Significant Feature Clustering

by

John Samuel Whissell

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2006

Abstract

In this thesis, we present a new clustering algorithm we call *Significance Feature Clustering*, which is designed to cluster text documents. Its central premise is the mapping of raw frequency count vectors to discrete-valued significance vectors which contain values of -1, 0, or 1. These values represent whether a word is *significantly negative*, *neutral*, or *significantly positive*, respectively. Initially, standard tf-idf vectors are computed from raw frequency vectors, then these tf-idf vectors are transformed to significance vectors using a parameter α, where α controls the mapping -1, 0, or 1 for each vector entry. SFC clusters agglomeratively, with each document's significance vector representing a cluster of size one containing just the document, and iteratively merges the two clusters that exhibit the most similar average using cosine similarity. We show that by using a good α value, the significance vectors produced by SFC provide an accurate indication of which words are significant to which documents, as well as the type of significance, and therefore correspondingly yield a good clustering in terms of a well-known definition of clustering quality. We further demonstrate that a user need not manually select an α as we develop a new definition of clustering quality that is highly correlated with text clustering quality. Our metric extends the family of metrics known as *internal similarity*, so that it can be applied to a tree of clusters rather than a set, but it also factors in an aspect of recall that was absent from previous internal similarity metrics. Using this new definition of internal similarity, which we call *maximum tree internal similarity*, we show that a close to optimal text clustering may be picked from any number of clusterings created by different α's. The automatically selected clusterings have qualities that are close to that of a well-known and powerful hierarchical clustering algorithm.

Acknowledgments

I would like to thank my supervisors, Chrysanne DiMarco and Charlie Clarke, for their help in finishing this thesis. Chrysanne's guidance in style and form of writing was most helpful, but even more important, she helped me with all the other academic matters that would have otherwise bogged me down. Charlie's technical help and understanding was and is greatly appreciated. I would also like to thank Frank Tompa and Chris Eliasmith for agreeing to be my readers, even though they where given such short notice. My friends, in the Artificial Intelligence lab and elsewhere, also deserve note here, as they provided not only useful intellectual discussions but friendship to make the work seem much lighter. I want to thank my family, my mother has been amazingly supportive, offering insights on academic life and research comments, but mostly doing what mothers do best, making sure their children are well. All my family has helped with this even if they don't know they did.

Dedication

This thesis is dedicated to my daughter Melissa.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In recent years there has been a massive proliferation of data that is accessible electronically, ranging in form from highly structured databases, such as article collections and company records, to web sites and pages that vary greatly in content. Despite the large differences in structure between the various forms of data, the principal type of information available in all these formats is essentially the same: text. With this in mind, we can refer to any logically encapsulated unit of text, regardless of its source, as a *document*. A huge amount of research has been devoted to creating effective ways to access various document collections based on user needs. For example, research into effectively accessing web pages has led to the proliferation of *search engines* such as Google [5] and Yahoo [11], where a small piece of text, a *query*, is submitted to the engine. The query represents the user's information needs. The search engine matches the query to its indexed web pages. More organized document collections, such as article databases (e.g., ACM Portal [1] and CiteSeer [2]), frequently offer search methods based on information that is given either by humans or found in structures within the text (e.g., bibliographies in TeX, XML tags in web pages, etc.). These search tools represent one of many approaches to analyzing large volumes of text documents. Fields of research such as *Data Mining*, *Information Retrieval*, and *Natural Language Processing* have all dealt extensively with the problem. One particular approach that has been employed by all three is the topic of this thesis: *Clustering*.

Clustering algorithms are generally unsupervised learning techniques, not requiring data labels, that take some data set of objects, such as documents, gene sequences, proteins, consumers surveys, or geological information, and group that set into groups of similar objects. Clustering algorithms are similar to *classification algorithms*, but while clustering algorithms create groups without using labels (generally), classification algorithms are supervised learning techniques that assign objects to known labels, implying that all the label types of interest are known in advance.

The usefulness of creating groups of similar objects through either clustering or classification may be seen in various applications. For example, if proteins can be grouped by structural similarity, it may be possible to extrapolate the function of unknown proteins by the properties of other proteins in the same group. Companies might try to group consumers by preferences in order to better direct advertising and sales. Medical professionals might wish to identify patients at high risk for certain diseases by finding other patients with similar physical/internal parameters and checking the number who have the ailment. In text applications, clustering can facilitate a task we refer to as *thematic querying*, that is a request for documents on a certain topic. If a clustering algorithm could generate groups of documents with similar topics, a thematic query could be answered by matching the query to a whole group. This is significantly easier and faster, although usually not more accurate, than determining how well each document matches a thematic query. Another use of text clustering is to analyze the relationships between documents, which is a specific case of a more-general clustering application, *exploratory data analysis*. Exploratory data analysis refers to an attempt to analyze ('data mine') a data set for useful information when there is uncertainty about the information that is actually present.

A myriad of algorithms have been developed for text clustering, so many that it is not possible to cover them all in any reasonable depth. Hierarchical clustering algorithms are a popular subtype of clustering algorithms well-suited to text. Such approaches generate a *dendrogram*, a tree of clusters within clusters, where leaves are individual documents, and nodes further up in the tree represent clusters consisting of all their descendant leaves. Dendrograms are good for browsing, and are arguably the most natural way to represent document collections.

Previous hierarchical clustering algorithms have varied greatly in approach. Some progressively split up a data set into smaller and smaller clusters, while other merge clusters together into larger clusters until a certain number of clusters remain. Various methods exist for selecting the clusters to be merged or split. Each algorithm has a unique approach, but the large majority of them (with a few notable exceptions) are based on a single concept, *the vector space model*, in which documents are represented as vectors. The way in which algorithms create these vectors is usually identical, this being a form of weighting function applied to word counts.

In this thesis, we present a new clustering algorithm called SFC (Significant

Feature Clustering), which uses a standard merge-based hierarchical template: find the most similar clusters at each iteration using a metric X and representation for each cluster Y, and merge them into one cluster. The novelty of our approach is in the method SFC uses to generate both document and cluster representations. SFC uses a basic tf-idf approach to generate initial weighted vectors, then overlays the basic tf-idf function with a secondary weighting function that maps each weight from a real number to one of three possible significance types, positive, negative, or neutral, where significance is defined as the significance of a word to the topic of a document. This mapping is done such that the number of documents with positive and negative significance values for any one word is restricted by a parameter α. Using significances, SFC is able to represent clusters in a useful form we refer to as a *probabilistic significance vector*. These vectors represent the type of significance each word has to a cluster, along with the probability of that significance. During each iteration of SFC, clusters that have a maximal similarity in their significance vectors are merged together. We will show that by carefully selecting the parameter α, the performance of SFC can be improved as well as improving the quality of the clustering solution produced. SFC is competitive with UPGMA [89], a well-known and high-quality clustering algorithm.

An overview of the material in this thesis is as follows. In Chapter 2, we present an overview of clustering.The first part of the chapter covers the preliminary aspects of clustering, including: a definition of clustering, how objects can be represented in clustering algorithms, how better representations can be created through feature extraction/selection, how similarity can be quantified, how clustering algorithms can be evaluated, and various other issues clustering algorithms must deal with. Following this, a literature review of clustering algorithms is provided. The review is organized by major types of clustering algorithms, and within each major type there may be several smaller subtypes. For each, we give a representative sample of the algorithms that exist. Note that our organization of the clustering algorithms is by no means the only way to organize them—every review has a slightly different organization scheme.

Chapter 3 presents our new clustering algorithm called SFC, Significance Feature Clustering. We first reiterate the problem domain in which our algorithm is intended to function, then we build up the basis for our algorithm, starting with how we generate data representations, followed by our notion of significance and similarity, and how it can be used to cluster. We then present the full algorithm, and discuss how it deals with important clustering issues.

An evaluation of SFC is presented in Chapter 4. The principal purpose of this chapter is to show that SFC is competitive with other text-clustering algorithms in terms of one of the most commonly-used definition of quality. In addition, we examine the effects of varying SFC's parameters on the trade-off between quality of clustering and improvement in speed, as well as on other properties of the algorithm. Lastly, we discuss directions for future work.

Chapter 5 provides conclusions on our research, a summary of our algorithm, assessment of its uniqueness and quality, and a reiteration of future work to pursue.

Chapter 2

Background on Clustering

In this chapter we will review the background of clustering. We begin with a preliminary section that explains the goals of clustering, the terminology associated with it, and its general concepts. Whenever possible, we emphasize how these aspects are related to text. The preliminary section is followed by a discussion of specific clustering algorithms. Each main type of clustering approach will be described in its own section. Within each section, we give a brief definition of the general model algorithms of that type follow, then a representative sample of algorithms of that type are presented. Note that the algorithms we will present are applicable to various data types. We will therefore specify the data type for which each algorithm is intended in its description. In particular, we will highlight text-oriented approaches.

2.1 Preliminaries

2.1.1 Definition of Clustering

A *clustering* of a data set is a splitting of the data set into a collection of subsets. These subsets are called *clusters*. Figure 2.1(a) shows a set of data points, and Figure 2.1(b) illustrates a possible clustering using spatial distance. Subsets may be nested recursively, as in Figure 2.1(c), creating a hierarchy. Although Figure 2.1(b) shows that a cluster may be a simple partitioning, it should be clear from the other examples that it need not be. Clusters may overlap or be nested within each other. Each object in the data set may belong to only one cluster, probabilistically to any number of clusters, or wholly belong to multiple clusters (*disjunctive* *clustering*). When dealing with probabilities of membership, we use the notation $P(x|y)$ to denote the probability of x given that y has occurred. The type of subsets produced vary based on the algorithm used to cluster. Objects in the same *cluster* are 'similar'. In Figure 2.1, lower spatial distance is taken as an indicator of similarity.

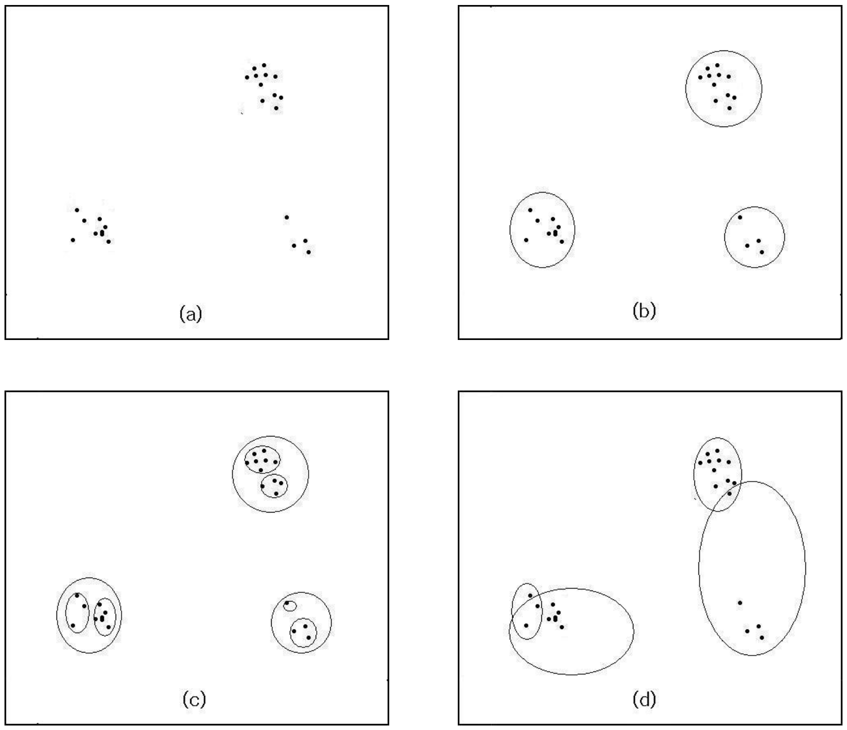

Figure 2.1: What is a cluster?
(a) shows a set of data points in two dimension. For example, a height versus weight plot, or a time versus speed plot. (b) illustrates a clustering of the points in (a). (c) illustrates a clustering of the points in (a) with subsets within subsets. (d) illustrates a clustering with overlapping clusters. Any of these are valid examples of clustering.

A *clustering algorithm* is an algorithm that takes a data set (such as shown in Figure 2.1(a), and produces some clustering of the data set (such as shown in Figure 2.1(b)-(d)). The basic goal of any clustering algorithm is to generate clusters that contain similar objects. Consequently, distinct clusters should contain dissimilar objects. The definition of similarity is highly varied. The type of data objects being clustered restricts the definition of similarity. For example, in Figure 2.1, we have numeric point data objects, so we can use distance to indicate similarity. However, as we will see in following sections, this is only one of many ways to define similarity.

Clustering is associated with many fields of study. It has been used in statistics for a long time (Arabie et al. [14]). It has been applied in various fields including bioinformatics (Kasturi & Acharya [63], Guo [49], Au et al. [15]), information retrieval (Crouch et al. [31], Leuski [68], Ng et al. [76]), marketing (Wu & Lin [98]), and many others too numerous to mention. Clustering is a powerful tool for data mining, applicable to virtually every field where there are large amounts of information requiring organization. Unfortunately, it is not possible to cover even a significant fraction of the clustering algorithms that exist. Rather, in this chapter we will focus on explaining the main techniques for clustering and some of the better-known algorithms that implement these techniques. The next few sections will explain the basics of clustering before we move on to the specific types of algorithms. Where applicable, we will focus on clusterings from a text perspective. For those interested, some other recent reviews of clustering include Jain et al. [59], Berkhin [21], and Zhao & Karypis [102] (for hierarchical text-specific algorithms).

2.1.2 Data Set

Before we design a clustering algorithm, we need to know the type of data set on which we are operating. The data set might be a collection of patient information for a medical study, genome sequences, sales figures, or something as simple as height and weight information. For the remainder of this thesis, we will refer to any data set as D, with D_i being the ith object in the data set, and n being the number of objects in the data set. This notation is given formally below:

$$D = \langle D_1, D_2 \cdots D_n \rangle$$

We will refer to the output clustering produced by running a clustering algorithm on D as C, where C has k clusters. Each cluster C_i is a subset of C:

$$C = \langle C_1, C_2 \cdots C_k \rangle$$

When dealing with text, the data-set objects may be as small as single letters. *Stemming* algorithms, algorithms designed to strip word endings and/or lemmatize them to their roots (*foods* to *food*, *flight* to *fly*, etc.) sometimes rely on single-letter data objects (or multiple letters, but still not a word). An example of a stemming clustering algorithm based on letters is given in Xu & Croft's [99]. At the next level up from letters, we may look at syllable objects, then words. *Word Sense Disambiguation*, which involves assigning one sense of many possible senses to a word, treats text as words. Chen & Chang [28] gives an example of a disambiguation tool based on word clustering. Moving up from words, there are phrases, sentences, and blocks of texts at subsequent levels. Finally, the most common view of text is the document. The vast majority of text-clustering algorithms are designed to handle document clustering. It is important to note that techniques applicable to one level of text clustering may not translate to another. Also, it is important to note that text clustering is potentially very language-sensitive.

2.1.3 Data Representation

The *data representation* of a clustering algorithm refers to the form used to represent objects of the data-set. Consider a medical study in which we wish to classify patients as high risk, medium risk, or low risk for cancer. Our data-set is the collection of patients' data. Our data representation for each patient might then be weight, age, and gender. By selecting these features, we are saying that we believe (or have factual evidence) that they provide a good indication of cancer risk. But why not use eye colour, diet, or height as well? In general, objects have a massive number of features, and only a small fraction of them are selected to be used in the data representation.

We will refer to each property of an object in a data-set as a *feature*. Each feature of a data representation takes the form of one of three types: *numeric*, *Boolean*, or *categorical*. A numeric feature refers to an ordinal number. Height, weight, and age are examples of numeric attributes. For text blocks, a basic numeric feature is a word frequency count, although modern algorithms dealing with text manipulate this basic count considerably. Boolean features are those that are present or absent (e.g., having or not having a disease for example). Older *Information Retrieval* (*IR*) Systems often used Boolean word lists, recording only whether a word was present or absent in a document, and not a specific count. This data representation is generally now outmoded for text. Categorical features are those that draw from a group of categories. Eye colour may be green, brown, black, or blue. Gender may be male or female, etc.

Individual features of each D_i are almost always combined into a single representation. The most common way of doing this is to use the *vector space model*, discussed in Salton's 1975 work on automated indexing in Information Retrieval [82], which is illustrated in Figure 2.2. In this model, each D_i is represented as an

m-dimensional vector, where m is the number of features. D_{ij} represents the jth feature's value in D_i. The vector representation of D_i is then:

$$D_i = \langle D_{i1}, D_{i2} \cdots D_{im} \rangle$$

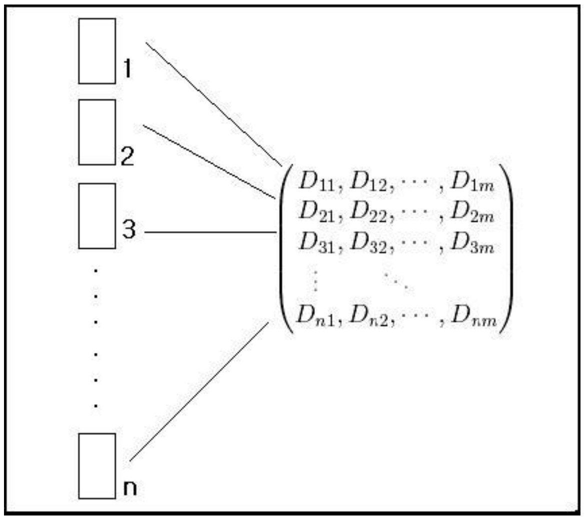
1
2
3
n
$D_{11}, D_{12}, \cdots, D_{1m}$
$D_{21}, D_{22}, \cdots, D_{2m}$
$D_{31}, D_{32}, \cdots, D_{3m}$
$D_{n1}, D_{n2}, \cdots, D_{nm}$

Figure 2.2: The vector space model
The vector space model takes a set of n documents with a total of m different terms and maps each object to a vector of length m, where D_{ij} is a numeric quantity representing the importance of term j to object i.

The effectiveness of the vector space model is due to the key attribute that the space must have: similarity of data objects should be inversely proportional to distance. We can use this aspect of the vector space to create clusters of similar objects. We group vectors with low spatial distance (high similarity) together. The problem is to find a feature set that has the desired property of similarity being the inverse of distance.

Certain objects are not amenable to the vector space model in their raw form. For example, how do we represent text in the vector space model. This has led to the development of data representations. For text, the data representation is based on the text unit being examined. In situations where the text object itself is small (letter, word, phoneme) the data representation is typically a form of modified count information of surrounding neighbors. For example, in word sense disambiguation, a word may represented by its ‘context window’, which is just the surrounding words. In Gale et al. [44], the authors give an example of this form of representation. A context window may be viewed as a vector of word counts. Larger text blocks provide all the information needed for clustering within themselves. Thus, a document's data representation is based on counting words within the document (or other structures). If tf_{ij} refers to the number of occurrences of *term* j in D_i (note we use the word *term* and not word, because we may be dealing with phrases, a set of letters, or any other structure within text), then D_i's raw count data representation is:

$$D_i = \langle tf_{i1}, tf_{i2} \cdots tf_{im} \rangle$$

Data representations based on raw term counts (when terms are words only) are not overly useful for clustering, information retrieval, or most text-related applications. They do not accurately capture significance of terms to documents and therefore lead to poor results. Because of this, a great deal of research has gone into *weighting functions*. In IR and typically text-clustering as well, a weighting function has as its parameters a document D_i, a term k, and the document set D. Such a function returns the significance value of term k to document D_i based on term distribution information of k in D and D_i. We refer to the weight for term k in document i as w_{ik}. Thus the weighted data representation of D_i is:

$$D_i = \langle w_{i1}, w_{i2} \cdots w_{im} \rangle$$

Good weighting functions have been shown to produce enormous improvements in IR and clustering over simple frequency counts. A weighted term data representation is the accepted standard when not dealing with categorical features. A particularly useful family of weighting functions is known as tf-idf. We will discuss tf-idf in more depth in Chapter 3, but to give an introduction to its concepts, tf-idf factors document frequency, which is the number of different documents containing a term, and term frequency, which is the count for a word within a specific document. For example, a simple tf-idf weighting is $w_{ij} = tf_{ij}\frac{|D|}{n_j}$, where n_j is the document frequency of j. Document frequency makes tf-idf metrics very useful, as it has been verified that terms with lower document frequencies better distinguish documents.

Boolean counts can be represented in the vector space model (0's and 1's) as well, but categorical data cannot be represented in the vector space model. The vector space model takes a 'bag' approach to modelling objects. That is to say, order of occurrence within a data object does not affect its vector representation.

In some cases, where no ordering is present, this makes sense. But text is ordered: 'Don't touch that, it will explode if you do' and 'Do touch that, if you don't it will explode' have the same word vector representation, but clearly have opposite meanings. In addition to ignoring word order, the vector space model does not automatically handle homographs (words with the same spelling but different meanings) or synonyms. Despite these shortcomings, the vector space model has had marked success with text and many other kinds of data-sets. Most clustering algorithms use it.

2.1.4 Feature Selection and Dimensionality Reduction

A key component of learning algorithms is the identification of features. For example, consider a set of documents with a vector space of 1,000,000 words. Performing clustering, or any learning at all, on a data-set with this high of a dimensionality can be very difficult, assuming it is possible at all. If we can reduce the vector sizes, say several hundred dimensions, learning will be faster. Not only this, but a well-defined mapping to the smaller vectors can increase the accuracy of learning algorithms. However, only a small fraction of this information may be useful in any given study. In general, we want to include features that improve the accuracy of learning algorithms, and omit others. There are two approaches for identifying important features: *feature extraction (dimensionality reduction)*, and *feature selection*. We discuss these two methods below. It should be noted that not all methods of clustering require feature selection or dimensionality reduction, for example, some, but not all, Support Vector Machine methods, mentioned very briefly in Section 2.2.6, don't use these concepts.

Feature Selection

Feature Selection, as the name implies, involves selecting which subset of features to use. It is often applied in unsupervised situations such as clustering, but supervised learning algorithms also make use of it (Cardie [26], Kohavi & John [66]).

