Incremental Hierarchical Clustering of Text Documents

Nachiketa Sahoo
Heinz School
Carnegie Mellon University
nsahoo@cmu.edu

Jamie Callan
Language Technology Institute
Carnegie Mellon University
callan@cs.cmu.edu

Ramayya Krishnan
Heinz School
Carnegie Mellon University
rk2x@andrew.cmu.edu

George Duncan
Heinz School
Carnegie Mellon University
gd17@andrew.cmu.edu

Rema Padman
Heinz School
Carnegie Mellon University
rpadman@andrew.cmu.edu

ABSTRACT

Incremental hierarchical text document clustering algorithms are important in organizing documents generated from streaming on-line sources, such as, Newswire and Blogs. However, this is a relatively unexplored area in the text document clustering literature. Popular incremental hierarchical clustering algorithms, namely COBWEB and CLASSIT, have not been widely used with text document data. We discuss why, in the current form, these algorithms are not suitable for text clustering and propose an alternative formulation that includes changes to the underlying distributional assumption of the algorithm in order to conform with the data. Both the original CLASSIT algorithm and our proposed algorithm are evaluated using Reuters newswire articles and OHSUMED dataset.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering*; I.5.3 [**Pattern Recognition**]: Clustering; I.2.6 [**Artificial Intelligence**]: Learning—*Concept Learning*

General Terms

Algorithms

Keywords

Incremental Clustering, Hierarchical Clustering, Text Clustering

1. INTRODUCTION

Document clustering is an effective tool to manage information overload. By clustering similar documents together, we can quickly browse large document collections [4], easily grasp the distinct topics and subtopics (concept hierarchies) in them, efficiently query them [12] among many other applications. Hence, it has been widely studied. One survey of existing clustering literature can be found in Jain et al.[9].

Most of the document clustering algorithms studied in the literature operate in a batch mode, i.e., they collect all the documents to be clustered before the start of the exercise and cluster them by making multiple iterations over them. But, this is not always feasible. With the advent of online publishing in the World Wide Web, the number of documents being generated everyday has increased considerably. Popular sources of informational text documents such as Newswire and Blogs generate documents virtually continuously. To organize such documents using existing batch clustering algorithms one might attempt to cluster documents collected up to certain points in time. This would lead to repeated processing of older documents, the large majority of which do not change cluster membership. This would result in redundancy that would make the exercise extremely time consuming, if not impossible, due to the volume of documents being generated. One might be tempted to address this problem by batch clustering periodically collected small batches of documents and then merge the generated clusters. However, there will always be a wait time before a newly generated document can appear in the cluster hierarchy. This delay would be unacceptable in several important scenarios, e.g., financial services, where trading decisions depend on breaking news, and quick access to appropriately classified news documents is important. A clustering algorithm in such a setting needs to process the documents as soon as they arrive. This calls for the use of an *incremental* or *online* clustering algorithm.

Although, incremental text document clustering algorithms has been studied before, most notably as a part of the Topic Detection and Tracking initiative, they produce clusters without any hierarchical relationship among them ([1] and [6]). Thus, they lack certain important utilities of hierarchical clustering. By identifying broad and narrow clusters and describing the relationship between them hierarchical clustering algorithms generate knowledge of topic and subtopic. Another utility of a hierarchy is that often users do not agree on the number of clusters that should be present in the dataset. Depending on the level of granularity they want among the clusters they might want more or fewer clusters. So, one fixed number of clusters made out of the dataset may not satisfy all. But, if we present the user with a cluster hierarchy of documents, which she can browse at her desired level of specificity, we would circumvent the problem of finding the right number of clusters while generating a solution that would satisfy users with different needs.

In spite of the potential benefits of an incremental algorithm, that can cluster text documents as they arrive into an informative cluster hierarchy, this is a relatively less explored area in text document clustering literature. In this work we examine a well known incremental hierarchical clustering algorithm Cobweb that has been used in non-text domain and its variant Classit. We discuss why they are not suitable to be directly applied to text clustering and propose a variant of these algorithm that is based on the properties of text document data—specifically, the word occurrence distribution. Then we evaluate both the algorithms using real world data. By the end of this work we hope to have demonstrated a separation of the attribute distribution and parameter estimation from the control structure of the algorithm. This would enable one to use the existing control structure for incremental hierarchical clustering of various types of data items, while using different attribute distribution assumptions and parameter estimation methods that might be more appropriate for the data at hand.

In Section 2 we briefly review the text clustering literature and Cobweb & Classit algorithms. In Section 3 we describe key properties of text that are central to this work. In Section 4 we explain the contributions of our work. In Section 5 we describe the cluster quality metrics that we have used to evaluate the results obtained. In Section 6 we explain the setup of the experiment and discuss the results. In Section 7 we conclude with scope for future research.

2. LITERATURE REVIEW

Cutting et al. were one of the first to suggest a cluster based approach to browse large document collections[4]. They have suggested two fast partitional clustering algorithms, called "Buckshot" and "Fractionation", which enables one to topically browse large document collections by generating clusters in real time. Clustering algorithms are usually completely partitional or completely agglomerative in nature. However, Zhao and Karypis in a recent work has proposed a hybrid approach in which the agglomeration of documents is constrained by a partitional algorithm. They have shown that such a strategy exploits the strengths of both partitional and agglomerative algorithm and leads to better solutions[16]. In a prior work Zhao and Karypis have compared the performance of different criterion functions for document clustering algorithms[17]. Dittenbach et al. have shown a method called *Growing Hierarchical Self-Organizing Map* to cluster a set of documents into a hierarchy [5]. The algorithm generates clusters in a layered manner starting from the top most layer. The strength of the algorithm is that the width and depth of the cluster tree is adapted to the data. This is done by a clever thresholding of the quantization error at each layer of the tree and at each unit in the layers. Smeaton et al. has proposed a way to carry out fast hierarchical clustering of large document datasets by using a search engine [14]. Top k most similar documents for each document in the dataset are retrieved and similarities are stored. This results in a much smaller, $N \times k$, similarity matrix and leads to fast clustering of the documents. They have also designed a data structure to update this generated cluster efficiently.

