CHAPTER

Ensemble Learning 4

CHAPTER OUTLINE

4.1 INTRODUCTION

Ensemble learning was originally proposed for classification tasks in a manner of supervised learning in 1965 (Nilsson, 1965); the basic concept of ensemble learning is to train multiple base learners as ensemble members and combine their predictions into a single output that should have better performance on average than any other ensemble member with uncorrelated error on the target data sets. For classification tasks, many researchers have demonstrated the outstanding performance of ensemble learning in their works (Wittner and Denker, 1988; Ho et al., 1994; Cho and Kim, 1995; Breiman, 1996a,b; Raftery et al., 1997; Kittler et al., 1998; Schapire, 1999; Jain et al., 2000; Oza and Tumer, 2001; Tumer and Oza, 2003). Although all these approaches are proposed from various perspectives, they all have to address the fundamental problems of ensemble learning: how to train each of base learners (Ensemble learning algorithms), how to combine the outputs obtained from the multiple base learners (combining methods), and what is the critical factor to determine the success of ensemble learning (ensemble diversity). In this chapter, we are going to answer these questions.

Recently, ensemble learning has been extended to unsupervised learning with different strategies, named as clustering ensemble. This has led to many real-world applications, such as gene classification, image segmentation, video retrieval, and so on.

In this chapter, we are going to describe a general framework of clustering ensemble with two major functions including consensus function and objective function.

4.2 ENSEMBLE LEARNING ALGORITHMS

In ensemble learning approach, the multiple base learner can be trained either coordinately in a sequential manner or individually in a parallel manner. Although many ensemble learning approaches are designed by following these training processes, we are typically going to review two of the most attentional algorithms, which give the initial inspiration to propose the *Iteratively constructed clustering ensemble* model presented in Chapter 6.

Bagging

It is the simplest and most appealing sampling-based ensemble approach. Originally, introduced by Breiman (1996a,b) for supervised learning, this idea has recently been extended to clustering tasks (Dudoit and Fridlyand, 2003; Fischer and Buhmann, 2003). In essence, bagging generates multiple ensemble members built on a bootstrap with replicates of the training set. Then, these ensemble members are combined into the final output by majority voting. The literature (Breiman, 1996a,b) shows us that the performance of a ensemble approach often depends on the diversity of ensemble members in the combination. The sampling procedure of bagging simply creates various training samples from the training set by bootstrap sampling, resulting in diversity among the ensemble members. The ensemble output is then able to reduce the bias and variability, for example, the sensitivity to starting conditions and the possibility of convergence to local minima, in the classification results via the averaging procedure of majority voting. The process of bagging involves following the steps:

1. Create a training set with replacement by randomly sampling the data sets
2. Train the base learner on the training set
3. Repeat 1−2 until satisfying the stopping criteria, for example, the maximum number of loop reached
4. A final classifier is formed by aggregating the trained base learners
5. A testing set can be classified by a majority voting under the final classifier

Boosting

Boosting was originally started from an online learning algorithm named Adaboost (Freund and Schapire, 1997). It sequentially build up a linear combination of base learners that concentrate on difficult examples, such approach results in great success in supervised learning. Recent work (Frossyniotis et al., 2004; Pavlovic, 2004; Liu et al., 2007; Saffari and Bischof, 2007) has extended it with unsupervised learning manner with different strategies. The basic idea of boosting clustering is that each training instance is assigned an adapted weight that determines how "good" the instance was when classified in the previous iteration. Subsets of training data with high (badly clustered) value are more likely to be trained during each iteration, it means paying more attention to instances which are difficult to classify. Clearly, a boosting ensemble is based on sequential runs with pertinent feedback from previous trained base learner. This is different from parallel approaches such as bagging. A general procedure of boosting algorithm follows these steps:

1. Initialize the weights of instances in the data set as $1/N$ (N is the number of instances in the data set)
2. Create a training set by sampling the data set based on the instance weights, where the instances with higher weights are more likely to be selected
3. Train the base learner on the training set, and compute the training error
4. Based on the training error, compute the weight of current trained base learner, and update the weights of instances
5. Repeat steps 1—2 until satisfying the stopping criteria, for example, the maximum number of loops reached or the training error exceeds a threshold
6. A final classifier is form by aggregating the trained base learners associated with their weights
7. A testing set can be classified by a weighted majority voting under the final classifier

4.3 COMBINING METHODS

As long as the multiple base learners have been trained, ensemble learning algorithms have to employ an appropriate method as a combiner to combine their training outputs into a single form as final classifier. Although there are a large number of combining methods appeared in the literature, three of them have been most commonly used and demonstrated good performance for a numerous applications of ensemble learning, which are linear combiner, product combiner, and majority voting combiner (Brown, 2009).

Linear Combiner

In ensemble learning algorithms, a linear combiner is specially applied for supervised learning tasks including classification and regression, where the outputs of the trained base learner are real-valued probability estimates of class label given the input data. Therefore, the combination of these base learners can be formulated as an ensemble probability estimate:

$$p(y|x) = \sum_{t=1}^{T} w_t p_t(y|x) \tag{4.1}$$

where $p_t(y|x)$ is the probability of estimating the class label y given the input data x by the trained base learner t and assigned a weight w_t to determine its performance based on the training set. If the weights of the trained base learners are uniformly distributed $w_t = 1/T,\ \forall t$, the probability estimates of the trained base learners are uniformly averaged into the ensemble probability estimate. Otherwise, the ensemble probability estimate is a weighted average of the probability estimates of the trained base learners. In general, a linear combiner based on weighted average provides a more meaningful combination of base learners and achieves the better performance than the linear combiner based on uniform average.