Learning algorithms associated with text benefit greatly from feature selection. In most large collections of text, the vocabulary (number of distinct words) numbers well into the hundreds of thousands. If the count of each distinct word in the vocabulary is a feature of each text block in the collection, then every text block has a feature set the same size as the vocabulary. Only a fraction of these words may be useful in a given clustering algorithm. The useless words impose both a

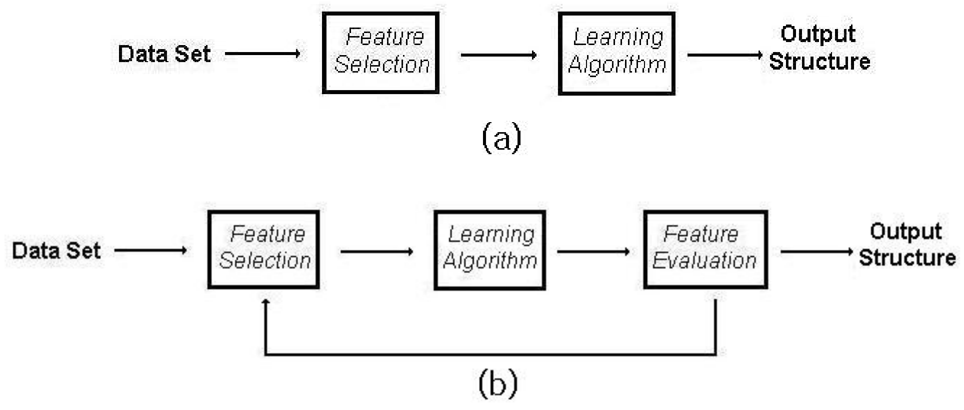

Figure 2.3: Models of feature selection
The two models of feature selection. (a) Filter model: feature selection is a preprocessing step. (b) Wrapper model: features are selected iteratively by running the learning algorithm and observing the results.

large run-time penalty on learning, and lower accuracy due to a huge amount of noise words.

There are two ways of performing feature selection, the *filter model* (John et al. [60]) and the *wrapper model* (Dy & Brodley [37]). The filter model is illustrated in Figure 2.3(a). In filter model feature selection, the set of all features possible is submitted to the filter algorithm, which then filters out the less informative features, leaving a good feature set for the learning algorithm to work on. The learning algorithm and the feature selection algorithm are separate. In the wrapper model (Figure 2.3(b)) we begin with some set of features (possibly one, or any number of them). The learning algorithm is then run to select which features to add (or remove). With this new feature set, the algorithm is rerun. With each iteration the set of features is refined so as to produce increased accuracy in the learning algorithm.

Filter algorithms rely on *labelled data*. For example, in a text classification algorithm, we would 'label' each text block with a topic. We could then use a filter algorithm that removes all words from the feature set which occur in greater than a certain fraction of topics. Some of the more common techniques used with the filter model include χ^2 significance testing, mutual information (Weiner et al. [97]), and information gain (Lewis & Ringuette [70]). In text categorization, document frequency (*DF*) is also used. Manning & Schütze [74] give an example of feature selection using χ^2 significance testing on a modified Reuters news wire collection (from Apté et al. [13]). They use the test to determine the 20 words (features) that best define the Reuters' earnings category. Methods such as χ^2 have been shown to produce increased accuracy and faster run-time in algorithms when they are used as feature filters. In addition to the types previously mentioned, there are also entropy-based filter methods such as Dash & Liu's [32].

In the wrapper model of feature selection, as the name implies, feature selection is 'wrapped around' the learning algorithm. The basic idea is to repeatedly refine the feature set by evaluating the output structure. Fukunaga [42] illustrates a basic wrapper-model feature-selection process, referred to as *sequential search*. Suppose we have the function $\phi(X)$, which evaluates the quality of the feature set X, and we desire ω features in our output feature set. Algorithm 1 shows the sequential search algorithm for this situation. For each iteration, the best feature (α) is selected to add to the output set. If α would decrease quality, the algorithm stops. There are many other wrapper model feature selection techniques, such as the evolutionary search (Kim [65]), that are faster and more accurate than sequential search.

Algorithm 1 Sequential Feature Search

```
1: Input: $f$: set of features
2:        $\omega$: desired number of features
3: $S = \{\}$
4: while $|S| < \omega$ do
5:     $\alpha = \arg\max_{f_i \in f} \phi(f_i \cup S)$
6:     if $\phi(\alpha \cup S) < \phi(S)$ then
7:         break
8:     else
9:         $S = S \cup \alpha$
10:        $f = f - \alpha$
11:    end if
12: end while
```

Both the filter model and the wrapper model can be applied to clustering problems, depending on the information we have about the data-set. *Exploratory data analysis (EDA)* is one of the primary functions of clustering. In EDA, we are unsure of true labels, so filter-model feature selection is not an option. When dealing with text though, we often have labels (such as document classes for the Reuters'

$$\begin{pmatrix} 0 & 4 & -8 & 13 & -2 & 1 & 6 \end{pmatrix}$$
$$\begin{pmatrix} 0 & 4 & 3 & -3 & -2 & 16 & 6 \end{pmatrix}$$
(a)

$$\begin{pmatrix} 0 & 4 & -2 & 6 \end{pmatrix}$$
$$\begin{pmatrix} 0 & 4 & -2 & 6 \end{pmatrix}$$
(b)

Figure 2.4: Subspace similarity
The vectors in (a) require some examination to note similarity. However, if we take the subspace dimensions 1, 2, 5, and 7 as in (b), the similarity becomes obvious. This is what subspace clustering attempts to do.

collection used by Apté [13], or word parts-of-speech tags in any number of online corpora). Consequently, either model of feature selection may be applicable to text.

Subspace clustering is a form of feature selection used in clustering. It follows the wrapper model in being performed during the clustering algorithm. The basic concept here is that, given a data-set of m features, similarities between objects are likely over a set of k features, $k < m$. Subspace clustering involves searching for those subspaces where object similarities are easier to detect algorithmically. Figure 2.4 gives an example of this.

Unfortunately, detection of subspaces in actual data is a much more complex problem than our example, and a great deal of research has been devoted to it. A review of various clustering algorithms employing subspace detection/manipulation can be found in Parsons et al. [78].

Feature Extraction

Feature Extraction may also be called *feature transformation* or *dimensionality reduction*. It serves much the same purpose as feature selection, but operates in different ways. Feature selection involves selecting only. Features may or may not be used in the learning algorithm. A feature extraction algorithm maps a set of objects from a high-dimensional space with M dimensions into lower dimensional space of m dimensions, where $m < M$. Feature extraction has an additional problem in that it is difficult to be sure what each feature represents in the new space.

Principal Component Analysis, or *PCA*, is a common tool for feature extraction in any number of applications, including clustering. A good tutorial on the concepts involved can be found online [8]. In PCA, for text clustering, the data-set is represented as a document-by-word feature matrix (with a mean adjusted to zero). A covariance matrix is then computed for the matrix. Eigenvectors are computed for the covariance matrix, which are ordered by length. Each eigenvector is a *component*, with length indicating importance. Now, to perform feature extraction, we need only pick the p largest eigenvectors (in matrix form), transpose them, then multiply them by the original data. The resulting matrix is a transformed feature set with only p dimensions. PCA is *lossy*, as omitting any eigenvector results in information loss. However, we can minimize the impact by removing only those eigenvectors with low length. Explaining why this method works is beyond the scope of this thesis, but it is very useful. Two good examples of PCA applied to clustering are given by Ding et al. [36] [35].

Another method of feature extraction is *Latent Semantic Analysis*, or *LSA*, which uses *Singular Value Decomposition*, or *SVD*. SVD is very similar to the eigenvector analysis used in PCA, but here we have *singular values* and *singular vectors* rather then eigenvalues and eigenvectors. A singular value y is a real-number for a document-by-word matrix X that fulfills the condition $Xv = yu$ and $X^T u = yv$, where v is a *left singular vector* and u is a right *right singular vector*. Identifying y's, u's, and v's for a given X follows in a manner similar to finding eigenvectors. Once this has been done, again, like in PCA, we simply pick the number of singular vectors we want to use and recombine these with the original data to create a new matrix that is reduced in dimensionality.

Hoffman [57] illustrates how LSA can be applied to clustering. LSA has been shown to be a useful tool in many applications. With regard to text, LSA has the property of mapping words that occur in similar contexts to the same dimensions. For example *mist* and *fog* might be collapsed into a single feature as they occur around similar words. There are many other methods for feature extraction, and we have given just a small sample here.

2.1.5 Similarity

We have stated the goal of clustering is to produce subsets of similar objects from an initially ungrouped data-set. As a bare minimum, clustering algorithms provide some definition of similarity between two individual objects of a data-set based on their respective feature sets. We denote the similarity between data objects D_i and D_j as $sim_x(D_i, D_j)$, with increasing values indicating higher similarity. When giving a specific type of similarity, x will be substituted with a short form of that name. For example, Euclidian similarity would be $sim_{euc}(D_i, D_j)$. The term *distance* used here is analogous to *similarity* in clustering, except low distance means high similarity, and vice versa. $Dist_x(D_i, D_j)$ is the distance between D_i and D_j using metric x. Similarity/distance equations provide a basis for creating clusters.

The exact definition of similarity is dependent on the clustering algorithm and the data-set. We have already explained the basic vector space model's definition of similarity (inverse of distance). Using the basic vector space model, similarity may be calculated using the cosine (Equation 2.1).

The cosine value is between -1 and 1, with 1 indicating complete similarity, and -1 complete dissimilarity. If each D_i has been normalized such that $||D_i|| = 1$, then the cosine calculation can be substituted by one minus the *Euclidian distance metric* in Equation 2.2, where m is the number of dimensions in the vector space.

$$sim_{cos}(D_i, D_j) = \frac{D_i \cdot D_j}{||D_i|| ||D_j||} \tag{2.1}$$

$$dist_{euc}(D_i, D_j) = \sqrt{\sum_{k=1}^{m} (D_{ik} - D_{jk})^2} \tag{2.2}$$

$$dist_{man}(D_i, D_j) = \sum_{k=1}^{m} |D_{ik} - D_{jk}| \tag{2.3}$$

$$dist_{mah}(D_i, D_j) = \sqrt{(D_i - D_j)^T \Sigma^{-1} (D_i - D_j)} \tag{2.4}$$

Two other vector distance definitions used in clustering are *Manhattan* (Equation 2.3) and *Mahalanobis* (Equation 2.4). Manhattan distance is simply the sum of the differences over every dimension. It is also referred to as the 'city block distance'. Mahalanobis distance is used in the EM family of clustering algorithms, presented in the clustering algorithm section. x^T is the transpose of x, and Σ is the covariance matrix of the multivariate vector that produced D_i and D_j. Numerous other distance metrics exist [3], including Minkowski, Chebyshev, and others, but these four are the most common by far.

Data-sets of objects with Boolean features can use some of the metrics from Equation 2.1-2.4 by representing their features as 1s and 0s. However, it is far more common to use distances designed specifically for Boolean vectors. Some of these distances are presented in Table 2.1. The matching coefficient is simply the number of Boolean values that overlap in D_i and D_j. The Dice and Jaccard

Equation Name	Similarity
Matching Coefficient	$sim_{match}(D_i, D_j) = \|D_i \bigcap D_j\|$
Dice Coefficient	$sim_{dice}(D_i, D_j) = \frac{2\|D_i \bigcap D_j\|}{\|D_i\|+\|D_j\|}$
Jaccard Coefficient	$sim_{jacc}(D_i, D_j) = \frac{\|D_i \bigcap D_j\|}{min(\|D_i\|,\|D_j\|)}$

Table 2.1: Similarity measures for binary vectors.

coefficients are designed to compensate for each D_i being a different length, and as such are more robust. The Jaccard coefficient may be extended (Strehl [91]) so that it works with numeric features as well, as long as they are non-negative). Equation 2.5 gives the extended Jaccard equation, where D_i^T is the transpose of D_i.

$$sim_{xjacc}(D_i, D_j) = \frac{D_i^T D_j}{|D_i|^2 + |D_j|^2 - D_i^T D_j} \tag{2.5}$$

Another method for defining similarity is a *neighbourhood* system. Suppose that for all D_i, D_j, we know whether D_i and D_j are similar. We will then refer to D_i and D_j as linked, and write this as $linked(D_i, D_j)$. One way to use these links is to calculate the neighbourhood overlap between all D_i,D_j pairs. Equation 2.6 gives the neighbourhood overlap of D_i and D_j. Neighbourhood clustering techniques are popular because they can handle any kind of data. However, similarity calculations must still be performed by some other algorithm to obtain the links with which to calculate neighbourhood overlaps.

$$sim_{neighbourhood}(D_i, D_j) = |\{D_k : linked(D_i, D_k) \wedge linked(D_j, D_k)\}| \tag{2.6}$$

There are many other methods for determining similarity, likely as numerous as clustering algorithms themselves.

2.1.6 Noise and Outliers

Given a data-set, it is possible that some of the objects are *noise* or *outliers* (see Figure 2.5). The former word is commonly used to refer to erroneous data, the latter to data that, while not erroneous, does not fit in with the other objects of the data-set. While outliers and noise are not the same thing, they are both treated the same way in clustering: as objects to be detected and isolated so as to not corrupt the clusters produced by the more typical, accurate data. As such, we will refer to outliers and noise as the same phenomenon.

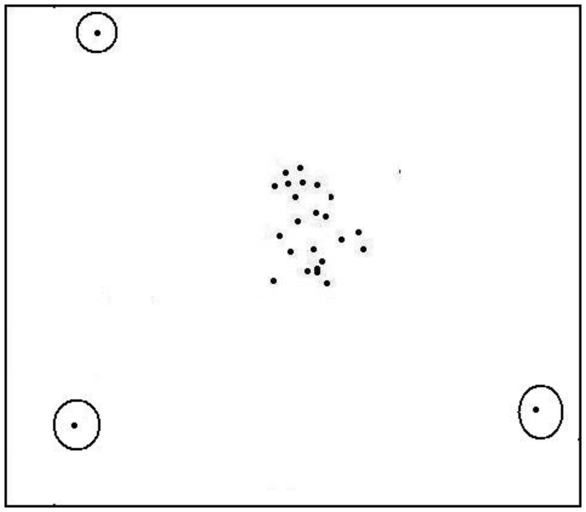

Figure 2.5: Outliers and noise in clustering
A data-set with outliers/noise circled. In this case, it is obvious that these three points are outliers, but in practice, identifying outliers is difficult and often requires some subjective definition of what an outlier/noise is.

Outliers may be useful in their own right, as it is often informative to isolate and examine such entities. Regardless of the motivation, almost all of the more recent clustering algorithms have some form of outlier detection/isolation integrated into them. We will explain how various algorithms handle outliers in the section on clustering algorithms (2.2).

2.1.7 Data Streams versus Batch Data

One of the major issues with clustering algorithms is the ability to handle *data streams*. A data stream, as the name implies, is a source that generates data objects in real-time. For example, a news wire is a data stream of article objects. This is in contrast to *batch data*, in which the entire collection of objects is available at once (for example, a collection of data from a consumer survey).

The ability to handle data streams is beneficial as many data-sets are produced in real-time, and any algorithm that works on data streams can work on batch data as well (by simply submitting the batch objects as a stream). Many algorithms assume batch data is available, and those that can handle streams are often referred to as *incremental* to denote their stream-handling ability. Unless noted when explained in Section 2.2, a clustering algorithm is batch-data-based.

2.1.8 Evaluation of Clustering Algorithms

There are several ways to evaluate the quality of a clustering algorithm. In situations where data-sets have labels, an obvious solution is to compare the clusters obtained against the labels, with the hope that clusters match up to distinct labels. *Purity* (used by Mandhani et al. [73] for example) is one means of quantifying the relatedness between a cluster and a particular label. It is calculated as follows: for cluster C_i, the purity with respect to the label L_j from a set of labels L, where C_i has n_{ij} data objects of label L_j, is $n_{ij}/|C_i|$. Purity may also be referred to as *precision*. *Recall* is another quality metric, and may be thought of as the 'flip side' of purity: while purity tells you to what degree a cluster belongs to a certain label, recall tells you the portion of a label that belongs to a cluster. Continuing with the this notation, the recall of cluster C_i with respect to L_j, where there are n_j data objects of label L_j in the entire data-set, is n_{ij}/n_j. While either precision or recall on its own misses something about quality, combining the two into a single metric, referred to as an *F-measure*, results in a much better calculation of quality. F-measures come in a variety of forms, for example, the F-measure used for a single cluster in Zhao and Karypis' hierarchical clustering review [102] is given in Equation 2.7. The overall quality of a clustering solution tree is calculated as $\sum\limits_{L_j \in L} \frac{n_j}{|D|} \max\limits_{C_i \in T} F(C_i, L_j)$, where T is the entire tree of clusters produced during the clustering algorithm.

$$F(C_i, L_j) = \frac{2\frac{n_{ij}}{|C_i|}\frac{n_{ij}}{n_j}}{\frac{n_{ij}}{C_i} + \frac{n_{ij}}{n_j}} \tag{2.7}$$

HFTC uses a metric almost identical to Zhao and Karypis', except HFTC considers only top-level clusters when evaluating F-measure, e.g., HFTC's metric applies only to a specific number of clusters, whereas Zhao and Karypis' metric is calculated by running a clustering algorithm until only one cluster remains then evaluating the metric. The F-measures from Zhao and Karypis' method are strictly equal to or greater then those produced by HFTC (as they include the clusters HFTC uses, and others as well, and we are taking maximums).

Without labels, the variety of potential quality measures is somewhat limited, as only the similarity definitions and the properties present in the data objects themselves can be used. Most often, unlabelled data quality metrics are based on the particular metric the algorithm is trying to maximize/minimize. For example, most entropy-based algorithms have been evaluated by their creators using entropy. This may sound like a dubious practice, but it does make sense. Since the authors are inferring that a cluster is based on a certain principle, it makes sense to evaluate the output quality based on that principle. Unfortunately, an algorithm trying to maximize a quality x is often compared against another algorithm that was designed to maximize some different quality metric y, but the evaluation is done using x on both. An example of such a comparison can be found in Barbará et al. [19], in which they compare COOLCAT, an entropy based algorithm, to ROCK, which is based on links, using only entropy. As a result, it is often hard to determine which algorithm is performing better.

Besides the labelled and unlabelled issue, we must also consider the source of test and training data on which an algorithm is run. There are a huge variety of sources (non-standardized) and many algorithms function much better on certain data types (text, geological data, image data, etc.) and poorly (or not at all) on others.

In the next section, we move on to a discussion of specific types of clustering algorithms. The general issues that all such algorithms must deal with as their common purpose is clustering are:

- Feature types used: Boolean, categorical, and/or numerical.
- Handling of outliers/noise.
- Applicable to real-time data (data streams).
- Invariant clustering (order for considering data points has no effect on final clustering).
- Scalability (including high dimensionality, high number of clusters, high number of points, etc.)

- Type of clusters produced: disjoint (a point may belong fully to any number of clusters), hard (every point is in one cluster), soft (probabilistic assignment to clusters), etc.
- Parameters and the difficulty of parameter estimation.
- Easily understandable results.

We will discuss these various issues as we review the algorithms. Major categories are presented, then subcategories. Each subcategory has several algorithms representative of its type. It should be noted that many algorithms fall under multiple categories, so that the organization scheme used here is by no means perfect.

We point out a curious aspect of clustering algorithms before we move on to describing them. It is somewhat of a convention to ignore the fact that vector operations are not constant time. That is to say, for example, a cosine operation on two vectors is $O(1)$ when it is actually $O(m)$, m being the length of each vector. Unless we note otherwise or mention the variable m directly in our complexity notation, we use the former definition of time complexity, e.g., a single vector operation is $O(1)$. Also, note that many algorithms assume that they are supplied with pairwise similarities for the entire dataset and do not consider the computation of these similarities as part of their run-time cost.

2.2 Clustering Algorithms

Here we present an overview of various clustering methodologies/algorithms. The overview is organized by major type, sub-type, then specific algorithms.

2.2.1 Density-Based

Density-based clustering algorithms are used mainly with spatial databases. Data objects for such databases typically consist of m-dimensional points (m numeric features), although generalizations to objects with categorical features exist. The defining characteristic we use for density-based clustering algorithms is: clusters are sets of data points that meet some density requirements.

DBSCAN Family - Density Connected Sets

DBSCAN (Ester et al. [38]) is a numeric feature density-based clustering algorithm. There are two input parameters for DBSCAN: ε and $MinPts$. The central concept of DBSCAN is the density-connected set. It may be conceptually defined as follows: Given some point x, draw an arc from x to all y where $dist(x, y) \leq \varepsilon$. Repeat this recursively. When all arcs are drawn, erase all points with number of arcs less than $MinPts$. The remaining connected-points are a density-connected set. In DBSCAN, such sets are clusters (Algorithm 2 illustrates conceptually how DBSCAN clusters a dataset uses density-connected sets).

Algorithm 2 DBSCAN

```
1: Input: $D$: dataset
2:        $\varepsilon$: maximum neighbourhood distance
3:        $MinPts$: minimum neighbourhood size
4: $C =$ empty set of clusters
5: while $|D| > 0$ do
6:    $p =$ random point in D
7:    $N =$ density connect set of $p$ with diameter $\varepsilon$
8:    $D = D - N$
9:    if $|N| \geq MinPts$ then
10:      $C = C \cup N$
11:   end if
12: end while
13: return C
```

DBSCAN is both simpler and faster than many other clustering algorithms: using advanced data structures for indexing it runs in $O(n \log(n))$ time. The algorithm allows for arbitrarily shaped clusters (Figure 2.6) that most other clustering algorithms cannot detect. Outliers are handled well (consider the $MinPts$ value, we may note that an outlier will not have that many points within ε). GDBSCAN (Sander et al. [83]) is a generalized version of DBSCAN designed to handle non-spatial (non-numeric) features as well as spatial features. (G)DBSCAN's quality is highly sensitive to its parameters, making it ill-suited to large/high dimensionality datasets. The 4C algorithm (Böhm et al. [23]) defines many of the same concepts as DBSCAN, except that they are applied with respect to correlation between points and not density. It yields better results than DBSCAN, but requires a run-time of $O(m^2 n \log(n) + m^3 n)$ on high dimensional data (m features).