Experiments in *incrementally* clustering of text documents have been done as a part of online new event detection task of Topic Detection and Tracking (TDT) initiative ([1] and [6]). In this exercise a decision is taken based on a heuristic whether to group an incoming news document with one of the existing clusters or to create a new cluster of its own. The heuristic makes use of similarity of the document to the existing clusters and the time stamp on the document. The formation of new cluster is used to signal the outbreak of a new event. Outside the TDT initiative, Zhang and Liu has proposed a competitive learning algorithm, which is incremental in nature [15]. The algorithm, called *Self Splitting Competitive Learning*, starts with a prototype vector that is a property of the only cluster present initially. As data points are added, the prototype vector is split and updated to approximate the centroids of the clusters in the dataset. Yet another on-line algorithm called *frequency sensitive competitive learning* has been proposed and evaluated on text datasets by Banerjee and Ghosh[2], which is designed to produce clusters of approximately equal sizes. This is done by making it harder for new data points to join larger clusters.

All of these incremental algorithms produce flat clusters: forgoing the benefits of hierarchical clustering. Also, TDT experiments effectively exploit the information in the time stamp available with news stories, i.e., assumes that news stories that describe the same event will occur within a brief span of time. Such information may not always be available.

Cobweb and Classit

Methods have been proposed in the non-text domain to incrementally cluster items into hierarchies. Notable among them is the Cobweb algorithm by Fisher [7] (assumes nominal attributes) and its derivative Classit [8] (for real valued attributes).

At the heart of Cobweb is a cluster quality measure called Category Utility (CU).

Let $C_1, \ldots, C_K$ be the child clusters of a larger cluster C_p. The Category Utility $CU_p[C_1, \ldots, C_K]$ of $\{C_1, \ldots, C_K\}$ partition of C_p is computed as

$$\frac{1}{K}\sum_{k=1}^{K} P(C_k)\sum_i \sum_j [P(A_i = V_{ij} \mid C_k)^2 - P(A_i = V_{ij} \mid C_p)^2] \quad (1)$$

where, $P(C_k)$ = Probability that a random document belongs to child cluster C_k.

A_i = The ith attribute of the items being clustered

V_{ij} = jth value of the ith attribute

One can interpret Category Utility as the gain in the expected number of correct guesses due to clustering of the data when one predicts the attribute values of an item following a probability-matching guessing strategy, i.e., when $P(A_i = V_{ij})$ fraction of the time attribute A_i is predicted to take value V_{ij}. Cobweb assigns a new item to the child node that maximizes this gain. This is only a local decision since the algorithm processes items one at a time and does not reassign an item to a different cluster. Therefore, Cobweb uses a node *split* and node *merge* operation to reorient the cluster tree if that leads to a higher Category Utility (Figure 1). The Cobweb control structure is shown in Figure 2.

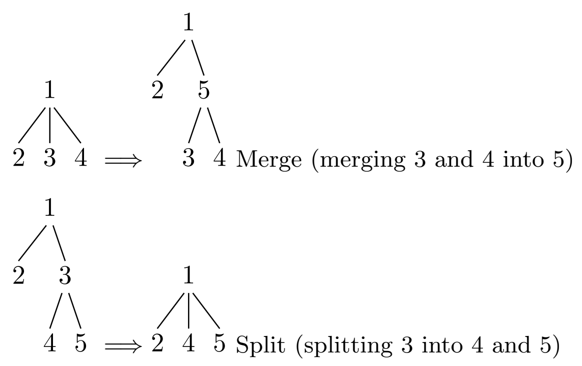

Merge (merging 3 and 4 into 5)
Split (splitting 3 into 4 and 5)

Figure 1: Merge and split operations illustrated.

```
Algorithm (Adapted from Fisher's original work [7])
function Cobweb(item, root)
 Update the attribute value statistics at the root
 If root is a leaf node then
       Return the expanded node that accommodates the new Object
 else
   Find the best child of the root to host the item and perform the
     qualifying step (if any) among the following:

     1. Create a new node for the item instead of adding it to the
        best host, if that leads to improved Category Utility.

     2. Merge the best and second best nodes if it leads to improved
        Category
        Utility and call Cobweb(item, Merged Node)

     3. Split the best node if it leads to improved Category Utility and
        call
        Cobweb(item, root)

   If none of the above steps are performed then
     Call Cobweb(item, best child of root)
   end if
 end if
```

Figure 2: Cobweb control structure.

In a subsequent work, Gennari et al.[8] have shown Category Utility can be computed for real valued attributes if we make some assumption about their distribution. If we assume that the attributes follow Normal distribution, then Category Utility of the partition can be written as

$$CU_p[C_1, \ldots, C_K] = \frac{\sum_k P(C_k) \sum_i \left(\frac{1}{\sigma_{ik}} - \frac{1}{\sigma_{ip}}\right)}{K}$$

where, σ_{in} = standard deviation of the attribute i in node n. This is known as the Classit algorithm.

We have not seen any prior application of either of these algorithms to text clustering. Hence, their performance on text document data is uncertain at the time of this work. Further, word occurrence counts, attributes of text documents that are commonly used to represent a document, are known to follow a skewed distribution—unlike the Normal distribution (Figure 3). Also, Normal distribution assumes that the attributes are Real numbers, but, word occurrence counts are Nonnegative Integers. They can't be treated as nominal attributes either, because the occurrence counts are not contained in a bounded set, which one would have to assume while treating them as nominal attributes. Even when treating occurrence counts as nominal attributes that can take values in a sufficiently large set of integers, say 1 to 1000, we fail to make use of the skewness of the occurrence count. A more suitable distribution for such count data is Negative Binomial, or Katz's distribution [10].

3. TEXT DOCUMENTS AND WORD DISTRIBUTIONS

Text, as we commonly know it, is available in the form of unstructured documents. Before we can cluster such documents, we need to convert them to items with attributes and values. A common way to do this is to use the words[1] in a document as attributes and the number of times the word occurs in the document, or some function of it, as the value of the attribute. This is called the "Bag of Words" approach. One consequence of using such a method to convert documents to an actionable form is that one loses the information contained in the order of the word. Despite this drawback, the bag-of-words approach is one of the most successful and widely used method of converting text documents into actionable form.

