tag{6.15.60}$$

where $\mathbf{s}(\mathbf{a}, \Delta) = [\text{soft}(a_1, \Delta), \ldots, \text{soft}(a_p, \Delta)]^T$, *and* $\Delta = 0$ *if it results in* $\|\mathbf{u}\|_1 \leq c$, *otherwise* Δ *is chosen to be a positive constant such that* $\|\mathbf{u}\|_1 = c$.

For the penalized (sparse) CCA problem

$$\begin{aligned} &(\mathbf{u}, \mathbf{v}) = \arg\max_{\mathbf{u},\mathbf{v}} \mathbf{u}^T \mathbf{X} \mathbf{v} \\ &\text{subject to } \|\mathbf{u}\|_2^2 \leq 1, \ \|\mathbf{v}\|_2^2 \leq 1, \ \|\mathbf{u}\|_1 \leq c_1, \|\mathbf{v}\|_1 \leq c_2. \end{aligned} \tag{6.15.61}$$

Algorithm 6.30 [271] gives the sparse canonical variants $\mathbf{u}_i, \mathbf{v}_i, \ i = 1, \ldots, K$.

Algorithm 6.30 Penalized (sparse) canonical component analysis (CCA) algorithm

```
1.  input: The matrix $\mathbf{X}$.
2.  initialization: Make the truncated SVD $\mathbf{X} = \sum_{i=1}^K d_i \mathbf{u}_i \mathbf{v}_i$. Put $\mathbf{X}_1 = \mathbf{X}$.
3.  for $k = 1$ to $K$ do
4.    $\mathbf{u}_k \leftarrow \frac{\mathbf{s}(\mathbf{X}_k \mathbf{v}_k, \Delta_1)}{\|\mathbf{s}(\mathbf{X}_k \mathbf{v}_k, \Delta_1)\|_2}$, where $\Delta_1 = 0$ if this results in $\|\mathbf{u}\|_1 \leq c_1$; otherwise, $\Delta_1$ is chosen
      to be a positive constant such that $\|\mathbf{u}\|_1 = c_1$.
5.    $\mathbf{u}_k \leftarrow \mathbf{u}_k / \|\mathbf{u}_k\|_2$.
6.    $\mathbf{v}_k \leftarrow \frac{\mathbf{s}(\mathbf{X}_k^T \mathbf{u}_k, \Delta_2)}{\|\mathbf{s}(\mathbf{X}_k^T \mathbf{u}_k, \Delta_2)\|_2}$, where $\Delta_2 = 0$ if this results in $\|\mathbf{v}\|_1 \leq c_2$; otherwise, $\Delta_2$ is chosen
      to be a positive constant such that $\|\mathbf{v}\|_1 = c_2$.
7.    $\mathbf{v}_k \leftarrow \mathbf{v}_k / \|\mathbf{v}_k\|_2$.
8.    if $\mathbf{u}_k, \mathbf{v}_k$ are converged then do
9.      $d_k = \mathbf{u}_k^T \mathbf{X}_k \mathbf{v}_k$,
10.   else goto Step 4.
11.   end if
12.   exit if $k = K$.
13.   $\mathbf{X}_{k+1} \leftarrow \mathbf{X}_k - d_k \mathbf{u}_k \mathbf{v}_k^T$.
14. end for
15. output: $\mathbf{u}_k, \mathbf{v}_k, \ k = 1, \ldots, K$.
```

6.16 Graph Machine Learning

Many practical applications in machine learning build on irregular or *non-Euclidean structures* rather than regular or *Euclidean structures*. Non-Euclidean structures are also called *graph structures* because this structure is a topological map in abstract sense of graph theory. Prominent examples of underlying *graph-structured data* are social networks, information networks, gene data on biological regulatory networks, text documents on word embeddings [73], log data on telecommunication networks, genetic regulatory networks, functional networks of the brain, 3D shapes represented as discrete manifolds, and so on [158]. Moreover, the signal values associated with the vertices of the graph carry the information of interest in observations or physical measurements. Numerous examples can be found in real world applications, such as temperatures within a geographical area, transportation capacities at hubs in a transportation network, or human behaviors in a social network [76].

When signal values are defined on the vertex set of a weighted and undirected graph, structured data are referred to as *graph signals*, where the vertices of the graph represent the entities and the edge weights reflect the pairwise relationships or similarities between these entities.

Graphs can encode complex geometric structures and can be studied with strong mathematical tools such as *spectral graph theory* [53]. One of the main goals in spectral graph theory is to deduce the principal properties and structure of a graph from its graph spectrum. Spectral graph theory contains three basic aspects: similarity graph, graph Laplacian matrices, and graph spectrum.

This section focuses on graph-based machine learning and contains the following two parts:

- *Spectral graph theory:* Similarity graph, graph Laplacian matrices, and graph spectrum.
- *Graph-based learning:* Learning the structure of a graph, called also graph construction, from training samples in semi-supervised and unsupervised learning cases.

6.16.1 Graphs

The standard *Euclidean structure* refers to a space structure on n-dimensional vector space $\mathbb{R}^n$ equipped with the following four *Euclidean measures*:

1. the standard inner product (also known as the dot product) on $\mathbb{R}^m$,

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T \mathbf{y} = x_1 y_1 + \cdots + x_m y_m;$$

2. the Euclidean length of a vector $\mathbf{x}$ on $\mathbb{R}^m$,

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{y}\rangle} = \sqrt{x_1^2 + \cdots + x_m^2};$$

3. the Euclidean distance between $\mathbf{x}$ and $\mathbf{y}$ on $\mathbb{R}^m$,

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\| = \sqrt{(x_1 - y_1)^2 + \cdots + (x_m - y_m)^2};$$

4. the (nonreflex) angle θ ($0° \leq \theta \leq 180°$) between vectors $\mathbf{x}$ and $\mathbf{y}$ on $\mathbb{R}^m$,

$$\theta = \arccos\left(\frac{\langle \mathbf{x}, \mathbf{y}\rangle}{\|\mathbf{x}\|\,\|\mathbf{y}\|}\right) = \arccos\left(\frac{\mathbf{x}^T\mathbf{y}}{\|\mathbf{x}\|\,\|\mathbf{y}\|}\right).$$

A space structure with no Euclidean measure above is known as the *non-Euclidean structure*.

Given a set of data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$ with $\mathbf{x}_i \in \mathbb{R}^d$ and some kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ between all pairs of data points $\mathbf{x}_i$ and $\mathbf{x}_j$.

The kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ must be symmetric and nonnegative:

$$K(\mathbf{x}_i, \mathbf{x}_j) = K(\mathbf{x}_j, \mathbf{x}_i) \quad \text{and} \quad K(\mathbf{x}_i, \mathbf{x}_j) \geq 0, \tag{6.16.1}$$

with $K(\mathbf{x}_i, \mathbf{x}_i) = 0$ for $i = 1, \ldots, N$.

The kernel functions satisfying the above conditions may be any distance measure $d(\mathbf{x}_i, \mathbf{x}_j)$, any similarity measure $s(\mathbf{x}_i, \mathbf{x}_j)$ or Gaussian similarity function $s(\mathbf{x}_i, \mathbf{x}_j) = \exp\big(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/(2\sigma^2)\big)$ (where the parameter σ controls the width of the neighborhoods), depending on application objects.

If we do not have more information than similarities between data points, a nice way of data representing is in the form of a weighted undirected *similarity graph* or (called simply *graph*) $G = (V, E, \mathbf{W})$ or simply $G = (V, E)$.

Definition 6.28 (Graph [241, 261]) A *graph* $G(V, E, \mathbf{W})$ or simply $G(V, E)$ is a collection of a *vertex set* (or *node set*) $V = \{v_1, \ldots, v_N\}$ and an *edge set* $E = \{e_{ij}\}_{i,j=1}^N$. A graph G contains nonnegative *edge weights* associated with each edge: $w_{ij} \geq 0$. If v_i and v_j are not connected to each other, then $w_{ij} = 0$. For an *undirected graph* G, $e_{ij} = e_{ji}$ and $w_{ij} = w_{ji}$; if G is a *directed graph*, then $e_{ij} \neq e_{ji}$ and $w_{ij} \neq w_{ji}$.

Clearly, any graph is of non-Euclidean structure.

The edge weight w_{ij} is generally treated as a measure of similarity between the nodes v_i and v_j. The higher the edge weight, the more similar the two nodes are expected to be.

Definition 6.29 (Adjacency Matrix) The adjacency matrix of a graph, denoted as $\mathbf{W} = [w_{ij}]_{i,j=1}^{N,N}$, refers to the matrix used to represent the connection of nodes in the graph. The adjacency matrix $\mathbf{W}$ can be either binary or weighted. For undirected graphs with N nodes, the adjacency matrix is an $N \times N$ real symmetric matrix.

The adjacency matrix is also called the *affinity matrix* of the graph.

Definition 6.30 (Degree Matrix) The *degree* (or degree function) of node j in the graph $G(V, E, \mathbf{W})$, denoted as $d_j : V \to \mathbb{R}$, represents the number of edges connected to node j:

$$d_j = \sum_{i \sim j} w_{ij}, \tag{6.16.2}$$

where $i \sim j$ denotes all vertices i connected to j by the edges $(i, j) \in E$. The *degree matrix* of a graph is a diagonal matrix $\mathbf{D} = \mathbf{Diag}(D_{11}, \ldots, D_{NN}) = \mathbf{Diag}(d_1, \ldots, d_N)$ whose ith diagonal element $D_{ii} = d_i$ is used to describe the degree of node i in the graph.

That is, the degree function d_i is defined by the sum of all entries of the ith row of the affinity matrix $\mathbf{W}$.

Each vertex (or node) of the graph corresponds to a datum, and the edges encode the pairwise relationships or similarities among the data. For example, the vertices of the web are just the web pages, and the edges denote the hyperlinks; in market basket analysis, the items also form a graph by connecting any two items which have appeared in the same shopping basket [301].

A signal $\mathbf{x} : V \to \mathbb{R}$ defined on the nodes of the graph may be regarded as a vector $\mathbf{x} \in \mathbb{R}^d$. Let the vertex $\mathbf{v}_i \in V = \{\mathbf{v}_1, \ldots, \mathbf{v}_N\}$ represent a data point $\mathbf{x}_i$ or a document doc$_i$. Then, each edge $e(i, j) \in E$ is assigned an affinity score w_{ij} forming matrix $\mathbf{W}$ which reflects the similarity between vertex $\mathbf{v}_i$ and vertex $\mathbf{v}_j$.

The nodes of the graph are the points in the feature space, and an edge is formed between every pair of nodes. The nonnegative weight on each edge, $w_{ij} \geq 0$, is a function of the similarity between nodes i and j. If the *edge weighting functions* $w_{ij} = 0$, then the vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ are not connected by an edge.

Clustering seeks to partition the vertex set V into disjoint sets $V_1, \ldots, V_m$, where by some measure the similarity among the vertices in a set V_i is high, and across different sets (V_i, V_j) is low, where $V_i = \{\mathbf{x}_{i1}, \ldots, \mathbf{x}_{i,m_i}\}$ is the ith clustered vertex subset such that $m_1 + \cdots + m_k = N$. Disjoint subsets $V_1, \ldots, V_m$ of the vertex set V imply that $V_1 \cup V_2 \cup \cdots \cup V_m = V$ and $V_i \cap V_j = \emptyset$ for all $i \neq j$.

Due to the assumption of undirected graph $G = (V, E)$, we require $w_{ij} = w_{ji}$. The affinity matrix or *weighted adjacency matrix* of the graph is the matrix $\mathbf{W} = [w_{ij}]_{i,j=1}^{N,N}$ with *edge weights* w_{ij} as entries

$$w_{ij} = \begin{cases} K(\mathbf{x}_i, \mathbf{x}_j), & i \neq j; \\ 0, & i = j. \end{cases} \tag{6.16.3}$$

Definition 6.31 (Weighed Graph [301]) A graph is weighted if it is associated with a function $w : V \times V \to \mathbb{R}$ satisfying

$$w_{ij} > 0, \quad \text{if } (i, j) \in E, \tag{6.16.4}$$

and

$$w_{ij} = w_{ji}. \tag{6.16.5}$$

Consider a graph with N vertices where each vertex corresponds to a data point. For two data points $\mathbf{x}_i$ and $\mathbf{x}_j$, the weight w_{ij} of an edge connecting vertices i and j has four most commonly used choices as follows [41, 232].

1. 0–1 *weighting:* $w_{ij} = 1$ if and only if nodes i and j are connected by an edge; otherwise $w_{ij} = 0$. This is the simplest weighting method and is very easy to compute.
2. *Heat kernel weighting:* If nodes i and j are connected, then

$$w_{ij} = \mathrm{e}^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma}} \tag{6.16.6}$$

   is called the *heat kernel weighting*. Here σ is some pre-selected parameter.
3. *Thresholded Gaussian kernel weighting:* The weight of an edge connecting vertices i and j is defined, via a thresholded Gaussian kernel weighting function, as

$$w_{ij} = \begin{cases} \exp\left(-\frac{[\mathrm{dist}(i,j)]^2}{2\theta^2}\right), & \mathrm{dist}(i, j) \le \kappa, \\ 0, & \text{otherwise}, \end{cases} \tag{6.16.7}$$

   for some parameters θ and κ. Here, $\mathrm{dist}(i, j)$ may represent a physical distance between vertices i and j or the Euclidean distance between two feature vectors describing i and j, the latter of which is especially common in graph-based semi-supervised learning methods.
4. *Dot-product weighting:* If nodes i and j are connected, then

$$w_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j = \mathbf{x}_i^T \mathbf{x}_j \tag{6.16.8}$$

   is known as the *dot-product weighting*. Note that if $\mathbf{x}$ is normalized to have unit norm, then the dot product of two vectors is equivalent to the cosine similarity of the two vectors.

There are different popular similarity graphs, depending on transforming a given set $\mathbf{x}_1, \ldots, \mathbf{x}_N$ of data points with pairwise similarities s_{ij} or pairwise distances d_{ij} into a graph. The following are three popular methods for constructing similarity graphs [257].

1. *ϵ-neighborhood graph:* All points are connected with pairwise distances smaller than ϵ. As the distances between all connected points are roughly of the same scale (at most ϵ), weighting the edges would not incorporate more information about the data to the graph. Hence, the ϵ-neighborhood graph is usually considered as an unweighted graph.
2. *K-nearest neighbor graph:* Connect vertex $\mathbf{v}_i$ with vertex $\mathbf{v}_j$ if $\mathbf{v}_j$ is among the K-nearest neighbors of $\mathbf{v}_i$. Due to the nonsymmetric neighborhood relationship, this definition leads to a directed graph. There are two methods for making this graph undirected:

   - One can simply ignore the directions of the edges, that is, $\mathbf{v}_i$ and $\mathbf{v}_j$ are connected with an undirected edge if $\mathbf{v}_i$ is among the K-nearest neighbors of $\mathbf{v}_j$ or if $\mathbf{v}_j$ is among the K-nearest neighbors of $\mathbf{v}_i$. The resulting graph is called the K-nearest neighbor graph.
   - Vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ are connected if $\mathbf{v}_i$ is among the K-nearest neighbors of $\mathbf{v}_j$ and $\mathbf{v}_j$ is among the K-nearest neighbors of $\mathbf{v}_i$, which results in a *mutual K-nearest neighbor graph*. In these cases, after connecting the appropriate vertices, the edges are weighted by the similarity of their endpoints.

3. *Fully connected graph:* Connect simply all points with positive similarity with each other, and weight all edges by s_{ij}. Since the graph should represent the local neighborhood relationships, this construction is only useful if using the similarity function itself to model local neighborhoods. The Gaussian similarity function $s(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/(2\sigma^2)\right)$ is an example of such a positive similarity function. This parameter plays a similar role as the parameter ϵ in case of the ϵ-neighborhood graph.

The main differences between the three graphs above-mentioned are as follows.

- The ϵ-neighborhood graph uses the distance measure smaller than a threshold ϵ, and is an unweighted graph.
- The K-nearest neighbor graph considers K-nearest neighbors of vertices, and the edges are weighted by the similarity of their endpoints.
- The fully connected graph uses the positive (rather than nonnegative) similarity.

6.16.2 Graph Laplacian Matrices

The key problem of learning graph topology can be casted as a problem of learning the so-called *graph Laplacian matrices* as it uniquely characterizes a graph. In other words, graph Laplacian matrices are an essential operator in spectral graph theory [53].

Definition 6.32 (Graph Laplacian Matrix) The Laplacian matrix of graph $G(V, E, \mathbf{W})$ is defined as $\mathbf{L} = \mathbf{D} - \mathbf{W}$ with elements

$$L(u, v) = \begin{cases} d_v, & \text{if } u = v; \\ -1, & \text{if } (u, v) \in E; \\ 0, & \text{otherwise.} \end{cases} \tag{6.16.9}$$

Here, d_v denotes the degree of node i. The corresponding normalized Laplacian matrix is given by [53]

$$\mathcal{L}(u, v) = \begin{cases} 1, & \text{if } u = v \text{ and } d_v \neq 0; \\ -\frac{1}{\sqrt{d_u}\sqrt{d_v}}, & (u, v) \in E; \\ 0, & \text{otherwise.} \end{cases} \tag{6.16.10}$$

The unnormalized graph Laplacian $\mathbf{L}$ is also called *combinatorial graph Laplacian*.

There are two common normalized graph Laplacian matrices [257]:

- *Symmetric normalized graph Laplacian:*

$$\mathbf{L}_{\text{sym}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}. \tag{6.16.11}$$

- *Random walk normalized graph Laplacian:*

$$\mathbf{L}_{\text{rw}} = \mathbf{D}^{-1}\mathbf{L} = \mathbf{D}^{-1}\mathbf{W}. \tag{6.16.12}$$

The quadratic form of Laplacian matrices is given by

$$\begin{aligned} \mathbf{u}^T\mathbf{L}\mathbf{u} = \mathbf{u}^T\mathbf{D}\mathbf{u} - \mathbf{u}^T\mathbf{W}\mathbf{u} &= \sum_{i=1}^N d_i u_i^2 - \sum_{j=1}^N \left(\sum_{i=1}^N u_i w_{ij}\right) u_j \\ &= \sum_{i=1}^N d_i u_i^2 - \sum_{i=1}^N \sum_{j=1}^N u_i u_j w_{ij} \\ &= \frac{1}{2}\left(\sum_{i=1}^N d_i u_i^2 - 2\sum_{i=1}^N \sum_{j=1}^N u_i u_j w_{ij} + \sum_{j=1}^N d_i u_j^2\right) \\ &= \frac{1}{2}\left(\sum_{i=1}^N u_i^2 \sum_{j=1}^N w_{ij} - 2\sum_{i=1}^N \sum_{j=1}^N u_i u_j w_{ij} + \sum_{j=1}^N u_j^2 \sum_{i=1}^N w_{ij}\right), \end{aligned}$$

namely

$$\mathbf{u}^T\mathbf{L}\mathbf{u} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}(u_i - u_j)^2 \geq 0, \tag{6.16.13}$$

the equality holds if and only if $u_i = u_j, \forall i, j \in \{1, \ldots, N\}$, i.e., $\mathbf{u} = \mathbf{1}_N = [1, \ldots, 1]^T \in \mathbb{R}^N$.

If $\mathbf{u} = \mathbf{1}_N$, then

$$[\mathbf{L}\mathbf{u}]_i = [(\mathbf{D} - \mathbf{W})\mathbf{u}]_i = \sum_{j=1}^{N} d_{ij}u_j - \sum_{j=1}^{N} w_{ij}u_j = d_i - d_i = 0$$

because $u_j \equiv 1$, $\sum_{j=1}^{N} d_{ij} = d_i$, and $\sum_{j=1}^{N} w_{ij} = d_i$. Therefore, we have

$$\mathbf{L}\mathbf{u} = \mathbf{0} \overset{\mathbf{Lu}=\lambda\mathbf{u}}{\Longleftrightarrow} \lambda = 0 \quad \text{for } \mathbf{u} = \mathbf{1}_N. \tag{6.16.14}$$

Equations (6.16.13) and (6.16.14) give the following important properties of Laplacian matrices.

- The Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is a positive semi-definite matrix.
- The minimum eigenvalue of the Laplacian matrix $\mathbf{L}$ is 0, and the eigenvector corresponding to the minimum eigenvalue is a vector with all values of 1.

6.16.3 Graph Spectrum

As the combinatorial graph Laplacian and the two normalized graph Laplacians are all real symmetric positive semi-definite matrices, they admit an eigendecomposition $\mathbf{L} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$ with $\mathbf{U} = [\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{n-1}] \in \mathbb{R}^{n\times n}$ and $\boldsymbol{\Lambda} = \mathbf{Diag}(\lambda_0, \lambda_1, \ldots, \lambda_{n-1}) \in \mathbb{R}^{n\times n}$. A complete set of orthonormal eigenvectors $\{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{n-1}\}$ is known as the *graph Fourier basis*. The multiplicity of a zero Laplacian eigenvalue is equal to the number of connected components of the graph, and hence the real nonnegative eigenvalues are ordered as $0 = \lambda_0 \leq \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n-1} = \lambda_{\max}$. The set of the graph Laplacian eigenvalues, $\sigma(L) = \{\lambda_0, \lambda_1, \ldots, \lambda_{n-1}\}$, is usually referred to as the *graph spectrum* of $\mathbf{L}$ (or the spectrum of the associated graph G).

Therefore, in graph signal processing and machine learning, eigenvalues of the Laplacian $\mathbf{L}$ are usually ordered increasingly, respecting multiplicities. By "the first k eigenvectors" we refer to the eigenvectors corresponding to the k smallest eigenvalues [257].

Proposition 6.4 ([53, 184]) *The graph Laplacian* $\mathbf{L}$ *satisfies the following properties:*

(1) For every vector $\mathbf{f} \in \mathbb{R}^n$ *and any graph Laplacian among* $\mathbf{L}, \mathbf{L}_{sys}, \mathbf{L}_{rw}$, *we have*

$$\mathbf{f}^T \mathbf{L}\mathbf{f} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(f_i - f_j)^2.$$

(2) Any graph Laplacian among $\mathbf{L}, \mathbf{L}_{sys}, \mathbf{L}_{rw}$ *is symmetric and positive semi-definite.*

(3) The smallest eigenvalue of either $\mathbf{L}$ *or* $\mathbf{L}_{rw}$ *is* 0, *and* $\mathbf{1}$ *is an eigenvector associated with eigenvalue* 0 *(here* $\mathbf{1}$ *is a constant vector whose entries are all* 1*). Although* 0 *is also an eigenvalue of* $\mathbf{L}_{norm}$, *its associated eigenvector is* $\mathbf{D}^{1/2}\mathbf{1}$.

(4) If a graph Laplacian has k eigenvalues 0, *then it has* $N - k$ *positive, real-valued eigenvalues.*

The irrelevant eigenvectors are the ones that correspond to the several smallest eigenvalues of the Laplacian matrix $\mathbf{L}$ except for the smallest eigenvalue equal to 0. For computational efficiency, we thus often focus on calculating the eigenvectors corresponding to the largest several eigenvalues of the Laplacian matrix $\mathbf{L}$. These results imply the following graph-based methods.

1. *Graph principal component analysis:* For given data vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ and a graph $G(V, E, \mathbf{W})$ with Laplacian $\mathbf{L}$, if p is the number of eigenvalues equal to 0 and q is the number of the smallest eigenvalues not equal to zero, then the Laplacian $\mathbf{L}$ has $N - (p + q)$ dominant eigenvalues. The corresponding eigenvectors of $\mathbf{L}$ give the graph principal component analysis (GPCA) after principal components in the standard PCA (see Sect. 6.8) are replaced by principal components of the Laplacian $\mathbf{L}$.
2. *Graph minor component analysis:* If the minor components in standard minor component analysis (MCA) (see Sect. 6.8) are replaced by the relevant eigenvectors corresponding to the q smallest eigenvalues of $\mathbf{L}$ not equal to zero, then the standard MCA is extended to the *graph minor component analysis* (GMCA).
3. *Graph K-means clustering:* If there are K distinct clustered regions within the N data samples, then there are $K = N - (p+q)$ dominant nonnegative eigenvalues that provide a means of estimating the possible number of clusters within the data samples in graph K-means clustering.

Definition 6.33 (Edge Derivative [301]) Let $e = (i, j)$ denote the edge between vertices i and j. The *edge derivative* of function f with respect to e at the vertex i is defined as

$$\left.\frac{\partial f}{\partial e}\right|_i = \sqrt{\frac{w_{ij}}{d_i}} f_i - \sqrt{\frac{w_{ij}}{d_j}} f_j. \tag{6.16.15}$$

The local variation of f at each vertex j is then defined to be

$$\|\nabla_j f\| = \sqrt{\sum_{e \vdash j} \left(\left. \frac{\partial f}{\partial e} \right|_j \right)^2}, \tag{6.16.16}$$

where $e \vdash j$ denotes the set of the edges incident with vertex j.

The smoothness of f is then naturally measured by the sum of the local variations at each vertex:

$$S(f) = \frac{1}{2} \sum_j \|\nabla_j f\|^2. \tag{6.16.17}$$

The graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is a difference operator, as, for any signal $\mathbf{f} \in \mathbb{R}^n$, it satisfies

$$(\mathbf{Lf})(i) = \sum_{j \in \mathcal{N}_i} w_{ij}[f_i - f_j], \tag{6.16.18}$$

where the neighborhood $\mathcal{N}_i$ is the set of vertices connected to vertex i by an edge.

The classical Fourier transform

$$\hat{f}(\xi) = \langle f, \mathrm{e}^{j2\pi\xi t} \rangle = \int_{\mathbb{R}} f(t) \mathrm{e}^{-j2\pi\xi t} \mathrm{d}t \tag{6.16.19}$$

is the expansion of a function f in terms of the complex exponentials, which are the eigenfunctions of the one-dimensional (1-D) Laplace operator

$$-L(\mathrm{e}^{j2\pi\xi t}) = -\frac{\partial^2}{\partial t^2} \mathrm{e}^{j2\pi\xi t} = (2\pi\xi)^2 \mathrm{e}^{j2\pi\xi t}. \tag{6.16.20}$$

Similarly, the *graph Fourier transform* $\hat{f}(\lambda)$ of any function $\mathbf{f} = [f(0), f(1), \ldots, f(n-1)]^T \in \mathbb{R}^n$ on the vertices of G is defined as the expansion of $\mathbf{f}$ in terms of the eigenvectors $\mathbf{u}_l = [u_l(0), u_l(1), \ldots, u_l(n-1)]^T$ of the graph Laplacian:

$$\hat{f}(\lambda_l) = \langle \mathbf{f}, \mathbf{u}_l \rangle = \sum_{i=0}^{n-1} f_i u_l^*(i), \; l = 0, 1, \ldots, n-1, \tag{6.16.21}$$

or written as matrix-vector form

$$\hat{\mathbf{f}} = \mathbf{U}^H \mathbf{f} \in \mathbb{R}^n, \; \text{where } \mathbf{U} = [\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{n-1}]. \tag{6.16.22}$$

The *graph inverse Fourier transform* is then given by

$$f_i = \sum_{l=0}^{n-1} \hat{f}(\lambda_l) u_l(i),\ i = 0, 1, \ldots, n-1 \quad \text{or} \quad \mathbf{f} = \mathbf{U}\hat{\mathbf{f}}. \tag{6.16.23}$$

In classical Fourier analysis, the eigenvalues $\{(2\pi\xi)^2\}_{\xi\in\mathbb{R}}$ carry a specific notion of frequency: for ξ close to zero (low frequencies), the associated complex exponential eigenfunctions are smooth, slowly oscillating functions. On the contrary, if ξ is far from zero (high frequencies), then the associated complex exponential eigenfunctions oscillate much more rapidly. In the graph setting, the graph Laplacian eigenvalues and eigenvectors provide a similar notion of frequency [232]: the graph Laplacian eigenvectors associated with low frequencies λ_l, vary slowly across the graph, i.e., if two vertices are connected by an edge with a large weight, then the eigenvectors at those locations are likely to be similar. The eigenvectors associated with larger eigenvalues oscillate more rapidly and are more likely to have dissimilar values on vertices connected by an edge with high weight.

6.16.4 Graph Signal Processing

In this subsection, we focus on some basic operations in graph signal processing.

Graph Filtering

In classical signal processing, given an input time signal $f(t)$ and a time-domain filter $h(t)$, the frequency filtering is defined as

$$\hat{f}_{\text{out}}(\xi) = \hat{f}_{\text{in}}(\xi)\hat{h}(\xi), \tag{6.16.24}$$

where $\hat{f}_{\text{in}}(\xi)$ and $\hat{f}_{\text{out}}(\xi)$ are the spectrum of the input and output signals, respectively; while $\hat{h}(\xi)$ is the transfer function of the filter. Taking an inverse Fourier transform of (6.16.24), we have the time filtering given by

$$f_{\text{out}}(t) = \int_{\mathbb{R}} \hat{f}_{\text{in}}(\xi)\hat{h}(\xi)\mathrm{e}^{j2\pi\xi t}\mathrm{d}\xi = \int_{\mathbb{R}} f_{\text{in}}(\tau)h(t-\tau)\,\mathrm{d}\tau = (f * h)(t), \tag{6.16.25}$$

where

$$(f * h)(t) = \int_{\mathbb{R}} f_{\text{in}}(\tau)h(t-\tau)\mathrm{d}\tau \tag{6.16.26}$$

denotes the convolution product of the continue signal $f(t)$ and the continue filter $h(t)$.

For discrete signal $\mathbf{f} = [f_1, \ldots, f_N]^T$, (6.16.25) gives the filtering result of discrete signals

$$f_{\text{out}}(i) = (f * h)(i) = \sum_{k=0}^{N-1} f(k)h(i-k) = \sum_{k=0}^{N-1} h(k)f(i-k). \tag{6.16.27}$$

Similar to (6.16.24), the graph spectral filtering can be defined as

$$\hat{f}_{\text{out}}(\lambda_\ell) = \hat{f}_{\text{in}}(\lambda_\ell)\hat{h}(\lambda_\ell), \tag{6.16.28}$$

where λ_ℓ is the ℓth eigenvalue of the Laplacian matrix $\mathbf{L}$. Taking the inverse Fourier transform of the above equation, we have the discrete-time graph filtering

$$f_{\text{out}}(i) = \sum_{\ell=0}^{N-1} \hat{f}_{\text{in}}(\lambda_\ell)\hat{h}(\lambda_\ell)u_\ell(i), \tag{6.16.29}$$

where $u_\ell(i)$ is the ith element of $N \times 1$ eigenvector $\mathbf{u}_\ell = [u_\ell(1), \ldots, u_\ell(N)]^T$ corresponding to the eigenvalue λ_ℓ.

Graph Convolution

Because the graph signal is discrete, the graph filtering (6.16.29) can be represented using the same convolution as discrete signals, i.e.,

$$f_{\text{out}}(i) = (f_{\text{in}} * h)_G(i), \tag{6.16.30}$$

But, the graph convolution does not have the standard form of classical convolution $(f * h)(i) = \sum_{k=0}^{N-1} f(k)h(i-k) = \sum_{k=0}^{N-1} h(k)f(i-k)$, because there is no delayed form $f(i-k)$ for any graphic signal $f(i)$.

By comparing (6.16.30) with (6.16.25), we get the convolution product of graph signals as follows:

$$(f * h)_G(i) = \sum_{\ell=0}^{N-1} \hat{f}_{\text{in}}(\lambda_\ell)\hat{h}(\lambda_\ell)u_\ell(i). \tag{6.16.31}$$

When using the Laplacian matrix $\mathbf{L}$ to represent the graph convolution, we denote

$$\mathbf{f}_{\text{in}} = [f_{\text{in}}(1), \ldots, f_{\text{in}}(N)]^T, \tag{6.16.32}$$

$$\mathbf{f}_{\text{out}} = [f_{\text{out}}(1), \ldots, f_{\text{out}}(N)]^T, \tag{6.16.33}$$

$$\hat{h}(\mathbf{L}) = \mathbf{U} \begin{bmatrix} \lambda_0 & & 0 \\ & \ddots & \\ 0 & & \lambda_{N-1} \end{bmatrix} \mathbf{U}^H. \tag{6.16.34}$$

Therefore, the graph convolution (6.16.31) can be rewritten in matrix-vector form as [232]

$$(\mathbf{f} * \mathbf{h})_G = \hat{h}(\mathbf{L})\mathbf{f}_{\text{in}}. \tag{6.16.35}$$

That is to say, the graph filtering can be represented, in Laplacian form, as follows:

$$\mathbf{f}_{\text{out}} = \hat{h}(\mathbf{L})\mathbf{f}_{\text{in}}. \tag{6.16.36}$$

p-Dirichlet Norm

Consider the approximation problem of discrete functions on a graph G.

A lot of learning algorithms operate on input spaces $\mathcal{X}$ other than $\mathbb{R}^n$, specifically, discrete input spaces, such as strings, graphs, trees, automata, etc. Such an input space is known as the manifold. In Euclidean space, the smoothness of a vector function $\mathbf{f}$ is usually represented in Euclidean norm as $\|\mathbf{f}\|_2^2 = \sum_{i=1}^N \sum_{j=1}^N (f_j - f_i)^2$. However, this representation is available for graph signals in manifold.

Consider an undirected unweighted graph $G(V, E, \mathbf{W})$ consisting of a set of vertices V numbered 1 to n, and a set of edges E (i.e., pairs (i, j)) where $i, j \in V$ and $(i, j) \in E \Leftrightarrow (j, i) \in E$. For convenience, we will sometimes use $i \sim j$ to denote that i and j are neighbors, i.e., $(i, j) \in E$. The adjacency matrix of G is an $n \times n$ real matrix $\mathbf{W}$ whose entries $w_{ij} = 1$ if $i \sim j$, and 0 otherwise. By construction, $\mathbf{W}$ is symmetric and its diagonal entries are zero.

To approximate a function on a graph G, with the weight matrix $\mathbf{W}$, we need finding a "good" function that does not make too many "jumps" for two connected nodes i and j. This good function can be formalized by the smoothness functional $S(\mathbf{f}) = \sum_{i \sim j} w_{ij}(f_i - f_j)^2$ that should be minimized. More generally, we have the following definition of smoothness function on graphs.

Definition 6.34 ([232]) The discrete p-Dirichlet norm of the graph signal $\mathbf{f} = [f_1, \ldots, f_N]^T$ is defined by

$$S_p(\mathbf{f}) = \frac{1}{p} \sum_{i \in V} \left[\sum_{j \in \mathcal{N}_i} w_{ij}(f_j - f_i)^2 \right]^{p/2}. \tag{6.16.37}$$

The following are two common p-Dirichlet norms of the graph signal $\mathbf{f}$.

- When $p = 1$, the 1-Dirichlet norm

$$S_1(\mathbf{f}) = \sum_{i \in V} \left[\sum_{j \in \mathcal{N}_i} w_{ij}(f_j - f_i)^2 \right]^{1/2} \tag{6.16.38}$$

is the total variation of the signal with respect to the graph.

- When $p = 2$, the 2-Dirichlet norm

$$S_2(\mathbf{f}) = \frac{1}{2}\sum_{i\in V}\sum_{j\in\mathcal{N}_i} w_{ij}(f_j - f_i)^2$$
$$= \sum_{i,j\in E} w_{ij}(f_j - f_i)^2 = \mathbf{f}^T\mathbf{L}\mathbf{f} \tag{6.16.39}$$

is a vector norm of the graph signal vector **f** weighted by the Laplacian matrix **L**.

