V

Clustering

Clustering is an unsupervised process through which objects are classified into groups called clusters. In categorization problems, as described in Chapter IV, we are provided with a collection of preclassified training examples, and the task of the system is to learn the descriptions of classes in order to be able to classify a new unlabeled object. In the case of clustering, the problem is to group the given unlabeled collection into meaningful clusters without any prior information. Any labels associated with objects are obtained solely from the data.

Clustering is useful in a wide range of data analysis fields, including data mining, document retrieval, image segmentation, and pattern classification. In many such problems, little prior information is available about the data, and the decision-maker must make as few assumptions about the data as possible. It is for those cases the clustering methodology is especially appropriate.

Clustering techniques are described in this chapter in the context of textual data analysis. Section V.1 discusses the various applications of clustering in text analysis domains. Sections V.2 and V.3 address the general clustering problem and present several clustering algorithms. Finally Section V.4 demonstrates how the clustering algorithms can be adapted to text analysis.

V.1 CLUSTERING TASKS IN TEXT ANALYSIS

One application of clustering is the analysis and navigation of big text collections such as Web pages. The basic assumption, called the *cluster hypothesis*, states that relevant documents tend to be more similar to each other than to nonrelevant ones. If this assumption holds for a particular document collection, the clustering of documents based on the similarity of their content may help to improve the search effectiveness.

V.1.1 Improving Search Recall

Standard search engines and IR systems return lists of documents that match a user query. It is often the case that the same concepts are expressed by different terms in different texts. For instance, a "car" may be called "automobile," and a query for

"car" would miss the documents containing the synonym. However, the overall word contents of related texts would still be similar despite the existence of many synonyms. Clustering, which is based on this overall similarity, may help improve the recall of a query-based search in such a way that when a query matches a document its whole cluster can be returned.

This method alone, however, might significantly degrade precision because often there are many ways in which documents are similar, and the particular way to cluster them should depend on the particular query.

V.1.2 Improving Search Precision

As the number of documents in a collection grows, it becomes a difficult task to browse through the lists of matched documents given the size of the lists. Because the lists are unstructured, except for a rather weak relevance ordering, he or she must know the exact search terms in order to find a document of interest. Otherwise, the he or she may be left with tens of thousands of matched documents to scan.

Clustering may help with this by grouping the documents into a much smaller number of groups of related documents, ordering them by relevance, and returning only the documents from the most relevant group or several most relevant groups.

Experience, however, has shown that the user needs to guide the clustering process so that the clustering will be more relevant to the user's specific interest. An interactive browsing strategy called scatter/gather is the development of this idea.

V.1.3 Scatter/Gather

The scatter/gather browsing method (Cutting et al. 1992; Hearst and Pedersen 1996) uses clustering as a basic organizing operation. The purpose of the method is to enhance the efficiency of human browsing of a document collection when a specific search query cannot be formulated. The method is similar to the techniques used for browsing a printed book. An index, which is similar to a very specific query, is used for locating specific information. However, when a general overview in needed or a general question is posed, a table of contents, which presents the logical structure of the text, is consulted. It gives a sense of what sorts of questions may be answered by more intensive exploration of the text, and it may lead to the particular sections of interest.

During each iteration of a scatter/gather browsing session, a document collection is *scattered* into a set of clusters, and the short descriptions of the clusters are presented to the user. Based on the descriptions, the user selects one or more of the clusters that appear relevant. The selected clusters are then *gathered* into a new subcollection with which the process may be repeated. In a sense, the method dynamically generates a table of contents for the collection and adapts and modifies it in response to the user's selection.

V.1.4 Query-Specific Clustering

Direct approaches to making the clustering query-specific are also possible. The hierachical clustering is especially appealing because it appears to capture the essense

of the cluster hypothesis best. The most related documents will appear in the small tight clusters, which will be nested inside bigger clusters containing less similar documents. The work described in Tombros, Villa, and Rijsbergen (2002) tested the cluster hypothesis on several document collections and showed that it holds for query-specific clustering.