Several attempts have been made to characterize the distribution of words across documents. This is useful in judging the information content of a word. For instance a word that occurs uniformly in every document of the corpus, e.g., "the" is not as informative as a word that occurs frequently in only a few, e.g., "Zipf". Occurrence frequency of a word in a document can be used along with the information content of the word to infer the topic of the document and cluster documents of similar topic into same group—as is done in this work. Manning and Schutze have discussed several models to characterize the occurrence of words across different documents [13].

3.1 Models based on Poisson distribution

3.1.1 Poisson

The Poisson distribution has been used to model number of times a word occurs in a document. The probability of a word occurring k times in a document is given by

$$P(k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{2}$$

where, λ is a rate parameter. However, from empirical observations, it has been found that Poisson distribution tends to over estimate the frequency of informative words (content words) [13].

3.1.2 Two Poisson Model

There have been attempts to characterize the occurrence of a word across documents using a mixture of Poisson distributions. One such attempt uses two Poisson distributions to model the probability of a word occurring a certain number of times in a document. One of the distributions captures the rate of the word occurrence when the word occurs because it is topically relevant to the document. The second distribution captures the rate of the word occurrence when the word occurs without being topically relevant to the document. This mixture of two probability distributions has the probability density function:

$$P(k) = \alpha \frac{\lambda_1^k e^{-\lambda_1}}{k!} + (1-\alpha) \frac{\lambda_2^k e^{-\lambda_2}}{k!} \tag{3}$$

[1] "words" or "terms" refer to an indexing unit that is derived from a word after lexical and morphological normalization

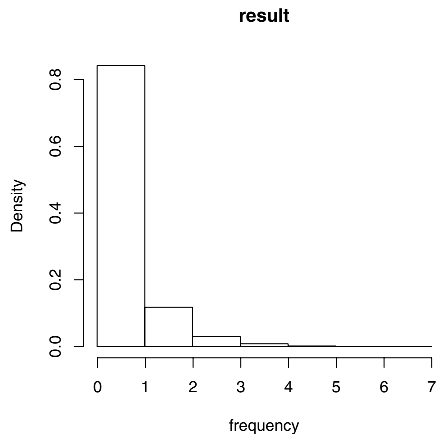


Figure 3: The occurrence of a typical word ("result") across different documents in our test collection.

where, α is the probability of the word being topically relevant.

It has been empirically observed that although, the two Poisson model fits the data better than single Poisson model[3], a spurious drop is seen for the probability of a word occurring twice in a document[10]. The fitted distribution has lower probability for a word occurring twice in a document than it occurring three times, i.e., it predicts that there are fewer documents that contain a word twice than there are documents that contain the same word three times. But, empirically it has been observed that document count monotonically decreases for increasing number of occurrences of a word (see Figure 3).

3.1.3 Negative Binomial

A proposed solution to the above problem is to use a mixture of more than two Poisson distributions to model the word occurrences. A natural extension of this idea is to use a Negative Binomial distribution, which is a gamma mixture of infinite number of Poisson distributions. The probability density functions of a Negative Binomial distribution is given below,

$$P(k) = \binom{k+r-1}{r-1} p^r (1-p)^k, \tag{4}$$

where p and r are parameters of the distributions.

Although, the Negative Binomial distribution fits the word occurrence data very well it can be hard to work with because it often involves computing a large number of coefficients[13]

3.2 Katz's K-mixture model

This simple distribution proposed by Katz[10] has been shown to model the occurrences of words in the documents better than many other distributions such as Poisson and Two Poisson, and about as well as the more complex Negative Binomial distribution[13]. Katz's distribution assigns the following probability to the event that word i occurs k times in a document[2].

$$P(k) = (1-\alpha)\delta_k + \frac{\alpha}{\beta+1}\left(\frac{\beta}{\beta+1}\right)^k \tag{5}$$

$\delta_k = 1 \,\text{iff}\, k = 0 \,\text{and}\, 0 \,\text{otherwise}.$

The MLE estimates of parameters α and β are:

$$\hat{\beta} = \frac{\text{cf} - \text{df}}{\text{df}} \tag{6}$$

$$\hat{\alpha} = \frac{1}{\beta} \times \frac{\text{cf}}{N} \tag{7}$$

cf = *collection frequency* = number of times word i occurred in the document collection.

df = *document frequency* = number of documents in the entire collection that contain the word i.

From (5) it follows that

$$\begin{aligned} P(0) &= 1 - \alpha + \frac{\alpha}{\beta+1} \\ &= 1 - \frac{\text{df}}{N} \\ &= 1 - \Pr(\text{the word occurs in a document}) \\ &= \Pr(\text{the word does not occur in a document}) \end{aligned} \tag{8}$$

Also, it follows that

$$P(k) = \frac{\alpha}{\beta+1}\left(\frac{\beta}{\beta+1}\right)^k, k = 1, 2, \ldots \tag{9}$$

Substituting p for $\frac{\beta}{\beta+1}$, we have

$$P(k) = \alpha(1-p)p^k \tag{10}$$

Let's define a parameter p_0 as

$$p_0 = P(0) \tag{11}$$

using (6) we find that

$$\begin{aligned} p &= \frac{\frac{\text{cf}-\text{df}}{\text{df}}}{\frac{\text{cf}}{\text{df}}} = \frac{\text{cf} - \text{df}}{\text{cf}} \\ &= \Pr(\text{the occurrence is a repeat} \mid \text{occurrence}) \end{aligned} \tag{12}$$

Hence, $1-p$ can be interpreted as the probability of a word occurrence not being a repeat occurrence. Or, it can be thought of as a scaling factor used to make (10) and (11) together a valid probability density function.

We can write Expression (5) for $k = 0$, using p as

$$\begin{aligned} P(0) &= (1-\alpha) + \alpha(1-p) \\ &= 1 - \alpha + \alpha - \alpha p \end{aligned}$$

Hence, α in terms of p_0 and p is

$$\alpha = \frac{1-p_0}{p} \tag{13}$$

Expression (10) can now be written as

$$P(k) = (1-p_0)(1-p)\,p^{k-1} \tag{14}$$

when $k > 0$.