As compared with Euclidean norm, the 2-Dirichlet norm can be reviewed as the Euclidean norm weighted by adjacency w_{ij} of two nodes.

Graph Tikhonov Regularization
Given a noisy graph signal $\mathbf{y} = \mathbf{f}_0 + \mathbf{e}$, where $\mathbf{f}_0$ is a graph signal vector and **e** is uncorrelated additive Gaussian noise. In order to recover $\mathbf{f}_0$, if using the 2-Dirichlet form $S_2(\mathbf{f}) = \mathbf{f}^T\mathbf{L}\mathbf{f}$ of the graph signal **f** instead of its regularization term $\|\mathbf{f}\|_2^2$ in the Tikhonov regularization method, we have that the Tikhonov regularization on the graph signal is given by

$$\arg\min_{\mathbf{f}}\left\{\|\mathbf{f}-\mathbf{y}\|_2^2 + \lambda\mathbf{f}^T\mathbf{L}\mathbf{f}\right\}. \tag{6.16.40}$$

Proposition 6.5 ([231]) *The Tikhonov solution* $\mathbf{f}_*$ *to (6.16.40) is given by*

$$\mathbf{f}_*(i) = \sum_{\ell=0}^{N-1}\left[\frac{1}{1+\gamma\lambda_\ell}\right]\hat{y}(\lambda_\ell)u_\ell(i), \tag{6.16.41}$$

for all $i = \{1, 2, \ldots, N\}$.

Graph Kerners
Kernel-based algorithms capture the structure of $\mathcal{X}$ via the kernel $K : \mathcal{X}\times\mathcal{X}\to\mathbb{R}$. One of the most general representations of discrete metric spaces are graphs. Regularization on graphs is an important step for manifold learning.

For a Mercer kernel $K : \mathcal{X}\times\mathcal{X}\to\mathbb{R}$, there is an associated reproducing kernel Hilbert space (RKHS) $\mathcal{H}_K$ of functions $\mathcal{X}\to\mathbb{R}$ with the corresponding norm $\|\cdot\|_K$. Given a set of labeled examples $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, l$ and a set of unlabeled examples $\{\mathbf{x}_j\}_{j=l+1}^{l+u}$, we consider the following optimization problem:

$$\mathbf{f}^* = \arg\min_{\mathbf{f}\in\mathcal{H}_K}\left\{\frac{1}{l}\sum_{i=1}^{l} V(\mathbf{x}_i, y_i, \mathbf{f}) + \gamma_A\|\mathbf{f}\|_K^2 + \frac{\gamma_I}{l+u}\hat{\mathbf{f}}^T\mathbf{L}\hat{\mathbf{f}}\right\}, \tag{6.16.42}$$

where $\hat{\mathbf{f}} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_{l+u})]^T$ as $n = l + u$ and $f_i = f(\mathbf{x}_i)$ for $i = 1, \ldots, n$, and **L** is the Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{W}$ where w_{ij} are the edge weights in the data adjacency graph. Here, the diagonal matrix **D** is given by $\mathbf{D}_{l+u}$ with

$D_{ii} = \sum_{j=1}^{l+u} w_{ij}$. The normalizing coefficient $\frac{1}{(l+u)^2}$ is the natural scale factor for the empirical estimate of the Laplace operator. On a sparse adjacency graph it may be replaced by $\sum_{i=1}^{l+u} \sum_{j=1}^{l+u} w_{ij}$.

Corollary 6.1 ([235]) *Denote by* $\mathbf{P} = r(\tilde{\mathbf{L}})$ *a regularization matrix, then the corresponding kernel matrix is given by the inverse* $\mathbf{K} = r^{-1}(\tilde{\mathbf{L}})$ *or the pseudo-inverse* $\mathbf{K} = r^{\dagger}(\tilde{\mathbf{L}})$ *wherever necessary. More specifically, if* $\{(\lambda_i, \mathbf{v}_i)\}$ *constitute the eigensystem of* $\tilde{\mathbf{L}}$*, we have*

$$\mathbf{K} = \sum_{i=1}^{m} r^{-1}(\lambda_i)\mathbf{v}_i\mathbf{v}_i^T, \tag{6.16.43}$$

where $0^{-1} = 0$.

In the context of spectral graph theory and segmentation, the following graph kernel matrices are of particular interest [235]:

$$\begin{aligned}
&\mathbf{K} = (\mathbf{I} + \sigma^2\tilde{\mathbf{L}})^{-1} && \text{(Regularized Laplacian)}\\
&\mathbf{K} = \exp(\sigma^2/2\tilde{\mathbf{L}}) && \text{(Diffusion Process)}\\
&\mathbf{K} = (a\mathbf{I} - \tilde{\mathbf{L}})^p \text{ with } a \geq 2 && \text{(One-Step Random Walk)}\\
&\mathbf{K} = \cos(\tilde{\mathbf{L}}\pi/4) && \text{(Inverse Cosine).}
\end{aligned}$$

Theorem 6.7 ([20]) *The minimizer of optimization problem (6.16.42) admits an expansion*

$$f^*(\mathbf{x}) = \sum_{i=1}^{l+u} \alpha_i K(\mathbf{x}_i, \mathbf{x}) \tag{6.16.44}$$

in terms of the labeled examples $(\mathbf{x}_i, y_i\}_{i=1}^{l}$ *and unlabeled examples* $\{\mathbf{x}_j\}_{j=1}^{l+u}$.

If taking squared loss function in (6.16.42) as

$$V(\mathbf{x}_i, y_i, f) = (y_i - f(\mathbf{x}_i))^2, \tag{6.16.45}$$

then the minimizer of the $(l+u)$-dimensional expansion coefficient vector $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_{l+u}]^T$ in (6.16.44) is given by [20]

$$\boldsymbol{\alpha}^* = \left(\mathbf{JK} + \gamma_A l\mathbf{I} + \frac{\gamma_I l}{(l+u)^2}\mathbf{LK}\right)^{-1} \mathbf{y}, \tag{6.16.46}$$

where $\mathbf{K}$ is the $(l+u) \times (l+u)$ Gram matrix over labeled and unlabeled points; $\mathbf{y}$ is an $(l+u)$ dimensional label vector given by $\mathbf{y} = [y_1, \ldots, y_l, 0, \ldots, 0]^T$; and

$\mathbf{J} = \mathbf{Diag}(1, \ldots, 1, 0, \ldots, 0)$ is an $(l + u) \times (l + u)$ diagonal matrix with the first l diagonal entries as 1 and the others as 0.

Algorithm 6.31 shows the manifold regularization algorithm mentioned above.

Algorithm 6.31 Manifold regularization algorithms [20]

```
1. input: $l$ labeled examples $\{\mathbf{x}_i, y_i\}_{i=1}^{l}$ and $u$ unlabeled examples $\{\mathbf{x}_j\}_{j=1}^{l+u}$
2. Construct data adjacency graph with $(l + u)$ nodes using, e.g., $k$-nearest neighbors. Choose
   edge weights $w_{ij} = \exp\big(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/(4l)\big)$
3. Choose a kernel function $K(x, y)$ and compute the Gram matrix $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
4. Compute graph Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{W}$ where $\mathbf{D} = \mathbf{Diag}(W_{1,1}, \ldots, W_{l+u,l+u})$
5. Choose $\gamma_A$ and $\gamma_I$
6. Compute $\boldsymbol{\alpha}^* = \Big(\mathbf{JK} + \gamma_A l\mathbf{I} + \frac{\gamma_I l}{(l+u)^2}\mathbf{LK}\Big)^{-1}\mathbf{y}$
7. output: $f^*(\mathbf{x}) = \sum_{i=1}^{l+u} \alpha_i K(\mathbf{x}_i, \mathbf{x})$
```

When $\gamma_I = 0$, Eq. (6.16.42) gives zero coefficients over unlabeled data, so the coefficients over labeled data are exactly those for standard RLS.

6.16.5 Semi-Supervised Graph Learning: Harmonic Function Method

In practical applications of graph signal processing, graph pattern recognition, graph machine learning (spectral clustering, graph principal component analysis, graph K-means clustering, etc.), and so on, even graph-structured data (such as social networks, information networks, text documents, etc.) are usually given in convenient sampled data rather than the graph form. To apply the spectral graph theory to the sampled data, we must learn these data and transform them to the graph form, i.e., the weighted adjacency matrix $\mathbf{W}$ or the graph Laplacian matrix $\mathbf{L}$.

The graph learning problems can be generally thought of as looking for a function $\mathbf{f}$ which is smooth and simultaneously close to another given function $\mathbf{y}$. This view can be formalized as the following optimization problem [301]:

$$\mathbf{f} = \arg\min_{\mathbf{f}\in\ell^2(V)} \left\{ S(\mathbf{f}) + \frac{\mu}{2}\|\mathbf{f} - \mathbf{y}\|_2^2 \right\}. \tag{6.16.47}$$

The first term $S(\mathbf{f})$ measures the smoothness of the function $\mathbf{f}$, and the second term $\frac{\mu}{2}\|\mathbf{f} - \mathbf{y}\|_2^2$ measures its closeness to the given function $\mathbf{y}$. The trade-off between these two terms is captured by a nonnegative parameter μ.

We are given l labeled points $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ and u unlabeled points $\{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}\}$ with $\mathbf{x}_i \in \mathbb{R}^d$, typically $l \ll u$. Let $n = l + u$ be the total number of data points.

Consider a connected graph $G(V, E, \mathbf{W})$ with node set V corresponding to the n data points, with nodes $L = \{1, \ldots, l\}$ corresponding to the labeled points with class labels $y_1, \ldots, y_l$, and nodes $U = \{l+1, \ldots, l+u\}$ corresponding to the unlabeled points. The task of semi-supervised graph labeling is to first estimate a real-valued function $f : V \to \mathbb{R}$ on the graph G such that f satisfies following two conditions at the same time [304]:

- f_i should be close to the given labels y_i on the labeled nodes,
- $\mathbf{f}$ should be smooth on the whole graph.

The function f is usually constrained to take values

$$f_i = f_i^{(l)} = f^{(l)}(\mathbf{x}_i) \equiv y_i \tag{6.16.48}$$

on the labeled data $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$. At the same time, the function $\mathbf{f}_u = [f_{l+1}, \ldots, f_{l+u}]^T$ assigns the labels y_{l+i} to the unlabeled instances $\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}$ via $\hat{y}_{l+i} = \operatorname{sign}(f_{l+i})$ in binary classification $y_i \in \{-1, +1\}$.

An approach to graph-based semi-supervised labeling was proposed by Zhu et al. [306]. In this approach, labeled and unlabeled data are represented as vertices in a weighted graph, with edge weights encoding the similarity between instances, and the semi-supervised labeling problem is then formulated in terms of a Gaussian random field on this graph, where the mean of the field is characterized in terms of harmonic functions, and is efficiently obtained using matrix algebra methods.

It is assumed that an $n \times n$ symmetric weight matrix $\mathbf{W}$ on the edges of the graph is given. For example, when $\mathbf{x}_i \in \mathbb{R}^d$, the (i, j)th entries of the weight matrix can be

$$w_{ij} = \exp\left(-\sum_{k=1}^{d} \frac{(x_{ik} - x_{jk})^2}{\sigma_k}\right), \tag{6.16.49}$$

where x_{ik} is the kth component of instance $\mathbf{x}_i \in \mathbb{R}^d$ and $\sigma_1, \ldots, \sigma_d$ are length scale hyperparameters for each dimension. Thus, the weight matrix can be decomposed into the block form [306]:

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_{ll} & \mathbf{W}_{lu} \\ \mathbf{W}_{ul} & \mathbf{W}_{uu} \end{bmatrix}, \tag{6.16.50}$$

where

$$W_{ll}(i, j) = w_{ij}, i, j = 1, \ldots, l; \quad W_{lu}(i, j) = w_{i,l+j}, i = 1, \ldots, l; \ j = 1, \ldots, u;$$
$$W_{ul}(i, j) = w_{l+i,j}, i = 1, \ldots, u; \ j = 1, \ldots, l;$$
$$W_{uu}(i, j) = w_{l+i,l+j}, i, j = 1, \ldots, u.$$

The Gaussian function (6.16.49) shows that nearby points in Euclidean space are assigned large edge weight or other more appropriate weight values.

Let the real-valued function vector

$$\mathbf{f} = \begin{bmatrix} \mathbf{f}_l \\ \mathbf{f}_u \end{bmatrix} = [f(1), \ldots, f(l), f(l+1), \ldots, f(l+u)]^T, \tag{6.16.51}$$

where

$$\mathbf{f}_l = [f_1(l), \ldots, f_l(l)]^T \quad \text{or} \quad f_l(i) = f(i) \equiv y_i,\ i = 1, \ldots, l; \tag{6.16.52}$$

$$\mathbf{f}_u = [f_u(1), \ldots, f_u(u)]^T \quad \text{or} \quad f_u(i) = f(l+i),\ i = 1, \ldots, u. \tag{6.16.53}$$

Since $f_l(i) \equiv y_i, i = 1, \ldots, l$ are known, $\mathbf{f}_u$ is the $u \times 1$ vector to be found. Once $\mathbf{f}_u$ can be found by using the graph, then the labels to unlabeled data $\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}$ can be assigned: $\hat{y}_{l+i} = \text{sign}(f_u(i)), i = 1, \ldots, u$. Intuitively, unlabeled points that are nearby in the graph should have similar labels. This results in the quadratic energy function

$$E(f) = \frac{1}{2} \sum_{i,j} w_{ij} (f_i - f_j)^2. \tag{6.16.54}$$

By [306], the minimum energy function $\mathbf{f} = \arg\min_{\mathbf{f}|_L = \mathbf{f}^{(l)}} E(f)$ is harmonic; namely, it satisfies $\mathbf{\Delta f} = \mathbf{0}$ on unlabeled data points U, and is equal to $\mathbf{f}_l$ on the labeled data points L, where $\mathbf{\Delta} = \mathbf{D} - \mathbf{W}$ is the Laplacian of the graph G.

From $\mathbf{\Delta f} = \mathbf{0}$ or

$$\left(\begin{bmatrix} \mathbf{D}_{ll} & \\ & \mathbf{D}_{uu} \end{bmatrix} - \begin{bmatrix} \mathbf{W}_{ll} & \mathbf{W}_{lu} \\ \mathbf{W}_{ul} & \mathbf{W}_{uu} \end{bmatrix} \right) \begin{bmatrix} \mathbf{f}_l \\ \mathbf{f}_u \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{l\times 1} \\ \mathbf{0}_{u \times 1} \end{bmatrix}$$

it follows that the harmonic solution is given by [306]:

$$\mathbf{f}_u = (\mathbf{D}_{uu} - \mathbf{W}_{uu})^{-1} \mathbf{W}_{ul} \mathbf{f}_l = (\mathbf{I} - \mathbf{P}_{uu})^{-1} \mathbf{P}_{ul} \mathbf{f}_l, \tag{6.16.55}$$

where

$$\begin{aligned} \mathbf{P} = \mathbf{D}^{-1}\mathbf{W} &= \begin{bmatrix} \mathbf{D}_{ll}^{-1}\mathbf{W}_{ll} & \mathbf{D}_{ll}^{-1}\mathbf{W}_{lu} \\ \mathbf{D}_{uu}^{-1}\mathbf{W}_{ul} & \mathbf{D}_{uu}^{-1}\mathbf{W}_{uu} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{P}_{ll} & \mathbf{P}_{lu} \\ \mathbf{P}_{ul} & \mathbf{P}_{uu} \end{bmatrix}. \end{aligned} \tag{6.16.56}$$

The above discussions can be summarized into the harmonic function algorithm for semi-supervised graph learning, as shown in Algorithm 6.32.

Semi-supervised labeling is closely related to important applications. For example, in electric networks the edges of a graph G can be imagined to be resistors with conductance W, where nodes labeled 1 are connected to a positive voltage source,

Algorithm 6.32 Harmonic function algorithm for semi-supervised graph learning [306]

```
1. input: Labeled data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ and unlabeled data $\{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}\}$ with $l \ll u$.
2. Choice a Gaussian weight function $w_{ij}$ on the graph $G(E, V)$, where $i, j = 1, \ldots, n$ and
   $n = l + u$.
3. Compute the degree $d_j = \sum_{i=1} w_{ij}, j = 1, \ldots, n$.
4. Compute $\mathbf{D}_{uu} = \mathbf{Diag}(d_{l+1}, \ldots, d_{l+u})$, $\mathbf{W}_{uu} = [w_{l+i,l+j}]_{i,j=1}^{u,u}$ and $\mathbf{W}_{ul} = [w_{l+i,j}]_{i,j=1}^{u,l}$.
5. $\mathbf{P}_{uu} \leftarrow \mathbf{D}_{uu}^{-1}\mathbf{W}_{uu}, \ \mathbf{P}_{ul} \leftarrow \mathbf{D}_{uu}^{-1}\mathbf{W}_{ul}$.
6. $\mathbf{f}_u \leftarrow (\mathbf{I} - \mathbf{P}_{uu})^{-1}\mathbf{P}_{ul}\mathbf{f}_l$.
7. output: the labels $\hat{y}_{l+i} = \text{sign}(f_u(i)), \ i = 1, \ldots, u$.
```

and points labeled 0 to ground [306]. Then $f_u(i)$ is the voltage in the resulting electric network on the ith unlabeled node.

Once the labels $\hat{y}_{l+1}$ are estimated by applying Algorithm 6.32, the semi-supervised graph classification or regression problem becomes a supervised one.

6.16.6 Semi-Supervised Graph Learning: Min-Cut Method

Semi-supervised graph learning aims at constructing a graph from the labeled data $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ and unlabeled data $\{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}\}$ with $\mathbf{x}_i \in \mathbb{R}^d, i = 1, \ldots, l + u$. These data are drawn independently at random from a probability density function $p(x)$ on a domain $M \subseteq \mathbb{R}^d$.

The question is how to partition the nodes in a graph into disjoint subsets S and T such that the source s is in S and the sink t is in T. This problem is called s/t cut problem. In combinatorial optimization, the cost of an s/t cut, $C = \{S, T\}$, is defined as the sum of the costs of "boundary" edges (p, q), where $p \in S$ and $q \in T$. The *minimum cut* (min-cut) problem on a graph is to find a cut that has the minimum cost among all cuts.

Min-cut

Let $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_{l+u}\}$ be a set of $l + u$ points, and S be a smooth hypersurface that is separated into two parts S_1 and S_2 such that $X_1 = X \cap S_1 = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\}$ and $X_2 = X \cap S_2 = \{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}\}$ are the data subsets which land in S_1 and S_2, respectively.

We are given two disjoint subsets of nodes $L = \{1, \ldots, l\}$, $S = L_+$, and $T = L_-$, corresponding to the l labeled points with labels $y_1, \ldots, y_l$, i.e., $L = S \cup T$ and $S \cap T = \emptyset$. The degree of dissimilarity between these two pieces S and T can be computed as total weight of the edges that have been removed. In graph theoretic language, it is called the *cut* that is denoted by cut(S, T) and is defined as [230]:

$$\text{cut}(S, T) = \sum_{i \in S, j \in T} w_{ij}. \tag{6.16.57}$$

The optimal binary classification of a graph is to minimize this cut value, resulting in a most popular method for finding the minimum cut of a graph.

Normalized Cut

However, the minimum cut criteria favors cutting small sets of isolated nodes in the graph [279]. To avoid this unnatural bias for partitioning out small sets of points, Shi and Malik [230] proposed the normalized cut (Ncut) as the disassociation measure to compute the cut cost as a fraction of the total edge connections to all the nodes in the graph, instead of looking at the value of total edge weight connecting the two partitions (S, T).

Definition 6.35 (Normalized Cut [230]) The *normalized cut* (Ncut) of two disjointed partitions $S \cup T = L$ and $S \cap T = \emptyset$ is denoted by $\mathrm{Ncut}(S, T)$ and is defined as

$$\mathrm{Ncut}(S,T) = \frac{\mathrm{cut}(S,T)}{\mathrm{assoc}(S,L)} + \frac{\mathrm{cut}(S,T)}{\mathrm{assoc}(T,L)}, \tag{6.16.58}$$

where

$$\mathrm{assoc}(S,L) = \sum_{i\in S,\, j\in L} w_{ij} \quad \text{and} \quad \mathrm{assoc}(T,L) = \sum_{i\in T,\, j\in L} w_{ij} \tag{6.16.59}$$

are, respectively, the total connections from nodes in S and T to all nodes in the graph.

In the same spirit, a measure for total normalized association within groups for a given partition can be defined as [230]

$$\mathrm{Nassoc}(S,T) = \frac{\mathrm{assoc}(S,S)}{\mathrm{assoc}(S,L)} + \frac{\mathrm{assoc}(T,T)}{\mathrm{assoc}(T,L)}, \tag{6.16.60}$$

where $\mathrm{assoc}(S, S)$ and $\mathrm{assoc}(T, T)$ are total weights of edges connecting nodes within S and T, respectively.

The association and disassociation of a partition are naturally related:

$$\begin{aligned}
\mathrm{Ncut}(S,T) &= \frac{\mathrm{cut}(S,T)}{\mathrm{assoc}(S,L)} + \frac{\mathrm{cut}(S,T)}{\mathrm{assoc}(T,L)} \\
&= \frac{\mathrm{assoc}(S,L) - \mathrm{assoc}(S,S)}{\mathrm{assoc}(S,L)} + \frac{\mathrm{assoc}(T,L) - \mathrm{assoc}(T,T)}{\mathrm{assoc}(T,L)} \\
&= 2 - \left(\frac{\mathrm{assoc}(S,S)}{\mathrm{assoc}(S,L)} + \frac{\mathrm{assoc}(T,T)}{\mathrm{assoc}(T,L)} \right) \\
&= 2 - \mathrm{Nassoc}(S,T).
\end{aligned} \tag{6.16.61}$$

This means that the two partition criteria, attempting to minimize $\text{Ncut}(S, T)$ (disassociation between the groups (S, T)) and attempting to maximize $\text{Nassoc}(S, T)$ (association within the groups), are in fact identical.

Graph Mincut Learning

Consider the complete graph whose vertices are associated with the points $\mathbf{x}_i, \mathbf{x}_j$, and where the weight of the edge between $\mathbf{x}_i$ and $\mathbf{x}_j$, $i \neq j$ is given.

Let $\mathbf{f} = [f_1, \ldots, f_N]^T$ be the indicator vector for X_1:

$$f_i = \begin{cases} 1, & \text{if } x_i \in X_1, \\ 0, & \text{otherwise.} \end{cases} \tag{6.16.62}$$

There are two quantities of interest [187]:

- $\int_S p(s)\mathrm{d}s$, which measures the quality of the partition S in accordance with the weighted volume of the boundary.
- $\mathbf{f}^T\mathbf{L}\mathbf{f}$, which measures the quality of the empirical partition in terms of its cut size.

When making graph constructions, it is necessary to consider choosing which graph to use from the following three graph types [19, 173].

1. *k-nearest neighbor* (kNN) *graph:* The samples $\mathbf{x}_i$ and $\mathbf{x}_j$ are considered as neighbors if $\mathbf{x}_i$ is among the k-nearest neighbors of $\mathbf{x}_j$ or $\mathbf{x}_j$ is among the k-nearest neighbors of $\mathbf{x}_i$, where k is a positive integer and the k-nearest neighbors are measured by the usual Euclidean distance.
2. *r-ball neighborhood graph:* The samples $\mathbf{x}_i$ and $\mathbf{x}_j$ are considered as neighbors if and only if $\|\mathbf{x}_i - \mathbf{x}_j\| < r$, where the norm is the usual Euclidean norm, r is a fixed radius and two points are connected if their distance does not exceed the threshold radius r. Note that due to the symmetry of the distance we do not have to distinguish between directed and undirected graphs. The r-ball neighborhood graph is simply called the "r-neighborhood" as well.
3. *Complete weighted graph:* There is an edge $e = (i, j)$ between each pair of distinct nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ (but no loops). This graph is not considered as a neighborhood graph in general, but if the weight function is chosen in such a way that the weights of edges between nearby nodes are high and the weights between points far away from each other are almost negligible, then the behavior of this graph should be similar to that of a neighborhood graph. One such weight function is the Gaussian weight function $w_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/(2\sigma^2))$.

The k-nearest neighbor graph is a directed graph, while the r-ball neighborhood graph is an undirected graph.

The weights used on neighborhood graphs usually depend on the distance of the vertices of the edge and are non-increasing. In other words, the weight w_{ij} of an edge $(\mathbf{x}_i, \mathbf{x}_j)$ is given by $w_{ij} = f(\text{dist}(\mathbf{x}_i, \mathbf{x}_j))$ with a non-increasing weight function f. The weight functions to be considered here are the unit weight function

$f \equiv 1$, which results in the unweighted graph, and the Gaussian weight function

$$f(u) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(-\frac{1}{2}\frac{u^2}{\sigma^2}\right), \tag{6.16.63}$$

where the parameter $\sigma > 0$ defines the bandwidth of the Gaussian function.

It is usually assumed that for a measurable set $S \subseteq \mathbb{R}^d$, the surface integral along a decision boundary S, $\mu(S) = \int_S p(s)\mathrm{d}s$, is the measure on $\mathbb{R}^d$ induced by probability distribution p. This measure is called "*weighted boundary volume*."

Narayanan et al. [187] prove that the weighted boundary volume is approximated by $\frac{\sqrt{\pi}}{N\sqrt{\sigma}}\mathbf{f}^T\mathbf{L}\mathbf{f}$:

$$\int_S p(s)\mathrm{d}s \approx \frac{\sqrt{\pi}}{N\sqrt{\sigma}}\mathbf{f}^T\mathbf{L}\mathbf{f}, \tag{6.16.64}$$

where $\mathbf{L}$ is the normalized graph Laplacian, $\mathbf{f} = [f_1, \ldots, f_{l+u}]^T$ is the weight vector function, and σ is the bandwidth of the edge weight Gaussian function.

For the classification problem, we are given the training sample data as $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$, where $\mathbf{x}_i \in \mathbb{R}^d$ and N is the total number of training samples. Traditional graph construction methods typically decompose the graph construction process into two steps: graph adjacency construction and graph weight calculation [286].

1. *Graph adjacency construction:* There are two main construction methods [19]: ϵ-ball neighborhood and k-nearest neighbors.
2. *Three Graph weight calculation approaches* [286]

   - *Heat Kernel* [19]

   $$w_{ij} = \begin{cases} \frac{e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}}{t}, & \text{if } \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are neighbors;} \\ 0, & \text{otherwise,} \end{cases} \tag{6.16.65}$$

   where t is the heat kernel parameter. Note when $t \to \infty$, the heat kernel will produce binary weight and the graph constructed will be a binary graph, i.e.,

   $$w_{ij} = \begin{cases} 1, \text{ if } \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are neighbors;} \\ 0, \text{ otherwise.} \end{cases} \tag{6.16.66}$$

   - *Inverse Euclidean Distance* [63]

   $$w_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2^{-1}, \tag{6.16.67}$$

   where $\|\mathbf{x}_i - \mathbf{x}_j\|_2$ is the Euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$.

- *Local linear reconstruction coefficient* [216]

$$\xi(\mathbf{W}) = \sum_i \left\| \mathbf{x}_i - \sum_j w_{ij}\mathbf{x}_j \right\|^2, \quad \text{subject to } \sum_j w_{ij} = 1,\ \forall\, i, \tag{6.16.68}$$

  where $w_{ij} = 0$ if sample $\mathbf{x}_i$ and $\mathbf{x}_j$ are not neighbors.

If there are N samples in the original data, then *leave-one-out cross-validation* (LOOCV) uses each sample as a testing set separately, and the other $N-1$ samples are used as a training set. Therefore, LOOCV will get N models, and the average of the classification accuracy of the final testing set of N models will be used as the performance index of LOOCV classifier under this condition.

LOOCV error is defined as the count of $\mathbf{x} \in L \cup U$ with label of $\mathbf{x}$ different from label of its nearest neighbor. This is also the value of the cut.

Let $nn_{\mathbf{uv}}$ be the indicator of "$\mathbf{v}$ is the nearest neighbor of $\mathbf{u}$" for K-nearest neighborhood graph, and $w_{\mathbf{uv}}$ be the weight of the label of $\mathbf{v}$ when classifying $\mathbf{u}$ for K-averaged nearest neighborhood.

Theorem 6.8 ([29]) *Suppose edge weights between example nodes are defined as follows: for each pair of nodes* $\mathbf{u}$ *and* $\mathbf{v}$, *define* $nn_{\mathbf{uv}} = 1$ *if* $\mathbf{v}$ *is the nearest neighbor of* $\mathbf{u}$, *and* $nn_{\mathbf{uv}} = 0$ *otherwise. If defining* $w(\mathbf{u},\mathbf{v}) = nn_{\mathbf{uv}} + nn_{\mathbf{vu}}$ *for K-nearest neighborhood graph or* $w(\mathbf{u},\mathbf{v}) = nn_{\mathbf{uv}} + nn_{\mathbf{vu}}$ *for K-averaged nearest neighborhood graph, then for any binary labeling of the examples* $\mathbf{u} \in U$, *the cost of the associated cut is equal to the number of LOOCV mistakes made by 1-nearest neighbor on* $L \cup U$.

The above theorem implies that min-cut for K-nearest neighborhood graph or K-averaged nearest neighborhood graph is identical to minimizing the LOOCV error on $L \cup U$.

The following summarize the steps in *Graph Mincut Learning Algorithm* [29].

1. Construct a weighted graph $G = (V, E, \mathbf{W})$, where $V = L \cup U \cup \{v_+, v_-\}$, and $E \subseteq V \times V$. Associated with each edge $e \in E$ is a weight $w(e)$. The vertices v_+ and v_- are called the *classification vertices*, and all other vertices the *example vertices*.
2. $w(v, v_+) = \infty$ for all $v \in L_+$ and $w(v, v_-) = \infty$ for all $v \in L_-$.
3. The function assigning weights to edges between example nodes $\mathbf{x}_i, \mathbf{x}_j$ are referred to as the edge weighting function w_{ij}.
4. v_+ is viewed as the source, v_- is viewed as the sink, and the edge weights are treated as capacities. Removing the edges in the cut partitions the graph into two sets of vertices: V_+ with $v_+ \in V_+$ and V_- with $v_- \in V_-$. For concreteness, if there are multiple minimum cuts, then the algorithm chooses the one such that V_+ is smallest (this is always well defined and easy to obtain from the flow).
5. Assign a positive label to all unlabeled examples in the set V_+ and a negative label to all unlabeled examples in the set V_-.

Minimum Normalized Cut

If a node in $S = L_+$ is viewed as a source $s \in S$ and a node in $T = L_-$ is viewed as a sink $t \in T$, then minimizing Ncut(S, T) is a minimum s/t cut problem. One of the fundamental results in combinatorial optimization is [34] that the minimum s/t cut problem can be solved by finding a maximum flow from the source s to the sink t. Loosely speaking, maximum flow is the maximum "amount of water" that can be sent from the source to the sink by interpreting graph edges as directed "pipes" with capacities equal to edge weights. The theorem of Ford and Fulkerson [95] states that a maximum flow from s to t saturates a set of edges in the graph dividing the nodes into two disjoint parts $\{S, T\}$ corresponding to a minimum cut. Thus, min-cut and max-flow problems are equivalent. In this sense, min-cut, max-flow, and *min-cut/max-flow methods* refer to as the same method.

Let $\mathbf{f}$ be an $N = |L|$ dimensional indicator vector, $f_i = 1$ if node i is in L_+ and -1, otherwise. Under the assumption that $d_i = \sum_{j=1} w_{ij}$ is the total connection from node i to all other nodes, Ncut(S, T) can be rewritten as

$$\begin{aligned}\text{Ncut}(S, T) &= \frac{\text{cut}(S, T)}{\text{assoc}(S, L)} + \frac{\text{cut}(S, T)}{\text{assoc}(T, L)} \\ &= \frac{\sum_{(f_i>0, f_j<0)} -w_{ij} f_i f_j}{\sum_{f_i>0} d_i} + \frac{\sum_{(f_i<0, f_j>0)} -w_{ij} f_i f_j}{\sum_{f_i<0} d_i} \\ &= \text{Ncut}(\mathbf{f}). \end{aligned} \tag{6.16.69}$$

It was shown [230] that the minimization of normalized cut Ncut$(S, T) =$ Ncut$(\mathbf{f})$ is identical to the minimization of Rayleigh quotient $R(\mathbf{f})$:

$$\min_{\mathbf{f}} \text{ Ncut}(\mathbf{f}) = \min_{\mathbf{u}} \left\{ R(\mathbf{u}) = \frac{\mathbf{u}^T(\mathbf{D} - \mathbf{W})\mathbf{u}}{\mathbf{u}^T\mathbf{D}\mathbf{u}} \right\} \tag{6.16.70}$$

subject to the normalization constraint $\mathbf{f}^T\mathbf{1} = 0$ and $\|\mathbf{f}\| = 1$.

Proposition 6.6 ([230]) *The constrained minimization of Rayleigh quotient (6.16.70) can be formulated as a generalized eigenvalue problem* $\mathbf{Lf} = \lambda\mathbf{Df}$*: the solution is the second smallest generalized eigenvector of the matrix pencil* $(\mathbf{L}, \mathbf{D})$*, where* $\mathbf{L} = \mathbf{D} - \mathbf{W}$ *is the Laplacian in the graph G.*

In image segmentation, a key problem is how to partition the domain I of an image into subsets, and pick the "right" one. Based on the graph theoretic formulation of grouping, a minimum normalized cuts based grouping approach was proposed by Shi and Malik [230]. In this grouping, the set of vertices is partitioned into disjoint sets $V_1, V_2, \ldots, V_m$, where by some measure the similarity among the vertices in a set V_i is high and, across different sets V_i, V_j is low.

Using the relation between minimum normalized cut and the generalized eigenvalue decomposition of the matrix pencil $(\mathbf{D} - \mathbf{W}, \mathbf{D})$, a normalized cuts based algorithm for image segmentation is shown below [230].

1. Given an image or image sequence, set up a weighted graph $G(V, E, \mathbf{W})$ and set the weight on the edge connecting two nodes to be a measure of the similarity between the two nodes.
2. Solve $(\mathbf{D} - \mathbf{W})\mathbf{u} = \lambda\mathbf{D}\mathbf{u}$ for eigenvectors with the smallest eigenvalues.
3. Use the eigenvector with the second smallest eigenvalue to bipartition the graph.
4. Decide if the current partition should be subdivided and recursively repartition the segmented parts if necessary.

6.16.7 Unsupervised Graph Learning: Sparse Coding Method

In Sects. 6.16.5 and 6.16.6 we have discussed the harmonic function method and the min-cut method for semi-supervised graph learning or graph construction. Now we turn to graph construction from only unlabeled training examples.