Product Combiner

If the base learners are trained on the feature spaces, where training sets are highly corrected, the linear combination rule is able to achieve the optimal performance of ensemble learning. However, when the base learner is trained on independent feature spaces, where training sets are uncorrected, it is efficient to combine the trained base learners by multiplying their real-valued probabilistic outputs, which can be formulated as a product combiner:

$$p(y|x) = \frac{1}{Z}\prod_{t}^{T} p_t(y|x) \tag{4.2}$$

where Z is constant-valued factor to normalize the ensemble probability estimate into a valid distribution.

Majority Voting Combiner

Actually both linear and product combination rules are employed for the real-value outputs obtained from the trained base learners. If the base learners directly estimate the class label given the input data, a majority voting combiner can be used to decide the class label of the input data by counting the votes among all trained base learners, which is formulated as:

$$H^i = \arg\max_j \sum_{t}^{T} w_t H_t^{i,j} \tag{4.3}$$

where $H_t^{i,j} = \begin{cases} 1 & \textit{if data i is assigned to class j} \\ 0 & \textit{Otherwise} \end{cases}$ in partition t. If the weights of the trained base learners are uniformly distributed $w_t = 1/T$, $\forall t$, a simple majority voting combiner is formed and commonly used for bagging. Otherwise, a weighted majority voting can be used for boosting.

4.4 DIVERSITY OF ENSEMBLE LEARNING

If a single classification solution does not make any error, the ensemble is unnecessary. If, however, the classification solution does make errors, we seek to complement it with another ensemble member, which makes errors on different objects. Therefore, we realize that the diversity of the ensemble members/trained base learners would therefore be a critical requirement for the success of the ensemble learning. Although many diversity measures have been designed to qualify the diversity of ensemble members, they are generally classified into two categories including pairwise measures and nonpairwise measures. Pairwise measures consider difference between a pair of ensemble members at a time; the overall differences as ensemble diversity value is obtained by averaging across all pairs, for example, *Disagreement measure* (Skalak, 1996) and *Double-fault measure* (Giacinto and Roli, 2001). In contrast, nonpairwise measures consider all the ensemble members together and calculate directly one diversity value for the ensemble, for example, *Entropy measure* (Cunningham and Carney, 2000), *Kohavi-Wolpert variance* (Kohavi and Wolpert, 1996), *The measure of difficulty* (Hansen and Salamon, 1990), *Measurement of inter-rater agreement* (Fleiss, 1973), and *Generalized diversity* (Partridge and Krzanowski, 1997).

In general, diversity alone is not to be beneficial to the ensemble performance, the trained base learners have to be at the same time accurate and diverse in order to produce an optimal ensemble output. Moreover, it is difficult to determine a generally accepted definition of ensemble diversity and even more difficult to relate that measure to the ensemble performance in a neat and expressive dependency. However, it practically provides a guideline for the design of ensemble learning models with better performance. Normally, an ensemble learning algorithm can be designed in one of the three ways:

1. Combining the different base learners trained on the same training set into a final ensemble output such as random subspace (Ho, 1998)
2. Combining the same base learners trained on the different training sets into a final ensemble output by applying a resampling technique on the entire data set such as bagging (Breiman, 1996a,b) and boosting (Freund and Schapire, 1997)
3. Combining the same base learners trained with different parameters and initialization into a final ensemble output such as neural network ensemble (Sharkey, 1999)

4.5 CLUSTERING ENSEMBLE

As described in Section 3.2, it is obvious that each clustering algorithm behaves differently and with its own characteristics. No single clustering algorithm is able to achieve satisfactory results for all types of data set. From the perspective of machine learning, clustering analysis is also an extremely difficult unsupervised learning task since it is inherently an ill-posed problem with solutions which often violate some common assumptions (Kleinberg, 2003). However, there are many feasible approaches developed to improve the performance of clustering analysis. Among them is the ensemble learning method.

Although ensemble method has become an active research area in supervised learning domain and provides a popular way to improve the performance of classification task, it is still quite difficult to directly apply such ideas to unsupervised domain (clustering ensemble) due to two major issues:

1. Unlike classification ensemble, it is not possible to precisely evaluate the quality of different ensemble members (clustering results) for clustering ensemble without providing the priori labeling information of training data. Therefore, the idea of classification ensemble such as boosting cannot be directly applied for clustering task.
2. Unlike classification ensemble, it cannot immediately determine which cluster from one particular clustering result corresponds to which in an other result for clustering ensemble, where different partitions produce incompatible labeling information, resulting in intractable correspondence problems, especially when the numbers of clusters are different. Therefore, directly combining the clustering results with the combination rules (linear combiner, product combiner, and majority voting combiner) becomes meaningless for clustering task.

Recently, clustering ensemble techniques have been studied from different perspectives, for example, clustering ensembles with graph partitioning (Karypis and Kumar, 1999; Strehl and Ghosh, 2003; Fern and Brodley, 2004; Analoui and Sadighian, 2007) evidence aggregation (Monti et al., 2003; Fred and Jain, 2005; Gionis et al., 2007; Ailon et al., 2008; Yang and Chen, 2011a,b) and optimization via semidefinite programming (Singh et al., 2007). The basic idea underlying clustering ensemble techniques is the combining of multiple partitions of a data set by a consensus function yielding a final partition that more likely reflects the intrinsic structure of the data set. Due to the nature of combining the strengths of multiple clustering solutions, clustering ensembles generally outperform the single clustering from several aspects including robustness, novelty, stability, confidence estimation, parallelization, and scalability (Ghaemi et al., 2009).