[2]In this section we shall discuss the case of one word, the ith word. Hence, we shall drop the subscript i from the equations and expressions.

Using Expressions (11) and (14), we can fully specify the Katz's distribution. The two parameters are p_0 and p, which can be estimated as

$$\widehat{p_0} = 1 - \frac{\text{df}}{N} \tag{15}$$

and

$$\hat{p} = \frac{\text{cf} - \text{df}}{\text{cf}} \tag{16}$$

It can be shown that if a distribution is defined by Expressions (11) and (14), then the estimates (15) and (16) are the MLE of the parameters p_0 and p (The proof has been omitted due to lack of space).

4. EXTENDING COBWEB FOR TEXT

The COBWEB algorithm can be extended for text document clustering by using distributions that has been suggested in the information retrieval literature for modeling word occurrence distribution. We explored two of them: Negative Binomial distribution and Katz's distribution.

The Category Utility score turned out to be hard to compute for Negative Binomial. It involves sum over a large number of binomial coefficients-squared that does not reduce to any simpler expression. This renders it unsuitable for the purpose of online document clustering. Therefore, we shall fully explore only Katz's distribution based algorithm and compare it with the original CLASSIT algorithm in this work.

Using words as attributes, we can derive the Category Utility function assuming that word occurrences follow Katz's distribution. For reference, the *Category Utility* formula as given in COBWEB is

$$\frac{1}{K}\sum_k P(C_k)\left[\sum_i\sum_j \left(P(A_i = V_{i,j}|C_k)^2 - P(A_i = V_{i,j}|C_p)^2\right)\right]$$

Notice that for each attribute indexed i we need to compute

$$\sum_j \left(P(A_i = V_{i,j}|C_k)^2 - P(A_i = V_{i,j}|C_p)^2\right) \tag{17}$$

where, j is an index of value of the attribute i. In this case $V_{i,j}$ would take values 0, 1, 2 ... because we are working with count data.

Hence, the first part of Expression (17) can be written as

$$\text{CU}_{i,k} = \sum_{f=0}^{\infty} P(A_i = f|C_k)^2 \tag{18}$$

Let's use $\text{CU}_{i,k}$ to refer to the contribution of the attribute i towards the *Category Utility* of the cluster k.

Substituting Expressions (11) and (14) in Expression (18), we obtain

$$\text{CU}_{i,k} = \sum_{f=0}^{\infty} P(A_i = f|C_k)^2 = \frac{1 - 2p_0(1-p_0) - p(1-2p_0)}{1+p} \tag{19}$$

Here, p_0 and p can be estimated using Expressions (15) and (16).

Expression (19) specifies how to calculate the *Category Utility* contribution of an attribute in a category. Hence, the *Category Utility* of the CLASSIT algorithm, when the distribution of attributes follows Katz's model, is given by

$$\text{CU}_p = \frac{1}{K}\sum_k P(C_k)\left[\sum_i \text{CU}_{i,k} - \sum_i \text{CU}_{i,p}\right] \tag{20}$$

5. CLUSTER EVALUATION METHODS

In the literature hierarchical clustering algorithms have been evaluated qualitatively by presenting the descriptions of the clusters produced [5], quantitatively by demonstrating the benefit of the clustering for a particular retrieval task [14] and by comparing clusters at a layer of the hierarchy to externally provided labels [16]. We shall describe a quantitative approach of the later kind in Section 5.1 and propose a new approach to evaluate the hierarchy in Section 5.2. But, before describing these evaluation methods we need to distinguish between two types of document groupings: *class* and *cluster*. A *class* is a document category that has been provided to us as a part of the dataset. It is what the documents have been labeled with by an external entity and help us in evaluating how good our algorithm is. On the other hand, a *cluster* is a grouping of documents that our algorithm generates. It does so by grouping together the documents it considers similar.

5.1 Evaluating the clusters

One commonly used cluster quality measure is the purity of clustering solution [16]. The Purity of a cluster is defined as

$$p_k = \frac{\max_c\{\text{CF}_k(c)\}}{N_k} \tag{21}$$

where,

- c is the index of classes
- k is the index of clusters

$\text{CF}_k(c)$ = number of items from class c occurring in cluster k. Or, the frequency of class c in cluster k.

N_k = number of items in the cluster k.

Purity of the entire collection of clusters were evaluated by taking the micro and macro average of the cluster qualities.

The drawback of relying only on purity to evaluate the quality of a set of clusters, becomes apparent in hierarchical clustering. When we collect clusters occurring near the lowest level of the hierarchy, we get clusters with very few documents in them. Hence, we obtain clusters with high purity score. In the limit, at the lowest level there are N clusters each containing only one item. Hence, $\max_c\{\text{CF}_k(c)\}$ is 1 for each $k \in \{1, \ldots, N\}$ resulting in purity score of 1. We get larger clusters at a higher level in the hierarchy, which are more likely to contain documents belonging to different classes, leading to a lower purity score. This illustrates how purity score can be misleading when the number of clusters formed is different than the number of classes in the dataset. If we make more number of clusters than there are in the dataset we bias the purity score up. If we make less number of clusters than there are in the dataset we bias the purity score down.

To correct this problem, we define another score of the clustering solution in the following manner.

$$r_c = \frac{\max_k\{\text{CF}_k(c)\}}{N_c}$$

where, N_c is the size of the class c. The other variables are as defined for the expression of the purity score in Expression (21). Here, also we can compute the micro average or the macro average to compute the score for the entire solution.

This is a purity computation with the clustering solution treated as the true classes of the data items and the human generated clusters as the solutions to be evaluated. Using this measure we evaluate how well the "true" *classes* in the datasets are represented in the clusters formed.

These metrics, p_k and r_c, have interpretations that parallel the *precision* and *recall* metrics, respectively, in information retrieval literature. Precision is the fraction of the retrieved documents that are relevant. Our p_k has the precision interpretation when we think of a cluster to retrieve documents from the class to which majority of its elements belong. On the other hand recall is the fraction of all the relevant documents that are retrieved. In the framework we described for p_k, our metric r_c has the recall interpretation.