Given a set of N d-dimension training examples $\mathbf{x}_1, \ldots, \mathbf{x}_N$ that constitute an $N \times d$ training matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$, typically $d \ll N$. Let $\mathbf{y} = [y_1, \ldots, y_d]^T$ be the target signal and $\mathbf{a} = [a_1, \ldots, a_N]^T$ be a coefficient vector such that

$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{a}. \tag{6.16.71}$$

The coefficient vector $\mathbf{a}$ can be estimated via the optimization $\min \|\mathbf{y} - \mathbf{X}\mathbf{a}\|_2^2$. The corresponding matrix equation is

$$[\mathbf{X}, \mathbf{I}_d] \begin{bmatrix} \mathbf{a} \\ \mathbf{e} \end{bmatrix} = \mathbf{y} \quad \text{or} \quad \mathbf{B}\boldsymbol{\alpha} = \mathbf{y}, \tag{6.16.72}$$

where $\mathbf{B} = [\mathbf{X}, \mathbf{I}_d] \in \mathbb{R}^{d\times(N+d)}$ with the $d \times d$ identity matrix $\mathbf{I}_d$, and $\boldsymbol{\alpha} = \begin{bmatrix} \mathbf{a} \\ \mathbf{e} \end{bmatrix} \in \mathbb{R}^{d+N}$. However, due to $d \ll N$, the matrix equation $\mathbf{B}\boldsymbol{\alpha} = \mathbf{y}$ is under-determined, so it has infinite solutions $\boldsymbol{\alpha}$.

Using the sparse coding method, the unique solution to the under-determined equation (6.16.72) can be obtained by minimizing both the reconstruction error $\|\mathbf{e}\|_2^2$ and the ℓ_1-norm of the coefficient vector $\boldsymbol{\alpha}$, and turned to solve [273]:

$$\hat{\boldsymbol{\alpha}} = \arg\min_{\boldsymbol{\alpha}} \|\boldsymbol{\alpha}\|_1 \quad \text{subject to } \mathbf{B}\boldsymbol{\alpha} = \mathbf{y}. \tag{6.16.73}$$

In unsupervised cases, the target $\mathbf{y}$ is not available, and one can think of the ith training sample $\mathbf{x}_i$ as the target $\mathbf{y}$ to transform the ℓ_1 optimization problem (6.16.73) to [286]:

$$\hat{\boldsymbol{\alpha}}_i = \arg\min_{\boldsymbol{\alpha}} \|\boldsymbol{\alpha}\|_1 \quad \text{subject to } \mathbf{B}\boldsymbol{\alpha} = \mathbf{x}_i, \tag{6.16.74}$$

where

$$\boldsymbol{\alpha}_i = \left[a_1^{(i)}, \ldots, a_{N-1}^{(i)}, e_1^{(i)}, \ldots, e_d^{(i)}\right]^T \in \mathbb{R}^{N-1+d}, \tag{6.16.75}$$

$$\mathbf{B} = [\mathbf{X} \setminus \mathbf{x}_i, \mathbf{I}_d] \in \mathbb{R}^{d\times(N-1+d)},\ i = 1, \ldots, N \tag{6.16.76}$$

with $[\mathbf{X} \setminus \mathbf{x}_i] = [\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_N]$ for $i = 1, \ldots, N$.

Furthermore, graph weights are given by $w_{ij} = |a_j^{(i)}|$. This graph obtained by using sparse coding from the training samples $\mathbf{x}_1, \ldots, \mathbf{x}_N$ is called ℓ_1 *graph* [286].

Algorithm 6.33 shows an ℓ_1 directed graph construction algorithm.

Algorithm 6.33 ℓ_1 directed graph construction algorithm [286]

```
1.  input: Column sample matrix $\mathbf{X} = [\mathbf{x}_1, \ldots \mathbf{x}_N]$.
2.  initialization: Normalize the training samples to have unit $\ell_2$ norm.
3.  for $i = 1 : N$ do
4.      Set $[\mathbf{X} \setminus \mathbf{x}_k] = [\mathbf{x}_1, \ldots, \mathbf{x}_{k-1}, \mathbf{x}_{k+1}, \ldots, \mathbf{x}_N] \in \mathbb{R}^{d\times(N-1)}$, and $\mathbf{B} = [\mathbf{X} \setminus \mathbf{x}_k, \mathbf{I}_d] \in \mathbb{R}^{d\times(d+N-1)}$.
5.      Solve the $\ell_1$ optimization problem $\hat{\boldsymbol{\alpha}} = \arg\min_{\boldsymbol{\alpha}} \|\boldsymbol{\alpha}\|_1$ subject to $\mathbf{B}\boldsymbol{\alpha} = \mathbf{x}_k$.
6.      Let $\mathbf{a} = \hat{\boldsymbol{\alpha}}_{1:N}$ and $\mathbf{e} = \hat{\boldsymbol{\alpha}}_{N+1:N+d-1}$.
7.      for $j = 1 : N - 1$ do
8.          if $j < i$ then set $w_{ij} = |a_j|$,
9.          else set $w_{ij} = |a_{j-1}|$.
10.     end for
11. end for
12. output: $\mathbf{W} = [w_{ij}]_{i,j=1}^{N,N}$.
```

6.17 Active Learning

Machine learning can be divided into "passive" learning and "active" learning from the perspective of the relationship between learners and supervisors.

- *Passive learning* is instructor/supervisor-centered, where the learners/users receive information from the instructor/supervisor and internalize it to make classification or prediction (or regression).
- *Active learning* (AL) is generally used to refer to a learning problem or system where the learner has some active or participatory role in determining on what data it will be trained. This is in contrast to passive learning, where the learner is simply presented with a training set over which it has no control [58].

Active learning problems have been studied for many decades under the rubric of experimental design from 1970s ([51, 90]).

Active learning and semi-supervised learning face the same issue: data without class labels are quite massive, and labeled data is rather scarce and hard to obtain. In these cases, we can make the learning algorithm take the initiative to propose which data to be labeled, and then add these new-labeled data to the training sample set to train the learning algorithm. This process is usually referred to as “*active learning*.” Hence, active learning is a special case of semi-supervised machine learning in which a learning algorithm is able to interactively query the user (or some other information source) to obtain the desired outputs at new data points [40]. It is quite natural to combine active learning and semi-supervised learning to address this issue from both ends.

The basic difference between active learning and semi-supervised learning is that the learner has the ability or need to influence or select its own training data in active learning, while semi-supervised learning has neither this ability nor need.

6.17.1 Active Learning: Background

In many machine learning problems, the training data are treated as a fixed and given part of the problem definition. In practice, however, the training data are often not fixed beforehand. Rather, as an active participant in the training process, the learner has an opportunity to play a role in deciding what data will be acquired for training. In active learning, the learner must take actions to gain information, and must decide what actions will give him/her the information that will best minimize future loss [58].

In many applications, unlabeled data are usually abundant but manual labeling is costly. If more sophisticated supervised (and semi-supervised) learning algorithms are used in these applications, labeled instances are very difficult, time-consuming, or costly to obtain. A few examples are given below [40]:

- *Information extraction*. Good information extraction systems must be trained by using labeled documents with detailed annotations. Locating entities and relations can take half an hour for even simple newswire stories [224]. Annotations for other knowledge domains may require additional expertise, e.g., annotating gene and disease mentions for biomedical information extraction often requires Ph.D-level biologists.
- *Classification and filtering*. Learning to classify documents (e.g., articles or web pages) or any other kind of media (e.g., image, audio, and video files) requires users to label each document or media file with particular labels, like “relevant” or “not relevant” to the query. It can be tedious and even redundant for one to have to annotate thousands of these instances .
- *Clinical named entity recognition*. Identification of clinical concepts or clinical named entity recognition (NER) is an important task for building clinical natural language processing (NLP) systems. Chen et al. [50] reported that for the annotated NER corpus from the 2010 i2b2/VA NLP challenge that contained 349 clinical documents with 20,423 unique sentences, it is very difficult to simulate experiments using an existing clinical NER corpus with annotated medical problems, treatments, and lab tests in clinical notes.

Effective management of digital video resources for human action retrieval is a difficult task, as action retrieval is more challenging than action recognition.

Relevance feedback is one common technique used for mitigating the effect of the lack of training in human action retrieval systems. In order to improve accuracy of a retrieval system, by relevance feedback the user can feedback each result returned to the retrieval system for marking them either as relevant or irrelevant to their query. Then, the retrieval system will perform an initial query and return the top T most relevant results to the user for labeling as many "relevant" or as few "irrelevant" as desired. Next, the user run the query again to generate improved results until intraclass variability is better represented in the query.

Even with relevance feedback, however, retrieval results can still be poor [135]: the amount of feedback is often very small, as users have limited patience to provide feedback to improve the query; with only few training samples the retrieval results will often still be unstable and unreliable.

Active learning is a better choice than relevance feedback for retrieval systems. Similar to relevance feedback, "*Active learning*" (also called query learning) also requires the user to label several database items according to their relevance to the query. These labels are then used to update the query and improve the retrieval results.

The key difference between relevance feedback and active learning is which database items the user provides labels for [135]:

- In relevance feedback, the user himself/herself labels the top T most relevant results returned by the retrieval system.
- In active learning, the user can (actively) query the teacher/expert for labels in order to achieve higher accuracy.

With more informative feedback, active learning has a faster learning rate, and typically performs better than relevance feedback.

Active learning is a subfield of machine learning and, more generally, artificial intelligence [40]. The key idea behind active learning is that if a machine learning algorithm is allowed to choose the data from which it learns, it will perform better with less training.

6.17.2 Statistical Active Learning

When active learning is used for classification or regression, there are three most important paradigms: membership query learning, pool-based active learning, and stream-based active learning:

- *Membership query learning [9]:* It is also called *constructive active learning*. In membership query learning, the learner is allowed to create or select unlabeled instances for the human expert to label [42].

- *Pool-based active learning [159, 176]:* It is popular for many applications such as text classification and speech recognition where unlabeled data are plentiful and cheap, but labels are expensive and slow to acquire. In pool-based active learning, the learner may not propose arbitrary points to label, but instead has access to a set of unlabeled examples, and is allowed to select which of them to request labels for.
- *Stream-based active learning [11]:* It is also called *sequential active learning* or *selective sampling*. Stream-based active learning resembles pool-based learning in many ways, except that the learner must decide whether to query or discard the unlabeled instances as a stream.

The choice of examples to label can be seen as a dilemma between the exploration and the exploitation over the data space representation [33]. Active learning aims at involving the best method to choose the data points for $\mathcal{T}_{C,i}$ (a subset chosen to be labeled) via some query strategy.

Query strategies for determining which data points should be labeled can be organized into a number of different categories [40]:

- *Uncertainty sampling* [159] is one of the most popular query criterions, and can be used to select the most uncertain instances of the unlabeled data.
- *Query by committee (QBC) method* [226] is an effective active learning method. Given a committee $C = \{\theta^{(1)}, \ldots, \theta^{(C)}\}$ of models, these models are all trained on the current labeled set $\mathcal{L}$, and are query candidates. The QBC method directly measures the voting differences of committee members, and selects samples with inconsistent voting by class labels as training data, while the most informative query is considered to be the instance about which they most disagree.
- *Expected model change method* [225] is to query the instance that would impart the greatest change to the current model if we knew its label.
- *Variance reduction method* [59] labels those points that would minimize output variance, which is one of the components of error.
- *Balance exploration and exploitation:* in this setting, the choice of examples to label is seen as a dilemma between the exploration and the exploitation over the data space representation. This strategy manages this compromise by modeling the active learning problem as a contextual conflict problem. The commonly used algorithms have the greedy algorithm, the Softmax algorithm, Bayes Bandit algorithm, etc.
- *Least certainty* selects subset $\mathcal{N}_a$ whose elements are in the bottom of the rank queue Q.
- *Middle certainty* selects subset $\mathcal{N}_a$ whose elements are in the middle of the rank queue Q.
- *Exponentiated gradient exploration* [32] can improve any active learning algorithm by an optimal random exploration.

Relying on the learner's ability to statistically model its own uncertainty, the objective of statistical active learning is usually to find model parameters that minimize some form of expected loss. The process of statistical active learning is usually as follows [58]:

1. Begin by requesting labels for a small random sub-sample of the examples and fit a statistical model to the labeled data.
2. For any $\mathbf{x}$, a statistical model is used to estimate both the conditional expectation of its label $y = y(\mathbf{x})$ and the variance of that expectation, $\sigma^2_{\hat{y}(\mathbf{x})}$.
3. Consider a candidate point $\tilde{\mathbf{x}}$, and ask what reduction in loss we would obtain if we had labeled it $\tilde{y}$.
4. Given the ability to estimate the expected effect of obtaining label $\tilde{y}$ for candidate point $\tilde{\mathbf{x}}$, repeat this computation for a sample of M candidates, and then request a label for the candidate with the largest expected decrease in loss. Add the newly labeled example to the training set, retrain, and begin looking at candidate points to add on the next iteration.

6.17.3 Active Learning Algorithms

Given a large pool $\mathcal{U}$ of unlabeled samples and a small set of labeled samples $\mathcal{L}$, with $|\mathcal{L}| \ll |\mathcal{U}|$ and $\mathcal{X} = \mathcal{L} \cup \mathcal{U}$, where $|\cdot|$ represents the cardinality of a set. Typically, *pool-based active learning* (PAL) uses $\mathcal{L}$ to train a classifier $\mathbf{h}$. Then, a query set $\mathcal{S}$ of unlabeled samples is determined for labeling based on a selection strategy Q, which uses the "knowledge" contained in $\mathbf{h}$ for sample selection, and presented to the oracle $\mathcal{O}$. The labeled samples are then added to $\mathcal{L}$ and the classifier $\mathbf{h}$ is updated.

The pool-based active learning framework can be summarized as follows [50]:

1. *Initial model generation:* At the beginning, a small number of samples are queried for annotation to build the initial model. Two common initial sampling strategies are random sampling or application-oriented sampling (e.g., longest sentence sampling for named entity recognition in clinical text).
2. *Querying:* The unannotated samples are then ranked based on the querying algorithm. Different algorithms use different query strategies for ranking.
3. *Training:* The selected unlabeled subset is retrained on the updated annotated set.
4. *Iteration:* Steps 2 and 3 are repeated until the stop criterion is met.

Pool-based active learning has been widely used in many real-world applications, e.g., text categorization [119, 250], video search [266], image classification [160], action retrieval [135], and so on.

Algorithm 6.34 is a traditional active learning approach based on the least (or middle) certainty query strategy [295].

Active learning is easily extended to the case of two-view instances. This extended active learning is called *co-active learning*, sketched in Algorithm 6.35.

Algorithm 6.34 Active learning [295]

```
1.  input: labeled dataset $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^l$ and unlabeled dataset $\mathcal{U} = \{\mathbf{x}_i\}_{i=l+1}^{l+u}$.
2.  repeat
3.    Upsample the labeled dataset $\mathcal{L}$ to obtain even class distribution $\mathcal{L}_D$;
4,    Use $\mathcal{L}/\mathcal{L}_D$ to train a classifier $\mathbf{h}$, and then classify the unlabeled dataset $\mathcal{U}$;
5.    Rank the data based on the prediction confidence value $C$ and store them in queue $Q$.
6.    Select subset $\mathcal{N}_a$ whose elements are in the bottom of the rank queue $Q$ (least certainty);
      or select $\mathcal{N}_a$ whose elements are in the middle of the rank queue $Q$ (middle certainty);
7.    Submit the selected subset $\mathcal{N}_a$ to human annotation;
8.    Remove $\mathcal{N}_a$ from the unlabeled dataset to get $\mathcal{U} = \mathcal{U} \setminus \mathcal{N}_a$;
9.    Add $\mathcal{N}_a$ to the labeled dataset $\mathcal{L}$, i.e., $\mathcal{L} = \mathcal{L} \cup \mathcal{N}_a$;
10. until unlabeled data is used up.
11. output: $\{\mathbf{x}_i, y_i\}_{i=1}^{l+u}$.
```

Algorithm 6.35 Co-active learning [295]

```
1.  input: a learning domain with features $V = \{\mathbf{x}_i, y_i\}_{i=1}^l \cup \{\mathbf{x}_i\}_{i=l+1}^{l+u}$.
2.  repeat
3.    Split the domain $V$ into two "views": $V_1 = \{\mathbf{x}_i^{(1)}\}_{i=1}^l$ and $V_2 = \{\mathbf{x}_i^{(2)}\}_{i=1}^l$, where $V_1 \cap V_2 = \emptyset$;
4.    Upsample "views" $V_1$ and $V_2$ to obtain even class distributions $V_{D_1}$ and $V_{D_2}$, respectively;
5.    Use $V/V_{D_1}$ and $V/V_{D_2}$ to train classifiers $\mathbf{h}_1$ and $\mathbf{h}_2$, respectively; and then use $\mathbf{h}_1$ and $\mathbf{h}_2$
      to classify the unlabeled dataset $\mathcal{U}$, respectively;
6.    Rank the data based on the prediction confidence value $C$ and store them in queue $Q$;
7.    Select subset $\mathcal{N}_a$ whose elements are in the middle of the rank queue $Q$ (middle certainty);
8.    Submit the selected subset $\mathcal{N}_{ca} = \mathcal{U}_{a1} \cup \mathcal{U}_{a2}$ to human annotation;
9.    Remove $\mathcal{N}_{ca}$ from the unlabeled dataset $\mathcal{U}$: $\mathcal{U} = \mathcal{U} \setminus \mathcal{N}_{ca}$;
10.   Add $\mathcal{N}_{ca}$ to the labeled dataset $\mathcal{L}$, i.e., $\mathcal{L} = \mathcal{L} \cup \mathcal{N}_{ca}$;
11. until unlabeled data is used up.
12. output: $\{\mathbf{x}_i, y_i\}_{i=1}^{l+u}$.
```

6.17.4 Active Learning Based Binary Linear Classifiers

Consider binary classification with two classes "+" and "−." We are given a small labeled instance set $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^l$ and a large unlabeled instance set $\mathcal{U} = \{\mathbf{x}_i\}_{i=l+1}^{l+u}$, where $\mathbf{x}_i \in \mathbb{R}^d$, $y_i \in \{+1, -1\}$. Our task is that given any testing instance $\mathbf{x}$, we want to make a decision on which class $\mathbf{x}$ belongs to by using pool-based active learning.

To this end, we need to find an opposite pair $\{\mathbf{x}_+, \mathbf{x}_-\}$ close to the separating hyperplane.

Let C_+ and C_- be the centroid of positive and negative labeled instances, respectively:

$$\mathbf{c}_+ = \frac{1}{|\mathcal{L}_+|}\sum_{i\in\mathcal{L}_+}\mathbf{x}_i, \quad \mathbf{c}_- = \frac{1}{|\mathcal{L}_-|}\sum_{i\in\mathcal{L}_-}\mathbf{x}_i. \tag{6.17.1}$$

When the labeled set $\mathcal{L}$ is small, we cannot use $\mathbf{x}_+ = \mathbf{c}_+$ and $\mathbf{x}_- = \mathbf{c}_-$ as the estimation results of $\mathbf{x}_+$ and $\mathbf{x}_-$, respectively, as these estimates are usually very poor.

By pool-based active learning, we synthesize the first instance $\mathbf{x}_s = \frac{1}{2}(\mathbf{x}_+ + \mathbf{x}_-)$ and query its label. Since this is a pseudo-instance, it might not be recognized by the human annotator, especially in vision-based tasks [260]. Instead, its nearest neighbor (denoted by $\mathbf{x}_q$) should be searched from the unlabeled set $\mathcal{U}$ and its label queried. If the label is positive, we use the midpoint of $\mathbf{x}_q$ and $\mathbf{c}_-$ as a new synthesized instance. Repeating this process until we can find an opposite pair $\{\mathbf{x}_+, \mathbf{x}_-\}$ close to the classification boundary.

Algorithm 6.36 shows a pool-based active learning for finding opposite pair close to separating hyperplane, which was introduced by Wang et al. [260].

Algorithm 6.36 Active learning for finding opposite pair close to separating hyperplane [260]

```
1.  input: labeled data set $\mathcal{L}^0 = \{\mathbf{x}_i, y_i\}_{i=1}^l$, unlabeled data set $\mathcal{U}^0 = \{\mathbf{x}\}_{i=l+1}^{l+u}$.
2.  initialization: let $\mathbf{x}_+^0 = \frac{1}{|\mathcal{L}_+|}\sum_{i\in\mathcal{L}_+}\mathbf{x}_i$ and $\mathbf{x}_-^0 = \frac{1}{|\mathcal{L}_-|}\sum_{i\in\mathcal{L}_-}\mathbf{x}_i$.
3.  repeat
4.    Synthesize an instance $\mathbf{x}_s = \frac{1}{2}(\mathbf{x}_+ + \mathbf{x}_-)$;
5.    Use the nearest neighbor method to compute the instance $\mathbf{x}_q$ in $\mathcal{U}$ for query, e.g.,
6.      $\mathbf{x}_q \leftarrow \arg\min_{\mathbf{x}_i\in\mathcal{U}} d(\mathbf{x}_s, \mathbf{x}_i)$;
7.    Query $\mathbf{x}_q$;
8.    $\mathcal{L} = \mathcal{L} \cup \mathbf{x}_q$ and $\mathcal{U} = \mathcal{U}\backslash\mathbf{x}_q$;
9.    if $\mathbf{x}_q$ is positive, then
10.     $\mathbf{x}_+ \leftarrow \mathbf{x}_q$.
11.   else
12.     $\mathbf{x}_- \leftarrow \mathbf{x}_q$.
13.   end if
14. until $\mathbf{x}_+$ and $\mathbf{x}_-$ are converged, respectively; or the number of binary search being run.
15. output: $(\{\mathbf{x}_+, \mathbf{x}_-\}, \mathcal{L}, \mathcal{U}) = findPair(\{\mathbf{x}_+^0, \mathbf{x}_-^0\}, \mathcal{L}^0, \mathcal{U}^0)$.
```

In the above algorithm, $d(\mathbf{a}, \mathbf{b})$ is a distance function available for practical application.

Once the enhanced labeled data set $\mathcal{L}$ and the remained unlabeled data set $\mathcal{U}$ are found by Algorithm 6.36, we can apply any semi-supervised binary classification/regression method to find the binary classifier or regressor $\mathbf{h}$. Then, for any testing instance $\mathbf{x}$, the classification $\text{sign}(\mathbf{h}^T\mathbf{x})$ or the regression $\hat{y} = \mathbf{h}^T\mathbf{x}$ can be obtained.

6.17.5 Active Learning Using Extreme Learning Machine

Suppose we are given N arbitrary distinct training instances $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $\mathbf{y}_i \in \mathbb{R}^m$.

Consider a single-hidden layer feedforward network (SLFN) with L hidden nodes that approximates these N samples with zero error: there exist $\boldsymbol{\beta}_i, a_i$, and b_i such that

$$f_L(\mathbf{x}_j) = \sum_{i=1}^{L} \boldsymbol{\beta}_i G(a_i, b_i, \mathbf{x}_j) = \mathbf{y}_j, \quad j = 1, \ldots, N, \tag{6.17.2}$$

where a_i and b_i denote the ith weight and bias of the hidden layer, respectively, and $\boldsymbol{\beta}_i$ is the weight vector connecting the ith hidden node to the output nodes.

Letting

$$\mathbf{H} = \begin{bmatrix} G(a_1, b_1, \mathbf{x}_1) & \cdots & G(a_L, b_L, \mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ G(a_1, b_1, \mathbf{x}_N) & \cdots & G(a_L, b_L, \mathbf{x}_N) \end{bmatrix} \in \mathbb{R}^{N\times L}, \tag{6.17.3}$$

$$\mathbf{B} = \begin{bmatrix} \boldsymbol{\beta}_1^T \\ \vdots \\ \boldsymbol{\beta}_L \end{bmatrix} \in \mathbb{R}^{L\times m}, \tag{6.17.4}$$

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^T \\ \vdots \\ \mathbf{y}_N \end{bmatrix} \in \mathbb{R}^{N\times m}, \tag{6.17.5}$$

then (6.17.2) can be written as the following compact matrix form:

$$\mathbf{HB} = \mathbf{Y}. \tag{6.17.6}$$

Here, $G(a_i, b_i, \mathbf{x}_j)$ denotes the activation function used to calculate the output of the ith hidden node on the jth training instance, $\mathbf{H}$ is called the hidden layer output matrix, where its ith column denotes the ith hidden node's output vector with respect to inputs $\mathbf{x}_1, \ldots, \mathbf{x}_N$ and its jth row represents the output vector of the hidden layer with respect to the input $\mathbf{x}_j$. The solution of Eq. (6.17.6) is given by $\hat{\mathbf{B}} = \mathbf{H}^\dagger\mathbf{Y}$, where $\mathbf{H}^\dagger$ is the Moore–Penrose inverse of the hidden layer output matrix $\mathbf{H}$, and $\mathbf{H}^\dagger = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T$ if $\mathbf{H}$ is of full column rank or $\mathbf{H}^\dagger = \mathbf{H}^T(\mathbf{HH}^T)^{-1}$ if $\mathbf{H}$ is of full row rank. The solution $\hat{\mathbf{H}}$ of the overdetermined matrix equation (6.17.6) is the minimum norm least squares solution, i.e., $\hat{\mathbf{B}} = \mathbf{H}^\dagger\mathbf{Y}$ has the minimum norm $\|\hat{\mathbf{H}}\| \leq \|\mathbf{Z}\|$ and the least variance $\mathrm{var}(\hat{\mathbf{H}}) \leq \mathrm{var}(\mathbf{Z})$ for all other positive solutions $\mathbf{Z}$. Because the norm of output weights $\|\mathbf{B}\|$ is closely related with the generalization ability of the neural network [16], extreme learning machine should minimize both $\|\mathbf{B}\|$ and $\|\mathbf{HB}-\mathbf{Y}\|$ simultaneously, namely

$$\min_{\mathbf{B}} \left\{ L_{\text{ELM}} = \frac{1}{2}\|\mathbf{B}\|^2 + \frac{C}{2}\|\mathbf{HB}-\mathbf{Y}\|^2 \right\}. \tag{6.17.7}$$

By the first-order optimality condition, we have

$$\frac{\partial L_{\text{ELM}}}{\partial \mathbf{B}} = \mathbf{O} \quad \Rightarrow \quad \mathbf{B} + C(\mathbf{H}^T\mathbf{HB} - \mathbf{H}^T\mathbf{Y}) = \mathbf{O}. \tag{6.17.8}$$

Letting $\lambda = 1/C$, then the solution of the optimization problem (6.17.7) is given by

$$\mathbf{B} = \begin{cases} (\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{H}^T\mathbf{Y}, \text{ if } \mathbf{H}^T\mathbf{H} \text{ is nonsingular;} \\ \\ \mathbf{H}^T(\mathbf{HH}^T + \lambda\mathbf{I})^{-1}\mathbf{Y}, \text{ if } \mathbf{HH}^T \text{ is nonsingular.} \end{cases} \tag{6.17.9}$$

In other words, the solution of the optimization problem (6.17.7) is equivalent to the Tikhonov regularization solution of the overdetermined matrix equation (6.17.6).

Define the output function (row) vector $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})] \in \mathbb{R}^{1\times m}$ and the active function (row) vector $\mathbf{h}(\mathbf{x}) = [G(a_1, b_1, \mathbf{x}), \ldots, G(a_L, b_L, \mathbf{x})] \in \mathbb{R}^{1\times L}$, where $f_i(\mathbf{x})$ denotes the actual output of the ith output node corresponding to the instance $\mathbf{x}$.

From (6.17.6) it follows $\mathbf{f}(\mathbf{x}) = \mathbf{h}(\mathbf{x})\mathbf{B} = \mathbf{y}$.

In extreme learning machine, an instance is always labeled as the category having the maximum output [123, 124]. Once the weight matrix $\mathbf{B}$ is found using (6.17.9) and a new testing instance $\mathbf{x}$ is given, one can make the binary classification $\text{label}(\mathbf{x}) = \arg\max_i f_i(\mathbf{x}),\ i = [1, \ldots, m]$ or regression $\hat{\mathbf{y}} = \mathbf{h}(\mathbf{x})\hat{\mathbf{B}}$.

By defining the posteriori probabilities of the instance $\mathbf{x}$ defined from the actual output $f(\mathbf{x})$ in extreme learning machine as

$$P(y = 1|f_i(\mathbf{x})) = \frac{1}{1 + \exp(-f_i(\mathbf{x}))} \tag{6.17.10}$$

and the normalized posteriori probabilities as

$$\bar{P}(y = 1|f_i(\mathbf{x})) = \frac{P(y = 1|f_i(\mathbf{x}))}{\sum_{j=1}^m P(y = 1|f_j(\mathbf{x}))}, \tag{6.17.11}$$

an active learning-extreme learning machine (AL-ELM) algorithm was developed by Yu et al. [291], as shown in Algorithm 6.37.

Algorithm 6.37 Active learning-extreme learning machine (AL-ELM) algorithm [291]

```
1. input: a small labeled set $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^{l}$, $\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \mathbb{R}^m$ and a large unlabeled set $\mathcal{U} = \{\mathbf{x}_i\}_{i=1}^{l+u}$ activation function $G(\mathbf{x})$, penalty factor $C$ and the number of hidden nodes $L$.
2. initialization: let $k = 0$.
   2.1 Assign random hidden nodes by randomly generating parameters $(a_i, b_i)$ according to any continuous sampling distribution, where $i = 1, \ldots, L$;
   2.2 Calculate the initial hidden layer output matrix $\mathbf{H}_0$ by using (6.17.3) from the labeled instances $\mathbf{x}_i$ in set $\mathcal{L}$;
   2.3 Use (6.17.9) to find the solution $\mathbf{B}^{(0)}$ of the matrix equation (6.17.8).
3. repeat
4.   for $k = 1$ to $M$ do
5.     Use $\mathbf{B}^{(k)}$ to calculate the actual outputs of each instance in the unlabeled set $\mathcal{U}$;
6.     Calculate converted posteriori probabilities by (6.17.10). For multiclass problems, they need to be further converted to the normalized posteriori probabilities by (6.17.11);
7.     Calculate uncertainty level of each instance in $\mathcal{U}$ by margin sampling strategy;
8.     Sort instances based on the uncertainty levels in ascending order;
9.     Extract some instances corresponding to the smallest uncertainty levels and remove them from the unlabeled set $\mathbf{U}$;
10.    Label the extracted unlabeled instances manually by the human annotators, and then put them into $\mathcal{L}$;
11.    Update the hidden layer output matrix, getting $\mathbf{H}_{k+1}$ by using new set $\mathcal{L}$;
12.    Update the output weights, getting $\mathbf{B}^{(k+1)}$ by using (6.17.9).
13.  end for
14. until the pre-designed stopping criterion being satisfied.
15. output: $\mathbf{h}(\mathbf{x}) = [G(a_1, b_1, \mathbf{x}), \ldots, G(a_L, b_L, \mathbf{x})]$.
```

6.18 Reinforcement Learning

One of the primary goals of artificial intelligence (AI) is to produce fully autonomous agents that are able to interact with their environments through trial-and-error way in order to learn optimal behaviors. This experience-driven autonomous learning is known as "*reinforcement learning*" (RL) [269].

6.18.1 *Basic Concepts and Theory*

The reinforcement learning machine learns to achieve a goal by trial-and-error interaction within a dynamic environment. At each discrete-time step $t \in \{0, 1, 2, \ldots\}$, the agent observes the state description vector $\mathbf{s}_t$ of the environment, and decides and takes an action $\mathbf{a}_t \in A(\mathbf{s}_t)$ based on the state $\mathbf{s}_t$. After receiving a reward signal from the environment, $r_t \in \mathbb{R}$, the agent decides the action to be performed via a function called a policy $\pi(\mathbf{s}) = \mathbf{a}$, which could also be stochastic $\pi(\mathbf{a}|\mathbf{s}) = \Pr\{\mathbf{a}_t = \mathbf{a}|\mathbf{s}_t = \mathbf{s}\}$. A policy that maximizes the cumulative reward is denoted as π^*. The policy gives probabilities of selecting actions for a certain state. Then the environment state moves to the next state $\mathbf{s}_{t+1}$ and thus the agent obtains a new reward r_{t+1}, where the state $\mathbf{s}_{t+1}$ is drawn from transition probability with respect to the previous state $\mathbf{s}_t$

and the action $\mathbf{a}_t$. The sequence of the steps from an initial state to a terminal state is called as *episode*.

By [137], a standard reinforcement learning model usually consists of

- an agent,
- a discrete set of environment states $\mathbf{s}_t$, $S = \{\mathbf{s}_t\}$,
- a discrete set of agent actions $\mathbf{a}_t$, $A = \{\mathbf{a}_t\}$; and
- a reward function $R : S \times A \to R$.

Definition 6.36 (Agent [276]) An *agent* refers to a hardware or (more usually) software-based computer system that enjoy the following properties:

- *Autonomy:* agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.
- *Social ability:* agents interact with other agents (and possibly humans) via some kind of agent-communication language.
- *Reactivity:* agents operate on their environment (which may be the physical world, a user of these combined), and respond in a timely fashion to changes that occur in it.
- *Pro-activeness:* agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.

Definition 6.37 (State [220]) State refers to all variables (discrete or continuous) that are necessary to model a system. All states of a system are compactly denoted as a vector $\mathbf{s}$.

A state $\mathbf{s}$ contains all relevant information about the current situation to predict future states (or observables) of a system.

Definition 6.38 (State-Action Space [220]) The mathematical space spanned by actions $\mathbf{a}$ and states $\mathbf{s}$ jointly is known as *state-action space*. Solving a movement task can be thought of as finding a path between two points in this state, the initial state and the goal state.

Reinforcement learning is based on the interaction between an agent and its environment. The agent selects the action $\mathbf{a}_t$, resulting in a change in state $\mathbf{s}_t$ and a reward r_t returned by the environment.

Definition 6.39 (Action [220]) An *action* refers to all variables (discrete or continuous) that can actively change the state of a system. Usually, actions are motor commands, abbreviated as a vector $\mathbf{a}$.

Definition 6.40 (Reward Function [220]) A function r_t is called a *reward function*, if it provides a scalar (discrete or continuous) value about the goodness of an action $\mathbf{a}$ in a state $\mathbf{s}$.

There are three different types of commonly used reward functions [147].

- Rewards depending only on the current state $R = R(\mathbf{s})$.
- Rewards depending on the current state and action $R = R(\mathbf{s}, \mathbf{a})$.

- Rewards including the state transitions $R = R(\mathbf{s}', \mathbf{a}, \mathbf{s})$, where $\mathbf{s}'$ is the next state and $\mathbf{s}$ is the previous state.

Reinforcement learning usually requires an unambiguous representation of states and actions of the movement system and the existence of a scalar reward function.

Reinforcement learning proceeds by trying actions in a particular state and updating an evaluation function that assigns expected rewards to possible actions. To deal with possibly delayed rewards, often many iterations are performed. After learning, the action $\mathbf{a}$ with the highest expected value in each state is chosen to achieve the task goal [220].

The goal of reinforcement learning is to find a mapping from states $\mathbf{s}_t$ to actions $\mathbf{a}_t$, called policy $\boldsymbol{\pi}$, that picks actions $\mathbf{a}$ in given states $\mathbf{s}$ maximizing the cumulative expected reward.

Definition 6.41 (Control Policy [220]) A function is known as a *control policy*, if it maps the state $\mathbf{s}$ of a movement system and its environment into an appropriate action $\mathbf{a}$ for a particular task, i.e., $\mathbf{a} = \boldsymbol{\pi}(\mathbf{s}, t, \boldsymbol{\alpha})$. As indicated, the function $\boldsymbol{\pi}$ can directly depend on the time t, and the vector of open parameters, $\boldsymbol{\alpha}$, that may be useful to adjust the policy for a particular task goal.

The policy has the following two types [147].

- *Deterministic policy:* $\boldsymbol{\pi}$ always uses the exact same action for $\mathbf{a}$ given state in the form $\mathbf{a} = \boldsymbol{\pi}(\mathbf{s})$,
- *Probabilistic policy:* $\boldsymbol{\pi}$ draws a sample from a distribution over actions when it encounters a state, i.e., $\mathbf{a} \sim \boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) = P(\mathbf{a}|\mathbf{s})$.

