Recent Developments in Document Clustering

Nicholas O. Andrews and Edward A. Fox

Department of Computer Science, Virginia Tech,
Blacksburg, VA 24060

{nandrews, fox}@vt.edu

October 16, 2007

Abstract

This report aims to give a brief overview of the current state of document clustering research and present recent developments in a well-organized manner. Clustering algorithms are considered with two hypothetical scenarios in mind: online query clustering with tight efficiency constraints, and offline clustering with an emphasis on accuracy. A comparative analysis of the algorithms is performed along with a table summarizing important properties, and open problems as well as directions for future research are discussed.

Contents

1 Introduction

Document (or text) clustering is a subset of the larger field of data clustering, which borrows concepts from the fields of information retrieval (IR), natural language processing (NLP), and machine learning (ML), among others. Document clustering will hereafter be simply referred to as *clustering*.

The process of clustering aims to discover natural groupings, and thus present an overview of the classes (topics) in a collection of documents. In the field of artificial intelligence, this is known as unsupervised machine learning. Clustering should not to be confused with classification. In a classification problem, the number classes (and their properties) are known a priori, and documents are assigned to these classes. Conversely, in a clustering problem, neither the number[1], properties, or membership (composition) of classes is known in advance. This distinction is illustrated in Figure 1. Classification would be an example of supervised machine learning.

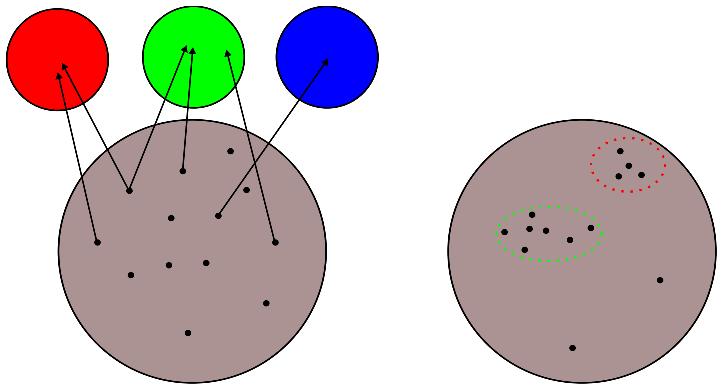

(a) Classification (b) Clustering

Figure 1: In (a), three classes are known *a priori*, and documents are assigned to each of them. In (b), an unknown number of groupings must be inferred from the data based on a similarity criterion (in this case, distance).

A good clustering can be viewed as one that organizes a collection into groups such that the documents within each group are both similar to each other and dissimilar to those in other groups. Clustering can

[1]In practice, the problem is often simplified by supplying a target number of clusters.

either produce disjoint or overlapping[2] partitions. In an overlapping partition, it is possible for a document to appear in multiple clusters. As the question of what constitutes a good clustering is central to both the formulation of clustering algorithms and their evaluation, Section 2 describes commonly used metrics.

The first challenge in a clustering problem is to determine which features of a document are to be considered discriminatory. In other words, we seek a *document model*. A majority of existing clustering approaches choose to represent each document as a vector, therefore reducing a document to a representation suitable for traditional data clustering approaches. This model is discussed in Section 3.

There is a daunting amount of literature on clustering. Exploring this literature is complicated by the fact that there are many domains in which clustering can be applied. This overview makes no attempt to present all existing algorithms, but rather presents classes of algorithms along with representatives. While an effort has been made to select the "best" representative algorithms in each class, there are certainly algorithms not considered and potentially even classes of algorithms not considered. In this respect, an emphasis has been placed on fuzzy clustering algorithms, since text data inherently spans multiple topics and accounting for this produces better clusters [17, 55, 15].

Clustering algorithms can be divided into discriminative and generative types. Broadly speaking, discriminative algorithms operate on pairwise similarities between every document, and based on these similarities optimize a criterion (or objective) function to produce an optimal clustering. Generative algorithms, on the other hand, assume an underlying distribution of the data, and maximize the fit of distribution to produce cluster centroids. Recent developments in both of these classes of algorithms are discussed in Section 4 and Section 5.

An alternative view of the vector space resulting from the terms and documents is as an adjacency matrix. An adjacency matrix, in graph theory, defines the connectivity of vertices in a graph. Under the vector model, this results in a weighted graph. Producing a clustering in a graph is the multiway cut problem, where cuts are found to partition the vertices while breaking as few edges as possible. These approaches are described in Section 6.

As a result of the choice of the vector representation, the term-document matrix can grow to be very large. There has been recent interest in the equivalence between producing an optimal low-rank approximation of a matrix and producing an optimal clustering of the same data. Section 7 describes a number of matrix factorization techniques and explores these connections.

A potential weakness of the vector model is that it encodes no information about word order. There has been some recent interest in alternative representations that include in their definition of similarity not only occurrence of terms, but how frequently terms occur together, or in sequence. These representations are discussed in Section 8. A closely related issue is that of generating "readable" cluster labels, as opposed to a list of most significant terms per cluster as is typically used.

Finally, the properties of the algorithms discussed in this work are discussed in Section 9, including their applicability to query clustering and the clustering of larger, more static collections.

2 Good, Bad, and Ugly Clustering

Clustering algorithms have been evaluated in many ways. Unfortunately, there is little agreement over which is the best way to do so. The choice of evaluation methods frequently depends on the domain in which the research is being conducted. For example, an AI researcher might favor mutual information, while someone from the field of IR would choose F-measure. These metrics, and others, will be discussed here.

Two intuitive notions of performance (accuracy) are precision and recall. In the field of IR, recall is defined as the proportion of relevant documents that are retrieved out of all relevant documents available, while precision is the proportion of retrieved *and* relevant documents out of all retrieved documents. Because it is trivial to get perfect recall by retrieving all documents for any query, the F-measure, which combines both recall and precision, is introduced. Let R be recall and P be precision, then the generalized F-measure is defined as

[2]Also called fuzzy or soft partitions.

$$F_\alpha = \frac{(1+\alpha)RP}{\alpha P + R}. \tag{1}$$

Precision and recall are typically given equal weight (F_1), but variations exist which weight them differently, e.g. precision twice as much as recall, or vice versa (these are labeled $F_{0.5}$, F_2, respectively).

To extend this to clustering, we assume the existence of a set of reference classes, and the found clusters (the output of the clustering algorithm) are treated as retrieved documents from these classes. Adopting the notation in [50], for class i and cluster j,

$$F = \sum_i \frac{n_i}{n} \operatorname*{argmax}_j F(i,j). \tag{2}$$

where n is the total number of documents. In words, a one-to-one correspondence is established between classes and clusters based on those that are most similar.

While F-measure addresses the total quality of the clustering in terms of retrieval performance, it does not address the composition of the clusters themselves. Two additional measures are cluster purity and entropy. Purity measures the percentage of the dominant class members in a given cluster (larger is better), while entropy looks at the distribution of documents from each reference class within clusters (smaller is better). These are written as[3]

$$Purity = \sum_j \frac{n_j}{n} \operatorname*{argmax}_i P(i,j) \tag{3}$$

$$Entropy = -\frac{1}{\log k} \sum_j \frac{n_j}{n} \sum_i P(i,j) \log P(i,j) \tag{4}$$

If the number of clusters is the same as number of categories, and a correspondence can be established, e.g. Equation 2, then the measures above can be applied with some success. However, if there is a discrepancy between the number of reference clusters and those found by the algorithm, then the accuracy is sometimes not very indicative of the quality of the clusters.

It has been shown that in these cases mutual information (MI) is a superior measure than purity or entropy [44]. In practice, MI is normalized to unit length (NMI). Following the definition in [59], let n_h be the number of documents in class h, n_l be the number of samples in cluster l, and $n_{h,l}$ be the number of samples in class h and cluster l, then:

$$NMI = \frac{\sum_{h,l} n_{h,l} \log \frac{n n_{h,l}}{n_h n_l}}{\sqrt{(\sum_h n_h \log \frac{n_h}{n})(\sum_l n_l \log \frac{n_l}{n})}} \tag{5}$$

The NMI range is $[0,1]$, where a value of one denotes a perfect match between clusters and reference classes. Again, the motivation for NMI is that it is a clustering accuracy measure that is tolerant to mismatches between number of clusters and the number of reference classes. A significant number, if not a majority, of the clustering literature assumes a given number of clusters k, and so in this case traditional measures can be used.

A different view of the quality of a clustering algorithm is to look at the stability of the partitions it produces [39] accross runs. An intuitive approach to measuring stability is to look at average performance. In terms of mutual information, this results in an average normalized mutual information: let $\mathbf{\Lambda}$ be a set of r clusterings, and $\hat{\lambda}$ a single clustering, then ANMI is defined as

$$\varphi(\mathbf{\Lambda}, \hat{\lambda}) = \frac{1}{r} \sum_i NMI(\hat{\lambda}, \lambda_i) \tag{6}$$

[3]Note that the entropy is normalized.

While confusion matrices do not occur as frequently in the clustering literature as the other methods described above, they occur frequently enough to warrant a short description. A confusion matrix[4] is a visualization tool that provides a summary of misclassifications (errors) made by the system.

Table 1: A confusion matrix for the classes A (4 elements), B (4 elements), and C (8 elements).

