# HIERARCHICAL DOCUMENT CLUSTERING USING FREQUENT ITEMSETS

by

Benjamin Chin Ming Fung

B.Sc., Simon Fraser University, 1999

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science


SIMON FRASER UNIVERSITY
September 2002

# APPROVAL

**Name:** Benjamin Chin Ming Fung

**Degree:** Master of Science

**Title of thesis:** Hierarchical Document Clustering Using Frequent Itemsets

**Examining Committee:** Dr. Qianping Gu
Chair

Dr. Ke Wang, Senior Supervisor

Dr. Martin Ester, Supervisor

Dr. Oliver Schulte, SFU Examiner

**Date Approved:** _____

# Abstract

Most state-of-the art document clustering methods are modifications of traditional clustering algorithms that were originally designed for data tuples in relational or transactional database. However, they become impractical in real-world document clustering which requires special handling for high dimensionality, high volume, and ease of browsing. Furthermore, incorrect estimation of the number of clusters often yields poor clustering accuracy. In this thesis, we propose to use the notion of frequent itemsets, which comes from association rule mining, for document clustering. The intuition of our clustering criterion is that there exist some common words, called frequent itemsets, for each cluster. We use such words to cluster documents and a hierarchical topic tree is then constructed from the clusters. Since we are using frequent itemsets as a preliminary step, the dimension of each document is therefore, drastically reduced, which in turn increases efficiency and scalability.

*To my parents*
*and Akina*

*"When we see persons of worth, we should think of equaling them; when we see persons of a contrary character, we should turn inwards and examine ourselves."*

— Confucius

# Acknowledgments

I owe deep debt of gratitude to my senior supervisor, Dr. Ke Wang, for his skillful guidance, enthusiasm and valuable criticism of my work over the last year, not to mention his vast amount of knowledge. I am very thankful to my supervisor, Dr. Martin Ester, for his insightful advice and continuous encouragement during my research. My gratitude also goes to Dr. Oliver Schulte, for his patiently reading through my thesis, and providing valuable feedback that has served to improve this thesis. This thesis would not have been possible without their strongest support to me.

I would like to express my sincere thanks to my fellow graduate students Leo Chen and Linda Wu whom I had extensive discussions with in the initial phase of this work.

Thanks full of mercy and affection go to my parents who supported me morally all along my education. Thanks of a special kind to my fiancee, Akina Lo, for her understanding and never ending encouragement during my research in data mining.

# Contents

**Approval** **ii**

**Abstract** **iii**

**Dedication** **iv**

**Quotation** **v**

**Acknowledgments** **vi**

**List of Tables** **x**

**List of Figures** **xi**

**1 Introduction** **1**

1.1 Data Mining and Association Mining . . . . . 1

1.2 Cluster Analysis . . . . . 4

1.3 What is hierarchical document clustering? . . . . . 7

1.4 Motivation . . . . . 8

1.5 Frequent Itemset-based Hierarchical Clustering . . . . . 9

1.6 Thesis Organization . . . . . 10

**2 Related Work** **11**

2.1 Hierarchical Methods . . . . . 13

2.1.1 Agglomerative and Divisive Hierarchical Clustering . . . . . 13

2.1.2 Evaluation of Hierarchical Methods . . . . . 14
2.2 Partitioning Methods . . . . . 15
2.2.1 The k-means algorithm and its variants . . . . . 15
2.2.2 Evaluation of Partitioning Methods . . . . . 16
2.3 Frequent Itemset-based Methods . . . . . 17
2.3.1 The Apriori Algorithm . . . . . 17
2.3.2 The Frequent Pattern-growth Algorithm . . . . . 18
2.3.3 Transaction Clustering . . . . . 19
2.3.4 Hierarchical Frequent Term-based Clustering . . . . . 19
2.4 Other Clustering Methods . . . . . 20
2.4.1 Density-based Methods . . . . . 20
2.4.2 Grid-based Methods . . . . . 20

**3 Constructing Clusters** **23**
3.1 Constructing Initial Clusters . . . . . 26
3.2 Making Clusters Disjoint . . . . . 27

**4 Building the Cluster Tree** **32**
4.1 Tree Construction . . . . . 32
4.2 Tree Pruning . . . . . 35
4.2.1 Child Pruning . . . . . 36
4.2.2 Sibling Merging . . . . . 38

**5 Experimental Evaluation** **41**
5.1 Data Sets . . . . . 41
5.2 Evaluation Method: F-measure . . . . . 43
5.3 Experimental Results . . . . . 44
5.3.1 Accuracy . . . . . 44
5.3.2 Sensitivity to Parameters . . . . . 47
5.3.3 Efficiency and Scalability . . . . . 48

**6 Discussions and Conclusions** **52**

6.1 Browsing . . . . . 52

6.2 Complexity Analysis . . . . . 53

6.3 Contributions . . . . . 54

6.4 Future Work . . . . . 56

**Bibliography** **59**

# List of Tables


1.1 Transaction database $T$ . . . . . 3
1.2 Frequent itemsets of $T$ . . . . . 3

3.1 Document set . . . . . 25
3.2 Global frequent itemsets . . . . . 25
3.3 Initial clusters . . . . . 27
3.4 Disjoint clusters . . . . . 30

4.1 Inter-cluster similarity calculation . . . . . 40

5.1 Summary descriptions of data sets . . . . . 42
5.2 F-measure comparison . . . . . 45
5.3 Comparison on class/cluster frequent items . . . . . 46

# List of Figures


1.1 Balls with marks in random order . . . . . 5
1.2 Balls in groups . . . . . 5

2.1 Agglomerative hierarchical clustering algorithm . . . . . 14
2.2 Basic k-means algorithm . . . . . 15
2.3 Bisecting k-means algorithm . . . . . 16
2.4 A hierarchical structure for STING clustering . . . . . 21

4.1 Tree construction algorithm . . . . . 33
4.2 Cluster tree built from table 3.4 . . . . . 34
4.3 Child pruning algorithm . . . . . 36
4.4 Cluster tree after child pruning . . . . . 37
4.5 Sibling merging algorithm . . . . . 39
4.6 Sibling merged tree . . . . . 40
4.7 Cluster tree after child pruning and sibling merging . . . . . 40

5.1 Sensitivity to MinSup without pre-specifying # of clusters . . . . . 47
5.2 Comparison on efficiency . . . . . 49
5.3 Comparison on efficiency with scale-up document set . . . . . 50
5.4 Scalability of FIHC . . . . . 51

6.1 Cluster labels . . . . . 58

# Chapter 1

# Introduction

Everyday a vast amount of documents, reports, e-mails, and web pages are generated from different sources, such as enterprises, governments, organizations, and individuals. This kind of unstructured data is usually not stored on relational or transaction database systems, but on web servers, file servers, or even personal workstations. Large enterprises often spend lots of manpower on organizing these documents into a logical structure for later use. They pursue a systematic and automatic approach in organizing these documents without human intervention or preparation work. This thesis takes on the challenge of developing an accurate, efficient, and scalable method for clustering documents into a hierarchical structure that facilitates browsing.

Cluster analysis is one of the major topics in data mining. We start this chapter by describing data mining in brief.

## 1.1 Data Mining and Association Mining

The nature of database technology and automated data collection tools leads to tremendous amounts of data stored in databases, data warehouses, and other information repositories. These large amounts of data are worthless unless they become knowledge - not to mention analyzing them is a trivial task either. This problem is called data explosion meaning rich in data, but starved in knowledge.

Data mining, also known as Knowledge Discovery in Databases (KDD), is a solution of data explosion. Simply stated, data mining is a method of extracting interesting knowledge, such as rules, patterns, regularities, or constraints, from data in large databases. The extracted knowledge should be non-trivial, previously unknown, implicit, and potentially useful in that it may serve as an important input for making decisions.

The key functionalities of data mining are association mining, classification and prediction, and cluster analysis. We often apply these techniques to different types of data to solve different problems. Some applications of data mining are target marketing, customer relation management, market basket analysis, cross selling, market segmentation, forecasting, quality control, fraud detection, and intelligent query answering.

A major breakthrough of this thesis is that we utilize an important notion, frequent itemset, in association mining to cluster text documents. Thus, let us briefly explain association mining.

Association mining [6, 7] searches for interesting frequent patterns, associations, correlations, or causal relationships among sets of items or objects in transactional databases, relational databases, and other information repositories. Market basket analysis is a typical example of association mining on transaction data. It analyzes customer buying habits by finding associations among the different items that are purchased together. The manager of a supermarket may make use of this knowledge to increase the sales of the associated items. The output of association mining is usually a rule form, i.e., $A \Rightarrow B$. For example,

$$Laptop \Rightarrow Modem\ [support = 5\%, confidence = 80\%]$$

*Support* and *confidence* are two measures of rule interestingness. The above association rule means that 5% of all transactions under analysis show that laptop and

| Transaction ID | Items Bought |
|---|---|
| 100 | A, B, C |
| 200 | A, C |
| 300 | A, D |
| 400 | B, E, F |

Table 1.1: Transaction database $T$

| Frequent Itemset | Support |
|---|---|
| {A} | 75% |
| {B} | 50% |
| {C} | 50% |
| {A, C} | 50% |

Table 1.2: Frequent itemsets of $T$ (minimum support = 50%)

modem are purchased together, and 80% of the customers who purchase a laptop also buy a modem. Association rule mining has two steps:

1. Compute all *frequent itemsets*, where frequent itemsets are a set of items that occur together at least as frequently as a pre-determined minimum support count, i.e., a minimum fraction of transactions contains these itemsets. We will use Example 1a to illustrate this concept.

2. Generate strong association rules from the frequent itemsets, where these rules are association rules that satisfy minimum support and minimum confidence. This step is easy to compute, but it is only useful for association rule mining and is not applicable to this thesis.

Before presenting an example to explain the notion of frequent itemset, we first give some formal definitions. A set of items is referred to as an *itemset*. An itemset containing $k$ items is called *k-itemset*. The *support* of an itemset refers to the percentage of transactions containing the itemset. If an itemset satisfies a user-specified minimum support, then it is a *frequent itemset*.

**Example 1a**: Table 1.1 shows a small transaction database $T$ of a supermarket that contains only four transactions with the corresponding items bought. Suppose the minimum support is set to 50%; that is, an itemset $i$ is frequent only if at least two out of the four transactions contain itemset $i$. Table 1.2 presents all the frequent itemsets of $T$. For example, the 1-itemset {A} is frequent because it appears in three transactions and its support is 75%. Similarly, both itemsets {B} and {C} have support 50%, so they are frequent 1-itemsets. {A, C} is a frequent 2-itemset because both items A and C appear together in two transactions, so it has support 50%. However, {A, B} is not a frequent itemset because both items A and B appear together in only one transaction.

Many algorithms [25] were proposed for computing frequent itemsets, and the most well-known methods are the *Apriori* [5, 6] and the *FP-growth* [24] algorithms. More details of both algorithms are presented in sections 2.3.1 and 2.3.2 respectively.

## 1.2 Cluster Analysis

Cluster analysis is an important human activity and it often forms the basis of learning and knowledge. An example can be found from a child who learns how to distinguish between animals and plants, or between birds and fishes, by continuously improving subconscious clustering schemes. Basically, the scheme is learnt by observing the properties or characteristics (e.g., the presence of wings) of objects. This type of binary property is easy to measure, but some properties may be more difficult to measure. An example would be ones that are expressed in a numerical value, e.g., the height of a person.

**Example 1b**: This example demonstrates the clustering of balls of the same mark. Figure 1.1 shows a total of ten balls which are of three different marks. We are interested in grouping the balls into three clusters by their marks as shown in figure 1.2.

Clustering is applicable to a wide variety of research problems. In the field of

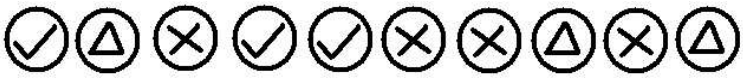

Figure 1.1: Balls with marks in random order

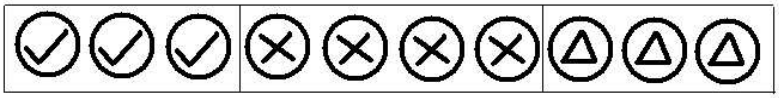

Figure 1.2: Balls in groups

business, clustering can help marketers discover distinct groups in their customer bases and characterize customer groups based on purchasing patterns. In the field of medicine, clustering diseases, symptoms of diseases, and cures for diseases often leads to useful taxonomies. In the field of biology, it can be used to categorize genes with similar functionality and derive plant and animal taxonomies. In the field of psychiatry, successful therapy always depends on the correct diagnosis of clusters of symptoms such as paranoia, schizophrenia, etc. In archeology, researchers often apply cluster analytic techniques to establish taxonomies of stone tools, funeral objects, etc. In general, cluster analysis often provides a feasible solution whenever one needs to classify a large amount of information into manageable meaningful structures.