Because an action taken does not necessarily have an immediate impact on the reward, but can also affect a reward in the distant future, the goal of the agency is to minimize its average long-term penalty. That is, the agent needs to deduce a "good" (or ideally optimal) policy (i.e., strategy or controller) $\boldsymbol{\pi}$, i.e., a good mapping between the states and the actions.

The essence of reinforcement learning is to perform learning through interaction with its environment. The reinforcement learning agent starts at receiving a scalar value called a *payoff* for transitions from one state to another. Then the agent determines a control policy which maximizes the *return*. "Return" here refers to the expected future discounted sum of payoffs received. Upon observing the consequences of its actions, a reinforcement learning agent can learn to alter its own behavior in response to *rewards* received.

Reinforcement learning differs from the more widely studied problem of supervised learning in several ways [138]:

- The biggest difference is that there is no presentation of input/output pairs in reinforcement learning. Instead, after choosing an action the agent is given the immediate reward and the subsequent state, but is not told which action would have been in its best long-term interests. It is necessary for the agent to gather useful experience about the possible system states, actions, transitions, and rewards actively to act optimally.

- Reinforcement learning uses the Markov decision process as the model, while the supervised learning uses the linear or nonlinear classification/regression model.
- In supervised learning, expected future predictive accuracy or statistical efficiency are the prime concerns.
- In supervised learning, an agent is directly presented a sequence of independent examples of correct predictions to make in different circumstances. In imitation learning, an agent is provided demonstrations of actions of a good strategy to follow in given situations [147].

By Kaelbling et al. in 1998 [138], reinforcement learning is the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment. Thus it is appropriately thought of as a class of problems, rather than as a set of techniques.

Reinforcement learning with function approximation is a popular framework for approximate policy evaluation and dynamic programming.

Definition 6.42 (Function Approximation [213]) This is a family of mathematical and statistical techniques used to represent a function of interest when it is computationally or information-theoretically intractable to represent the function exactly or explicitly (e.g., in tabular form).

Typically, in reinforcement learning the function approximation is based on sample data collected during interaction with the environment. Function approximation is critical in nearly every reinforcement learning problem, and becomes inevitable in continuous state ones [147].

There are three common *optimal behavior models* [137].

1. *Finite-horizon model:* At a given moment in time, the agent should optimize its expected reward for the next h steps:
$$E\left\{\sum_{t=0}^{h} r_t\right\}, \tag{6.18.1}$$
where r_t represents the scalar reward received t steps into the future. The finite-horizon model is not always appropriate. In many cases we may not know the precise length of the agent's life in advance.
2. *Infinite-horizon discounted model:* It takes the long-run reward of the agent into account, but rewards that are received in the future are geometrically discounted according to discount factor γ (where $0 \leq \gamma < 1$):
$$E\left\{\sum_{t=0}^{\infty} \gamma^{t} r_t\right\}. \tag{6.18.2}$$
Here, γ can be seen as an interest rate, a probability of living another step, or as a mathematical trick to bound the infinite sum. The discounted model is more mathematically tractable than the finite-horizon model. This is a dominant reason for the wide attention this model has received.

3. *Average-reward model:* In this model, the agent is supposed to take actions that optimize its long-run average reward:

$$\lim_{h\to\infty} E\left\{\sum_{t=0}^{\infty} r_t\right\} \tag{6.18.3}$$

which can be seen as the limiting case of the infinite-horizon discounted model (6.18.2) as the discount factor γ approaches 1 [24]. Such a policy is referred to as a *gain optimal policy*.

6.18.2 Markov Decision Process (MDP)

A *Markov decision process* (MDP) is discrete-time stochastic control process and is used to model the synchronous interaction between agent and environment [138], as shown in Fig. 6.3.

The Markov decision process setting corresponds to the standard reinforcement learning model: the environment at time step t can be represented by a finite discrete set of state description vectors, $S = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$, with a finite discrete set of actions in each state performed by the agent, $A = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$. Associated with each action in each state is a *state transition function* $P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) : S \times A \times S \to \mathbb{R}$ which determine the transition probability distribution of ending in state $\mathbf{s}_{t+1}$ given environment state $\mathbf{s}_t$ and agent action $\mathbf{a}_t$. Therefore, in the standard reinforcement learning framework [138, 239], a Markov decision process can be described as a five-tuple (S, A, P, R, γ).

1. $S = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$ is a finite set of states of the environment, called the *state space*.
2. $A = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$ is a finite set of actions that can be performed by the agent, called the *action space*.
3. The state transition function $P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) : S \times A \times S \to \mathbb{R}$ gives for environment state $\mathbf{s}_t$ and agent action $\mathbf{a}_t$, a probability distribution of ending in state $\mathbf{s}_{t+1}$.

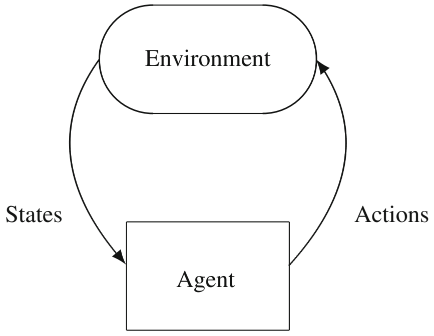

Fig. 6.3 An MDP models the synchronous interaction between agent and environment

4. $R(\mathbf{s}_t, \mathbf{a}_t) : S \times A \to \mathbb{R}$ is a *reward function*, and denotes an immediate/instantaneous reward function generating an *immediate reward* $r_{t+1} = R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ obtained by doing action $\mathbf{a}_t$ at state transition from state $\mathbf{s} \in S$ to $\mathbf{s}_{t+1} \in S$, and taking the expected rewards $R_s^a = E\{r_{t+1}|\mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}\}$.
5. $\gamma \in [0, 1]$ is a predefined *discount factor* that represents the difference in importance between future rewards and present rewards, in which lower values place more emphasis on immediate rewards $r_t = R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$.

In other words, an agent is connected to its environment through perception and action as follows.

- On each step of interaction the agent receives as input i some indication of the current state $\mathbf{s}_t$ of the environment.
- The agent chooses an action $\mathbf{a}_t$ to generate an output.
- The action changes the state of the environment, and the value of this state transition is communicated to the agent through a scalar "reinforcement signal" r_t.
- The agent's behavior B chooses actions that tend to increase the long-run sum of values of the reinforcement signal.

A common way to solve the reinforcement learning problem is to describe the environment as a Markov decision problem with a set of state value function pairs, a set of action-values, a policy, and a reward function.

When action $\mathbf{a}_t$ is triggered in state $\mathbf{s}_t$, the system emits a reinforcement signal $R(\mathbf{s}_t, \mathbf{a}_t)$. The action choice in each state, i.e., the learner's behavior B is determined by its policy. The policy is a mapping from states to actions, and is denoted by $\boldsymbol{\pi} = [\pi(\mathbf{s}_1), \ldots, \pi(\mathbf{s}_{|S|})]$, where $|S|$ is the cardinal of state set S. Following policy $\boldsymbol{\pi}$, whenever the learner is in state $\mathbf{s}_t$, it applies action $\mathbf{a}_t = \boldsymbol{\pi}(\mathbf{s}_t)$ (in the stationary policies). Each policy has an evaluation function

$$V^{\pi}(\mathbf{s}) = E_{\pi}\left\{\sum_{t=0}^{\infty} \gamma^{\,t} R(\mathbf{s}_t, \boldsymbol{\pi}(\mathbf{s}_t))|\mathbf{s}_0 = \mathbf{s}\right\} \tag{6.18.4}$$

qualifying the corresponding learner's behavior with respect to the reinforcement function. Here, E_π denotes the expected value under the assumptions that policy $\boldsymbol{\pi}$ is used and $\mathbf{s}$ is the starting state. The discount factor $\gamma \in [0, 1]$ is used to weight primary reinforcements with respect to time. $\gamma = 0$ means that evaluation functions represent primary reinforcements. Otherwise, they denote the expected discounted return over an infinite number of time steps.

Definition 6.43 (Improvement [136]) Let $\boldsymbol{\pi}$ and $\boldsymbol{\pi}'$ be two polices; $\boldsymbol{\pi}'$ is said to be an *improvement* over $\boldsymbol{\pi}$ if $V^{\pi'}(\mathbf{s}) \geq V^{\pi}(\mathbf{s}), \forall\, \mathbf{s} \in S$ with strict inequality holding for at least one state.

The general framework of reinforcement learning encompasses a broad variety of problems ranging from various forms of function optimization at one extreme to learning control at the other [269]. In the following, we discuss reinforcement learning from the view of point of learning control [14].

Reinforcement learning is considered as one of the most crucial and effective online control strategies. At the beginning, selection is based on the trial-and-error model. However, after a certain time horizon is passed, the algorithm is trained to apply suitable control actions. For each state in this method, a different control action is applied. For instance, if $\mathbf{s}_t$ and $\mathbf{a}_t$ denote the system state and system control at time t, then the state at time $t + 1$ is given by [21]

$$\mathbf{s}_{t+1} = \mathbf{f}(\mathbf{s}_t, \mathbf{a}_t), \quad \mathbf{a}_t \in A, \forall t > 0. \tag{6.18.5}$$

For each control action $\mathbf{a}_t$, a reward or a punishment is presumed. In other words, if this action improves the system error, then a reward will be anticipated for this action; otherwise a punishment will be applied. The selection of reward or punishment is specified by comparison between errors in t and $t + 1$. Thus, the immediate reward function at time t for each action $\mathbf{a}_t$ is given by

$$R(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t=0}^{\infty} \gamma^{\,t} r(\mathbf{s}_t, \mathbf{a}_t), \tag{6.18.6}$$

where $0 < \gamma < 1$ and $r(\mathbf{s}_t, \mathbf{a}_t) \leq B$ in which B is the maximum value of the reward and is selected through trial-and-error.

The *state value function* is defined as the maximum value of the immediate reward functions:

$$V(\mathbf{s}) = \max_{\mathbf{a}_t \in A} R(\mathbf{s}_t, \mathbf{a}_t). \tag{6.18.7}$$

The state value function is described by the *Bellman equation* [21]

$$V(\mathbf{s}) = \max_{\mathbf{a} \in A} \{r(\mathbf{s}, \mathbf{a}) + \gamma\, V(\mathbf{f}(\mathbf{s}, \mathbf{a}))\}. \tag{6.18.8}$$

A scalar value called a *payoff* is received by the control system for transitions from one state to another. The aim of the system is to find a control policy which maximizes the expected future discount of the payoff received, known as the "*return*" [218].

The state value function $V(\mathbf{s}_t)$ represents the discounted sum of payoff received from time step t onwards, and this sum will depend on the sequence of actions taken. The taken sequence of actions can be determined by the *policy*.

Hence, the *optimal control policy* can be expressed as

$$\mathbf{a}^*(\mathbf{s}) = \arg \max_{\mathbf{a} \in A} \{r(\mathbf{s}, \mathbf{a}) + \gamma\, V(\mathbf{f}(\mathbf{s}, \mathbf{a}))\}, \tag{6.18.9}$$

from which a reward or punishment function, called *action-value function* or simply *Q-function*, can be obtained as

$$Q(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma\, V(\mathbf{f}(\mathbf{s}, \mathbf{a})). \tag{6.18.10}$$

In terms of Q-function, the state value function $V(\mathbf{s})$ can be reexpressed as

$$V(\mathbf{s}) = \max_{\mathbf{a}\in A} Q(\mathbf{s}, \mathbf{a}), \tag{6.18.11}$$

and thus the optimal control policy is given by

$$\mathbf{a}^*(\mathbf{s}) = \arg\max_{\mathbf{a}\in A} Q(\mathbf{s}, \mathbf{a}). \tag{6.18.12}$$

The state value function $V(\mathbf{s})$ or action-value function $Q(\mathbf{s}, \mathbf{a})$ is updated by a certain policy $\boldsymbol{\pi}$, where $\boldsymbol{\pi}$ is a function that maps states $\mathbf{s} \in S$ to actions $\mathbf{a} \in A$, i.e., $\boldsymbol{\pi} : S \to A \Rightarrow \mathbf{a} = \boldsymbol{\pi}(\mathbf{s})$.

6.19 Q-Learning

Q-learning, proposed by Watkins in 1989 [263], is a popular model-free reinforcement learning algorithm, and can be used to optimally solve Markov decision processes (MDPs). Moreover, Q-learning is a technology based on action-value function to evaluate which action to take, and action-value function determines the value of taking an action in a certain state.

6.19.1 Basic Q-Learning

As an important model-free asynchronous dynamic programming (DP) technique, Q-learning provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains.

As the name implies, *Q-learning* is how to learn a value (i.e., Q) function. There are two methods for learning the *Q-function* [248].

- *On-policy methods* are namely Sarsa for on-policy control [218]:

$$\Delta_{\text{Sarsa}} \leftarrow \big[r_{t+1} + \gamma Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - Q(\mathbf{s}_t, \mathbf{a}_t)\big], \tag{6.19.1}$$

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q(\mathbf{s}_t, \mathbf{a}_t) + \alpha \Delta_{\text{Sarsa}}. \tag{6.19.2}$$

- *Off-policy methods* are Q-learning for off-policy control [263]:

$$\mathbf{b}^* \leftarrow \arg\max\nolimits_{\mathbf{b}\in A(\mathbf{s}_{t+1})} Q(\mathbf{s}_{t+1}, \mathbf{b}), \tag{6.19.3}$$

$$\Delta_{\text{Qlearning}} \leftarrow \big[r_{t+1} + \gamma Q(\mathbf{s}_{t+1}, \mathbf{b}^*) - Q(\mathbf{s}_t, \mathbf{a}_t)\big], \tag{6.19.4}$$

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q(\mathbf{s}_t, \mathbf{a}_t) + \alpha \Delta_{\text{Qlearning}}, \tag{6.19.5}$$

where α is a stepsize parameter [106].

The basic framework of Q-learning is as follows. Under the assumption that the number of system states and the number of agent actions must be finite, the learning system consists of a finite set of environment states $S = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$ and a finite set of agent actions $A = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$. At each step of interaction with the environment, the agent observes the environment state $\mathbf{s}_t$ and issues an action $\mathbf{a}_t$ based on the system state. By performing the action $\mathbf{a}_t$, the system moves from one state $\mathbf{s}_t$ to another $\mathbf{s}_{t+1}$. The new state $\mathbf{s}_{t+1}$ gives the agent a penalty $p_t = p(\mathbf{s}_t)$ which indicates the value of the state transition $\mathbf{s}_t \to \mathbf{s}_{t+1}$. The agent keeps a value function (named Q-function) $Q^\pi(\mathbf{s}, \mathbf{a})$ for each state-action pair $(\mathbf{s}, \mathbf{a})$, which represents the expected long-term penalty or reward if the system starts from state $\mathbf{s}$, taking action $\mathbf{a}$, and then following policy π. Based on this Q-function, the agent decides which action should be taken in current state to achieve the minimum long-term penalties or the maximum rewards.

Therefore, the core of the Q-learning algorithm is a value iteration update of the Q-value function. The Q-value for each state-action pair is initially chosen by the designer and then it will be updated each time an action is issued and a penalty is received based on the following expression [229]:

$$Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) = \underbrace{Q(\mathbf{s}_t, \mathbf{a}_t)}_{\text{old value}} + \underbrace{\eta_t(\mathbf{s}_t, \mathbf{a}_t)}_{\text{learning rate}} \times \left(\overbrace{\underbrace{p_{t+1}}_{\text{pernalty}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\min_{\mathbf{a}} Q(\mathbf{s}_{t+1}, \mathbf{a})}_{\text{min future value}}}^{\text{expected discount pernalty}} - \overbrace{Q(\mathbf{s}_t, \mathbf{a}_t)}^{\text{old value}} \right). \tag{6.19.6}$$

Here, $\mathbf{s}_t$, $\mathbf{a}_t$, and p_t are the state visited, action taken, and penalty received at time t, respectively; and $\eta_t(\mathbf{s}, \mathbf{a}) \in (0, 1)$ is the learning rate. The discount factor γ is a value between 0 and 1 which gives more weight to the penalties in the near future than the far future.

Interestingly, if we let $Q(\mathbf{s}_t, \mathbf{a}_t) = x_i$, $\eta_t(\mathbf{s}_t, \mathbf{a}_t) = \alpha$, $\gamma \min_{\mathbf{a}} Q(\mathbf{s}_{t+1}, \mathbf{a}) = F_i(\mathbf{x})$ and $p_{t+1} = n_i$, then Q-learning (6.19.6) has the same general structure as stochastic approximation algorithms [252]:

$$x_i = x_i + \alpha\big(F_i(\mathbf{x}) - x_i + n_i\big), \quad i = 1, \ldots, n, \tag{6.19.7}$$

where $\mathbf{x} = [x_1, \ldots, x_n]^T \in \mathbb{R}^n$, $F_1, \ldots, F_n$ are mappings from $\mathbb{R}^n$ into $\mathbb{R}$, i.e., $F_i : \mathbb{R}^n \to \mathbb{R}$, n_i is a random noise term and α is a small, usually decreasing, stepsize.

On the mapping vector function $\mathbf{f}(\mathbf{x}) = [F_1(\mathbf{x}), \ldots, F_n(\mathbf{x})]^T \in \mathbb{R}^n$, one makes the following assumptions [252].

(a) The mapping $\mathbf{f}$ is monotone; that is, if $\mathbf{x} \leq \mathbf{y}$, then $\mathbf{f}(\mathbf{x}) \leq \mathbf{f}(\mathbf{y})$.
(b) The mapping $\mathbf{f}$ is continuous.
(c) The mapping $\mathbf{f}(\mathbf{x})$ has a unique fixed point $\mathbf{x}^*$.

(d) If $\mathbf{1} \in \mathbb{R}^n$ is the summing vector with all components equal to 1, and r is a positive scalar, then

$$\mathbf{f}(\mathbf{x}) - r\mathbf{1} \leq \mathbf{f}(\mathbf{x} - r\mathbf{1}) \leq \mathbf{f}(\mathbf{x} + r\mathbf{1}) \leq \mathbf{f}(\mathbf{x}) + r\mathbf{1}. \tag{6.19.8}$$

In basic Q-learning, the agent's experience consists of a sequence of distinct stages or *episodes*. In the tth episode, the agent [264]:

- observes its current state $\mathbf{s}_t$,
- selects and performs an action $\mathbf{a}_t$,
- observes the subsequent state $\mathbf{s}_{t+1}$,
- receives an immediate payoff r_t, and
- adjust its Q-function value using a learning factor $\eta_t(\mathbf{s}_t, \mathbf{a}_t)$, according to Eq. (6.19.6).

Algorithm 6.38 shows a basic Q-learning algorithm.

Algorithm 6.38 Q-learning [296]

```
1. initialization: $Q, \mathbf{s}$
2. Repeat (for each step of episode)
3.    Choose action $\mathbf{a}$ from state $\mathbf{s}$ based on $Q$ and some exploration strategy (e.g., $\epsilon$-greedy)
4.    Take action $\mathbf{a}$, observe $r, \mathbf{s}'$
5.    $\mathbf{a}^* \leftarrow \arg\max_{\mathbf{a}} Q(\mathbf{s}', \mathbf{a})$
6.    $\delta \leftarrow r + \gamma Q(\mathbf{s}', \mathbf{a}^*) - Q(\mathbf{s}, \mathbf{a})$
7.    $Q(\mathbf{s}, \mathbf{a}) \leftarrow Q(\mathbf{s}, \mathbf{a}) + \alpha(\mathbf{s}, \mathbf{a})\delta$
8.    $\mathbf{s} \leftarrow \mathbf{s}'$
9. until end
10. output: s.
```

Q-learning has four important variants: double Q-learning, weighted double Q-learning, online connectionist Q-learning, and Q-learning with experience replay. They are the topics in the next three subsections.

6.19.2 Double Q-Learning and Weighted Double Q-Learning

Although Q-learning can be used to optimally solve Markov decision processes (MDPs), its performance may be poor in stochastic MDPs because of large overestimation of the action-values [255].

To cope with overestimation caused by Q-learning, *double Q-learning* that was developed in [255] uses the *double estimator* method due to using two sets of estimators: $\mu^A = \{\boldsymbol{\mu}_1^A, \ldots, \boldsymbol{\mu}_M^A\}$ and $\mu^B = \{\boldsymbol{\mu}_1^B, \ldots, \boldsymbol{\mu}_M^B\}$ to estimate the maximum expected value of the Q-function for the next state, $\max_{\mathbf{a}'} E\{Q(\mathbf{s}', \mathbf{a}')\}$. Both sets of estimators are updated with a subset of the samples drawn such that $S = S^A \cup S^B$ and $S_A \cap S_B = \emptyset$. Like the single estimator $\boldsymbol{\mu}_i$, both $\boldsymbol{\mu}_i^A$ and $\boldsymbol{\mu}_i^B$ are unbiased if one assumes that the samples are split in a proper manner, for instance, randomly, over the two sets of estimators.

Suppose we are given a set of M random variables $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$. In many problems, one is interested in finding the maximum expected value $\max_i E\{\mathbf{x}_i\}$. However, it is impossible to determine $\max_i E\{\mathbf{x}_i\}$ exactly if there is no knowledge of the functional form and parameters of the underlying distributions of the variables in X. Therefore, this maximum expected value $\max_i\{\mathbf{x}_i\}$ is usually approximated by constructing approximations for $E\{\mathbf{x}_i\}$ for all i.

Let $S = \bigcup_{i=1}^{M} S_i$ be a set of samples, where S_i is the subset containing samples for the variable $\mathbf{x}_i$. It is assumed that the samples in S_i are independent and identically distributed (i.i.d.). Then, unbiased estimates for the expected values can be obtained by computing the sample average for each variable: $E\{\mathbf{x}_i\} = E\{\boldsymbol{\mu}_i\} \approx \boldsymbol{\mu}_i(S) \overset{\text{def}}{=} \frac{1}{|S_i|}\sum_{\mathbf{s}\in S_i}\mathbf{s}$, where $\boldsymbol{\mu}_i$ is a mean estimator for variables $\mathbf{x}_i$. This approximation is unbiased since every sample $\mathbf{s} \in S_i$ is an unbiased estimate for the value of $E\{\mathbf{x}_i\}$.

As shown in Algorithm 6.39, the double Q-learning stores two Q-functions, Q^A and Q^B, and uses two separate subsets of experience samples to learn them. The action selected is calculated based on the average of the two Q-values for each action and the ϵ-greedy exploration strategy. With the same probability, each Q-function is updated using a value from the other Q-function for the next state. Then, the action $\mathbf{a}^*$ is the maximizing action in state $\mathbf{s}'$ based on the Q_1 value.

Algorithm 6.39 Double Q-learning [255]

```
1.  input: $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$
2.  initialize $Q^A$, $Q^B$, $\mathbf{s}$
3.  repeat
4.     Choose $\mathbf{a}$ from $\mathbf{s}$ based on $Q^A(\mathbf{s},\cdot)$ and $Q^B(\mathbf{s},\cdot)$ (e.g., Q-greedy in $Q^A + Q^B$)
5.     Take action $\mathbf{a}$, observe $r$, $\mathbf{s}'$
6.     With 0.5 probability to Choose whether to update $Q^A$ or $Q^B$
7.     if update $Q^A$ then
8.        $\mathbf{a}^* \leftarrow \arg\max_{\mathbf{a}} Q^A(\mathbf{s}',\mathbf{a})$
9.        $\alpha^A(\mathbf{s},\mathbf{a}) \leftarrow r + \gamma Q^B(\mathbf{s}',\mathbf{a}^*) - Q^A(\mathbf{s},\mathbf{a})$
10.       $\delta^A \leftarrow r + \gamma Q^B(\mathbf{s}',\mathbf{a}^*) - Q^A(\mathbf{s},\mathbf{a})$
11.       $Q^A(\mathbf{s},\mathbf{a}) \leftarrow Q^A(\mathbf{s},\mathbf{a}) + \alpha^A(\mathbf{s},\mathbf{a})\delta^A$
12.       else if update $Q^B$ then
13.       $\mathbf{a}^* \leftarrow \arg\max_{\mathbf{a}} Q^B(\mathbf{s}',\mathbf{a})$
14.       $\alpha^B(\mathbf{s},\mathbf{a}) \leftarrow r + \gamma Q^A(\mathbf{s}',\mathbf{a}^*) - Q^B(\mathbf{s},\mathbf{a})$
15.       $\delta^B \leftarrow r + \gamma Q^A(\mathbf{s}',\mathbf{a}^*) - Q^B(\mathbf{s},\mathbf{a})$
16.       $Q^B(\mathbf{s},\mathbf{a}) \leftarrow Q^B(\mathbf{s},\mathbf{a}) + \alpha^B(\mathbf{s},\mathbf{a})\delta^B$
17.       $\mathbf{s} \leftarrow \mathbf{s}'$
18.    end if
19. until end
20. output: $\hat{\mathbf{x}}_i \approx \mu_i(S) = \frac{1}{|S_i|}\sum_{\mathbf{s}\in S_i}\mathbf{s}$.
```

As opposed to Q-learning, the double Q-learning uses the value $Q^B(\mathbf{s}',\mathbf{a}^*)$ rather than the value $Q^A(\mathbf{s}',\mathbf{a}^*)$ to update Q^A.

Since Q^B is updated with a different set of experience samples, $Q^B(\mathbf{s}', \mathbf{a}^*)$ is an unbiased estimate for the value of the action $\mathbf{a}^*$ in the sense that $E\{Q^B(\mathbf{s}', \mathbf{a}^*)\} = E\{Q(\mathbf{s}', \mathbf{a}^*)\}$. Similarly, the value of $Q^A(\mathbf{s}', \mathbf{a}^*)$ is also unbiased. However, since $E\{Q^B(\mathbf{s}', \mathbf{a}^*)\} \leq \max_{\mathbf{a}} E\{Q^B(\mathbf{s}', \mathbf{a})\}$ and $E\{Q^A(\mathbf{s}', \mathbf{a}^*)\} \leq \max_{\mathbf{a}} E\{Q^A(\mathbf{s}', \mathbf{a})\}$, both estimators $Q^B(\mathbf{s}', \mathbf{a}^*)$ and $Q^A(\mathbf{s}', \mathbf{a}^*)$ sometime have negative biases. This results in double Q-learning underestimating action values in some stochastic environments.

Weighted double Q-learning [296] combines Q-learning and double Q-learning for avoiding the overestimation in Q-learning and the underestimation in double Q-learning, as shown in Algorithm 6.40.

Algorithm 6.40 Weighted double Q-learning [296]

```
1. **input:** $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$
2. **initialize** $Q^A, Q^B, \mathbf{s}$
3. **repeat**
4.    Choose $\mathbf{a}$ from $\mathbf{s}$ based on $Q^A$ and $Q^B$ (e.g., $\epsilon$-greedy in $Q^A + Q^B$)
5.    Take action $\mathbf{a}$, observe $r, \mathbf{s}'$
6.    With 0.5 probability to Choose whether to update $Q^A$ or $Q^B$
7.    **if** chose to update $Q^A$ **then**
8.       $\mathbf{a}^* \leftarrow \arg\max_{\mathbf{a}} Q^A(\mathbf{s}', \mathbf{a})$
9.       $\mathbf{a}_L \leftarrow \arg\min_{\mathbf{a}} Q^A(\mathbf{s}', \mathbf{a})$
10.      $\beta^A \leftarrow \frac{|Q^B(\mathbf{s}',\mathbf{a}^*)-Q^B(\mathbf{s}',\mathbf{a}_L)|}{c+|Q^B(\mathbf{s}',\mathbf{a}^*)-Q^B(\mathbf{s}',\mathbf{a}_L)|}$
11.      $\delta^A \leftarrow r + \gamma[\beta^A Q^A(\mathbf{s}', \mathbf{a}^*) + (1-\beta^A)Q_B(\mathbf{s}', \mathbf{a}^*)] - Q^A(\mathbf{s}, \mathbf{a})$
12.      $\alpha^A(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q^B(\mathbf{s}', \mathbf{a}^*) - Q^A(\mathbf{s}, \mathbf{a})$
13.      $Q^A(\mathbf{s}, \mathbf{a}) \leftarrow Q^A(\mathbf{s}, \mathbf{a}) + \alpha^A(\mathbf{s}, \mathbf{a})\delta^A$
14.      **else if** chose to update $Q^B$ **then**
15.      $\mathbf{a}^* \leftarrow \arg\max_{\mathbf{a}} Q^B(\mathbf{s}', \mathbf{a})$
16.      $a_L \leftarrow \arg\min_{\mathbf{a}} Q^B(\mathbf{s}', \mathbf{a})$
17.      $\beta^B \leftarrow \frac{|Q^A(\mathbf{s}',\mathbf{a}^*)-Q^A(\mathbf{s}',\mathbf{a}_L)|}{c+|Q^A(\mathbf{s}',\mathbf{a}^*)-Q^A(\mathbf{s}',\mathbf{a}_L)|}$
18.      $\delta^B \leftarrow r + \gamma[\beta^B Q^B(\mathbf{s}', \mathbf{a}^*) + (1-\beta^B)Q^A(\mathbf{s}', \mathbf{a}^*)] - Q^B(\mathbf{s}, \mathbf{a})$
19.      $\alpha^B(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q^A(\mathbf{s}', \mathbf{a}^*) - Q^B(\mathbf{s}, \mathbf{a})$
20.      $Q^B(\mathbf{s}, \mathbf{a}) \leftarrow Q^B(\mathbf{s}, \mathbf{a}) + \alpha^B(\mathbf{s}, \mathbf{a})\delta^B$
21.      $\mathbf{s} \leftarrow \mathbf{s}'$
22.   **end if**
23. **until end**
24. **output:** $\hat{\mathbf{x}}_i \approx \mu_i(S) = \frac{1}{|S_i|}\sum_{\mathbf{s}\in S_i} \mathbf{s}$.
```

The following are the comparison among the Q-learning, the double Q-learning, and the weighted double Q-learning.

1. The Q-learning is a single estimator for $Q(\mathbf{s}, \mathbf{a})$.
2. The double Q-learning is a double estimator for both $Q^A(\mathbf{s}, \mathbf{a})$ and $Q^B(\mathbf{s}, \mathbf{a})$.
3. The weighted double Q-learning is a weighted double estimator for both $Q^A(\mathbf{s}, \mathbf{a})$ and $Q^B(\mathbf{s}, \mathbf{a})$. The main differences between the double Q-learning

Algorithm 6.39 and the weighted double Q-learning Algorithm 6.40 are:

- Algorithm 6.40 uses the weighted term $\beta^A Q^A(\mathbf{s}', \mathbf{a}^*) + (1 - \beta^A) Q^B(\mathbf{s}', \mathbf{a}^*)$ to replace $Q^B(\mathbf{s}', \mathbf{a}^*)$ when updating δ^A in Algorithm 6.39;
- Algorithm 6.40 uses the weighted term $\beta^B Q^B(\mathbf{s}', \mathbf{a}^*) + (1 - \beta^B) Q^A(\mathbf{s}', \mathbf{a}^*)$ to replace $Q^A(\mathbf{s}', \mathbf{a}^*)$ when updating δ^B in Algorithm 6.39.

These changes result in the name "weighted double Q-learning."

6.19.3 Online Connectionist Q-Learning Algorithm

In order to represent the Q-function, Lin [163] chose to use one neural network for each action, which avoids the hidden nodes receiving conflicting error signals from different outputs.

The neural network weights are updated by [163]

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta \left(r_t + \gamma \max_{\mathbf{a} \in A} Q_{t+1} - Q_t \right) \nabla_w Q_t, \tag{6.19.9}$$

where η is the learning constant, r_t is the payoff received for the transition from state $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$, $Q_t = Q(\mathbf{x}_t, \mathbf{a}_t)$, and $\nabla_w Q_t$ is a vector of the output gradients $\partial Q_t / \partial \mathbf{w}_t$ calculated by backpropagation.

Rummery and Niranjan [218] proposed an alternative update algorithm, called *modified connectionist Q-learning*. This algorithm is based more strongly on temporal difference, and uses the following update rule:

$$\Delta \mathbf{w}_t = \eta \left(r_t + \gamma Q_{t+1} - Q_t \right) \sum_{k=0}^{t} (\gamma \lambda)^{t-k} \nabla_w Q_k. \tag{6.19.10}$$

This update rule is different from the normal Q-learning in the use of the Q_{t+1} associated with the action selected instead of the greedy $\max_{\mathbf{a} \in A} Q_{t+1}$.

In the following, we discuss how to calculate the gradient $\nabla_w Q_t$. For this end, define a backpropagation neural network as a collection of interconnected units arranged in layers. Let i, j, and k refer to the different neurons in the network; with error signal propagating from right to left, as shown in Fig. 6.4.

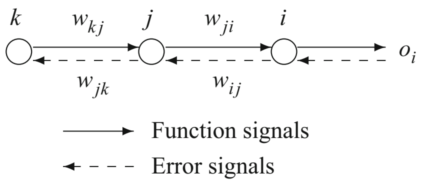

Fig. 6.4 One neural network as a collection of interconnected units, where i, j, and k are neurons in output layer, hidden layer, and input layer, respectively

.

Neuron j lies in a layer to the left of neuron i, and neuron k lies in a layer to the left of the neuron j when neuron j is a hidden unit.

Letting w_{ij} be a weight on a connection from layer i to j, then the output from layer i is given by [218]:

$$o_i = f(\sigma_i), \tag{6.19.11}$$

where $\sigma_i = \sum_j w_{ij} o_j$ and $f(\sigma)$ is a sigmoid function.

The output gradient with respect to the output layer weights w_{ij} is defined as

$$\frac{\partial o_i}{\partial w_{ij}} = \frac{\partial f(\sigma_i)}{\partial w_{ij}} = \frac{\partial f(\sigma_i)}{\partial \sigma_i} \cdot \frac{\partial \sigma_i}{\partial w_{ij}} = f'(\sigma_i) o_j, \tag{6.19.12}$$

where $f'(\sigma_i) = \frac{\partial f(\sigma_i)}{\partial \sigma_i}$ is the first-order differential of the sigmoid function $f(\sigma_i)$.

The gradient of the output o_i with respect to the hidden layer weights w_{jk} is given by

$$\begin{aligned}\frac{\partial o_i}{\partial w_{jk}} = \frac{\partial f(\sigma_i)}{\partial w_{jk}} &= \frac{\partial f(\sigma_i)}{\partial \sigma_i} \cdot \frac{\partial \sigma_i}{\partial o_j} \cdot \frac{\partial o_j}{\partial \sigma_j} \cdot \frac{\partial \sigma_j}{\partial w_{jk}} \\ &= f'(\sigma_i) \cdot w_{ij} \cdot f'(\sigma_j) \cdot o_k. \end{aligned} \tag{6.19.13}$$

Modified connectionist Q-learning of Rummery and Niranjan [218] is shown in Algorithm 6.41.