Recent experiments with cluster-based retrieval (Liu and Croft 2003) using language models show that this method can perform consistently over document collections of realistic size, and a significant improvement in document retrieval can be obtained using clustering without the need for relevance information from by the user.

V.2 THE GENERAL CLUSTERING PROBLEM

A clustering task may include the following components (Jain, Murty, and Flynn 1999):

- Problem representation, including feature extraction, selection, or both,
- Definition of proximity measure suitable to the domain,
- Actual clustering of objects,
- Data abstraction, and
- Evaluation.

Here we describe the representation of a general clustering problem and several common general clustering algorithms. Data abstraction and evaluation of clustering results are usually very domain-dependent and are discussed in Section V.4, which is devoted to clustering of text data.

V.2.1 Problem Representation

All clustering problems are, in essence, optimization problems. The goal is to select the best among all possible groupings of objects according to the given clustering quality function. The quality function maps a set of possible groupings of objects into the set of real numbers in such a way that a better clustering would be given a higher value.

A good clustering should group together similar objects and separate dissimilar ones. Therefore, the clustering quality function is usually specified in terms of a *similarity* function between objects. In fact, the exact definition of a clustering quality function is rarely needed for clustering algorithms because the computational hardness of the task makes it infeasible to attempt to solve it exactly. Therefore, it is sufficient for the algorithms to know the similarity function and the basic requirement – that similar objects belong to the same clusters and dissimilar to separate ones.

A similarity function takes a pair of objects and produces a real value that is a measure of the objects' proximity. To do so, the function must be able to compare the internal structure of the objects. Various *features* of the objects are used for this purpose. As was mentioned in Chapter I, *feature extraction* is the process of generating the sets of features representing the objects, and *feature selection* is the process of identifying the most effective subset of the extracted features.

The most common *vector space model* assumes that the objects are vectors in the high-dimensional *feature space*. A common example is the bag-of-words model of text documents. In a vector space model, the similarity function is usually based on the distance between the vectors in some metric.

V.2.2 Similarity Measures

The most popular metric is the usual Euclidean distance

$$D(\mathbf{x_i}, \mathbf{x_j}) = \sqrt{\sum_k (x_{ik} - x_{jk})^2},$$

which is a particular case with $p = 2$ of Minkowski metric

$$D_p(\mathbf{x_i}, \mathbf{x_j}) = \left(\sum_k (x_{ik} - x_{jk})^p \right)^{1/p}.$$

For the text documents clustering, however, the cosine similarity measure is the most common:

$$Sim(\mathbf{x_i}, \mathbf{x_j}) = (x'_i \cdot x'_j) = \sum_k x'_{ik} \cdot x'_{jk},$$

where x' is the normalized vector $\mathbf{x} = x/|x|$.

There are many other possible similarity measures suitable for their particular purposes.

V.3 CLUSTERING ALGORITHMS

Several different variants of an abstract clustering problem exist. A *flat (or partitional)* clustering produces a single partition of a set of objects into disjoint groups, whereas a *hierarchical* clustering results in a nested series of partitions.

Each of these can either be a *hard* clustering or a *soft* one. In a hard clustering, every object may belong to exactly one cluster. In soft clustering, the membership is fuzzy – objects may belong to several clusters with a fractional degree of membership in each.

Irrespective of the problem variant, the clustering optimization problems are computationally very hard. The brute-force algorithm for a hard, flat clustering of n-element sets into k clusters would need to evaluate $k^n/k!$ possible partitionings. Even enumerating all possible single clusters of size l requires $n!/l!(n-l)!$, which is exponential in both n and l. Thus, there is no hope of solving the general optimization problem exactly, and usually some kind of a greedy approximation algorithm is used.

Agglomerative algorithms begin with each object in a separate cluster and successively merge clusters until a stopping criterion is satisfied. *Divisive* algorithms begin with a single cluster containing all objects and perform splitting until a stopping criterion is met. "*Shuffling*" algorithms iteratively redistribute objects in clusters.

The most commonly used algorithms are the K-means (hard, flat, shuffling), the EM-based mixture resolving (soft, flat, probabilistic), and the HAC (hierarchical, agglomerative).