Clustering is a process of partitioning a set of data objects into a set of meaningful subclasses, called clusters. Formally, given a collection of $n$ objects each of which is described by a set of $p$ attributes, clustering aims to derive a useful division of the $n$ objects into a number of clusters. A cluster is a collection of data objects that are similar to one another based on their attribute values, and thus can be treated collectively as one group. Clustering is useful in getting insight into the distribution of a data set.

A clustering algorithm attempts to find natural groups of data based on similarity of attributes. The following are some typical requirements of clustering in data mining [23].

- *Scalability.* Many clustering algorithms work fine on small data sets; however, some of them fail to handle large data set containing over ten thousands of data objects. An immediate solution to this problem is to perform clustering on a subset (or sample) of a given large data set, but it may lead to biased results.

- *High dimensionality.* A database can contain several dimensions or attributes. Most of the clustering algorithms work well on low-dimensional data, but may fail to cluster data objects in high-dimensional space, especially when the data objects are very sparse and highly skewed. In high dimensional data sets, natural clusters usually do not exist in the full dimensional space, but only in the subspace formed by a set of correlated dimensions. Locating clusters in the subspace can be challenging. One typical example is document clustering which is also the focus of this thesis. Many clustering algorithms simply construct a new dimension for each distinct word in the document set. Due to the large corpus in English, the space usually contains over ten thousands of dimensions, which greatly reduces the performance of the algorithm. This problem is also closely related to the issues of scalability and efficiency.

- *Arbitrary shape of clusters.* Many algorithms perform clustering based on Euclidean or Manhattan distance measure. Algorithms using this kind of distance measurement always tend to find spherical clusters with similar density and size. This limitation often degrades the accuracy.

- *Insensitivity to the order of input data.* Some clustering algorithms are very sensitive to the order of input data. The clustering solutions produced from the same set of data objects may be completely different depending on different orderings of input data. In other words, the quality of clustering solutions may vary substantially and become unpredictable.

- *Noisy data handling.* Outliers or erroneous data is a common problem in database. A robust clustering algorithm should minimize the impact of this noise; otherwise, it may lead to poor clustering accuracy.

- *Prior domain knowledge.* Many clustering algorithms require the user to specify some input parameters. To determine reasonable values of these input parameters, some prior domain knowledge is often needed. However, they are hard to estimate in some cases, especially for data sets containing high-dimensional objects. Clustering accuracy may degrade drastically if a clustering algorithm is too sensitive to these input parameters. This not only burdens users, but also makes the quality of clustering difficult to control.

We will use these requirements to evaluate our clustering method in the conclusion of this thesis. The major categories of clustering algorithms are discussed in Chapter 2.

## 1.3 What is hierarchical document clustering?

Document clustering is the automatic organization of documents into clusters or groups so that documents within a cluster have high similarity in comparison to one another, but are very dissimilar to documents in other clusters. In other words, the grouping is based on the principle of maximizing intra-cluster similarity and minimizing inter-cluster similarity. The major challenge of clustering is to efficiently identify meaningful groups that are concisely annotated.

Document clustering differs from other techniques, such as classification [1, 10, 48] or taxonomy building, in that it is fully automated: there is no human intervention at any point in the whole process and no labeled documents are provided. Thus, clustering is also called unsupervised learning because we learn by "observation" rather than by "examples".

The accuracy of the clustering solution is measured by an external evaluation method *F-measure* together with a set of manually pre-classified documents which is

also known as the set of natural classes. F-measure first identifies the cluster that can best represent a given natural class in the document set, then it measures the accuracy of the best cluster against the natural class. Finally, it calculates the weighted average on the accuracy of each natural class. Section 5.2 will explain F-measure in details.

Instead of producing a flat list of clusters, hierarchical document clustering organizes clusters into a hierarchy or a tree that facilitates browsing. The parent-child relationship among the nodes in the tree can be viewed as topics and subtopics in a subject hierarchy.

## 1.4 Motivation

Document clustering has been studied intensively because of its wide applicability in areas such as web mining [30, 22], information retrieval [45], and topological analysis. Another catalyst for developing an effective document clustering algorithm is the huge amount of unstructured data on the Internet. The majority of this information is in text format, for example, emails, news, web pages, reports, etc. Organizing them into a logical structure is a challenging task. More recently, clustering is employed for browsing a collection of documents [14] or organizing the query results returned by a search engine [51]. It may also serve as a preprocessing step for other data mining algorithms such as document classification [19]. An ambitious goal of document clustering is to automatically generate hierarchical clusters of documents [29] that is similar to the Yahoo! subject hierarchy.

Although standard clustering techniques such as k-means [16, 28] can be applied to document clustering, they usually do not satisfy the special requirements for clustering documents: high dimensionality, high volume of data, ease for browsing, and meaningful cluster labels. In addition, many existing document clustering algorithms require the user to specify the number of clusters as an input parameter. Incorrect estimation of the value always leads to poor clustering accuracy. Furthermore, many clustering algorithms are not robust enough to handle different types of document

sets in a real-world environment. In some document sets, cluster sizes may vary from few to thousands of documents. This variation tremendously reduces the resulting clustering accuracy for some of the state-of-the art algorithms.

The concept of hierarchical clustering and the weaknesses of the standard clustering methods formulate the goal of this research: Provide an accurate, efficient, and scalable clustering method that addresses the special challenges of document clustering. The resulting hierarchy of clusters should facilitate browsing and be suitable for further processing by other data mining algorithms.

## 1.5 Frequent Itemset-based Hierarchical Clustering

In this thesis, we propose a novel approach, *Frequent Itemset-based Hierarchical Clustering* (FIHC), for document clustering based on the idea of frequent itemsets, which comes from association rule mining. The intuition of our clustering criterion is that there exists some frequent itemsets (sets of common words) for each cluster (topic) in the document set. In other words, some minimum fraction of documents in the cluster contains these itemsets. Since each cluster has different frequent itemsets, they can be used to cluster documents. The major features of our approach are as follows:

- *Reduced dimensionality.* As we are using only frequent itemsets, the dimension of a document vector, which keeps track of the frequency of the words appearing in a document, is drastically reduced. This is a key factor for the efficiency and scalability of FIHC.

- *Consistently high clustering accuracy.* Experimental results show that FIHC outperforms the well-known clustering algorithms in terms of accuracy. It is robust and consistent even when it is applied to large and complicated document sets.

- *Number of clusters as an optional input parameter.* Many existing clustering algorithms require the user to specify the desired number of clusters as an input parameter. FIHC treats it only as an optional input parameter. Close to optimal clustering quality can be achieved even when this value is unknown.

- *A sensible pruning strategy.* Building the hierarchical topic tree from frequent itemsets provides a concrete foundation for pruning in case there are too many clusters. Pruning does not only remove overly specific clusters, but also increases the clustering accuracy by merging similar clusters together.

- *Easy to browse with meaningful cluster labels.* Another feature of the topic tree is its logical structure for browsing. Each cluster in the tree has a corresponding frequent itemset as its cluster label which a user may utilize for browsing.

- *Efficient and scalable.* It is very common that a real world document set may contain a few hundred thousand of documents. Clustering on this high volume of data is a challenging task. Our method can complete the clustering process within two minutes while some of the traditional clustering algorithms cannot even produce a clustering solution after hours of operation. Experiments show that our method is significantly more efficient and scalable than all of the tested competitors.

## 1.6 Thesis Organization

The outline of this thesis is as follows. Chapter 2 briefly discusses few essential topics in document clustering and some well-known clustering algorithms. Chapters 3 and 4 present our algorithm in two stages, cluster construction and tree building, with a running example. Chapter 5 shows the experimental results and the comparison with other algorithms. We conclude the paper and outline future directions of research in Chapter 6.

# Chapter 2

# Related Work

We first briefly review a few essential topics to provide some background knowledge in document clustering. Some topics originate in the field of information retrieval [45, 31].

Most document clustering algorithms employ several preprocessing steps including removing stop words and stemming on the document set. Stop words are the most common words (e.g., "and", "or", "in") in a language, but they do not convey any significant information so they are stripped from the document set. Word stemming is language-specific algorithm that aims to reduce a word to its canonical form. For example, "computation" might be stemmed to "comput". For clustering purposes, it usually does not make any difference whether the stems generated are genuine words or not. Stemming does not only conflate different variants of a term to a single representative form, but also reduces the number of distinct terms needed for representing a set of documents. A smaller number of distinct terms results in a saving of memory space and processing time.

Each document is represented by a vector of frequencies of remaining items within the document. These *document vectors* form the *vector model* on which all of the operations for clustering are performed. There are typically several thousands to ten thousands of remaining items after stop words removal and stemming. In other words,

the vector space still has a very high dimensionality [39].

As an extra preprocessing step, many document clustering algorithms would replace the actual term frequency of an item by the weighted frequency, i.e., *term frequency - inverse document frequency* (TF-IDF), in the document vector. The idea is that if an item is too common across different documents, then it would have little discriminating power, and vice versa [45]. Experiments show that TF-IDF increases the clustering accuracy in all tested algorithms. The weighted frequency of term $k$ in document $i$ is defined as follows:

$$W_{ik} = f_{ik} * (log_2\ N - log_2\ d_k + 1) \tag{2.1}$$

where $N$ is the number of documents, $d_k$ is the number of documents containing term $k$, $f_{ik}$ is the absolute frequency of term $k$ in document $i$, and $W_{ik}$ is the weighted frequency of term $k$ in document $i$.

Similar to other document clustering algorithms, our method also employs stop words removal, stemming, vector model, and TF-IDF. The effect of TF-IDF on our algorithm is explained in Chapter 3.

To cluster similar documents together, most of the traditional clustering algorithms require a similarity measure between two documents $d_1$ and $d_2$. Many possible measures are proposed in the literature, but the most common one is the *cosine measure* [42] and it is defined below:

$$similarity(d_1, d_2) = cosine(d_1, d_2) = \frac{(d_1 \bullet d_2)}{\parallel d_1 \parallel \cdot \parallel d_2 \parallel} \tag{2.2}$$

where $\bullet$ represents the vector dot product and $\parallel\ \parallel$ represents the length of a vector.

In sections 2.1 and 2.2, we provide an overview of two major categories of document clustering, hierarchical and partitioning methods. These traditional methods do not address the special problem of high dimensionality in document clustering, but some recently proposed frequent itemsets-based clustering methods do. We briefly explain

them in section 2.3 and examine the difference between these algorithms and ours. Some other clustering methods that are not commonly used for clustering documents are presented in section 2.4 for completeness.

## 2.1 Hierarchical Methods

A hierarchical method works by grouping data objects (documents) into a tree of clusters. A hierarchical method can be further classified into *agglomerative* or *divisive* approach [16, 28].

### 2.1.1 Agglomerative and Divisive Hierarchical Clustering

The *agglomerative* approach builds the hierarchy from bottom-up. It starts with the data objects as individual clusters and successively merges the most similar pair of clusters until all the clusters are merged into one cluster which is the topmost level of the hierarchy. Algorithms in this family follow a similar template as shown in figure 2.1. Note that the similarity between two objects or two clusters can be measured using different methods such as the cosine measure in equation 2.2.

The inter-cluster similarity in step 3 of the algorithm can be computed in different ways [28]. In *single-link clustering*, we consider the similarity between one cluster and another cluster to be equal to the greatest similarity from any member of one cluster to any member of the other cluster. In *complete-link clustering*, we consider the similarity between one cluster and another cluster to be equal to the least similarity from any member of one cluster to any member of the other cluster. In *average-link clustering*, we consider the similarity between one cluster and another cluster to be equal to the average similarity from any member of one cluster to any member of the other cluster. Different agglomerative algorithms employ different similarity measuring schemes. A recent comparison [42] shows that the average-link clustering, UPGMA [16, 28], is the most accurate one in its category.

1. Compute the similarity between all pairs of clusters and store the result in a similarity matrix whose $ij^{th}$ entry denotes the similarity between the $i^{th}$ and the $j^{th}$ clusters.
2. Select the most similar pair of clusters and merge them into a single cluster, i.e., the total number of clusters is reduced by 1.
3. Compute similarities between the new cluster and each of the old clusters
4. Repeat steps 2 and 3 until a single cluster remains.

Figure 2.1: Agglomerative hierarchical clustering algorithm

The *divisive* approach builds the hierarchy from top-down. It starts with all the data objects in the same cluster and iteratively split a cluster into smaller pieces, until only singleton clusters of individual data objects remain or the distance between the two closest clusters is above a certain threshold.