Algorithm 6.41 Modified connectionist Q-learning algorithm [218]

```
1. **initialization:** Reset all eligibilities $\mathbf{e}_0 = \mathbf{0}$ and put $t = 0$.
2. **repeat**
3.    Select action $\mathbf{a}_t$;
4.    If $t > 0$,
5.       $\mathbf{w}_t = \mathbf{w}_{t-1} + \eta(r_{t-1} + Q_t - Q_{t-1})\mathbf{e}_{t-1}$.
6.    Calculate $\nabla_w Q_t$ with respect to selected action $\mathbf{a}_t$ only.
7.    $\mathbf{e}_t = \nabla_w Q_t + \gamma\lambda\mathbf{e}_{t-1}$.
8.    Perform action $\mathbf{a}_t$, and receive the payoff $r_t$.
9.    If trial has not ended, $t \leftarrow t + 1$ and go to Step 3.
10. **until end**
11. **output:** $Q_t$.
```

6.19.4 Q-Learning with Experience Replay

Reinforcement learning is known to be unstable and can even diverge when a nonlinear function approximator such as a neural network is used to represent the action-value, i.e., Q function. This instability has a few causes [183]:

- The correlations present in the sequence of observations.
- Small updates to Q may significantly change the policy and therefore change the data distribution.
- The correlation between the action-value Q and the target value $r + \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}')$.

To deal with these instabilities, a novel variant of Q-learning was developed by Mnih et al. [183], which uses two key ideas.

1. A biologically inspired mechanism termed experience replay is used, which randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution.
2. An iterative update is used for adjusting the action-values Q towards target values that are only periodically updated, thereby reducing correlations with the target.

To generate the experience replay, an approximate value function $Q(\mathbf{s}, \mathbf{a}; \boldsymbol{\theta}_i)$ is first parameterized, where $\boldsymbol{\theta}_i$ are the parameters (that is, weights) of the Q-network at iteration i. Then, the agent's experiences $e_t = (\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ are stored at each time-step t in a data set $D_t = \{e_1, \ldots, e_t\}$ to perform experience replay.

Algorithm 6.42 shows the deep Q-learning with experience replay developed by Mnih et al. [183].

Algorithm 6.42 Deep Q-learning with experience replay [183]

```
1.  initialization:
    1.1 replay memory $D$ to capacity $N$,
    1.2 action-value function $Q$ with random weights $\boldsymbol{\theta}$,
    1.3 target action-value function $\hat{Q}$ with weights $\boldsymbol{\theta}^- = \boldsymbol{\theta}$.
2.  for episode = 1, $M$ do
3.    Initialize sequence $s_1 = \{\mathbf{x}_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$.
4.    for $t = 1, T$ do
5.      With probability $\epsilon$ select a random action $\mathbf{a}_t$,
6.      otherwise select $\mathbf{a}_t = \arg\max_{\mathbf{a}} Q(\phi(s_t), \mathbf{a}; \boldsymbol{\theta})$.
7.      Execute action $\mathbf{a}_t$ in emulator and observe reward $r_t$ and image $\mathbf{x}_{t+1}$.
8.      Set $s_{t+1} = s_t, \mathbf{a}_t, \mathbf{x}_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$.
9.      Store transition $(\phi_t, \mathbf{a}_t, r_t, \phi_{t+1})$ in $D$.
10.     Sample random mini batch of transitions $(\phi_j, \mathbf{a}_j, r_j, \phi_{j+1})$ from $D$.
11.     Set $y_j = \begin{cases} r_j, & \text{if episode terminates at step } j+1; \\ r_j + \max_{\mathbf{a}'} \hat{Q}(\phi_{j+1}, \mathbf{a}'; \theta^-), & \text{otherwise.} \end{cases}$
12.     Perform a gradient descent step on $\big(y_j - Q(\phi_j, \mathbf{a}_j; \theta)\big)^2$ with respect to the network
        parameters $\theta$.
13.     Every $C$ steps reset $\hat{Q} = Q$.
14.   end for
15. end for
```

During learning, Q-learning updates are applied on samples (or mini-batches) of experience $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}') \sim U(D)$, drawn uniformly at random from the pool of stored samples. The Q-learning update at iteration i uses the following loss function:

$$L_i(\theta_i) = E_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim U(D)}\left\{ r + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}'; \theta_i^-) - Q(\mathbf{s}, \mathbf{a}; \theta_i) \right\}, \tag{6.19.14}$$

where γ is the discount factor determining the agent's horizon, θ_i are the parameters of the Q-network at iteration i and θ_i^- are the network parameters used to compute the target at iteration i. The target network parameters θ_i^- are only updated with the Q-network parameters θ_i every C steps and are held fixed between individual updates.

6.20 Transfer Learning

Traditional machine learning/data mining works well only when both training set and test set come from the same feature space and have the same distribution. This implies that whenever the data is changed, the model needs to be retrained, which may be too troublesome because

- it is very expensive and difficult to obtain new training data for new data sets,
- some data sets are easily outdated, that is, the data distribution in different periods will be different.

Therefore, in many real-world applications, when the distribution changes or data are outdated, most statistical models are no longer applicable, and need to be rebuilt using newly collected training data. It is expensive or impossible to recollect the needed training data and rebuild the models. In such cases, knowledge transfer or transfer learning between tasks and domains would be desirable.

Unlike traditional machine learning, transfer learning allows the domains, tasks, and distributions used in training and testing to be different.

On the other hand, we get easily a large number of unlabeled images (or audio samples, or text documents) randomly downloaded from the Internet, and it is desirable to use such unlabeled images for improving performance on a given image (or audio, or text) classification task. Clearly, in such cases it is not assumed that the unlabeled data follows the same class labels or generative distribution as the labeled data. Using unlabeled data in supervised classification tasks is known as self-taught learning (transfer learning from unlabeled data) [210]. Therefore, transfer learning or self-taught learning is widely applicable for typical semi-supervised or transfer learning settings in many practical learning problems.

Transfer learning can transfer previously knowledge to a new task, helping the learning of a new task.

Transfer learning has attracted more and more attention since 1995 in different names: learning to learn, life-long learning, knowledge transfer, inductive learning, multitask learning, knowledge consolidation, context-sensitive learning, knowledge-based inductive bias, metalearning, incremental/cumulative learning, and self-taught learning [199, 210, 243].

6.20.1 Notations and Definitions

Definition 6.44 (Domain [199]) A *domain* $\mathcal{D}$ consists of two parts: a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, denoted as $\mathcal{D} = \{\mathcal{X}, P(X)\}$, where $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathcal{X}$ is a particular learning sample (set).

For document classification with a bag-of-words representation, $\mathcal{X}$ is the space of all document representations, $\mathbf{x}_i$ is the ith term vector corresponding to some document and X is the sample set of documents used for training.

In supervised learning, unsupervised learning, semi-supervised learning, inductive learning and semi-supervised transductive learning discussed before, the source domain and the target domain are assumed to have the same feature space or the same marginal probability distribution. In these cases, we denote the labeled set $D^{(l)}$, the unlabeled set $D^{(u)}$ and the test set $D^{(t)}$ as

$$D^{(l)} = \left\{(\mathbf{x}_{l_1}, y_{l_1}), \ldots, (\mathbf{x}_{l_{n_l}}, y_{l_{n_l}})\right\}, \tag{6.20.1}$$

$$D^{(u)} = \{\mathbf{x}_{u_1}, \ldots, \mathbf{x}_{u_{nu}}\}, \tag{6.20.2}$$

$$D^{(t)} = \{\mathbf{x}_{t_1}, \ldots, \mathbf{x}_{t_{n_t}}\}, \tag{6.20.3}$$

respectively.

There are two domains in machine learning: *source domain* $\mathcal{D}_S = \{\mathcal{X}_S, P(X_S)\}$ and *target domain* $\mathcal{D}_T = \{\mathcal{X}_T, P(X_T)\}$. These two domains are generally different: they may have different feature spaces $\mathcal{X}_S \neq \mathcal{X}_T$ or different marginal probability distributions $P(X_S) \neq P(X_T)$.

Let $X_S = \{\mathbf{x}_{S_1}, \ldots, \mathbf{x}_{S_{ns}}\}$ and $Y_S = \{y_{S_1}, \ldots, y_{S_{ns}}\}$ be, respectively, the sets of all labeled instances of source-domain $\mathcal{D}_S$ and all corresponding labels of source-domain. Similarly, denote $X_T = \{\mathbf{x}_{T_1}, \ldots, \mathbf{x}_{T_{n_t}}\}$ and $Y_T = \{y_{T_1}, \ldots, y_{T_{n_t}}\}$. The *source-domain data* and the *target-domain data* are, respectively, represented as

$$D_S = (X_S, Y_S) = X_S \cup Y_S = \left\{(\mathbf{x}_{S_1}, y_{S_1}), \ldots, (\mathbf{x}_{S_{n_s}}, y_{S_{n_s}})\right\} \in \mathcal{D}_S, \tag{6.20.4}$$

$$D_T = (X_T, Y_T) = X_T \cup Y_T = \left\{(\mathbf{x}_{T_1}, y_{T_1}), \ldots, (\mathbf{x}_{T_{n_t}}, y_{T_{n_t}})\right\} \in \mathcal{D}_T, \tag{6.20.5}$$

where

- $\mathbf{x}_{S_i} \in \mathcal{X}_S$ is the ith data instance of source-domain $\mathcal{D}_S$ and $y_{S_i} \in \mathcal{Y}_S$ is the corresponding class label for $\mathbf{x}_{S_i}$. For example, in document classification, the label set is a binary set containing true and false, i.e., $y_{S_i} \in \{\text{true, false}\}$ takes on a value of true or false, and $f(\mathbf{x})$ is the learner that predicts the label value for $\mathbf{x}$.
- $\mathbf{x}_{T_i} \in \mathcal{X}_T$ is the ith data instance of target-domain $\mathcal{D}_T$ and $y_{T_i} \in \mathcal{Y}_T$ is the corresponding class label for $\mathbf{x}_{T_i}$.
- n_s and n_t are the numbers of source-domain data and target-domain data, respectively. In most cases, $0 \le n_t \ll n_s$.

Let $D_S^{(l)}$ and $D_S^{(u)}$ be the labeled set and unlabeled set of data drawn from source domain $\mathcal{D}_S$, respectively. Similarly, $D_T^{(l)}$ and $D_T^{(u)}$ denote the labeled set and unlabeled set drawn from $\mathcal{D}_T$, respectively. Then we have the following notations:

$$D_S^{(l)} = \left\{\left(\mathbf{x}_{S_1}^{(l)}, y_{S_1}^{(l)}\right), \ldots, \left(\mathbf{x}_{S_{n_l}}^{(l)}, y_{S_{n_l}}^{(l)}\right)\right\} \quad \text{and} \quad D_S^{(u)} = \left\{\mathbf{x}_{S_1}^{(u)}, \ldots, \mathbf{x}_{S_{nu}}^{(u)}\right\},$$

$$D_T^{(l)} = \left\{\left(\mathbf{x}_{T_1}^{(l)}, y_{T_1}^{(l)}\right), \ldots, \left(\mathbf{x}_{T_{n_l}}^{(l)}, y_{T_{n_1}}^{(l)}\right)\right\} \quad \text{and} \quad D_T^{(u)} = \left\{\mathbf{x}_{T_1}^{(u)}, \ldots, \mathbf{x}_{T_{nu}}^{(u)}\right\},$$

where

- $\mathbf{x}_{S_i}^{(l)}, \mathbf{x}_{S_i}^{(u)} \in \mathcal{D}_S$ are the S_i-th labeled and unlabeled data vectors drawn from source domain, respectively; $y_{S_i}^{(l)} \in \mathcal{Y}_S$ is the class label corresponding to $\mathbf{x}_{S_i}^{(l)}$ drawn from source domain;
- $\mathbf{x}_{T_i}^{(l)}, \mathbf{x}_{T_i}^{(u)} \in \mathcal{D}_T$ are the T_i-th labeled and unlabeled data vectors drawn from target domain, respectively; $y_{T_i}^{(l)} \in \mathcal{Y}_T$ is the class label corresponding to $\mathbf{x}_{T_i}^{(l)}$ drawn from target domain;
- n_l and n_u are the numbers of labeled and unlabeled data in source (or target) domain.

It should be noted that $D_S^{(l)}$ and $D_S^{(u)}$ are disjoint: $D_S^{(l)} \cap D_S^{(u)} = \emptyset$. Similarly, we have also $D^{(l)} \cap D^{(u)} = \emptyset$, as required in supervised learning, unsupervised learning, semi-supervised learning, inductive learning and semi-supervised transductive learning discussed before.

When discussing machine learning in two different domains $\mathcal{D}_S$ and $\mathcal{D}_T$, we have two possible training sets D_S^{train} drawn from source domain $\mathcal{D}_S$ and D_T^{train} drawn from target domain $\mathcal{D}_T$ as follows:

$$D_S^{\text{train}} = D_S^{(l)} \cup D_S^{(u)} \in \mathcal{D}_S, \tag{6.20.6}$$

$$D_T^{\text{train}} = D_T^{(l)} \cup D_T^{(u)} \in \mathcal{D}_T, \tag{6.20.7}$$

where one of $D_S^{(l)}$ and $D_S^{(u)}$ may be empty, while $D_T^{(l)}$ or $D_T^{(u)}$ may be empty also, which corresponds to unsupervised and supervised learning cases, respectively.

Definition 6.45 (Task [199]) For a given (source or target) domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$, a *task* $\mathcal{T}$ consists of two parts: a label space $\mathcal{Y}$ and an objective predictive function $f(\cdot)$, denoted by $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$. The objective prediction function $f(\cdot)$ is not observed but can be typically learned from the feature vector and label pairs $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i \in X$ and $y_i \in \mathcal{Y}$.

The function $f(\cdot)$ can be used to predict the corresponding label $f(\mathbf{x}_i)$ of a new instance $\mathbf{x}_i$. From a probabilistic viewpoint, $f(\mathbf{x}_i)$ is usually written as $P(y_i|\mathbf{x}_i)$.

If the given specific domain is the source domain, i.e., $\mathcal{D} = \mathcal{D}_S$, then learning task is called the *source learning task*, denoted as $\mathcal{T}_S = \{\mathcal{Y}_S, f_S(\cdot)\}$. Similarly, learning task in target domain is known as the *target learning task*, denoted as $\mathcal{T}_T = \{\mathcal{Y}_T, f_T(\cdot)\}$.

For two different but relevant domains or tasks, if we transfer the model parameters trained in one domain (or task) to the new model in another domain (or task), it helps clearly to train the new data set because this transfer can share the learned parameters with the new model so as to speed up and optimize the learning of the new model without learning from zero as before. Such a machine learning is called transfer learning.

The following is a unified definition of transfer learning.

Definition 6.46 (Transfer Learning [199]) Given a source domain $\mathcal{D}_S$ and source learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and target learning task $\mathcal{T}_T$. *Transfer learning* emphasizes the transfer of knowledge across domains, tasks, and distributions that are similar but not the same and aims to help improve the learning of the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$.

This definition shows that transfer learning consists of three parts [302]:

- define source and target domains;
- learn on the source domain; and
- predict or generalize on the target domain.

In transfer learning the knowledge gained in solving the source (or target) task in the source (or target) domain is stored and then applied to solving another problem of interest. Figure 6.5 shows this transfer learning setup.

In most cases, there is only a single-loop transfer learning (shown by arrows with solid line). However, in machine translation of two languages there are two-loop transfer learning setups, as shown in Fig. 6.5.

In transfer learning, there are the following three main research issues [199, 268].

- *What to transfer:* Which part of knowledge can be transferred across domains or tasks? If some knowledge is specific for individual domains or tasks, then there is no need to transfer it. Conversely, when some knowledge may be common

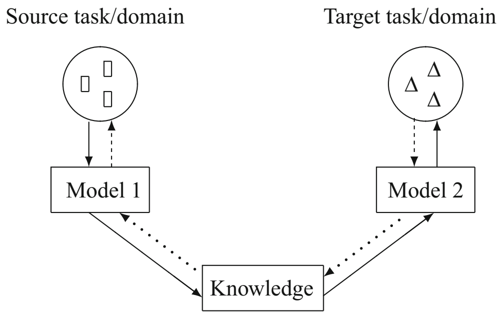

Fig. 6.5 Transfer learning setup: storing knowledge gained from solving one problem and applying it to a different but related problem

between different domains, it is necessary to transfer this part of knowledge because they may help improve performance for the target domain or task.

- *How to transfer:* How to train a suitable model? After discovering which knowledge can be transferred, learning algorithms need to be developed to transfer the knowledge.
- *When to transfer:* The high-level concept of transfer learning is to improve a target learner by using data from a related source domain. But if the source domain is not well-related to the target, then the target learner can be negatively impacted by this weak relation, which is referred to as *negative transfer*.

In Definition 6.46 on transfer learning, a domain is defined as a pair $\mathcal{D} = \{\mathcal{X}, P(X)\}$, and a task is a pair $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$. Hence, the condition on two different domains $\mathcal{D}_S \neq \mathcal{D}_T$ has two possible scenarios: $\mathcal{X}_S \neq \mathcal{X}_T$ and/or $P(X_S) \neq P(X_T)$, while the condition on two different tasks $\mathcal{T}_S \neq \mathcal{T}_T$ has also two possible scenarios: $\mathcal{Y}_S \neq \mathcal{Y}_T$ and/or $P(Y_S|X_S) \neq P(Y_T|X_T)$. In other words, there are the following four transfer learning scenarios:

1. $\mathcal{X}_S \neq \mathcal{X}_T$ means that the feature spaces between the source and target domains are different. For example, in the context of natural language processing, the documents written in two different languages belong to this scenario, which is generally referred to as *cross-lingual adaptation*.
2. $P(X_S) \neq P(X_T)$ implies that the marginal probability distributions between source and target domains are different, for example, when the documents focus on different topics. This scenario is generally known as *domain adaptation*.
3. $\mathcal{Y}_S \neq \mathcal{Y}_T$ means that the label spaces between the source and target domains are different. The case of $\mathcal{Y}_S \neq \mathcal{Y}_T$ refers to a mismatch in the class space. An example of this case is when the source software project has a binary label space of true for defect prone and false for not defect prone, and the target domain has a label space that defines five levels of fault prone modules [268].
4. $P(Y_S|X_S) \neq P(Y_T|X_T)$ implies that conditional probability distributions between the source and target domains are different. For example, source and target documents are unbalanced with regard to their classes. This scenario is quite common in practice.

6.20.2 Categorization of Transfer Learning

Feature-based transfer learning approaches are categorized in two ways [268], as shown in Fig. 6.6.

- *Asymmetric transformation* transforms the features of the source via reweighting to more closely match the target domain [201], as depicted in Fig. 6.6b.
- *Symmetric transformation* discovers underlying meaningful structures between the domains, shown in Fig. 6.6a.

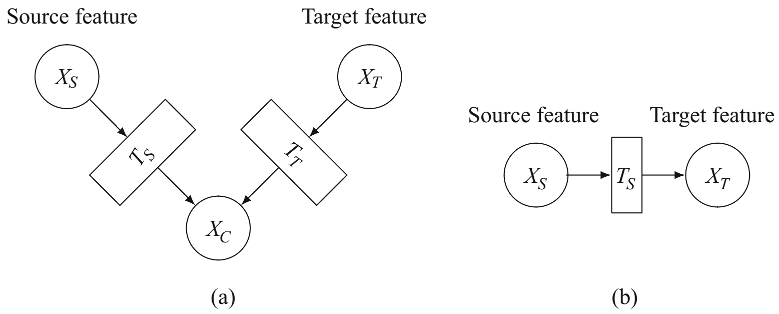

Fig. 6.6 The transformation mapping [268]. (**a**) The symmetric transformation (T_S and T_T) of the source-domain feature set $X_S = \{\mathbf{x}_{S_i}\}$ and target-domain feature set $X_T = \{\mathbf{x}_{T_i}\}$ into a common latent feature set $X_C = \{\mathbf{x}_{C_i}\}$. (**b**) The asymmetric transformation T_T of the source-domain feature set X_S to the target-domain feature set X_T

According to different perspectives, there are different categorization methods in transfer learning.

1. According to feature spaces, transfer learning can be categorized to two classes [268].
   - *Homogeneous transfer learning:* Source domain and target domain have the same feature space, i.e., $\mathcal{X}_S = \mathcal{X}_T$, including instance-based transfer learning [48], asymmetric feature-based transfer learning [70], domain adaptation [203], parameter-based transfer learning [249], relational-based transfer learning [161], hybrid-based (instance and parameter) transfer learning [280] and so on.
   - *Heterogeneous transfer learning:* Source domain and target domain have different feature spaces, i.e., $\mathcal{X}_S \neq \mathcal{X}_T$, including symmetric feature-based transfer learning in [207], asymmetric feature-based transfer learning in [153], and so on.
2. According to computational intelligence, transfer learning can be categorized as follows [167].
   - *Transfer learning using neural network* including transfer learning using deep neural network [55], transfer learning using convolutional neural networks [196], transfer learning using multiple task neural network [45, 233], and transfer learning using radial basis function neural networks [285].
   - *Transfer learning using Bayes* including transfer learning using naive Bayes [170, 217], transfer learning using Bayesian network [168, 192], and transfer learning using hierarchical Bayesian model [93].
   - *Transfer learning using fuzzy system and genetic algorithm* including fuzzy-based transductive transfer learning [18], framework of fuzzy transfer learning to form a prediction model in intelligent environments [227, 228], generalized hidden-mapping ridge regression [74], genetic transfer learning [148], and so on.

3. According to tasks, transfer learning can be categorized into the following three classes [199]:

   - inductive transfer learning,
   - transductive transfer learning [10], and
   - unsupervised transfer learning.

Definition 6.47 (Inductive Transfer Learning [199]) Let the target task $\mathcal{T}_T$ be different from the source task $\mathcal{T}_S$, i.e., $\mathcal{T}_S \neq \mathcal{T}_T$, no matter when the source and target domains are the same or not. *Inductive transfer learning* uses some labeled data in the target domain to induce an objective predictive model $f_T(\cdot)$ for use in the target domain.

Definition 6.48 (Transductive Transfer Learning [10, 199]) Given a source domain $\mathcal{D}_S$ and a corresponding learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and a corresponding learning task $\mathcal{T}_T$, *transductive transfer learning* aims to improve the learning of the target predictive function $f_T(\cdot)$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$. In addition, some unlabeled target-domain data must be available at training time.

Definition 6.49 (Unsupervised Transfer Learning [199]) When $\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{T}_S \neq \mathcal{T}_T$, and labeled data in source domain and target domain are not available, *unsupervised transfer learning* is to help to improve the learning of the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$.

From the above three definitions, we can compare the three transfer learning settings as follows.

1. *Common point:* The inductive transfer learning, the transductive transfer learning, and the unsupervised transfer learning do not assume that the unlabeled data $\mathbf{x}_j^{(u)}$ was drawn from the same distribution as, nor that it can be associated with the same class labels as, the labeled data. This is the common point in any type of transfer learning, and is the basic difference from the traditional machine learning.
2. *Comparison of assumptions:* The transductive transfer learning requires the source task and the target task must be the same, i.e., $\mathcal{T}_S = \mathcal{T}_T$, but both the inductive transfer learning and the unsupervised transfer learning require $\mathcal{T}_S \neq \mathcal{T}_T$.
3. *Comparison of data used:* Different from the unsupervised transfer learning that uses no labeled data, the inductive transfer learning uses some labeled data in the target domain, both transductive transfer learning and self-taught learning use some unlabeled data available at training time, while unsupervised transfer learning does not use any labeled data.
4. *Comparison of tasks:* The inductive transfer learning induces the target predictive model $f_T(\cdot)$, while both the transductive transfer learning and the unsupervised transfer learning aim to improve the learning of the target predictive function $f_T(\cdot)$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, but the inductive and unsupervised

transfer learning require $\mathcal{T}_S \neq \mathcal{T}_T$, while the transductive transfer learning requires $\mathcal{T}_S = \mathcal{T}_T$.

According to different situations of labeled and unlabeled data in the source domain, the inductive transfer learning setting can be categorized further into two cases [199].

- *Multitask learning:* a lot of labeled data in the source domain are available. In this case, the inductive transfer learning setting is similar to the multitask learning setting. However, the inductive transfer learning setting only aims at achieving high performance in the target task by transferring knowledge from the source task, while the multitask learning tries to learn the target and source tasks simultaneously.
- *Self-taught transfer learning:* no labeled data in the source domain are available. In this case, the inductive transfer learning setting is similar to the self-taught transfer learning setting proposed by Raina et al. [210].

Definition 6.50 (Multitask Learning) For multiple related tasks, *multitask learning* is defined as an inductive transfer learning method based on shared representation, in which multiple related tasks are learned together by using labeled data for source domain. What is learned for each task can help other tasks to learn better.

This definition shows that multitask learning consists of three parts [302]:

- model the task relatedness;
- learn all tasks simultaneously;
- tasks may have different data/features.

Definition 6.51 (Self-Taught Transfer Learning [210]) Given a labeled training set of m examples $\{(\mathbf{x}_1^{(l)}, y_1^{(l)}), \ldots, (\mathbf{x}_m^{(l)}, y_m^{(l)})\}$ drawn i.i.d. from some distribution D, where each $\mathbf{x}_i^{(l)} \in \mathbb{R}^n$ denotes an input feature vector (the subscript "l" indicates that it is a labeled example), and $y_i^{(l)} \in \{1, \ldots, C\}$ is the corresponding class label in the classification problems. In addition, we are given a set of k unlabeled examples $\mathbf{x}_1^{(u)}, \ldots, \mathbf{x}_k^{(u)} \in \mathbb{R}^n$. *Self-taught transfer learning* aims to determine if $\mathbf{x}_i^{(l)}$ and $\mathbf{x}_j^{(u)}$ come from the same input "type" or "modality" through transfer learning from unlabeled data.

In the self-taught learning setting, the label spaces between the source and target domains may be different, which implies the side information of the source domain cannot be used directly. Thus, it is similar to the inductive transfer learning setting where the labeled data in the source domain are unavailable.

In the traditional machine learning setting, the transductive learning [131] refers to the situation where all test data are required to be seen at training time, and that the learned model cannot be reused for future data. In contrast, the term "transductive" in Definition 6.48 emphasizes the concept in this type of transfer learning, the tasks must be the same and there must be some unlabeled data available in the target domain. Thus, when some new test data arrive, they must be classified together with all existing data.

In the transductive transfer learning setting, the source and target tasks are required to be the same, while the source and target domains are different. In this situation, no labeled data in the target domain are available, while a lot of labeled data in the source domain are available. According to different situations between the source and target domains, the transductive transfer learning can be further categorized into two cases.

- The feature spaces between the source and target domains are different, namely $\mathcal{X}_S \neq \mathcal{X}_T$. This is a so-called heterogeneous transfer learning.
- The feature spaces between domains are the same, i.e., $\mathcal{X}_S = \mathcal{X}_T$, but the marginal probability distributions of the input data are different: $P(X_S) \neq P(X_T)$. This special case is closely related to "domain adaptation".

Definition 6.52 (Domain Adaptation) Given two different domains $\mathcal{D}_S \neq \mathcal{D}_T$ and a single task $\mathcal{T}_S = \mathcal{T}_T$, *domain adaptation* aims to improve the learning of the target predictive function $f_T(\cdot)$ using labeled source-domain data and unlabeled or few labeled target-domain data.

There are two main existing approaches to transfer learning.

1. *Instance-based approach* learns different weights to rank training examples in a source domain for better learning in a target domain. For example, boosting for transfer learning of Dai et al. [66].
2. *Feature-based approach* tries to learn a common feature structure from different domains that can bridge the two domains for knowledge transfer. For example, multitask learning of Ando and Zhang [8]; multi-domain learning of Blitzer et al. [27]; self-taught learning of Raina et al. [210]; and transfer learning via dimensionality reduction of Pan et al. [201].

6.20.3 Boosting for Transfer Learning

Consider transfer learning in which there are only labeled data from a similar old domain, when a task from one new domain comes. In these cases, labeling the new data can be costly and it would also be a waste to throw away all the old data. A natural question to ask is whether it is possible to construct a high-quality classification model using only a tiny amount of new data and a large amount of old data, even when the new data are not sufficient to train a model alone. For this end, as an extension of adaptive boosting (AdaBoost) [97], an AdaBoost tailored for transfer learning purpose, called *TrAdaBoost*, was proposed by Dai et al. [66].

For the more or less outdated training data, there are certain parts of the data that can still be reused. In other words, knowledge learned from this part of the data can still be used in training a classifier for the new data. The aim of TrAdaBoost is to iteratively reduce low quality source domain data and to remain the reusable training data.

Assume that there are two types of training data [66].

- *Same-distribution training data:* a part of the labeled training data has the same distribution as the test data. The quantity of the same-distribution training data is often inadequate to train a good classifier for the test data, but access to this data is helpful for voting on the usefulness of each of the old data instances.
- *Diff-distribution training data:* the training data has a different distribution from the test data, perhaps because they are outdated. These data are assumed to be abundant, and the classifiers learned from these data cannot classify the test data well due to different data distributions.

The TrAdaBoost model uses boosting to filter out the different-distribution training data which are very different from the same-distribution data by automatically adjusting the weights of training instances, so that the remaining different-distribution data are treated as the additional training data which greatly boost the confidence of the learned model even when the same-distribution training data are scarce.

Let X_s be the same-distribution instance space, X_d be the diff-distribution instance space, and $Y = \{0, 1\}$ be the set of category labels. A concept is a Boolean function c mapping from X to Y, where $X = X_s \cup X_d$. The test data set is denoted by $S = \{(\mathbf{x}_i^t)\}$, where $\mathbf{x}_i^t \in X_s, i = 1, \ldots, k$. Here, k is the size of the unlabeled test set S. The training data set $T \subseteq \{X \times Y\}$ is partitioned into two labeled sets T_d and T_s, where T_d represents the diff-distribution training data that $T_d = \{(\mathbf{x}_i^d, c(\mathbf{x}_i^d))\}$ with $\mathbf{x}_i^d \in X_d, i = 1, \ldots, n$. T_s represents the same-distribution training data that $T_s = \{(\mathbf{x}_j^s, c(\mathbf{x}_j^s))\}$, where $\mathbf{x}_j^s \in X_s, j = 1, \ldots, m$, and n and m are the sizes of T_d and T_s, respectively. The combined training set $T = \{(\mathbf{x}_i, c(\mathbf{x}_i))\}$ is defined as

$$\mathbf{x}_i = \begin{cases} \mathbf{x}_i^d, & \text{if } i = 1, \ldots, n; \\ \\ \mathbf{x}_i^s, & \text{if } i = n + 1, \ldots, n + m. \end{cases} \tag{6.20.8}$$

Algorithm 6.43 gives a *transfer AdaBoost learning framework* (TrAdaBoost) developed by Dai et al. [66].

6.20.4 Multitask Learning

The standard supervised learning task is to seek a predictor that maps an input vector $\mathbf{x} \in \mathcal{X}$ to the corresponding output $y \in \mathcal{Y}$. Consider m learning problems indexed by $\ell \in \{1, \ldots, m\}$. Suppose we are given a finite set of training examples $\{(\mathbf{x}_1^\ell, y_1^\ell), \ldots, (\mathbf{x}_{n_\ell}^\ell, y_{n_\ell}^\ell)\}$ that are independently generated according to some unknown probability distribution $\mathcal{D}$, where $\ell = 1, \ldots, m$. Based on this finite set of training examples, the predictor is selected from a set $\mathcal{H}$ of functions. The set $\mathcal{H}$, called the *hypothesis space*, consists of functions mapping from $\mathcal{X}$ to $\mathcal{Y}$ that can be used to predict the output in $\mathcal{Y}$ of an input datum in $\mathcal{X}$.

Algorithm 6.43 TrAdaBoost algorithm [66]

```
1.  input: two labeled data sets $T_d$ and $T_s$, the unlabeled data set $S$, a base learning algorithm
    Learner, and the maximum number of iterations $N$.
2.  initialization: the initial weight vector $\mathbf{w}^1 = [w_1^1, \ldots, w_{n+m}^1]^T$, or the initial values is
    specified for $\mathbf{w}^1$.
3.  for $t = 1$ to $N$ do
4.    Set $\mathbf{p}^t = \mathbf{w}^t / (\sum_{i=1}^{n+m} \mathbf{w}_i^t)$.
5.    Call Learner, providing it the combined training set $T$ with the distribution $\mathbf{p}^t$ over $T$ and
      the unlabeled data set $S$. Then, get back a hypothesis $h_t : X \to Y$ (or [0, 1] by confidence).
6.    Calculate the error of $h_t$ on $T_s$:
        $\epsilon_t = \sum_{i=n+1}^{n+m} \frac{w_i^t \cdot |h_t(\mathbf{x}_i) - c(\mathbf{x}_i)|}{\sum_{i=n+1}^{n+m} w_i^t}$.
7.    Set $\beta_t = \epsilon_t/(1-\epsilon_t)$ and $\beta = 1/(1+\sqrt{2\ln n/N})$. Note that $\epsilon_t \leq 1/2$ is required.
8.    Update the new weight vector:
        $w_i^{t+1} = \begin{cases} w_i^t \beta^{|h_t(\mathbf{x}_i)-c(\mathbf{x}_i)|}, & \text{for } 1 \leq i \leq n; \\ w_i^t \beta_t^{|h_t(\mathbf{x}_i)-c(\mathbf{x}_i)|}, & \text{for } n+1 \leq i \leq n+m. \end{cases}$
9.  end for
10. output: the hypothesis:
      $h_f(\mathbf{x}) = \begin{cases} 1, & \text{if } \prod_{t=\lceil N/2 \rceil} \beta_t^{-h_t(\mathbf{x})} \geq \prod_{t=\lceil N/2 \rceil} \beta_t^{-1/2}; \\ 0, & \text{otherwise.} \end{cases}$
```

In the multitask scenario the goal of a learning system is to solve all multiple supervised learning tasks (such as recognizing objects or predicting attributes) by sharing information between them.

Definition 6.53 (Structural Learning [8]) *Structural learning* aims to learn some underlying predictive functional structures (smooth function classes) that can characterize what good predictors are like. In other words, its goal is to find a predictor f so that its error with respect to $\mathcal{D}$ is as small as possible.