V.3.1 K-Means Algorithm

The K-means algorithm partitions a collection of vectors $\{x_1, x_2, \ldots x_n\}$ into a set of clusters $\{C_1, C_2, \ldots C_k\}$. The algorithm needs k cluster seeds for initialization. They can be externally supplied or picked up randomly among the vectors.

The algorithm proceeds as follows:

Initialization:

k seeds, either given or selected randomly, form the core of k clusters. Every other vector is assigned to the cluster of the closest seed.

Iteration:

The *centroids* M_i of the current clusters are computed:

$$M_i = |C_i|^{-1} \sum_{x \in c_i} x.$$

Each vector is reassigned to the cluster with the closest centroid.

Stopping condition:

At convergence – when no more changes occur.

The K-means algorithm maximizes the clustering quality function Q:

$$Q(C_1, C_2, \ldots, C_k) = \sum_{C_1} \sum_{x \in C_i} Sim(x - M_i).$$

If the distance metric (inverse of the similarity function) behaves well with respect to the centroids computation, then each iteration of the algorithm increases the value of Q. A sufficient condition is that the centroid of a set of vectors be the vector that maximizes the sum of similarities to all the vectors in the set. This condition is true for all "natural" metrics. It follows that the K-means algorithm always converges to a local maximum.

The K-means algorithm is popular because of its simplicity and efficiency. The complexity of each iteration is $O(kn)$ similarity comparisons, and the number of necessary iterations is usually quite small.

A major problem with the K-means algorithm is its sensitivity to the initial selection of seeds. If a bad set of seeds is used, the generated clusters are often very much suboptimal. Several methods are known to deal with this problem. The simplest way is to make several clustering runs with different random choices of seeds. Another possibility is to choose the initial seeds utilizing external domain-dependent information.

Several algorithmic methods of dealing with the K-means suboptimality also exist. One possibility is to allow *postprocessing* of the resulting clusters. For instance, the ISO-DATA algorithm (Jensen 1996) merges clusters if the distance between their centroids is below a certain threshold, and this algorithm splits clusters having excessively high variance. Another possibility is employed by the Buckshot algorithm described at the end of this section.

The best number of clusters, in cases where it is unknown, can be computed by running the K-means algorithm with different values of k and choosing the best one according to any clustering quality function.

V.3.2 EM-based Probabilistic Clustering Algorithm

The underlying assumption of *mixture-resolving* algorithms is that the objects to be clustered are drawn from k distributions, and the goal is to identify the parameters of each that would allow the calculation of the probability $P(C_i \mid x)$ of the given object's belonging to the cluster C_i.

The expectation maximization (EM) is a general purpose framework for estimating the parameters of distribution in the presence of hidden variables in observable data. Adapting it to the clustering problem produces the following algorithm:

Initialization:
The initial parameters of k distributions are selected either randomly or externally.
Iteration:
E-Step: Compute the $P(C_i \mid x)$ for all objects x by using the current parameters of the distributions. Relabel all objects according to the computed probabilities.
M-Step: Reestimate the parameters of the distributions to maximize the likelihood of the objects' assuming their current labeling.
Stopping condition:
At convergence – when the change in log-likelihood after each iteration becomes small.

After convergence, the final labelings of the objects can be used as the fuzzy clustering. The estimated distributions may also be used for other purposes.

V.3.3 Hierarchical Agglomerative Clustering (HAC)

The HAC algorithm begins with each object in separate cluster and proceeds to repeatedly merge pairs of clusters that are most similar according to some chosen criterion. The algorithm finishes when everything is merged into a single cluster. The history of merging provides the binary tree of the clusters hierarchy.

Initialization:
Every object is put into a separate cluster.
Iteration:
Find the pair of most similar clusters and merge them.
Stopping condition:
When everything is merged into a single cluster.

Different versions of the algorithm can be produced as determined by how the similarity between clusters is calculated. In the *single-link* method, the similarity between two clusters is the maximum of similarities between pairs of objects from the two clusters. In the *complete-link* method, the similarity is the minimum of similarities of such pairs of objects. The single-link approach may result in long and thin chainlike clusters, whereas the complete-link method results in tight and compact clusters. Although the single-link method is more versatile, experience suggests that the complete-link one produces more useful results.