	A	**B**	**C**
A	2	2	0
B	2	2	0
C	0	0	8

In Figure 1, there are three classes of objects (e.g. documents). It is interpreted as follows: the clustering algorithm cannot distinguish between classes A and B, but distinguishes objects from class C perfectly. That is, along row and down column C, the zeros indicate that no objects from A or B were classed as objects from C. On the other hand, half of A was misclassified in B, and vice versa.

The metrics described have been for disjoint clusters. A common method of evaluating the output of a fuzzy clustering algorithm is to produce hard clusters from a fuzzy output (this has been called *hardening* the clusters), which can be accomplished by thresholding document memberships. That is, documents within a threshold value γ are considered inside the cluster, and so if a document is within γ for two clusters, it will appear in both.

3 Vector Space Model

The vector model [40] was originally developed for automatic indexing. Under the vector model, a collection of n documents with m unique terms is represented as an $m \times n$ term-document matrix (where each document is a vector of m dimensions). Although the model itself is well-established, it forms the basis for subsequent discussion, and there has been some recent interest on alternative document representations. Therefore, a brief description follows.

Several term weighing schemes have been used, including binary term frequency and simple term frequency (i.e. how many times the words occur in the document). In the most popular scheme, the document vectors are composed of weights reflecting the frequency of the terms in the document multiplied by the inverse of their frequency in the entire collection ($tf \times idf$). The assumption is that words which occur frequently in a document but rarely in the entire collection are of highly discriminative power. Under all these schemes, it is typical to normalize document vectors to unit length.

Most clustering algorithms use a vector space representation in one way or another, although it should be noted at this point that no information about word order is encoded, which is why the vector model is sometimes called the *bag of words* or *dictionary* model. Section 8 describes alternative representations that maintain word order information.

Two important properties should be stressed. First, in a collection of heterogeneous topics (as is typical of a collection we wish to cluster), the number of unique terms will be quite large. This results in document vectors of high dimensionality (the so called "curse of dimensionality"). A number of preprocessing steps are described next that can help address this issue. Second, a matrix resulting from a typical corpus under the vector model will be highly sparse, since the corpus contains many more terms than the individual documents that compose it.

3.1 Preprocessing

Preprocessing consists of steps that take as input a plain text document and output a set of tokens (which can be single terms or n-grams[5]) to be included in the vector model. These steps typically consist of:

[4]In unsupervised learning, it is sometimes called a *matching* matrix.

[5]A sequence of one or more terms.

Filtering The process of removing special characters and punctuation that are not thought to hold any discriminative power under the vector model. This is more critical in the case of formatted documents, such as web pages, where formatting tags can either be discarded or identified and their constituent terms attributed different weights [18].
Tokenization Splits sentences into individual tokens, typically words. More sophisticated methods, drawn from the field of NLP, parse the grammatical structure of the text to pick significant terms or chunks (sequences of words), such as noun phrases [32].
Stemming The process of reducing words to their base form, or stem. For example, the words "connected," "connection", "connections" are all reduced to the stem "connect." Porter's algorithm [37] is the *de facto* standard stemming algorithm.
Stopword removal A stopword is defined as a term which is not thought to convey any meaning as a dimension in the vector space (i.e. without context). A typical method to remove stopwords is to compare each term with a compilation of known stopwords. Another approach is to first apply a part-of-speech tagger and then reject all tokens that are not nouns, verbs, or adjectives.
Pruning Removes words that appear with very low frequency throughout the corpus. The underlying assumption is that these words, even if they had any discriminating power, would form too small clusters to be useful. A pre-specified threshold is typically used, e.g. a small fraction of the number of words in the corpus. Sometimes words which occur too frequently (e.g. in 40% or more of the documents) are also removed.

In addition to the fairly common steps above, a lexical database, WordNet [31], has been used to overcome synonymy and introduce more general concepts with some success [42]. Note that the most effective method actually introduces more terms into the vector representation.

3.2 An Example

To illustrate this process, consider the first few sentances of the abstract of [6]:

> Document clustering has not been well received as an information retrieval tool. Objections to its use fall into two main categories: first, that clustering is too slow for large corpora (with running time often quadratic in the number of documents); and second, that clustering does not appreciably improve retrieval. We argue that these problems arise only when clustering is used in an attempt to improve conventional search techniques.

After filtering, tokenization, stemming, and stopword removal, we obtain:

> document cluster receiv inform retriev tool object us fall main categori first cluster slow larg corpora run time often quadrat number document second cluster appreci improv retriev argu problem aris cluster us attempt improv convent search techniqu

While notably harder to read, the preprocessed abstract has roughly half as many unique terms as the original, and can actually improve retrieval performance [20]. However, in even in moderate sized corpus document vectors can still have thousands of dimensions. Methods to address this problem by matrix factorization are discussed in Section 7.

4 Extensions to *k*means

Historically, hierarchical and partitional algorithms have been the dominant clustering methods. As many recent developments have basis in these approahes, they will be briefly described next.

In hierarchical clustering, each document is initially its own cluster. Hierarchical algorithms work by succesively merging documents, converging at the root of the hierarchy (also called a dendogram or tree), which includes all documents in the corpus ("bottom up" clustering). Advantages of this method are that a number of clusters need not be supplied in advance, and that the resulting cluster hierarchy is "browsable."

On the other hand, hierarchical algorithms do not scale (at each merge phase, similarities must be compared), and so are not appropriate for real-time applications or large corpora. Furthermore, it is generally accepted that partitional algorithms perform better than hierarchical ones [43]. In the cases where a hierarchy is desired, superior results have been achieved by using partitional approaches at each level of the hierarchy [56].

Partitional methods, of which the classical example is $k$means, start by choosing k initial documents as clusters, and iteratively assign documents to clusters while updating the centroids of these clusters ("top down" clustering). It is well known that text data is directional, and so it is typical to normalize document vectors, and to use a cosine similarity measure rather than Euclidian distance. The resulting algorithm is called spherical $k$means [9].

Although popular, in part due to its ease of implementation, $k$means suffers from a number of drawbacks: it relies on what generally amounts to stochastic (random) initialization; it can converge to suboptimal local minima; it is succeptible to outliers and noise; and its complexity is $\mathcal{O}(nkl)$, where n is the number of documents in the corpus, k is the desired number of clusters, and l is the number of iterations.

4.1 Online spherical $k$means

The online spherical $k$means [57] (oskmns) is an extension of the original spherical $k$means (skmns) that uses competitive learning techniques to speed up clustering while achieving similar or better accuracy. This algorithm produces clusters as accurate (± 0.05 NMI) as CLUTO, a graph-based clustering algorithm (see Section 6).

In an online competitive learning scheme, documents are streamed continuously. Contrast this with a batch algorithm, which is run on the entire data collection in a single "batch." As documents are added to the clustering, cells (clusters) compete for the input and the "winner" (the cluster to which the document is added) adjusts to respond more strongly to future input according to a certain learning rate. This learning rate is described below.

Let $\mathbf{d}$ be a document vector in corpus $\mathcal{D}$. Then both skmns and oskmns aim to minimize the following objective function:

$$J = \sum_{\mathbf{d} \in \mathcal{D}} \mathbf{d}^T \mu_{k(d)} \tag{7}$$

where the index $k(d) = \operatorname{argmax}_k \mathbf{d}^T \mu_k$ denotes the respective cluster center of each $\mathbf{d} \in \mathcal{D}$, and $\mathbf{d}^T \mu_{k(d)}$ is the dot product of the document vector and this centroid. Rather than updating the cluster centroids by computing the mean, the online variant factors a learning rate η into the update function:

$$\mu^*_{k(x)} = \frac{\mu_{k(d)} + \eta \mathbf{d}}{|\mu_{k(d)} + \eta \mathbf{d}|} \tag{8}$$

The learning rate η controls how clusters "adjust" to future input (documents). An annealing learning rate (gradually decreasing) was found to work best, where

$$\eta_t = \eta_0 \left(\frac{\eta_f}{\eta_0} \right)^{\frac{t}{NM}} \tag{9}$$

Here, N is the number of documents, M is the number of batch iterations, and η_f the desired final learning rate ($\eta_f = 0.01$ has been used).

There are two main advantages to oskmns: first, it is online, and so as documents are added to a corpus, there is no need to rerun a batch clustering on the entire set of documents. Second, opkmns converges about twice as quickly as skmns, and so while it cannot achieve the same levels of accuracy as other algorithms described in this work, it can be appropriate for realtime applications.

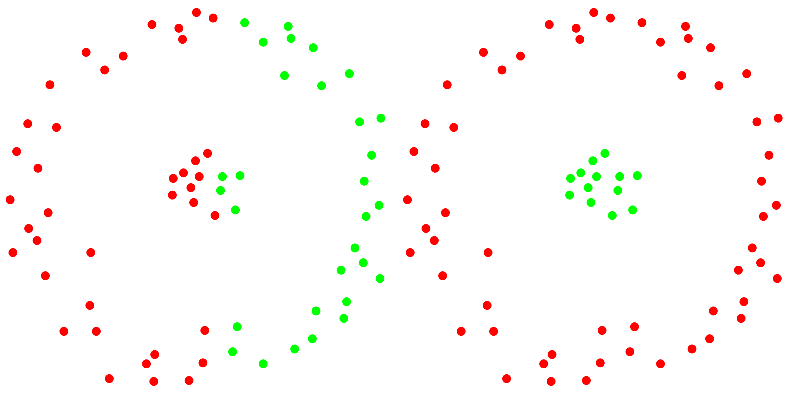

(a) *k*means (b) Kernel *k*means

Figure 2: (a) shows a non-convex shape which the traditional *k*means has trouble clustering. In (b), kernel *k*means succesfully separates the two clusters as a result of projecting the document vectors to higher dimensional space.