### 2.1.2 Evaluation of Hierarchical Methods

A traditional hierarchical clustering method constructs the hierarchy by subdividing a parent cluster or merging similar children clusters. It usually suffers from its inability to perform adjustment once a merge or split decision has been performed. This inflexibility may lower the clustering accuracy. Furthermore, due to the fact that a parent cluster in the hierarchy always contains all objects of its descendants, this kind of hierarchy is not suitable for browsing. The user may have difficulty to locate her target object in such a large cluster.

Our hierarchical clustering method is completely different. We first form all the clusters by assigning documents to the most similar cluster and then construct the hierarchy based on their inter-cluster similarities. The clusters in the resulting hierarchy are non-overlapping. The parent cluster contains only the general documents

1. Randomly select $k$ data objects as the initial centroids.
2. Assign all data objects to the closest (the most similar) centroid.
3. Recompute the centroid of each cluster.
4. Repeat steps 2 and 3 until the centroids do not change.

Figure 2.2: Basic k-means algorithm

of the topic. If a document belongs to a more specific topic, then it is assigned to a descendant cluster of the parent. This hierarchy is similar to the human-generated Yahoo! subject hierarchy and is more suitable for browsing.

## 2.2 Partitioning Methods

To construct $k$ clusters, a partitioning method creates all $k$ clusters at once and then iteratively improves the partitioning by moving data objects from one group to another. K-means and its variants [14, 28, 33] are the most well-known partitioning methods.

### 2.2.1 The k-means algorithm and its variants

The basic k-means algorithm partitions a set of data objects into $k$ clusters so that the inter-cluster similarity is low but the intra-cluster similarity is high. The algorithm is shown in figure 2.2.

There are many variants of the k-means method. They may be different in the selection of the initial $k$ centroids, the calculation of dissimilarity, and the methods for calculating cluster means. In document clustering, [42] demonstrates that one of the

1. Select a cluster to split. There are several ways to pick which cluster to split, but experiment shows that there is no significant difference in terms of clustering accuracy. Usually, either the largest cluster or the one with the least overall similarity is chosen at this step.
2. Employ the basic k-means algorithm to subdivide the chosen cluster.
3. Repeat step 2 for a constant number of times. Then perform the split that produces the clustering with the highest overall similarity.
4. Repeat the above three steps until the desired number of clusters is reached.

Figure 2.3: Bisecting k-means algorithm

variants, bisecting k-means, outperforms the basic k-means as well as the agglomerative approach in terms of accuracy and efficiency. The bisecting k-means algorithm is illustrated in figure 2.3. Strictly speaking, the bisecting k-means algorithm is a divisive hierarchical clustering method.

### 2.2.2 Evaluation of Partitioning Methods

Both the basic and bisecting k-means algorithms are relatively efficient and scalable. The complexity of both algorithms is linear in the number of documents. In addition, they are so easy to implement that they are widely used in different clustering applications.

A major disadvantage of k-means is that it requires the user to specify $k$, the number of clusters, in advance which may be impossible to estimate in some cases. Incorrect estimation of $k$ may lead to poor clustering accuracy. Also, it is not suitable for discovering clusters of very different size which is very common in document clustering. Moreover, the k-means algorithm is sensitive to noise and outlier data objects as they may substantially influence the mean value, which in turn lower the clustering

accuracy. The k-medoids algorithm [28] is proposed to resolve this problem. Instead of using the mean value as a reference point for clustering, the medoid, which is the most centrally located object in a cluster, can be used. The k-medoids algorithm is more robust than k-means, but it is less efficient than k-means.

A partitioning method aims for flat clustering, but the repeated application of the same method can also provide a hierarchical clustering. Similarly, a hierarchical method can be used to generate a flat partition of $k$ clusters.

## 2.3 Frequent Itemset-based Methods

Both hierarchical and partitioning methods do not really address the problem of high dimensionality in document clustering. Frequent itemset-based clustering method is shown to be a promising approach for high dimensionality clustering in recent literature [9]. It reduces the dimension of a vector space by using only frequent itemsets for clustering.

Frequent itemset extraction is a preliminary step for the clustering methods, including ours, in this category. Therefore, we first briefly mention two well-known methods, Apriori [5, 6] and FP-growth [24], for this purpose. Details of the algorithms can be found in the referenced papers.

### 2.3.1 The Apriori Algorithm

The Aprori algorithm [5, 6] is a well-known method for computing frequent itemsets in a transaction database. Corresponding with the concept of transaction data, we treat documents as transactions, and words in documents as items in transactions. The Apriori algorithm uses a *level-wise search*, where $k$-itemsets are used to explore $(k+1)$-itemsets, to mine frequent itemsets from the database. The generate-and-test approach of the algorithm works well in terms of reducing the candidate set. However, a huge number of candidate itemsets may be generated. Suppose there are $m$

frequent 1-itemsets. Then $\frac{m(m-1)}{2}$ candidate 2-itemsets are generated. In addition, the algorithm requires multiple scans of the entire database to check for frequent itemsets. More specifically, it requires $(n+1)$ scans, where $n$ is the size of the largest frequent $k$-itemset. These bottlenecks may greatly affect the overall efficiency of frequent itemset-based clustering methods. Many variations of the Apriori and frequent itemset extraction algorithm have been proposed [25, 23, 2, 3] to address these weaknesses. Some examples are hash-based technique, transaction reduction, partitioning, sampling, and dynamic itemset counting.

### 2.3.2 The Frequent Pattern-growth Algorithm

To avoid generating a huge set of candidate itemsets as in the Apriori algorithm, [24] presents an efficient frequent itemset extraction algorithm, *Frequent-pattern growth (FP-growth)*. This algorithm adopts a *divide-and-conquer* approach to minimize the candidate generation process to only those most likely to be frequent, and employs a compact prefix-tree data structure, *frequent-pattern tree (FP-tree)*, to avoid repetitive scanning of the database.

The FP-growth algorithm performs exactly two scans of the transaction database and mines on the compact data structure, FP-tree, to find all frequent itemsets without generating all possible candidate sets. [24] shows that FP-growth is about an order of magnitude faster than Apriori in large databases. This gap grows wider when the minimum support threshold reduces. Although the FP-growth algorithm is efficient, sometimes, it is infeasible to construct a main memory-based FP-tree when the database is large, which is very common for the case of document clustering. To create a scalable version of FP-growth, we can first partition the database into a set of projected databases, and then construct an FP-tree and mine it in each projected database [23].

### 2.3.3 Transaction Clustering

[47] introduces a new criterion for clustering transactions using frequent itemsets. The intuition of the criterion is that there should be many frequent items within a cluster and little overlapping of such items across clusters.

In principle, this method can also be applied to document clustering by treating a document as a transaction; however, the method does not create a hierarchy for browsing. The repeated application of the same clustering method on each level of clusters can provide a hierarchy, but the resulting hierarchy, similar to the traditional hierarchical methods, suffers from the problem that the parent cluster contains too many documents. As a result, it is not suitable for browsing.

### 2.3.4 Hierarchical Frequent Term-based Clustering

The recently developed algorithm *Hierarchical Frequent Term-based Clustering* (HFTC) [9] attempts to address the special requirements in document clustering using the notion of frequent itemsets. HFTC suggests that this frequent term-based approach is efficient and the resulting hierarchy is natural for browsing. Although both HFTC and our algorithm are frequent itemsets-based hierarchical clustering algorithms, they are completely different in terms of their clustering criterion, their strategy for identifying clusters, and their hierarchical structure results. HFTC greedily picks up the next frequent itemset (representing the next cluster) to minimize the overlap of the documents that contain both the itemset and some remaining itemsets. The clustering result very much depends on the order of picking up itemsets, which in turn depends on the greedy heuristic used. The resulting clusters are further partitioned applying the same method to build a lattice of overlapping clusters. In our algorithm, we do not follow a sequential order of selecting clusters. Rather, we assign documents to the best clusters with all clusters available. Experiments show that our algorithm produces better clusters and is more scalable.

## 2.4 Other Clustering Methods

The clustering methods presented in this section are not commonly used or in some cases, are not even suitable for clustering documents; however, they play important roles in cluster analysis in data mining. We introduce the spirit behind these methods which may initiate some ideas on document clustering for future research.

### 2.4.1 Density-based Methods

Density-based clustering methods are based on a simple concept: clusters are dense regions in the data space that are separated by regions of lower object density. Their general idea is to continue growing the given cluster as long as the density in the neighborhood exceeds some threshold. In other words, for each data point within a given cluster, the neighborhood of a given radius has to contain at least a minimum number of data points. Methods in this category are good at filtering out outliers and discovering clusters of arbitrary shapes. The well-known algorithms in this category are DBSCAN [18] and OPTICS [8].

### 2.4.2 Grid-based Methods

Grid-based clustering methods quantize the space into a finite number of cells that form a grid structure. Then all of the clustering operations are performed on this grid structure. The computational complexity of all of the previously mentioned clustering methods is at least linearly proportional to the number of objects. The unique property of grid-based clustering approach is that its computational complexity is independent of the number of data objects, but dependent only on the number of cells in each dimension in the quantized space.

STING (STatistical INformation Grid) [49] is a typical grid-based clustering method which divides the spatial area into rectangular cells. The algorithm constructs several levels of such rectangular cells, and these cells form a hierarchical structure; that is, each cell is partitioned to form a number of cells at the next lower level. Figure 2.4

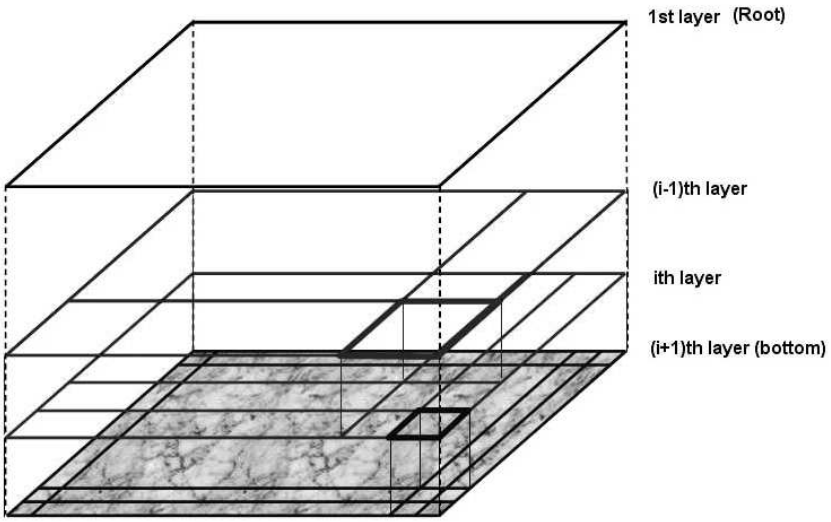


Figure 2.4: A hierarchical structure for STING clustering

illustrates the idea. Statistical information, such as means, maximum, and minimum values, of each grid cell are precomputed and stored for later query processing. The clustering quality of STING highly depends on the the granularity of the lowest level of the grid structure. If the granularity is too coarse, then the accuracy of clustering solution will degrade. However, if the granularity is too fine, the processing time will increase drastically. Another limitation of STING is that it can only represent clusters in either horizontal or vertical rectangular shape. Although this method is efficient, its limitations substantially lower the accuracy of the clustering result.

CLIQUE (CLustering In QUEst) [4] is a hybrid clustering method that combines the idea of both grid-based and density-based approaches. Its goal is to perform clustering on high dimensional data in large databases efficiently. CLIQUE first partitions the $n$-dimensional data space into non-overlapping rectangular units. It attempts to discover the overall distribution patterns of the data set by identifying the sparse and dense units in the space. A unit is dense if the fraction of total data points contained in it exceeds an input model parameter. It explores the space based on a simple

property of the candidate search space: If a k-dimensional unit is dense, then its projections in $(k-1)$-dimensional space are also dense. This heuristic greatly reduces the search space and is the key factor of efficiency of CLIQUE; however, the simplicity of the algorithm often degrades the accuracy of the clustering result.

# Chapter 3

# Constructing Clusters

The agglomerative or partitioning methods are "document-centered" in that the pairwise similarity between documents plays a central role in constructing a cluster. Our method is "cluster-centered" in that we measure the "cohesiveness" of a cluster directly, using frequent itemsets. In this chapter, we first introduce some definitions and then present the cluster construction method.

A *global frequent itemset* refers to a set of items (words) that appear together in more than a user-specified fraction of the document set. A *global frequent item* refers to an item that belongs to some global frequent itemset. The *global support* of an itemset is the percentage of documents containing the itemset. A global frequent itemset containing $k$ items is called a *global frequent k-itemset*.