Other possible similarity measures include "center of gravity" (similarity between centroids of clusters), "average link" (average similarity between pairs of objects of

clusters), and a "group average" (average similarity between all pairs of objects in a merged cluster), which is a compromise between the single- and complete-link methods.

The complexity of HAC is $O(n^2 s)$, where n is the number of objects and s the complexity of calculating similarity between clusters. For some object similarity measures it is possible to compute the group average cluster similarity in constant time, making the complexity of HAC truly quadratic. By definition, the group average similarity between clusters C_i and C_j is

$$Sim(C_i, C_j) = \frac{1}{|C_i \cup C_j|(|C_i \cup C_j| - 1)} \sum_{x,y \in C_i \cup C_j, x \neq y} Sim(x, y).$$

Assuming that the similarity between individual vector is the cosine similarity, we have

$$Sim(C_i, C_j) = \frac{(S_i + S_j) \cdot (S_i + S_j) - (|C_i| + |C_j|)}{|C_i \cup C_j|(|C_i \cup C_j| - 1)},$$

where $S_i = \sum_{x \in Ci} x$ is the sum of all vectors in the ith cluster. If all S_i's are always maintained, the cosine similarity between clusters can always be computed in a constant time.

V.3.4 Other Clustering Algorithms

Several graph-theoretic clustering algorithms exist. The best known is based on construction of the minimal spanning tree (MST) of the objects and then deleting the edges with the largest lengths to generate clusters. In fact, the hierarchical approaches are also related to graph theoretic clustering. Single-link clusters are subgraphs of the MST, which are also the connected components (Gotlieb and Kumar 1968). Complete-link clusters are the maximal complete subgraphs (Backer and Hubert 1976).

The nearest neighbor clustering (Lu and Fu 1978) assigns each object to the cluster of its nearest labeled neighbor object provided the similarity to that neighbor is sufficiently high. The process continues until all objects are labeled.

The Buckshot algorithm (Cutting et al. 1992) uses the HAC algorithm to generate a good initial partitioning for use by the K-means algorithm. For this purpose, $\sqrt{kn}$ objects are randomly selected, and the group-average HAC algorithm is run on the set. The k clusters generated by HAC are used to initialize the K-means algorithm, which is then run on the whole set of n objects. Because the complexity of HAC is quadratic, the overall complexity of Buckshot remains $O(kn)$ linear in the number of objects.

V.4 CLUSTERING OF TEXTUAL DATA

The clustering of textual data has several unique features that distinguish it from other clustering problems. This section discusses the various issues of representation, algorithms, data abstraction, and evaluation of text data clustering problems.

V.4.1 Representation of Text Clustering Problems

The most prominent feature of text documents as objects to be clustered is their very complex and rich internal structure. In order to be clustered, the documents must be converted into vectors in the feature space. The most common way of doing this, the bag-of-words document representation, assumes that each word is a dimension in the feature space. Each vector representing a document in this space will have a component for each word. If a word is not present in the document, the word's component of the document vector will be zero. Otherwise, it will be some positive value, which may depend on the frequency of the word in the document and in the whole document collection. The details and the different possibilities of the bag-of-words document representation are discussed in Section IV. One very important problem arises for clustering – feature selection.

With big document collections, the dimension of the feature space may easily range into the tens and hundreds of thousands. Because of this, feature selection methods are very important for performance reasons. Many good feature selection methods are available for categorization, but they make use of the distribution of features in classes as found in the training documents. This distribution is not available for clustering.

There are two possible ways of reducing the dimensionality of documents. *Local* methods do not reduce the dimension of the whole feature space but simply delete "unimportant" components from individual document vectors. Because the complexity of calculating the similarity between documents is proportional to the number of nonzero components in the document vectors, such truncation is effective. In practice, the document vectors themselves are already quite sparse, and only the centroids, which can be very dense, need truncation.

The alternative approach is a global *dimension reduction*. Its disadvantage is that it does not adapt to unique characteristics of each document. The advantage is that this method better preserves the ability to compare dissimilar documents because every document undergoes identical transformation. One increasingly popular technique of dimension reduction is based on latent semantic indexing (LSI).