4.2 Kernel *k*means

Figure 2 illustrates a dataset which *k*means has trouble correctly clustering because the points are not linearly separable. The idea behind kernel *k*means is to find a mapping $\phi(\mathbf{d})$ to a higher dimensional space where it is more likely that the documents can be linearly seperated. In practice, a kernel function is defined which measures the affinity between every pair of documents. Examples include the polynomial kernel, gaussian kernel, or, for instance, the sigmoid kernel defined in Equation 10 (for appropriate values of c and θ):

$$\kappa(\mathbf{d_i}, \mathbf{d_j}) = \tanh\left(c(\mathbf{d_i} \cdot \mathbf{d_j}) + \theta\right) \tag{10}$$

The kernel function κ can be represented by an $n \times n$ square kernel (or Gram) matrix $\mathbf{K}$, where $K_{ij} = \kappa(\mathbf{d_i}, \mathbf{d_j})$. A generalization of this formulation assigns a weight $w(\mathbf{d})$ for each document. In this case, the affinity matrix K can be viewed as defining the edge weights of a graph, from which spectral methods can applied to find optimal cuts [8] (see for example the multiway normalized cuts algorithm [51]).

The choice of kernel function is important. For text data, cosine similarity usually works, although for some collections other functions might yield more accurate results. The choice of kernel function remains an open issue with regards to kernel methods.

Kernel *k*means [41] is slower than *k*means, although there has been some effort in speeding it up for higher dimensional data (case in point: text data) [54]. Furthermore, it has been shown that the weighted kernel *k*means objective function is identical to that of the normalized cut. The implication is that *k*means type iterative algorithms can be used for minimizing the normalized cut of a graph, which is an equivalent clustering problem. This is one of many connections between different formulations of the clustering problem.

Another problem with kernel methods is that of “diagonal dominence” [16]. Briefly, this problem occurs when objects (i.e. documents) are considered very similar with respect to themselves compared with their similarity to other objects. The resulting similarity matrix will have very large values down the diagonal compared to all other values. It has been observed that this can limit the extent to which the solution space is explored beyond the initial state [8]. That said, empirical studies have shown that kernel *k*means can outperform even the multi-stage algorithms of e.g. Section 7.3.

Unsupervised kernel methods can be applied to produce overlapping clusterers; for instance, the fuzzy *c*means algorithm has been extended using kernel functions [23].

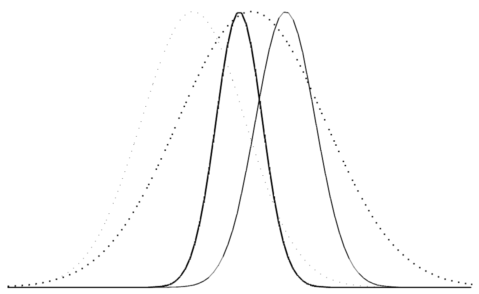


Figure 3: A mixture of Gaussians. The model assumes that the documents in the corpus are generated from clusters following a Gaussian distribution.

5 Generative Algorithms

Algorithms such as fuzzy *c*means are sensitive to outliers. In heterogenous collections of documents, these are quite frequent. By making certain assumptions about the distribution of the data, more robust statistical methods can be applied to discover clusters in the presence of noise while naturally accounting for multiple document memberships (i.e. overlapping clusters).

Discriminative methods based on pairwise document similarity have by definition $\mathcal{O}(n^2)$ runtime complexity. It is often the case that these similarities can be precomputed and stored in a matrix. Generative models, on the other hand, do not require such a matrix using an iterative procedure that alternates between model estimation and document assignment steps.

5.1 Gaussian model

The Gaussian mixture model represents a dataset as a set of means and covariance matrices. Under this model, each cluster is centered at the mean and is described by its associated matrix. The clustering problem is then the problem of finding the cluster means and covarience matrices which best fit the set of documents. This is illustrated in Figure 5.1.

Adopting a slightly modified version of the notation from [27], the model $\mathbf{\Theta}$ consists of a known number of clusters, with clusters each denoted θ. Assuming an $m \times n$ vector model generated from a corpus $\mathcal{D}$, every cluster $\theta \in \mathbf{\Theta}$ is an m-dimensional Gaussian distribution, with centroid μ and covarieance matrix Σ. Thus, each cluster θ contributes to document vectors $\mathbf{d} \in \mathcal{D}$ according to

$$\mathcal{P}(\mathbf{d}|\theta) = \frac{1}{(2\pi)^{\frac{m}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{(\mathbf{d}-\mu)^T \Sigma^{-1} (\mathbf{d}-\mu)}{2} \right). \tag{11}$$

Under this model, the total probability is given by

$$\mathcal{P}(\mathbf{d}|\mathbf{\Theta}) = \sum_{\theta \in \Theta} \mathcal{P}(\theta)\mathcal{P}(\mathbf{d}|\theta). \tag{12}$$

We are interested in maximizing this expression to obtain local maximizers, one for each cluster. The next section describes a tractable algorithm that does just this.

5.2 Expectation maximization

The em[6] algorithm [33] is an efficient iterative procedure to compute a Maximum Likelihood (ML) solution to a model. It is composed of two steps. In the expectation step, or E-step, the missing data is estimated

[6] It is conventional to refer to the algorithm in lowercase.

given the observed data (the collection of documents) and current estimate of the model (the clusters). In the maximization, or M-step, the likelihood function is maximized under the assumption that the missing data are known. For a more in depth discussion, refer to [4].

One iteration of the algorithm consists of the E-step, in which probability (or likelihood) $\mathcal{P}$ is computed for each document given the cluster estimates,

$$\mathcal{P}(\theta|\mathbf{d}) = \frac{\mathcal{P}(\theta)\mathcal{P}(\mathbf{d}|\theta)}{\sum_{\theta\in\Theta}\mathcal{P}(\theta)\mathcal{P}(\mathbf{d}|\theta)} \tag{13}$$

$$\mathcal{P}(\theta)^* = \sum_{d\in\mathcal{D}} \mathcal{P}(\theta|\mathbf{d}). \tag{14}$$

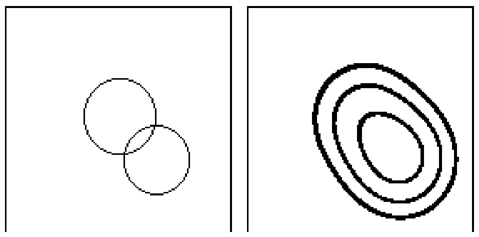
.

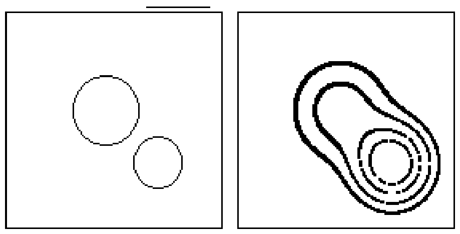
And the M-step, which updates the parameters of model θ to maximize the likelihood given the likelihoods computed in the E-step,

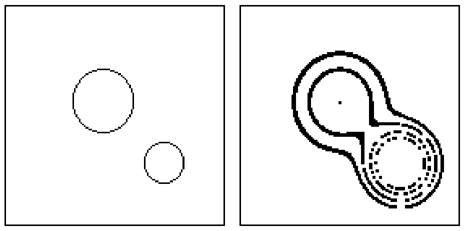
$$\mu = \frac{\sum_{\mathbf{d}\in\mathcal{D}}\mathcal{P}(\theta|\mathbf{d})\mathbf{d}}{\sum_{\mathbf{d}\in\mathcal{D}}\mathcal{P}(\theta|\mathbf{d})} \tag{15}$$

$$\Sigma = \frac{\sum_{\mathbf{d}\in\mathcal{D}}\mathcal{P}(\theta|\mathbf{d})(\mathbf{d}-\mu)(\mathbf{d}-\mu)^T}{\sum_{\mathbf{d}\in\mathcal{D}}\mathcal{P}(\theta|\mathbf{d})}. \tag{16}$$

It can proved that the algorithm converges to local minima with the log-likelihood that the corpus $\mathcal{D}$ is generated from the model Θ as a termination condition [28]. Figure 4 shows four iterations of the algorithm[7].


(a) $n = 1$


(b) $n = 2$


(c) $n = 3$

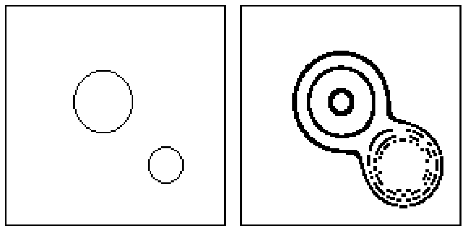

(d) $n = 4$

Figure 4: The em algorithm after n iterations. An insightful view of the algorithm is to think of it in terms of lower-bound estimation: at each iteration, a "tighter" lower bound is computed, and the cluster estimates "climb" closer to the (unknown) true distribution.