The main idea of the clustering stage is based on a simple observation: the documents under the same topic should share a set of common words. Some minimum fraction of documents in the document set must contain these common words, and they correspond to the notion of global frequent itemsets which form the basis of the initial clusters. An essential property of frequent itemset is its representation of words that commonly occur together in documents. To illustrate that this property is important for clustering, we consider two global frequent items, "apple" and "window". The documents that contain the word "apple" may discuss about fruits or farming.

The documents that contain the word "window" may discuss about renovation. However, if both words occur together in many documents, then we may identify another topic that discusses about operating systems or computers. By precisely identifying these hidden topics as the first step and then clustering documents based on them, we can improve the accuracy of the clustering solution.

To generate all global frequent itemsets from a document set, we apply the Apriori or the FP-growth algorithm on the document vectors with a user-specified minimum global support. While mining the global frequent itemsets, we treat documents as transactions, and words in documents as items in transactions. Then for each document, we store the weighted frequencies only for global frequent items. We call these frequencies the *feature vector* for the document. This low-dimensional feature vector is used in place of the original high-dimensional document vector. In other words, our vector model is formed by the feature vectors rather than by the document vectors. The reduced dimension is equal to the number of global frequent items. As a result, it significantly improves the efficiency and scalability of all subsequent clustering operations.

**Example 3a**: Consider the twelve documents in table 3.1. They are selected from the *Classic* [13] document set and their document names indicate their natural classes. After applying the Apriori algorithm to the document vectors, we compute the global frequent items: "flow", "form", "layer", "patient", "result", and "treatment". Thus, each document is represented by a feature vector which is supposed to be a vector of inverse document frequencies (IDF), as discussed in Chapter 2. For the purpose of better understandability, however, we use simply the frequency of an item, i.e., the number of occurrences of a word in a document, without applying TF-IDF in the running example in this thesis. For example, the feature vector of document *med*.6 is (0, 0, 0, 9, 1, 1) which represents the frequencies of the global frequent items "flow", "form", "layer", "patient", "result", and "treatment" in document *med*.6 respectively. The twelve feature vectors in table 3.1 form the vector model for our subsequent clustering operations.

| # | **Document name** | **Feature vector** (flow, form, layer, patient, result, treatment) |
|---|---|---|
| 1 | cisi.1 | ( 0 1 0 0 0 0 ) |
| 2 | cran.1 | ( 1 1 1 0 0 0 ) |
| 3 | cran.2 | ( 2 0 1 0 0 0 ) |
| 4 | cran.3 | ( 2 1 2 0 3 0 ) |
| 5 | cran.4 | ( 2 0 3 0 0 0 ) |
| 6 | cran.5 | ( 1 0 2 0 0 0 ) |
| 7 | med.1 | ( 0 0 0 8 1 2 ) |
| 8 | med.2 | ( 0 1 0 4 3 1 ) |
| 9 | med.3 | ( 0 0 0 3 0 2 ) |
| 10 | med.4 | ( 0 0 0 6 3 3 ) |
| 11 | med.5 | ( 0 1 0 4 0 0 ) |
| 12 | med.6 | ( 0 0 0 9 1 1 ) |

Table 3.1: Document set

| **Global frequent itemset** | **Global support** |
|---|---|
| {flow} | 42% |
| {form} | 42% |
| {layer} | 42% |
| {patient} | 50% |
| {result} | 42% |
| {treatment} | 42% |
| {flow, layer} | 42% |
| {patient, treatment} | 42% |

Table 3.2: Global frequent itemsets
(minimum global support = 35%)

Table 3.2 specifies all the global frequent $k$-itemsets with their global supports. For example, the global support of the global frequent item {patient} is 50% because half of the documents in the set contain the item "patient". In this example, an itemset is frequent only if its global support is larger than or equal to 35%.

The cluster construction has two steps: constructing initial clusters and making clusters disjoint.

## 3.1 Constructing Initial Clusters

An initial cluster is constructed for each global frequent itemset. All the documents containing this itemset are included in the same cluster. Since a document usually contains more than one global frequent itemset, the same document may appear in multiple initial clusters, i.e., initial clusters are overlapping. The purpose of initial clusters is to ensure the property that all the documents in a cluster contain all the items in the global frequent itemset that defines the cluster. These items can be considered as the mandatory items for every document in the cluster. We use this global frequent itemset as the *cluster label* to identify the cluster. The cluster label has two other purposes. First, it establishes the hierarchical structure in the tree construction stage. Second, it is presented to the user to facilitate browsing. We remove the overlapping of clusters in section 3.2.

**Example 3b**: For each global frequent itemset in table 3.2, we construct a cluster where its cluster label is formed by the items in the corresponding global frequent itemset. For example, the cluster label of $C(patient, treatment)$ is $\{patient, treatment\}$. Let us use document *med*.6 to illustrate how to construct the initial clusters. Document *med*.6 appears in clusters $C(patient, treatment)$, $C(patient)$, $C(result)$, and $C(treatment)$ because it contains all the global frequent itemsets of these clusters, i.e., it contains all the cluster labels of these clusters. The initial clusters are shown in table 3.3. The third column is explained in the next example.

| **Cluster** | **Documents in cluster** | **Cluster frequent items & their cluster supports (CS)** |
|---|---|---|
| C(flow) | cran.1, cran.2, cran.3, cran.4, cran.5 | {flow, CS=100%}, {layer, CS=100%} |
| C(form) | cisi.1, cran.1, cran.3, med.2, med.5 | {form, CS=100%} |
| C(layer) | cran.1, cran.2, cran.3, cran.4, cran.5 | {layer, CS=100%}, {flow, CS=100%} |
| C(patient) | med.1, med.2, med.3, med.4, med.5, med.6 | {patient, CS=100%}, {treatment, CS=83%} |
| C(result) | cran.3, med.1, med.2, med.4, med.6 | {result, CS=100%}, {patient, CS=80%}, {treatment, CS=80%} |
| C(treatment) | med.1, med.2, med.3, med.4, med.6 | {treatment, CS=100%}, {patient, CS=100%}, {result, CS=80%} |
| C(flow, layer) | cran.1, cran.2, cran.3, cran.4, cran.5 | {flow, CS=100%}, {layer, CS=100%} |
| C(patient, treatment) | med.1, med.2, med.3, med.4, med.6 | {patient, CS=100%}, {treatment, CS=100%}, {result, CS=80%} |

Table 3.3: Initial clusters
(minimum cluster support = 70%)

## 3.2 Making Clusters Disjoint

Each document belongs to one or more initial clusters (all documents that do not belong to any initial cluster are assigned to a "null" cluster). Therefore, initial clusters overlap. In this step, we assign a document to the "best" initial cluster so that each document belongs to exactly one cluster. This step also guarantees that every document in the cluster still contains the mandatory items (i.e., the items in the cluster label).

Intuitively, an initial cluster $C_i$ is good for a document $doc_j$ if there are many

global frequent items in $doc_j$ that appear in many documents in $C_i$. Thus, we can consider the set of frequent items of each cluster as a reference point for the cluster, and then use these *cluster frequent items* for clustering similar documents. Formally, we say that an item $x$ is *cluster frequent* in a cluster $C_i$ if $x$ is contained in some minimum fraction of documents in $C_i$. The *cluster support* of $x$ in $C_i$ is the percentage of the documents in $C_i$ that contain $x$. Example 3c illustrates how to compute cluster frequent items from initial clusters.

**Example 3c**: The third column in table 3.3 shows the cluster frequent items and their cluster supports for each initial cluster. For example, the cluster frequent items of cluster $C(patient, treatment)$ are "patient", "treatment", and "result". Both "patient" and "treatment" have cluster supports 100% because all the documents in the cluster contain these items. The cluster support of "result" is 80% because four out of the five documents contain this item.

Equation 3.1 measures the goodness of an initial cluster $C_i$ for a document $doc_j$. To make clusters non-overlapping, we assign each $doc_j$ to the initial cluster $C_i$ of the highest $score_i$. After this assignment, each document belongs to exactly one cluster.

$$Score(C_i \leftarrow doc_j) = [\sum_{x} n(x) * cluster\_support(x)] - [\sum_{x'} n(x') * global\_support(x')] \tag{3.1}$$

where $x$ represents a global frequent item in $doc_j$ and the item is also cluster frequent in $C_i$, $x'$ represents a global frequent item in $doc_j$ that is not cluster frequent in $C_i$, $n(x)$ is the weighted frequency of $x$ in the feature vector of $doc_j$, and $n(x')$ is the weighted frequency of $x'$ in the feature vector of $doc_j$.

The weighted frequencies $n(x)$ and $n(x')$ are defined by the standard inverse document frequency (TF-IDF) of items $x$ and $x'$ respectively, as discussed in Chapter 2. Let us explain the rationale behind the score function. The first term of the function

rewards cluster $C_i$ if a global frequent item $x$ in $doc_j$ is cluster frequent in $C_i$. In order to capture the importance (weight) of item $x$ in different clusters, we multiply the frequency of $x$ in $doc_j$ by its cluster support in $C_i$. The second term of the function penalizes cluster $C_i$ if a global frequent item $x'$ in $doc_j$ is not cluster frequent in $C_i$. The frequency of $x'$ is multiplied by its global support which can be viewed as the importance of $x'$ in the entire document set or as the weight of the penalty on this item. This part encapsulates the concept of dissimilarity into the score.

A unique property of our score function is that the local frequency, i.e., the number of occurrences of an item in a document, is taken into account as part of the clustering criterion. This is different from other frequent itemset-based document clustering methods [9, 47] where only the presence or the absence of an item in a document is considered, but the local frequency, which is an important piece of information, is not utilized. To understand why the local frequency is crucial, consider a global frequent item "tennis" that appears twenty times in document $doc_k$ and a global frequent item "soccer" that appears only once in $doc_k$. Suppose there are two clusters: one is about tennis, and another one is about soccer. If the frequency of an item in $doc_k$ is ignored, then both global frequent items are considered to be equally important. Nevertheless, it is more sensible to classify $doc_k$ into the "tennis" cluster, rather than the "soccer" cluster. This important insight is encapsulated in our score function.

**Example 3d**: Consider table 3.3 again. To find the most suitable cluster for document $med.6$, for example, we need to calculate its scores against each of its initial cluster:

$$Score(C(patient) \leftarrow med.6) = 9 * 1 + 1 * 0.83 - 1 * 0.42 = 9.41$$
$$Score(C(result) \leftarrow med.6) = 10.6$$
$$Score(C(treatment) \leftarrow med.6) = 10.8$$
$$Score(C(patient, treatment) \leftarrow med.6) = 10.8$$

| **Cluster** | **Documents in cluster** | **Cluster frequent items & their cluster supports (CS)** |
|---|---|---|
| C(flow) | cran.1, cran.2, cran.3, cran.4, cran.5 | {flow, CS=100%}, {layer, CS=100%} |
| C(form) | cisi.1 | {form, CS=100%} |
| C(layer) | | none |
| C(patient) | med.5 | {patient, CS=100%}, {treatment, CS=83%} |
| C(result) | | none |
| C(treatment) | | {treatment, CS=100%}, {patient, CS=100%}, {result, CS=80%} |
| C(flow, layer) | | none |
| C(patient, treatment) | med.1, med.2, med.3, med.4, med.6 | {patient, CS=100%}, {treatment, CS=100%}, {result, CS=80%} |

Table 3.4: Disjoint clusters

We use $Score(C(patient) \leftarrow med.6)$ to explain the calculation. The global frequent items in $med.6$ are "patient", "result", and "treatment". Their frequencies in the feature vector are 9, 1, and 1 respectively. "Patient" and "treatment" are cluster frequent in cluster $C(patient)$; hence these two items appear in the rewarding part of the function and their frequencies are multiplied by their corresponding cluster supports 1 and 0.83 respectively. "Result" is not cluster frequent in cluster $C(patient)$; therefore, it appears in the penalty part and its frequency is multiplied by its global support 0.42.

After computing the scores against each of its initial cluster, both clusters $C(treatment)$ and $C(patient, treatment)$ get the same highest score. Document $med.6$ is assigned to $C(patient, treatment)$, which has a larger number of items in its cluster label, i.e., a cluster with a more specific topic. After assigning each document to a cluster, table 3.4 shows the disjoint clusters. Ignore the third column at this moment.

There is an important difference between the cluster label and the set of cluster frequent items associated with a cluster. A cluster label is a set of mandatory items in the cluster because every document in the cluster must contain all the items in the label. On the other hand, a cluster frequent item is required to appear in some fraction of documents in the cluster. We shall use the cluster label as the identity of a cluster and the set of cluster frequent items as the topic description of a cluster.