Taking a cue from the F measure commonly used in IR literature to combine precision and recall, we computed the F score as the harmonic mean of the P and R values:

$$\frac{1}{F} = \frac{1}{2}\left(\frac{1}{P} + \frac{1}{R}\right) \tag{22}$$

The F score is the metric by which we shall measure the quality of our clusters.

5.2 Evaluating the hierarchy

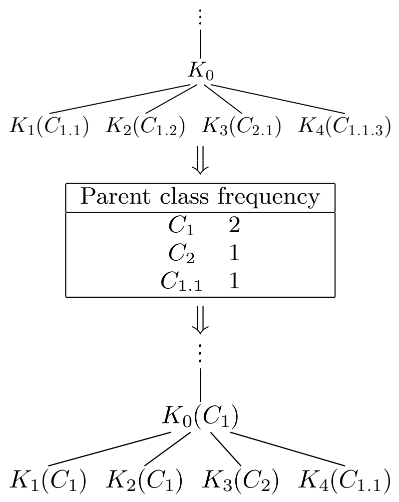
Another question of interest when evaluating a hierarchical clustering algorithm is "To what extent the generated cluster hierarchy agrees with the class hierarchy present in the data?". As we shall describe in Section 6, the datasets we have used in our experiments have hierarchies of classes and provide us a rare opportunity to evaluate our generated cluster hierarchy.

Matching the generated cluster hierarchy with the existing class hierarchy is a non-trivial task. In stead, in this work we focus on measuring *how often the sibling clusters in the generated hierarchy have sibling classes.* For instance, consider the generated cluster subtree shown in Figure 4.

In this case we have already determined the classes of child clusters[3]. To be able to measure if they are filed under the correct class, we need to find the class of the parent cluster. To do this we tabulate the parent classes of the child clusters and assign the most frequent parent class to the parent cluster K_0. So, in this case the parent cluster K_0 gets the label C_1. Then we evaluate this cluster configuration as if K_0 is merely a cluster of four other smaller entities, each of which has a class label same as the parent class for which they voted. This is equivalent of saying that as long as the children clusters of K_0 have children classes of the class of K_0, i.e., C_1 in this case, they are correct. Clusters with all other class labels that occur under that parent cluster are incorrect classifications by the algorithm. They should have been somewhere else.

So, in the above example the precision of K_0 would be $\frac{2}{4} = 0.5$. We compute this precision for all the *internal nodes* of the cluster tree and take their average (both micro average and macro average) to compute the overall precision of the hierarchy. This gives us a measure of how much the generated cluster hierarchy agrees with the class hierarchy

[3]At the lowest level each cluster has only one document and its class can be read from the data directly.



K_0
$K_1(C_{1.1})$ $K_2(C_{1.2})$ $K_3(C_{2.1})$ $K_4(C_{1.1.3})$
Parent class frequency
C_1 2
C_2 1
$C_{1.1}$ 1
$K_0(C_1)$
$K_1(C_1)$ $K_2(C_1)$ $K_3(C_2)$ $K_4(C_{1.1})$

Figure 4: A sample subtree with the children nodes. Class labels of the children node are given in parenthesis.

present in the data. We call it sibling precision score of the cluster hierarchy.

We needed to make a few decisions while evaluating the hierarchy in this manner. For instance, we used only the internal nodes to compute the precision of any node. This is because, often times leaf nodes co-exist with internal nodes as children of another internal node. In this case if we compute precision based on leaf nodes, i.e., single documents, then we are mixing the precision of the kind we described in Section 5.1 with the precision of the hierarchy and it is not clear how we should interpret the resulting number. Another decision that needed to be made was, what should we do if a child cluster has the broadest class label assigned to it? Since, we can not find a parent class for these classes, we explored the possibility of (i) dropping such child clusters from our evaluation and (ii) treating them as their own parent cluster since, they are the broadest level classes. In our experiments the results do not change much if we take either of these strategy. So, we shall report only the results we got by dropping the broadest classes.

6. EXPERIMENT SETUP AND RESULTS

We evaluate our algorithm over two text document collections: Reuters-RCV1 and OHSUMED (88-91). These datasets were picked because of the presence of human labeled hierarchical class labels and reasonably large number of documents in them. They are described in more detail in the following section.

6.1 Reuters-RCV1

Reuters-RCV1 dataset is a collection of over 800,000 English newswire articles collected from Reuters over a period of one year(20th Aug 1996 to 19th Aug 1997)[11]. These documents have been classified by editors at Reuters simultaneously under three category hierarchies: "Topic" hierarchy, "Industry" hierarchy and "Region" hierarchy. Only Topic labels and Region labels are guaranteed to be present for each document.

Incremental clustering algorithms are sensitive to the order in which the data items are presented to them. Hence, it is imperative to evaluate them by presenting them data items in the order in which they are expected to receive them in practice. We envision the proposed document clustering algorithms to process documents as they arrive. Therefore, we evaluate them by presenting them documents from Reuters-RCV1 dataset in time order.

6.1.1 Evaluating clusters

We used articles from the first 30 days of the Reuters-RCV1 dataset for our experiments. There were 62935 articles. Stop words were removed from the documents and the terms were stemmed. Then the most informative terms were selected by their cf $\times \log(N/df)$ scores to represent the documents. We repeated the experiments using 100 to 800 terms at step size of 100.

number of documents	62935
number of unique words	93792
average document length	222
number of classes	63

Table 1: rcv1 dataset (First 30 days). Classes are the region classes

We have evaluated the clustering solutions for the correctness of assignment of documents to the clusters using the Topic categories in stead of the Country categories because they have a richer hierarchy. The Countries hierarchy has just one level of categories that are the names of the countries. Topic labels are also available for every document. The average depth of a node in Topic hierarchy is 3 and the average fanout is 3.3. There were 63 topic categories present in the used documents. So, we have extracted 63 clusters from the dendrogram constructed by the clustering algorithms by pruning the nodes with lowest Category Utility from the bottom of the tree. Then we measured their quality using the topic categories of the documents.