As in any maximum likelihood scheme, having too many free parameters with insufficient data may lead to overfitting. In [27], this problem is alleviated by applying the singular value decomposition to the vector space of words and documents. They then pick the twenty dimensions which have the largest singular values to form a reduced space on which to conduct the clustering.

[7]Source code and more examples are available at `http://www.cs.ucsd.edu/users/ibayrakt/java/em/`

[27] further introduces an iterative cluster refinement stategy from the initial set of clusters. The approach is based on finding a set of discriminative features features for each cluster, where discriminative features as simply those that that occur more frequently inside a given cluster than out of it (based on a threshold). The procedure then reassigns documents based on shared discriminative features, and the entire procedure is repeated until convergence. While this approach might seem ad hoc, the experimental results show consistent improvements (on the order of 30% on some datasets) over em clustering alone.

Such improvements can be interpreted as evidence that the assumptions concering the distribution of the data (Gaussian) are false, and that other models are necessary. For instance, the em algorithm has been derived for the von Mises-Fisher model, described in Section 5.3 [3].

A problem with the em algorithm is that its runtime complexity is quadratic in the number of clusters k, or $\mathcal{O}(k^2n)$ (since likelihoods are recalculated for every cluster). Other practical difficulties include instability as well as stochastic initialization (recall that $k$means also suffers from this issue). A more detailed investigation of em, including an exhaustive runtime analysis, is presented in e.g. [34].

5.3 von Mises-Fisher model

The von Mises-Fisher (vMF) distribution is the analogue of the Gaussian distribtion for directional data. The spherical $k$means algorithm (see Section 4), unsurprisingly, has been shown to follow the vMF distribution [2]. There is empirical evidence that this model approximates the distribution of text data better than other models, such as the multinomial or Bernouilli models [59].

Let $\mathbf{d} \in \mathcal{D}$ be a document from the corpus, $\theta \in \mathbf{\Theta}$ a cluster, then the likelihood that $\mathbf{d}$ belongs to θ is defined as

$$\mathcal{P}(\mathbf{d}|\theta) = \frac{1}{Z(\Sigma)} \exp\left(\Sigma \frac{\mathbf{d}^T \mu}{|\mu|}\right) \tag{17}$$

here Z denotes the Bessel function (a normalization factor). This function makes estimation of the Σ parameter difficult in a mixture of vMF distributions, and so instead a deterministic annealing scheme can be used. Simply put, this amounts to starting with a low value of Σ and gradually increasing it.

Alternatively, an em derivation using the vMF model as the underlying distribution can be used, where Σ is estimated in the M-step. Under this model, the em algorithm has been shown to significantly outperform spherical $k$means [3]. This follows from the fact that spherical $k$means amounts to a special case of the em algorithm using the vMF model, where instead of estimating Σ it is assumed constant accross clusters.

5.4 Model-based $k$means

A more constrained (disjoint) version of the em algorithm is model-based $k$means. This algorithm alternates between a model re-estimation step and a sample re-assignment step, resulting in $\mathcal{O}(kn)$ runtime complexity (recall that em has $\mathcal{O}(k^2n)$ complexity).

Despite these advantageous properties, the algorithm doesn't perform much worse than the full em algorithm [58]. The important distinction between soft and disjoint generative algorithms is that model $k$means doesn't fractionally assign each document to every cluster, and doesn't train the models based on posterior probability.

6 Spectral Clustering

A matrix is a natural representation for adjacency information between vertices, and therefore the vector model can be interpreted as a graph. Spectral clustering involves finding cuts in this graph that produce good clusters. This is illustrated for the special case of a bipartite graph in Figure 5.

The problem then becomes that of finding good cuts in the graph, which has resulted in a number of criterion functions that spectral clustering algorithms aim to optimize. These include multiway *ratio cut*, *normalized cut*, and *min-max cut*. On graphs where the clusters are well separated, all tend to perform similarly. However, when there is considerable overlap between clusters, as is typical of text data, min-max cuts can perform better [13]. An explanation for this behavior is that the min-max objective function favors balanced clusters, and is resistant to the skew resulting from overlapping clusters.

A popular graph-based document clustering package is CLUTO[8], available in an open-source fashion and free of charge, which uses a sophisticated coarsening and uncoarsening scheme [22].

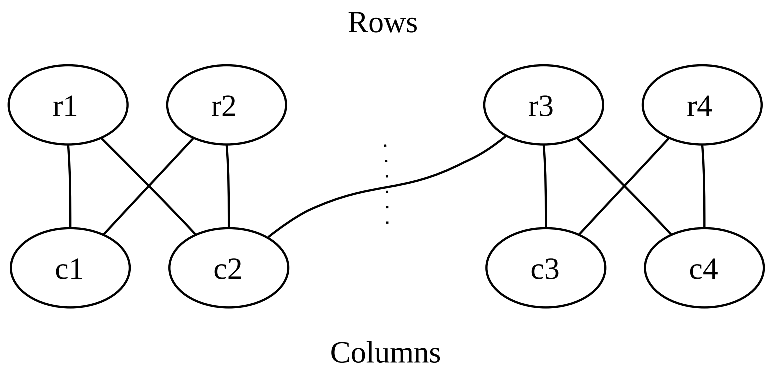


Figure 5: Bipartite graph partitioning. The dotted line represents a cut that results in *coclusters* of the rows and colums of the term-document matrix.

A special case of graph-based clustering, that has enjoyed much recent interest, constructs a bipartite graph of the rows and columns of the input data. The underlying assumption behind coclustering is that words which occur together are associated with similar concepts, and so it not just groups of similar documents that are important, but also groups of similar words. This representation is illustrated in Figure 5.

Cuts in this bipartite graph produce *coclusters* of words (rows) and documents (columns). It has been shown that optimizing these cuts is an equivalent problem to computing the singular value decomposition of the original matrix [7]. Singular value decomposition, often referred to as principal component analysis, is a well established dimensionality reduction technique from linear algebra, and it is described in Section 7.1.

6.1 Divide & merge clustering

The notion of conductance, which measures "how tightly nit" a graph is, has been proposed as a criterion for finding optimal cuts in a graph [21]. Optimizing such cuts is NP-complete, however an approximate recursive algorithm has been proposed that runs in $\mathcal{O}(n \log n)$.

The divide and merge clustering algorithm[9] proceeds in two phases [5]. First, a hierachical clustering is produced by recursive cuts on the graph resulting from the term-document matrix. It is well established that partitions can be derived from the second eigenvector of the similarity matrix. This second eigenvector can be approximated using the power method, which is an iterative algorithm whose benefit is avoiding expensive matrix-matrix computations. From the second eigenvector, a conductance minimizing cut can be found, and this procedure is repeated on the two submatrices resulting from the cut.

The second phase finds a tree-respecting clustering from the output of the divide phase. A variety of objective functions (e.g. $k$means) can be used for this phase, however correlation clustering [45] has the advantage of not depending on a predefined number of clusters (using a "correlation" threshold instead). This is a significant advantage in the case of post-retrieval applications, such as when querying a web search engine, where the true number of clusters is not known a priori.

[8]Available at http://glaros.dtc.umn.edu/gkhome/views/cluto.
[9]Try it out at http://eigencluster.csail.mit.edu/.

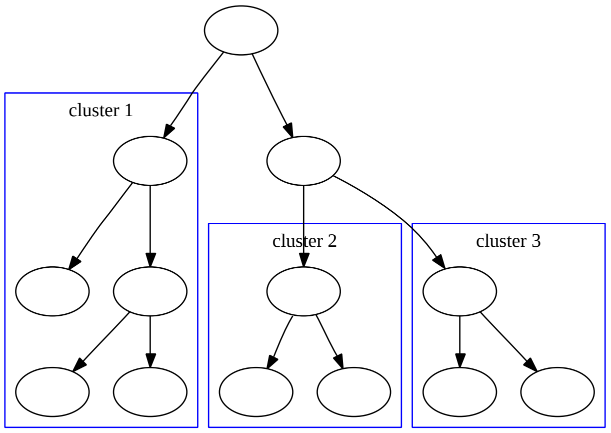


Figure 6: Divide and merge: a hierarchy is computed by repeated cuts, and subtrees are grouped using correlation clustering.

6.2 Fuzzy coclustering

Spectral projection has been the basis for a number of algorithms. Of particular interest are those that generate fuzzy clusters, since intuitively documents can span multiple topics. While these fuzzy relationships can be captured by regular fuzzy clustering algorithms that capture the *degree* to which documents belong to each cluster, fuzzy *co*clustering implicitely assigns degrees of membership to words as well.

The difference between fuzzy coclustering and regular coclustering is that the boundaries between clusters in the former case are *fuzzified* in accordance with certain membership functions.