After assigning all documents to their best initial clusters, we need to recompute the cluster frequent items for each cluster in order to reflect the updated clustering. While re-computing the cluster frequent items of cluster $C_i$, we also include the documents in all of its potential descendants, whose cluster labels are the superset of $C_i$'s label. The intuition is that potential descendants are likely to be subtopics of a parent; therefore, it is sensible to include them.

**Example 3e**: The third column in table 3.4 reflects the updated cluster frequent items in the non-overlapping clusters. The potential descendant of cluster $C(patient)$ is cluster $C(patient, treatment)$. While recomputing the cluster frequent items of $C(patient)$, we would consider all the documents in both $C(patient, treatment)$ and $C(patient)$. The cluster support of the item "treatment" in cluster $C(patient)$ is 83% because five out of the six documents contain this item.

# Chapter 4

# Building the Cluster Tree

The set of clusters produced by the previous stage can be viewed as a set of topics and subtopics in the document set. In this section, a cluster (topic) tree is constructed based on the similarity among clusters. In case a tree contains too many clusters, two pruning methods are applied to efficiently shorten and narrow a tree by merging similar clusters together.

## 4.1 Tree Construction

In this section, we explain how to construct a non-overlapping hierarchical cluster tree. The resulting cluster tree has two objectives: to form a foundation for pruning and to provide a logical structure for browsing. Each cluster has exactly one parent. The topic of a parent cluster is more general than the topic of a child cluster and they must be similar to a certain degree.

Recall that each cluster uses one global frequent $k$-itemset as its cluster label. Such clusters are called *$k$-clusters* in the following. In the cluster tree, the root node, which collects the unclustered documents and has cluster label "null", constitutes level 0. The 1-clusters appear in level 1 of the tree, and so forth for every level. The depth of the tree is equal to the size of the largest global frequent $k$-itemsets.

```
Sort all clusters by the number of items in their cluster labels in descending order;
For each cluster C_i in the list {
   // remove empty leaf node
   If C_i contains no document and it has no children clusters then {
      Skip this empty cluster C_i, and try cluster C_{i+1};
   }

   // identify all potential parents
   k = the number of items in C_i's cluster label;
   PotentialParents = Find all clusters containing cluster label with k − 1 items and
   the cluster label is a subset of C_i's cluster label;

   // choose the most similar parent
   doc(C_i) = Merge all documents in the subtree C_i into a single combined document;
   Compute the scores of doc(C_i) against each PotentialParents;
   Set the potential parent cluster that has the highest score to be the parent of C_i;
}
```

Figure 4.1: Tree construction algorithm

Figure 4.1 illustrates the tree construction algorithm. As $k$-clusters always appear in a higher level than $(k-1)$-clusters in a tree, we can build a tree bottom-up by choosing a parent for each cluster starting from the highest level. Given that a cluster label represents the mandatory items in a cluster, we can construct a natural hierarchy based on these labels as follows. For each $k$-cluster $C_i$, we identify all potential parents which are $(k-1)$-clusters and have the cluster label being a subset of $C_i$'s cluster label. The next step is to choose the "best" parent among these potential parents. The criterion for selecting the best parent is similar to choosing the best cluster for a document in section 3.2. We first merge all the documents in the subtree of $C_i$ into a single conceptual document $doc(C_i)$, and then compute the score of $doc(C_i)$ against each potential parent. The one which has the highest score would become the parent of $C_i$. In actual implementation, the operation of merging documents into $doc(C_i)$ can be accomplished efficiently by adding up all the feature vectors in the clusters. Note that all empty leaf nodes are removed during the tree construction.

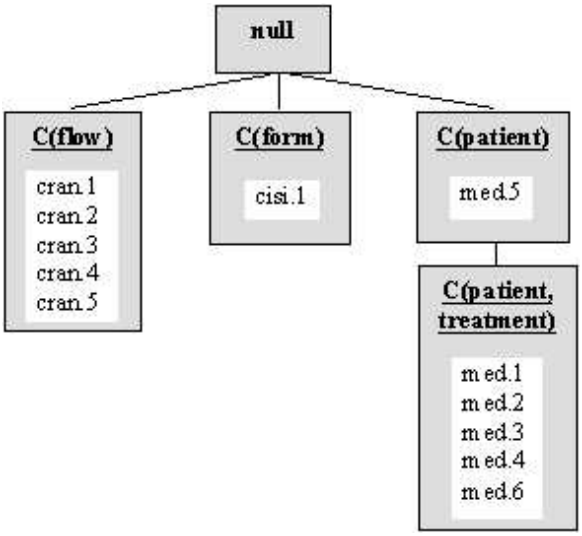


Figure 4.2: Cluster tree built from table 3.4

It is possible that a small fraction of non-leaf nodes in the tree are empty clusters. Experiments show that these empty non-leaf nodes often have many children clusters containing many documents. This situation occurs when documents under a topic are well categorized into subtopics. An empty node may serve as a good intermediate node for organizing subtopics under the same category, which in turn ease browsing. Thus, these empty non-leaf nodes should be kept. In the next pruning step, some of them will be pruned or become non-empty.

**Example 4a**: Consider the clusters in table 3.4. We start to build the tree from 2-clusters (i.e., clusters with 2-itemsets as the cluster label). Cluster $C(flow, layer)$ is removed since it is an empty leaf node. Next, we select a parent for $C(patient, treatment)$. The potential parents are $C(patient)$ and $C(treatment)$. $C(patient)$ gets a higher score and becomes the parent of $C(patient, treatment)$. Figure 4.2 depicts the resulting cluster tree.

## 4.2 Tree Pruning

A cluster tree can be broad and deep, especially when a low minimum global support is used. Therefore, it is likely that documents of the same topic are distributed over several small clusters, which would lead to poor clustering accuracy. The aim of tree pruning is to merge similar clusters in order to produce a natural topic hierarchy for browsing and to increase the clustering accuracy. Before introducing the pruning methods, we will first define the *inter-cluster similarity*, which is a key notion for merging clusters.

To measure the inter-cluster similarity between two clusters $C_a$ and $C_b$, we measure the similarity of $C_b$ to $C_a$, and the similarity of $C_a$ to $C_b$. The idea is to treat one cluster as a document (by combining all the documents in the cluster) and measure its score against another cluster using our score function defined in equation 3.1. The only difference is that the score has to be normalized to avoid the effect of varying document sizes. Formally, the similarity of $C_j$ to $C_i$ is defined as:

$$Sim(C_i \leftarrow C_j) = \frac{Score(\ C_i \leftarrow doc(C_j)\ )}{\sum_x n(x) + \sum_{x'} n(x')} + 1 \tag{4.1}$$

where $C_i$ and $C_j$ are two clusters; $doc(C_j)$ stands for combining all the documents in the subtree of $C_j$ into a single document; $x$ represents a global frequent item in $doc(C_j)$ that is also cluster frequent in $C_i$; $x'$ represents a global frequent item in $doc(C_j)$ that is not cluster frequent in $C_i$; $n(x)$ is the weighted frequency of $x$ in the feature vector of $doc(C_j)$; $n(x')$ is the weighted frequency of $x'$ in the feature vector of $doc(C_j)$.

To explain the normalization by $\sum_x n(x) + \sum_{x'} n(x')$, notice that the global support and cluster support in the score function are always between 0 and 1. Thus, the maximum value of the score is $\sum_x n(x)$ and the minimum value of the score is $-\sum_{x'} n(x')$. We can normalize the score by dividing it by $\sum_x n(x) + \sum_{x'} n(x')$, and the normalized score is within the range of [-1,1]. To avoid negative similarity values,

```
Scan the tree from bottom-up;
For each non-leaf node C_i in the tree {
   Calculate the Inter_Sim of C_i with each of its children including their
   descendants;
   Prune the child cluster if the Inter_Sim is above 1;
}
```

Figure 4.3: Child pruning algorithm

we add the term +1. As a result, the range of the $Sim$ function is [0,2]. We define the inter-cluster similarity between $C_a$ and $C_b$ as the geometric mean of two normalized scores $Sim(C_a \leftarrow C_b)$ and $Sim(C_b \leftarrow C_a)$:

$$Inter\_Sim(C_a \leftrightarrow C_b) = [Sim(C_a \leftarrow C_b) * Sim(C_b \leftarrow C_a)]^{\frac{1}{2}} \tag{4.2}$$

where $C_a$ and $C_b$ are two clusters including their descendants; $Sim(C_a \leftarrow C_b)$ is the similarity of $C_b$ against $C_a$; $Sim(C_b \leftarrow C_a)$ is the similarity of $C_a$ against $C_b$.

The advantage of the geometric mean is that two clusters are considered to be similar only if both values of $Sim(C_a \leftarrow C_b)$ and $Sim(C_b \leftarrow C_a)$ are high. Given that the range of $Sim$ function is [0,2], the range of $Inter\_Sim$ function is also [0,2]. Higher values imply higher similarity between two clusters. An $Inter\_Sim$ value below 1 implies the weight of dissimilar items has exceeded the weight of similar items. Hence, the $Inter\_Sim$ value of 1 serves as a good threshold in distinguishing whether two clusters are similar.

We now present two pruning methods.

### 4.2.1 Child Pruning

The objective of child pruning is to efficiently shorten a tree by replacing child clusters by their parent. The pruning criterion is based on the inter-cluster similarity between

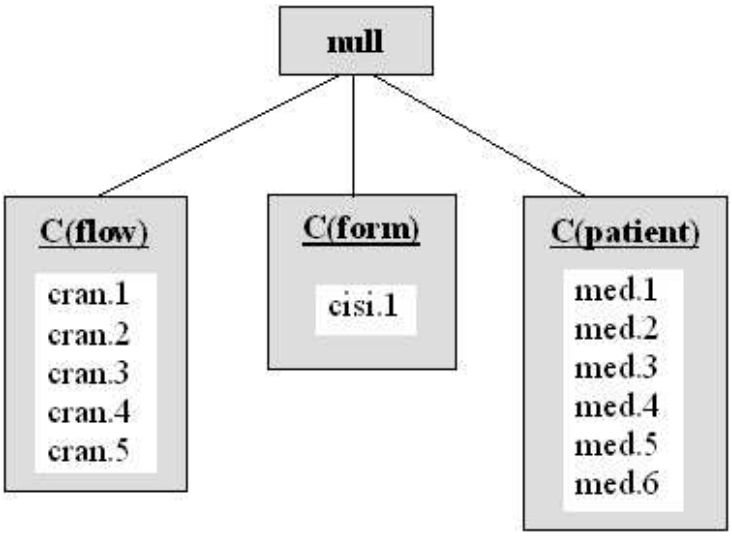


Figure 4.4: Cluster tree after child pruning

a parent and its child. A child is pruned only if it is similar to its parent; therefore, child pruning does not degrade the purity of the parent. The rationale behind this method is that when a subtopic (e.g. tennis ball) is very similar to its parent topic (e.g. tennis), then the subtopic is probably too specific and can be removed.

The procedure is presented in figure 4.3. It scans a tree from bottom-up. For each non-leaf node, we calculate the $Inter\_Sim$ with each of its children together with their descendants and prune the child cluster if the $Inter\_Sim$ is above 1. When a cluster is pruned, its children become the children of their grandparent. Note that child pruning is only applicable from level 2 down to the bottom of a tree since it is illogical to compare clusters at level 1 with the root, which collects unclustered documents.

**Example 4b**: Consider figure 4.2. To determine whether cluster $C(patient, treatment)$ should be pruned, the inter-cluster similarity between $C(patient)$ and $C(patient, treatment)$ is calculated as follows:

$Sim(C(patient) \leftarrow C(patient, treatment))$
$= (30 * 1 + 9 * 0.83 - 1 * 0.42 - 8 * 0.42) \,/\, 48 + 1 = 1.70$
$Sim(C(patient, treatment) \leftarrow C(patient))$
$= (34 * 1 + 8 * 0.8 + 9 * 1 - 2 * 0.42) \,/\, 53 + 1 = 1.92$
$Inter\_Sim(C(patient) \leftarrow C(patient, treatment))$
$= (1.70 * 1.92)^{\frac{1}{2}} = 1.81$

To calculate $Sim(C(patient) \leftarrow C(patient, treatment))$, we combine all the documents in cluster $C(patient, treatment)$ by adding up their feature vectors. The summed feature vector is (0, 1, 0, 30, 8, 9). Then we calculate the score of this combined document against $C(patient)$ and normalize the score by the sum of the frequencies which is 48. $Sim(C(patient, treatment) \leftarrow C(patient))$ is computed using the same method. Since the inter-cluster similarity is above 1, cluster $C(parent, treatment)$ is pruned. See figure 4.4.