V.4.2 Dimension Reduction with Latent Semantic Indexing

Latent semantic indexing linearly maps the N-dimensional feature space F onto a lower dimensional subspace in a provably optimal way, in the following sense: among all possible subspaces $V \in F$ of dimension k, and all possible linear maps M from F onto V, the map given by the LSI perturbs the documents the least, so that the $\sum_{d \in \text{documents}} |D - M(d)|^2$ is minimal. LSI is based upon applying the singular value decomposition (SVD) to the term-document matrix.

V.4.3 Singular Value Decomposition

An SVD of a real $m \times n$ matrix A is a representation of the matrix as a product

$$A = UDV^{\mathrm{T}},$$

where U is a column-orthonormal $m \times r$ matrix, D is a diagonal $r \times r$ matrix, and V is a column-orthonormal $n \times r$ matrix in which r denotes the rank of A. The term "column-orthonormal" means that the column vectors are normalized and have a zero dot-product; thus,

$$UU^{\mathrm{T}} = V^{\mathrm{T}}V = I.$$

The diagonal elements of D are the singular values of A and can all be chosen to be positive and arranged in a descending order. Then the decomposition becomes unique.

There are many methods of computing the SVD of matrices. See Berry (1992) for methods adapted to large but sparse matrices.

Using SVD for Dimension Reduction

The dimension reduction proceeds in the following steps. First, a terms-by-documents rectangular matrix A is formed. Its columns are the vector representations of documents. Thus, the matrix element A_{td} is nonzero when the term t appears in the document d.

Then, the SVD of the matrix A is calculated:

$$A = UDV^{\mathrm{T}}.$$

Next the dimension reduction takes place. We keep the k highest values in the matrix D and set others to zero, resulting in the matrix D'. It can be shown that the matrix

$$A' = UD'V^{\mathrm{T}}$$

is the matrix of rank k that is closest to A in the least-squares sense.

The cosine similarity between the original document vectors is given by the dot product of their corresponding columns in the A matrix. The reduced-dimensional approximation is calculated as the dot product of the columns of A'. Of course, the A' itself need never be calculated. Instead, we can see that

$$A'^{\mathrm{T}}A' = VD'^{\mathrm{T}}U^{\mathrm{T}}UD'V^{\mathrm{T}} = VD'^{\mathrm{T}}D'V^{\mathrm{T}},$$

and thus the representation of documents in the low-dimensional LSI space is given by the rows of the VD^{T} matrix, and the dot product can be calculated between those k-dimensional rows.

Medoids

It is possible to improve the speed of text clustering algorithms by using *medoids* instead of centroids (mentioned in Section V.3). Medoids are actual documents that are most similar to the centroids. This improves the speed of algorithms in a way similar to feature space dimensionality reduction because sparse document vectors are substituted for dense centroids.

Using Naïve Bayes Mixture Models with the EM Clustering Algorithm

For the EM-based fuzzy clustering of text documents, the most common assumption is the Naïve Bayes model of cluster distribution. This model has the following

parameters: the prior cluster probability $P(C_i)$ and the probabilities $P(f_i \mid C_i)$ of features in the cluster.

Given the model parameters, the probability that a document belongs to a cluster is

$$P(C_i|x) = P(C_i)\prod_f P(f|C_i)\Big/\sum_C P(C)\prod_f P(f|C).$$

On the assumption that the current document labeling is $L(x)$, the maximum likelihood estimation of the parameters is

$$P(C_i) = |\{x : L(x) = C_i\}|/N,$$

$$P(f|C_i) = |\{x : L(x) = C_i \quad \text{and} \quad f \in x\}|/|\{x : L(x) = C_i\}|,$$

where N is the number of documents.

Using this method it is possible to improve *categorization* systems in cases in which the number of labeled documents is small but many unlabeled documents are available. Then the labeled documents can be used to train the initial NB models, which are then used within the EM algorithm to cluster the unlabeled documents. The final cluster models are the output classifiers produced by this technique. The experiments have shown a significant improvement in accuracy over the classifiers that are trained only on labeled data (Nigam et al. 2000).