The results of the clustering exercise is given in Table 2. We can see that both the algorithms perform about equally well on micro averaging but, Katz's distribution based CLASSIT algorithm dominates Normal distribution based CLASSIT algorithm across varying vocabulary sizes in the macro average of F scores (Table 2 Figure 5).

We are cautious in interpreting the micro averaged-F score. Because, performance over a few big clusters dominates the entire performance metric. Therefore, we also compute the macro-averaged F score, where each cluster gets equal weight, and find that Katz based CLASSIT performs better than Normal based CLASSIT over a wide range of vocabulary sizes.

			K	N
V	100	micro	0.34	0.26
		macro	0.67	0.48
	200	micro	0.32	0.37
		macro	0.71	0.67
	300	micro	0.31	0.27
		macro	0.72	0.49
	400	micro	0.32	0.34
		macro	0.75	0.7
	500	micro	0.29	0.3
		macro	0.66	0.58
	600	micro	0.31	0.33
		macro	0.67	0.68
	700	micro	0.31	0.33
		macro	0.68	0.64
	800	micro	0.32	0.27
		macro	0.67	0.43

Table 2: Cluster quality comparison on RCV1 data

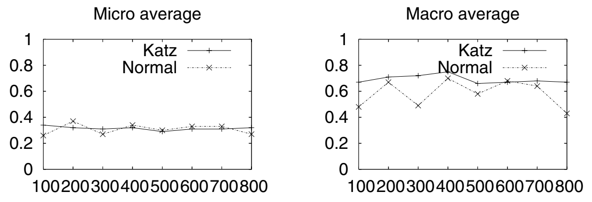


Figure 5: Cluster quality comparison on RCV1 data. The left panel shows the micro average of F-score and the right panel shows the macro average of the F-score.

V	Normal Macro avg	Katz Macro avg	Normal Micro avg	Katz Micro avg
100	0.925	0.956	0.814	0.959
200	0.924	0.935	0.797	0.943
300	0.926	0.874	0.825	0.871
400	0.92	0.866	0.814	0.789
500	0.918	0.896	0.812	0.871
600	0.922	0.841	0.814	0.989
700	0.929	0.836	0.846	0.653
800	0.918	0.855	0.832	0.718

Table 3: Evaluation of the cluster hierarchy using Rcv1 data

Note that the highest micro averaged F score is obtained from Normal based CLASSIT at vocabulary size of 200. This suggests that Normal based CLASSIT can perform better than Katz based CLASSIT, but, it requires careful tuning.

6.1.2 Evaluating hierarchy

We evaluate the generated cluster hierarchy using the *Topic* hierarchy[4] as our reference. The accuracy of the parent/child cluster configurations was evaluated as described in Section 5.2. The results are given in Table 3.

The values in the table cells are the average sibling precision of internal nodes of the cluster hierarchy. As we can see there is no clear winner in this case, although, both the algorithms do reasonably well in assigning sibling classes under the same cluster. However, we must be careful to interpret these values as the correctness of the sibling classes getting grouped together and not as recovering all of the original class hierarchy.

6.2 OHSUMED (88-91)

The OHSUMED test collection is a set of 348,566 abstracts collected from 270 medical journals over a period of 5 years. Each abstract is annotated with MeSH (Medical Subject Heading) labels by human observers. This indicates the topic of the abstract. Unlike the RCV1 dataset, these documents do not have time stamps. Another property of this dataset is, being from a specific subject area, they contain words from a much smaller vocabulary. Due to the presence of human assigned MeSH keywords over such a large collection, this dataset provides us with an opportunity to evaluate our algorithm over a large dataset and against real topic labels.

[4]This can be obtained from [11] Appendix 2.

number of documents	196555
number of unique words	16133
average document length	167
number of classes	14138

Table 4: Ohsumed dataset (88-91)

6.2.1 *Evaluating clusters*

We used the Ohsumed 88-91 dataset from the TREC-9 filtering track to evaluate our algorithm. We selected only those articles for which both the MeSH labels and the abstract text were present. There were 196,555 such articles. As with the RCV1 dataset most informative words in the dataset were selected using $\mathrm{cf} \times \log\left(\frac{N}{\mathrm{df}}\right)$ score of the words. We repeated the clustering exercise using 25 to 200 words at a step size of 25. To determine the number of different topics present in this dataset one can look at the unique MeSH labels present in the dataset. But, as there are 14138 such labels present we used smaller number of clusters to evaluate the algorithms. The F-score results of the experiments are given in Table 5.

We can see from the table that Normal-Classit is the most competitive when the vocabulary size is small *and* the number of clusters formed is large. For all other settings, i.e., when the size of the vocabulary used is larger or when the number of clusters formed is smaller, Katz-Classit performs better. This shows that the Katz-Classit algorithm is more robust as it performs well across a much larger range of parameter values. Performances of both the algorithms suffer when we create more number of clusters, which makes sense because, there are fewer features based on which to distinguish between a large number of clusters.

6.2.2 *Evaluating hierarchy*

MeSH labels present in the Ohsumed collection has a hierarchical structure to it. This provides us with another opportunity to evaluate the correctness of generated hierarchy. This class hierarchy is much larger than the *topic* hierarchy of RCV1 dataset. There are 42610 different MeSH labels. Each MeSH label has a code attached to it. The class hierarchy information can be directly read from this code. For instance the first three records of 2005 "ASCII MeSH collection" reads

Body Regions;A01
Abdomen;A01.047
Abdominal Cavity;A01.047.025
...

The class hierarchy of these three labels is apparent from their "." separated codes.

Documents were pre-processed as described in the previous section. Entire cluster hierarchy was generated and the correctness of the hierarchy was evaluated as described in Section 5.2. The precision values are reported in Table 6. Here again both the algorithms do reasonably well in grouping classes with common parents under the same cluster with Katz-Classit seeming to have an advantage over Normal-Classit across all vocabulary sizes. But, we must be careful here not to interpret these precision values as closeness of the entire cluster hierarchy to the existing class hierarchy. Instead it is the accuracy of the algorithms in classifying sibling classes under same parent cluster.