### 4.2.2 Sibling Merging

Sibling merging narrows a tree by merging similar subtrees at level 1. It resolves the problem that a natural class may split into different subtrees. It is a key factor for yielding high clustering accuracy.

Figure 4.5 shows the sibling merging algorithm. The procedure is to calculate the $Inter\_Sim$ for each pair of clusters at level 1 of a tree. We then keep merging the cluster pair that has the highest $Inter\_Sim$ until the user-specified number of clusters is reached. In case a user has not specified the desired number of clusters, the algorithm would terminate when all cluster pairs have $Inter\_Sim$ below or equal to 1. The pairwise comparison ensures that only similar clusters are merged. This often becomes a scalability bottleneck in agglomerative algorithms. However, in our algorithm, the number of non-empty clusters at level 1 is always limited by the number of global frequent items. Thus, it does not affect the scalability of our method. Applying this

```
For each pair of clusters at level 1 of the tree {
   Calculate the Inter_Sim together with their descendants;
   Store the result in a matrix;
}

Repeat {
   Select the cluster pair that has the highest Inter_Sim;
   Merge the smaller cluster into the larger cluster with their descendants;
   Update the inter–cluster similarity matrix;
} Until the user specified number of clusters are left;
```

Figure 4.5: Sibling merging algorithm

method to every level in the tree is too computationally expensive. The child pruning method, however, can efficiently achieve the same goal by pruning similar children to their parent.

Figure 4.6 demonstrates how the merge should be performed. Suppose we would like to merge cluster $C(b)$ into cluster $C(a)$. All the documents of $C(b)$ are moved into $C(a)$ and all children clusters of $C(b)$ are moved under $C(a)$.

**Example 4c**: Consider the tree in figure 4.4. Sibling merging computes the $Inter\_Sim$ for each pair of clusters at the level 1 as in table 4.1. If the user has not specified the desired number of clusters, then FIHC would terminate and return the tree as in figure 4.4. Suppose the user has specified the number of clusters to 2. Then the algorithm would prune one cluster at level 1 based on the inter-cluster similarity among clusters $C(flow)$, $C(form)$, and $C(patient)$. Since $C(flow)$ and $C(form)$ is the pair with the highest $Inter\_Sim$, the smaller cluster $C(form)$ would merge with the larger cluster $C(flow)$. Figure 4.7 depicts the resulting tree.

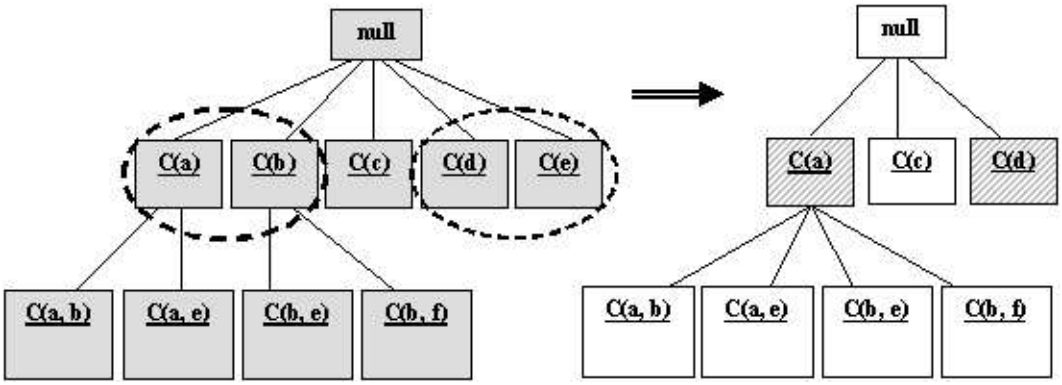


Figure 4.6: Sibling merged tree
Note: a, b, c, d, e, and f are global frequent items. They form the cluster labels.

| $Clusterpair(C_i, C_j)$ | $Sim(C_j \leftarrow C_i)$ | $Sim(C_i \leftarrow C_j)$ | $Inter\_Sim(C_i \leftrightarrow C_j)$ |
|---|---|---|---|
| C(flow) & C(form) | 0.71 | 0.58 | 0.64 |
| C(flow) & C(patient) | 0.58 | 0.54 | 0.56 |
| C(form) & C(patient) | 0.58 | 0.58 | 0.58 |

Table 4.1: Inter-cluster similarity calculation

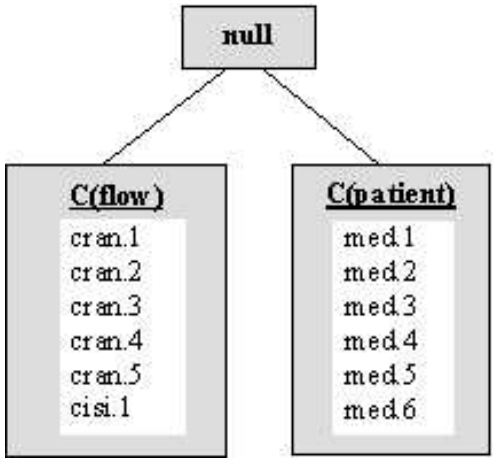


Figure 4.7: Cluster tree after child pruning and sibling merging

# Chapter 5

# Experimental Evaluation

This section presents the experimental evaluation of our method (FIHC) and compares its result with several popular document clustering algorithms, agglomerative UPGMA [16, 28], bisecting k-means [16, 28, 42], and HFTC [9]. We make use of the CLUTO-2.0 Clustering Toolkit [27] to generate the results of UPGMA and bisecting k-means. For HFTC, we obtained the original Java program from the author and then compiled the program into Windows native code to avoid the overhead of the Java Virtual Machine. All algorithms, except HFTC, employ IDF as a preprocessing step. HFTC applies its own preprocessing technique, term frequency variance selection. We use the Apriori algorithm to extract global frequent itemsets in both HFTC and our method. The produced results are then fetched into the same evaluation program to ensure fair comparison across all algorithms.

## 5.1 Data Sets

Five data sets which have been widely used in document clustering research [42, 9] were used for our evaluation. They are heterogeneous in terms of document size, cluster size, number of classes, and document distribution. Their general characteristics are summarized in table 5.1. The smallest of these data sets contained 1,504 documents and the largest contained 8,649 documents. To ensure diversity in the data sets, we obtained them from different sources. For all data sets, we applied a stop-list to

| Data Set | Number of Documents | Number of Classes | Class Size | Average Class Size | Number of Terms |
|---|---|---|---|---|---|
| *Classic4* | 7094 | 4 | 1033 – 3203 | 1774 | 12009 |
| *Hitech* | 2301 | 6 | 116 – 603 | 384 | 13170 |
| *Re0* | 1504 | 13 | 11 – 608 | 116 | 2886 |
| *Reuters* | 8649 | 65 | 1 – 3725 | 131 | 16641 |
| *Wap* | 1560 | 20 | 5 – 341 | 78 | 8460 |

Table 5.1: Summary descriptions of data sets

remove common words, and the words were stemmed using Porters suffix-stripping algorithm [37]. Each document is pre-classified into a single topic, i.e., a natural class. The class information is utilized in the evaluation method for measuring the accuracy of the clustering result. During the cluster construction, the class information is hidden from all clustering algorithms.

The *Classic4* data set is combined from the four classes CACM, CISI, CRAN, and MED abstracts [13]. It was widely used to evaluate various information retrieval systems in the past. The *Hitech* data set was derived from the San Jose Mercury newspaper articles that are distributed as part of the TREC collection [43]. It contains documents about computers, electronics, health, medical, research, and technology. The *Wap* data set was originally from the WebAce project [22]. Each document corresponds to a web page listed in the Yahoo! subject hierarchy [50]. A recent research [35] uses this *Wap* data set to represent the characteristics of web pages in a comprehensive comparison of document clustering algorithms. Data sets *Reuters* and *Re0* were extracted from newspaper articles [34]. For both data sets, we only use the articles that are uniquely assigned to exactly one topic for evaluation purpose. All of these data sets, except *Reuters*, can also be obtained from [27].

## 5.2 Evaluation Method: F-measure

A commonly used external measurement, the F-measure [33, 42], is employed to evaluate the accuracy of the produced clustering solutions. It is a standard evaluation method for both flat and hierarchical clustering structures. It produces a balanced measure of precision and recall. We treat each cluster as if it were the result of a query and each class as if it were the relevant set of documents for a query. The *recall*, *precision*, and *F-measure* for natural class $K_i$ and cluster $C_j$ are calculated as follows:

$$Recall(K_i, C_j) = \frac{n_{ij}}{|K_i|} \tag{5.1}$$

$$Precision(K_i, C_j) = \frac{n_{ij}}{|C_j|} \tag{5.2}$$

where $n_{ij}$ is the number of members of class $K_i$ in cluster $C_j$.

$$F(K_i, C_j) = \frac{2 * Recall(K_i, C_j) * Precision(K_i, C_j)}{Recall(K_i, C_j) + Precision(K_i, C_j)} \tag{5.3}$$

$F(K_i, C_j)$ represents the quality of cluster $C_j$ in describing class $K_i$. While computing $F(K_i, C_j)$ in a hierarchical structure, all the documents in the subtree of $C_j$ are considered as the documents in $C_j$. The *overall F-measure*, $F(C)$, is the weighted sum of the maximum F-measure of all the classes as defined below:

$$F(C) = \sum_{K_i \in K} \frac{|K_i|}{|D|} \, max_{C_j \in C}\{ \, F(K_i, C_j) \, \} \tag{5.4}$$

where $K$ denotes the set of natural classes; $C$ denotes all clusters at all levels; $|K_i|$ denotes the number of documents in class $K_i$; and $|D|$ denotes the total number of documents in the data set.

Taking the maximum of $F(K_i, C_j)$ can be viewed as selecting the cluster that can best describe a given class, and $F(C)$ is the weighted sum of the F-measure of these best clusters. The range of $F(C)$ is [0,1]. A larger $F(C)$ value indicates a higher accuracy of clustering.

## 5.3 Experimental Results

We evaluated our algorithm, FIHC, and its competitors in terms of accuracy, sensitivity to parameters, efficiency and scalability. Recent research in [42] shows that UPGMA and bisecting k-means are the most accurate clustering algorithms in their categories. We also compared FIHC with another frequent itemset-based algorithm, HFTC [9].

### 5.3.1 Accuracy

Table 5.2 shows the F-measure values for all four algorithms with different user-specified numbers of clusters. Since HFTC does not take the number of clusters as an input parameter, we use the same minimum support, from 3% to 6%, for both HFTC and our algorithm in each data set to ensure fair comparison.

Our algorithm, FIHC, apparently outperforms all other algorithms in terms of accuracy. Although UPGMA and bisecting k-means perform slightly better than FIHC in several cases, we argue that the exact number of clusters in a document set is usually unknown in real world clustering problem, and FIHC is robust enough to produce consistently high quality clusters for a wide range number of clusters. This fact is reflected by taking the average of the F-measure values over the different numbers of clusters. Due to the pairwise similarity comparison in agglomerative algorithms, UPGMA is not scalable for large data sets. It fails to provide a clustering solution even after it has consumed all of the main memory. Hence, some experiment results could not be generated for UPGMA.

| **Data Set (# of natural classes)** | **# of Clusters** | **Overall F-measure** | | | |
|---|---|---|---|---|---|
| | | **FIHC** | **UPGMA** | **Bi. k-means** | **HFTC** |
| *Classic4* (4) | 3 | 0.62* | × | 0.59 | n/a |
| | 15 | 0.52* | × | 0.46 | n/a |
| | 30 | 0.52* | × | 0.43 | n/a |
| | 60 | 0.51* | × | 0.27 | n/a |
| | Average | 0.54 | × | 0.44 | 0.61* |
| *Hitech* (6) | 3 | 0.45 | 0.33 | 0.54* | n/a |
| | 15 | 0.42 | 0.33 | 0.44* | n/a |
| | 30 | 0.41 | 0.47* | 0.29 | n/a |
| | 60 | 0.41* | 0.40 | 0.21 | n/a |
| | Average | 0.42* | 0.38 | 0.37 | 0.37 |
| *Re0* (13) | 3 | 0.53* | 0.36 | 0.34 | n/a |
| | 15 | 0.45 | 0.47* | 0.38 | n/a |
| | 30 | 0.43* | 0.42 | 0.38 | n/a |
| | 60 | 0.38* | 0.34 | 0.28 | n/a |
| | Average | 0.45* | 0.40 | 0.34 | 0.43 |
| *Reuters* (65) | 3 | 0.58* | × | 0.48 | n/a |
| | 15 | 0.61* | × | 0.42 | n/a |
| | 30 | 0.61* | × | 0.35 | n/a |
| | 60 | 0.60* | × | 0.30 | n/a |
| | Average | 0.60* | × | 0.39 | 0.49 |
| *Wap* (20) | 3 | 0.40* | 0.39 | 0.40* | n/a |
| | 15 | 0.56 | 0.49 | 0.57* | n/a |
| | 30 | 0.57 | 0.58* | 0.44 | n/a |
| | 60 | 0.55 | 0.59* | 0.37 | n/a |
| | Average | 0.52* | 0.51 | 0.45 | 0.35 |

Table 5.2: F-measure comparison
× = not scalable to run * = best competitor

| **Frequent items in *CRANFIELD*** [13] | |
|---|---|
| **Natural class** | **FIHC cluster** |
| aerodynamic, aircraft, **angle**, **boundary**, **effect**, **flow**, ft, height, **layer**, maximum, measurement, **number**, **present**, **pressure**, shape, **speed**, system, stream, **theory**, value | **angle**, approximate, body, **boundary**, calculate, condition, distribution, **effect**, equation, experiment, **flow**, investigation, **layer**, machine, method, **number**, **present**, **pressure**, **speed**, surface, **theory**, velocity |

Table 5.3: Comparison on class/cluster frequent items

To demonstrate that the cluster labels determined by FIHC are indeed meaningful, we further extend the F-measure evaluation method by using the idea of cluster frequent items. A set of cluster frequent items is similar to a set of keywords within a topic. We first compute a set of frequent items from a natural class, and then compute a set of cluster frequent items from the corresponding cluster that has the highest F-measure value. Then we compare these two sets of frequent items and calculate the percentage of overlapped items.