Given the input space $\mathcal{X}$, a linear predictor is not necessarily linear on the original space $\mathcal{X}$, but rather can be regarded as a linear functional on a high-dimensional feature space $\mathcal{F}$. Assume there is a known feature map $\Phi : \mathcal{X} \to \mathcal{F}$. Therefore, a known high-dimensional feature map is given by $\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$, where $\mathbf{w}$ is a weight vector on high-dimensional feature map $\boldsymbol{\phi}(\mathbf{x})$. In order to apply the structural learning framework, consider a parameterized low-dimensional feature map given by $\mathbf{v}^T \boldsymbol{\psi}_\theta(\mathbf{x})$, where $\mathbf{v}$ is a weight vector on low-dimensional feature map $\boldsymbol{\psi}(\mathbf{x})$, θ denotes the common structure parameter shared by all m learning problems. That is, the linear predictor $f(\mathbf{x})$ in structural learning framework has a form [8]:

$$f(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) + \mathbf{v}^T \boldsymbol{\psi}_\theta(\mathbf{x}). \tag{6.20.9}$$

In order to simplify numerical computation, Ando and Zhang [8] propose using a simple linear form of feature map, $\boldsymbol{\psi}_\theta(\mathbf{x}) = \boldsymbol{\Theta}\boldsymbol{\psi}(\mathbf{x})$, where $\boldsymbol{\Theta}$ is an $h \times p$ matrix, and $\boldsymbol{\psi}$ is a known p-dimensional vector function. Then, the linear predictor can be reformulated as [8]:

$$f_\Theta(\mathbf{w}, \mathbf{v}; \mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) + \mathbf{v}^T \boldsymbol{\Theta}\boldsymbol{\psi}(\mathbf{x}). \tag{6.20.10}$$

Letting $\boldsymbol{\phi}(\mathbf{x}) = \boldsymbol{\psi}(\mathbf{x}) = \mathbf{x}_i^\ell \in \mathbb{R}^p$, then

$$f_\ell\left(\mathbf{w}_\ell, \mathbf{v}_\ell; \mathbf{x}_i^\ell\right) = \left(\mathbf{w}_\ell^T + \mathbf{v}_\ell^T \boldsymbol{\Theta}\right)\mathbf{x}_i^\ell, \quad \ell = 1, \ldots, m, \tag{6.20.11}$$

and the empirical error $L(f_\ell(\mathbf{w}_\ell, \mathbf{v}_\ell; \boldsymbol{\Theta}), y_i^\ell)$ on training data $\{(\mathbf{x}_1^\ell, y_i^\ell), \ldots, (\mathbf{x}_{n_\ell}^\ell, y_{n_\ell}^\ell)\}$ can be taken as the modified Huber loss function [8]:

$$L(p, y) = \begin{cases} \max(0, 1 - py)^2, \text{ if } py \geq -1; \\ \\ -4py, \qquad \text{otherwise.} \end{cases} \tag{6.20.12}$$

Hence, the optimization problem for multitask learning can be written as

$$[\{\hat{\mathbf{w}}_\ell, \hat{\mathbf{v}}_\ell\}; \hat{\boldsymbol{\Theta}}] = \underset{\{\mathbf{w},\mathbf{v}\};\boldsymbol{\Theta}}{\arg\min} \left\{ \frac{1}{n_\ell} \sum_{i=1}^{n_\ell} L\big(f_\ell(\mathbf{w}_\ell, \mathbf{v}_\ell; \boldsymbol{\Theta}), y_i^\ell\big) + \lambda_\ell \|\mathbf{w}\|_2^2 \right\}. \tag{6.20.13}$$

Algorithm 6.44 shows the SVD-based alternative LS solution for $\mathbf{w}_\ell$.

Algorithm 6.44 SVD-based alternating structure optimization algorithm [8]

```
1.  input: training data $\{(\mathbf{x}_i^\ell, y_i^\ell)\}, l = 1, \ldots, m; i = 1, \ldots, n_\ell$; parameters: $h$ and $\lambda_1, \ldots, \lambda_m$.
2.  initialization: $\mathbf{u}_\ell = \mathbf{0}$ ($\ell = 1, \ldots, m$) and arbitrary matrix $\boldsymbol{\Theta}$.
3.  iterate
4.    for $\ell = 1$ to $m$ do
5.      With fixed $\boldsymbol{\Theta}$ and $\mathbf{v}_\ell = \boldsymbol{\Theta}\mathbf{u}_\ell$, approximately solve for $\hat{\mathbf{w}}_\ell$:
          $\hat{\mathbf{w}}_\ell = \arg\min_{\mathbf{w}_\ell} \left\{ \frac{1}{n_\ell} \sum_{i=1}^{n_\ell} L(\mathbf{w}_\ell^T \mathbf{x}_i^\ell + (\mathbf{v}_\ell^T \boldsymbol{\Theta})\mathbf{x}_i^\ell, y_i^\ell) + \lambda_\ell \|\mathbf{w}_\ell\|_2^2 \right\}$
6.      Let $\mathbf{u}_\ell = \hat{\mathbf{w}}_\ell + \boldsymbol{\Theta}^T \mathbf{v}_\ell$.
7.    end for
8.    Compute the SVD of $\mathbf{U} = [\sqrt{\lambda_1}\mathbf{u}_1, \ldots, \sqrt{\lambda_m}\mathbf{u}_m]$:
          $\mathbf{U} = \mathbf{V}_1 \mathbf{D} \mathbf{V}_2^T$ (with diagonals of $\mathbf{D}$ in descending order).
9.    Let the rows of $\boldsymbol{\Theta}$ be the first $h$ rows of $\mathbf{V}_1^T$.
10. until converge
11. output: $h \times p$ dimensional matrix $\boldsymbol{\Theta}$.
```

For many natural language processing (NLP) tasks, due to new domains in which labeled data is scarce or non-existent, one seeks to adapt existing models from a resource-rich source domain to a resource-poor target domain. For this end, a structural correspondence learning is introduced by Blitzer et al. [27] to automatically induce correspondences among features from different domains.

Definition 6.54 (Structural Correspondence Learning [27]) *Structural correspondence learning* (SCL) is to identify correspondences among features from different domains by modeling their correlations with *pivot features*. Pivot features are features which behave in the same way for discriminative learning in both domains. *Non-pivot features* from different domains which are correlated with many of the same pivot features are assumed to correspond, and they are treated similarly in a discriminative learner.

The structural correspondence learning method consists of the following three steps.

- Find pivot features, e.g., using one of the feature selection methods presented in Sect. 6.7. The pivot predictors are the key element in structural correspondence learning.
- Use the weigh vectors $\hat{\mathbf{w}}_\ell$ to encode the covariance of the non-pivot features with the pivot features in order to learn a mapping matrix $\boldsymbol{\Theta}$ from the original feature spaces of both domains to a shared, low-dimensional real-valued feature space. This is the core of structural correspondence learning.
- For each source $\mathcal{D}_\ell$, $\ell = 1, \ldots, m$, construct its linear predictor on target domain:

$$f_\ell(\mathbf{x}) = \text{sign}\left(\hat{\mathbf{w}}_\ell^T \mathbf{x}\right), \quad \ell = 1, \ldots, m. \tag{6.20.14}$$

Algorithm 6.45 summarizes the structural correspondence learning algorithm developed by Blitzer et al. [27].

Algorithm 6.45 Structural correspondence learning algorithm [27]

```
1. input: labeled source data $\{(\mathbf{x}_t, y_t)_{t=1}^T\}$ and unlabeled data $\{\mathbf{x}_j\}$ from both domains.
2. Choose $m$ pivot features $\tilde{\mathbf{x}}_i, i = 1, \ldots, m$ using feature selection method.
3. for $\ell = 1$ to $m$ do
4.    $\hat{\mathbf{w}}_\ell = \arg\min_{\mathbf{w}} \left\{\sum_j L(\mathbf{w}^T\mathbf{x}_j, p_\ell(\mathbf{x}_j)) + \lambda\|\mathbf{w}\|_2^2\right\}$,
   where $L(p, q)$ is the modified Huber loss function given in (6.20.12).
5. end for
6. Construct the matrix $\mathbf{W} = [\hat{\mathbf{w}}_1, \ldots, \hat{\mathbf{w}}_m]$.
7. Compute the SVD $[\mathbf{U}, \mathbf{D}, \mathbf{V}^T]$ = SVD of $\mathbf{W}$.
8. Put $\boldsymbol{\Theta} = \mathbf{U}_{1:h,:}^T$.
9. output: predictor $f_\ell : X_\ell \to Y_T$ is given by $f_\ell(\mathbf{x}) =$ sign$(\hat{\mathbf{w}}_\ell^T\mathbf{x})$, where $\ell = 1, \ldots, m$.
```

6.20.5 EigenTransfer

In transfer learning, we are given two target data sets: a target training data set $\mathcal{X}_t = \{\mathbf{x}_i^{(t)}\}_{i=1}^n$ with labels, and a target test data set $\mathcal{X}_u = \{\mathbf{x}_i^{(u)}\}_{i=1}^k$ to be predicted. Different from traditional machine learning, we are also given an auxiliary data set $\mathcal{X}_a = \{\mathbf{x}_i^{(t)}\}_{i=1}^m$ to help the target learning.

In spectral learning [53], the input is a weighted graph $G = (V, E)$ to represent the target task, where $V = \{v_i\}_{i=1}^n$ and $E = \{e_{ij}\}_{i,j=1}^{n,n}$ represent the node set and the edge set in the graph, respectively.

To apply spectral learning to transfer learning, construct a task graph $G(V, E)$ in the following way [68].

- *Node construction:* the nodes V in the graph G represent features $\{f^{(i)}\}_{i=1}^{s}$, instances $\{\mathbf{x}_i(t)\}_{i=1}^{n}$, $\{\mathbf{x}_i^{(a)}\}_{i=1}^{m}$, $\{\mathbf{x}_i^{(u)}\}_{i=1}^{k}$ or class labels $c(\mathbf{x}_i)$.
- *Edge construction:* the edges E represent the relations between these nodes, where the weights of the edges are based on the number of co-occurrences between the end nodes in the target and auxiliary data.

Consequently, the task graph $G(V, E)$ contains almost all the information for the transfer learning task, including the target data and the auxiliary data. Usually, the task graph G is sparse, symmetric, real, and positive semi-definite. Thus, the spectra of the graph G can be calculated very efficiently.

The following are two steps for a unified framework for graph transfer learning [68].

1. Construct the weight graph $G(V, E)$ for three types of transfer learning.
   - *Cross-domain learning:* a graph $G(V, E)$ for representing the problem of cross-domain learning is given by

$$V = \mathcal{X}_t \cup \mathcal{X}_a \cup \mathcal{X}_u \cup \mathcal{F} \cup \mathcal{C}, \tag{6.20.15}$$

$$e_{ij} = \begin{cases} \phi_{v_i,v_j}, & \text{if } v_i \in \mathcal{X}_t \cup \mathcal{X}_a \cup \mathcal{X}_u \wedge v_j \in \mathcal{F}; \\ \phi_{v_j,v_i}, & \text{if } v_i \in \mathcal{F} \wedge v_j \in \mathcal{X}_t \cup \mathcal{X}_a \cup \mathcal{X}_u; \\ 1, & \text{if } v_i \in \mathcal{X}_t \wedge v_j \in \mathcal{C} \wedge C(v_i) = v_j; \\ 1, & \text{if } v_i \in \mathcal{C} \wedge v_j \in \mathcal{X}_t \wedge C(v_j) = v_i; \\ 1, & \text{if } v_i \in \mathcal{X}_a \wedge v_j \in \mathcal{C} \wedge C(v_i) = v_j; \\ 1, & \text{if } v_i \in \mathcal{C} \wedge v_j \in \mathcal{X}_a \wedge C(v_j) = v_i; \\ 0, & \text{otherwise.} \end{cases} \tag{6.20.16}$$

   - *Cross-category learning:* a graph $G(V, E)$ for representing the problem of cross-category learning is given by

$$V = \mathcal{X}_t \cup \mathcal{X}_a \cup \mathcal{X}_u \cup \mathcal{F} \cup \mathcal{C}_t \cup \mathcal{C}_a, \tag{6.20.17}$$

$$e_{ij} = \begin{cases} \phi_{v_i,v_j}, & \text{if } v_i \in \mathcal{X}_t \cup \mathcal{X}_a \cup \mathcal{X}_u \wedge v_j \in \mathcal{F}; \\ \phi_{v_j,v_i}, & \text{if } v_i \in \mathcal{F} \wedge v_j \in \mathcal{X}_t \cup \mathcal{X}_a \cup \mathcal{X}_u; \\ 1, & \text{if } v_i \in \mathcal{C}_t \wedge v_j \in \mathcal{C}_t \wedge \mathcal{C}(v_i) = v_j; \\ 1, & \text{if } v_i \in \mathcal{C}_t \wedge v_j \in \mathcal{X}_t \wedge \mathcal{C}(v_j) = v_i; \\ 1, & \text{if } v_i \in \mathcal{X}_a \wedge v_j \in \mathcal{C}_a \wedge \mathcal{C}(v_i) = v_j; \\ 1, & \text{if } v_i \in \mathcal{C}_a \wedge v_j \in \mathcal{X}_a \wedge \mathcal{C}(v_j) = v_i; \\ 0, & \text{otherwise.} \end{cases} \tag{6.20.18}$$

- *Self-taught learning:* $G(V, E)$ is given by

$$V = \mathcal{X}_t \cup \mathcal{X}_a \cup \mathcal{X}_u \cup \mathcal{F} \cup \mathcal{C}_t, \tag{6.20.19}$$

$$e_{ij} = \begin{cases} \phi_{v_i,v_j}, & \text{if } v_i \in \mathcal{X}_t \cup \mathcal{X}_a \cup \mathcal{X}_u \wedge v_j \in \mathcal{F}; \\ \phi_{v_j,v_i}, & \text{if } v_i \in \mathcal{F} \wedge v_j \in \mathcal{X}_t \cup \mathcal{X}_a \cup \mathcal{X}_u; \\ 1, & \text{if } v_i \in \mathcal{X}_t \wedge v_j \in \mathcal{C}_t \wedge \mathcal{C}(v_i) = v_j; \\ 1, & \text{if } v_i \in \mathcal{C}_t \wedge v_j \in \mathcal{X}_t \wedge \mathcal{C}(v_j) = v_i; \\ 0, & \text{otherwise.} \end{cases} \tag{6.20.20}$$

In the above equations, $\phi_{x,f}$ denotes the importance of the feature $f \in \mathcal{F}$ appearing in the instance $\mathbf{x} \in \mathcal{X}_t \cup \mathcal{X}_a \cup X_u$, and $\mathcal{C}(\mathbf{x})$ means the true label of the instance $\mathbf{x}$.

2. Learning graph spectra.

- Construct an adjacency matrix $\mathbf{W} \in \mathbb{R}^{n\times n}$ with respect to the graph $G(V, E)$:

$$\mathbf{W} = \begin{bmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nn} \end{bmatrix} \quad (w_{ij} = e_{ij}). \tag{6.20.21}$$

- Construct the diagonal matrix $\mathbf{D} = [d_{ij}]_{i,j=1}^{n,n}$:

$$d_{ij} = \begin{cases} \sum_{t=1}^{n} w_{it}, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases} \tag{6.20.22}$$

- Calculate the Laplacian matrix of G: $\mathbf{L} = \mathbf{D} - \mathbf{W}$.
- Use the normalized cut technique [230] to learn the new feature representation to enable transfer learning:
  (a) calculate the first m generalized eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_m$ of the generalized eigenproblem $\mathbf{L}\mathbf{v} = \lambda\mathbf{D}\mathbf{v}$.
  (b) use the first m generalized eigenvectors to construct the feature representation for transfer learning: $\mathbf{U} = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$.

The above graph transfer learning based on spectrum features is called *Eigen-Transfer* [68].

EigenTransfer can model a variety of existing transfer learning problems and solutions. In this framework, a task graph is first constructed to represent the transfer learning task. Then, an eigen feature representation is learned from the task graph based on spectral learning theory.

Under the new feature representation, the knowledge from the auxiliary data tends to be transferred to help the target learning.

Algorithm 6.46 gives an *EigenCluster* algorithm.

Algorithm 6.46 EigenCluster: a unified framework for transfer learning [68]

```
1.  input: the target data set $\mathcal{X}_t = \{\mathbf{x}_i^{(t)}\}_{i=1}^n$, the auxiliary data set $\mathcal{X}_a = \{\mathbf{x}_i^{(a)}\}_{i=1}^m$, and the test
    data set $\mathcal{X}_u = \{\mathbf{x}_i^{(u)}\}_{i=1}^k$.
2.  Construct the task graph $G(V, E)$ based on the target clustering task (c.f. (6.20.15)–(6.20.19)).
3.  Construct the $n \times n$ adjacent matrix $\mathbf{W}$ whose entries $w_{ij} = e_{ij}$ are based on the task graph
    $G(V, E)$.
4.  Use (6.20.22) to calculate the diagonal matrix $\mathbf{D}$.
5.  Construct the Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{W}$.
6.  Calculate the first $N$ generalized eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_N$ of $\mathbf{Lv} = \lambda\mathbf{Dv}$.
7.  Let $\mathbf{U} = [\mathbf{v}_1, \ldots, \mathbf{v}_N]$.
8.  for each $\mathbf{x}_i^{(t)} \in \mathcal{X}_t$ do
9.      Let $y_i^{(t)}$ be the corresponding row in $\mathbf{U}$ with respect to $\mathbf{x}_i^{(t)}$.
10. end for
11. Train a classifier based on $\mathcal{Y}^{(t)} = \{y_i^{(t)}\}_{i=1}^n$ instead of $\mathcal{X}_t = \{\mathbf{x}_i^{(t)}\}_{i=1}^n$ using a traditional
    classification algorithm, and then classify the test data $\mathcal{X}_u = \{\mathbf{x}_i^{(u)}\}_{i=1}^k$.
12. output: classification result on $\mathcal{X}_u$.
```

In the next section, we focus on domain adaptation.

6.21 Domain Adaptation

The task of *domain adaptation* is to develop learning algorithms that can be easily ported from one domain to another. This problem is particularly interesting and important when we have a large collection of labeled data in one "source" domain but truly desire a model that performs well in a second "target" domain.

Let $\mathcal{S} = \{\mathbf{x}_i^{(s)}, y_i^{(s)}\}_{i=1}^{N_s}$, where $\mathbf{x}_i^{(s)} \in \mathbb{R}^N$ denotes the labeled data from the source domain and is referred to as an observation, and $y_i^{(s)}$ is the corresponding class label. Labeled data from the target domain is denoted by $\mathcal{T}_l = \{\mathbf{x}_i^{(t)}, y_i^{(t)}\}_{i=1}^{N_{t_l}}$, where $\mathbf{x}_i^{(t)} \in \mathbb{R}^M$. Similarly, unlabeled data in the target domain is denoted by $\mathcal{T}_u = \{\mathbf{x}_i^{(u)}\}_{i=1}^{N_{t_u}}$, where $\mathbf{x} \in \mathbb{R}^M$. It is usually assumed that $N = M$. Denoting $\mathcal{T} = \mathcal{T}_l \cup \mathcal{T}_u$, then $N_t = N_{t_l} + N_{t_u}$ denotes the total number of samples in the target domain.

Let $\mathbf{S} = [\mathbf{x}_1^{(s)}, \ldots, \mathbf{x}_{N_s}^{(s)}]$ be the matrix of N_s data points from the source domain $\mathcal{S}$. Similarly, for target domain $\mathcal{T}$, let $\mathbf{T}_l = [\mathbf{x}_1^{(t)}, \ldots, \mathbf{x}_{N_{t_l}}^{(t)}]$ be the matrix of N_{t_l} data from $\mathcal{T}_l$, $\mathbf{T}_u = [\mathbf{x}_1^{(u)}, \ldots, \mathbf{x}_{N_{t_l}}^{(u)}]$ be the matrix of N_{t_u} data from $\mathcal{T}_u$, and $\mathbf{T} = [\mathbf{T}_l, \mathbf{T}_u] = [\mathbf{x}_1^{(t)}, \ldots, \mathbf{x}_{N_t}^{(t)}]$ be the matrix of N_t data from $\mathcal{T}$.

The goal of domain adaptation is to learn a function $f(\cdot)$ that predicts the class label of a new test sample from the target domain. Depending on the availability of the source and target-domain data, the domain adaptation problem can be defined in many different ways [205].

- In semi-supervised domain adaptation, the function $f(\cdot)$ is learned using the knowledge in both $\mathcal{S}$ and $\mathcal{T}_l$.
- In unsupervised domain adaptation, the function $f(\cdot)$ is learned using the knowledge in both $\mathcal{S}$ and $\mathcal{T}_u$.
- In multi-source domain adaptation, $f(\cdot)$ is learned from more than one domain in $\mathcal{S}$ accompanying each of the first two cases.
- In the heterogeneous domain adaptation, the dimensions of features in the source and target domains are assumed to be different. In other words, $N \neq M$.

In what follows, we present a number of domain adaptation approaches.

6.21.1 Feature Augmentation Method

One of the simplest domain adaptation approaches is the *feature augmentation* of Daumé [70].

For each given feature $\mathbf{x} \in \mathbb{R}^F$ in the original problem, make its three versions: a general version, a source-specific version, and a target-specific version. The augmented source data $\boldsymbol{\phi}_s(\mathbf{x}) : \mathbb{R}^F \to \mathbb{R}^{3F}$ will contain only general and source-specific versions, and the augmented target data $\boldsymbol{\phi}_t(\mathbf{x}) : \mathbb{R}^F \to \mathbb{R}^{3F}$ contains general and target-specific versions as follows:

$$\boldsymbol{\phi}_s(\mathbf{x}) = \begin{bmatrix} \mathbf{x} \\ \mathbf{x} \\ \mathbf{0} \end{bmatrix}, \quad \boldsymbol{\phi}_t(\mathbf{x}) = \begin{bmatrix} \mathbf{x} \\ \mathbf{0} \\ \mathbf{x} \end{bmatrix}, \tag{6.21.1}$$

where $\mathbf{0}$ is an $F \times 1$ zero vector.

Consider the *heterogeneous domain adaptation* (HDA) problem where the dimensions of data from the source and target domains are different. By introducing a common subspace for the source and target data, both the source and target data of dimension N and M are, respectively, projected onto a latent domain of dimension l using two projection matrices $\mathbf{W}_1 \in \mathbb{R}^{l+N}$ and $\mathbf{W}_2 \in \mathbb{R}^{l+M}$. The augmented feature maps for the source and target domains in the common space are then defined as [162]:

$$\boldsymbol{\phi}_s\left(\mathbf{x}_i^{(s)}\right) = \begin{bmatrix} \mathbf{W}_1\mathbf{x}_i^{(s)} \\ \mathbf{x}_i^{(s)} \\ \mathbf{0}_M \end{bmatrix} \in \mathbb{R}^{l+N+M}, \tag{6.21.2}$$

$$\boldsymbol{\phi}_t\left(\mathbf{x}_i^{(l)}\right) = \begin{bmatrix} \mathbf{W}_2\mathbf{x}_i^{(t)} \\ \mathbf{0}_N \\ \mathbf{x}_i^{(t)} \end{bmatrix} \in \mathbb{R}^{l+N+M}, \tag{6.21.3}$$

where $\mathbf{x}_i^{(s)} \in \mathcal{S}, \mathbf{x}_i^{(t)} \in \mathcal{T}_l$ and $\mathbf{0}_M$ is an $M \times 1$ zero vector.

After introducing $\mathbf{W}_1$ and $\mathbf{W}_2$, the data from two domains can be readily compared in the common subspace. Once the data from both domains are transformed onto a common space, *heterogeneous feature augmentation* (HFA) [162] can be readily formulated as follows:

$$\min_{\mathbf{W}_1,\mathbf{W}_2}\ \min_{\mathbf{w},b,\xi_i^{(s)},\xi_i^{(t)}} \left\{\frac{1}{2}\|\mathbf{w}\|^2 + C\left(\sum_{i=1}^{N_s}\xi_i^{(s)} + \sum_{i=1}^{N_t}\xi_i^{(t)}\right)\right\}, \tag{6.21.4}$$

$$\text{subject to}\quad y_i^{(s)}\left(\mathbf{w}^T\boldsymbol{\phi}_s\left(\mathbf{x}_i^{(s)}\right) + b\right) \geq 1 - \xi_i^{(s)},\ \xi_i^{(s)} \geq 0, \tag{6.21.5}$$

$$y_i^{(t)}\left(\mathbf{w}^T\boldsymbol{\phi}_t\left(\mathbf{x}_i^{(t)}\right) + b\right) \geq 1 - \xi_i^{(t)},\ \xi_i^{(t)} \geq 0, \tag{6.21.6}$$

$$\|\mathbf{W}_1\|_F^2 \leq \lambda_{w1},\ \|\mathbf{W}_2\|_F^2 \leq \lambda_{w2}, \tag{6.21.7}$$

where $C > 0$ is a tradeoff parameter which balances the model complexity and the empirical losses on the training samples from two domains, and λ_{w1} and $\lambda_{w2} > 0$ are predefined parameters to control the complexities of $\mathbf{W}_1$ and $\mathbf{W}_2$, respectively.

Denote the kernel on the source domain samples as $\mathbf{K}_s = \boldsymbol{\Phi}_s^T\boldsymbol{\Phi}_s \in \mathbb{R}^{n_s\times n_s}$, where $\boldsymbol{\Phi}_s = [\boldsymbol{\phi}_s(\mathbf{x}_1^{(s)}), \ldots, \boldsymbol{\phi}_s(\mathbf{x}_{n_s}^{(s)})]$ and $\boldsymbol{\phi}_s(\cdot)$ is the nonlinear feature mapping function induced by $\mathbf{K}_s$. Similarly, the kernel on the target-domain samples is denoted as $\mathbf{K}_t = \boldsymbol{\Phi}_t^T\boldsymbol{\Phi}_t \in \mathbb{R}^{n_t\times n_t}$ where $\boldsymbol{\Phi}_t = [\boldsymbol{\phi}_t(\mathbf{x}_1^{(t)}), \ldots, \boldsymbol{\phi}_t(\mathbf{x}_{n_t}^{(t)})]$ and $\boldsymbol{\phi}_t(\cdot)$ is the nonlinear feature mapping function induced by $\mathbf{K}_t$.

Define the feature weight vector $\mathbf{w} = [\mathbf{w}_c, \mathbf{w}_s, \mathbf{w}_t] \in \mathbb{R}^{d_c+d_s+d_t}$ for the augmented feature space, where $\mathbf{w}_c \in \mathbb{R}^{d_c}$, $\mathbf{w}_s \in \mathbb{R}^{d_s}$ and $\mathbf{w}_t \in \mathbb{R}^{d_t}$ are also the weight vectors defined for the common subspace, the source domain, and the target domain, respectively. Introduce a *transformation metric*

$$\mathbf{H} = [\mathbf{W}_1, \mathbf{W}_2]^T[\mathbf{W}_1, \mathbf{W}_2] \in \mathbb{R}^{(d_s+d_t)\times(d_s+d_t)} \tag{6.21.8}$$

which is positive semi-definite, i.e., $\mathbf{H} \succeq 0$.

With $\mathbf{H}$, the optimization problem can be reformulated as follows [162]:

$$\min_{\mathbf{H}\succeq 0}\max_{\boldsymbol{\alpha}} \left\{J(\boldsymbol{\alpha}) = \mathbf{1}^T\boldsymbol{\alpha} - \frac{1}{2}(\boldsymbol{\alpha}\odot\mathbf{y})^T\mathbf{K}_{\mathrm{H}}(\boldsymbol{\alpha}\odot\mathbf{y})\right\}, \tag{6.21.9}$$

$$\text{subject to}\quad \mathbf{y}^T\boldsymbol{\alpha} = 0,\ \mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1},\qquad \mathrm{tr}(\mathbf{H}) \leq \lambda, \tag{6.21.10}$$

where $\mathbf{a} \odot \mathbf{b}$ denotes the elementwise product of two vectors $\mathbf{a}$ and $\mathbf{b}$, and

- $\boldsymbol{\alpha} = [\alpha_1^{(s)}, \ldots, \alpha_{n_s}^{(s)}, \alpha_1^{(t)}, \ldots, \alpha_{n_t}^{(t)}]^T$ is the vector of dual variables;
- $\mathbf{y} = [y_1^{(s)}, \ldots, y_{n_s}^{(s)}, y_1^{(t)}, \ldots, y_{n_t}^{(t)}]^T$ is the label vector of all training instances;
- $\mathbf{K}_{\mathrm{H}} = \boldsymbol{\Phi}^T(\mathbf{H}+\mathbf{I})\boldsymbol{\Phi}$ is the derived kernel matrix for the samples from source and target domains, where $\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\Phi}_s & \mathbf{O}_{d_s \times n_t} \\ \mathbf{O}_{d_t \times n_s} & \boldsymbol{\Phi}_t \end{bmatrix} \in \mathbb{R}^{(d_s+d_t)\times(n_s+n_t)}$;
- $\lambda = \lambda_{w1} + \lambda_{w2}$.

Let transformation metric $\mathbf{H}$ consist of the linear combination of rank-one normalized positive semi-definite (PSD) matrices:

$$\mathbf{H}_{\boldsymbol{\theta}} = \sum_{r=1}^{\infty} \theta_r \mathbf{M}_r, \quad \mathbf{M}_r \in \mathcal{M}, \tag{6.21.11}$$

where θ_r is the linear combination coefficients, and $\mathcal{M} = \{\mathbf{M}_r|_{r=1}^{\infty}\}$ is the set of rank-one normalized PSD matrices $\mathbf{M}_r = \mathbf{h}_r\mathbf{h}_r^T$ with $\mathbf{h}_r \in \mathbb{R}^{n_s+n_t}$ and $\mathbf{h}_r^T\mathbf{h}_r = 1$. If denoting $\mathbf{K} = \boldsymbol{\Phi}^T\boldsymbol{\Phi} = \mathbf{K}^{1/2}\mathbf{K}^{1/2}$, then the optimization problem of heterogeneous feature augmentation in Eqs. (6.21.9)–(6.21.10) can be reformulated as [162]:

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\alpha}\in\mathcal{A}} \left\{ \mathbf{1}^T\boldsymbol{\alpha} - \frac{1}{2}(\boldsymbol{\alpha}\odot\mathbf{y})^T\mathbf{K}^{1/2}(\mathbf{H}_{\boldsymbol{\theta}}+\mathbf{I})\mathbf{K}^{1/2}(\boldsymbol{\alpha}\odot\mathbf{y}) \right\}, \tag{6.21.12}$$

$$\text{subject to} \quad \mathbf{H}_{\boldsymbol{\theta}} = \sum_{r=1}^{\infty} \theta_r \mathbf{M}_r,\ \mathbf{M}_r \in \mathcal{M}, \quad \mathbf{1}^T\boldsymbol{\theta} \le \lambda,\ \boldsymbol{\theta} \ge 0. \tag{6.21.13}$$

By setting $\boldsymbol{\theta} \leftarrow \frac{1}{\lambda}\boldsymbol{\theta}$, then the above heterogeneous feature augmentation can be further rewritten as [162]:

$$\min_{\boldsymbol{\theta}\in\mathcal{D}_\theta} \max_{\boldsymbol{\alpha}\in\mathcal{A}} \left\{ \mathbf{1}^T\boldsymbol{\alpha} - \frac{1}{2}(\boldsymbol{\alpha}\odot\mathbf{y})^T \left(\sum_{r=1}^{\infty} \theta_r \mathbf{K}_r \right) (\boldsymbol{\alpha}\odot\mathbf{y}) \right\}, \tag{6.21.14}$$

where $\mathcal{A} = \{\boldsymbol{\alpha}|\boldsymbol{\alpha}^T\mathbf{y} = 0, \mathbf{0}_n \le \boldsymbol{\alpha} \le C\mathbf{1}_n\}$ is the feasible set of the dual variables $\boldsymbol{\alpha}$, and $\mathbf{K}_r = [k_r(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^{n,n} = [\boldsymbol{\phi}_k^T(\mathbf{x}_i)\boldsymbol{\phi}_k(\mathbf{x}_j)]_{i,j=1}^{n,n} \in \mathbb{R}^{n\times n}$ is the kth base kernel matrix of the labeled patterns, while the linear combination coefficient vector $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_\infty]^T$ is nonnegative, i.e., $\boldsymbol{\theta} \ge 0$.

The optimization problem in (6.21.14) is an *infinite kernel learning* (IKL) problem with each base kernel as $\mathbf{K}_r$, which can be readily solved with the multiple kernel learning solver.

The ordinary machine learning and support vector machine operate on a single kernel matrix, representing information from a single data source. However, many machine learning problems are often better characterized by incorporating data from more than one source of data. By combining multiple kernel matrices into a single classifier, multiple kernel learning realizes the representation of multiple feature sets, which is conducive to the processing of multi-source data.

Definition 6.55 (Multiple Kernel Learning) *Multiple kernel learning* (MKL) is a machine learning that aims at the unified representation of multiple domain data with multiple kernels and consists of the following three parts.

- Use kernel matrices to serve as the unified representation of multi-source data.
- Transform data under each feature representation to a kernel matrix $\mathbf{K}$.
- M kinds of features will lead to M kernels $\{k_m(\mathbf{x}_i, \mathbf{x}_j)\}_{m=1}^M$. That is, the ensemble kernel for unified representation is the linear combination of base ones given by

$$K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) = \sum_{m=1}^{M} \theta_m k_m(\mathbf{x}_i, \mathbf{x}_j). \tag{6.21.15}$$

The (i, j)th entry of the kernel matrix $\mathbf{K}$ may be the linear kernel function $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$, the polynomial kernel function $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + b)^d, \gamma > 0$, the Gaussian kernel function $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma d^2(\mathbf{x}_i, \mathbf{x}_j))$, and so on. Here, $d(\mathbf{x}_i, \mathbf{x}_j)$ is the dissimilarity or distance between $\mathbf{x}_i$ and $\mathbf{x}_j$.

With the training data $\{\mathbf{x}_i, y_i\}_{i=1}^N$ where $y_i \in \{-1, +1\}$ and base kernels $\{k_m\}_{m=1}^M$, the learned model is of the form

$$\begin{aligned} h(\mathbf{x}) &= \sum_{i=1}^{N} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b \\ &= \sum_{i=1}^{N} \alpha_i y_i \sum_{m=1}^{M} \theta_m k_m(\mathbf{x}_i, \mathbf{x}) + b, \end{aligned} \tag{6.21.16}$$

where linear combination coefficient θ_m is the importance of the mth base kernel $k_m(\mathbf{x}_i, \mathbf{x}_j)$, and b is a learning bias.

Then, task of multiple kernel learning is to optimize both $\{\alpha_i\}_{i=1}^N$ and $\{\theta_m\}_{m=1}^M$ by using all the training data $\{(\mathbf{x}_n, y_n)\}_{n=1}^N = \{(\mathbf{x}_i^{(s)}, y_i^{(s)})\}_{i=1}^{N_s} \cup \{(\mathbf{x}_j^{(t)}, y_j^{(t)})\}_{j=1}^{N_t}$ with $N = N_s + N_t$.

Algorithm 6.47 shows the ℓ_p-norm multiple kernel learning algorithm for interleaved optimization of $\boldsymbol{\theta}$ and $\boldsymbol{\alpha}$ in (6.21.9).

Algorithm 6.48 is the heterogeneous feature augmentation algorithm [162].