Like the fuzzy *c*means algorithm, the coclustering variant optimizes a fuzzy objective function. The important difference is the notion of aggregation, in which the algorithm optimizes for coclusters of terms and documents rather than clusters of documents alone. Aggregation can be written as

$$\sum_{c=1}^{C}\sum_{i=1}^{N}\sum_{j=1}^{K} u_{ci} v_{cj} d_{ij} \tag{18}$$

An example of such an algorithm, Fuzzy Codok [24], updates as follows. Let u_{ci} denote membership of documents, v_{cj} denote membership of words, and d_{ij} denote the degree of correlation between documents and words. We assume an $m \times n$ term-document matrix, where there are C clusters of documents and K clusters of terms. Then the algorithm updates according to:

$$\mu_{ci} = \frac{1}{C} + \frac{1}{2T_u}\left(\sum_{j=1}^{m} v_{cj} d_{ij} - \frac{1}{C}\sum_{j=1}^{m} v_{cj} d_{ij}\right) \tag{19}$$

$$v_{cj} = \frac{1}{K} + \frac{1}{2T_v}\left(\sum_{i=1}^{n} u_{ci} d_{ij} - \frac{1}{K}\sum_{i=1}^{n} u_{ci} d_{ij}\right) \tag{20}$$

An empirical evaluation has shown that fuzzy coclustering algorithms outperform a modern variant of fuzzy *c*means [29] (which uses cosine similarity instead of Euclidian distance) on certain datasets [46], while performing as well in others. Another advantage of this method is that word memberships result naturally from the clustering, since terms are clustered simultaneously as documents.

A weakness of fuzzy partitional algorithms in general is that the *fuzzifying* parameters must be set manually, e.g. "setting $T_u = 1.5$ seemed to produce better results." Furthermore, optimal values can vary considerably based on the corpus. As such, while possibly appropriate for static collections, it is hard to tune the parameters without knowing the distribution of the data in advance. This suggests that statistical models might be able to better describe the data, assuming an underlying probability distribution. An algorithm that iteratively estimates the distribution of the data is described in Section 5.2.

The difference between the Fuzzy Codok algorithm and that of Section 7.3 should be pointed out. On face value, both generate fuzzy memberships for documents and terms. However, while Fuzzy Codok employs a notion of aggregation in the original feature space, the latter employs PCA techniques, described next, and works in a reduced space resulting from matrix factorization.

7 Dimensionality Reduction

While proprocessing can achieve significant reduction in the size of the vector space, post-retrieval applications call for higher efficiency. This section describes describes two matrix factorization techniques that have been shown to not only significantly reduce the size of document vectors (by several orders of magnitude), but also to increase clustering accuracy. In fact, these methods can be viewed as clustering methods themselves (although post-processing is required in the case of spectral coclustering, see Section 7.1).

The remainder of this section assumes a term-document matrix $\mathbf{A} \in \mathbb{R}_{+}^{m \times n}$. The goal of dimensionality reduction techniques is to produce a rank k approximation of $\mathbf{A}$, $\mathbf{A}_k$, while introducing managable error. A common measure of the quality of this approximation is the Frobenius norm, which is defined as

$$|\mathbf{A} - \mathbf{A}_k| = \sqrt{\sum_{a \in \mathbf{A}} \sum_{a_k \in \mathbf{A_k}} (a - a_k)^2} \tag{21}$$

The smaller the Frobenius norm, the better the matrix $\mathbf{A}_k$ approximates the original matrix $\mathbf{A}$.

7.1 Principal component analysis

Principal components are orthogonal (i.e. uncorrelated) projections that together explain the maximum amount of variation in a dataset. In practice, principle components can be found by computing the singular value decomposition on the correlation matrix of the dataset. This method is sometimes called spectral projection. This is illustrated in Figure 7.

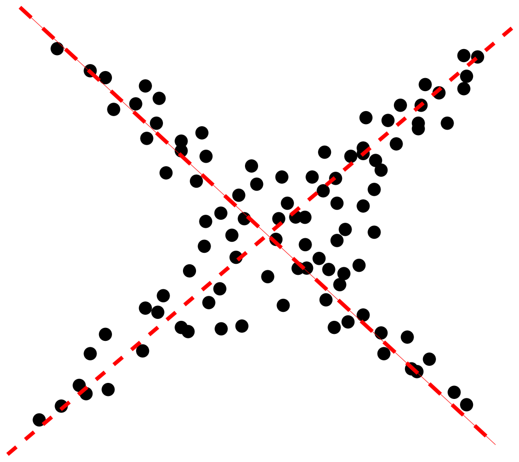


Figure 7: The two dashed lines represent principal components capturing the variability in the dataset.

The singular value decomposition of the original matrix $\mathbf{A}$ involves breaking it up into the matrices:

$$\mathbf{A_n} \approx \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \tag{22}$$

where $\mathbf{\Sigma}$ is a diagonal matrix and $\mathbf{U}$ and $\mathbf{V}$ are orthonomal matrices. The spectral projection of $\mathbf{A}$ of rank k is written $\mathbf{A}_k = \mathbf{U}\mathbf{\Sigma}_k\mathbf{V}^T$, where $\mathbf{\Sigma}_k$ is a diagonal matrix with the first k singular values down the diagonal. The higher the rank, the closer the approximation is to the original matrix.

The base vectors of the matrix $\mathbf{U}$ are ordered according to their significance to reconstructing the original term-document matrix. This information has been exploited for estimating the optimal number of clusters [53].

It has been observed that PCA has two important properties that make it appropriate for clustering: approximation and distinguishability [47]. Approximation states that PCA introduces manageable error as dimensionality is reduced, i.e. that the approximation is optimal. It has been found that on text data the percentage error was about 60% when the dimensionality was reduced from 400 to 128. The second property, distinguishability, is more critical. Experiments have shown that PCA increases the ratio between intra-cluster similarity and inter-cluster similarity. In other words, this means that in the reduced space clusters are more apparent, which is not surprising since, as described in Section 6, PCA is a relaxed solution to the problem of finding optimal cuts in the bipartite graph of rows and columns.

A problem with PCA is that the resulting approximation contains negative values, and so the reduced space cannot be directly interpreted as a clustering. Section 7.2 describes an alternative method which produces a readily interpretable reduced space. Alternatively, clustering (by e.g. $k$means) can be performed in the reduced space to produce final clusters, which (as per the properties described above) actually produces a clustering more accurate than if the algorithm was run in the original vector space. In fact, it can further be proved that the principal components can be interpreted as solutions of the $k$means objective function [10].

Another problem is that the principal components are constrained to be orthogonal. This is problematic for text data, as documents might span multiple topics, and so in this case the underlying semantic variables will not be orthogonal [15]. Also, the computation of the SVD is expensive, and cannot be performed iteratively. That is, unlike algorithms such as $k$means, the computation of the singular value decomposition is not an optimization procedure and does not produce intermediary results.

7.2 Nonnegative matrix factorization

Nonnegative Matrix Factorization (NMF) [26], originally developed for computer vision applications, has been effectively used for document clustering [49]. Unlike spectral decomposition methods, the resulting approximation contains only non-negative factors, which means that a readily interpretable clustering can be found from the reduced space without need for further post-processing. The technique breaks the original term-document matrix $\mathbf{A}$ into the matrices

$$\mathbf{A} \approx \mathbf{U}\mathbf{V}^T \tag{23}$$

where $\mathbf{U}$ is a set of base vectors of size $m \times k$, and V is an $n \times k$ coefficient matrix. The base vectors in $\mathbf{U}$ can be interpreted as a set of semantic variables corresponding to topics in the corpus, while $\mathbf{V}$ describes the contribution of the documents to these topics. Tthe matrices $\mathbf{U}$ and $\mathbf{V}$ are randomly initialized, and their contents iteratively estimated using the expectation maximization algorithm (described in Section 5.2).

The objective function that is optimized is usually Euclidian distance (which is minimized) between each column of $\mathbf{A}$ and its corresponding approximation in $\mathbf{U}\mathbf{V}^T$. This can be written

$$\mathbf{\Theta} = \sum_i \sum_j \left(\mathbf{A}_{ij} - \sum_l \mathbf{U}_{il}\mathbf{V}_{jl} \right) \tag{24}$$

A problem with NMF is that it relies on random initialization. As a result, the same data might produce different results across runs. An alternative strategy is to use the centroids produced from the spherical $k$means algorithm as seeds for NMF [48]. It is interesting to observe that in this case the post-processing required for PCA (recall that the reduced space can contain negative factors) is traded for pre-processing. That said, NMF is typically faster than computing the SVD of the entire term-document matrix.

There has been some recent interest on the equivalence between nonnegative matrix factorization and spectral clustering [11], and with probabilistic latent sematic indexing [12].

7.3 Soft spectral coclustering

A hurdle for generating fuzzy document memberships in reduced space is that the truncation of the matrix (e.g. selecting the k highest principal components) introduces a distortion. An alternative method is to induce membership weights from the disjoint partitions of words and documents. That is, a soft partition of documents can be induced from a partition of terms, and a soft partition of terms can be be induced from a partition of documents [15].