4.5.1 CONSENSUS FUNCTIONS

The core of a clustering ensemble is the consensus function which must address three issues: how to combine different clustering solutions, how to overcome the label corresponding problem, and how to ensure symmetrical and unbiased consensus with respect to all the input partitions. Many approaches have been developed to solve consensus clustering problems. Ghaemi et al. (2009) have summarized the consensus functions shown in Table 4.1, where K is the number of clusters, N is the size of data set/number of objects, T is the number of ensemble members/input partitions, I is the number of iterations to reach convergence, d is the number of dimension in original feature space, and d' is the number of dimension in transformed feature space. Based on the comparison of their advantages, disadvantages, and computational complexity, we release all the approaches that are able to produce a robust consensus clustering result but most just obtain a trade-off solution between computational cost and accuracy.

Table 4.1 Summarized Consensus Functions

Algorithm	Advantages	Disadvantages	Computing Complexity
METIS (Karypis and Kumar, 1999)	• Coarsens the graph by collapsing vertices and edges • Partitions the coarsened graph and refines the partitions	• Compared to HBGF, low robust clustering performance of METIS • High computational cost	$O(KNT)$
SPEC (Ng et al., 2001)	• A popular multiway spectral graph partitioning algorithm (SPEC) that seeks to optimize the normalized cut criterion	• Compared to HBGF, low robust clustering performance of SPEC • High computational cost	$O(N^3)$
CSPA HGPA MCLA (Strehl and Ghosh, 2003)	• Knowledge reuse; to influence a new cluster based on a different set of features • Controls size of partitions • Low computational cost of HGPA • Improves the quality and robustness of the solution • Allows one to add a stage that selects the best consensus function without any supervisory information by objective function	• High computational cost of CSPA and MCLA • The proposed greedy approach is slowest and is often intractable for large data sets	CSPA-$O(KN^2T)$ HGPA-$O(KNT)$ MCLA-$O(K^2NT^2)$
HBGF (Fern and Brodley, 2004)	• Low computational cost • High robust clustering performance against instance-based and cluster-based approaches • Compared to IBGF and CBGF, the reduction of HBGF is negligible	• Retains all the information of an ensemble • Difficult implementation	$O(KN)$

Continued

Table 4.1 Summarized Consensus Functions—cont'd

Algorithm	Advantages	Disadvantages	Computing Complexity
PBC (Fischer and Buhmann, 2003)	• Extracting arbitrary shaped structures from the data • Automatic outlier detection • Avoiding the dependency on small fluctuations in the data • High stability	• Requires a very high volume of clustering to obtain a reliable result • High computational cost	$O(K^3)$
BagClust (Dudoit and Fridlyand, 2003)	• Reduces variability in the partitioning results via averaging • Improves clustering accuracy by using bagging to cluster analysis • More robust to the variable selection scheme by bagged clustering procedures	• The increase in accuracy observed with PAM is due to a decrease in variability achieved by aggregating multiple clustering • High computational cost	$O(K^3)$
CMWC (Topchy et al., 2003)	• Uses two different weak clustering algorithms: clustering of multidimensional data; clustering by splitting the data using a number of random hyperplanes • Low computational cost	• Requires a few restarts in order to avoid convergence to low-quality local minima	$O(KNT)$
CMCITG (Luo et al., 2006)	• Uses a consensus scheme via the genetic algorithm • Improves accuracy and robustness	• High computational cost	$O(K^3)$
NEACE (Azimi et al., 2007)	• Generates a new feature space from initial clustering outputs better than pure or normalized feature space • Uses a modification of k means for initial clustering named intelligent k-means • Fast convergence • Low computational cost	• Difficult implementation • Unsuitable accuracy	$O(K! + KTIdd')$

Table 4.1 Summarized Consensus Functions—cont'd

Algorithm	Advantages	Disadvantages	Computing Complexity
VKM (Fred, 2001)	• Uses a minimum spanning tree for consistent cluster development • Handles the problem of initialization dependency and selection of the number of clusters by voting-k-means algorithm • Does not entail any specificity toward a particular clustering strategy by the proposed technique	• Does not correspond to known number of classes with number of concluded clusters • High computational cost • Fixed threshold	$O(N^2)$
ClusterFusion (Kellam et al., 2001)	• Uses a comparison metric known as weighted kappa and finds known biological relationships among genes • Generates robust clusters	• Difficult implementation • High computational cost	$O(N^2)$
DCUEA (Fred and Jain, 2005)	• Applies a minimum spanning tree (MST)-based clustering algorithm on a coassociation matrix based on a voting mechanism • The ability of the proposed method to identify arbitrary shaped clusters in multidimensional data	• High computational cost • Poor performance of method in situations involving touching clusters	$O(N^2)$
ACE (Topchy et al., 2004)	• Uses a finite mixture of multinominal distributions in the space of cluster labels • Excellent scalability of algorithm • Comprehensible underlying model • Completely avoids having to solve the label correspondence problem	• High computational cost for minimal weight bipartite matching problem	$O(K^3)$ $O(KNT)$