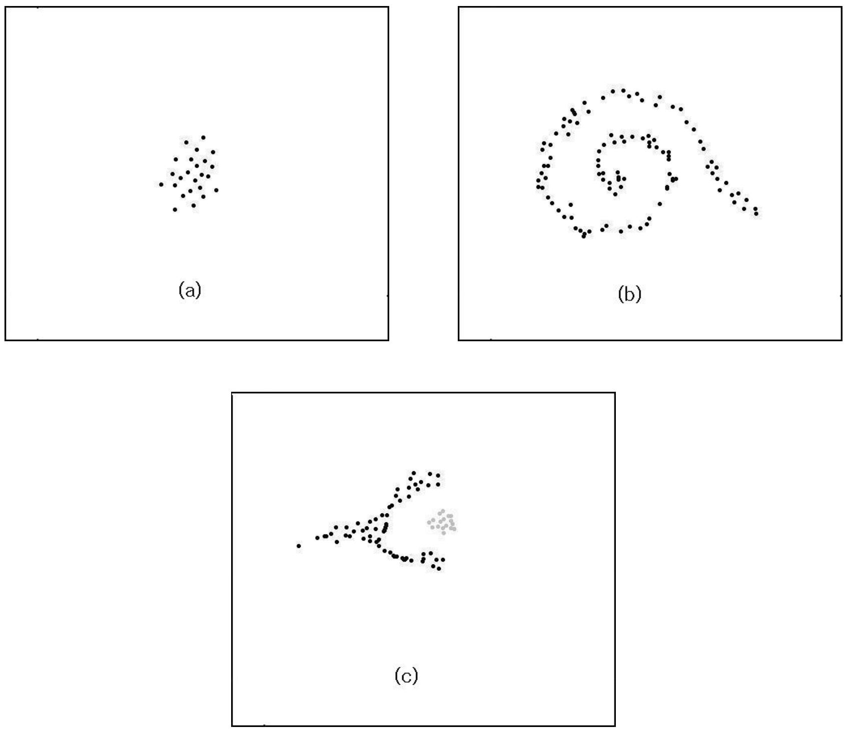

Figure 2.6: DBSCAN clusters
DBSCAN can detect both spherical clusters as shown in (a) and oddly shaped ones such as the spiral in (b) or the horizontal 'y' in (c). We refer to any non-spherical cluster, such as (b) and (c) as *arbitrarily shaped*.

FDBSCAN (Fuzzy DBSCAN, Kriegel & Pfeifle [67]) is DBSCAN designed to handle fuzzy data. OPTICS (Ankerst et al. [12]) is another extension of DBSCAN. Given a fixed $MinPts$ in DBSCAN, every high-density cluster is contained within a lower density one (Figure 2.7). OPTICS uses this situation to create a *cluster-ordering* of data points. Conceptually (although the algorithm works differently in practice for speed) OPTICS first orders pairs of points by ascending distance. The algorithm then iterates over this sorted list and arcs are drawn between each pair of points it iterates over. As the DBSCAN clusters present change, clustering-order information is recorded in the order obtained. Ordering is by distance (since the point pairs are so ordered). Now, to get a clustering with ε equal to some value X, we need only read the summary until X distance. The actual algorithm for generating the cluster-ordering is somewhat complex and runs in $O(n \log(n))$ time, but the output has the following highly desirable property: Given any ε value, creating a DBSCAN clustering can be done in $O(n)$ by sequentially moving through the cluster-ordering. In effect, OPTICS is like infinitely many DBSCAN runs of a dataset with $0 \leq \varepsilon \leq \infty$.

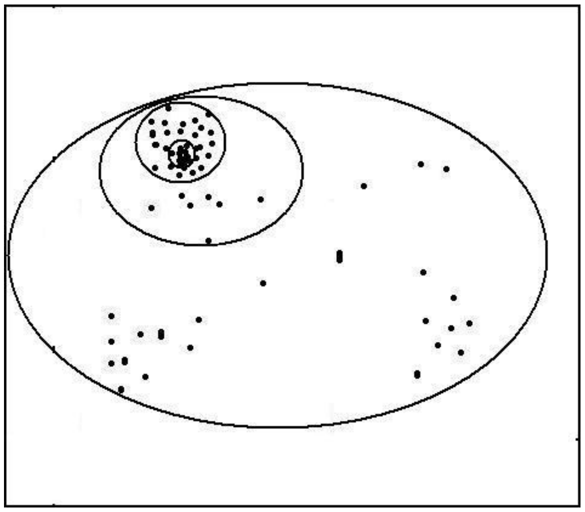

Figure 2.7: Cluster density observation in OPTICS
As illustrated here the observation that a high density clustering is always contained within a lower density one allows the construction of OPTICS' cluster-ordering.

Unlike OPTICS and DBSCAN, DBCLASD (Xu et al. [100]) is good for larger databases where parameter estimation is problematic. In addition, it is solidly grounded in mathematics, relying on an assumption of uniform distribution to generate clusterings. DBCLASD is slightly slower than DBSCAN. Unfortunately, the assumption of uniform distribution means that DBCLASD is not suited to some kinds of data (text word counts, for example). Foss & Zaïane [40] give another parameterless algorithm called TURN*. TURN* is interesting in that it takes existing point similarities, and scales them iteratively. For each scaling of points, a basic density-type clustering algorithm is run. Then, the clustering is evaluated to judge if it of sufficiently high quality.

DENCLUE - Density Function Clustering

A different approach to density-based clustering uses *density functions.* Hinneburg & Kim [54] illustrate this with DENCLUE (DENsity based CLUstEring). The basic concept is that every object x has an *influence* with respect to every other object in the dataset y. The influence between two points x and y is written as $f(x, y)$. Influence may be some form of similarity equation, scaled by division by a parameter σ. A data point's total influence is $f(x) = \sum\limits_{y \in D, y \neq x} f(x, y)$k (a *density function*). Given the density function, DENCLUE can form either centre-based clusters by finding local maxima in the graph, or arbitrarily-shaped clusters by linking sequences of points with density functions values above a parameter ε. σ is used to control the granularity of the searching for maxima. DENCLUE runs very fast, in sublinear time.

2.2.2 Grid-Based

Grid-based clustering algorithms are very similar to density-based clustering algorithms, but where density-based algorithms are centred around points, grid-based methods partition the vector space into chunks (see Figure 2.8 for an example). Then, density within these chunks is used to build a clustering.

Static Grid Partitioning

Many useful algorithms can be designed around a simple, static-type partitioning of the data space into a grid. WaveCluster (Sheikholeslami et al. [87]) offers a novel clustering approach that treats the data space as waves. The first step involves 'cutting' the data space into blocks (*quantizing*, essentially making a grid). At this point, wave theory can be applied to detect connected blocks. Connected blocks are considered to be clusters. This algorithm runs in $O(n)$ time. Multiple clusterings are produced by a single run of WaveCluster, equivalent to progressively increasing the definition of connected blocks (analogous to relaxing the similarity required for objects to be in the same clustering). Unfortunately, the algorithm does this by skipping rows of quantized blocks. This results in progressively cruder clusterings.

STING (Wang et al. [95]) was implemented a year after DBSCAN. It uses the concept of *summarization*. The vector space is partitioned recursively into blocks of four (see Figure 2.8) until a desired granularity (cell size) is reached. Summary information is then computed for each cell. The use of cells rather than individual points can have benefits. If we have k cells, $k << n$, and store only cell summaries, the memory requirement is less. The STING algorithm initially examines the top-level partitioning (a single cell) and recursively examines finer regions as required to create a clustering. For example, it may determine that the entire dataset may be a cluster. It will then look at the dataset split into four cells. If two cells are not clusters, they are flagged and not processed further. The other two cells are each split into four and each piece is examined, and so on. This yields a run-time of $O(k)$. If each cell contains only one (or zero) data points, STING's output is equivalent to DBSCAN. STING+ (Wang et al. [96]) builds on STING, and is designed to handle data streams. It defines a language in which one can specify 'triggers' that will fire when certain conditions are met in the hierarchy of grid cells.

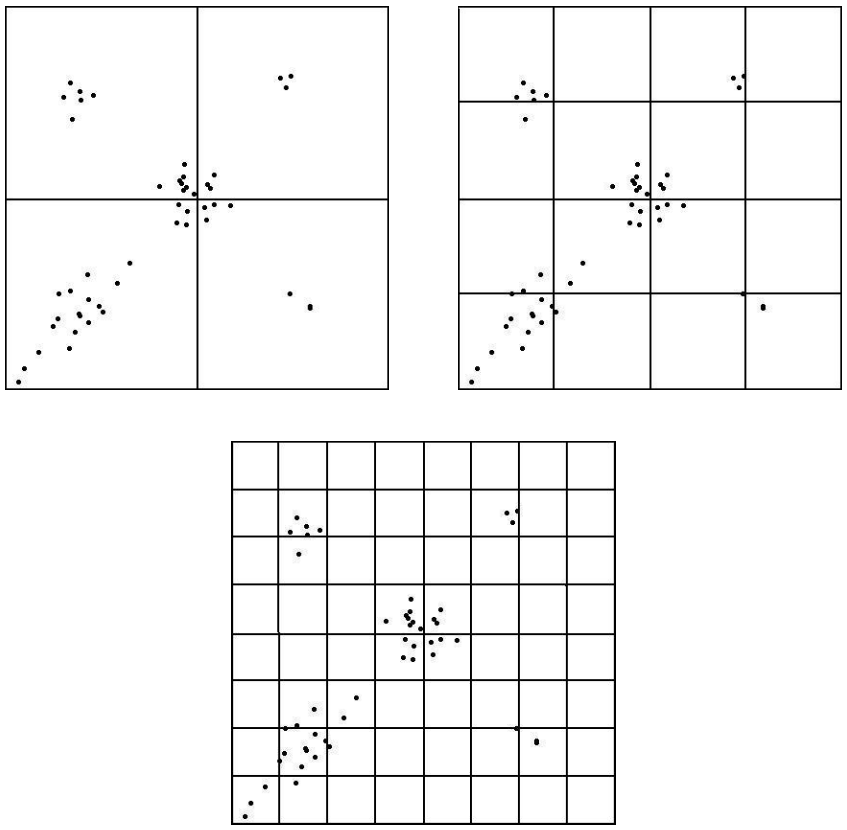

Figure 2.8: STING's partitioning approach
Note that blocks are always split perfectly into four. Applied recursively, this creates a uniform grid.

FC (Fractal Clustering, Barbará & Chen [18]) uses the concept of the *fractal dimension*. Given a cluster C consisting of r boxes (hypercubes), let $N(r)$ be the number of such boxes that contain at least one point. A *box-counting plot* is a log-log plot of $N(r)$ versus r. The fractal dimension used in FC is the negative slope of that plot (Figure 2.9). Initially, the data space is divided evenly into hypercubes.

Clusters are created by bootstrapping a starting algorithm that clusters a small fraction of the points. Then, further points are added one at a time. A point x is added to the cluster C, where C's fractal dimension will change the least among all clusters by adding x. Minimizing the change in fractal dimensions roughly equates to maintaining the structure of a clustering as much as possible. Splitting and merging of clusters is also possible in FC. FC can be used on streaming datasets, as it constructs clusters incrementally.

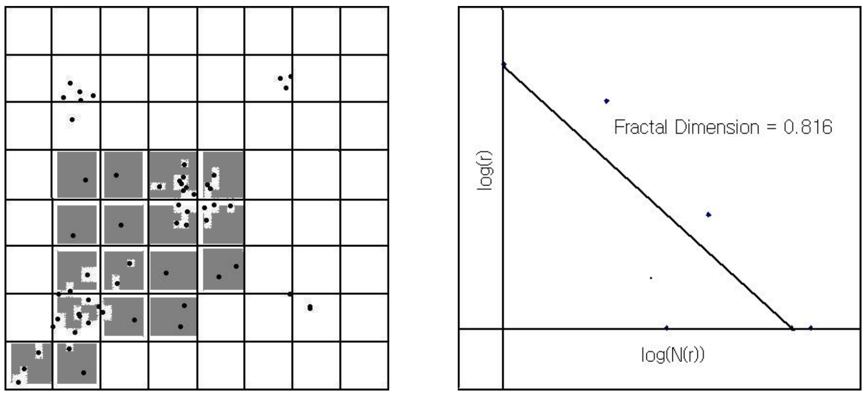

Figure 2.9: FC: Box counting and the fractal dimension
A dataset partitioned into blocks. The grey-shaded blocks represent a cluster. The accompanying graph shows the box-counting plot and the fractal dimension (calculated using linear regression in this case).

Adaptive Grid Partitioning

Thus far, all the grid-based algorithms we have discussed partition the data space into perfect grids. If we remove this restriction, we are immediately confronted with the question: how can the data space be adaptively partitioned in a useful and meaningful way? Adaptive grid partitioning methods address this. As an example, BANG-Clustering (Schikuta [85], Schikuta & Erhat [86]) operates in the following manner: Start with one block containing the entire data set. With each iteration, all blocks remaining are split in half. Each block has its *density index* calculated, which is its volume divided by the number of points in it. If the density index is above some threshold, the block becomes a new cluster centre. The remaining non-cluster blocks are sorted by density index, and processed further, either merging into existing cluster centers or just being passed on to the next iteration. Only adjacent blocks may merge. This method creates a hierarchy of clusters as grid squares/recentangles.

OptiGrid (Hinneburg & Keim [55]) is another algorithm that uses intelligent grid partitioning and belongs to a family of dimensionality-reduction type clustering algorithms referred to as *projective clustering*. The meaning of "projection" for this family may be taken literally. An example is projecting a three-dimensional universe onto a two-dimensional screen. In general, in projective clustering an n dimensional dataset is projected to an m dimensional one, $m < n$.

Optigrid first creates a projection plane for every dimension of the dataset. For example, a three-dimensional data space x, y, z would have three projections, one with only x, y co-ordinates, the other y, z, and the other x, z. Then, for each of these projections, a set of 'cuts' which split the plane into two are calculated. The quality of a cut is defined by the amount of density function (as in DENCLUE [54]) it goes through, with lower density cuts being higher in quality. All the cuts for each projection are pooled together, then the best q cuts are used to partition the dataset. OptiGrid is then run recursively on each partition until no more good cuts can be found. Optigrid runs superlinear to n.

2.2.3 Partition Reallocation Clustering

Partition reallocation clustering algorithms rely on iterative refinement. In iterative refinement, the entire dataset, or sample of the dataset, is clustered, then certain properties of the clustering are evaluated. These properties are used to re-cluster the dataset into a higher quality clustering. This refinement step may be done until the clustering converges (e.g., two consecutive clusterings have exactly the same structure) or some other criterion is satisfied. Note that partition reallocation algorithms suffer the 'chicken and the egg' problem, in order to produce a clustering, we must have certain statistics. In order to have these statistics, we must have a clustering. This problem forces partition reallocation algorithms to bootstrap an initial clustering, which is a difficult task in itself. Note that the term refinement is specific to what the algorithm is trying to minimize or maximize (the *objective function*). We will hereafter refer to the objective function of an algorithm X as ϕ_X. Most partition reallocation algorithms are non-hierarchical. Some of the oldest and best-known clustering algorithms fall into this class, and almost every field of science has applied these in one form or another.

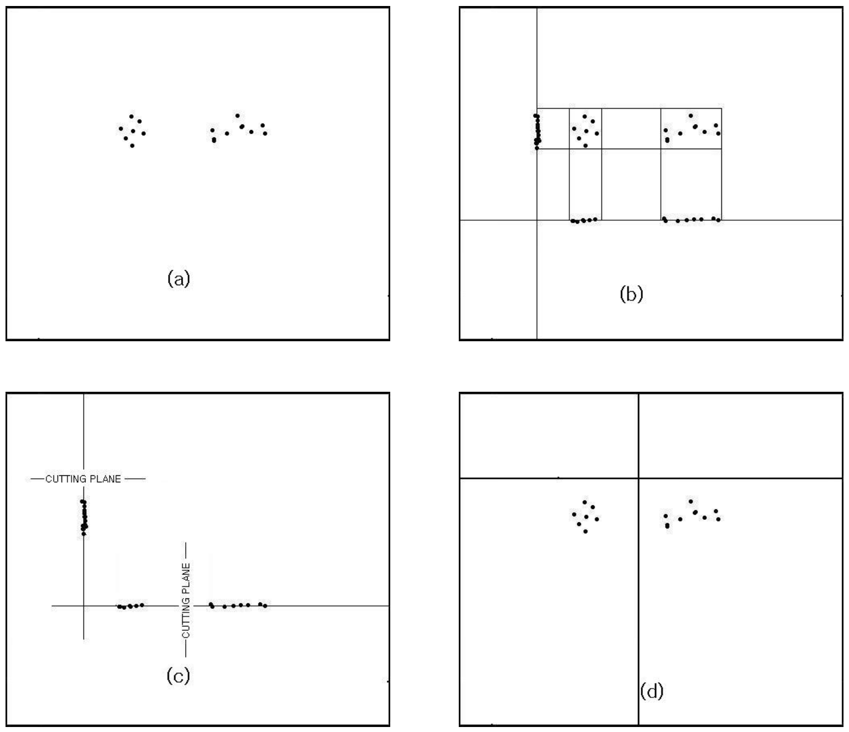

Figure 2.10: Optigrid
(a) Illustrates a sample two-dimensional data set. (b) shows the two projections (onto x and y). (c) depicts possible low density cuts. (d) illustrates when both of these cuts in (c) are applied to (a).

K-Means Methods

K-Means (Hartigan [53]) is the most well-known and widely applied clustering algorithm, with so many variants that enumerating all of them would be a feat! This may be attributed to its age as well as its simplicity, and that it produces good enough results in most applications. The 'K' in K-Means is a parameter dictating the number of clusters to be used throughout the algorithm. *Centroids* are the key elements of the K-Means methodology. Given a cluster C_i, the centroid of C_i, which we denoted as c_i, is defined by Equation 2.8.

$$c_i = \frac{\sum\limits_{D_j \in C_i} D_j}{|C_i|} \tag{2.8}$$

Thus, a cluster's centroid is a single point with every dimension's value equal to the average of that dimension within the cluster.

Algorithm 3 K-Means

```
Input: $D$: dataset
       $C$: initial set of clusters from $D$
       $\omega$: improvement threshold
loop
  $C_{old} = C$
  for all $C_j \in C$ do
    $c_j = computeCentroid(C_j)$
    $C_j = \{\}$
  end for
  for all $D_i \in D$ do
    $c_j =$ closest centroid to $D_i$
    $C_j = C_j \cup D_i$
  end for
  if $(C_{old} = C)$ or $(\phi_{KM}(C_{old}) - \phi_{KM}(C) > \omega)$ then
    break
  end if
end loop
return C
```

Algorithm 3 shows the K-Means algorithm. Basically, in each iteration, centroids are re-computed and data points are assigned to the closest centroid. Iterations terminate on convergence or when the solution improvement is below a threshold (using the objective function in Equation 2.9).

$$\phi_{KM}(C) = \sum_{C_i \in C} \sum_{D_j \in C_i} \|D_j - c_i\| \tag{2.9}$$

K-Means has an appealing theoretical basis, as its objective function is assured to improve every iteration until convergence is reached. One criticism of K-Means methods (and most partition reallocation approaches) is its favoritism towards nicely shaped (circular) clusters. Some other problematic issues include: bounding time, probabilistic membership, selecting the number of clusters, applicability to Euclidian data spaces only, and selecting good initial starting centroids (*seeds*).

The run-time complexity of K-Means is non-trivial to bound. It is unclear how long convergence takes. Har-Peled & Sadri [52] examined the convergence-time issue and produced upper and lower bounds for some variations of K-means (such as integer-only data), but there are no bounds for basic K-means with real data without making some assumptions about metrics/data.

With regards to selecting the number of clusters for K-Means, one method can be found in Pelleg & Moore 2000 [79]. Another issue is that basic K-Means uses Euclidian distance, and is therefore not applicable to categorical data. Huang [58] presents a categorical version of K-Means called *K-Modes*. K-Modes centroids use the *mode* value for each feature (the most commonly occurring category). Distance (and therefore quality as well) is defined in terms of the matching coefficient, as shown in Table 2.1. Interestingly, Huang also presents an algorithm for combining categorical and numeric data, which he refers to as the *K-Prototypes*.

On-line (Adaptive) K-Means [72] is designed for streaming data. Initially, K points are selected to be clusters of size one (and therefore each is a centroid). Then, non-clustered points are considered one at a time, and assigned to the cluster with the nearest centroid. After each assignment, the centroid of the affected cluster is recomputed. With an n-points dataset, a maximum of n/K operations are needed for centroid recalculations each iteration, and there are a maximum of n iterations for a run time of $O(n^2/K)$. On-line K-Means is very sensitive to order of point submission. Ordonez [77] gives an improved version of On-line K-Means called *Incremental K-Means*. Rather than use random points for initializing clusters, Incremental K-Means uses a sample of the dataset for a good initial set of clusters. Then, it recomputes the clustering and centroids by adding small batches of n/L points, where L is a parameter. If $L = n$, it is equivalent to On-line K-Means. *Scalable K-Means* [25] can also handle a data stream, but was actually designed for very large datasets (those that can't fit into main memory). Like Incremental K-Means, it samples batches at a time. However, it uses an interesting approach where it determines which points must be kept in memory, and which can be dumped and replaced with summaries to greatly reduce memory consumption.