From a reduced space $\mathbf{Z}$ resulting from PCA, a clustering is generated by applying $k$means (i.e. spectral coclustering). Let $\mathbf{P} \in \mathbb{R}^{(m+n)\times k}$ be the partition matrix, written as

$$\mathbf{P} = \begin{bmatrix} \mathbf{P_1} \\ \mathbf{P_2} \end{bmatrix}$$

where $\mathbf{P_1}$ indicates the assignments of terms to clusters and $\mathbf{P_2}$ indicates the assignment of documents to clusters. Term weights can be produced from the transformation $\mathbf{A\hat{P}_2}$ (the hat denotes normalization of the columns to unit length), which projects the centroids of the partition in reduced space to the original feature space. Document weights are derived similarly from the transformation $\mathbf{A\hat{P}_1}$.

Another matrix $\mathbf{S}$ stores a projection of the centroid-similarity values from the embedded clustering to the original data, which can be similarly divided as

$$\mathbf{S} = \begin{bmatrix} \mathbf{S_1} \\ \mathbf{S_2} \end{bmatrix}$$

where $\mathbf{S_1} \in \mathbb{R}^{m\times k}$ is a term-centroid similarity matrix and $\mathbf{S_2} \in \mathbb{R}^{n\times k}$ is a document-centroid similarity matrix. The projections $\mathbf{AS_2}$ and $\mathbf{A^TS_1}$ generate memberships weights that account for both the frequency values in the original dataset as well as the similarity of points in the reduced space $\mathbf{Z}$.

Since similarity values (i.e. fuzzy memberships) result in more accurate clusters than binary memberships, the projection $\mathbf{A^TS_1}$ is used as a document membership function, while $\mathbf{A\hat{P}_2}$ is used for term membership (it was observed that these terms were more descriptive of the cluster contents). The resulting algorithm performs marginally better than coclustering on well-separated corpora, but can show significant improvements when the dataset contains overlapping topics.

It was observed in Section 7.2 that the stochastic initialization for NMF can be problematic, in that it can cause the algorithm to converge to suboptimal local minima. As a solution, the cluster centroids resulting from the soft spectral projection algorithm outlined above can be used as seeds for NMF [15]. This "refined" soft spectral clustering improves over soft spectral clustering in all cases, sometimes to the order of 40%. However, there is of course a runtime cost associated with the conjunction of these two approaches.

7.4 Lingo

Lingo [36] follows a "description-comes-first" approach to clustering. The first step of the algorithm, dubbed cluster label induction, uses dimensionality reduction techniques on the term-document matrix and from the base vectors of the reduced spaces produces cluster labels. Typical measures such as cosine similarity can then be used to find the word or phrase which best approximates the cluster. The second step treats each cluster label as a query in the traditional IR sense, and the retrieved documents (under the vector model) are added to these clusters.

For the cluster label induction phase, NMF was found to produce better clusters than PCA methods [35]. In terms of accuracy, Lingo has outperformed suffix tree clustering [52] on collections from the Open Directory Project[10]. A runtime analysis [35] shows that clustering 500 snippets can take between 1 and 3 seconds, depending on quality settings of the algorithm.

[10] A large human-edited directory of the web, see http://dmoz.org/.

8 Phrase-Based Models

Consider the phrases[11] “the dog chased a cat” and “the cat chased a dog.” Although their vector representation is identical, $d = \{chase, cat, dog\}$, the meaning is obviously different. It has been hypothesized that clustering accuracy can be improved by encoding word order information, resulting in a number of alternative document models.

The generation of legible cluster labels is another important motivation for phrase-based models. Models discussed previously describe each cluster with it its most significant terms (for example, the three most significant). Phrase-based models can naturally describe clusters by phrases, and it is generally agreed that these are more descriptive of the cluster contents.

8.1 Suffix tree clustering

Suffix tree clustering (STC) was originally developed for web snippet clustering [52], and uses the tokens output from the preprocessing phase to construct a suffix tree structure. Figure 8 illustrates the construction of a suffix tree.

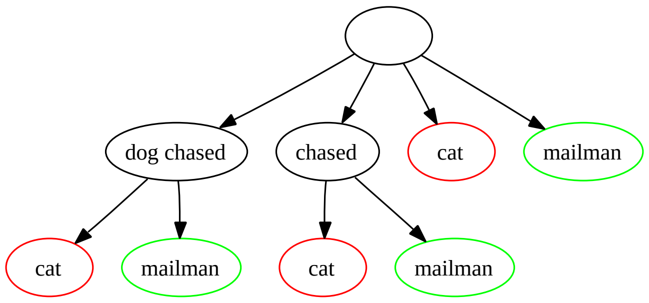


Figure 8: The suffix tree resulting from the phrases “dog chased cat” (red) and “dog chased mailman” (green).

The tree structure represents shared suffixes between documents. The algorithm uses these shared suffixes to identify base clusters of documents. The base clusters are then combined into final clusters using a connected-component graph algorithm. The entire suffix tree clustering algorithm is reported as having $\mathcal{O}(n \log(n))$ runtime.

[30] presents an analysis of the STC heuristic. The most important properties can be summarized thus:

- A clustering might not include all documents in the corpus; for example, if documents share only very short sentences with the rest of the collection.
- The clustering depends highly on document frequency, potentially leading to large clusters.
- Only suffix matches are considered, without considering suffix mismatches.

While the original publication on the suffix tree model presented promising results clustering web snippets, two recent evaluations of STC have shown that is does not scale well to larger corpora or longer documents. On subsets of the Reuters corpus, STC performed about 30% worse than hierarchical or graph-based approaches, with an average F-measure of 0.44 [30]. On subsets of the MEDLINE biomedical digital library, STC performed better than hierarchical approaches (but as corpus size increased, its performance decreased to similar levels), and consistently worse than partitional approaches [50].

These results show that word order matches alone are not sufficient to produce good clusters, which raises the question of if both word order and traditional vector space similarity measures can be combined

[11]In this context a phrase means a sequence of words, and doesn’t imply any grammatical structure.

to improve perfomance. In [30], a measure to quantify the similarity of two documents under the suffix tree model is introduced:

$$\varphi\text{ST} = \frac{|E^+ \cup E^-|}{|E^+ \cap E^-|} \tag{25}$$

In words, the similarity of two documents is the ratio of the similar and dissimilar edges in their mutual suffix tree. Suffix tree matches (φST) are combined with cosine similarity (φ_{cos}) using a weighted average $\varphi\text{HYB} = \lambda \cdot \varphi\text{ST} + (1 - \lambda) \cdot \varphi_{cos}$. Under this hybrid similarity measure, both graph-based and hierarchical clustering algorithms showed average improvements of about 20%. It should be noted that values of λ had to be tuned manually for each algorithm.

8.2 Document index graph

The Document Index Graph (DIG) proposed in [18] is similar in spirit to the suffix tree model in that it encodes word order information, and defines similarity based on matches in word order.

Put simply, the DIG represents each word as a vertex in a directed multigraph. The edges of the graph represent sequences of words. Each vertex maintains a table of the documents the word appears in, as well as sentence information by recording the edge along which each sentence continues (creating an inverted list of documents). If a phrase appears more than once in a document, then the frequency count of the corresponding vertices in the graph are increased accordingly. An example graph is shown in Figure 9.

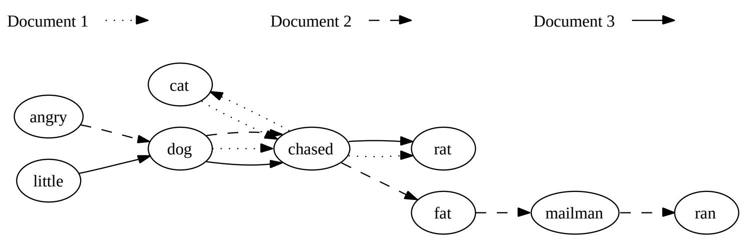


Figure 9: Document 1 consists of the phrases "cat chased rat" and "dog chased cat." Document 2 consists of the phrases "angry dog chased fat mailman" and "mailman ran." Document 3 consists of the phrase "little dog chased rat." Phrases are apparent in the graph by following the directed edges.

The main differences between the DIG and the suffix model is that the DIG stores words explicitely as vertices, and maintains frequency information at each vertex, thereby avoiding storing redundant information (as seen before, suffixes are not unique in STC). It is also important to note that unlike STC, DIG is not a clustering algorithm, merely a document model that stores word order information explicitely as directed edges. Based on this model, document similarity based on overlapping subgraphs can be computed to obtain a similarity matrix. This similarity matrix can then be used in any discriminative clustering algorithm. In [18], they use a partitional algorithm that tries to maintain cluster distribution using a histogram representation of the clusters.

Both [18] and [30] observe that word order information alone is an insufficient measure of document similarity, and both adopt a weighted average method to combine term frequency and word order similarities. The DIG model was weighted between 70% and 80%, while the suffix tree similarity measure was weighted between 30% and 50%. This discrepancy might indicate that the DIG is a superior measure of word order similarity, or might simply reflect differement properties of the clustering algorithms used.

The important point is that word order information together with word frequency can improve clustering accuracy, and in this respect both studies report improvements of about 20% using hybrid measures. It should be noted, however, that there is a certain cost associated with the use of two measures of similarity. If a similarity matrix can be precomputed, then hybrid measures are equivalent to traditional measures in terms of efficiency. On the other hand, if documents are retrieved online, then the construction of a DIG and, to a lesser extent, a suffix tree, is expensive.