Continued

Table 4.1 Summarized Consensus Functions—cont'd

Algorithm	Advantages	Disadvantages	Computing Complexity
	• Good ability to handle missing data in the case of missing cluster labels for certain patterns in the ensembles • Operates with arbitrary partitions with varying numbers of clusters		
SCECMGA (Analoui and Sadighian, 2007)	• Uses the genetic algorithm to produce the most stable partitions • Uses a correlation matrix to find the best samples • Excellent scalability of the proposed genetic algorithm • Comprehensible model for clustering of large data sets • Selects clusters with the least perturbation from multiple partitions by objective function	• High computational cost • Unsuitable accuracy	$O(N^2)$

Although these consensus functions summarized in Table 4.1 have implemented in different ways and behaves differently based on their advantages, disadvantages, and computational complexity, there is a taxonomy appearing in the study by Ghaemi et al. (2009) to further classify these consensus functions based on the similar principle, which results five categories of consensus functions in clustering ensemble including hypergraphic partitioning approach, the coassociation-based approach, the voting-based approach, mutual information–based approach, and finite mixture model. Based on the simplicity in implementation, computational complexity, and clustering performance, three of these approaches specially draw our attention and provide the fundamental concepts to propose three clustering ensemble models presented in this book.

In the proposed *Hidden Markov Model (HMM)-based meta clustering ensemble* presented in Chapter 5, hypergraphic partitioning-based consensus functions are employed to combine the multiple clustering results by transforming the original problem domain to the hypergraphic-based domain, which easily avoids the corresponding problem inherited in most of the clustering ensemble algorithms and is quite flexible to apply any graphic-based clustering algorithm on the hypergraphic-based domain to produce a quality and robust ensemble solution. However, these approaches are often time consuming and intractable implementation for a large data set, especially time series data sets, which is demonstrated in our simulation described in Section 5.3.

Thus, we introduce the voting-based consensus function into another proposed clustering ensemble model, *Iteratively constructed clustering ensemble with a hybrid sampling*, presented in Chapter 6. In this approach, the implementation of voting-based consensus becomes much simpler, and significantly reduces the computational cost, which is demonstrated in the simulation described in Section 6.3, than the hypergraphic partitioning-based consensus functions. However, this approach has to explicitly solve the corresponding problem in order to produce a meaningful combination of multiple clustering results. Moreover, it requires that the input partitions must have the same number of clusters, which potentially cause infeasibility in the real-world applications.

Derived from above clustering ensemble models, our proposed clustering ensemble model *Weighted clustering ensemble with multiple representations* presented in Chapter 7 uses the coassociation-based consensus function to provide a trade-off ensemble solution between computational cost and clustering performance. By modifying the existing coassociation-based consensus function, our proposed coassociation-based consensus function named dendrogram-based similarity partitioning algorithm (DSPA) not only results a comparably high robustness in the ensemble solution but also has an ability to automatically determine the number of clusters in the final partition, which is demonstrated on various time series data sets in Section 7.3.

4.5.1.1 Hypergraphic Partitioning Approach

In hypergraphic partitioning, the clusters can be represented as hyperedges on a graph with vertices corresponding to the objects for clustering. Thus, each hyperedge describes a set of objects belonging to the same clusters. This reduces the problem of consensus clustering to simply finding the minimum cut of a hypergraph.

In the work proposed by Strehl and Ghosh (2003), multiple partitions are first mapped onto a hypergraph where its edge, named the hyperdge, is allowed to connect any set of vertices. In the hypergraph, one vertex corresponds to one object in the data set and one cluster forms an edge linked to all objects in the cluster. For example, there are three clustering results based on the same input data. Table 4.2 shows their label vectors and Table 4.3 shows the corresponding hypergraph.

In the hypergraph representation, a data set of N objects $\{x_n\}_{n=1}^{N}$ is represented as the vertices of the hypergraph. The hyperedge, h_i, is a generalization of an edge in that it can connect any set of vertices, which is a column of the matrix for the corresponding cluster, and the entry of a column with 1 indicates that relative objects are grouped into the corresponding cluster and 0 otherwise. Thus, concatenating

Table 4.2 Example of Clustering Ensemble (Label V)

	P_1	P_2	P_3
X_1	1	2	3
X_2	1	2	3
X_3	1	2	2
X_4	2	3	2
X_5	2	3	1
X_6	3	1	1
X_7	3	1	1

Table 4.3 Exmaple of Clustering Ensemble (Hypergraphic)

	H^1			H^2			H^3		
	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9
V_1	1	0	0	0	1	0	1	0	0
V_2	1	0	0	0	1	0	1	0	0
V_3	1	0	0	0	1	0	0	1	0
V_4	0	1	0	0	0	1	0	1	0
V_5	0	1	0	0	0	1	0	0	1
V_6	0	0	1	1	0	0	0	0	1
V_7	0	0	1	1	0	0	0	0	1

all hyperedge of multiple partitions leads to an adjacency matrix $H = \{h_i\}\sum_{i=1}{}^{k_t}$, where k_t is the number of clusters in the partition t, by all objects in the data set versus all the available clusters. In this example, the number of vertices (number of objects) is 7, and the number of hyperedges $(k_1 + k_2 + k_3)$ is 9.