6.21.2 Cross-Domain Transform Method

Let $\mathbf{W} : \mathbf{x}^{(t)} \to \mathbf{x}^{(s)}$ denote a linear transformation matrix from the target domain $\mathbf{x}^{(t)} \in \mathcal{D}_T$ to the source domain $\mathbf{x}^{(s)} \in \mathcal{D}_S$. Then, the inner product similarity function between $\mathbf{x}^{(s)}$ and the transformed $\mathbf{W}\mathbf{x}^{(t)}$ can be described as

$$\text{sim}(\mathbf{W}) = \langle \mathbf{x}^{(s)}, \mathbf{W}\mathbf{x}^{(t)} \rangle = (\mathbf{x}^{(s)})^T \mathbf{W}\mathbf{x}^{(t)}. \tag{6.21.17}$$

Algorithm 6.47 ℓ_p-norm multiple kernel learning algorithm [146]

```
1.  input: the accuracy parameter $\epsilon$ and the subproblem size $Q$.
2.  initialization: $g_{m,i} = \hat{g}_i = \alpha_i = 0, \forall i = 1, \ldots, N$; $L = S = -\infty$; $\theta_m = \sqrt[p]{1/M}$,
      $\forall m = 1, \ldots, M$.
3.  iterate
4.     Calculate the gradient $\hat{g}$ of $J(\boldsymbol{\alpha})$ in (6.21.9) w.r.t. $\boldsymbol{\alpha}$;
          $\hat{\mathbf{g}} = \frac{\partial J(\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} = \mathbf{1} - \left(\sum_{m=1}^{M} \theta_m \mathbf{K}_m\right)(\boldsymbol{\alpha} - \mathbf{y})$.
5.     Select Q variables $\alpha_{i1}, \ldots, \alpha_{iQ}$ based on the gradient $\hat{\mathbf{g}}$.
6.     Store $\boldsymbol{\alpha}^{\text{old}} = \boldsymbol{\alpha}$ and then update $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \mu\hat{\mathbf{g}}$.
7.     Update gradient $g_{m,i} \leftarrow g_{m,i} + \sum_{q=1}^{Q} (\alpha_{i_q} - \alpha_{i_q}^{\text{old}}) k_m(x_{i_q}, x_i), \forall m = 1, \ldots, M$;
       $i = 1, \ldots, N$.
8.     Compute the quadratic terms $S_m = \frac{1}{2}\sum_i g_{m,i}\alpha_i,\ q_m = 2\theta_m^2 S_m, \forall m = 1, \ldots, M$.
9.     $L_{\text{old}} = L,\ L = \sum_i y_i\alpha_i,\ S_{\text{old}} = S,\ S = \sum_m \theta_m S_m$.
10.    if $\left|1 - \frac{L-S}{L_{\text{old}} - S_{\text{old}}}\right| \geq \epsilon$ then
          $\theta_m = (q_m)^{1/(p+1)} \Big/ \left(\sum_{m'=1}^{M} (q_{m'})^{p/(p+1)}\right)^{1/p}, \forall m = 1, \ldots, M$.
11.    else
12.       break
13.    end if
14.    $\hat{g}_i = \sum_m \theta_m g_{m,i}, \forall i = 1, \ldots, N$.
15. output: $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_M]^T$ and $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_N]^T$.
```

Algorithm 6.48 Heterogeneous feature augmentation [162]

```
1. input: labeled source samples $\{(\mathbf{x}_i^{(s)}, y_i^{(s)})\}_{i=1}^{N_s}$ and labeled target samples $\{(\mathbf{x}_j^{(t)}, y_j^{(t)})\}_{j=1}^{N_t}$.
2. initialization: $\mathcal{M}_1 = \{\mathbf{M}_1\}$ with $\mathbf{M}_1 = \mathbf{h}_1\mathbf{h}_1^T$ and $\mathbf{h}_1 = \frac{1}{\sqrt{N_s+N_t}}\mathbf{1}_{N_s+N_t}$.
3. repeat
4.    Use Algorithm 6.47 to solve $\boldsymbol{\theta}$ and $\boldsymbol{\alpha}$ in (6.21.9) based on $\mathbf{M}_r$.
5.    Obtain a rank-one PSD matrix $\mathbf{M}_{r+1} = \mathbf{h}\mathbf{h}^T \in \mathbb{R}^{(N_s+N_t)\times(N_s+N_t)}$ with $\mathbf{h} = \frac{\mathbf{K}^{1/2}(\boldsymbol{\alpha}\odot\mathbf{y})}{\|\mathbf{K}^{1/2}(\boldsymbol{\alpha}\odot\mathbf{y})\|}$.
6.    Set $\mathcal{M}_{r+1} = \mathcal{M}_r \cup \{\mathbf{M}_{r+1}\}$, and $r = r + 1$.
7. exit if the objective converges.
8. output: $\boldsymbol{\alpha}$ and $\mathbf{H} = \lambda\sum_{r=1}^{Q}\theta_r\mathbf{M}_r$.
```

The goal of cross-domain transform-based domain adaptation [219] is to learn the linear transformation $\mathbf{W}$ given some form of supervision, and then to utilize the learned similarity function in a classification or clustering algorithm.

Given a set of N points $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \in \mathbb{R}^d$, we seek a positive definite Mahalanobis matrix $\mathbf{W}$ which parameterizes the (squared) Mahalanobis distance:

$$d_W(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^T\mathbf{W}(\mathbf{x}_i - \mathbf{x}_j). \tag{6.21.18}$$

Sample a random pair consisting of a labeled source domain sample $(\mathbf{x}_i^{(s)}, y_i^{(s)})$ and a labeled target-domain sample $(\mathbf{x}_j^{(t)}, y_j^{(t)})$, and then create two constraints described by

$$d_W\left(\mathbf{x}_i^{(s)}, \mathbf{x}_j^{(t)}\right) \leq u, \quad \text{if } y_i^{(s)} = y_j^{(t)}; \tag{6.21.19}$$

$$d_W\left(\mathbf{x}_i^{(s)}, \mathbf{x}_j^{(t)}\right) \geq \ell, \quad \text{if } y_i^{(s)} \neq y_j^{(t)}, \tag{6.21.20}$$

where u is a relatively small value and ℓ is a sufficiently large value. The above two constraints can be rewritten as

$$\mathbf{x}_i^{(s)} \text{ similar to } \mathbf{x}_j^{(t)}, \quad \text{if } d_W\left(\mathbf{x}_i^{(s)}, \mathbf{x}_j^{(t)}\right) \leq u; \tag{6.21.21}$$

$$\mathbf{x}_i^{(s)} \text{ dissimilar to } \mathbf{x}_j^{(t)}, \quad \text{if } d_W\left(\mathbf{x}_i^{(s)}, \mathbf{x}_j^{(t)}\right) \geq \ell. \tag{6.21.22}$$

Learning a Mahalanobis distance can be stated as follows.

1. Given N source feature points $\{\mathbf{x}_1^{(s)}, \ldots, \mathbf{x}_N^{(s)}\} \in \mathbb{R}^d$ and N target feature points $\{\mathbf{x}_1^{(t)}, \ldots, \mathbf{x}_N^{(t)}\} \in \mathbb{R}^d$.
2. Given inequality constraints relating pairs of feature points.
   - Similarity constraints: $d_W(\mathbf{x}_i^{(s)}, \mathbf{x}_j^{(t)}) \leq u$;
   - Dissimilarity constraints: $d_W(\mathbf{x}_i^{(s)}, \mathbf{x}_j^{(t)}) \geq \ell$.
3. Problem: Learn a positive semi-definite Mahalanobis matrix $\mathbf{W}$ such that the Mahalanobis distance $d_W(\mathbf{x}_i^{(s)}, \mathbf{x}_j^{(t)}) = (\mathbf{x}_i^{(s)} - \mathbf{x}_j^{(t)})^T \mathbf{W}(\mathbf{x}_i^{(s)} - \mathbf{x}_j^{(t)})$ satisfies the above similarity and dissimilarity constraints.

This problem is known as a *Mahalanobis metric learning* problem which can be formulated as the constrained optimization problem [219]:

$$\min_{\mathbf{W} \succeq 0} \quad \{\text{tr}(\mathbf{W}) - \log\det(\mathbf{W})\}, \tag{6.21.23}$$

$$\text{subject to} \quad d_W\left(\mathbf{x}_i^{(s)}, \mathbf{x}_j^{(t)}\right) \leq u, \ (i, j) \in S, \tag{6.21.24}$$

$$d_W\left(\mathbf{x}_i^{(s)}, \mathbf{x}_j^{(t)}\right) \geq \ell, \ (i, j) \in D. \tag{6.21.25}$$

Here, the regularizer $\text{tr}(\mathbf{W}) - \log\det(\mathbf{W})$ is defined only between positive semi-definite matrices, and S and D are the set of similar pairs and the set of dissimilar pairs, respectively.

The constrained optimization (6.21.23) provides a Mahalanobis metric learning method and is called *information theoretic metric learning* (ITML), as shown in Algorithm 6.49.

6.21.3 Transfer Component Analysis Method

Let $\mathcal{P}(X_S)$ and $\mathcal{Q}(X_T)$ be the marginal distributions of $X_S = \{\mathbf{x}_{S_i}\}$ and $X_T = \{\mathbf{x}_{T_i}\}$ from the source and target domains, respectively; where $\mathcal{P}$ and $\mathcal{Q}$ are different.

Algorithm 6.49 Information-theoretic metric learning [71]

```
1.  **input:** $d \times n$ matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$, set of similar pairs $S$, set of dissimilar pairs $D$, distance
    threshold $u$ for similarity constraint, distance threshold $\ell$ for dissimilarity constrain,
    Mahalanobis matrix $\mathbf{W}_0$, slack parameter $\gamma$, constraint index function $c(\cdot)$.
2.  **initialization:** $\mathbf{W} \leftarrow \mathbf{W}_0$, $\lambda_{ij} \leftarrow 0$, $\forall i, j$.
3.  Let $\xi_{c(i,j)} = \begin{cases} u, & \text{if } (i,j) \in S; \\ \ell, & \text{otherwise.} \end{cases}$
4.  **repeat**
5.      Pick a constraint $(i, j) \in S$ or $(i, j) \in D$.
6.      Update $p \leftarrow (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{W}(\mathbf{x}_i - \mathbf{x}_j)$.
7.      $\delta = \begin{cases} 1, & \text{if } (i,j) \in S; \\ -1, & \text{otherwise.} \end{cases}$
8.      $\alpha \leftarrow \min\left\{\lambda_{ij}, \frac{\delta}{2}\left(\frac{1}{p} - \gamma \xi_{c(i,j)}\right)\right\}$.
9.      $\beta \leftarrow \delta\alpha/(1 - \delta\alpha p)$.
10.     $\xi_{c(i,j)} \leftarrow \gamma \xi_{c(i,j)}/(\gamma + \delta\alpha\xi_{c(i,j)})$.
11.     $\lambda_{ij} \leftarrow \lambda_{ij} - \alpha$.
12.     $\mathbf{W} \leftarrow \mathbf{W} + \beta \mathbf{W}(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{W}$.
13. **until convergence**
14. **output:** Mahalanobis matrix $\mathbf{W}$.
```

Most domain adaptation methods are based on a key assumption: $\mathcal{P} \neq \mathcal{Q}$, but $P(Y_S|X_S) = P(Y_T|X_T)$. However, in many real-world applications, the conditional probability $P(Y|X)$ may also change across domains due to noisy or dynamic factors underlying the observed data.

Let $\boldsymbol{\phi}$ be a transformation vector such that $\boldsymbol{\phi}(X_S) = [\phi(\mathbf{x}_{S_1}), \cdots, \phi(\mathbf{x}_{S_{n_s}})] \in \mathbb{R}^{1\times n_s}$ and $\boldsymbol{\phi}(X_T) = [\phi(\mathbf{x}_{T_1}), \ldots, \phi(\mathbf{x}_{T_{n_t}})] \in \mathbb{R}^{1\times n_t}$, where n_s and n_t are the numbers of the source and target features. Consider domain adaptation under weak assumption that $\mathcal{P} \neq \mathcal{Q}$ but there exists a transformation vector $\boldsymbol{\phi}$ such that $P(\boldsymbol{\phi}(X_S)) \approx P(\boldsymbol{\phi}(X_T))$ and $P(Y_S|\boldsymbol{\phi}(X_S)) \approx P(Y_T|\boldsymbol{\phi}(X_T))$.

A key problem is how to find this transformation $\boldsymbol{\phi}$. Since there are no labeled data in the target domain, $\boldsymbol{\phi}$ cannot be learned by directly minimizing the distance between $P(Y_S|\boldsymbol{\phi}(X_S))$ and $P(Y_T|\boldsymbol{\phi}(X_T))$. Pan et al. [203] propose to learn $\boldsymbol{\phi}$ such that:

- the distance between the marginal distributions $P(\boldsymbol{\phi}(X_S))$ and $P(\boldsymbol{\phi}(X_T))$ is small,
- $\boldsymbol{\phi}(X_S)$ and $\boldsymbol{\phi}(X_T)$ preserve important properties of X_S and X_T, respectively.

For finding a transformation $\boldsymbol{\phi}$ satisfying $P(Y_S|\boldsymbol{\phi}(X_S)) \approx P(Y_T|\boldsymbol{\phi}(X_T))$, assume that $\boldsymbol{\phi}$ is the feature mapping vector induced by a universal kernel. *Maximum mean discrepancy embedding* (MMDE) [201] embeds both the source and target-domain data into a shared low-dimensional common latent space using a nonlinear mapping $\boldsymbol{\phi}$, which is a symmetric feature transformation with $\phi = T_S = T_T$ shown in Fig. 6.6b.

Define the Gram matrices defined on the source domain (S, S), target domain (T, T), and cross-domain (S, T) in the embedded space as follows:

$$\mathbf{K}_{S,S} = \boldsymbol{\phi}^T(X_S)\boldsymbol{\phi}(X_S) = \begin{bmatrix} \phi(\mathbf{x}_{S_1}) \\ \vdots \\ \phi(\mathbf{x}_{S_{n_s}}) \end{bmatrix} \begin{bmatrix} \phi(\mathbf{x}_{S_1}) \cdots \phi(\mathbf{x}_{S_{n_s}}) \end{bmatrix} \in \mathbb{R}^{n_s \times n_s}, \tag{6.21.26}$$

$$\mathbf{K}_{S,T} = \boldsymbol{\phi}^T(X_S)\boldsymbol{\phi}(X_T) = \begin{bmatrix} \phi(\mathbf{x}_{S_1}) \\ \vdots \\ \phi(\mathbf{x}_{S_{n_s}}) \end{bmatrix} \begin{bmatrix} \phi(\mathbf{x}_{T_1}) \cdots \phi(\mathbf{x}_{T_{n_t}}) \end{bmatrix} \in \mathbb{R}^{n_s \times n_t}, \tag{6.21.27}$$

and

$$\mathbf{K}_{T,S} = \boldsymbol{\phi}^T(X_T)\boldsymbol{\phi}(X_S) = \mathbf{K}_{S,T}^T \in \mathbb{R}^{n_t \times n_s}, \tag{6.21.28}$$

$$\mathbf{K}_{T,T} = \boldsymbol{\phi}^T(X_T)\boldsymbol{\phi}(X_T) = \begin{bmatrix} \phi(\mathbf{x}_{T_1}) \\ \vdots \\ \phi(\mathbf{x}_{T_{n_t}}) \end{bmatrix} \begin{bmatrix} \phi(\mathbf{x}_{T_1}) \cdots \phi(\mathbf{x}_{T_{n_t}}) \end{bmatrix} \in \mathbb{R}^{n_t \times n_t}. \tag{6.21.29}$$

To learn the symmetric kernel matrix

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{S,S} & \mathbf{K}_{S,T} \\ \mathbf{K}_{T,S} & \mathbf{K}_{T,T} \end{bmatrix} \in \mathbb{R}^{(n_s+n_t)\times(n_s+n_t)}, \tag{6.21.30}$$

construct a matrix $\mathbf{L}$ with entries

$$L_{ij} = \begin{cases} 1/n_s^2, & \text{if } \mathbf{x}_i, \mathbf{x}_j \in X_S; \\ 1/n_t^2, & \text{if } \mathbf{x}_i, \mathbf{x}_j \in X_T; \\ -1/(n_s n_t), & \text{otherwise.} \end{cases} \tag{6.21.31}$$

Then, the objective function of MMDE can be written as [203]:

$$\max_{\mathbf{K}\succeq 0} \{\text{tr}(\mathbf{KL}) - \lambda \text{tr}(\mathbf{K})\} \quad \text{subject to constraints on } \mathbf{K}, \tag{6.21.32}$$

where the first term in the objective minimizes the distance between distributions, while the second term maximizes the variance in the feature space, and $\lambda \geq 0$ is a trade-off parameter. However, $\mathbf{K}$ is a high-dimensional kernel matrix due to the fact that n_s and/or n_t are large. As such it is necessary to use a semi-orthogonal transformation matrix $\bar{\mathbf{W}} \in \mathbb{R}^{(n_s+n_t)\times m}$ such that $\bar{\mathbf{W}}\bar{\mathbf{W}}^T = \mathbf{I}_m$ to transform $\mathbf{K}^{-1/2}$

to an m-dimensional matrix $\mathbf{W} = \mathbf{K}^{-1/2}\bar{\mathbf{W}}$ (where $m \ll n_s+n_t$). Then, the distance between distributions, $\text{dist}(X'_S, X'_T) = \text{tr}(\mathbf{KL})$, becomes

$$\text{tr}(\mathbf{KL}) = \text{tr}(\mathbf{KK}^{-1/2}\bar{\mathbf{W}}(\mathbf{K}^{-1/2}\bar{\mathbf{W}})^T\mathbf{KL}) = \text{tr}(\mathbf{KWW}^T\mathbf{KL}) = \text{tr}(\mathbf{W}^T\mathbf{KLKW}).$$

Therefore, the objective function of MMDE in (6.21.32) can be rewritten as [203]:

$$\min_{\mathbf{W}} \left\{\text{tr}(\mathbf{W}^T\mathbf{KLKW}) + \mu\text{tr}(\mathbf{W}^T\mathbf{W})\right\}, \tag{6.21.33}$$

$$\text{subject to} \quad \mathbf{W}^T\mathbf{KHKW} = \mathbf{I}_m, \tag{6.21.34}$$

where $\mathbf{H}$ is the centering matrix given by

$$\mathbf{H} = \mathbf{I}_{n_s+n_t} - \frac{1}{n_t+n_t}\mathbf{1}\mathbf{1}^T. \tag{6.21.35}$$

The constrained optimization problem (6.21.33) and (6.21.34) can be reformulated as the unconstrained optimization problem [203]:

$$\max_{\mathbf{W}} \left\{J(\mathbf{W}) = \text{tr}\left(\left(\mathbf{W}^T(\mathbf{KLK}+\mu\mathbf{I})\mathbf{W}\right)^{-1}\mathbf{W}^T\mathbf{KHKW}\right)\right\}. \tag{6.21.36}$$

Under the condition that the column vectors of $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_m]$ are orthogonal to each other, the above optimization problem can be divided into m subproblems:

$$\begin{aligned}\mathbf{w}_i &= \arg\max_{\mathbf{w}_i} \left\{\text{tr}\left(\left(\mathbf{w}_i^T(\mathbf{KLK}+\mu\mathbf{I})\mathbf{w}_i\right)^{-1}\mathbf{w}_i^T\mathbf{KHK}\mathbf{w}_i\right)\right\}\\ &= \arg\max_{\mathbf{w}_i} \frac{\mathbf{w}_i^T\mathbf{KHK}\mathbf{w}_i}{\mathbf{w}_i^T(\mathbf{KLK}+\mu\mathbf{I})\mathbf{w}_i}, \quad i=1,\ldots,m.\end{aligned} \tag{6.21.37}$$

This is a typical generalized Rayleigh quotient, and thus $\mathbf{w}_i$ is the ith generalized eigenvector corresponding to the ith maximum generalized eigenvalue of the matrix pencil $(\mathbf{KHK}, \mathbf{KLK}+\mu\mathbf{I})$ or the ith eigenvector corresponding to the ith maximum eigenvalue of the matrix $(\mathbf{KLK}+\mu\mathbf{I})^{-1}\mathbf{KLK}$. Moreover, m is the number of leading generalized eigenvalues of $(\mathbf{KHK}, \mathbf{KLK}+\mu\mathbf{I})$ or leading eigenvalues of $(\mathbf{KLK}+\mu\mathbf{I})^{-1}\mathbf{KLK}$. This is the unsupervised *transfer component analysis* (TCA) for domain adaptation, which was proposed by Pan et al. [203].

Transfer learning has been applied to many real-world applications [199, 268].

- Natural language processing [27] including sentiment classification [28], cross-language classification [165], text classification [67], etc.
- Image classification [277], WiFi localization classification [200, 202, 289, 297, 298], video concept detection [82], and so on.

- Pattern recognition including face recognition [139], head pose classification [211], disease prediction [195], atmospheric dust aerosol particle classification [171], sign language recognition [88], image recognition [205], etc.

Brief Summary of This Chapter

- This chapter presents the machine learning tree.
- Machine learning aims at using learning machines to establish the mapping of input data to output data.
- The main tasks of machine learning are classification (for discrete data) and prediction (for continue data).
- This chapter highlights selected topics and advances in machine learning: graph machine learning, reinforcement learning, Q-learning, and transfer learning, etc.

References

1. Acar, E., Camtepe, S.A., Krishnamoorthy, M., Yener, B.: Modeling and multiway analysis of chatroom tensors. In: Proceedings of the IEEE International Conference on Intelligence and Security Informatics, pp. 256–268. Springer, Berlin (2005)
2. Acar, E., Aykut-Bingo, C., Bingo, H., Bro, R., Yener, B.: Multiway analysis of epilepsy tensors. Bioinformatics **23**, i10–i18 (2007)
3. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 207–216 (1992)
4. Ali, M.M., Khompatraporn, C., Zabinsky, Z.B.: A numerical evaluation of several stochastic algorithms on selected continuous global optimization on test problems. J. Global Optim. **31**, 635–672 (2005)
5. Aliu, O.G., Imran, A., Imran, M.A., Evans, B.: A survey of self organisation in future cellular networks. IEEE Commun. Surveys Tutorials. **15**(1), 336–361 (2013)
6. Anderberg, M.R.: Cluster Analysis for Application. Academic, New York (1973)
7. Anderson, T.W.: An Introduction to Multivariate Statistical Analysis, 2nd edn. Wiley, New York (1984)
8. Ando, R.K., Zhang, T.: A framework for learning predictive structures from multiple tasks and unlabeled data. J. Mach. Learn. Res. **6**, 1817–1853 (2005)
9. Angluin D.: Queries and concept learning. Mach. Learn. **2**(4), 319–342 (1988)
10. Arnold, A., Nallapati, R., Cohen, W.W.: A comparative study of methods for transductive transfer learning. In: Proceedings of the Seventh IEEE International Conference on Data Mining Workshops, pp. 77–82 (2007)
11. Atlas, L., Cohn, D., Ladner, R., El-Sharkawi, M.A., Marks II, R.J.: Training connectionist networks with queries and selective sampling. In: Advances in Neural Information Processing Systems 2, Morgan Kaufmann, pp. 566–573 (1990)
12. Auslender, A.: Optimisation Méthodes Numériques. Masson, Paris (1976)
13. Bach, F.R., Jordan, M.I.: Kernel independent component analysis. J. Mach. Learn. Res. **3**, 1–48 (2002)
14. Bagheri, M., Nurmanova, V., Abedinia, O., Naderi, M.S.: Enhancing power quality in microgrids with a new online control Strategy for DSTATCOM using reinforcement learning algorithm. IEEE Access **6**, 38986–38996 (2018)

15. Bandyopdhyay, S., Maulik, U.: An evolutionary technique based on K-means algorithm for optimal clustering in $\mathbb{R}^N$. Inform. Sci. **146**(1–4), 221–237 (2002)
16. Bartlett, P.L.: The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. IEEE Trans. Inf. Theory. **44**(2), 525–536 (1998)
17. Baum, L.E., Eagon, J.A.: An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. Bull. Amer. Math. Soc. **73**(3), 360 (1967)
18. Behbood, V., Lu, J., Zhang, G.: Fuzzy bridged refinement domain adaptation: long-term bank failure prediction. Int. J. Comput Intell. Appl. **12**(1), Art. no. 1350003 (2013)
19. Belkin, M., Niyogi, P.: Laplacian eigenmaps for dimensionality reduction and data representation. Neural Comput. **15**(6), 1373–1396 (2003)
20. Belkin, M., Niyogi, P., Sindhwani, V.: Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. J. Mach. Learn. Res. **7**, 2399–2434 (2006)
21. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton (1957)
22. Bengio, Y., Courville, A., Vincent, P.: Representation learning: a review and new perspectives. IEEE Trans. Pattern Anal. Mach. Intell. **35**(8), 1798–1828 (2013)
23. Bersini, H., Dorigo, M., Langerman, S.: Results of the first international contest on evolutionary optimization. In: Proceedings of IEEE International Conference on Evolutionary Computation, Nagoya, pp. 611–615 (1996)
24. Bertsekas, D.P.: Dynamic Programming and Optimal Sequence of States of the Markov Decision Process. Control, vol. 11. Athena Scientific, Nashua (1995)
25. Bertsekas, D.P.: Nonlinear Programming, 2nd edn. Athena Scientific, Nashua (1999)
26. Beyer, H.G., Schwefel, H.P.: Evolution strategies: a comprehensive introduction. J. Nat. Comput. **1**(1), 3–52 (2002)
27. Blitzer, J., McDonald, R., Pereira, F.: Domain adaptation with structural correspondence learning. In: Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, pp. 120–128 (2006)
28. Blitzer, J., Dredze, M., Pereira, F.: Biographies, Bollywood, Boom-Boxes and Blenders: Domain adaptation for sentiment classification. In: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pp. 432–439 (2007)
29. Blum, A., Chawla, S.: Learning from labeled and unlabeled data using graph mincuts. In: Proceedings of the 18th International Conference on Machine Learning (2001)
30. Blum, A., Mitchell, T.: Combining labeled and unlabeled data with co-training. In: Proceedings of the 11th Annual Conference on Computational Learning Theorem (COLT 98), pp. 92–100 (1998)
31. Bottou, L., Curtis, F.E., Nocedal, J.: Optimization methods for large-scale machine learning. SIAM Rev. **60**(2), 223–311 (2018)
32. Bouneffouf, D.: Exponentiated gradient exploration for active learning. Computers **5**(1), 1–12 (2016)
33. Bouneffouf, D., Laroche, R., Urvoy, T., Fèraud, R., Allesiardo, R.: Contextual bandit for active learning: Active Thompson sampling. In: Proceedings of the 21st International Conference on Neural Information Processing, ICONIP (2014)
34. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. IEEE Trans. Pattern Analy. Mach. Intell. **26**(9), 1124–1137 (2004)
35. Breiman, L.: Better subset selection using the nonnegative garrote. Technometrics **37**, 738–754 (1995)
36. Bro, R.: PARAFAC: tutorial and applications. Chemome. Intell. Lab. Syst. **38**, 149–171 (1997)
37. Bu, F.: A high-order clustering algorithm based on dropout deep learning for heterogeneous data in Cyber-Physical-Social systems. IEEE Access **6**, 11687–11693 (2018)
38. Buczak, A.L., Guven, E.: A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Commun. Surv. Tut. **18**(2), 1153–1176 (2016)

39. Burges, C.J.C.: A tutorial on support vector machines for pattern recognition. Data Min. Knowl. Disc. **2**, 121–167 (1998)
40. Burr, S.: Active Learning Literature Survey. Computer Sciences Technical Report 1648, University of Wisconsin-Madison, Retrieved 2014-11-18 (2010)
41. Cai, D., Zhang, C., He, S.: Unsupervised feature selection for multi-cluster data. In: Proceedings of the 16th ACM SIGKDD, July 25–28, Washington, pp. 333–342 (2010)
42. Campbell, C., Cristianini, N., Smola, A.: Query learning with large margin classifiers. In: Proceedings of the International Conference on Machine Learning (ICML) (2000)
43. Candès, E.J., Wakin, M.B., Boyd, S.P.: Enhancing sparsity by reweighted ℓ_1 minimization. J. Fourier Analy. Appl. **14**(5–6), 877–905 (2008)
44. Candès, E.J., Li, X., Ma, Y., Wright, J.: Robust principal component analysis? J. ACM **58**(3), 1–37 (2011)
45. Caruana, R.A.: Multitask learning. Mach. Learn. **28**, 41–75 (1997)
46. Chandrasekaran, V., Sanghavi, S., Parrilo, P.A., Wilisky, A.S.: Rank-sparsity incoherence for matrix decomposition. SIAM J. Optim. **21**(2), 572–596 (2011)
47. Chang, C.I., Du, Q.: Estimation of number of spectrally distinct signal sources in hyperspectral imagery. IEEE Trans. Geosci. Remote Sens. **42**(3), 608–619 (2004)
48. Chattopadhyay, R., Sun, Q., Fan, W., Davidson, I., Panchanathan, S., Ye, J.: Multisource domain adaptation and its application to early detection of fatigue. ACM Trans. Knowl. Discov. From Data **6**(4), 1–26 (2012)
49. Chen, T., Amari, S., Lin, Q.: A unified algorithm for principal and minor components extraction. Neural Netw. **11**, 385–390 (1998)
50. Chen, Y., Lasko, T.A., Mei, Q., Denny, J.C, Xu, H.: A study of active learning methods for named entity recognition in clinical text. J. Biomed. Inform. **58**, 11–18 (2015)
51. Chernoff, H.: Sequential analysis and optimal design. In: CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 8. SIAM, Philadelphia (1972)
52. Choromanska, A., Jebara, T., Kim, H., Mohan, M., Monteleoni, C.: Fast spectral clustering via the Nyström method. In: International Conference on Algorithmic Learning Theory ALT 2013, pp. 367–381 (2013)
53. Chung, F.R.K.: Spectral graph theory. In: CBMS Regional Conference Series, vol.92. Conference Board of the Mathematical Sciences, Washington (1997)
54. Chung, C.J., Reynolds, R.G.: CAEP: An evolution-based tool for real-valued function optimization using cultural algorithms. Int. J. Artif. Intell. Tool **7**(3), 239–291 (1998)
55. Ciresan, D.C., Meier, U., Schmidhuber, J.: Transfer learning for Latin and Chinese characters with deep neural networks. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN), Brisbane, pp. 1–6 (2012)
56. Coates, A., Ng, A.Y.: Learning feature representations with K-means. In: Montavon, G., Orr, G.B., Müller, K.-R. (eds.) Neural Networks: Tricks of the Trade, 2nd edn., pp. 561–580. Springer, Berlin (2012)
57. Cohen, W.W.: Fast effective rule induction. In: Proceedings of the 12th International Conference on International Conference on Machine Learning, Lake Tahoe, pp. 115–123 (1995)
58. Cohn, D.: Active learning. In: Sammut, C., Webb, G.I. (eds.) Encyclopedia of Machine Learning, pp. 10–14 (2011)
59. Cohn, D., Ghahramani, Z., Jordan, M.I.: Active learning with statistical models. J. Artific. Intell. Res. **4**, 129–145 (1996)
60. Comon, P., Golub, G., Lim, L.H., Mourrain, B.: Symmetric tensors and symmetric tensor rank. SIAM J. Matrix Anal. Appl. **30**(3), 1254–1279 (2008)
61. Corana, A., Marchesi, M., Martini, C., Ridella, S.: Minimizing multimodal functions of continuous variables with simulated annealing algorithms. ACM Trans. Math. Softw. **13**(3), 262–280 (1987)
62. Correa, N.M., Adali, T., Li, Y.Q., Calhoun, V.D.: Canonical correlation analysis for data fusion and group inferences. IEEE Signal Proc. Mag. **27**(4), 39–50 (2010)

63. Cortes, C., Mohri, M.: On transductive regression. In: Proceedings of the Neural Information Processing Systems (NIPS), pp. 305–312 (2006)
64. Crammer, K., Singer, Y.: On the algorithmic implementation of multiclass kernel-based vector machines. J. Mach. Learn. Res. **2**, 265–292 (2001)
65. Cristianini, N., Shawe-Taylor, J., Elisseeff, A., Kandola, J.S.: On kernel-target alignment. In: NIPS'01 Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, pp. 367–373 (2001)
66. Dai, W., Yang, Q., Xue, G.R., Yu, Y.: Boosting for transfer learning. In: Proceedings of the 24th International Conference on Machine Learning, pp. 193–200 (2007)
67. Dai, W., Xue, G., Yang, Q., Yu, Y.: Transferring naive Bayes classifiers for text classification. In: Proc. 22nd Association for the Advancement of Artificial Intelligence (AAAI) Conference on Artificial Intelligence, pp. 540–545 (2007)
68. Dai, W., Jin, O., Xue, G.-R., Yang, Q., Yu, Y.: EigenTransfer: A unified framework for transfer learning. In: Proceedings of the the 26th International Conference on Machine Learning, Montreal, pp. 193–200 (2009)
69. Dash, M., Liu, H.: Feature selection for classification. Intell. Data Anal. **1**(1–4), 131–156 (1997)
70. Daumé III, H.: Frustratingly easy domain adaptation. In: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pp. 256–263 (2007)
71. Davis, J.V., Kulis, B., Jain, P., Sra, S., Dhillon, I.S.: Information-theoretic metric learning. In: Proceedings of the international Conference on Machine Learning, pp. 209–216 (2007)
72. Defazio, A., Bach, F., Lacoste-Julien, S.: SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In: Advances in Neural Information Processing Systems, vol. 27, pp. 1646–1654 (2014)
73. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Proceedings of the 30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, pp. 3837–3845 (2016)
74. Deng, Z., Choi, K., Jiang, Y.: Generalized hidden-mapping ridge regression, knowledge-leveraged inductive transfer learning for neural networks, fuzzy systems and kernel method. IEEE Trans. Cybern. **44**(12), 2585–2599 (2014)
75. Dhillon, I.S., Modha, D.M.: Concept decompositions for large sparse text data using clustering. Mach. Learn. **42**(1), 143–175 (2001)
76. Dong, X., Thanou, D., Frossard, P., Vandergheynst, P.: Learning Laplacian matrix in smooth graph signal representations. IEEE Trans. Sign. Proc. **64**(23), 6160–6173 (2016)
77. Donoho, D.L., Johnstone, I.: Adapting to unknown smoothness via wavelet shrinkage. J. Amer. Statist. Assoc. **90**, 1200–1224 (1995)
78. Dorigo, M., Gambardella, L.M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Trans. Evol. Comput. **1**(1), 53–66 (1997)
79. Douglas, S.C., Kung, S.-Y., Amari, S.: A self-stabilized minor subspace rule. IEEE Sign. Proc. Lett. **5**(12), 328–330 (1998)
80. Downie, J.S.: A window into music information retrieval research. Acoust. Sci. Technol. **29**(4), 247–255 (2008)
81. Du, Q., Faber, V., Gunzburger, M.: Centroidal Voronoi tessellations: applications and algorithms. SIAM Rev. **41**, 637–676 (1999)
82. Duan, L., Tsang, I.W., Xu, D., Maybank, S.J.: Domain transfer SVM for video concept detection. In: Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1375–1381 (2009)

83. Duda, R.O., Hart, P.E.: Pattern Classification and Scene Analysis. Wiley, New York (1973)
84. Efron, B., Hastie, T., Johnstone, I., Tibshirani, R.: Least angle regression. Ann. Statist. **32**, 407–499 (2004)
85. El-Attar, R.A., Vidyasagar, M., Dutta, S.R.K.: An algorithm for II-norm minimization with application to nonlinear II-approximation. SIAM J. Numer. Anal. **16**(1), 70–86 (1979)
86. Estienne, F., Matthijs, N., Massart, D.L., Ricoux, P., Leibovici, D.: Multi-way modeling of high-dimensionality electroencephalographic data. Chemometr. Intell. Lab. Syst. **58**(1), 59–72 (2001)
87. Fan, J., Han, F., Liu, H.: Challenges of big data analysis. Nat. Sci. Rev. **1**(2), 293–314 (2014)
88. Farhadi, A., Forsyth, D., White, R.: Transfer learning in sign language. In: Proceedings of the IEEE 2007 Conference on Computer Vision and Pattern Recognition, pp. 1–8 (2007)
89. Farmer, J., Packard, N., Perelson, A.: The immune system, adaptation and machine learning. Phys. D: Nonlinear Phenom. **2**, 187–204 (1986)
90. Fedorov, V.V.: Theory of Optimal Experiments. (Trans. by Studden, W.J., Klimko, E.M.). Academic, New York (1972)
91. Fercoq, O., Richtárk, P.: Accelerated, parallel, and proximal coordinate descent. SIAM J. Optim. **25**(4), 1997–2023 (2015)
92. Figueiredo, M.A.T., Nowak, R.D., Wright, S.J.: Gradient projection for sparse reconstruction: application to compressed sensing and other inverse problems. IEEE J. Sel. Top. Signa. Proc. **1**(4), 586–597 (2007)
93. Finkel, J.R., Manning, C.D.: Hierarchical Bayesian domain adaptation. In: Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics, Los Angeles, pp. 602–610 (2009)
94. Fisher, R.A.: The statistical utilization of multiple measurements. Ann. Eugenic. **8**, 376–386 (1938)
95. Ford, L., Fulkerson, D.: Flows in Networks. Princeton University Press, Princeton (1962)
96. Freund, Y.: Boosting a weak learning algorithm by majority. Inform. Comput. **12**(2), 256–285 (1995)
97. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci. **55**, 119–139 (1997)
98. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Mach. Learn. **29**, 131–163 (1997)
99. Friedman, J., Hastie, T., Höeling, H., Tibshirani, R.: Pathwise coordinate optimization. Ann. Appl. Stat. **1**(2), 302–332 (2007)
100. Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. Ann. Stat. **28**(2), 337–407 (2000)
101. Fu, W.J.: Penalized regressions: the bridge versus the Lasso. J. Comput. Graph. Stat. **7**(3), 397–416 (1998)
102. Fuchs, J.J.: Multipath time-delay detection and estimation. IEEE Trans. Signal Process. **47**(1), 237–243 (1999)
103. Furey, T.S., Cristianini, N., Duffy, N., Bednarski, D.W., Schummer, M., Haussler, D.: Support vector machine classification and validation of cancer tissue samples using microarray expression data. Bioinformatics **16**(10), 906–914 (2000)
104. Ge, Z., Song, Z., Ding, S.X., Huang, B.: Data mining and analytics in the process industry: the role of machine learning. IEEE Access **5**, 20590–20616 (2017)
105. Geladi, P., Kowalski, B.R.: Partial least squares regression: a tutorial. Anal. Chim. Acta **186**, l–17 (1986)
106. George, A.P., Powell, W.B.: Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. Mach. Learn. **65**(1), 167–198 (2006)
107. Goldberg, D.E., Holland, J.H.: Genetic algorithms and machine learning. Mach. Learn. **3**(2), 95–99 (1988)
108. Golub, G.H., Zha, H.: The canonical correlations of matrix pairs and their numerical computation. In: Linear Algebra for Signal Processing, pp. 27–49. Springer, Berlin (1995)