V	Normal	Katz	Normal	Katz
	Macro avg	Macro avg	Micro avg	Micro avg
25	0.786	0.795	0.626	0.749
50	0.781	0.831	0.667	0.784
75	0.79	0.857	0.654	0.831
100	0.801	0.888	0.742	0.891
125	0.828	0.939	0.788	0.976
150	0.847	0.935	0.812	0.963
175	0.876	0.91	0.859	0.858
200	0.894	0.958	0.819	0.919

Table 6: Evaluation of the cluster hierarchy using Ohsumed data

We also tracked the sibling precision scores at different depths of the generated cluster tree (Figure 6). These plots show the general trend at different vocabulary sizes. As we can see there is considerable variation in the sibling precision over different depths. Amidst these variation we can observe that the sibling precision is more consistent when we look at the nodes occurring at the lower layers of the tree. Also, we find that on these layers the Katz-Classit usually performs better than the Normal-Classit.

The general consistency of the averages at the lower levels of the tree and lack of it at higher levels are artifacts of the available number of clusters at different layers. At the lower levels we have a large number of nodes at each layer. When we average the performance of each algorithm over these large number of nodes we get a score that is robust to random mistakes. So, we get a consistent score from layer to layer and it is easier to see which algorithm does better. But, it is not so at the higher levels. At the higher levels we have only a few nodes in each layer over which to average the score. So, the average is more sensitive to random mistakes. Note that both the micro average and macro average are sensitive to these random mistakes. The wrong nodes in the higher levels of the tree either get a weight equal to other nodes (macro average) or they get a weight that is proportional to the number of *documents* in them. Both of these weights are significant at these levels of the tree. This is the reason why we find the plot of average sibling precision fluctuating a lot at these levels and we do not get a clear winner across the layers in the upper part of the tree.

7. CONCLUSION

We have evaluated an incremental hierarchical clustering algorithm, which is often used with non-text datasets, for text document datasets. We have also proposed a variation of the same that has more desirable properties when used for incremental hierarchical text clustering.

The variation of Cobweb/Classit algorithm that we have demonstrated in this work uses Katz's distribution instead of Normal distribution used in the original formulation of the Classit algorithm. Katz's distribution is more appropriate for the word occurrence data as has been shown in prior work[10] and empirically observed in our work. We have evaluated both the algorithms over Reuters-RCV1 dataset, which allows us to carry out the experiments in a scenario very similar to the real life. We tested the algorithms by presenting them Newswire articles from Reuters-RCV1 dataset in time order and have shown that although Normal assumption for the word occurrence distribution works surprisingly

k⟶ V↓	5			10			20			40			80			160			320			640		
25	?	K	N	?	K	N	√	K	N	?	K	N	√	K	N	×	K	N	?	K	N	×	K	N
	μ	57	55	μ	57	53	μ	57	53	μ	57	53	μ	55	38	μ	55	61	μ	36	34	μ	27	34
	M	62	62	M	60	61	M	63	62	M	62	62	M	60	54	M	37	54	M	49	52	M	43	52
50	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	×	K	N	×	K	N
	μ	70	57	μ	70	57	μ	69	57	μ	69	57	μ	69	57	μ	69	57	μ	48	51	μ	47	51
	M	74	65	M	75	63	M	75	65	M	76	69	M	76	70	M	76	71	M	60	65	M	59	65
75	?	K	N	?	K	N	?	K	N	×	K	N	×	K	N	×	K	N	√	K	N	√	K	N
	μ	70	70	μ	70	70	μ	70	70	μ	69	70	μ	69	70	μ	69	**70**	μ	69	39	μ	69	35
	M	71	71	M	69	70	M	73	77	M	76	80	M	77	81	M	78	**82**	M	78	56	M	79	53
100	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N
	μ	70	62	μ	69	62	μ	69	62	μ	69	62	μ	68	62	μ	68	45	μ	69	45	μ	69	45
	M	72	69	M	71	70	M	75	73	M	78	75	M	78	76	M	79	62	M	80	62	M	80	62
125	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	×	K	N	√	K	N	√	K	N
	μ	71	61	μ	71	61	μ	69	61	μ	69	61	μ	69	61	μ	53	61	μ	53	47	μ	53	46
	M	74	68	M	76	68	M	77	71	M	78	72	M	80	74	M	68	74	M	69	61	M	69	60
150	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	×	K	N
	μ	72	54	μ	72	51	μ	59	51	μ	59	51	μ	55	51	μ	54	51	μ	54	51	μ	48	51
	M	72	65	M	77	61	M	72	64	M	74	66	M	71	66	M	71	67	M	71	67	M	66	67
175	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N
	μ	71	52	μ	71	51	μ	71	51	μ	**71**	51	μ	59	51	μ	58	43	μ	54	43	μ	54	41
	M	74	64	M	78	62	M	81	64	M	**83**	66	M	75	67	M	74	60	M	71	60	M	71	58
200	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N	√	K	N
	μ	62	52	μ	62	50	μ	62	50	μ	62	50	μ	62	50	μ	62	50	μ	62	50	μ	62	50
	M	72	63	M	75	62	M	77	65	M	78	65	M	79	66	M	79	67	M	79	67	M	79	67

Table 5: Cluster quality comparison on Ohsumed data at different number of clusters (k) and vocabulary size (V). The figures in the table are F-score×100. K stands for Katz-Classit, N for the original Classit. μ and M row in the smallest table hold the micro and macro average of the F-score respectively. The cells where Katz-Classit performs better are marked with a $\surd$, the cells where Normal-Classit performs better are marked with a × and the cells where there is no clear winner are marked with a ?. Best Katz-Classit and the best Normal-Classit are shown in bold.