Cluster-Based Similarity Partitioning Algorithm

For each input partition, an $N \times N$ binary similarity matrix encodes the piecewise similarity between any two objects, that is, the similarity of one indicates that two objects are grouped into the same cluster and a similarity of zero otherwise. The coassociation matrix S, which is an entrywise average of all $N \times N$ binary similarity matrices, can be calculated by adjacency matrix H: $S = HH^T$ via multiple-round clustering analyses. It is assumed that a pair of input patterns in the "natural" cluster is more likely to be colocated in the same clusters in different clustering. Therefore, we can construct the cluster-based similarity partitioning algorithm (CSPA) by the following steps:

1. Use multiple clustering results to establish a coassociation matrix based on the measure of pairwise similarity

2. Partition the hypergraph obtained from the coassociation matrix to produce a single clustering by a graphic-based clustering algorithm such as METIS (Karypis and Kumar, 1999).

The clustering ensemble model based on such consensus function can also be described by a pseudo code:

```
Input:
• a set of input partitions $\{P_1, P_2, \ldots, P_T\}$, with number of partitions $T$
• The graphic-based clustering algorithm METIS

for $t = 1$ to $T$
        Find the number of clusters $k_t$ in partition $P_t$
        for $i = 1$ to $k_t$
            construct the hyper-edge $h_i$
        end for
end for
Compute the adjacency matrix $H = \{h_i\} \sum_{i=1} k_t$
Compute the co-association matrix $\mathbf{S} = HH^T$
$K = \max(k_t)$
$P_{consensus} = METIS(S, K)$
Output: the final clustering $P_{consensus}$.
```

Fig. 4.1 shows the CSPA for the clustering ensemble example problem given in Table 4.2. Each entry of the similarity matrix has a value between 0 and 1 (gray level), which is proportional to how likely a pair of input patterns is in the same cluster.

Hypergraph-Partitioning Algorithm

This hypergraph-partitioning algorithm (HGPA) (Strehl and Ghosh, 2003) offers a consensus function by casting the clustering ensemble problem into how to partition the hypergraph by cutting a minimal number of hyperedges. Such a graph partitioning problem has been well studied in graph theory, and the hypergraph partitioning package, HMETIS (Karypis et al., 1997) has been used in this work (Strehl and Ghosh, 2003). Unlike the CSPA that takes the local piecewise similarity into account, the HGPA considers a relatively global relationship of objects across different partitions. In addition, the function has one property that tends to yield a final

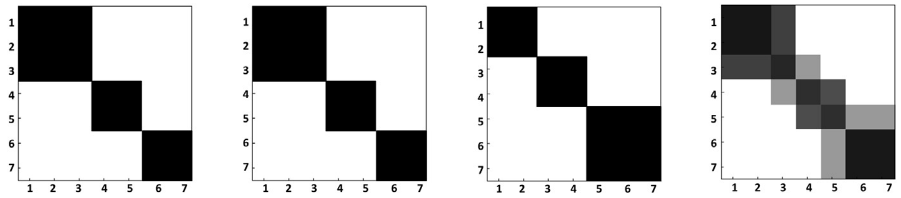

FIGURE 4.1

CSPA similarity matrix.

partition where all clusters are of approximately the same size. Therefore, the HGPA can be constructed by following steps:

1. Construct a hypergraph obtained from input partitions, where each hyperedge describes a set of objects belonging to the same clusters with equal weights, in form of adjacency matrix H.
2. Use hypergraph partitioning package, HMETIS (Karypis et al., 1997) to segment the hypergraph to produce a single consensus clustering by cutting a minimal number of hyperedges

A pseudo code is also given to describe this consensus function:
Input:

- *a set of input partitions $\{P_1, P_2, \ldots, P_T\}$, with number of partitions T*
- *The graphic-based clustering algorithm HMETIS*

```
for $t = 1$ to $T$
        Find the number of clusters $k_t$ in partition $P_t$
        for $i = 1$ to $k_t$
            construct the hyper-edge $h_i$
        end for
end for
Compute the adjacency matrix
$H = \{h_i\} \sum_{i=1} k_t$
$K = \max(k_t)$
$P_{consensus} = HMETIS(H, K)$
Output: the final clustering $P_{consensus}$.
```

For the same clustering ensemble example problem in Table 4.2, Fig. 4.2 shows that each hyperedge is represented by a closed curve enclosing the vertices it connects. The combined clustering {(x1,x2,x3), (x4,x5), (x6,x7)} cuts these hyperedges with a minimal number of 2 and is balanced as possible for three clusters of seven objects.

Meta-Clustering Algorithm

The meta-clustering algorithm (MCLA) (Strehl and Ghosh, 2003) is based on obtained clusters. Each cluster is represented by a hyperedge in the hypergraph.

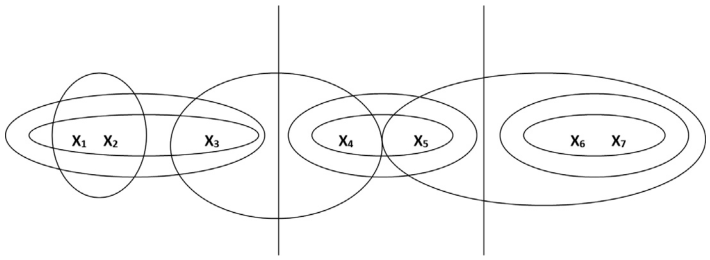

FIGURE 4.2

HGPA hyperedge cutting.