Bisecting K-Means (Steinbach et al. [90]) has been shown to work considerably better then basic K-Means, in particular with text. It produces a hierarchy, so may be more accurately classified under hierarchical clustering, but since it is based on K-Means we mention it here. Initially, every point is in one large cluster. For each iteration, a cluster C_i is selected to be partitioned in two. Then, a simple one-pass version of K-Means is performed using the following steps: find two centroids in C_i, denoted c_{i1} and c_{i2} as defined by equations 2.11 and 2.12. Then, perform a normal K-Means step 2 to create two new clusters. Note that there are better ways to pick the two centroids to split a cluster on then by random. Bisecting K-Means may be generalized to n-secting K-Means, where in each iteration a cluster is split into n smaller clusters.

$$\begin{aligned} c_{i1} &= random(C_i) && (2.10) \\ c_{i2} &= c_i - (c_i - c_{i1}) && (2.11) \end{aligned}$$

As with other clustering algorithms, K-Means has variants designed to deal with probabilistic cluster membership. As it turns out, the centroid computation in Equation 2.8 can be generalized to probabilistic membership. Equation 2.12 illustrates this. $P(D_j|C_i)$ is the probability that D_j is a member of cluster C_i, and $w(D_j)$ is the weight of point D_j. Note that if all points have weight one, and $P(D_j|C_i)$ equals one if C_i is the closest centroid to D_j and zero otherwise, Equation 2.12 reduces to the regular K-Means centroid calculation of Equation 2.8.

$$c_i = \sum_{D_j \in D} P(D_j|C_i) w(D_j) D_j \tag{2.12}$$

Fuzzy K-Means (Bezdek [22]) and *Harmonic K-Means* (Zhang et al. [101],Hamerly & Elkan [50]) both employ probabilistic membership. In Fuzzy K-Means, $w(D_j)$ is always one, and $P(D_j|C_i)$ is defined in Equation 2.13. The objective function is given in Equation 2.14.

$$Pr(D_j|C_i) = \frac{\|D_j - c_i\|^{-2/(r-1)}}{\sum_{C_k \in C} \|D_j - c_k\|^{-2/(r-1)}} \tag{2.13}$$

$$\phi_{FKM}(C) = \sum_{C_i \in C} \sum_{D_j \in D} Pr(D_j|C_i)(D_j - c_i)^2 \tag{2.14}$$

The r value may be thought of as a fuzz factor. As r approaches one, the algorithm becomes similar to regular K-Means. Larger values for r allow more effect on the centroid by lower probability memberships. Harmonic K-Means is a better fuzzy algorithm, developed later. We will not mention its specifics here, only that it incorporates the interesting notion that a point is well-placed if it is close to one centroid and it is far from other centroids as well.

Medoids

A simple adjustment to K-Means is to use *medoids* instead of centroids. A cluster's medoid is the data point within the cluster that is closest to the centroid. Equation 2.15 shows how it is calculated.

$$m_i = \arg\min_{D_j \in C_i} \|D_j - c_i\| \tag{2.15}$$

$$\phi_{PAM}(C) = \sum_{C_i \in C} \sum_{D_j \in C_i} \|D_j - m_i\| \tag{2.16}$$

PAM (Kaufman & Rousseeu [64]) is a medoid-based method closely related to K-Means. For each iteration, every pair of points (x,y), where x is a medoid and y is a non-medoid, is evaluated for swapping potential. That is to say, x will no longer be a medoid and y will be one instead. For each of these swaps, the quality is calculated using Equation 2.16. If this best swap would degrade quality the algorithm ends, otherwise the best swap is performed and iterations continue. PAM is better then K-Means, but costly, with an $O(K(n-K)^2)$ run-time. CLARA (Kaufman & Rousseeu [64]) is an extension of PAM which draws random samples of size $40 + 2k$ points and performs the PAM algorithm on these samples. After this, all points of the dataset are assigned to the cluster of the nearest medoid. Samples are drawn multiple times, but only the best result is returned. The run-time of CLARA is $O(K^3 + Kn)$. Since $K << n$, this reduces to $O(n)$, much faster then PAM. However, CLARA is vulnerable to bad sampling. CLARANS (Kaufman & Rousseeu [64]) is an improved CLARA. It creates a graph, where each node is a different set of possible medoids (Figure 2.11). Arcs connect those nodes that differ by only one medoid. The algorithm is then a simple graph traversal problem, starting at a random point and traversing arcs that lead to higher quality, as defined by Equation 2.16. CLARANS uses multiple random start points in the graph, and returns the best quality found from all graph traversals. It outperforms CLARA in average solution quality.

Probability Distribution Methods

In the probability distribution framework, the viewpoint is that a dataset has been generated by a collection of distributions (*generative* clustering). Let us say we have k distinct distributions that are thought to have generated a dataset. Every point in the data is assumed to have been generated by one, and only one of the k

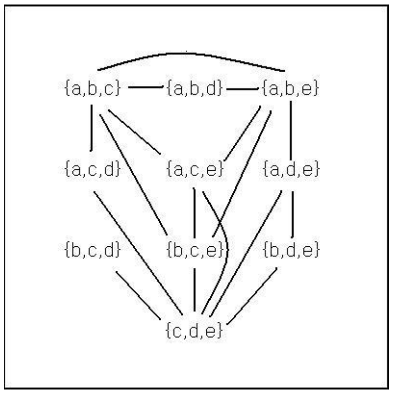

Figure 2.11: CLARANS medoid graph
The graph for CLARANS operating on a dataset of points $\{a, b, c, d, e\}$ with the number of medoids set to three. Arcs exist between nodes differing by one medoid.

distributions. This leads to a natural definition of a cluster: all the points generated by one distribution. Each distribution can also be referred to as a *model*, and combined together they are known as a *mixture model*. Unfortunately, we do not know which data point belongs to which distribution. We can, however, compute the probability that a point D_i was produced by any specific distribution numbered j. As cluster j (C_j) is produced by distribution j, we refer to this probability as $P(D_i|C_j)$. The goal of a probability distribution clustering algorithm is usually to maximize the *likelihood* that the dataset D was generated by the set of distributions C. Equation 2.17 is the likelihood of D given the distributions in C. π_j is the weight of the probability distribution j, under the restriction $\sum\limits_{C_j \in C} \pi_j = 1$.

$$L(D|C) = \prod_{D_i \in D, C_j \in C} \pi_j P(D_i|C_j) \tag{2.17}$$

The log of Equation 2.17 is used as an objective function (Equation 2.18) in *Expectation Maximization* (*EM*, Dempster et al. [34]). EM is not a specific clustering algorithm, but rather a general learning method that can be applied in almost all situations. It follows a two-step process. The *Expectation (E)* step is the process of calculating expectations, which we denote as $E(D_i|C_j)$ (the likelihood of D_i belonging to cluster C_j). First, we estimate $P(D_i|C_j)$'s based on the distributions, and then we calculate the $E(D_i|C_j)$'s. $P(D_i|C_j)$'s are not normalized, but are just the probability that C_j generated D_i without considering other elements in C.

$E(D_i|C_j)$'s are normalized such that $\sum\limits_{C_j \in C} E(D_i|C_j) = 1$. When the EM algorithm terminates, it is these E values that are used as probabilistic membership values. The *Maximization (M)* step attempts to maximize Equation 2.18 by adjusting the distributions based on the $E(D_i|C_j)$s. Clearly, this is a cyclic problem. E depends on M, and M depends on E. This is the same problem K-Means encounters as it needs an initial clustering to iterate. The solution is to guess as at attributes (mean, variances, etc.) of the distributions and start at the E step.

$$\phi_{EM} = \log(L(D|C)) \tag{2.18}$$

EM iterates until convergence or the improvement in Equation 2.18 is below a threshold. EM clustering may be even more popular than K-Means. As with K-Means, its number of variants is large. It has many appealing properties: a strong mathematical basis, easily understood results, and guaranteed improvement per iteration. Unlike K-Means, it has built-in probabilistic memberships and the ability to handle categorical data. Like K-means, EM suffers from having to select the number of distributions to use.

Assessing the number of distributions to use in probability distribution clustering is somewhat easier then for other partitioning methods. It is simply a matter of determining if adding a new parameter(s) (distribution) improves the model. The function for determining quality given a number of parameters p always include the likelihood function and p. One such method is the *Minimum Description Length (MDL)* [80] given in equation 2.19. The best number of clusters to use is the one that yields the minimum for this equation.

$$MDL(C, D, p) = -L(D|C) + p/2 - \log(p) \tag{2.19}$$

Berkhin [21] gives a more thorough discussion of the various methods for determining the number of clusters.

On-line (Sato & Ishii [84]), and Incremental EM (Neal & Hinton [75]), like the K-Means algorithms of the same name, exist to handle streaming data. Also, the work focusing on Scalable K-Means by Bradley et al. [25] is applicable to Scalable EM as well. Thus variants of EM exist to handle streams and very large databases.

Gaussian Expectation Maximization (GEM) is a common form of EM clustering. GEM assumes D was generated by a set of multivariate Gaussian distributions. The $P(D_i|C_j)$ for the E is computed using Equation 2.20. The reader may notice that the exponent of the e is the Mahalanobis distance given by Equation 2.4. μ_j is the mean of distribution j, m is the number of dimensions, and Σ_j is the covariance matrix of distribution j. Given the $P(D_i|C_j)$'s, the $E(D_i|C_j)$'s are easy to compute (Equation 2.21).

$$P(D_i|C_j) = \frac{1}{\sqrt{(2\Pi)^m|\Sigma_j|}} e^{-\frac{1}{2}(D_i-\mu_i)^T\Sigma_j^{-1}(D_i-\mu_j)} \tag{2.20}$$

$$E(D_i|C_j) = \frac{\pi_j P(D_i|C_j)}{\sum\limits_{C_k\in C} \pi_k P(D_i \in C_k)} \tag{2.21}$$

The M step in GEM re-estimates the means (Equation 2.22), covariance matrices (Equation 2.23), and weights (Equation 2.24) of each Gaussian.

$$\mu_j = \frac{\sum\limits_{D_i\in D} E(D_i|C_j)D_i}{\sum\limits_{D_i\in D} E(D_i|C_j)} \tag{2.22}$$

$$\Sigma_j = \frac{\sum\limits_{D_i\in D} E(D_i|C_j)(D_i-\mu_j)(D_i-\mu_j)^T}{\sum\limits_{D_i\in D} E(D_i|C_j)} \tag{2.23}$$

$$\pi_j = \frac{\sum\limits_{D_i\in D} E(D_i|C_j)}{\sum\limits_{C_k\in C}\sum\limits_{D_i\in D} E(D_i|C_k)} \tag{2.24}$$

A virtue of Gaussian clusters is that they are nicely formed. By this we mean that they are circular, with a large density of points around the mean that gradually thins out moving further away. This corresponds to most people's notion of what a cluster is. MCLUST (Fraley & Raftery [41]) is an example of a Gaussian-distribution–based clustering approach. Gaussians are by no means the only distributions used in distribution clustering. AutoClass (Cheeseman & Stutz [27]) uses a wide variety of distribution types, including Gaussian, Bernoulli, and Poisson. As well, it uses learning techniques to estimate the number of distributions. A more recent work by Banerjee et al. [16] uses Von Mises-Fisher distributions in EM clustering. Von Mises-Fisher distributions are designed to deal with *directional data* (e.g., numeric vector, where $\|D_i\| = 1$). Such data representations produce good results in text clustering.

2.2.4 Information Theoretic Clustering Approaches

Information Theory, to simplify greatly, involves determining the maximum compression that data can undergo. Compression may be *lossless* (the original structure can be reconstructed perfectly) or *lossy* (it can't). This is a useful concept in clustering. Consider the Minimum Description Length principle we mentioned in Section 2.2.3 (Equation 2.19), which states that more compact solutions are better. The concept MDL provides is to produce a clustering that can be expressed in the smallest amount of space possible. Such a clustering, by MDL, is a good solution. Li et al. [71] give a more concrete explanation of how information-theoretic clustering is a good approach using *entropy*. The authors show that certain forms of entropy calculations on clusterings are related to EM algorithms (discussed previously), which are widely accepted as good techniques.

Entropy is a fundamental concept in information theory. The entropy of a random variable X is the uncertainty in that variable. Not only this, but it represents the space required for the X if it is maximally compressed. If the values that X can take are denoted $S(X) = \{x_1, x_2, ...x_k\}$, then the entropy of X, denoted $H(X)$, is given by Equation 2.25. Expanding this, the *joint entropy* of a vector of random variables D is given by Equation 2.26.

$$H(X) = \sum_{x_i \in S(X)} P(X|x_i) \log(P(X|x_i)) \tag{2.25}$$

$$H(D) = \sum_{x_1 \in S(D_i)} \cdots \sum_{x_n \in S(D_i)} P(D_1|x_1 \cdots D_n|x_n) \log(P(D_1|x_i \cdots D_n|x_n)) \tag{2.26}$$

A clustering algorithm may be designed around minimizing entropy using local minima heuristics, as finding minimum entropy is generally NP-Complete.

ENCLUS (Cheng [29]) is an entropy-based subspace-clustering algorithm. For each iteration, the number of dimensions being considered is increased by one. After determining candidate dimension subsets for clustering, each candidate has its entropy calculated by first partitioning the space into a grid (see Figure 2.8) and then applying Equation 2.25. The random variable X is a single grid cell, obtained by picking a point at random from D. Thus $P(X|x_i)$ is just $|x_i|/|D|$ (the percentage of points in cell x_i). Candidates that have an entropy below ω and a mutual information above ϵ (another entropy-based metric that quantifies the connectivity of a set of variables) are returned as good clusterings. In addition to suffering normal grid problems (see Section 2.2.2), ENCLUS requires three parameters (ω, ϵ, grid size) making optimization difficult. In addition, it is too slow when looking for subspaces of larger size (30+ dimensions, so it is not applicable to text).

COOLCAT (Barbará et al. [19]) is designed to handle categorical features using entropy. Categorical features have a natural notion of entropy with respect to a cluster. Consider a single feature j within a cluster C_k. The entropy may be calculated using Equation 2.25, where X is the feature, and $S(X)$ is the set of possible categories for the feature. $P(X|x_i)$ is $\frac{|\{D_i \in C_k : D_{ij} = x_i\}|}{|C_k|}$ (the percentage of points within C_k that have x_i for feature j. Expanding this, COOLCAT uses the joint entropy from Equation 2.26 as an objective function (Equation 2.27). After obtaining an initial clustering, points are taken one at a time, and placed in the cluster that minimizes Equation 2.27. Li et al.'s work [71] is another entropy-based method for categorical clustering that uses Monte Carlo sampling. Each point is initially placed in the same cluster. Then a point is randomly selected, and is moved to a new (or simply different) cluster if it reduces the entropy. The objective function is slightly different then COOLCAT (equation 2.28). The Monte Carlo approach makes it run faster. There are various other entropy approaches, with these three being just a sample.

$$\phi_{COOLCAT} = \sum_{C_k \in C} \left(\frac{|C_k|}{|D|} H(C_k)\right) \tag{2.27}$$

$$\phi_{LI} = \sum \frac{1}{|C|} \sum_{C_k \in C} |C_k| H(C_k) \tag{2.28}$$

2.2.5 Hierarchical Clustering

Hierarchical clustering algorithms, as their name implies, create a *hierarchy* of clusters, in contrast to *flat-table* clustering algorithms (most of the methods we have discussed this far are of this type). Data objects in flat-table algorithms are either an element of only one cluster, or an element of several clusters with a degree of probability, but there is no notion of clusters within clusters. Hierarchies of clusters can be visualized by a *dendrogram*, a tree in which properties of the parent are shared by all the children (in this case, the property is cluster membership). Figure 2.12 illustrates an example dendrogram. A real-life example of a dendrogram is the *Linnean Taxonomy tree*, which organizes all living beings by kingdom, phylum, class, order, family, tribe, genus, and then species.

The dendrogram of a hierarchical clustering can be constructed bottom-up, in which case the algorithm is referred to as *agglomerative*, or top-down, in which case the algorithm is termed *divisive*. With the agglomerative approach, each data

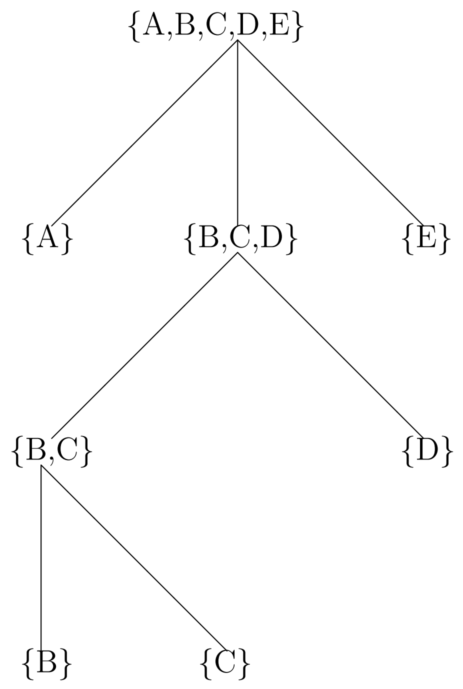

Figure 2.12: A dendrogram
Dendrograms are visualizations of hierarchical clustering structures. In this case, if we have a clustering $\{\{A\}, \{\{\{B\}, \{C\}\}, \{D\}\}, \{E\}\}$ (with {} defining a cluster's contents) the dendrogram presented here is a possible visualization of the hierarchy of clusters.

object is initially placed in its own cluster, then clusters are merged repeatedly until the desired number of clusters remain. In effect, the dendrogram is generated from the leaves upwards. Divisive algorithms initially put all data objects in a single cluster and iteratively split clusters into multiple smaller clusters.

Many of the flat-table clustering algorithms we have already discussed can be used as hierarchical clustering algorithms (see the section below). Hierarchical algorithms tend to be slower than other clustering methods, and sometimes lack a strong mathematical component. Despite this, they are quite popular since they provide a very natural view of datasets in the form of dendrograms. As well many hierarchical methods are applicable to numeric, boolean, and categorical data and may be used with any similarity metric. It should be noted that hierarchical methods are particularly applicable to text data, as such datum (in the form of documents, for example) lends itself to a hierarchical representation. A good example of a hierarchy of documents can be found on the web page for DMOZ [4].

Non-Hierarchical Methods as Hierarchical

Many of the clustering algorithms we have previously discussed, although not based directly on the construction of hierarchies, can be considered to be hierarchical clustering methods. Bisecting K-Means [50] can be thought of as a divisive hierarchical algorithm, STING [95] generates a hierarchy of grid-cells, which can also be viewed as hierarchy of clusters. OPTICS [12] creates a hierarchy of dense clusters within less dense ones. Indeed, most partitional clustering algorithm can be used to create a hierarchy if framed appropriately.

Linkage-Based Methods

Linkage-based clustering uses *linkage metrics* to cluster. A linkage metric is analogous to a similarity metric, but while similarity is defined between points, a linkage metric is defined between clusters. Using a linkage metric, clustering is done agglomeratively, so that with each iteration, the two clusters that yield a maximum value for a chosen linkage metric are merged. Equations 2.29-2.31 list the major basic linkage metrics (for more details see & Dubes [59]): *single-link*, *complete-link*, and *group average*. The group average metric is also referred to as *UPGMA* (*Unweighted Pair Group With Arithemtic Mean*) in many discussions. This should not be confused with the the UPGMA clustering algorithm used in biology, as the two are not the same.

Algorithm 4 is an algorithmic template for linkage metrics. Although it is possible to use linkage metrics to cluster divisively, it is not done in practice.

$$linkage_{SL}(C_i, C_j) = \arg \max_{x \in C_i, y \in C_j} sim_X(x, y) \tag{2.29}$$

$$linkage_{CL}(C_i, C_j) = \arg \min_{x \in C_i, y \in C_j} sim_X(x, y) \tag{2.30}$$

$$linkage_{GA}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i, y \in C_j} sim_X(x, y) \tag{2.31}$$

Algorithm 4 Agglomerative Hierarchical Clustering

```
1: Input: $D$: the dataset
2:        $s$: type of linkage clustering
3:        $k$: desired number of clusters
4: $C$ = cluster set, with each element of $D$ in its own cluster
5: while $|C| > k$ do
6:     $C_i, C_j = \arg\min_{x \in C, y \in C, x \neq y} linkage_s(x, y)$
7:     $C_l = C_i \cup C_j$
       $C = C - C_i - C_j$
8:     $C = C \cup C_l$
9: end while
10: return C
```

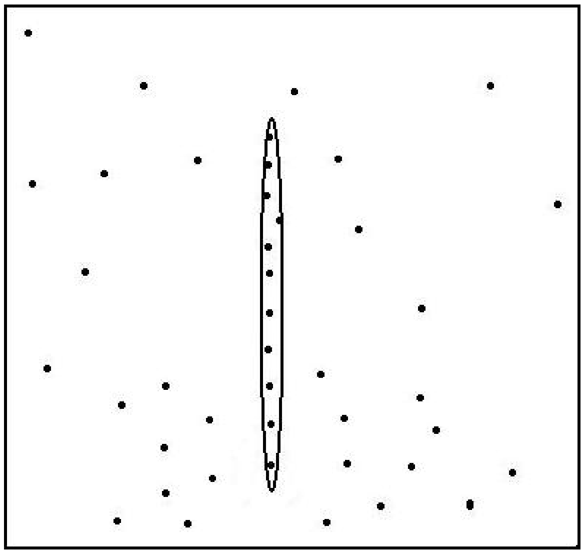

Figure 2.13: Single-link clustering
An example of how single-link clusters are 'stringy'. A thin, long cluster can easily be created as only the most similar points are used to evaluate the quality of merging two clusters (Equation 2.29).

SLINK (Sibson [88]) is an example of a single-link clustering implementation and CLINK (Defays [33]) is an implementation of complete-link clustering. Single-link clustering is known to be one of the worst clustering methods possible, often producing 'stringy' clusters such as those illustrated in Figure 2.13. While complete-link clusterings tend to be of somewhat higher quality (as shown in Figure 2.14), the clusterings are still worse then practically all of the more recent hierarchical approaches. In contrast, clustering techniques based on the group-average metric, such as Voorhees' [94] method (a direct implementation of group average with no alterations), are very effective algorithms. In fact, it is a challenge for any hierarchical clustering algorithm to beat basic group-average, regardless of the metric used to evaluate clustering quality. The major flaw of group-average clustering is the same as for other linkage metrics: a large runtime. Group-average runs in ($O(n^2 \log(n))$ time, with complete-link having a similar complexity and single-link having $O(n^2)$) run-time.