9 Comparative Analysis

This section summarizes important properties of the algorithms described. In deciding which techniques are appropriate for query clustering and which for clustering static collections, the following is assumed:

- The number of classes is unknown in query clustering, whereas some idea of the number of classes is generally known when clustering static collections.
- In terms of efficiency, query clustering operates on smaller datasets than collection clustering; for instance, the first few hundred documents that most accurately match the query are typically presented to the user versus thousands of documents for static collections.
- Query clustering aims for a balance between accuracy and efficiency, while collection clustering is performed offline (i.e. accuracy is the prime concern).

In the complexity column, n denotes the number of documents in the corpus and k denotes the number of clusters. Big-Oh's are as reported in related publications, and were not re-derived as part of this work. It should be noted that there can be inconsistencies in the literature due to what is considered preprocessing and what is not. In particular, the affinity (or similarity) matrix is assumed to be stored in main memory, i.e. precomputed.

The "fuzzy" column reports if the algorithm outputs cluster membership weights or simply membership (in cluster or not in cluster). The "finds k" column specifies if the algorithm requires an estimate of the number of clusters from the user (a priori) or is able to produce such an estimate itself. This has also been called model selection. Finally, the "open" column reports if a freely available, open-source implementation of the algorithm could be found.

Finally, the algorithms are ranked according to how they are expected to perform, the higher the better. That is, on a given collection, algorithms ranked higher are expected to produce more accurate clusters than those ranked lower.

Algorithm	**Section**	**Complexity**	**Fuzzy**	**Finds k**	**Open**	**Online**
Kernel $k$means	4.2	-	No	No	No	No
RSSC	7.3	-	Yes	No	No	No
NMF	7.2	$\mathcal{O}(k^2n)$	No	No	No	No
EM vMF	5.2	$\mathcal{O}(k^2n)$	Yes	No	Yes (Java)	No
Fuzzy codok	6.2	-	Yes	No	No	No
Eigencluster	6.1	$\mathcal{O}(kn\log n)$	No	Yes	No	No
Spectral coclustering	7.1	$\mathcal{O}(kn^2)$	No	No	No	No
Online spherical $k$means	4.1	$\mathcal{O}(kn)$	No	No	No	Yes
CLUTO	6	$\mathcal{O}(kn)$	No	No	Yes (C)	No
Lingo	7.4	$\mathcal{O}(kn^2)$	Yes	Yes	Yes (Java)	No
STC	8.1	$\mathcal{O}(n\log n)$	Yes	Yes	Yes (Java)	Yes

9.1 Query Clustering

Query clustering refers to the grouping documents returned from an IR query, such as a Google search, into clear equivalence classes. An important consideration in this case is the highly variable nature of the set of documents returned. For instance, a query might be very specific, returning documents that all belong to one class, while another might return documents from many different equivalence classes.

This property makes the estimation of k, the number of clusters, difficult. While presenting the user with the option of selecting k is appealing, ideally this decision should be made by the system itself, as in exploratory searches the user has no clear idea of the number of groupings in the data.

The only algorithms considered that are capable of estimating k are STC and Eigencluster. Both are capable of clustering small collections of documents in under a second, which is acceptable for most live query sessions. However, both also suffer from notable drawbacks.

STC has been shown not to scale well as the length of documents or the size of the collection increase. In terms of accuracy, STC performs worse than the traditional $k$means. As discussed in Section 8, this accuracy can be increased under a hybrid measure of similarity, but in this case model selection capabilities are lost (i.e. finding k). The STC algorithm could also be run independantly to estimate the number of the clusters, but this estimate would suffer from the accuracy issues described above as well.

Eigencluster, on the other hand, has been reported as having accuracy comparable to PCA methods (i.e. coclustering). On the other hand, it does not generate overlapping clusters, and therefore cannot account for documents spanning multiple topics. An interesting direction for future research would be to consider fuzzy algorithms for the merge phase of eigencluster.

For these reasons, the clustering of documents returned from IR queries can be considered to a large extent an open research area. To reiterate, the problems are model selection, accuracy, and the generation of fuzzy memberships.

9.2 Collection Clustering

Collection clustering refers to the grouping of documents in a static collection. This is an easier problem than query clustering, as the model selection and initialization problems can both be addressed by brute force strategies. This amounts to, simply put, "seeing what works best."

For instance, kernel clustering has shown superior accuracy for some collections, but the choice of kernel function is not always obvious, and can depend on the collection. If the collection is static, an appropriate kernel function can be found (tuned). Similarly, by inspecting the output of the clustering algorithm, various values of k can be tried. Of course, this assumes that inspecting the output of the clustering is feasible. In the case of very large collections, it might not be obvious what constitutes a good choice of k, in which case model selection capabilities are desirable.

Assuming inspection is feasible, the most accurate algorithms are easy candidates for clustering static collections, namely kernel $k$means (its fuzzy variant) and RSSC. An open question is how these compare to generative algorithms (e.g. em), which theoretically should provide more robust fuzzy memberships, but lack the higher dimensional mapping (kernel) or coclustering capabilities (RSSC) of the algorithms mentioned.

10 Conclusions

It should be observed that even though the algorithms described in this work can produce good clusterings according to typical measures, the most "natural" grouping is not always produced. A simple explanation is that, for a given corpus, there are many correct ways to arrange the documents. In fact, it has been noted that clustering is ultimately in the eye of the beholder [14].

If Google is queried with the term "clustering," a grouping based on similarity alone will not produce intuitive results[12]. This is because the results will be highly similar to begin with, since clustering refers to a specific concept. On the other hand, vague (i.e. multi-aspect) queries such as "mickey" will return a number of clear equivalence classes (baseball, disney, and so on).

The example described above hints at the problem of estimating the number of clusters when it is not known a priori. Most algorithms described in this work assume a given number of clusters (i.e. supplied by the user), with the notable exception of hierarchical algorithms such as described in 6.1. In practice, the true number of groupings is often unknown, especially in the important special case of query clustering, where there might not be clear, separable, equivalence classes.

Some more sophisticated approaches do exist, although tend to be quite computationally expensive. For example, one algorithm applies an overfitted $k$means algorithm (that is, overestimates k) to produce a reduced

[12]For example, try `http://eigencluster.csail.mit.edu//cgi-bin/main.cgi?query=clustering`.

set of cluster representatives on which to conduct an agglomerative clustering [38]. From a dendogram, the elbow criterion [1] (a rule of thumb) can be used to select a number of clusters.

Another open problem is the choice of initialization strategy for most $k$means or em based algorithms. Recall that $k$means, in its classical formulation, randomly picks k documents as cluster centroids. Similarly, generative algorithms must pick initial cluster means and covarience matrices from which to iteratively optimize likelihood. As a result, both approaches risk getting stuck at suboptimal local optima if these initial values are not chosen wisely. For instance, if one of the initial documents picked as a cluster centroid is an outlier, it will be dissimilar to all other documents, resulting in an empty cluster. A potential solution would be to use spectral methods, which are not susceptible to suboptimal local minima, to initialize the centroids [8].

The choice of cluster labels is another issue that remains largely unaddressed in the literature. In most cases, the most significant terms—those closest to the cluster centroid—are listed. While in some cases this is acceptable, it is frequently the case these terms all belong to the same noun phrase, such as the name of a person. This suggests that a linguistically inspired approach might be more appropriate, or one based on word order [19].

The combination of word order and term frequency was shown to improve clustering accuracy in Section 8, and the addition of synonyms to the document vector similarity improved accuracy (Section 3). An interesting direction for future research would be to investigate other measures of similarity not captured by the vector model, such as matches in mathematical formula.

Section 9 summarizes important properties of the algorithms cited in this work. While only algorithms that have succesfully been applied to the task of document clustering were considered, some others such as DCA [25] applied for image segmentation might have potential.

References

[1] M.S. Aldenderfer and R.K. Blashfield. *Cluster Analysis.* Newbury Park (CA): Sage, 1984.

[2] A. Banerjee and J. Ghosh. Frequency sensitive competitive learning for clustering on high-dimensional hyperspheres. In *IEEE international joint conference on neural networks*, pages 1590–1595, Honolulu, Hawaii, May 2002.

[3] Arindam Banerjee, Inderjit S. Dhillon, Joydeep Ghosh, and Suvrit Sra. Clustering on the unit hypersphere using von mises-fisher distributions. *J. Mach. Learn. Res.*, 6:1345–1382, 2005.

[4] J. Bilmes. A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical report, University of California at Berkeley, 1997.

[5] David Cheng, Santosh Vempala, Ravi Kannan, and Grant Wang. A divide-and-merge methodology for clustering. In *PODS '05: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems*, pages 196–205, New York, NY, USA, 2005. ACM Press.

[6] Douglass R. Cutting, David R. Karger, Jan O. Pedersen, and John W. Tukey. Scatter/gather: a cluster-based approach to browsing large document collections. In *SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on research and development in information retrieval*, pages 318–329, New York, NY, USA, 1992. ACM Press.

[7] Inderjit S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining*, pages 269–274, New York, NY, USA, 2001. ACM Press.

[8] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 551–556, New York, NY, USA, 2004. ACM Press.

[9] Inderjit S. Dhillon and Dharmendra S. Modha. Concept decompositions for large sparse text data using clustering. *Mach. Learn.*, 42(1-2):143–175, 2001.