We group the related hyperedges as a meta-cluster C^m and collapse the related hyperedges by assigning each input item to the collapsed hyperedge in which it participates most strongly. We then reduce the number of hyperedges from $\sum k_t$ to K. The entire process of MCLA is comprised of four steps:

1. Construct meta-graph: the hyperedges h_i, $\forall t$ of the hypergraph are represented as vertices in the meta-graph. The edge weights are proportional to the similarity between vertices. The binary Jaccard measure is used as the similarity measure, which is the ratio of the intersection to the union of the sets of input patterns corresponding to the two hyperedges shown as: $w_{a,b} = \frac{h_a^T h_b}{\|h_a\|_2^2 + \|h_b\|_2^2 - h_a^T h_b}$, where h_a and h_b are two hyperedges representing two vertices in the meta-graph, and $w_{a,b}$ is the edge weight between two vertices.
2. Cluster hyperedges: matching labels are found by partitioning the meta-graph into k balanced meta-clusters. Each vertex is weighted proportionally to the size of the corresponding cluster. Balancing ensures that the sum of vertex weights is approximately the same in each meta-cluster. Graph partitioning package METIS is used in this step resulting in a clustering of the h vectors. As each vertex in the meta-graph represents a distinct cluster label, a meta-cluster represents a group of corresponding labels.
3. Collapse meta-clusters: the related hyperedges of a meta-cluster are collapsed into a single meta-hyperedge in the meta-graph. For each meta-cluster, an association vector h_k^m is calculated by an entrywise average of the corresponding h vectors, which indicates how strongly the corresponding input pattern associates with this meta-cluster.
4. Compete for objects: according to the association vectors, the input pattern has to be assigned to the meta-cluster with the highest entry. The input pattern is assigned to the meta-cluster with the highest entry in the association vector. Ties are broken randomly. The confidence of an assignment is reflected by the winner's share of association (ratio of the winner's association to the sum of all other associations). Note that not every meta-cluster can be guaranteed to win at least one object. Thus, there are at most K labels in the final combined clustering.

The clustering ensemble model based on such consensus function can be also described by a pseudo code:

Input:

- *a set of input partitions $\{P_1, P_2,\ldots, P_T\}$, with number of partitions T*
- *The graphic-based clustering algorithm METIS*

```
for $t = 1$ to $T$
    Find the number of clusters $k_t$ in partition $P_t$
    for $i = 1$ to $k_t$
        construct the hyper-edge $h_i$
    end for
end for
Compute the adjacency matrix $H = \{h_i\} \sum_{i=1} k_t$
for $a = 1$ to $\Sigma k_i$
        for $b = 1$ to $\Sigma k_i$
            Compute the edge weights of meta-graph $w_{a,b} = \frac{h_a^T h_b}{\|h_a\|_2^2 + \|h_b\|_2^2 - h_a^T h_b}$
        end for
end for
Construct a matrix of edge weights $W = \left[w_{a,b}\right]_{a=1,2,\ldots,\Sigma k_i}^{b=1,2,\ldots,\Sigma k_i}$
$K = \max(k_t)$
$P^m = METIS(W, H, K)$
for $k = 1$ to $K$
    compute association vector $h_k^m = \frac{1}{\|C_K^m\|} \sum_{C_i \in C_K^m} h_i$ for meta cluster $C_k^m$ in
    partition $P^m$ of meta-graph
end for
$P_{consensus}(x) = \arg\max_{k=1,\ldots K} h_k^m(x)$
Output: the final clustering $P_{consensus}$.
```

For the given clustering ensemble example in Tables 4.2 and 4.3, we consider the first meta-cluster C_1^m associated with corresponding vectors $\{h_3, h_4, h_9\}$ and collapse the hyperedges of the meta-cluster by calculating the association vector $h_1^m = (0,0,0,0,1/3,1,1)$ and apply the same procedure for the rest of meta-clusters. Hence, the C_1^m wins the input pattern X6 and X7. $C_1^m = (x_6, x_7)$, subsequently results in the label of the final clustering result (2,2,2,3,3,1,1).

4.5.1.2 *Coassociation-Based Approach*

In this approach, a coassociation matrix is initially generated by the measure of pairwise similarity based on multiple input partitions, and numerous hierarchical agglomerative algorithms can be applied to the coassociation matrix to obtain the consensus partition.

In hypergraphic partitioning-based consensus functions lie major drawbacks of the cluster number in final clustering has to be manually predefined, or is equal to the largest cluster number among multiple clustering. A coassociation-based consensus function is therefore proposed. This is a DSPA (Yang, 2006) that is able to automatically determine the number of clusters in the final partitioning. The entire process of DSPA consists of following steps:

1. Construct a coassociation matrix used to reflect the relationship among all items in multiple partitions, where the element at location (*i, j*) describes the similarity defined as the number of occurrences as two objects *i* and *j* are grouped into the same cluster
2. Converts the coassociation matrix into a dendrogram, where the horizontal axis indexes the objectives in the given data sets and the vertical axis indicates the lifetimes of clusters. The lifetime of clusters in the dendrogram is defined as an interval from the time the cluster is established to the time the cluster disappears by merging with other clusters.

3. Apply a hierarchical clustering algorithm on the dendrogram to produce the consensus partition and automatically determine the number of clusters in a consensus partition by cutting the dendrogram at a range of threshold values corresponding to the longest cluster's lifetime.