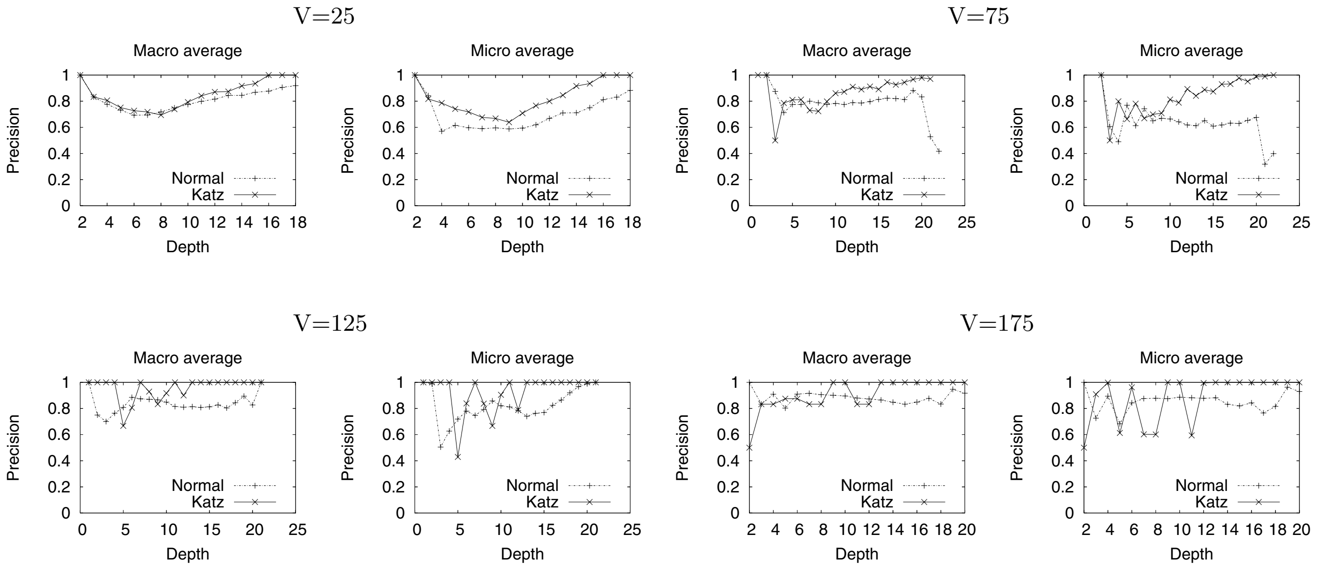


Figure 6: Tracing the sibling precision over the height of the tree. Vocabulary 25, 75, 125 and 175.

well, the performance can be improved by using a more appropriate distribution. We have also evaluated both the algorithms using OHSUMED 88-91 dataset and have found that Katz-CLASSIT performs better except for the narrow range of parameter values with small vocabulary sizes *and* large number of clusters. This shows that the performance of Katz-CLASSIT is more robust across broad parameter settings.

We have also proposed a way to evaluate the quality of the hierarchy generated by the hierarchical clustering algorithms, by observing how often children clusters of a cluster get children classes of the class assigned to the cluster. We found that although, the existing Normal distribution based algorithm and proposed Katz distribution based algorithm perform well in this metric, Katz distribution based algorithm performs marginally better on OHSUMED dataset.

The most important contribution we think we have made in this work is a separation of attribute distribution and its parameter estimation from the control structure of the CLASSIT algorithm. Thus, one can use a new attribute distribution, which may be different from Normal or Katz, and is more appropriate for the data at hand, inside the well-established control structure of the CLASSIT algorithm to carry out incremental hierarchical clustering of a new kind of data. For instance, if it is considered that Negative Binomial could be better fit for the word distribution than Katz distribution, and one can come up with an efficient way to estimate the parameters of the distribution, it can be used in the framework of the existing CLASSIT algorithm as demonstrated in this work. One can also experiment using a Bayesian approach to estimate the parameters of the distribution and carry out incremental hierarchical clustering in this framework, which might lead to better results due to more reliable parameter estimates for clusters with a small number of documents.

Acknowledgement

This work was partially supported by "Cylab ARO grant No. 9097".

8. REFERENCES

[1] J. Allan, R. Papka, and V. Lavrenko. On-line new event detection and tracking. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 37–45. ACM Press, 1998.

[2] J. Banerjee, A.; Ghosh. Competitive learning mechanisms for scalable, incremental and balanced clustering of streaming texts. In *Proceedings of the International Joint Conference on, Neural Networks*, volume 4, pages 2697– 2702, Jul 2003.

[3] A. Bookstein and D. R. Swanson. A decision theoretic foundation for indexing. *Journal of the American Society for Information Science*, pages 45–50, Jan-Feb 1975.

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

[5] M. Dittenbach, D. Merkl, and A. Rauber. Organizing and exploring high dimensional data with the growing hierarchical self organizing map. In L. Wang, S. Halgamuge, and X. Yao, editors, *Proceedings of the 1st International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2002)*, volume 2, pages 626–630, Singapore, November 18–22 2002.

[6] G. Doddington, J. Carbonell, J. Allan, J. Yamron, U. Amherst, and Y. Yang. Topic detection and tracking pilot study final report, Jul 2000.

[7] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[8] J. H. Gennari, P. Langley, and D. Fisher. Models of incremental concept formation. *Journal of Artificial Intelligence*, 40:11–61, 1989.

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

[10] S. M. Katz. Distribution of content words and phrases in text and language modelling. *Nat. Lang. Eng.*, 2(1):15–59, 1996.

[11] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[12] X. Liu and W. B. Croft. Cluster-based retrieval using language models. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Language models, pages 186–193, 2004.

[13] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, England, 2000.

[14] A. F. Smeaton, M. Burnett, F. Crimmins, and G. Quinn. An architecture for efficient document clustering and retrieval on a dynamic collection of newspaper texts. In *BCS-IRSG Annual Colloquium on IR Research*, Workshops in Computing. BCS, 1998.

[15] Y.-J. Zhang and Z.-Q. Liu. Refining web search engine results using incremental clustering. *International journal of intelligent systems*, 19:191–199, 2004.

[16] Y. Zhao and G. Karypis. Evaluation of hierarchical clustering algorithms for document datasetscikm577-sahoo.ps. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 515–524. ACM Press, 2002.

[17] Y. Zhao and G. Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Mach. Learn.*, 55(3):311–331, 2004.