We use the *CRANFIELD* class from the *Classic* [13] document set to illustrate the idea. *CRANFIELD* documents are abstracts from aeronautical system papers. Table 5.3 shows two sets of frequent items. The items in the left column are extracted from the labeled natural class. The items in the right column are extracted from the corresponding cluster. We observe that many items (in bold font) overlap which implies the cluster can truly reflect the natural class. Another interesting observation is that the cluster frequent items also capture some keywords that are not shown in the natural class but are definitely reasonable to appear under this topic. For example, the items "body", "machine", "surface", and "velocity" are related to aeronautical system. However, the algorithm also misses some important items, such as, "aerodynamic" and "aircraft". To qualify this, 50% of the frequent items in the *CRANFIELD*

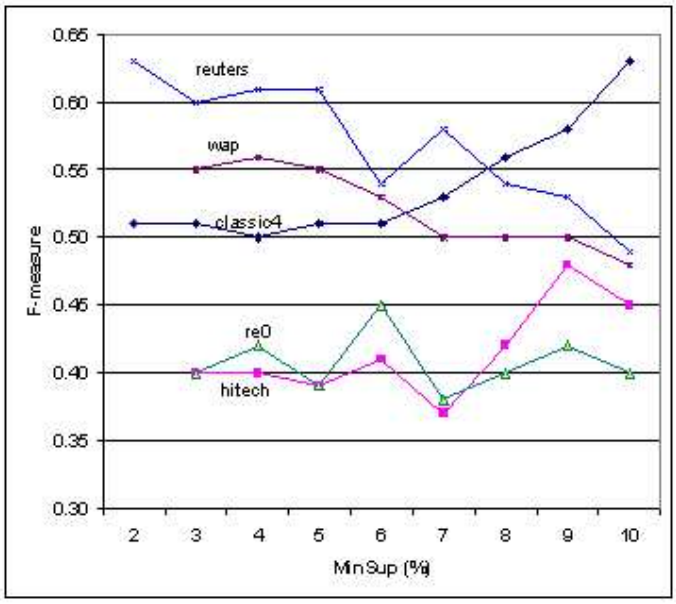


Figure 5.1: Sensitivity to MinSup without pre-specifying # of clusters

class are captured by the corresponding cluster. We obtain similar capturing rates for other three classes in *Classic*, as well as in other document sets.

### 5.3.2 Sensitivity to Parameters

Our algorithm, FIHC, allows for two input parameters: MinSup is the minimum support for global frequent itemset generation and is a mandatory input; KClusters is the number of clusters at level 1 of the tree and is an optional input. Table 5.2 does not only demonstrate the accuracy of the produced solutions, but also shows the sensitivity of the accuracy to KClusters. Both UPGMA and our algorithm are insensitive to KClusters, but bisecting k-means is not. For example, in the *Reuters* document set, the range of F-measure values of our algorithm is 0.58 to 0.61 for different number of KClusters while the range of F-measure values of bisecting k-means is 0.30 to 0.48. In other words, the accuracy of bisecting k-means highly depends on the input parameter, KClusters. This is a very common weakness for traditional clustering methods.

Figure 5.1 depicts the F-measure values of FIHC with respect to MinSup without pre-specifying a value for KClusters. We observe that high clustering accuracy is fairly consistent while MinSup is set between 3% and 9%. As KClusters is unspecified in this case, the sibling merging algorithm has to decide the most appropriate number of output clusters, but this number may vary for different MinSup. This explains why the accuracy fluctuates within a small range of F-measure values for different MinSup.

A general guidance drawn from numerous experiments is: If a data set contains less than 5000 documents, then MinSup should be set between 5% and 9%; otherwise, MinSup should be set between 3% and 5%. However, we would like to emphasize that MinSup should not be treated as a parameter for finding optimal accuracy. Instead, it allows user to adjust the shape of the cluster tree. If the value of MinSup is small, then the tree is broad and deep, and vice versa.

Another threshold is the minimum cluster support, which distinguishes whether an item is cluster frequent. Experiments show that setting it to around 25% always yields good result in different document sets, provided that there are at least several hundreds of documents.

### 5.3.3 Efficiency and Scalability

The largest data set, *Reuters*, is chosen to exam the efficiency and scalability of our algorithm on a Pentium III 667 MHz PC. Figure 5.2 compares the runtime of our algorithm only with bisecting k-means and HFTC. UPGMA is excluded again because it is not scalable. The MinSup of HFTC and our algorithm is set to 10% to ensure that the accuracy of all produced clustering solutions is approximately the same. The efficiency of HFTC is comparable with other algorithms in the first 5000 documents, but its runtime grows rapidly while there are 6000 or more documents. Our algorithm FIHC runs twice faster than the best competitor, bisecting k-means. We conclude that FIHC is significantly more efficient than other algorithms.

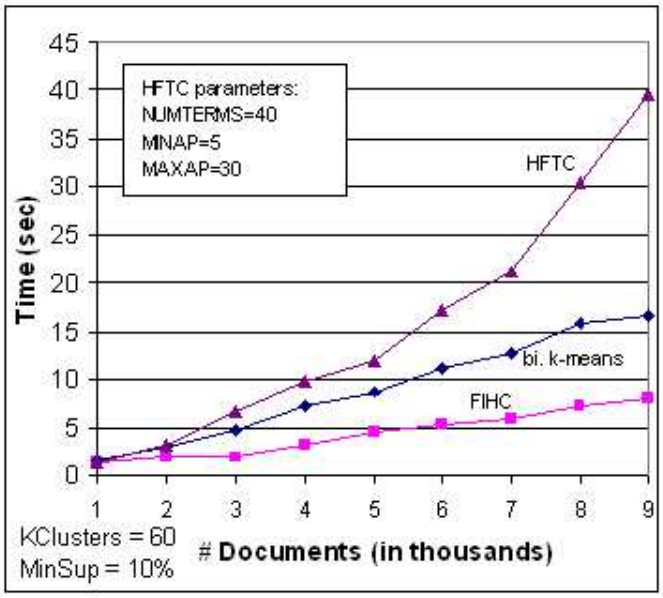


Figure 5.2: Comparison on efficiency

Many experiments were conducted to analyze the scalability of our algorithm. To create a larger data set for examining the scalability, we duplicated the files in *Reuters* until we get 100000 documents. Figure 5.3 once again illustrates that our algorithm runs approximately twice faster than bisecting k-means in this scaled up document set. Figure 5.4 depicts the runtimes with respect to the number of documents for different stages of our algorithm. The whole process was completed within two minutes while UPGMA and HFTC could not even produce a clustering solution. It demonstrates that FIHC is a very scalable method. The figure also shows that the Apriori and the clustering are the most time-consuming stages in FIHC, while the runtimes of tree building and pruning are comparatively short. The efficiency of the Apriori is very sensitive to the input parameter MinSup. Consequently, the runtime of FIHC is inversely related to MinSup. In other words, runtime increases as MinSup decreases. Nevertheless, many scalable and efficient frequent itemset generation algorithms have been proposed [24, 25]. For example, the FP-growth algorithm that we have discussed

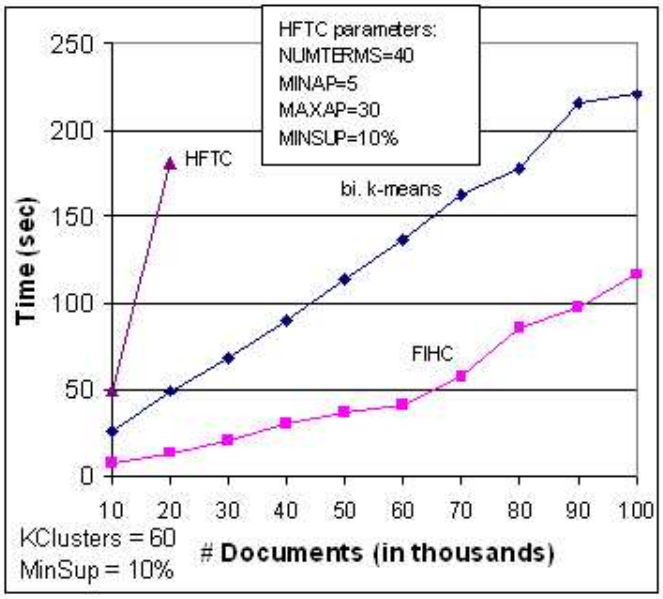


Figure 5.3: Comparison on efficiency with scale-up document set

in Chapter 2 may be employed to further improve the efficiency of our method. In the clustering stage, most of the time is spent on constructing initial clusters and its runtime is linear with respect to the number of documents.

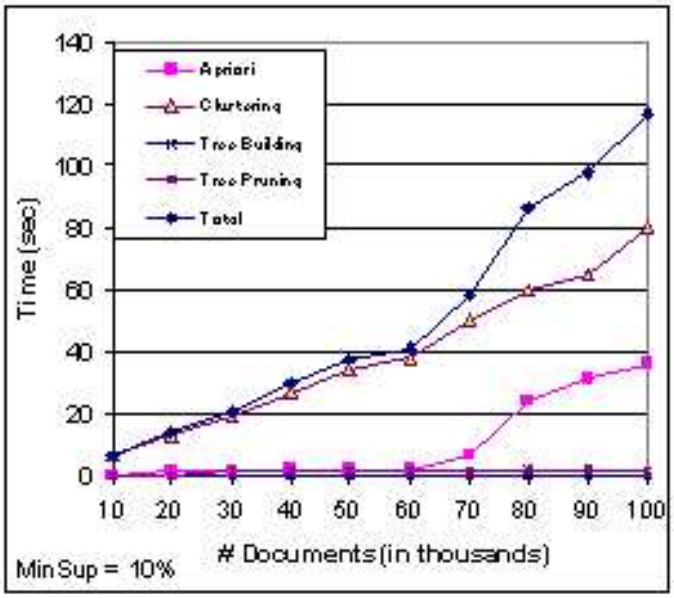


Figure 5.4: Scalability of FIHC

# Chapter 6

# Discussions and Conclusions

## 6.1 Browsing

To illustrate that our method provides meaningful cluster labels for browsing, figure 6.1 depicts part of the cluster tree. The parent topic discusses about "finance" and its cluster label is "dollar". It is further broken down into forty subtopics where most of them are either directly or indirectly related to the topic of finance. For example, the documents under the subtopics of "bank, growth" and "rate, yen" discuss about the future growth of banking sector and the interest rate of Japanese yen respectively.

Most of the existing agglomerative and divisive hierarchical clustering methods, e.g., bisecting k-means, generate relatively deep hierarchies. However, deep hierarchy may not be suitable for browsing. Suppose a user makes an incorrect selection while navigating the hierarchy. She may not notice her mistake until she browses into the deeper portion of the hierarchy. Due to fact that the depth of the tree is controlled by the number of clusters, this problem is unavoidable in many hierarchical methods.

Our hierarchy is relatively flat as shown in figure 6.1. Flat hierarchies reduce the number of navigation steps which in turn decreases the chance of making mistakes. Nevertheless, if the hierarchy is too flat, then a parent topic may contain too many

subtopics and it would increase the time and difficulty for the user to locate her target. Thus, a balance between the depth and the width of the tree is essential for browsing. Given a reasonable MinSup from 3% to 9%, our cluster tree usually has two to four levels in our experimental results. This number of levels is very close to those of human generated subject hierarchies, e.g., Yahoo!.