[10] Chris Ding and Xiaofeng He. K-means clustering via principal component analysis. In *ICML '04: Proceedings of the twenty-first international conference on machine learning*, page 29, New York, NY, USA, 2004. ACM Press.

[11] Chris Ding, Xioafeng He, and Horst D. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proc. SIAM international conference on data mining*, pages 606–610, 2005.

[12] Chris Ding, Tao Li, and Wei Peng. Nmf and plsi: equivalence and a hybrid algorithm. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval*, pages 641–642, New York, NY, USA, 2006. ACM Press.

[13] Chris H. Q. Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 107–114, Washington, DC, USA, 2001. IEEE Computer Society.

[14] Vladimir Estivill-Castro. Why so many clustering algorithms: a position paper. *SIGKDD Explor. Newsl.*, 4(1):65–75, 2002.

[15] Derek Greene and Padraig Cunningham. Producing accurate interpretable clusters from high-dimensional data. In *9th European conference on principles and practice of knowledge discovery in databases*, volume 3721, pages 486–494. University of Dublin, Trinity College, Dublin, October 2005.

[16] Derek Greene and Padraig Cunningham. Practical solutions to the problem of diagonal dominance in kernel document clustering. In *ICML '06: Proceedings of the 23rd international conference on machine learning*, pages 377–384, New York, NY, USA, 2006. ACM Press.

[17] Greg Hamerly and Charles Elkan. Alternatives to the k-means algorithm that find better clusterings. In *CIKM '02: Proceedings of the eleventh international conference on information and knowledge management*, pages 600–607, New York, NY, USA, 2002. ACM Press.

[18] K. M. Hammouda and M. S. Kamel. Efficient phrase-based document indexing for web document clustering. *IEEE Transactions on knowledge and data engineering*, 16(10):1279–1296, 2004.

[19] Khaled M. Hammouda, Diego N. Matute, and Mohamed S. Kamel. Corephrase: Keyphrase extraction for document clustering. *Machine Learning and Data Mining in Pattern Recognition*, pages 265–274, 2005.

[20] David A. Hull. Stemming algorithms: a case study for detailed evaluation. *J. Am. Soc. Inf. Sci.*, 47(1):70–84, 1996.

[21] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, May 2004.

[22] George Karypis and Vipin Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on supercomputing (CDROM)*, pages 1–13, Washington, DC, USA, 1998. IEEE Computer Society.

[23] Dae-Won Kim, Ki Young Lee, Doheon Lee, and Kwang Hyung Lee. Evaluation of the performance of clustering algorithms in kernel-induced space. *Pattern Recognition*, 38:607–611, 2005.

[24] K. Kummamuru, A. Dhawale, and R. Krishnapuram. Fuzzy co-clustering of documents and keywords. In *FUZZ '03: 12th IEEE international conference on fuzzy systems*, pages 772–777, 2003.

[25] Fernando De la Torre and Takeo Kanade. Discriminative cluster analysis. In *ICML '06: Proceedings of the 23rd international conference on machine learning*, pages 241–248, New York, NY, USA, 2006. ACM Press.

[26] Daniel D. Lee and Sebastian H. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, October 1999.

[27] Xin Liu, Yihong Gong, Wei Xu, and Shenghuo Zhu. Document clustering with cluster refinement and model selection capabilities. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on research and development in information retrieval*, pages 191–198. ACM Press, 2002.

[28] Geoffrey McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions.* John Wiley & Sons, 1996.

[29] M.E.S Mendes and L Sacks. Evaluating fuzzy clustering for relevance-based access. In *IEEE International Conference on Fuzzy Systems*, pages 648–653, May 2003.

[30] Sven Meyer zu Eißen, Benno Stein, and Martin Potthast. The Suffix Tree Document Model Revisited. In Klaus Tochtermann and Hermann Maurer, editors, *Proceedings of the 5th international conference on knowledge management (I-KNOW 05), Graz, Austria*, Journal of Universal Computer Science, pages 596–603. Know-Center, July 2005.

[31] George A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.

[32] Alessandro Moschitti and Roberto Basili. Complex linguistic features for text classification: A comprehensive study. In *ECIR '04: 27th European conference on IR research*, pages 181–196, Sunderland, UK, April 2004.

[33] R. Neal and G. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan, editor, *Learning in Graphical Models.* Kluwer, 1998.

[34] Carlos Ordonez and Edward Omiecinski. Frem: fast and robust em clustering for large data sets. In *CIKM '02: Proceedings of the eleventh international conference on information and knowledge management*, pages 590–599, New York, NY, USA, 2002. ACM Press.

[35] Stanislaw Osiński. Dimensionality reduction techniques for search results clustering. Master's thesis, The University of Sheffield, UK, 2004.

[36] Stanislaw Osinski, Jerzy Stefanowski, and Dawid Weiss. Lingo: Search results clustering algorithm based on singular value decomposition. In Mieczyslaw A. Klopotek, Slawomir T. Wierzchon, and Krzysztof Trojanowski, editors, *Intelligent Information Systems*, Advances in Soft Computing, pages 359–368. Springer, 2004.

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

[38] K. Punera and J. Ghosh. Clump: A scalable and robust framework for structure discovery. In *Fifth IEEE International Conference on Data Mining (ICDM)*, pages 757–760, November 2005.

[39] V Roth, M Braun, T Lange, and J Buhmann. A resampling approach to cluster validation. In *Proceedings of the 15th Symposium in Computational Statistics*, 2002.

[40] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975.

[41] Bernhard Scholkopf, Jason Weston, Eleazar Eskin, Christina Leslie, and William Stafford Noble. A kernel approach for learning from almost orthogonal patterns. In *ECML '02: Proceedings of the 13th European conference on machine Learning*, pages 511–528, London, UK, 2002. Springer-Verlag.

[42] A. Hotho S. Staab and G. Stumme. Wordnet improves text document clustering. In *Semantic Web Workshop at SIGIR 2003, 26th annual international ACM SIGIR conference*, July 2003.

[43] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. Technical report, Department of Computer Science and Engineering, University of Minnesota, 2000.

[44] Alexander Strehl and Joydeep Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *Journal on Machine Learning Research (JMLR)*, 3:583–617, December 2002.

[45] Chaitanya Swamy. Correlation clustering: maximizing agreements via semidefinite programming. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on discrete algorithms*, pages 526–527, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[46] William-Chandra Tjhi and Lihui Chen. Fuzzy co-clustering of web documents. In *CW '05: Proceedings of the 2005 International Conference on Cyberworlds*, pages 545–551, Washington, DC, USA, 2005. IEEE Computer Society.

[47] S. Vempala and G. Wang. The benefit of spectral projection for document clustering. In *3rd Workshop on Clustering High Dimensional Data and its Applications, SIAM International Conference on Data Mining*, 2005.

[48] S Wild, J Curry, and A Dougherty. Improving non-negative matrix factorizations through structured initialization. *Pattern Recognition*, 37:2217–2232, 2004.

[49] Wei Xu, Xin Liu, and Yihong Gong. Document clustering based on non-negative matrix factorization. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on research and development in information retrieval*, pages 267–273, New York, NY, USA, 2003. ACM Press.

[50] Illhoi Yoo and Xiaohua Hu. A comprehensive comparison study of document clustering for a biomedical digital library medline. In *JCDL '06: Proceedings of the 6th ACM/IEEE-CS joint conference on digital libraries*, pages 220–229, New York, NY, USA, 2006. ACM Press.

[51] Stella X. Yu and Jianbo Shi. Multiclass spectral clustering. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, Washington, DC, USA, 2003. IEEE Computer Society.

[52] Oren Zamir and Oren Etzioni. Web document clustering: a feasibility demonstration. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval*, pages 46–54, New York, NY, USA, 1998. ACM Press.

[53] Dell Zhang and Yisheng Dong. Semantic, hierarchical, online clustering of web search results. In *6th Asia Pacific Web Conference (APWEB), Hangzhou, China*, pages 69–78, April 2004.

[54] Rong Zhang and Alexander I. Rudnicky. A large scale clustering scheme for kernel k-means. In *ICPR '02: Proceedings of the 16th international conference on pattern recognition*, pages 289–292, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[55] Ying Zhao and George Karypis. Soft clustering criterion functions for partitional document clustering: a summary of results. In *CIKM '04: Proceedings of the thirteenth ACM international conference on information and knowledge management*, pages 246–247, New York, NY, USA, 2004. ACM Press.

[56] Ying Zhao, George Karypis, and Usama Fayyad. Hierarchical clustering algorithms for document datasets. *Data Mining and Knowledge Discovery*, 10(2):141–168, March 2005.

[57] Shi Zhong. Efficient online spherical k-means clustering. In *IEEE International Joint Conference on Neural Networks*, volume 5, pages 3180–3185, 2005.

[58] Shi Zhong and Joydeep Ghosh. Scalable, balanced model-based clustering. In *Proc. 3rd SIAM Int. Conf. Data Mining*, pages 71–82, San Francisco, CA, May 2003.

[59] Shi Zhong and Joydeep Ghosh. Generative model-based document clustering: a comparative study. *Knowl. Inf. Syst.*, 8(3):374–384, 2005.