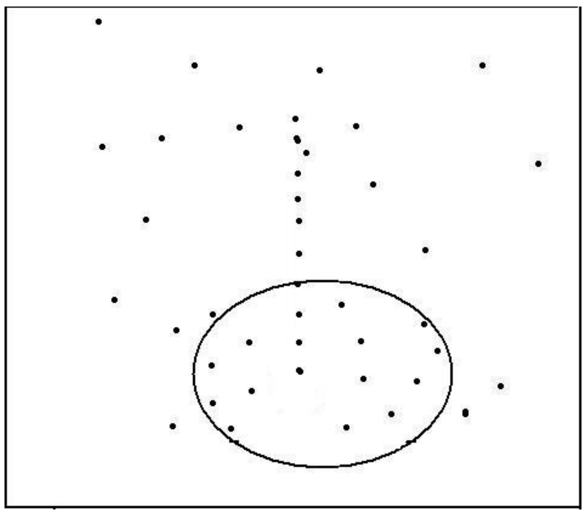

Figure 2.14: Complete-link clustering
Although complete-link clusters are often more hyper-spherical than single-link clusters, complete-link's linkage metric is too harsh. For example, a single outlier in a cluster may make the cluster seem dissimilar to every other cluster, regardless of the other objects within it.

Advanced Hierarchical Clustering

As noted, the main problem with group-average clustering algorithms is the runtime of $O(n^2 \log(n))$. This is simply not fast enough to handle the massive databases in use today (even $O(n)$ is too costly in some situations, which poses some interesting problems). CURE (Clustering Using REpresentatives, Guha et al. [47]) addresses the complexity problem of group-average clustering. In CURE, a random sample of objects from D is divided into subsets, with each subset being clustered in isolation. After this, all the subset clusterings are merged together and clustered. Objects not drawn in the random sample may be easily merged with their closest cluster in the final solution. Rather than use every point in a cluster for calculating the linkage-type metric (Equations 2.29-2.31) CURE selects c well-scattered points in a cluster to represent that cluster for linkage calculations. These representatives are drawn to the centroid of the cluster by a factor of α, a user-specified parameter (Figure 2.15). Note that this implies numerical attributes must be used, as categorical ones cannot be drawn towards a centroid. As clustering is done over a sample of objects only, the run-time of CURE is considerably faster than standard linkage clustering. In addition to this, CURE can handle outliers (as an outlier is further from the centroid, it will be drawn in more than regular points and therefore not alter similarity computations between clusters as much) and arbitrary shapes (the well-scattered property ensures that all parts of a cluster are represented).

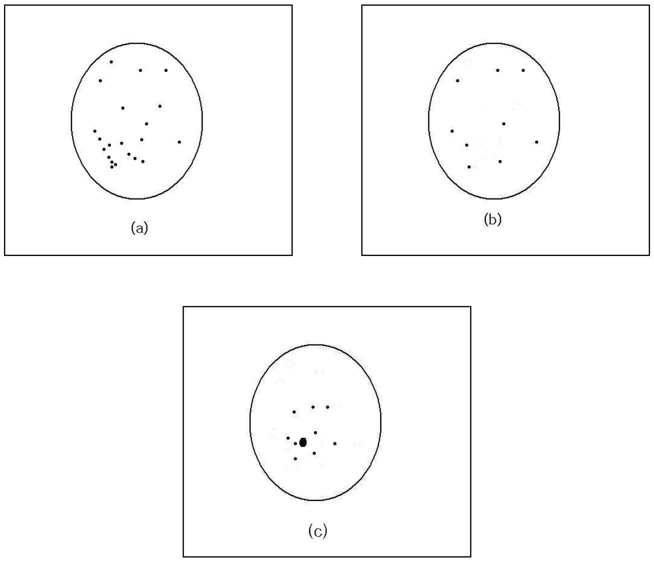

Figure 2.15: CURE: Clustering Using REpresentatives
(b) shows a representative set of well-scattered points within cluster (a). (c) shows the representatives being drawn towards centroid ($\alpha = 2$). (c) is thus the final representation that will be used in linkage calculations with other clusters.

ROCK (Guha et al. [48]), developed by the same authors as CURE, works with links rather than similarities (as discussed in the similarity subsection 2.1.5).

A simplified Algorithm for ROCK is given in Algorithm 5. As for CURE, only a sampling of the data is used during clustering and a parameter θ is a threshold for defining linked objects. Merging is done by finding the two clusters with the largest ratio of actual links between them, divided by the expected number of links by random chance. Although ROCK runs in $O(n \log(n))$ of the sample size, it handles outliers well (in the same way density-based algorithms do). Also, links give a better notion of global connectivity then simple linkage metrics. However, selecting a good θ is difficult.

Algorithm 5 ROCK

```
Input: $D$: the sample of the dataset
       $k$: desired number of clusters
       $\theta$: threshold for similarity
       $x$: similarity metric to use
for all $x$,$y \in D$, $x \neq y$ do
    if $sim_x \geq \theta$ then
        $link(x, y)$
    end if
end for
$C$ = cluster set, with each element of $D$ in its own cluster
while $|C| > k$ do
    $C_i, C_j = \arg \max_{x \in C, y \in C, x \neq y} \frac{linkage(x,y)}{E[linkage(x,y)]}$
    $C_l = C_i \cup C_j$
    $C = C - C_i - C_j$
    $C = C \cup C_l$
end while
return $C$
```

Frequent item-sets have been used for improved hierarchical clustering. In terms of the vector space model, a frequent item-set over a collection of objects is a set of dimensions that have 'high' values in some portion of the set (a significant portion). Discovering frequent item-sets is a well-researched rule-association (data-mining) problem (Hipp et al. [56]) that is useful for high-dimensionality datasets (and therefore text). The idea here is to cluster using the small dimensionality frequent item-sets rather then the large original dataset. Thus, using a frequent item-set is also a type of feature selection. *HFTC* (Hierarchical Frequent Term-Based Clustering, Fung et al. [43]) uses frequent item-sets. It assumes there is some function that calculates frequent item sets over a set of documents, and proceeds as in Algorithm 6. The bulk of the work involves calculating the frequent item-sets, which requires several parameters. Basically, the algorithm is divisive, frequent item-sets are clusters (as they are over a set of documents). Each iteration, the best group of frequent item-sets is used to break down the data into clusters. HFTC is called recursively on each cluster. HFTC produces clustering of a somewhat lower quality then bisecting k-means, and is sensitive to parameters used to create frequent item-sets as well as having a large run-time complexity. Another algorithm along the lines of HFTC, called FIHC (Frequent Item-Sets Hierarchical Clustering, Beil et al. [20]) yields better clustering using frequent item sets and is also much faster then HFTC.

Algorithm 6 Hierarchical Frequent Term Clustering

```
1: Input: $D$: set of data points
2: $C_{all} = FIS(D)$
3: $C_{suff} = \{\}$
4: while $\bigcup_{x \in C_{suff}} \neq D$ do
5:     $y =$ element of $C_{all}$ with least overlap with $C_{suff}$
6:     $C_{all} = C_{all} - y$
       $C_{suff} = C_{suff} \cup y$
7:     remove documents in $y$ from each cluster in $C_{all}$
8: end while
9: order $C_{suff}$ by size
10: for all $x \in C_{suff}$ do
11:     $children(x) = HFTC(x)$
12: end for
```

Mathematical Methods

A minority of hierarchical clustering algorithms are strongly mathematical. *PDDP* (*Principal Directions Divisive Partition*, Boley [24]) is one such mathematical method. It generates a binary tree of clusters by recursively splitting the dataset in two (as with Bisecting K-Means). The hyperplane to split the dataset on is selected using Singular Value Decomposition (see Section 2.1.4). Simply put, SVD is performed on the dataset and a hyperplane orthogonal to the largest eigenvector obtained is used to split the dataset in two. This is performed recursively. PDDP has a large run-time.

Conceptual (also called *model-based*) hierarchical clustering algorithms are similar to those discussed in probability distribution subsection of Section 2.2.3 (e.g., clusters are described as a model/distribution rather than the points assigned to them). Cobweb (Fisher [39]) is an incremental conceptual clustering algorithm for categorical data that builds a classification tree. Each node of the tree is a cluster (the tree is initialized with a single node), and at the same time it is a classifier. The methodology is to construct a tree of clusters/classifiers such that the predictive abilities of each node is high [1]. Points are 'dropped' into the tree at the root one at a time, and 'percolate down', causing possible splits, merges, and new leaves to form. The split/merge/creation process, and the path the object takes down the tree are all driven by a heuristic to maximize the predictive ability of the tree. Cobweb runs in $O(tn)$ time (t is the time through the tree) and is order-sensitive (different orders of data submission result in potentially very different trees). Furthermore, there is no assurance of a balanced tree as output. CLASSIT (Gennari et al. [45]) is similar to Cobweb, but uses normal distributions and numeric features. Labyrinth is an extension of Cobweb (Thompson & Langley [92]).

A different kind of conceptual clustering, called SUBDUE (Jonyer et al. [61]) is designed for structural data (such as the form of an XML document). For example, a cluster may be defined as the concept $\{A, B, C\}$, where A-B, B-C, and C-A are connected. All sets of points $\{x, y, z\}$ where x-y, y-z, and z-x are connected, match this pattern and might belong in the cluster corresponding to that structure. The hierarchical aspect of SUBDUE may be observed as follows: Within every structure of size greater then one there is a substructure of smaller size [2]. SUBDUE uses a depth search to find substructures and is based on MDL. Its main novelty is the data type it operates on: structural data. Clustering algorithms to handle structural data are rare.

2.2.6 Other Clustering Methodologies and Principles

As we mentioned at the outset, there is no universally agreed upon way of splitting clustering algorithms into categories. As a consequence, some algorithms don't fit in any of the categories we have discussed. We mention some other clustering techniques briefly.

Real clustering applications are often forced to deal with constraints. Tung et al. [93] explain the framework for constraint-based clustering. Constraints can come

[1]By predictive abilities, we mean nodes are able to accurately select a cluster for a data object.

[2]Continuing the previous example, $\{A, B\}$, where A-B is linked is a substructure, and therefore that concept is a child concept of the previous one.

in many forms: number of clusters, minimum/maximum cluster size, maximum deviance from a centroid for any point, points that must not be in same cluster, points that must be in the same cluster, and so forth. Constraints require some rethinking of previous approaches.

An example of a constraint-based modification is *frequency sensitive* K-Means (Banerjee & Ghosh [17]), which uses an altered objective function to help ensure the minimum number of points in a cluster. There are many algorithms to deal with practically every kind of constraint.

Support Vector Machines (SVMs) are useful tools that are associated with clustering. SVMs use statistical learning theory to generate functions that can classify (or perform general regression on) data. For an introduction to SVMs see Cristianini & Shawe-Taylor [30].

Additional clustering approaches include genetic algorithms (Goldberg [46]), artificial neural networks (ANN), others.

Chapter 3

Significant Feature Clustering

In this chapter we present our new clustering algorithm, called ”SFC” (Significant Feature Clustering). We begin by explaining the problem domain in which SFC is intended to be applied, that being hierarchical text document clustering, and follow this with a discussion of SFC’s logical basis. Finally, the SFC algorithm is presented, and we discuss its ability to handle various important issues, including text-specific and general clustering problems.

3.1 Problem Domain

Our problem domain is text clustering, specifically clustering at the document level, which includes web pages, newspaper articles, scientific papers, and any other logically encapsulated block of text. The typical purpose of document-level clustering, which is our purpose as well, is the creation of a set of clusters in which each cluster corresponds to a topic. The hope is that these *topic clusters* correspond to the way a human would organize a document collection. For example, an on-line library of multi-disciplinary scholarly articles would require topic clusters of fields of study (i.e., Computer Science, Mathematics, Geology, Psychology, etc.). We add one additional requirement to the goal of creating topic clusters, that of producing a hierarchical clustering such as that found on the DMOZ web site [4]. DMOZ’s hierarchy is a human-crafted organization of a massive collection of web pages in which leaves of the hierarchy are single web pages. These documents are grouped under a specific, narrowly defined topic, which is in turn grouped with other narrow topics to form a more general topic, and so on until everything is under a completely general root (see Figure 3.1), We aim for a DMOZ-like hierarchy for two reasons:

Hierarchical clustering methods naturally produce a structure very similar to DMOZ's.

Our goal is to create a topic hierarchy for browsing, and DMOZ is a human-made topic hierarchy intended for browsing that people actually use.

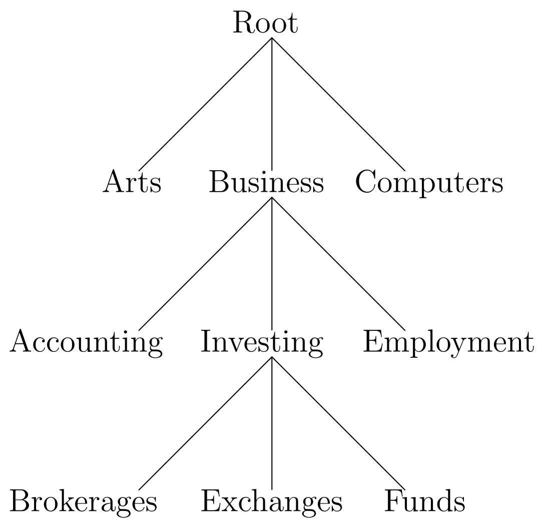

Figure 3.1: DMOZ
A small section of the hand-made hierarchy of DMOZ topics.

Generating a hierarchical set of topic clusters poses all the standard hierarchical clustering problems, including time complexity, ability to handle data streams, and so on. In addition to these, the domain of text poses some additional problems including the curse of high dimensionality, generating a data representation from raw text, the need for topics to correspond to existing human classification schemes, and the inherent ambiguity in assigning a topic, when even human judges may not agree.

The curse of high dimensionality has received more attention than any of these other text problems because it affects both the speed and accuracy of clustering, especially since vectors of weighted word counts are the most common data representations for text. These vectors have lengths potentially in the hundreds of thousands, which makes many hierarchical algorithms unusable on large datasets as they simply cannot calculate the similarities between all pairs of documents. Furthermore, these gigantic vectors obscure the small number of dimensions over which documents are likely to exhibit similarity, lowering the overall accuracy of a clustering algorithm. We have already examined a number of approaches to dealing with giant vectors including feature selection, dimensionality reduction, and clustering algorithms like HFTC [43]. In Section 3.2 below, we discuss a different way of handling high dimensionality which involves creating a hierarchy based on clustering with significant dimensions only. As only a fraction of dimensions should be significant to a single document, this will speed up the clustering as well as increasing accuracy, providing we compute significances correctly.

For the other problems we mentioned for text clustering, the second issue we wish to address in this thesis is the most commonly ignored problem in text clustering, that of generating the data representations of documents. How does one go from the various data formats for documents such as pdf or HTML to a data representation such as a vector? This process is not part of a clustering algorithm per se, as most algorithms simply assume data representations are provided from some other algorithm, and do not discuss how they were generated. Our data representation is unique so we discuss it here as well.

3.2 Basis of our Algorithm

3.2.1 Generating a Data Representation

Before explaining the basis of our algorithm, we will first describe our method for generating the weighted vectors to use in our algorithm. Given our document set D, we can assume we are supplied with a vector of word frequency counts $D_i = \langle tf_{i1}, tf_{i2}, ..., tf_{im} \rangle$ for each document of our collection as generating such vectors is simple enough, even when the original format of the document is not plain text[1].

As we observed in Section 2.1.3, raw count vectors are not particularly useful for clustering, so weighting functions are used to transform each document's vector entries D_{ik} into a value w_{ik}, where each w_{ik} represents the importance of term k to document D_i based on k's distribution within D_i and D. Most clustering algorithms borrow weighting functions from the information retrieval community (in which determining word significance is one of the principle research areas). The

[1]Methods such as pdf2text [7], ps2text [9], HTML Stripper [6], and the like convert non-plain text formats to plain text. It is then a simple matter to transform the texts to word frequency count vectors.

weighting scheme presented by Zhao and Karypis [102] in their study of hierarchical document clustering is a variant of the *tf-idf (term frequency-inverse document frequency)* family of weighting functions. Such functions consist of three components: one based on term frequency (count data), one based on inverse document frequency (one over the fraction of documents containing a term), and a normalization component. These components are multiplied together to generate a single weight. Table 3.1 lists some possible values for each component.

Tf-idf metrics work well because, in addition to factoring in term frequency and normalization, they incorporate the notion of *inverse document frequency*. It has been observed and empirically verified that, given a document collection of size $|D|$, as $n_j/|D|$ (document frequency) increases, the discrimination power of j decreases. By discrimination we mean the ability of j to distinguish one document from another. Clustering is an exercise in discrimination: we determine which points belong to which clusters. By extension, in the vector space model, a word with high document frequency is not significant for clustering as it doesn't discriminate well. Such a word should get a low weight because weights measure significance. The opposite is also true. Tf-idf measures factor in inverse document frequency and as such, are good weighting functions for clustering. Notice that tf-idf metrics can be used as simple counts (*txx* from Table 3.1) and boolean presence indicators (*bxx*) in addition to more complex weights such as that discussed by Zhao and Karypis (tfc).

	Term Frequency		Inverse Document Frequency		Normalization
b	1.0 (0.0 if absent)	x	1.0	x	1.0
t	tf_{ij}	f	$\log(\frac{n}{n_j})$	c	$\frac{1}{\sqrt{\sum\limits_{k=1}^{m} w_{ik}^2}}$
n	$0.5 + 0.5\frac{tf_{ij}}{\arg\max\limits_{k=1}^{m} tf_{ik}}$	p	$\log(\frac{n-n_j}{n_j})$		
l	$\log(1 + tf_{ij})$				

Table 3.1: Tf-idf weighting schemes
To calculate w_{ij} one function is picked from each column. These are combined together to generate a single weight by multiplying them together. $n = |D|$ (size of document collection, $n_j = |\{D_x \in D : tf_{xj} > 0\}|$ (number of documents containing 1 or more instances of term j).

Besides tf-idf metrics, there are also distribution-based methods for calculating w_{ij}s. The Two Poisson Model [81] is an example of such a weighting scheme. We do not discuss functions like this further, as they are not used in clustering.

The method of weighting we elect to use is the tf-idf variant lfc or:

$$w_{ij} = \frac{(\log(1 + tf_{ij}))\log(\frac{n}{n_j})}{\sqrt{\sum_{k=1}^{m} w_{ik}^2}} \tag{3.1}$$

This function is similar to tfc, except the term component has a log so it grows slowly. The 1+ within the log is there so that when $tf_{ij} = 0$, $\log(1 + tf_{ij}) = \log(1+0) = \log(1) = 0$. In other words, when $tf_{ij} = 0$, $w_{ij} = 0$. The normalization component is essential for dealing with documents that vary significantly in length.

Algorithm 7 is our method for converting raw frequency count vectors to lfc weighted *tf-idf* vectors.

Algorithm 7 GenerateRepresentations

```
1: Input: $D$: dataset of count vectors
2: $w =$   array of of vectors
3: //Assign Basic Weights
4: for $i = 1$ to $n$ do
5:     for $j = 1$ to $m$ do
6:         $w_{ij} = \log(1 + tf_{ij})\log(\frac{n}{n_j})$
7:     end for
8: end for
9: //Normalize Each Document's Weights to Length 1
10: for $i = 1$ to $n$ do
11:     $t = 0.0$
12:     for $j = 1$ to $m$ do
13:         $t = t + w_{ij}^2$
14:     end for
15:     $t = \sqrt{t}$
16:     for $j = 1$ to $m$ do
17:         $w_{ij} = \frac{w_{ij}}{t}$
18:     end for
19: end for
20: return $w$
```

3.2.2 Similarity and Significance

Our goal in this section is to explain our definition for *significance* of a dimension to a document. We show how to perform a mapping of real-numbered data vectors to *significance vectors*, which are vectors with each dimension assigned a value of -1, 0, or 1, representing the significance of that dimension to the document.

We begin by defining our notion of significance using weighted vectors produced by Algorithm 7. This is the standard data representation for text clustering, where increasing values of a dimension indicate increasing probability of importance. Consider a single dimension of w, numbered k, where μ_k is the mean value of dimension k in w:

$$\mu_k = \frac{\sum_{i=1}^{n} w_{ik}}{n} \tag{3.2}$$

If we place all the μ_k values for all dimensions together into a single vector, we have a pseudo-document which we will refer to as μ, which may be thought of as the average weighted document of D.

Now pick two documents from D, call them D_i and D_j, with values w_{ik} and w_{jk} for dimension k respectively. $w_{ik} > w_{jk}$ indicates that dimension k is probably more significant to D_i then D_j. Now consider what it means when $w_{ik} > \mu_k$. The mean is always the centre of gravity for a dataset, that is to say, the point such that the average absolute distance of other points from it is a minimum is always the mean.

We argue that, despite the way tf-idf weighting schemes work, for any dimension k, when $w_{ik} = \mu_k$, k's probability of significance to D_i is a minimum, and not when $w_{ik} = 0$ or when $w_{ik} = \arg\min_{w_m \in w} w_{mk}$. Consider what clustering does: it puts similar objects in the same cluster, and dissimilar ones in different clusters. If $w_{ik} = \mu_k$, then other w_{jk}'s have a minimum average distance from w_{ik}, meaning that k does not distinguish D_i well from other documents. We further note that, in general, as w_{ik} deviates further from the mean in either direction, for many distributions (uniform, normal, etc.), the average distance from w_{ik} is monotonically increasing.

The fact that deviation from the mean increases average distance from w_{ik} means that k is more significant to D_i, as it better differentiates D_i from other documents. However, the type of significance is not the same when below the mean as above though. Consider a text example where there is a mean weight of 50 for the word *apple* in D and we have two documents, one with zero weight and the other 100. Clearly, the way in which *apple* is significant to the document with zero differs from the significance *apple* has to the document with 100.