Another frequent itemset-based method, HFTC, also provides a relatively flat hierarchy and its lattice structure is suitable for browsing. Nevertheless, the resulting hierarchy usually contains many clusters at the first level. As a result, documents in the same natural class are likely to be distributed into different branches of the hierarchy which decreases the overall clustering accuracy. Our sibling merging method resolves this problem by joining similar clusters at the first level of the tree.

## 6.2 Complexity Analysis

Our method involves four phases: finding global frequent itemsets, initial clustering, tree construction, and pruning. The problem of finding frequent itemsets has been studied intensively in the data mining literature. In the initial clustering phase, the document feature vectors are scanned twice, once for constructing initial clusters and once for making clusters disjoint. Since an initial cluster labeled by a global frequent itemset $f$ contains $global\_support(f)$ documents, this step makes $\sum_{f \in F} global\_support(f)$ document-to-cluster assignments and score calculations. This amount of work is no more than the support counting in mining global frequent itemsets. In the tree construction, all empty clusters with a maximal cluster label are first removed. The remaining number of clusters is no more than, often much smaller than, the number of documents. The tree construction is essentially linear in the number of remaining clusters because finding a parent for a $k$-cluster only requires to examine $k$ of $k-1$-clusters where $k$ is usually small. Child pruning makes only one scan of clusters, and sibling merging is performed only at the first level of the tree. To summarize, the steps involved in initial clustering, tree construction and pruning are no more expensive than mining global frequent itemsets.

## 6.3 Contributions

Most traditional clustering methods do not satisfy the special requirements for document clustering, such as high dimensionality, high volume, and ease of browsing with meaningful cluster labels. This thesis has provided an innovative approach by using frequent itemsets as the basis for different stages. Our main contributions include:

- *Reduced dimensionality.* We use the low-dimensional feature vector, which is composed of global frequent items, in place of the original high-dimensional document vector. This replacement drastically reduces the dimension of the document vector space. Consequently, it greatly enhances the efficiency and scalability of our method.

- *Efficient and scalable.* Our method requires only two scans of the document set to cluster all the documents: one scan for constructing initial clusters and one scan for making clusters disjoint. Our experiments on different types of data sets suggest that our method is an extremely efficient and scalable. Its processing time is also predictable.

- *Accurate.* Our method consistently outperforms the well-known clustering algorithms in terms of accuracy on various types of document sets, even when the number of clusters is unknown. This result suggests that our score function formulates a sound clustering criterion and our pruning methods further improve its accuracy.

- *Robust to outliers.* Outliers in document clustering usually refers to the documents that are very different from the rest of the documents in the data set. Due to the fact that our algorithm only uses frequent items for clustering, outliers are basically ignored in the clustering process. Still, our score function is capable of assigning these documents to their most suitable clusters. In case none of the clusters is suitable for these outliers, they are assigned to the "null" cluster and remain intact for the rest of the algorithm. Thus, the presence of outliers does not degrade the overall accuracy of our clustering solution.

- *Easy for browsing.* The resulting clustering solution is a cluster (topic) tree where the nodes can be treated as topics and subtopics. User may easily navigate different topics in the document set through the tree. Each topic has a label which concisely summarizes the members in the cluster. Our method does not require additional processing to generate these cluster labels. Unlike other hierarchical methods where the parent cluster contains all the documents of its descendants, the parent cluster in our method contains only the general documents on the topic. Thus, our hierarchy is definitely more suitable for browsing.

- *Easy for data exchange.* Given that the resulting hierarchy is a tree structure, our output is an XML file which is the standard method of exchanging data in nowadays software development. Other text mining tools, e.g. document classification program, may easily utilize the tree for further processing.

- *Minimal requirements for domain knowledge.* Our method treats the number of desired clusters as an optional input parameter. Although we require another input parameter MinSup, we provide a clear guideline for on to choose a suitable value for this parameter depending on the size of the document set. Close to optimal accuracy can usually be obtained by following such guideline.

- *Arbitrary cluster shape.* Our clustering method can represent clusters in any shape because our clustering criterion is based on our innovative score function, instead of the traditional Euclidean or Manhattan distance measures. In other words, the shape of the natural clusters do not affect the accuracy of our method.

- *Independent on the order of input data.* The order of the data does not affect the clustering result at all. Our method always produces the same result given the same document set.

## 6.4 Future Work

Future study on document clustering using frequent itemsets has the following possible avenues:

- In the current implementation of FIHC, all feature vectors are stored in the main memory. Although the dimension of feature vectors is comparatively low, scalability may become a problem in cases where the data set is extremely large. A possible direction for future research is the development of a disk resident version of FIHC, and its application to very large data sets, for example, the Yahoo! subject hierarchy with millions of documents.

- We may want to incrementally update [12] the cluster tree when some new documents arrive, for example, a new research paper is submitted to the database or a new web page is found by a web crawler. In the current implementation, this task can be accomplished by assigning the new document to the most similar cluster, but the clustering accuracy may degrade over time because the global frequent itemsets may not necessary reflect the current state of the document set. Thus, an incremental updating version of FIHC is necessary for this situation. Incremental clustering is related to some of the recent research in data mining on stream data [20, 15, 26].

- Most of the current document clustering algorithms, including FIHC, consider a document as a bag of words. While the semantic relationships among the words may be crucial for clustering, they are not utilized. FIHC may incorporate the Universal Networking Language [44], a recently proposed semantic representation for sentences, for feature vector generation and score computation.

- Another possible research direction is to apply FIHC in a cross-language environment using the EuroWordNet multilingual database (Gonzalo et al., In press) [46] with wordnets for several European languages such as Dutch, Italian, Spanish, German, French, Czech and Estonian. The wordnets are structured in the same way as the American wordnet for English [36] in terms of synsets (sets

of synonymous words) with basic semantic relations between them.[1] Suppose we need to cluster documents that are written in different European languages. The basic approach is to first map the words in the document set into the American wordnet via the EuroWordNet InterLingual Index. A vector model is then constructed from this mapped monolingual document set.

In conclusion, the importance of document clustering will continue to grow along with the massive volumes of unstructured data generated. We believe exploiting an effective and efficient method in document clustering would be an essential direction for research in text mining.

[1] http://www.illc.uva.nl/EuroWordNet/


Figure 6.1: Cluster labels

# Bibliography


[1] C. Aggarwal, S. Gates, and P. Yu. On the merits of building categorization systems by supervised clustering. In *Proceedings of (KDD) 99, 5th (ACM) International Conference on Knowledge Discovery and Data Mining*, pages 352–356, San Diego, US, 1999. ACM Press, New York, US.

[2] R. Agrawal, C. Aggarwal, and V. V. V. Prasad. Depth-first generation of large itemsets for association rules. Technical Report RC21538, IBM Technical Report, October 1999.

[3] R. Agrawal, C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent item sets. *Journal of Parallel and Distributed Computing*, 61(3):350–371, 2001.

[4] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD98)*, pages 94–105, 1998.

[5] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD93)*, pages 207–216, Washington, D.C., May 1993.

[6] R. Agrawal and R. Srikant. Fast algorithm for mining association rules. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12-15 1994.

[7] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering*, pages 3–14, Taipei, Taiwan, March 1995.

[8] M. Ankerst, M. Breunig, H. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99)*, pages 49–60, Philadelphia, PA, June 1999.

[9] F. Beil, M. Ester, and X. Xu. Frequent term-based text clustering. In *Proc. 8th Int. Conf. on Knowledge Discovery and Data Mining (KDD)'2002*, Edmonton, Alberta, Canada, 2002. http://www.cs.sfu.ca/˜ ester/publications.html.

[10] H. Borko and M. Bernick. Automatic document classication. *Journal of the ACM*, 10:151–162, 1963.

[11] S. Chakrabarti. Data mining for hypertext: A tutorial survey. *SIGKDD Explorations: Newsletter of the Special Interest Group (SIG) on Knowledge Discovery & Data Mining, ACM*, 1:1–11, 2000.

[12] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the 29th Symposium on Theory Of Computing STOC 1997*, pages 626–635, 1997.

[13] Classic. ftp://ftp.cs.cornell.edu/pub/smart/.

[14] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 318–329, 1992.

[15] P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.

[16] R. C. Dubes and A. K. Jain. *Algorithms for Clustering Data*. Prentice Hall College Div, Englewood Cliffs, NJ, March 1998.

[17] A. El-Hamdouchi and P. Willet. Comparison of hierarchic agglomerative clustering methods for document retrieval. *The Computer Journal*, 32(3), 1989.

[18] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd int. Conf. on Knowledge Discovery and Data Mining (KDD 96)*, pages 226–231, Portland, Oregon, August 1996. AAAI Press.

[19] A. Griffiths, L. A. Robinson, and P. Willett. Hierarchical agglomerative clustering methods for automatic document classification. *Journal of Documentation*, 40(3):175–205, September 1984.

[20] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *IEEE Symposium on Foundations of Computer Science*, pages 359–366, 2000.

[21] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. In *Proceedings of the 15th International Conference on Data Engineering*, 1999.

[22] E. H. Han, B. Boley, M. Gini, R. Gross, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. Webace: a web agent for document categorization and exploration. In *Proceedings of the second international conference on Autonomous agents*, pages 408–415. ACM Press, 1998.

[23] J. Han and M. Kimber. *Data Mining: Concepts and Techniques*. Morgan-Kaufmann, August 2000.

[24] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD'00)*, Dallas, Texas, USA, May 2000.

[25] J. Hipp, U. Guntzer, and G. Nakhaeizadeh. Algorithms for association rule mining - a general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, July 2000.

[26] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, San Francisco, CA, 2001. ACM Press.

[27] G. Karypis. Cluto 2.0 clustering toolkit, April 2002. http://www-users.cs.umn.edu/˜ karypis/cluto/.

[28] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley and Sons, March 1990.

[29] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In D. Fisher, editor, *Proceedings of (ICML) 97, 14th International Conference on Machine Learning*, pages 170–178, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.

[30] Kosala and Blockeel. Web mining research: A survey. *SIGKDD Explorations: Newsletter of the Special Interest Group SIG on Knowledge Discovery & Data Mining*, 2, 2000.

[31] G. Kowalski and M. Maybury. *Information Storage and Retrieval Systems: Theory and Implementation*. Kluwer Academic Publishers, 2 edition, July 2000.

[32] J. Lam. Multi-dimensional constrained gradient mining. Master's thesis, Simon Fraser University, August 2001.

[33] B. Larsen and C. Aone. Fast and effective text mining using linear-time document clustering. *KDD'99*, 1999.

[34] D. D. Lewis. Reuters. http://www.research.att.com/˜ lewis/.

[35] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Knowledge Discovery and Data Mining (KDD) 98*, pages 80–86, 1998.

[36] Miller. Princeton wordnet, 1990.

[37] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980.

[38] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[39] K. Ross and D. Srivastava. Fast computation of sparse datacubes. In M. Jarke, M. Carey, K. Dittrich, F. Lochovsky, P. Loucopoulos, and M. Jeusfeld, editors, *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB97)*, pages 116–125, Athens, Greece, August 1997. Morgan Kaufmann.

[40] H. Schutze and H. Silverstein. Projections for efficient document clustering. In *Proceedings of SIGIR'97*, pages 74–81, Philadelphia, PA, July 1997.

[41] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[42] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. *KDD Workshop on Text Mining'00*, 2000.

[43] Text REtrival Conference TIPSTER, 1999. http://trec.nist.gov/.

[44] H. Uchida, M. Zhu, and T. Della Senta. Unl: A gift for a millennium. The United Nations University, 2000.

[45] C. J. van Rijsbergen. *Information Retrieval*. Dept. of Computer Science, University of Glasgow, Butterworth, London, 2 edition, 1979.

[46] P. Vossen. Eurowordnet, Summer 1999.

[47] K. Wang, C. Xu, and B. Liu. Clustering transactions using large items. In *CIKM'99*, pages 483–490, 1999.

[48] K. Wang, S. Zhou, and Y He. Hierarchical classification of real life documents. In *Proceedings of the 1st (SIAM) International Conference on Data Mining*, Chicago, US, 2001.

[49] W. Wang, J. Yang, and R. R. Muntz. Sting: A statistical information grid approach to spatial data mining. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 186–195, Athens, Greece, August 25-29 1997. Morgan Kaufmann.

[50] Yahoo! http://www.yahoo.com/.

[51] O. Zamir, O. Etzioni, O. Madani, and R. M. Karp. Fast and intuitive clustering of web documents. In *KDD'97*, pages 287–290, 1997.