109. Golub, T.R., Slonim, D.K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J.P., Coller, H., Loh, M.L., Downing, J.R., Caligiuri, M.A., Bloomfield, C.D., Lander, E.S.: Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. Science **286**, 531–537 (1999)
110. Grandvalet, Y., Bengio, Y.: Semi-supervised learning by entropy minimization. In: Advances in Neural Information Processing Systems, vol. 17, pp. 529–536 (2005)
111. Guo, W., Kotsia, I., Ioannis, P.: Tensor learning for regression. IEEE Trans. Image Process. **21**(2), 816–827 (2012)
112. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**, 1157–1182 (2003)
113. Guyon, I., Weston, J., Barnhill, S., Vapnik, V.: Gene selection for cancer classification using support vector machines. Mach. Learn. **46**, 389–422 (2002)
114. Handl, J., Knowles, J., Kell, D.B.: Computational cluster validation in post-genomic data analysis. Bioinformatics **21**(15), 3201–3212 (2005)
115. Hardoon, D.R., Szedmak, S., Shawe-Taylor, J.: Canonical correlation analysis: an overview with application to learning methods. Neural Comput. **16**(12), 2639–2664 (2004)
116. Hesterberg, T., Choi, N.H., Meier, L., Fraley, C.: Least angle and ℓ_1 penalized regression: a review. Stat. Surv. **2**, 61–93 (2008)
117. Hoerl, A.E., Kennard, R.W.: Ridge regression: biased estimates for non-orthogonal problems. Technometrics **12**, 55–67 (1970)
118. Hoerl, A.E., Kennard, R.W.: Ridge regression: applications to nonorthogonal problems. Technometrics **12**, 69–82 (1970)
119. Hoi, S.C.H., Jin, R., Lyu, M.R.: Batch mode active learning with applications to text categorization and image retrieval. IEEE Trans. Knowl. Data Eng. **21**(9), 1233–1247 (2009)
120. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**(5), 359–366 (1989)
121. Höskuldsson, A.: PLS regression methods. J. Chemometr. **2**, 211–228 (1988)
122. Hotelling, H.: Relations between two sets of variants. Biometrika **28**(3/4), 321–377 (1936)
123. Huang, G.-B., Zhu, Q.-Y., Siew, C.-K.: Extreme learning machine: theory and applications. Neurocomputing **70**(1–3), 489–501 (2006)
124. Huang, G.-B., Zhou, H., Ding, X., Zhang, R.: Extreme learning machine for regression and multiclass classification. IEEE Trans. Syst. Man Cybern. B Cybern. **42**(2), 513–529 (2012)
125. Hunter, D.R., Lange, K.: A tutorial on MM algorithms. Amer. Statist. **58**, 30–37 (2004)
126. Jain, K., Dubes, R.C.: Algorithms for Clustering Data. Prentice-Hall, Englewood Cliffs (1988)
127. Jain, A.K.: Data clustering: 50 years beyond K-means. Pattern Recogn. Lett. **31**, 651–666 (2010)
128. Jain, A.K., Duin, R.P.W., Mao, J.: Statistical pattern recognition: a review. IEEE Trans. Pattern Anal. Mach. Intell. **22**(1), 4–37 (2000)
129. Jamil, M., Yang, X.-S.: A literature survey of benchmark functions for global optimization problems. Int. J. Math. Modell. Numer. Optim. **4**(2), 150–194 (2013)
130. Jensen, F.V.: Bayesian Networks and Decision Graphs. Springer, New York (2001)
131. Joachims, T.: Transductive inference for text classification using support vector machines. In: Proceedings of the 16th International Conference on Machine Learning, pp. 200–209 (1999)
132. Johnson, S.C.: Hierarchical clustering schemes. Psycioietrika **32**(3), 241–254 (1967)
133. Johnson, R., Zhang, T.: Accelerating stochastic gradient descent using predictive variance reduction. In: Advances in Neural Information Processing Systems, vol. 26, pp. 315–323 (2013)
134. Jolliffe, I.: Principal Component Analysis. Springer, New York (1986)
135. Jonesb, S., Shaoa, L., Dub, K.: Active learning for human action retrieval using query pool selection. Neurocomputing **124**, 89–96 (2014)
136. Jouffe, L.: Fuzzy inference system learning by reinforcement methods. IEEE Trans. Syst. Man Cybern. Part C **28**(3), 338–355 (1998)

137. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. J. Artif. Intell. Res. **4**, 237–285 (1996)
138. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artif. Intell. **101**(1), 99–134 (1998)
139. Kan, M., Wu, J., Shan, S., Chen, X.: Domain adaptation for face recognition: targetize source domain bridged by common subspace. Int. J. Comput. Vis. **109**(1–2), 94–109 (2014)
140. Kearns, M., Valiant, L.: Crytographic limitations on learning Boolean formulae and finite automata. In: Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing, pp. 433–444 (1989); See J. ACM **41**(1), 67–95 (1994)
141. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. MIT Press, Cambridge (1994)
142. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks (ICNN), vol. IV, pp. 1942–1948 (1995)
143. Kiers, H.A.L.: Towards a standardized notation and terminology in multiway analysis. J. Chemometr. **14**, 105–122 (2000)
144. Kimura, A., Kameoka, H., Sugiyama, M., Nakano, T., Maeda, E., Sakano, H., Ishiguro, K.: SemiCCA: Efficient semi-supervised learning of canonical correlations. Inform. Media Technol. **8**(2), 311–318 (2013)
145. Klaine, P.V., Imran, M.A., Souza, R.D., Onireti, O.: A survey of machine learning techniques applied to self-organizing cellular networks. IEEE Commun. Surv. Tut. **19**(4), 2392–2431 (2017)
146. Kloft, M., Brefeld, U., Sonnenburg, S., and Zien, A.: ℓ_p-norm multiple kernel learning. J. Mach. Learn. Res. **12**, 953–997 (2011)
147. Kober, J., Bangell, J., Peters, J.: Reinforcement learning in robotics: a survey. Int. J. Robustics Res. **32**(11), 1238–1274 (2013)
148. Kocer, B., Arslan, A.: Genetic transfer learning. Expert Syst. Appl. **37**(10), 6997–7002 (2010)
149. Kolda, T.G.: Multilinear operators for higher-order decompositions. Sandia Report SAND2006-2081, California (2006)
150. Kolda, T.G., Bader, B.W., Kenny, J.P.: Higher-order web link analysis using multilinear algebra. In: Proceedings of the 5th IEEE International Conference on Data Mining, pp. 242–249 (2005)
151. Konečný J., Liu, J., Richtárik, P., Takáč, M.: Mini-batch semi-stochastic gradient descent in the proximal setting. IEEE J. Sel. Top. Signa. Process. **10**(2), 242–255 (2016)
152. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
153. Kulis, B., Saenko, K., Darrell, T.: What you saw is not what you get: domain adaptation using asymmetric kernel transforms. In: Proceedings of the IEEE 2011 Conference on Computer Vision and Pattern Recognition, pp. 1785–1292 (2011)
154. Lathauwer, L.D., Moor, B.D., Vandewalle, J.: A multilinear singular value decomposition. SIAM J. Matrix Anal. Appl. **21**, 1253–1278 (2000)
155. Lathauwer, L.D., Nion, D.: Decompositions of a higher-order tensor in block terms—part III: alternating least squares algorithms. SIAM J. Matrix Anal. Appl. **30**(3), 1067–1083 (2008)
156. Le Roux, N., Schmidt, M., Bach, F.R.: A stochastic gradient method with an exponential convergence rate for finite training sets. In: Advances in Neural Information Processing Systems, vol. 25, pp. 2663–2671 (2012)
157. Letexier, D., Bourennane, S., Blanc-Talon, J.: Nonorthogonal tensor matricization for hyperspectral image filtering. IEEE Geosci. Remote Sensing. Lett. **5**(1), 3–7 (2008)
158. Levie, R., Monti, F., Bresson, X., Bronstein, M.M.: CayleyNets: Graph convolutional neural networks with complex rational spectral filters (2018). Available at: https://arXiv:1705.07664v2
159. Lewis, D., Gale, W.: A sequential algorithm for training text classifiers. In Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 3–12. ACM/Springer, New York/Berlin (1994)

160. Li, X., Guo, Y.: Adaptive active learning for image classification. In: Proceedings of the 26th IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8 (2013)
161. Li, F., Pan, S.J., Jin, O., Yang, Q., Zhu, X.: Cross-domain co-extraction of sentiment and topic lexicons. In: Proceedings of the 50th annual meeting of the association for computational linguistics long papers, vol. 1, pp. 410–419 (2012)
162. Li, W., Duan, L., Xu, D., Tsang, I.: Learning with augmented features for supervised and semi-supervised heterogeneous domain adaptation. IEEE Trans. Pattern Anal. Mach. Intell. **36**(6), 1134–1148 (2014)
163. Lin, L.: Self-improving reactive agents based on reinforcement learning, planning and teaching. Mach. Learn. **8**, 293–321 (1992)
164. Lin, Z., Chen, M., Ma, Y.: The augmented Lagrange multiplier method for exact recovery of corrupted low-rank matrices. Technical Report UILU-ENG-09-2215 (2009)
165. Ling, X., G.-R. Xue, G. -R., Dai, W., Jiang, Y., Yang, Q., Yu, Y.: Can Chinese Web pages be classified with English data source? In: Proceedings of the 17th International Conference on World Wide Web, pp. 969–978 (2008)
166. Liu, J., Wright, S.J., Re, C., Bittorf, V., Sridhar, S.: An asynchronous parallel stochastic coordinate descent algorithm. J. Mach. Learn. Res., **16**, 285–322 (2015)
167. Lu, J., Behbood, V., Hao, P., Zuo, H., Xue, S., Zhang, G.: Transfer learning using computational intelligence: a survey. Knowl. Based Syst. **80**, 14–23 (2015)
168. Luis, R., Sucar, L.E., Morales, E.F.: Inductive transfer for learning Bayesian networks. Mach. Learn. **79**(1–2), 227–255 (2010)
169. Luo, F.L., Unbehauen, R., Cichock, R.: A minor component analysis algorithm. Neural Netw. **10**(2), 291–297 (1997)
170. Ma, Y., Luo, G., Zeng, X., Chen, A.: Transfer learning for cross-company software defect prediction. Inform. Softw. Technol. **54**(3), 248–256 (2012)
171. Ma, Y., Gong, W., Mao, F.: Transfer learning used to analyze the dynamic evolution of the dust aerosol. J. Quant. Spectrosc. Radiat. Transf. **153**, 119–130 (2015)
172. Mahalanobis, P.C.: On the generalised distance in statistics. Proc. Natl. Inst. Sci. India **2**(1), 49–55 (1936)
173. Maier, M., von Luxburg, U., Hein, M.: How the result of graph clustering methods depends on the construction of the graph. ESAIM: Probab. Stat. **17**, 370–418 (2013)
174. Masci, J., Meier, U., Ciresan, D., Schmidhuber, J.: Stacked convolutional auto-encoders for hierarchical feature extraction. In: Proceedings of the 21st International Conference on Artificial Neural Networks, Part I, Espoo, pp. 52–59 (2011)
175. Massy, W.F.: Principal components regression in exploratory statistical research. J. Am. Stat. Assoc. **60**(309), 234–256 (1965)
176. McCallum, A., Nigam, K.: Employing EM and pool-based active learning for text classification. In: ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning, pp. 359–367 (1998)
177. Michalski, R.: A theory and methodology of inductive learning. Mach. Learn. **1**, 83–134 (1983)
178. Miller, G.A., Nicely, P.E.: An analysis of perceptual confusions among some English consonants. J. Acoust. Soc. Am. **27**, 338–352 (1955)
179. Mishra, S.K.: Global optimization by differential evolution and particle swarm methods: Evaluation on some benchmark functions. Munich Research Papers in Economics (2006). Available at: https://mpra.ub.uni-muenchen.de/1005/
180. Mishra, S.K.: Performance of differential evolution and particle swarm methods on some relatively Harder multi-modal benchmark functions (2006). Available at: https://mpra.ub.uni-muenchen.de/449/
181. Mitchell, T.M.: Machine Learning, vol. 45. McGraw Hill, Burr Ridge (1997)
182. Mitra, P., Murthu, C.A., Pal, S.K.: Unsupervised feature selection using feature similarity. IEEE Trans. Pattern Anal. Mach. Intell. **24**(3), 301–312 (2002)
183. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)

184. Mohar, B.: Some applications of Laplace eigenvalues of graphs. In: Hahn, G., Sabidussi, G. (eds.) Graph Symmetry: Algebraic Methods and Applications. NATO Science Series C, vol.497, pp. 225–275. Kluwer, Dordrecht (1997)
185. Moulton, C.M., Roberts, S.A., Calatn, P.H.: Hierarchical clustering of multiobjective optimization results to inform land-use decision making. URISA J. **21**(2), 25–38 (2009)
186. Murthy, C.A., Chowdhury, N.: In search of optimal clusters using genetic algorithms. Pattern Recog. Lett. **17**, 825–832 (1996)
187. Narayanan, H., Belkin, M., Niyogi, P.: On the relation between low density separation, spectral clustering and graph cuts. In: Schölkopf, B., Platt, J., Hoffman, T. (eds.) Advances in Neural Information Processing Systems, vol. 19, pp. 1025–1032. MIT Press, Cambridge (2007)
188. Nesterov, Y.: Efficiency of coordinate descent methods on huge-scale optimization problems. SIAM J. Optim. **22**(2), 341–362 (2012)
189. Ng, V., Vardie, C.: Weakly supervised natural language learning without redundant views. In: Proceedings of the Human Language Technology/Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL), Main Papers, pp. 94–101 (2003)
190. Ng, A., Jordan, M., Weiss, Y.: On spectral clustering: analysis and an algorithm. In: Dietterich, T., Becker, S., Ghahramani, Z. (eds.) Advances in Neural Information Processing Systems, vol. 14, pp. 849–856. MIT Press, Cambridge (2002)
191. Nguyen, H.D.: An introduction to Majorization-minimization algorithms for machine learning and statistical estimation. WIREs Data Min. Knowl. Discovery **7**(2), e1198 (2017)
192. Niculescu-Mizil, A., Caruana, R.: Inductive transfer for Bayesian network structure learning. In: Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS), San Juan (2007)
193. Nigam, K., Ghani, R.: Analyzing the effectiveness and applicability of co-training. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM), pp. 86–93 (2000)
194. Oja, E., Karhunen, J.: On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. J. Math Anal. Appl. **106**, 69–84 (1985)
195. Ogoe, H.A., Visweswaran, S., Lu, X., Gopalakrishnan, V.: Knowledge transfer via classification rules using functional mapping for integrative modeling of gene expression data. BMC Bioinform. **16**, 1–15 (2015)
196. Oquab, M., Bottou, L., Laptev, I.: Learning and transferring mid-level image representations using convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1717–1724 (2014)
197. Ortega, J.M., Rheinboldt, W.C.: Iterative Solutions of Nonlinear Equations in Several Variables, pp. 253–255. Academic, New York (1970)
198. Owen, A.B.: A robust hybrid of lasso and ridge regression. Prediction and Discovery (Contemp. Math.), **443**, 59–71 (2007)
199. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Trans. Knowl. Data Eng. **22**(10), 1345–1359 (2010)
200. Pan, S.J., Kwok, J.T., Yang, Q., Pan, J.J.: Adaptive localization in a dynamic WiFi environment through multi-view learning. In: Proceedings of the 22nd Association for the Advancement of Artificial Intelligence (AAAI) Conference Artificial Intelligence, pp. 1108–1113 (2007)
201. Pan, S.J., Kwok, J.T., Yang, Q.: Transfer learning via dimensionality reduction. In: Proceedings of the 23rd National Conference on Artificial Intelligence, vol. 2, pp. 677–682 (2008)
202. Pan, S.J., Shen, D., Yang, Q., Kwok, J.T.: Transferring localization models across space. In: Proceedings of the 23rd Association for the Advancement of Artificial Intelligence (AAAI) Conference on Artificial Intelligence, pp. 1383–1388 (2008)
203. Pan, S.J., Tsang, I.W., Kwok, J.T, Yang, Q.: Domain adaptation via transfer component analysis. IEEE Trans. Neural Netw. **22**(2), 199–210 (2011)

204. Parra, L., Spence, C., Sajda, P., Ziehe, A., Muller, K.: Unmixing hyperspectral data. In: Advances in Neural Information Processing Systems, vol. 12, pp. 942–948. MIT Press, Cambridge (2000)
205. Patel, V.M, Gopalan, R., Li, R., Chellappa, R.: Visual domain adaptation: a survey of recent advances. IEEE Signal Process. Mag. **32**(3), 53–69 (2015)
206. Polikar, R.: Ensemble based systems in decision making. IEEE Circ. Syst. Mag. **6**(3), 21–45 (2006)
207. Prettenhofer, P., Stein, B.: Cross-language text classification using structural correspondence learning. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pp. 1118–1127 (2010)
208. Price, W.L.: A controlled random search procedure for global optimisation. Comput. J. **20**(4), 367–370 (1977)
209. Rahnamayan, S., Tizhoosh, H.R., Salama, N.M.M.: Opposition-based differential evolution. IEEE Trans. Evol. Comput. **12**(1), 64–79 (2008)
210. Raina, R., Battle, A., Lee, H., Packer, B., Ng, A.Y.: Self-taught learning: Transfer learning from unlabeled data. In: Proceedings of the 24th International Conference on Machine Learning, Corvallis, pp. 759–766 (2007)
211. Rajagopal, A.N., Subramanian, R., Ricci, E., Vieriu, R.L., Lanz, O., Ramak-rishnan, K.R., Sebe, N.: Exploring transfer learning approaches for head pose classification from multi-view surveillance images. Int. J. Comput. Vis. **109**(1–2), 146–167 (2014)
212. Richtárik, P., Takáč M.: Parallel coordinate descent methods for big data optimization. Math. Program. Ser. A **156**, 433–484 (2016)
213. Rivli, J.: An Introduction to the Approximation of Functions. Courier Dover Publications, New York (1969)
214. Robbins, H., Monro, S.: A stochastic approximation method. Ann. Math. Stat. **22**, 400–407 (1951)
215. Rosipal, R., Krämer, N.: Overview and recent advances in partial least squares. In: Proceedings of the Workshop on Subspace, Latent Structure and Feature Selection (SLSFS) 2005, pp. 34–51 (2006)
216. Roweis, S., Saul, L.: Nonlinear dimensionality reduction by locally linear embedding. Science **290**(5500), 2323–2326 (2000)
217. Roy, D.M., Kaelbling, L.P.: Efficient Bayesian task-level transfer learning. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, pp. 2599–2604 (2007)
218. Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University (1994)
219. Saenko, K., Kulis, B., Fritz, M., Darrell, T.: Adapting visual category models to new domains. In: Proceedings of the European Conference on Computer Vision, vol. 6314, pp. 213–226 (2010)
220. Schaal, S.: Is imitation learning the route to humanoid robots? Trends Cogn. Sci. **3**(6), 233–242 (1999)
221. Schapire, R.E.: The strength of weak learnability. Mach. Learn. **5**, 197–227 (1990)
222. Schmidt, M., Le Roux, N., Bach, F.: Minimizing finite sums with the stochastic average gradient. Technical Report, INRIA, hal-0086005 (2013). See also Math. Program. **162**, 83–112 (2017)
223. Schwefel, H.P.: Numerical Optimization of Computer Models. Wiley, Hoboken (1981)
224. Settles, B., Craven, M., Friedland, L.: Active learning with real annotation costs. In: Proceedings of the NIPS Workshop on Cost-Sensitive Learning, pp. 1–10 (2008)
225. Settles, B., Craven, M., Ray, S.: Multiple-instance active learning. In: Advances in Neural Information Processing Systems (NIPS), vol.20, pp. 1289–1296, MIT Press, Cambridge (2008)
226. Seung, H.S., Opper, M., Sompolinsky, H.: Query by committee. In: Proceedings of the ACM Workshop on Computational Learning Theory, pp. 287–294 (1992)

227. Shell, J., Coupland, S.: Towards fuzzy transfer learning for intelligent environments. Ambient. Intell. Lect. Notes Comput. Sci. **7683**, 145–160 (2012)
228. Shell, J., Coupland, S.: Fuzzy transfer learning: Methodology and application. Inform. Sci. **293**, 59–79 (2015)
229. Shen, H., Tan, Y., Lu, J., Wu, Q., Qiu, Q.: Achieving autonomous power management using reinforcement learning. ACM Trans. Des. Autom. Electron. Syst. **18**(2), 24:1–24:32 (2013)
230. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **22**(8), 888–905 (2000)
231. Shuman, D.I., Vandergheynst, P., Frossard, P.: Chebyshev polynomial approximation for distributed signal processing. In: Proceedings of the International Conference on Distributed Computing in Sensor Systems, Barcelona, pp. 1–8 (2011)
232. Shuman, D.I., Narang, S.K, Frossard, P., Ortega, A., Vandergheynst, P.: Extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Process. Mag. **30**(3), 83–98 (2013)
233. Silver, D.L., Mercer, R.E.: The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness. In: Thrun, S., Pratt, L.Y. (eds.) Learning to Learn, pp. 213–233. Kluwer Academic, Boston (1997)
234. Sindhwani, V., Niyogi, P., Belkin, M.: Beyond the point cloud: From transductive to semi-supervised learning. In: Proceedings of the 22nd International Conference on Machine Learning (ICML), pp. 824–831. ACM, New York (2005)
235. Smola, J., Kondor, R.: Kernels and regularization on graphs. In: Learning Theory and Kernel Machines, pp. 144–158. Springer, Berlin (2003)
236. Song, J., Babu, P., Palomar, D.P.: Optimization methods for designing sequences with low autocorrelation sidelobes. IEEE Trans. Signal Process. **63**(15), 3998–4009 (2015)
237. Sriperumbudur, B.K., Torres, D.A., Lanckriet, G.R.G.: A majorization-minimization approach to the sparse generalized eigenvalue problem. Mach. Learn. **85**, 3–39 (2011)
238. Sun, J., Zeng, H., Liu, H., Lu, Y., Chen, Z.: CubeSVD: a novel approach to personalized web search. In: Proceedings of the 14th International Conference on World Wide Web, pp. 652–662 (2005)
239. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge (1998)
240. Tang, K., Li, X., Suganthan, P.N., Yang, Z., Weise, T.: Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization. Technical Report, 2009. Available at: https://www.researchgate.net/publication/228932005
241. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J.,, Mei, Q.: LINE: Large-scale information network embedding. In: Proceedings of the International World Wide Web Conference Committee (IW3C2), Florence, pp. 1067–1077 (2015)
242. Tao, D., Li, X., Wu, X., Hu, W., Maybank, S.J.: Supervised tensor learning. Knowl. Inform. Syst. **13**, 1–42 (2007)
243. Thrun, S., Pratt, L. (eds.): Learning to Learn. Kluwer Academic, Dordrecht (1998)
244. Tibshirani, R.: Regression shrinkage and selection via the lasso. J. R. Statist. Soc. B **58**, 267–288 (1996)
245. Tibshirani, R., Walther, G., Hastie, T.: Estimating the number of clusters in a data set via the gap statistic. J. Roy. Statist. Soc. B **63**(2), 411–423 (2001)
246. Tikhonov, A.: Solution of incorrectly formulated problems and the regularization method. Soviet Math. Dokl., **4**, 1035–1038 (1963)
247. Tikhonov, A.N., Arsenin, V.Y.: Solutions of Ill-Posed Problems. Wiley, New York (1977)
248. Tokic, M., Palm, G.: Value-difference based exploration: Adaptive control between epsilon-greedy and softmax. In: KI 2011: Advances in Artificial Intelligence, pp. 335–346 (2011)
249. Tommasi, T., Orabona, F., Caputo, B.: Safety in numbers: learning categories from few examples with multi model knowledge transfer. In: Proceedings of the IEEE Conference on Computer Vision Pattern Recognition 2010, pp. 3081–3088 (2010)
250. Tong, S., Koller, D.: Support vector machine active learning with applications to text classification. J. Mach. Learn. Res. **3**, 45–66 (2001)

251. Tou, J.T., Gonzalez, R.C.: Pattern Recognition Principles. Addison-Wesley, London (1974)
252. Tsitsiklis, J.N.: Asynchronous stochastic approximation and Q-Learning. Mach. Learn. **16**, 185–202 (1994)
253. Uurtio, V., Monteiro, J.M., Kandola, J., Shawe-Taylor, J., Fernandez-Reyes, D., Rousu, J.: A tutorial on canonical correlation methods. ACM Comput. Surv. **50**(95), 14–38 (2017)
254. Valiant, L.G.: A theory of the learnable. Commun. ACM **27**, 1134–1142 (1984)
255. van Hasselt, H.: Double Q-learning. In: Proceedings of the Advances in Neural Information Processing Systems (NIPS), pp. 2613–2621 (2010)
256. Vasilescu, M.A.O., Terzopoulos, D.: Multilinear analysis of image ensembles: TensorFaces. In: Proceedings of the European Conference on Computer Vision, Copenhagen, pp. 447–460 (2002)
257. von Luxburg, U.: A tutorial on spectral clustering. Stat. Comput. **17**(4), 395–416 (2007)
258. Wang, H., Ahuja, N.: Compact representation of multidimensional data using tensor rank-one decomposition. In: Proceedings of the International Conference on Pattern Recognition, vol. 1, pp. 44–47 (2004)
259. Wang, X., Qian, B., Davidson, I.: On constrained spectral clustering and its applications. Data Min. Knowl. Disc. **28**, 1–30 (2014)
260. Wang, L., Hua, X., Yuan, B., Lu, J.: Active learning via query synthesis and nearest neighbour search. Neurocomputing **147**, 426–434 (2015)
261. Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining, pp. 1225–1234. ACM, New York (2016)
262. Watanabe, S.: Pattern Recognition: Human and Mechanical. Wiley, New York (1985)
263. Watldns, C.J.C.H.: Learning from delayed rewards. PhD Thesis, University of Cambridge, England (1989)
264. Watkins, C.J.C.H., Dayan, R.: Q-learning. Mach. Learn. **8**, 279–292 (1992)
265. Weenink, D.: Canonical correlation analysis. IFA Proc. **25**, 81–99 (2003)
266. Wei, X.-Y., Yang, Z.-Q.: Coached active learning for interactive video search. In: Proceedings of the ACM International Conference on Multimedia, pp. 443–452 (2011)
267. Wei, X., Cao, B. Yu, P.S.: Nonlinear joint unsupervised feature selection. In: Proceedings of the 2016 SIAM International Conference on Data Mining, pp. 414–422 (2016)
268. Weiss, K., Khoshgoftaar, T.M., Wang, D.D.: A survey of transfer learning. J. Big Data **3**(9), 1–40 (2016)
269. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**, 229–256 (1992)
270. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 3rd edn. Morgan Kaufmann, San Mateo (2011)
271. Witten, D.M., Tibshirani, R., Hastie, T.: A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. Biostatistics **10**(3), 515–534 (2009)
272. Wright, J., Ganesh, A., Rao, S., Peng, Y., Ma, Y.: Robust principal component analysis: exact recovery of corrupted low-rank matrices via convex optimization. In: Proceedings of the Advances in Neural Information Processing Systems, vol. 87, pp. 20:3–20:56 (2009)
273. Wright, J., Ganesh, A., Yang, A.Y., Ganesh, A., Sastry, S., Ma, Y.: Robust face recognition via sparse representation. IEEE Trans. Pattern Reconginit. Mach. Intell. **31**(2), 210–227 (2009)
274. Wold, H.: Path models with latent variables: The NIPALS approach. In: Blalock, H.M., et al. (eds.) Quantitative Sociology: International Perspectives on Mathematical and Statistical Model Building, pp. 307–357. Academic, Cambridge (1975)
275. Wold, S., Sjöström, M., Eriksson, L.: PLS-regression: a basic tool of chemometrics. Chemom. Intell. Lab. Syst. **58**(2), 109–130 (2001)
276. Wooldridge, M.J., Jennings, N.R.: Intelligent agent: theory and practice. Knowl. Eng. Rev. **10**(2), 115–152 (1995)
277. Wu, P., Dietterich, T.G.: Improving SVM accuracy by training on auxiliary data sources. In: Proceedings of the Twenty-First International Conference on Machine Learning, pp. 871–878 (2004)

278. Wu, T.T., Lange, K.: The MM alternative to EM. Statist. Sci. **25**(4), 492–505 (2010)
279. Wu, Z., Leahy, R.: An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **15**(11), 1101–1113 (1993)
280. Xia, R., Zong, C., Hu, X., Cambria, E.: Feature ensemble plus sample selection: domain adaptation for sentiment classification. IEEE Intell. Syst. **28**(3), 10–18 (2013)
281. Xu, L., Krzyzak, A., Suen, C.Y.: Methods of combining multiple classifiers and their applications to handwriting recognition. IEEE Trans. Syst. Man Cybern. **22**, 418–435 (1992)
282. Xu, L., Oja, E., Suen, C.: Modified Hebbian learning for curve and surface fitting. Neural Netw. **5**, 441–457 (1992)
283. Xu, H., Caramanis, C., Mannor, S.: Robust regression and Lasso. IEEE Trans. Inform. Theory **56**(7), 3561–3574 (2010)
284. Xue, B., Zhang, M., Browne, W.N., Yao, X.: A survey on evolutionary computation approaches to feature selection. IEEE Trans. Evol. Comput. **20**(4), 606–626 (2016)
285. Yamauchi, K.: Covariate shift and incremental learning. In: Advances in Neuro-Information Processing, pp. 1154–1162. Springer, Berlin (2009)
286. Yan, S., Wang, H.: Semi-supervised Learning by sparse representation. In: Proceedings of the SIAM International Conference on Data Mining, Philadelphia, pp. 792–801 (2009)
287. Yang, B.: Projection approximation subspace tracking. IEEE Trans. Signal Process. **43**, 95–107 (1995)
288. Yen, T.-J.: A majorization-minimization approach to variable selection using spike and slab priors. Ann. Stat. **39**(3), 1748–1775 (2011)
289. Yin, J., Yang, Q., Ni, L.M.: Adaptive temporal radio maps for indoor location estimation. In: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications (2005)
290. Yu, K., Zhang, T., Gong, Y.: Nonlinear learning using local coordinate coding. In: Advances in Neural Information Processing Systems, vol. 22, pp. 2223–2231 (2009)
291. Yu, H., Sun, C., Yang, W., Yang, X., Zuo, X.: AL-ELM: One uncertainty-based active learning algorithm using extreme learning machine. Neurocomputing **166**(20), 140–150 (2015)
292. Yuan, M., Lin, Y.: Model selection and estimation in regression with grouped variables. J. Roy. Stat. Soc. Ser. B **68**, 49–67 (2006)
293. Yuan, G.-X., Ho, C.-H., Lin, C.-J.: Recent advances of large-scale linear classification. Proc. IEEE **100**(9), 2584–2603 (2012)
294. Zhang, X.D.: Matrix Analysis and Applications. Cambridge University Press, Cambridge (2017)
295. Zhang, Z., Coutinho, E., Deng, J., Schuller, B.: Cooperative learning and its application to emotion recognition from speech. IEEE Trans. Audio Speech Lang. Process. **23**(1), 115–126 (2015)
296. Zhang, Z., Pan, Z., Kochenderfer, M.J.: Weighted Double Q-learning. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17), pp. 3455–3461 (2017)
297. Zheng, V.W., Pan, S.J., Yang, Q., Pan, J.J.: Transferring multi-device localization models using latent multi-task learning. In: Proceedings of the 23rd Association for the Advancement of Artificial Intelligence (AAAI) Conference on Artificial Intelligence, pp. 1427–1432 (2008)
298. Zheng, V.W., Yang, Q., Xiang, W., Shen, D.: Transferring localization models over time. In: Proceedings of the 23rd Association for the Advancement of Artificial Intelligence (AAAI) Conference on Artificial Intelligence, pp. 1421–1426 (2008)
299. Zhou, Y., Goldman, S.: Democratic co-learning. In: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), pp. 594–602 (2004)
300. Zhou, Z.-H., Li, M.: Tri-training: exploiting unlabeled data using three classifiers. IEEE Trans. Knowl. Data Eng. **17**, 1529–1541 (2005)
301. Zhou, D., Schölkopf, B.: A regularization framework for learning from graph data. In: Proceedings of the ICML Workshop on Statistical Relational Learning, pp. 132–137 (2004)

302. Zhou, J., Chen, J., Ye, J.: Multi-task learning: Theory, algorithms, and applications (2012). Available at: https://archive.siam.org/meetings/sdm12/zhou_-chen_-ye.pdf
303. Zhou, Z.-H., Zhan, D.-C., Yang, Q.: Semi-supervised learning with very few labeled training examples. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI-07) (2007)
304. Zhu, X.: Semi-Supervised Learning Literature Survey. Computer Sciences TR 1530, University of Wisconsin, Madison, (2005)
305. Zhu, X., Goldberg, A.B.: Introduction to Semi-Supervised Learning. In: Brachman, R.J., Dietterich, T. (eds.) Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypoo, San Rafael (2009)
306. Zhu, X., Ghahramani, Z., Laffer, J.: Semi-supervised learning using Gaussian fields and harmonic functions. In: Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington (2003)
307. Zou, H., Hastie, T.: Regularization and variable selection via the elastic net. J. Roy. Stat. Soc. B, **67**(2), 301–320 (2005)
308. Zou, H., Hastie,, T., Tibshirani, R.: Sparse principal component analysis. J. Comput. Graph. Stat. **15**(2), 265–286 (2006)