The clustering ensemble model based on such consensus function can be described by a pseudo code:

Input:

- *a set of input partitions* $\{P_1, P_2, \ldots, P_T\}$*, with number of partitions* T
- *the hierarchical clustering algorithm HCLUSTER*

```
for $t = 1$ to $T$
    Find the number of clusters $k_t$ in partition
    for $i = 1$ to $k_t$
        construct the hyper-edge $h_i$
    end for
end for
Compute the adjacency matrix $H = \{h_i\} \sum_{i=1} k_t$
Compute the co-association matrix $S = HH^T$
Construct the a dendrogram $G$ based on the co-association matrix $S$
Define a threshold $\Theta$ based on the longest cluster's lifetime in the dendrogram $G$
$P_{consensus} = HCLUSTER(G, \Theta)$
Output: the final clustering $P_{consensus}$.
```

For the given cluster ensemble example in Table 4.2, Fig. 4.3 shows the dendrogram based on the corresponding coassociation matrix. The cluster lifetimes are identified with 2 clusters: L2 = 0.1, 3 clusters: L3 = 0.5, 5 clusters: L5 = 0.3. Final partitioning, based on the 3 clusters corresponding to the longest lifetime, is chosen.

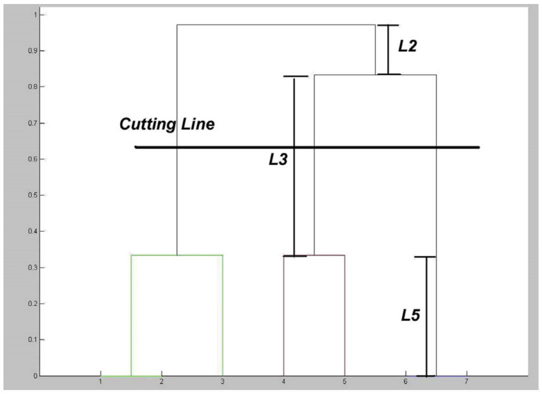

FIGURE 4.3

DSPA cutting line.

4.5.1.3 Voting-Based Approach

The voting approach attempts to solve the correspondence problem. Then, a simple voting schema such as majority voting can be used to assign objects in clusters to determine the final consensus partition. However, label correspondence is exactly what makes unsupervised combination difficult. The main idea is to permute the cluster labels so that best agreement between the labels of two partitions is obtained. All the partitions from the ensemble must be relabeled according to a fixed reference partition. The reference partition can be taken from the ensemble or from a new partition of the data set.

In this approach, the cluster number in each input partition has to be the same as in the reference partition, entire procedure simply involves two steps:

1. Use Hungarian algorithm (Winston and Goldberg, 1994) to reassign the labels of input partitions with selected reference partition such as one of ensemble members.
2. Apply the majority voting onto the relabeled input partitions to produce the cluster label of final consensus partition

This consensus function can be also described by a pseudo code:
Input:

- *an integer K (cluster number results in all input partitions)*
- *an integer N (number of objects/size of dataset)*
- *a set of input partitions {P_1, P_2,…, P_T}, with number of partitions T*
- *a reference partition $P' = P_1$*
- *the Hungarian algorithm HUNGARIAN*

```
for $t = 1$ to $T$
        re-assign the label of input partition: $P'_t = HUNGARIAN(P', P_t)$
end for
$P' = \{P'_t\}_t^T$
for $t = 1$ to $T$
        for $n = 1$ to $N$
             for $k = 1$ to $K$
                  $H_t^{n,k} = \begin{cases} 1 & \text{if data } n \text{ is assigned to cluster } j \text{ in } P'_t \\ 0 & \text{Otherwise} \end{cases}$
             end for
        end for
end for
for $n = 1$ to $N$
        $P_{consensus}(x_n) = \arg\max_k \sum_t^T w_t H_t^{n,k}$ where $w_t = 1/T, \forall t,$
end for
Output: the final clustering $P_{consensus}$.
```

For the given cluster ensemble example in Table 4.3 again, the application of Hungarian algorithm can be described in following steps:

1. Determine a reference partition $P' = P_{1,}$ (1,1,1,2,2,3,3), and construct a pairwise dissimilarity matrix between clusters obtained by reference partition P' and

matching matrix P_3 (3,3,2,2,1,1,1) separately. This matrix is calculated by $DisM_{i,j}(P', P_t) = Z - |C_i \cap C_j|$ where $Z = 3$, is maximum size of cluster in both of partitions.

	C_1^3	C_2^3	C_3^3
C_1'	3	2	1
C_2'	2	2	3
C_3'	1	3	3

2. Subtract the smallest number in each row from every number in the row. This is called a row reduction. Enter the results in a new table.

	C_1^3	C_2^3	C_3^3
C_1'	2	1	0
C_2'	0	0	1
C_3'	0	2	2

3. Subtract the smallest number in each column of the new table from every number in the column. This is called a column reduction. Enter the results in another table.

	C_1^3	C_2^3	C_3^3
C_1'	2	1	0
C_2'	0	0	1
C_3'	0	2	2

4. Test whether an optimum assignment can be made by determining the minimum number of lines needed to cover (i.e., cross out) all zeros. If the number of lines equals the number of rows, an optimum assignment is possible. In this case, go to Step 7. Otherwise go on to Step 5.