We refer to a dimension k as *positive* with respect to D_i when $w_{ik} > (\mu_k + \varepsilon)$, *negative* to D_i when $w_{ik} < (\mu_k - \varepsilon)$, and *neutral* where $w_{ik} = \mu_k$. ε is real number that quantifies how far away from the mean a dimension must be for us to conclude positive or negative significance. These three definitions can be used to map a real-numbered dimension to a significance of 1 (positive), 0 (neutral), or -1 (negative). We will refer to this simple mapping function as f_s:

$$f_s(w_{ik}) = \begin{cases} 1, \text{ if } \ w_{ik} > (\mu_k - \varepsilon). \\ -1, \text{ if } w_{ik} < (\mu_k - \varepsilon). \\ 0, \text{ otherwise} \end{cases} \tag{3.3}$$

If we apply f_s to the entire vector w_i, we will have mapped our tf-idf vector to what we refer to as a *significance vector*. We denote the significance vector produced by applying f_s dimension-wise to w_i as S_i:

$$S_i = \langle f_s(w_{i1}), f_s(w_{i2}), \cdots, f_s(w_{im}) \rangle \tag{3.4}$$

The concept behind a significance vector is to have a clear indication about which words are important to which document in what way, relative to the entire document collection. Such an idea is appealing for obvious reasons, but how do we pick a good ε. We might consider using $\varepsilon = 0$. But if we do this it is more than likely that there is no D_i and k such that $w_{ik} = \mu_k$, so every dimension of S_i will be either a 1 or -1. This is incorrect, as most dimensions of S_i should be zero, e.g. only a small fraction of the dimensions are significant to any one document. Instead, we select ε based on word distribution information on a dimension-by-dimension basis.

We define a variable α, $0 \leq \alpha < 2$, where α_k represents an interval length of the distribution of weights for dimension k in D, computed as:

$$\alpha_k = \alpha \frac{(\arg \max_{i=1..n} w_{ik} - \arg \min_{i=1..n} w_{ik})}{2} \tag{3.5}$$

α_k is our uncertainty threshold for dimension k, centred around the mean μ_k. When $\mu_k - \alpha_k \leq w_{ik} \leq \mu_k + \alpha_k$, we desire a value of 0 for S_{ik}, and if we make corresponding adjustments to the definition of -1 or 1 from f_s, we have a new f_s given in Equation 3.6. Figure 3.2 illustrates Equation 3.6.

$$f_s(w_{ik}) = \begin{cases} 1, \text{ if } \; w_{ik} > \mu_k + \alpha_k. \\ -1, \text{ if } w_{ik} < \mu_k - \alpha_k. \\ 0, \text{ otherwise} \end{cases} \tag{3.6}$$

As α increases, each α_k increases, and correspondingly, the number of 1's and -1's for every dimension decreases. There is tradeoff here as we adjust α, with larger values we are more certain that 1s and -1s assigned by f_s correctly reflect a dimension being significant, but we are less certain that our zeroes are correctly assigned. This is due to our requirement that the tf-idf values be further away from the mean to be 1 or -1 with higher α's, and so less weights satisfy the criterion.

Using our new f_s, we refer to dimension k, where $f_s(w_{ik}) = 1$ as *significantly positively* to D_i and *significantly negative* when $f_s(w_{ik}) = -1$. To illustrate our new definition of f_s, Table 3.2 gives a sample of four documents from an artificial dataset, with $\alpha = 0.5$ and means and α_k's calculated using the appropriate functions. Table 3.3 shows the significance vectors for each of the four documents, computed using Equation 3.4 with our second definition f_s from Equation 3.6. We will discuss our technique for selecting an α shortly.

Vector/Dimension	1	2	3	4	5	6	7	8	9
D_1	5	-2	4	16	-10	5	11	2	3
D_2	12	11	-3	15	-9	25	-9	2	-6
D_3	-2	5	-3	15	2	-10	-8	6	-11
D_4	6	-5	8	8	-6	0	9	-1	14
μ	5.25	1.75	1.5	12.5	-4.75	4	0.75	2.25	0
α_k's	3.5	4	2.75	2	3	8.75	5	1.75	6.25

Table 3.2: Significance vectors 1
Four vectors of an artificial dataset along with μ and the α_k values when $\alpha = 0.5$

Equation 3.6 is the final f_s we will use to generate our S_is. Although the function can deal with variable requirements of certainty by adjusting α, it still overlooks some other issues. With our current definition, every dimension will have a relatively equal number of documents with 1 and -1, but it is known that not all dimensions have equal distinguishing power, as we discussed in Section 3.2.1. Thus, for example, the word *the*, which likely does not help distinguish documents, will have the same distinguishing power in our model as an obscure word such as *aardvark*, which in all likelihood is a powerful distinguisher between documents. It is therefore desirable to adjust the number of 1's and -1's allowed per dimension,

Vector/Dimension	1	2	3	4	5	6	7	8	9
S_1	0	0	0	1	-1	0	1	0	0
S_2	1	1	-1	1	-1	1	-1	0	0
S_3	-1	0	-1	1	1	-1	-1	1	-1
S_4	0	-1	1	-1	0	0	1	-1	1

Table 3.3: Significance vectors 2
The significance vectors for the documents from Table 3.2

based on its distribution information. We do not deal with this aspect here though. Rather we make the assumption that insignificant words have been removed, via a method such as a stop list if it is text, or perhaps feature selection/extraction, and that all other words have roughly the same distinguishing power. This is obviously not true, but as we will see in the evaluation section, it suffices to produce a good clustering result.

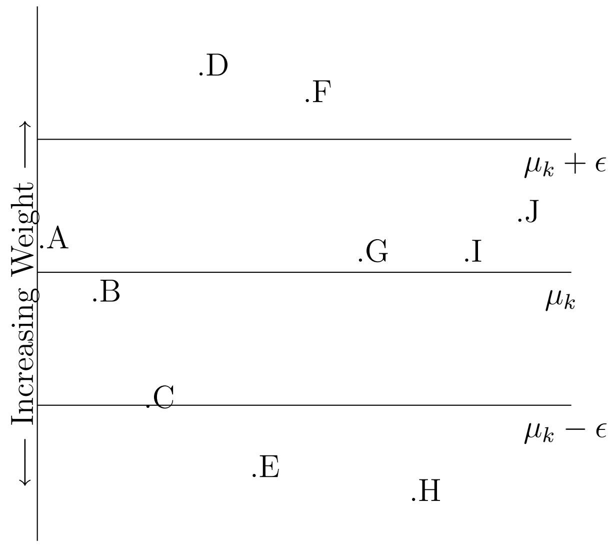

Figure 3.2: The modified f_s function
A, B, G, I, and J are all neutral with respect to dimension k. D and F are positive, while C, E, and H are negative.

The algorithm to compute significance vectors from D is given in Algorithm 8. The minimum and maximum value for each dimension are tracked so we can properly select the size of each α_k, and the overall run time is $O(nm)$.

Algorithm 8 SignificanceVectors

```
1: Input: $w$: dataset of weighted vectors, each length 1
2:        $\alpha$: threshold value
3: $\mu$ = new vector of 0's, length $m$
4: $S$ = new array of significance vectors
5: for $j = 1$ to $m$ do
6:     //Calculate $\mu_j$, $m_j$, and $M_j$
7:     for $i = 1$ to $n$ do
8:         $\mu_j = \mu_j + w_{ij}$
9:         $M_j = \max(M_j, w_{ij})$
10:        $m_k = \min(m_j, w_{ij})$
11:    end for
12:    $\mu_j = \frac{\mu_j}{n}$
13:    $\alpha_k = (M_j - m_j))/2\alpha$
14:    //Assign significance for dimension $j$
15:    for $i = 1$ to $n$ do
16:        $S_{ij} = f_s(w_{ij})$
17:    end for
18: end for
19: return S
```

The S_i's produced in from algorithm 8 are intended to be used in place of the normal tf-idf vectors for our algorithm during clustering. It may seem like this choice for data representations would result in an inferior quality clustering, as we have lost information by mapping from real number to significances, but in fact as we will show in the testing section, accuracy remains high.

3.2.3 Similarity between Significance Vectors

With our significance vectors created, we now present how to evaluate their similarity. Given D_i, S_i and D_j, S_j, the function we select to evaluate similarity is the cosine:

$$sim_{cos}(D_i, D_j) = \frac{S_i \cdot S_j}{\|S_i\|\|S_j\|}$$

It is easy to see that where positive and negative dimensions of S_i and S_j match up, similarity is increased, and where they mismatch similarity drops. This means that S_i and S_j have a high similarity when they overlap in types of dimension significance, which is exactly what we want.

The usage of cosines for similarity is a simple choice. Significance vectors are not length-normalized, making the Euclidian distance metric (Equation 2.2) a poor choice for similarity. This leaves us with the most prominent choice being cosine. It is possible to use some metric applicable to categorical data such as the Jaccard coefficient from Table 2.1, as we have mapped real numbers to discrete three-valued entries which may be viewed as three categories, but given our definition of similarity between clusters that follows, it should be clear why we did not select such a similarity measure. Cosine adequately captures the amount of overlap between our significance vectors relative to the maximum they could have.

3.2.4 Similarity between Clusters

Hierarchical clustering algorithms require some definition of similarity between clusters in order to facilitate merging or splitting. For instance, recall that single-link clustering defines the similarity between two clusters as the similarity of their closest two points, while complete-link uses the furthest pair of points. Rather than search for the furthest or closest pair of points, or calculating the average distance as in UPGMA, we elect to represent every cluster as a single vector, reducing a similarity comparison between clusters to a single cosine operation requiring $O(m)$ time.

Given some cluster C_i of documents for which we have significance vectors $\langle S_1, S_2, \cdots, S_{|C_i|} \rangle$, we refer to its single-vector representation as a *probabilistic significance vector*, which we denote as P_i. The value at P_{ik} corresponds to the expected significance of k for a document drawn at random from cluster C_i. Calculating P_{ik} is done by a weighting process [2]:

$$P_{ik} = \frac{1}{|C_i|} \sum_{i=1}^{|C_i|} S_{ik} \tag{3.7}$$

As P_i quantifies how well the expected significances of each dimension corresponds to a document drawn at random from C_i, we consider it to be a good

[2]It is clear from this definition that P_i is just the mean/centroid of cluster C_i from this definition.

representation for cluster C_i. An illustrative example of calculating P_{ik}'s is given in Table 3.4, which is based on Tables 3.2 and 3.3.

Cluster/$P_{ik}s$	1	2	3	4	5	6	7	8	9
{1,2}	0.5	0.5	-0.5	1	-1	0.5	0	0	0
{1,3}	-0.5	0	-0.5	1	0	-0.5	0	0.5	-0.5
{1,4}	0	-0.5	0.5	0	-0.5	0	1	-0.5	0.5
{2,3}	0	0.5	-1	1	0	0	-1	0.5	-0.5
{2,4}	0.5	0	0	0	-0.5	0.5	0	-0.5	0.5
{3,4}	-0.5	-0.5	0	0	0.5	-0.5	0	0	0
{1,2,3}	0	0.33	-0.66	1	-0.33	0	-0.33	0.33	-0.33
{1,2,4}	0.33	0	0	0.33	-0.66	0.33	0.33	-0.33	0.33
{2,3,4}	0	0	-0.33	0.33	0	0	-0.33	0	0
{1,2,3,4}	0	0	-0.25	0.75	-0.25	0	0	0	0

Table 3.4: Probabilistic significance vectors
P_{ik}'s for every cluster of size 2 or greater from table 3.3.

With P_i defined, the similarity between two clusters C_i and C_j can be calculated as the cosine between their probabilistic significance vectors [3]:

$$sim_{cos}(C_i, C_j) = \frac{P_i \cdot P_j}{\|P_i\| \|P_j\|}$$

If we are agglomeratively merging clusters, then computing a newly created cluster's PS can be done quickly using Algorithm 9. The evaluation of similarity between clusters is only a single cosine operation. These two factors mean an agglomerative implementation of SFC will iterate quickly.

Algorithm 9 SFCMerge

```
1: Input: $C_i, C_j$: clusters to be merged
2: $C_l = C_i \cup C_j$
3: **for** $k = 1$ to $m$ **do**
4:     $P_{lk} = \frac{|C_i|P_{lk} + |C_j|P_{jk}}{|C_l|}$
5: **end for**
6: **return** $C_l$
```

[3]This is the cosine between means/centroids of the two clusters.

It is important to note that using averages (otherwise known as centroids) to represent clusters and clustering based on cosines between averages is not a well-used clustering technique. It may seem similar to K-Means, but it should be noted that basic K-Means uses cosines between single points and averages, not between averages and averages. The reason average-average cosines are not used is simple: poor clustering results when using tf-idf vectors. We argue that our vectors, with a properly defined α, are amenable to using average-average cosine clustering.

3.2.5 Selecting an α

Selecting a value of α to use is critical in achieving optimal clustering results in our algorithm. Before describing a technique for finding such an α, we should note that there is no reason to believe that a single α value will yield optimal clustering over different datasets. This is because the weighting functions used (in our case, the lfc tf-idf function) can create a variety of weight distributions based on underlying frequency counts, and such changes can greatly alter the effect of a fixed α. Thus we must look at finding the optimal α for each dataset based on its own property.

As we have said in Section 2.1.8, optimal clustering is an ambiguous term and we must link it to a specific definition. To that end, we use "optimal" in the following sense: when running our final algorithm until we are left with one cluster, e.g., a tree with a single root, using a variety of different α's, an optimal α will be one where the F-measure calculation used in Zhao and Karypis' hierarchical clustering review [102] is maximized:

$$maximize \quad \sum_{L_j \in L} \frac{n_j}{|D|} \max_{C_i \in T} F(C_i, L_j) \tag{3.8}$$

Note that this maximization is defined over a tree of clusters such as the one in Figure 2.12, and not a set such as that the evaluation metric used in Hierarchical Frequent Term Clustering [20].

Obviously, we cannot directly use Equation 3.8 to find such an optimal α, as in real applications we have no labels. Instead we may use some function that approximates the equation. It suffices to apply a metric that factors in precision and recall as the F-measures of Equation 3.8 are based on these. To this end, we opt to use a metric that rewards clusters for having a high internal similarity, and further rewards larger clusters more for bearing the same average internal similarity as a group of smaller clusters that add up to the same size. The first aspect, rewarding larger internal similarity, may be thought of as *precision.* Objects of similar labels should be highly similar, therefore a cluster with high internal similarity should exhibit higher precision. The second aspect of rewarding larger clusters more than a group of smaller ones for exhibiting the same average internal similarity may be thought of as *recall*. A larger cluster exhibits higher recall on average.

Before we give the metric we will use in approximating Equation 3.8, we note that the approximation is based on the tf-idf vectors produced from Algorithm 7, not the significance vectors from Algorithm 8. The reason for this is simple: with each different α value, the significance vectors change. It is not possible to accurately compare a quality for, say, $\alpha = 0.1$ and $\alpha = 0.2$ on the basis of their significance vectors, since they are using different ones; their source tf-idf vectors are the same though. Thus when we say $sim_{cos}(x, y)$ in this section, we mean $sim_{cos}(w_x, w_y)$, where w is the set of tf-idf vectors.

The particular metric we select to try to fulfill our requirements is a variation of *internal similarity* given in Equation 3.9 (Zhao and Karypis [102]). Internal similarity metrics quantify the similarity that data objects have to data objects in the same cluster. In the case of our metric, $\phi_{isim}(C)$ is the sum over all clusters of each cluster's average internal cosine divided by its size:

$$\phi_{isim}(C) = \sum_{C_i \in C} \frac{\sum\limits_{x,y \in C_i, y \neq x} sim_{cos}(x, y)}{|C_i|} \tag{3.9}$$

As the values for the average within cluster cosine increases, the top part of the fraction increases, fulfilling our first requirement. The second requirement is not fulfilled by this equation though. For example, let us say we have two clusters, each of size y, and each exhibiting average internal similarity x. They each contribute y^2x/x to equation 3.9, for a combined total of $2y^2$. Now consider one cluster of size $2y$, with an average internal similarity of x. This contributes $(2y)^2x/2y$, or $2y^2$, the same as the two smaller clusters combined, so our second requirement is not fulfilled.

As it turns out, factoring recall and precision into a metric in a balanced way is difficult. The solution we developed is as follows. Recall that we have our cosine similarity defined between significance vectors. As we have n such vectors, we have n^2 cosines, one for each pair of vectors. Given some cluster C_i, we want to know how internally similar points are within C_i (equation 3.9) compared to the best internal similarity possible for a cluster of size $|C_i|$. Computing the most internally similar cluster of size $|C_i|$ from D is NP-Hard. To this end, we settle for an upper bound on this value, which we denote $max_{cos}(D, x)$. $max_{cos}(D, x)$ be the maximum average sum of x cosines between pairs of vectors in D. Computing $max_{cos}(D, x)$ for $x = 1$ to $x = n$, assuming cosines have been computed, requires $O(n^2 \log(n))$: $O(n^2 \log(n))$ to sort the n^2 cosines in descending order and $O(n^2)$ to iterate through the list adding the previous *max* to the next. As x increases, $max_{cos}(D, x)$ is monotonically decreasing. If we use $max_{cos}(D, x)$ to scale our Equation 3.9 by division as in Equation 3.10, clusters with lower number of edges, e.g. smaller ones, will be divided by larger values, and those with more edges will be divided by a smaller number. In essence, we reward larger clusters, hopefully giving a flavour of recall to the internal similarity metric without unbalancing the precision aspect.

$$\phi_{isim}(C) = \sum_{C_i \in C} \frac{\frac{\sum_{x,y \in C_i, y \neq x} sim_{cos}(x,y)}{|C_i|}}{max_{cos}(D, \sum_{C_i \in C} \frac{|C_i|(|C_i|-1)}{2})} \tag{3.10}$$

Equation 3.10 is defined over a set of clusters, not a hierarchy. As the F-measure we are trying to maximize with internal similarity relies on searching through the entire tree for the best cluster for each label, we should alter Equation 3.10 in a similar manner so that it applies to a hierarchy. Specifically, we wish to search through the hierarchy and select only the best group of clusters, such that their union equals D. The manner in which we do this can be summarized by a recursive function $\beta(C_i)$:

$$\beta(C_i) = \begin{cases} \phi_{isim}(children(C_i)), \text{ if } \phi_{isim}(\{C_i\}) < \phi_{isim}(children(C_i)). \\ \phi_{isim}(\{C_i\}), \text{ otherwise} \end{cases} \tag{3.11}$$

With this setup, we define the *maximum tree internal similarity*, or *mtis*, for a tree of clusters T:

$$\phi_{mtis}(T) = \beta(root(T)) \tag{3.12}$$

Equation 3.12 is our approximation for F-measure. With it, a procedure for selecting the optimal α can be easily implemented:

1 Select some range and granularity of α to sample. For example, $0 \leq alpha \leq 1$ with a granularity of 0.1 means we sample $\alpha = \{0.1, 0.2, \cdots, 1.0\}$.

2 For each α value, run the algorithm to generate a tree T.

3 Select the α such that equation 3.12 is maximized.

Algorithm 10 lists the necessary adjustments to Algorithm 9 to extract the $\beta(C_i)$'s.

3.3 Significant Feature Clustering

With our foundation defined, we now present our full algorithm, called SFC (Significant Feature Clustering). After presenting the full algorithm, we explain how it deals with various clustering issues.

3.3.1 The SFC Algorithm

We give an agglomerative implementation of SFC in Algorithm 12. Algorithm 11, called by SFC, computes the tf-idf cosines needed for computing the β's.

Algorithm 10 SFCMerge2

```
1: Input: $C_i, C_j$: clusters to be merged
2: $C_l = C_i \cup C_j$
3: for $k = 1$ to $m$ do
4:     $P_{lk} = \frac{|C_i|P_{lk}+|C_j|P_{jk}}{|C_l|}$
5: end for
6: $\beta(C_l) = \begin{cases} \phi_{isim}(\{C_i\}) + \phi_{isim}(\{C_j\}), \text{ if } \phi_{isim}(\{C_l\}) < \phi_{isim}(\{C_i\}) + \phi_{isim}(\{C_j\}). \\ \phi_{isim}(\{C_l\}), \text{ otherwise} \end{cases}$
7: return $C_l$
```

Algorithm 11 Cosines

```
1: Input: $w$: set of tf-idf vectors
2: cos = two dimensional array of cosines
3: for $i = 1$ to $n$ do
4:     for $j = 1$ to $n$ do
5:         $cos_{ij} = sim_{cos}(w_i, w_j)$
6:     end for
7: end for
8: return cos
```

3.3.2 Time Complexity

It should first be noted that most of the algorithms discussed in Chapter 2 abstract away from the notion that each data object is a vector. Clustering typically requires computing similarity between each pair of points for a run-time cost of $O(n^2m)$. We do not use the clustering convention of ignoring the time to create data representations. We now compute the overall run-time cost by breaking down the cost of the algorithm line by line and then grouping these until we have the complexity of the entire algorithm in our scope.

Lines 1-2 Parameters, no run-time cost.

Lines 3-4 Constant time operations to initialize the best tree to null and the best maximum tree internal similarity to zero.

Line 5 Calls Algorithm 7. This Algorithm contains two double for-loops, each one iterating over n then m for a run-time of $O(nm)$.

Line 6 Calls Algorithm 11. With n vectors over which cosines must be computed (each of length m) the cost is $O(n^2m)$.

Lines 7-10 Sorting the values from line 6 takes $O(n^2 \log(n))$ time. Iterating through *maxList* on lines 8-10 takes $O(n^2)$ time. Combining these, the complexity is $O(n^2 \log(n))$.

Algorithm 12 SFC

```
 1: Input: $D$: raw frequency count vectors
 2:        $F$: set of alpha values to sample
 3: $T_{best} = \{\}$
 4: $MaxMtis = 0$
 5: $w$ = GenerateRepresentation(D)
 6: $cos$ = Cosines(w)
 7: $maxList$ = descending sorted single dimensional array of values from cos
 8: for $i = 2$ to $\frac{n(n-1)}{2}$ do
 9:   $maxList_i = maxList_i + maxList_{i-1}$
10: end for
11: for $f \in F$ do
12:   S = SignificanceVectors(w,f)
13:   C = empty set of clusters
14:   for $i = 1$ to $n$ do
15:     $C_i$ = new cluster with $P_i = S_i$
16:     $\beta(C_i) = 0$
17:     $C = C \cup C_i$
18:   end for
19:   while $|C| > 1$ do
20:     $C_i, C_j = \arg \max_{x \in C, y \in C, x \neq y} sim_{cos}(P_x, P_y)$
21:     $C_l = SFCMerge2(C_i, C_j)$
22:     $C = (C - C_i) - C_j$
23:     $C = C \cup C_l$
24:   end while
25:   T = tree rooted at $C$
26:   if $\phi_{mtis}(T) > MaxMtis$ then
27:     $MaxMtis = \phi_{mtis}(T)$
28:     $T_{best} = T$
29:   end if
30: end for
31: return $T_{best}$
```

Lines 1-10 The complexity is dominated by the cost of sorting in lines 7-10, which means the overall complexity is $O(n^2 \log(n))$.