V.4.4 Data Abstraction in Text Clustering

Data abstraction in clustering problems entails generating a meaningful and concise description of the cluster for the purposes of further automatic processing or for user consumption. The machine-usable abstraction is usually easiest; natural candidates are cluster centroids or probabilistic models of clusters.

In the case of text clustering, the problem is to give the user a meaningful cluster label. For some applications, such as scatter/gather browsing, a good label is almost as important as good clustering. A good label would consist of a very small number of terms precisely distinguishing the cluster from the others. For instance, after clustering documents about "jaguar," we would like one cluster to be named "Animal" and another "Car."

There are many possibilities of generating cluster labels automatically:

- A title of the medoid document or several typical document titles can be used.
- Several words common to the cluster documents can be shown. A common heuristic is to present the five or ten most frequent terms in the centroid vector of the cluster.
- A distinctive noun phrase, if it can be found, is probably the best label.

V.4.5 Evaluation of Text Clustering

Measuring the quality of an algorithm is a common problem in text as well as data mining. It is easy to compare the exact measures, such as time and space complexity,

but the quality of the results needs human judgment, which introduces a high degree of subjectivity.

The "internal" measures of clustering quality are essentially the functions we would like to optimize by the algorithms. Therefore, comparing such measures for clusterings produced by different algorithms only shows which algorithm results in a better approximation of the general optimization problem for the particular case. This makes some sense, but what we would like to see is a measure of how good the clustering is for human consumption or for further processing.

Given a set of categorized (manually classified) documents, it is possible to use this benchmark labeling for evaluation of clusterings. The most common measure is *purity*. Assume $\{L_1, L_2, \ldots, L_n\}$ are the manually labeled classes of documents, and $\{C_1, C_2, \ldots, C_m\}$ are the clusters returned by the clustering process. Then,

$$Purity(C_i) = \max_j |L_j \cap C_i| / |C_i|.$$

Other measures include the *entropy* of classes in clusters, *mutual information* between classes and clusters, and so on. However, all these measures suffer from the limitation that there is more than one way to classify documents – all equally right.

Probably the most useful evaluation is the straightforward measure of the utility of the resulting clustering in its intended application. For instance, assume the clustering is used for improving the navigation of search results. Then it is possible to prepare a set of queries and the intended results manually and to measure the improvement produced by clustering directly using simulated experiments.

V.5 CITATIONS AND NOTES

Section V.1

The scatter/gather method was introduced by Cutting in Cutting et al. (1992) and further expanded in Cutting, Karger, et al. (1993). Application and analysis of the scatter/gather methods are described in Cutting, Karger, et al. (1992); Cutting, Karger, and Pedersen (1993); Hearst, Karger, and Pedersen (1995); and Hearst and Pedersen (1996).

Section V.3

Descriptions of general clustering algorithms and comparisons between them can be found in the following papers: Mock (1998); Zhong and Ghosh (2003); Jain and Dubes (1988); Goldszmidt and Sahami (1998); Jain et al. (1999); and Steinbach, Karypis, and Kumar (2000). Algorithms for performing clustering on very large amount of data are described in Bradley, Fayyad, and Reina (1998) and Fayyad, Reina, and Bradley (1998).

Section V.4

Clustering by using latent semantic indexing (LSI) is described in the following papers: Deerwester et al. (1990); Hull (1994); and Landauer, Foltz, and Laham (1998). In many cases there is a need to utilize background information and external knowledge bases. Clustering using backround information is described in Hotho et al.

(2003), and clustering using ontologies is described in Hotho, Staab, and Maedche (2001). Clustering using the popular WordNet resource is mentioned in Benkhalifa, Mouradi, and Bouyakhf (2001a, 2000b).

Specific clustering algorithms adapted for textual data are described in Iwayama and Tokunaga (1995a, 1995b); Goldszmidt and Sahami (1998); Zamir and Etzioni (1999); El-Yaniv and Souroujon (2001); and Dhillon, Mallela, and Kumar (2002).