	C_1^3	C_2^3	C_3^3
~~C_1'~~	~~2~~	~~1~~	~~0~~
~~C_2'~~	~~0~~	~~0~~	~~1~~
~~C_3'~~	~~0~~	~~2~~	~~2~~

5. If the number of lines is less than the number of rows, modify the table in this way:
   a. Subtract the smallest uncovered number from every uncovered number in the table.

   b. Add the smallest uncovered number to the numbers at intersections of covering lines.
   c. Numbers crossed out but not at intersections of crossout lines carry over unchanged to the next table.
6. Repeat Steps 3 and 4 until an optimal table is obtained.
7. Make the assignments. Begin with rows or columns with only one zero. Match items that have zeros, using only one match for each row and each column. Cross out both the rows and the columns after the match. $C_1^{'}$ - C_3^3, $C_2^{'}$ - C_2^3, $C_3^{'}$ - C_1^3.

After an overall consistent relabeling, the reassigned label results, based on the three input partitions, are shown in Table 4.4. Then majority voting can be simply applied to determine cluster membership for each object, where the label of consensus partition is assigned by the majority of input partitions. The label of consensus partition is (1,1,1,2,2,3,3).

4.5.2 OBJECTIVE FUNCTION

Although the individual use of various consensus functions can yield a single consensus partition, performance differs when applied to data sets of various characteristics. Without prior information, selecting a proper consensus function in advance to form a clustering ensemble is impossible. As with the cluster ensemble proposed in the early work of Strehl and Ghosh (2003), a normalized mutual-information–based objective function (Strehl and Ghosh, 2003) is proposed for measuring the consistency between any two partitions:

$$\mathrm{NMI}(P_a, P_b) = \frac{\sum_{i=1}^{K_a} \sum_{j=1}^{K_b} N_{ij}^{ab} \log\left(\frac{N N_{ij}^{ab}}{N_i^a N_j^b}\right)}{\sqrt{\left(\sum_{i=1}^{K_a} N_i^a \log\left(\frac{N_i^a}{N}\right)\right)\left(\sum_{j=1}^{K_b} N_j^b \log\left(\frac{N_j^b}{N}\right)\right)}}. \tag{4.4}$$

Here, P_a and P_b are labeling for two partitions that divide a data set of N objects into K_a and K_b clusters, respectively. N_i^a denotes the number of objects in cluster $C_i^a \in P_a$ and N_j^b the number of objects in cluster $C_j^b \in P_b$. N_{ij}^{ab} is the number of shared objects

Table 4.4 Reassigned Label

	P_1	P_2	P_3
X_1	1	1	1
X_2	1	1	1
X_3	1	1	2
X_4	2	2	2
X_5	2	2	3
X_6	3	3	3
X_7	3	3	3

between clusters C_i^a and C_j^b. Therefore, the mutual information $I(P_a, P_b)$ between P_a and P_b labeling variables is calculated by $\sum_{i=1}^{K_a} \sum_{j=1}^{K_b} N_{ij}^{ab} \log\left(\frac{NN_{ij}^{ab}}{N_i^a N_j^b}\right)$. However, the mutual information does not have an upper bound. For easy interpretation and comparisons, a normalized version of mutual information $\text{NMI}(P_a, P_b)$ is formulated as $\frac{I(P_a,P_b)}{\sqrt{H(P_a)H(P_b)}}$, where $H(P_a)$ denotes the entropy of P_a labeling variable calculated by $\sum_{i=1}^{K_a} N_i^a \log\left(\frac{N_i^a}{N}\right)$ and $H(P_b)$ denotes the entropy of P_b calculated by $\sum_{j=1}^{K_b} N_j^b \log\left(\frac{N_j^b}{N}\right)$. Based on (4.4), the optimal final partition can be determined by finding out the one that possesses maximal average mutual information with all T partitions available from multiple-round clustering analyses prior to the clustering ensemble (Strehl and Ghosh, 2003). Thus, finding the proper one from R various consensus functions can be performed by

$$P^* = \text{argmax}_{1 \leq r \leq R} \sum_{t=1}^{T} \text{NMI}(P_r, P_t). \tag{4.5}$$

In other words, the consensus function yielding the partition P^* is the correct one for the given data set.

4.6 SUMMARY

For ensemble learning, we have given the definition of ensemble in a manner of supervised learning and addressed the fundamental problems of ensemble learning: how to train each of base learners (ensemble learning algorithms), how to combine the outputs obtained from the multiple base learner (combining methods), and what is the critical factor to determine the success of ensemble learning (ensemble diversity). In ensemble learning algorithms, we have described both bagging and boosting algorithms which gave the initial inspiration to propose the *Iteratively constructed clustering ensemble model* presented in Chapter 6. In combining methods, linear, product, and majority voting combiners have been described in detail. Moreover, we have discussed the diversity issue related to the success of ensemble learning.

For the clustering ensemble, we have studied the problem of combining multiple partitions known as consensus function. By using a simple example, various consensus functions used in our proposed ensemble models described in the latter chapters are detailed in terms of a hypergraphic-based, a coassociation-based, and a voting-based approach. We have identified that three consensus functions based on a hypergraphic-partition approach (Strehl and Ghosh, 2003) suffer from a major weakness, that is, the number of clusters in a final partition needs to be determined manually in advance or we must simply use the maximal number of clusters appearing in multiple partitions for model selection. Motivated by the clustering ensemble based on evidence accumulation (Fred and Jain, 2005), we therefore introduce an alternative consensus function—DSPA—based on a coassociation approach that can automatically determine the number of clusters in the final partition by cutting the dendrogram at a range of threshold values corresponding to the longest cluster's lifetime. Eventually, we present a simple consensus function based on majority voting, which is used in our proposed *Iterative constructed clustering ensemble model* to be described in Chapter 6.