Line 12 Calls Algorithm 8, which iterates over m then n taking $O(nm)$ time.

Line 13 Constant time operation.

Lines 14-17 A for-loop to initialize clusters before starting clustering that takes $O(n)$ time.

Line 20 Assuming that cosines are stored in sorted lists for each cluster, we must look at the first entry for each of up to n such lists to find the best similarity, taking $O(n)$ time.

Line 21 Calls Algorithm 10, which calculates a new probabilistic significance vector in $O(n)$ time. Computing a new β value, if done naively, takes up to $O(n^2)$ time, but we will describe below a way around this. These two operations together take $O(n^2)$ time. As well, we must calculate cosines between the new cluster and other clusters which takes $O(nm)$ time, and sort these similarities in $O(n\log(n))$ time.

Lines 22-23 Constant-time operations.

Lines 19-24 The heart of the SFC algorithm. The while-loop assumes cosines have been precomputed and sorted for each cluster, for $O(n^2\log(n))$ complexity. The loop itself iterates $n-1$ times, which may appear to give the while-loop $O(n^3)$ complexity as line 21 requires $O(n^2)$ time. The principal cost in Line 21 is calculating a new β, specifically calculating the internal similarity of the new cluster. However, we have already computed the total internal cosines of its two children as part of calculating each one's β. Thus we need only add in the cosines between the clusters, then divide by the new cluster size times the appropriate value from *maxList*. No edge will have to be added twice, meaning computing new β's takes at most $O(n^2)$ time over the entire while loop, not per iteration. This means Line 21's complexity is actually $\max(O(n\log(n)), O(nm))$. Factoring in the while-loop over n and the required cosine calculations/sorting, lines 19-24 have an overall complexity of $\max(O(n^2\log(n)), O(n^2m))$.

Lines 25-29 All constant time operations.

Lines 11-30 Clearly, the most expensive operation within the for-loop is the while-loop. Adding in the additional complexity of $O(|F|)$, we have a run-time of $max(O(|F|n^2\log(n)), O(|F|n^2m))$.

Line 31 Constant time operation.

Entire Algorithm For the two large sections (lines 1-10 and lines 11-30) the latter is more costly, giving an overall run-time of $max(O(|F|n^2 \log(n)), O(|F|n^2m))$. This run-time may seem larger than other hierarchical methods, but it should be noted that $|F|$ is small (in our evaluation section, it is 100). Furthermore, as α increases, the number of features being used in the clustering is decreased significantly, increasing speed. It is not possible to quantify the expected increase as it varies based on the dataset, but it is significant as will be discussed in the evaluation section. As a side note, if we do not factor in the cost of generating data representations, as most clustering algorithms do not, the run-time becomes $O(|F|n^2 \log(n))$.

3.3.3 Feature Types

SFC is applicable to numeric data only. While other linkage-based hierarchical clustering algorithms may be applied to any kind of data because any similarity metric/data type combination may be used, SFC is based on transforming tf-idf vectors to significance vectors, both of which are numeric in nature. This is not a flaw, as the vast majority of text-clustering algorithms are numeric (even HFTC and FTHC, which do not use numbers during clustering, generate their frequent item-sets from initial numeric frequency count vectors).

3.3.4 Outliers and Noise

The potential for noise and outliers affecting clustering quality of SFC has not been evaluated formally. However, it is reasonable to assume that using an average-average calculation for cosines will-be fairly robust in the presence of outliers and noise. We may reason as follows: Consider a single point of outlier/noise and a cluster. If the cluster and this point exhibit a high similarity, then the outlier/noise must not be an outlier/noise at all, as it is near the average of the cluster and therefore fits nicely in it. There is no such assurance of position relative to the cluster for the other linkage metrics we have discussed.

3.3.5 Real-time or Batch Data

SFC is designed to handle only batch data. In its current form, it is not amenable to handling real-time data, but using significance vectors for data stream algorithms such as incremental K-Means is possible.

3.3.6 Scalability

Unfortunately, as it stands, SFC has poor scalability in terms of number of documents. It is highly desirable to be able to select a single α to use rather then iterate through a list of them as we do now. If this could be done, the run-time of SFC would be the same as other linkage metrics. This would still be too slow for large datasets though, so sample-based methodology would need to be introduced into SFC (as used in ROCK and CURE) for further speedup. In terms of dimensionality scalability, Section 4.5 will show that quality remains high relative to other approaches even when dealing with high dimensionality datasets.

3.3.7 Output and Understandability

Since SFC follows a standard agglomerative hierarchical clustering template, its output is a dendrogram. Dendrograms are particularly useful for browsing and are easy to understand. An interesting property of the output that is specific to SFC alone is that, given some threshold γ, $0 \leq \gamma \leq 1$, every cluster in the tree can be easily represented as a list of words where each word's entry in the probabilistic significance vector of cluster is greater then γ. In essence, we have frequent-word sets for each cluster.

3.3.8 Parameters

SFC requires only one parameter, α, or rather, the set of values of α which we wish to search through for an optimal *mtis* clustering to return. As the number of α values searched increases, we can expect correspondingly better results along with increased time. A user therefore does not need to select an α, rather just two items: the number of runs of SFC he/she is willing to run through ($|F|$) for results and range from α_{min} to α_{max} (which may be avoided by specifying the entire range of 0 to 2). With this setup, sample points are at α_{min}, $\alpha_{min} + \frac{\alpha_{max}-\alpha_{min}}{|F|}$,$\alpha_{min} + \frac{2(\alpha_{max}-\alpha_{min})}{|F|}$,$\cdots$, α_{max}.

3.3.9 Order Invariance

The notion of order invariance is used only in data-stream and sample-based methods. As SFC is neither, order invariance is not applicable to SFC.

Chapter 4

Evaluation

In this section, we present our evaluation of SFC. We begin by defining our quality metric, then the algorithms we will test SFC against, followed by test data and test method. We then discuss our test results, and various other observations about SFC.

4.1 Quality Metric

The metric we selected to use for our evaluation is the F-measure used in Zhao and Karypis' clustering review [102]. We have already explained it, but we do so here again for clarity. Given a dataset D, with each $D_i \in D$ having one label drawn from a fixed set of possible labels $L = \{L_1, L_2, ..., \}$, and a cluster C_i, let n_{ij} denote the number of documents in C_i with the label L_j, and n_j denote the number of documents in D with label j. The precision of C_i with respect to the label L_j is:

$$precision(C_i, L_j) = \frac{n_{ij}}{|C_i|} \tag{4.1}$$

and the recall of C_i with respect to L_j is:

$$recall(C_i, L_j) = \frac{n_{ij}}{n_j} \tag{4.2}$$

The F-measure of cluster C_i with respect to the label L_j, which we denote as $F(C_i, L_j)$, is calculated as:

$$F(C_i, L_j) = \frac{2 * precision(C_i, L_j) recall(C_i, L_j)}{precision(C_i, L_j) + recall(C_i, L_j)} \tag{4.3}$$

The overall quality of a tree of clusters T, which we denote as $\phi(T)$, is:

$$\phi(T) = \sum_{L_j \in L} \frac{n_j}{|C|} \max_{C_i \in T}(F(C_i, L_j)) \tag{4.4}$$

It seems reasonable to use this metric as a definition of quality for two reasons. First, and most important, f-measures are the standard way to measure the quality of text clustering. Secondly, we are returning a clustering that maximizes a metric designed to approximate this exact F-measure. This is why we use this F-measure and not one such as that given in HFTC [20], which is based on a set of clusters and not a tree.

4.2 Test Algorithms

We compare SFC against 3 standard algorithms:

1. SLINK [88]
2. CLINK [33]
3. UPGMA [89]

There are a number of reasons for selecting UPGMA, SLINK, and CLINK as SFC's competitors. SFC is from the linkage family of hierarchical clustering algorithms, as are the other three. UPGMA is one of the best agglomerative clustering algorithm in terms of any quality measure and serves as a reference for SFC to aim to surpass. As SLINK's typical results are very poor quality, it serves as a baseline. If an algorithm is doing worse then SLINK, it is a clear indication that it is not working well. CLINK is more robust then SLINK, but still considerably less so than UPGMA. All three of these algorithms are very well known, and their run-times are close to SFC's. We therefore feel they will be good comparisons to SFC. For testing purposes, we used the implementations of UPGMA, CLINK, and SLINK provided by Cluto [62], a clustering toolkit created by Karypis. This clustering toolkit has been used in several published works.

4.3 Test Data

The datasets we used to test the algorithms are also from the Cluto web site. The test data is in the form of document vectors files (called .mat files) containing word frequency counts. Additionally, files containing the correct labels for each document are supplied (.mat.rclass files). The specifications of the seven test sets we used are given in Table 4.1.

Set	Source	Size	# of Labels	# of Terms
fbis	FBIS (TREC)	2463	17	2000
hitech	San Jose Mercury (TREC)	2301	6	22498*
k1a	WebACE	2340	20	12879
k1b	WebACE	2340	6	12879
re0	Reuters-21578	1504	7	2886
re1	Reuters-21578	1657	25	3758
wap	WebACE	1560	20	8460

Table 4.1: Test data
The seven datasets UPGMA, CLINK, SLINK, and SFC used in our testing. The term count for the hitech dataset is so marked because in previous publications it had an incorrect value.

The hitech dataset is a collection of articles from the San Jose Mercury newspaper, and was used in TREC (Text REtrieval Conference [10]). Likewise, the FBIS dataset (Foreign Broadcast Information Service) was used at TREC. Wap is a collection of web pages from the yahoo web page directory [11], and, together with kla and klb, is from the WebACE project [51]. K1a and k1b are particularly interesting as they contain the same documents, but k1a has a finer-grained set of labels. The re0 and re1 datasets are parts of the Reuters-21578 collection [69]. All of these sets have been used in several text application tests, and provide a good range of document source type, number of terms, and number of different labels.

4.4 Method

For UPGMA, CLINK, and SLINK, we ran their Cluto implementation from the command line [1] [2]. The parameters we supplied give a term weighting model basically identical to ours (Equation 3.1), and indicate that the cosine is to be used to evaluate similarity (also like our method). For each clustering method/dataset pair, we ran Cluto until only one cluster remained, and evaluated the F-measure described in Section 4.1. The results of these tests are presented in Table 4.2.

For SFC, we implemented Algorithm 12 ourselves. For each dataset, we used the Algorithm with $\alpha = \{0.00,\ 0.02,\ 0.04,\ 0.06,\ \cdots, 1.98\}$ (100 values). In Table 4.2, we present the F-measure of clustering selected that maximizes our mtis Equation 3.12, along with the α value that generated it.

4.5 Results

4.5.1 F-measure

Table 4.2 presents the F-measures obtained on the seven test sets for our four algorithms.

From Table 4.2, it is clear SFC is superior, in terms of F-measure, then SLINK, and is likewise superior to CLINK in all cases. Although SFC only surpasses UPGMA once, only one of its clusterings is significantly worse then UPGMA's result, namely the result for re1.

4.5.2 Correlation of mtis and F-measure

Recall that when we were designing a metric to calculate the optimal clustering to return as α varies, we discussed the following equation for internal similarity:

[1]The command was *vcluster -clmethod=agglo -crfun=*$< X >$ *-sim=cos -rowmodel=log -colmodel=idf -rclassfile=*$< Y >$*.mat.rclass* $< Y >$*.mat* $< N >$

[2]$< X >$ is the appropriate clustering algorithm name (upgma, slink, or clink), $< Y >$ is the name of the dataset being tested (hitech, wap, etc.), and $< N >$ is the number of clusters desired in the final solution. Vcluster is the name of the Cluto clustering program that clusters data from .mat files as input.

Set/Method	UPGMA	CLINK	SLINK	SFC
hitech	**0.510**	0.351	0.336	0.453,$\alpha = 0.920$
fbis	**0.675**	0.574	0.216	0.633,$\alpha = 0.580$
re0	**0.586**	0.442	0.359	0.545,$\alpha = 0.620$
re1	**0.700**	0.500	0.291	0.532,$\alpha = 0.700$
k1a	0.641	0.550	0.257	**0.677**,$\alpha = 1.120$
k1b	**0.892**	0.684	0.603	0.871,$\alpha = 1.120$
wap	**0.630**	0.496	0.450	0.593,$\alpha = 0.760$

Table 4.2: F-measures test results
The results for UPGMA, CLINK, SLINK, and SFC run on the seven test sets. For SFC, we present the F-measure of the clustering selected automatically by the algorithm, along with the α that generated it.

$$\phi_{isim}(C) = \sum_{C_i \in C} \frac{\sum\limits_{x,y \in C_i, y \neq x} sim_{cos}(x, y)}{|C_i|}$$

We stated this equation did not accurately implement our recall requirement.

We modified this metric to compensate for recall, but the motivation for the modification was not backed by a formal proof, only an intuition. Here we give empirical evidence that our modified metric is indeed a better way of selecting an optimal clustering to return. To show this, we first calculate mtis values using Equation 3.9 instead of Equation 3.10 for the three datasets FBIS, kla, and wap. We use Equation 3.9 to illustrate how poorly correlated mtis is without a modification for recall (see Figure 4.1). The r values displayed in Figure 4.1 are linear correlation values, i.e., how much of the variance for F-measure can be accounted for by the variance in mtis. Note the low values, indicating poor correlation. Mtis does not correspond well to F-measure without our recall modification.

Figure 4.2 plots mtis versus F-measure for the same three datasets, except that we factor in our $max_{cos}(D, x)$ using Equation 3.10. The correlation for each dataset is much greater than that in Figure 4.1. We have omitted the other four datasets to save space, but all follow the same trend of higher correlation when using Equation 3.10. Given this, we can conclude our new metric is indeed a better way to select an optimal F-measure clustering.

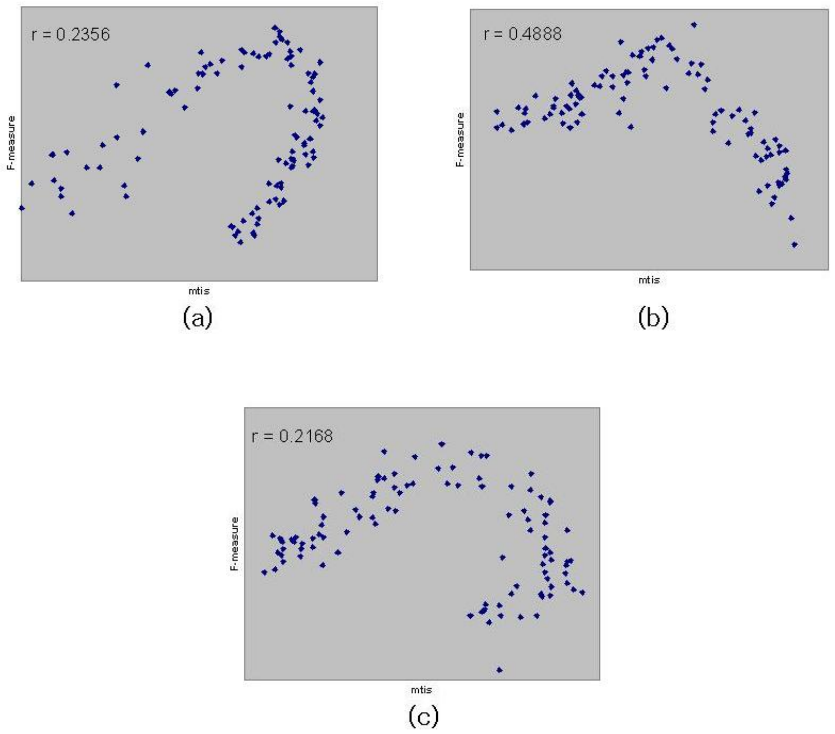

Figure 4.1: Correlation without compensating for cluster sizes.
(a) is the FBIS dataset, (b) is kla, and (c) is wap. The linear correlation values are the r's on each graph. All the correlations are low.

4.5.3 α and F-measure

It is desirable to understand how F-measure changes as α varies, to identify general trends over multiple datasets, and to possibly design a method with low time-complexity to select an optimal α than testing a large set of different values. Figure 4.3 shows how F-measure varies for four of our datasets.

From Figure 4.3, we can observe that each plot has a slight peak, some larger than others, but none of the peaks are overly "sharp". We further note that after each peak there is constant, slow downgrade in the F-measure of the clustering solution. The oscillations in each curve is a property common to all agglomerative clustering techniques, caused by a single decision early on have the potential to greatly affect the final output clustering.

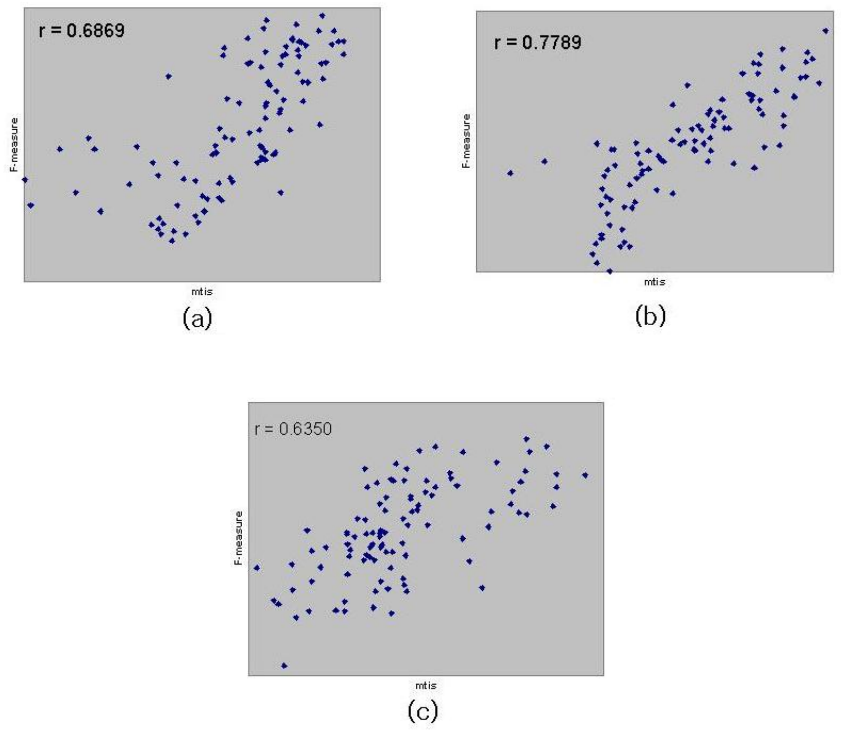

Figure 4.2: Correlation with compensating for cluster sizes.
(a) is the FBIS dataset, (b) is kla, and (c) is wap. The linear correlation values are the r's on each graph. Note that for all three datasets, the correlation between mtis and F-measure is substantially higher than the values without compensating for cluster size in Figure 4.1.

4.5.4 α and Features Remaining

As α increases, the number of non-zero features per vector decreases, resulting in decreases in both the time and space requirements of SFC. Figure 4.4 shows the percentage of features (words) remaining in total for all vectors versus α for each dataset.

The slope of the curves in Figure 4.4 shows that features drop off rapidly until roughly $\alpha = 1.00$. Furthermore the general form of the curve is identical for each dataset. An interesting thing to note is that all the optimal clusterings returned by SFC have $\alpha = 0.580$ or greater, at which point there is 50% or fewer features remaining. K1a and k1b peak at $\alpha = 1.120$, at which point only 20% of the features are remaining. This indicates many features are noise, at least as far as

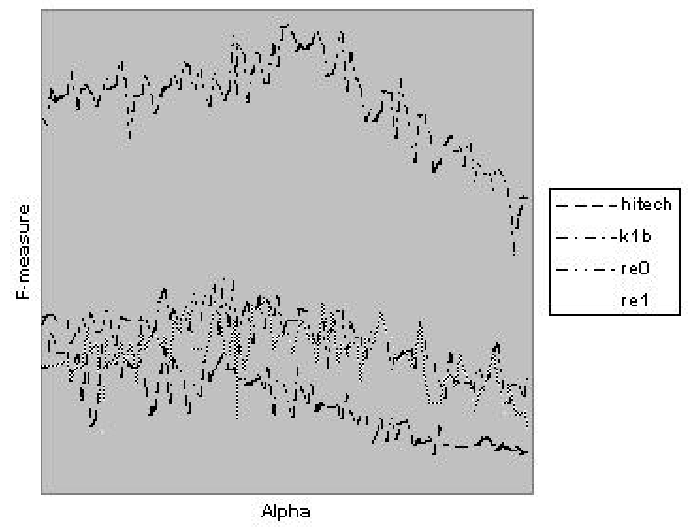

Figure 4.3: How F-measure varies with α
A plot of four of our datasets' (hitech, k1a, re0, and re1) F-measure values as α varies.

our algorithm is concerned.

4.6 Discussion

The significance vectors used in SFC are, by and large, boolean vectors of zeroes and ones. This conclusion can be drawn by understanding our tf-idf weighting scheme. In such a weighting scheme, the mean weight for each dimension is generally close to zero, leaving a very small range of values that are below the mean. Even a small α will enforce $\mu_k - \alpha_k \leq w_{ik}$ for any dimension k and document i. Examining equation 3.6, we note that having this property enforces either a one or zero value for every S_{ik}, thus we have boolean significance vectors unless α is very small.

Basic boolean vectors are very poor choices of data representations, but as SFC uses boolean representations and yields good F-measures, we must conclude that our significance vectors properly reflect, to some extent, which terms are significant to which document. If we can accurately represent significance using just our Boolean vectors, then perhaps tf-idf vectors are not the ideal choice for data representations in clustering. Certainly, other approaches already exist to cluster

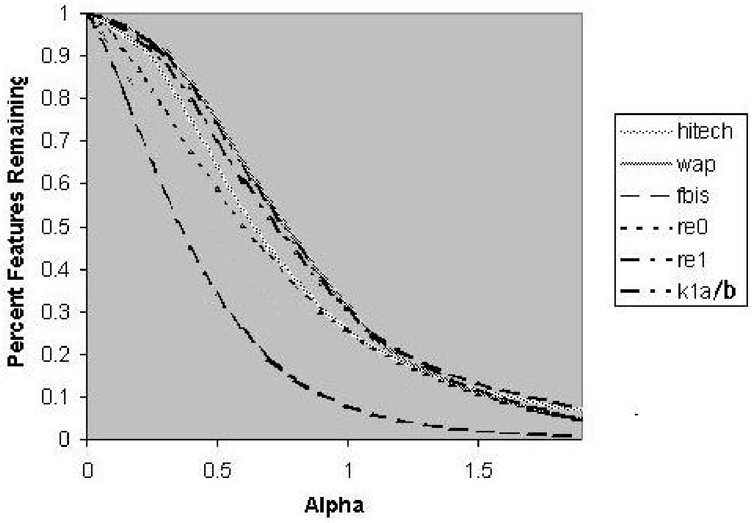

Figure 4.4: Percentage features remaining versus α
All the datasets follow a similar, smooth quickly dropping curve that levels off.

documents that are initially represented as frequency counts only, without mapping to tf-idf vectors. Such methods include links (ROCK [48]) and frequent item sets (HFTC [20], FTHC [43]), but few papers examine the effect that using different kinds of weighted vectors have on clustering quality. One conclusion we may draw from this thesis is that we should examine different and new vector-weighting methods, rather then just focus on new clustering algorithms. Just because tf-idf variants happen to be the de facto standard does not mean they should be so in clustering algorithms.

Another important aspect of our research is our new internal similarity metric which we call *maximum tree internal similarity*, which is defined over a tree of clusters. Defining the quality of a tree of clusters has always been a difficult process, but we have shown that mtis is better correlated with F-measure then internal similarity. The key, as we have also shown, is the $max_{cos}(D, x)$ component, which adds an element of recall to a metric that is otherwise exclusively quantifying precision. If a more refined way of factoring in recall could be found, mtis might be used to select an optimal clustering of text from any number of cluster hierarchies of a dataset, where different algorithms constructed each tree.

Despite the power of our mtis metric and the revelation our research provides about using different vectors, SFC itself is not practical in its current form. The run time is just too high, being equal to running a standard linkage multiplied by whatever number of α we examine. There is a pressing need for SFC to be able to select a single α value quickly, or just a small number. Examining Figure 4.3 suggests that it is possible to select an optimum α. We can observe there is a definite curve to each F-measure versus α plot, meaning a maximum can be found (ignoring the oscillations, which are, as we have said, not entirely avoidable in agglomerative clustering). If a good α can be found, then by combining it with existing speed-up methods such as sampling, SFC can be extremely fast as it uses vectors that are potentially only a fraction of the size of normal tf-idf vectors. Unfortunately, given SFC's current state, it is preferable to use UPGMA, as UPGMA is faster and slightly more accurate for almost all data sets.

4.7 Future Work

We consider the most notable directions for future work arising from our research to be:

Removal of Negative Significance As noted in Section 4.6, -1's are not even present when α is of any reasonable size. It would thus be desirable to revise some of our definitions so that our system is truly only dealing with zeroes and ones. The principal benefit of this change is clarity of methodology. Also, we could, for instance, use logical 'and' operations to compute dot products if our significance vectors are Boolean, making cosine computations faster. Alternatively, we could try methods to enforce -1's in equal proportion to 1's.

Estimating α Quickly Currently, we iterate over a range of α values and return the clustering which yields the best mtis value. This method produces good results, but is very slow. In order to select α quickly, we must use information that can be computed quickly, such as the basic word frequency count vectors or the tf-idf vectors. We are particularly interested in further study of tf-idf vectors, and believe that their distribution of weights may suggest an appropriate α.

Improving f_s As it stands, our f_s function assigns roughly the same number of non-zero values to every dimension, but as we discussed in Section 3.2.1, some words are better distinguishers than others. Such words should have a larger number of non-zero entries. To this end, we should alter Equation 3.6. One possible alteration is to assign the x top weights for a dimension k to be one, where x is some function of k's distribution in D, likely document frequency information.

Improving and Testing Mtis Mtis is well-correlated with F-measure, but the method with which we generated the correlation is suspect. We should incorporate a more principled notion of recall. This is a difficult task, as we must balance the recall aspect with the precision aspect, where both are just approximations (of true recall and true precision respectively). A better recall aspect than the one we selected would be some function of the amount of similarity a cluster has internally and the amount it has to objects outside itself. The function must be designed carefully, lest it overpower precision and always favour larger clusters. Another aspect of mtis is examining if one can select the optimal F-measure from trees produced by many different clustering algorithms using it.

Other Weighting Functions We would like to examine the effect of using various other weighting functions besides tf-idf values, ones not typically used in clustering, and to try and determine what kind of vectors are most suited to clustering.

Chapter 5

Conclusion

In this thesis, we presented a new text-clustering algorithm called Significant Feature Clustering. SFC's methodology is based on determining which words are significant to documents. To do this, our algorithm first generates standard tf-idf vectors from frequency count vectors, then maps these tf-idf vectors, using a parameter α, to significance vectors which contain entries representing whether each word is neutral, positive, or negative to a document. We demonstrated that by careful selection of α, the significance vectors produced accurately reflect the significances that words have to documents. By using an agglomerative implementation of SFC which merges clusters exhibiting the highest centroid cosine similarity, we were able to obtain final F-measures nearly as good as one of the more powerful agglomerative hierarchical clustering algorithms, group-average clustering. As part of this research, we developed a metric that is correlated with F-measure and does not use labels. This metric enabled the selection of a near optimal-clustering from a set of clusterings, each one created with a different α value.

However, SFC is too costly to run as presented here, but further research may allow its run-time to be significantly reduced. If so, then, considering the good F-measures we have obtained thus far, SFC may be applicable to large real data sets. The mtis value SFC uses to select a good clustering to return appears to be highly useful in its own right, and is certainly worth examining further. If this crude metric can correlate so well with F-measure relative to other metrics, then with further refinement it may be possible to create a mtis that is well-correlated with F-measure to use for any set of clusterings from a variety of algorithms, and reliably pick the best one.

Lastly, we again note that our research has revealed that standard tf-idf vectors may not be the best vector representation to use in clustering. This suggests that researchers should not only investigate new clustering algorithms, but also new data representations more amenable to clustering.

Bibliography

[1] Association for computing machinery. `http://portal.acm.org/dl.cfm`. Visited January 2006.

[2] Citeseer. `http://citeseer.ist.psu.edu/`. Visited January 2006.

[3] Distance metrics. `http://www.improvedoutcomes.com/docs/WebSiteDocs/Clustering/Clustering_Parameters/Distance_Metrics_Overview.htm`. Visited June 2006.

[4] Dmoz. `http://www.dmoz.org`. Visited January 2006.

[5] Google. `http://www.google.com`. Visited January 2006.

[6] Html stripper. `http://www.download.com/HTML-Stripper/3000-2058_4-10214035.html`. Visited September 2005.

[7] pdf2text. `http://www.traction-software.co.uk/pdf2text/`. Visited January 2006.

[8] Principal component analysis tutorial. `http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf`. Visited January 2006.

[9] ps2text. `http://www.square1.nl/TGC-SITE/ps2text.htm`. Visited January 2006.

[10] Trec. `http://trec.nist.gov/`. Visited Feburary 2006.

[11] Yahoo. `http://www.yahoo.com`. Visited January 2006.

[12] Ankerst, M., Breunig, M. M., Kriegel, H. P., and Sander, J. Optics: Ordering points to identify the cluster structure. In *Proceedings of the ACM SIGMOD 1999 International Conference on Management of Data* (1999), pp. 49–60.

[13] Apté, C., Damerau, F., and Weiss, S. M. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems 12*, 3 (1994), 233–251.

[14] Arabie, P., Hubert, L. J., and Soete, G. D. *Clustering and Classification.* World Scientific Publishing Co., New Jersey, 1996.

[15] Au, W., Chan, K. C. C., Wong, A. K. C., and Wang, Y. Attribute clustering for grouping, selection, and classification of gene expression. *IEEE/ACM Transactions on Computational Biology and Bioinformations 2*, 2 (2005), 83–101.

[16] Banerjee, A., Dhillon, I., Ghosh, J., and Sra, S. Generative model-based clustering of directional data. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2003), pp. 19–28.

[17] Banerjee, A., and Ghosh, J. On scaling up balanced clustering algorithms. In *Proceedings of the 2nd SIAM International Conference on Data Mining* (2002), pp. 333–349.

[18] Barbará, D., and Chen, P. Using the fractal dimension to cluster datasets. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining* (2000), pp. 260–264.

[19] Barbará, D., Li, Y., and Couto, J. COOLCAT: an entropy-based algorithm for categorical clustering. In *Proceedings of the 11th International Conference on Information and Knowledge Management* (2002), pp. 582–589.

[20] Beil, F., Ester, M., and Xu, X. Frequent term-based text clustering. In *Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining* (2002).

[21] Berkhin, P. Survey of clustering data mining techniques. Tech. rep., Accrue Software, San Jose, CA, 2002.

[22] Bezdek, J. C. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Kluwer Academic Publishers, Norwell, MA, 1981.

[23] Böhm, C., Kailing, K., Kröger, P., and Zimek, A. Computing clusters of correlation connected objects. In *Proceedings of the ACM SIGMOD Internation Conference on Data Management* (2004), pp. 455–466.

[24] Boley, D. Principal directions divisive partitioning. *Data Mining and Discovery 2*, 4 (1998), 325–344.

[25] Bradley, P. S., Fayyad, U., and Reina, C. Scaling clustering algorithms to large databases. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining* (1998), pp. 9–15.

[26] Cardie, C. Using decision trees to improve case based learning. In *Proceedings of the 10th International Conference on Machine Learning* (1993), pp. 25–32.

[27] Cheeseman, P., and Stutz, J. *Advances in Knowledge Discovery and Data Mining.* AAAI Press, Menlo Park, CA, 1996, ch. 6, pp. 158–180.

[28] Chen, J. N., and Chang, J. S. Topical clustering of mrd senses based on information retrieval techniques. *Computational Linguistics 24*, 1 (1998), 61–95.

[29] Cheng, C., Fu, A. W., and Zhaing, Y. Entropy-based subspace clustering for mining numerical data. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (1999), pp. 84–93.

[30] Cristianini, N., and Shawe-Taylor, J. *An Introduction to Support Vector Machines.* Cambridge University Press, Cambridge, MA, 1991.

[31] Crouch, D. B., Crouch, C. J., and Andreas, G. The use of cluster hierarchies in hypertext information retrieval. In *Proceedings of the 2nd ACM Conference on Hypertext* (1989), pp. 225–237.

[32] Dash, M., and Liu, H. Feature selection for clustering. In *Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2000), pp. 110–121.

[33] Defays, D. An efficient algorithm for a complete link method. *The Computer Journal*, 20 (1977), 364–366.

[34] Dempster, A. P., Laird, N., and Rubin, D. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society 39*, 1 (1977), 1–38.

[35] Ding, C., and He, X. K-means clustering via principal component analysis. In *Proceedings of the 21st International Conference on Machine Learning* (2004), pp. 29–37.

[36] Ding, C., He, X., Zha, H., and Simon, H. D. Feature subset selection and order identification for unsupervised learning. In *Proceedings of the 17th International Conference on Machine Learning* (2000), pp. 247–254.

[37] Dy, J. G., and Brodley, C. E. Feature selection for unsupervised learning. *The Journal of Machine Learning Research 5* (2004), 845–889.

[38] Ester, M., Kriegel, H., Sander, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases* (1996), pp. 226–231.

[39] Fisher, D. Knowledge acquistion via incremental conceptual clustering. *Machine Learning 2* (1987), 139–172.

[40] Foss, A., and Zaïane, O. R. A parameterless method for efficiently discovering clusters of arbitrary shape in large datasets. In *Proceedings of the 2002 IEEE International Conference on Data Mining* (2002), pp. 179–186.

[41] Fraley, C., and Raftery, A. MCLUST: Software for model-based cluster and discriminant analysis. Tech. Rep. 324, Department of Statistics, University of Washington, 1999.

[42] Fukunaga, K. *Introduction to Statistical Pattern Recognition*, 2nd ed. Academic Press, San Diego, CA, 1990.

[43] Fung, B. C., Wang, K., and Ester, M. Hierarchical document clustering using frequent itemsets. In *Proceedings of the 2003 SIAM International Conference on Data Mining* (2003), pp. 59–70.

[44] Gale, W. A., Church, K. W., and Yarowsky, D. A method for disambiguating word senses in a large corpus. *Computers and the Humanities*, 26 (1992), 415–439.

[45] Gennari, J., Langley, P., and Thompson, K. Models of incremental concept learning. *Artificial Intelligence*, 40 (1987), 11–61.

[46] Goldberg, D. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, 1989.

[47] Guha, S., Rastogi, R., and Shim, K. CURE: An efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD International Conference on Data Management* (1998), pp. 73–84.

[48] Guha, S., Rastogi, R., and Shim, K. ROCK: A robust clustering algorithm for categorical attributes. In *Proceedings of the 15th International Conference on Data Engineering* (1999), pp. 512–521.

[49] Guo, A. A new framework for clustering algorithm evaluation in the domain of functional genomics. In *Proceedings of the 2004 ACM Symposium on Applied Computing* (2004), pp. 143–146.

[50] Hamerly, G., and Elkan, C. Alternatives to the k-means algorithm that find better clusterings. In *Proceedings of the 11th International Conference on Information and Knowledge Management* (2002), pp. 600–607.

[51] Han, E., Boley, D., Ginii, M., Gross, R., Hastings, K., Karypis, G., Kumar, V., Mobasher, B., and Moore, J. WebACE: A web agent for document categorization and exploration. In *Proceedings of the 2nd Internation Conference on Autonomous Agents* (1998).

[52] Har-Peled, S., and Sadri, B. How fast is the k-means method. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms* (2005), pp. 887–885.

[53] Hartigan, J. *Clustering Algorithms*. John Wiley & Sons, Inc., New York, NY, 1975.

[54] Hinneburg, A., and Keim, D. A. An efficient approach to clustering large multimedia databases with noise. In *Proceedings of the 4th Internation Conference on Knowledge Discovery in Databases* (1998), pp. 58–65.

[55] Hinneburg, A., and Keim, D. A. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. *The Very Large Database Journal* (1999), 506–517.

[56] Hipp, J., Guntzer, U., and Nakhaeizadeh, G. Algorithms for association rule mining - a general survery and comparison. *ACM SIGKDD Explorations 2* (2000), 58–64.

[57] Hoffman, T. Probabilistic latent semantic analysis. In *Proceedings of the 22nd ACM Conference on Research and Development in Information Retrieval* (1999), pp. 50–57.

[58] Huang, Z. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery 2*, 3 (1998), 283–304.

[59] Jain, A. K., Murty, M. N., and Flynn, P. J. Data clustering: A review. *ACM Computing Survey 31*, 3 (1999), 264–323.

[60] John, G. H., Kohavi, R., and Pfleger, M. Irrelevant features and the subset selection problem. In *Proceedings of the 10th International Conference on Machine Learning* (1993), pp. 25–32.

[61] Jonyer, I., Cook, D. J., and Holder, L. Graph-based hierarchical conceptual clustering. *The Journal of Machine Learning 2* (2002), 19–43.

[62] Karypis, G. Cluto. `http://www-users.cs.umn.edu/~karypis/cluto/`. Visited January 2006.

[63] Kasturi, J., and Acharya, R. Clustering of diverse genomic data using information fusion. In *Proceedings of the 2004 ACM Symposium on Applied Computing* (2004), pp. 116–120.

[64] Kaufman, L., and Rousseeu, P. *Finding Groups in Data.* John Wiley & Sons, Inc., New York, NY, 1990.

[65] Kim, Y., Street, W. N., and Menczer, F. Feature selection in unsupervised learning via evolutionary search. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2000), pp. 365–369.

[66] Kohavi, R., and John, G. H. Wrappers for feature subset selection. *Artificial Intelligence 97*, 1-2 (1997), 273–324.

[67] Kriegel, H. P., and Pfeifle, M. Density-based clustering of uncertain data. In *Proceedings of 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining* (2005), pp. 672–677.

[68] Leuski, A. Evaluating document clustering for interactive information retrieval. In *Proceedings of the 10th International Conference on Information and Knowledge Management* (2001), pp. 33–40.

[69] Lewis, D. D. Reuters-21578 text categorization test collection distribution 1.0. `http://www.research.att.com/~lewis`. Visited January 2006.

[70] Lewis, D. D., and Ringuette, M. Comparison of two learning algorithm for text categorization. In *Proceedings of the 3rd Annual Symposium on Document Analysis and Information Retrieval* (1994), pp. 83–91.

[71] Li, T., Ma, S., and Ogihara, M. Entropy-based criterion in categorical clustering. In *Proceedings of the 21st International Conference on Machine Learning* (2004), pp. 68–75.

[72] MacQueen, J. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statitics and Probability* (1967), pp. 281–298.

[73] Mandhani, B., Joshi, S., and Kummamuru, K. A matrix density based algorithm to hierarchically co-cluster documents and words. In *Proceedings of the 12th International Conference on the World Wide Web* (2003).

[74] Manning, C. D., and Schütze, H. *Foundations of Statistical Natural Language Processing.* The MIT Press, Cambridge Massachusetts, London England, 2003, ch. 16.

[75] Neal, R., and Hinton, G. *A View of the EM Algorithm that Justifies Incremental, Sparse, and other Variants.* Kluwer Academic Publishers, Norwell, MA, 1998, ch. 12, pp. 355–368.

[76] Ng, C., Sia, K., and Chan, C. Peer clustering and firework query model.

[77] Ordonez, C. Clustering binary data streams with k-means. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery* (2003), pp. 12–19.

[78] Parsons, L., Haque, E., and Liu, H. Subspace clustering for high dimensional data: A review. *ACM SIGKDD Explorations Newsletter 6*, 1 (2004), 90–105.

[79] Pelleg, D., and Moore, A. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the 17th International Conference on Machine Learning* (2000), pp. 727–734.

[80] Rissanen, J. Modeling by shortest data description. *Automatica 14* (1979), 465–571.

[81] Robertson, S. E., and Walker, S. Some simple effective approximations to the 2-poisson model for probabilistic information retrieval. In *Proceedings of the 1994 ACM SIGIR* (1994), pp. 232–241.

[82] Salton, G., Yang, C. S., and Yu, C. T. A vector space model for automatic indexing. *Communications of the ACM 18* (1975), 613–620.

[83] Sander, J., Ester, M., Kriegel, H., and Xu, X. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Mining and Knowledge Discovery 2* (1998), 169–194.

[84] Sato, M., and Ishii, S. On-line EM algorithm for the normalized Gaussian network. *Neural Computing 2*, 12 (2000).

[85] Schikuta, E. Grid-clustering: An efficient hierarchical clustering method for very large data sets. In *Proceedings of the 13th International Conference on Pattern Recognition* (1996), pp. 101–105.

[86] Schikuta, E., and Erhat, M. The bang-clustering system: Grid-based data analysis. *Lecture Notes in Computer Science 1280* (1997), 513–524.

[87] Sheikholeslami, G., Chatterjee, S., and Zhang, A. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *Proceedings of the 24th International Conference on Very Large Databases* (1998), pp. 428–439.

[88] Sibson, R. SLINK: An optimally efficient algorithm for the single link cluster method. *The Computer Journal*, 16 (1973), 30–34.

[89] Sneath, P. H., and Snokal, R. R. *Numerical Taxonomy.* W.H. Freeman and Company, San Francisco, CA, 1973.

[90] Steinbach, M., Karypis, G., and Kumar, V. A comparison of document clustering techniques. In *KDD Workshop on Text Mining* (2000).

[91] Strehl, A., and Ghosh, J. Value-based customer grouping from large retail data-sets. In *Proceedings of the SPIE Conference on Data Mining and Knowledge Discovery* (2000), vol. 4057, pp. 50–57.

[92] Thompson, K., and Langley, P. *Concept Formation Knowledge and Experience in Unsupervised Learning.* Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1991.

[93] Tung, A. K. H., Ng, R. T., Lakshmanan, L. V. S., and Han, J. Constraint-based clustering in large databases. In *Proceedings of the 8th International Conference on Database Theory* (2001), pp. 405–419.

[94] Voorhees, E. M. Implementing agglomerative hierarchical clustering algorithms for use in document retrieval. *Information Processing and Management 20*, 6 (1986), 465–476.

[95] Wang, W., Yang, J., and Muntz, R. Sting: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd International Conference on Very Large Databases* (1997), pp. 185–196.

[96] Wang, W., Yang, J., and Muntz, R. Sting+: An approach to active spatial data mining. In *Proceedings of the 1999 International Conference on Data Engineering* (1999), pp. 116–125.

[97] Weiner, E., Penderson, J. O., and Weigend, A. S. A neural network approach to topic spotting. In *Proceedings of the 4th Annual Symposium on Document Analysis and Information Retrieval* (1995), pp. 317–322.

[98] Wu, J., and Lin, Z. Research on customer segmentation model by clustering. In *Proceedings of the 7th International Conference on Electronic Commerce* (2005), pp. 316–318.

[99] Xu, J., and Croft, W. B. Corpus-based stemming using co-occurrence of word variants. *ACM Transactions on Information Systems 16*, 1 (1998), 62–81.

[100] Xu, X., Ester, M., Kriegel, H., and Sander, J. A distribution based clustering algorithm for mining in large spatial databases. In *Proceedings of the 14th International Conference on Data Engineering* (1998), pp. 324–331.

[101] Zhang, B., Hsu, M., and Dayal, U. K-harmonic means - a data clustering algorithm. Tech. Rep. HPL-1999-124, Hewlett-Packard Labs, 1999.

[102] Zhao, Y., and Karypis, G. Evaluation of hierarchical clustering algorithms for document datasets. In *Proceedings of the 11th International Conference on Information and Knowledge Management* (2002), pp. 515–524.