Incremental Hierarchical Clustering of Text Documents

by Nachiketa Sahoo
Advisers:
Dr. James P. Callan (Joint Chair Adviser)
Dr. George Duncan
Dr. Ramayya Krishnan (Joint Chair Adviser)
Dr. Rema Padman

Abstract

A version of cobweb/classit is proposed to incrementally cluster text documents into cluster hierarchies. The modification to classit consists of changes to the underlying distributional assumption of the original algorithm that are suggested by text document data. Both the algorithms are evaluated using standard text document datasets. We show that the modified algorithm performs better than the original Classit when presented with Reuters newswire articles in temporal order, i.e., the order in which they are going to be presented in real life situation. It also performs better than the original Classit on the larger of eleven standard text clustering datasets we used.

1 Introduction

1.1 Motivation

Document clustering is an effective approach to manage information overload. Documents can be clustered, i.e. grouped into sets of similar documents, with the help of human editors or automatically with the help of a computer program. Examples of manual clustering of websites, each a collection of documents, can be found in Yahoo![5] and Open Directory Project [2]. In these examples one can see that websites are grouped into broad topics and narrower subtopics within each broad topic, as opposed to many groups at the same level. This is an example of *hierarchical clustering*. Attempts at manual clustering of web documents are limited by the number of available human editors. For example, although the Open Directory Project has 67,026 editors to file a submitted website into the right category, the average wait time of a newly submitted site before it enters the appropriate category could be up to two weeks. A more efficient approach would be to use a machine learning algorithm to cluster similar documents into groups that are easier to grasp by a human observer. Two examples of such use of automated clustering are Vivisimo [4] and Google News [1].

Vivisimo offers an application that can be used to cluster results obtained from a search engine as a response to a query. This clustering is done based on the *textual similarity* among result items and not based on the images or the multimedia components contained in them. Therefore, this type of clustering is known as text clustering or *text document clustering*. An example of Vivisimo clustering is shown in Figure 1. In this example the Vivisimo search engine was queried for ‘‘document clustering”. The returned results are grouped into clusters labeled ‘‘Methods”, ‘‘Information Retrieval”, ‘‘Hierarchical”, ‘‘Engine” etc. Thus a user interested in ‘‘hierarchical clustering” of documents can browse the results in the ‘‘Hierarchical” group. Note that in this example of document clustering there is no hierarchy of clusters, i.e., all the clusters are at the same level.

On the other hand, Google News collects news articles from about 4500 sources and automatically clusters them into different groups such as "World", "U.S.", "Business", "Sci/Tech", "Sports", "Entertainment", and "Health" (Figure 2). Inside each group the articles are grouped together according to the event they describe. A user interested in news articles related to science and technology may browse the articles in the "Sci/Tech" group. Another user who is interested in an in-depth coverage of a particular event might want to browse all the articles grouped under that event. This is an example of hierarchical clustering of documents, where the hierarchy of clusters has two levels.

Document datasets can be clustered in a batch mode or they can be clustered incrementally. In batch clustering all the documents need to be available at the time clustering starts. Then the clustering algorithm iterates multiple times over the dataset and improves the quality of clusters it forms. However, in some important scenarios documents arrive continuously without any obvious boundary as to where the collection process can be terminated and documents can be clustered. Hence, an incremental clustering solution is required in these cases. Three examples of such scenarios are given below.

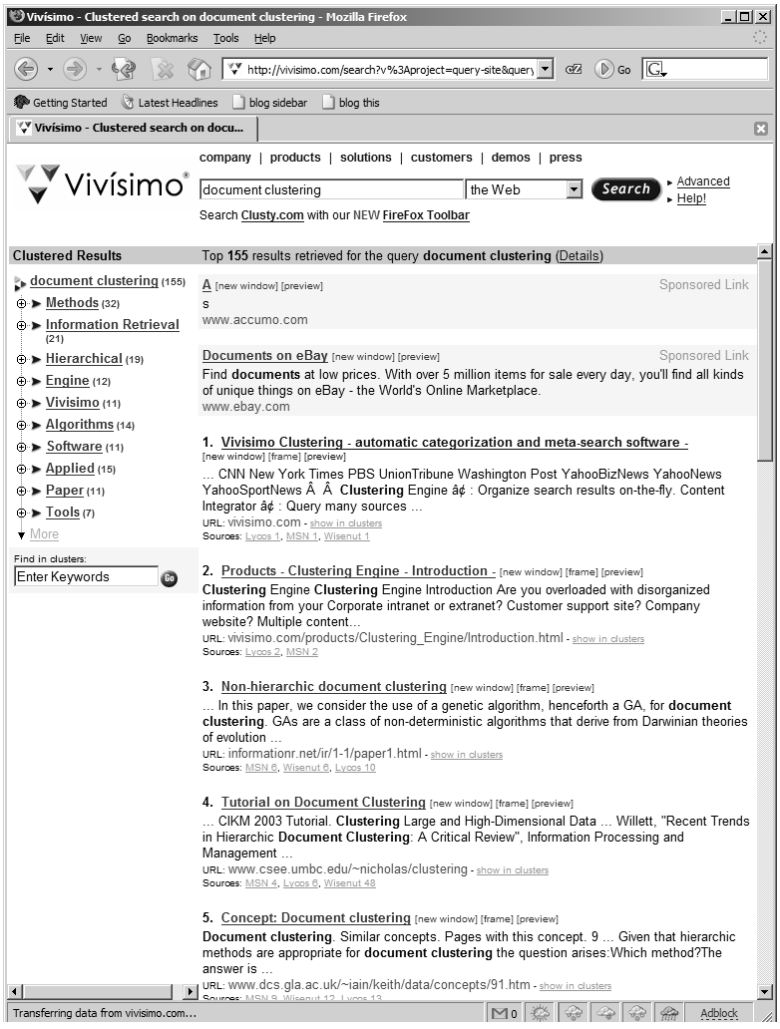


Figure 1. Vivisimo clustering solution.

News stories, Usenet postings and Blogs

News stories, Usenet postings and blogs are interesting because of the diversity of the topic they cover and the large volume of information they generate. This advantage also suggests that an end user of such information would benefit from an organization of documents into topically

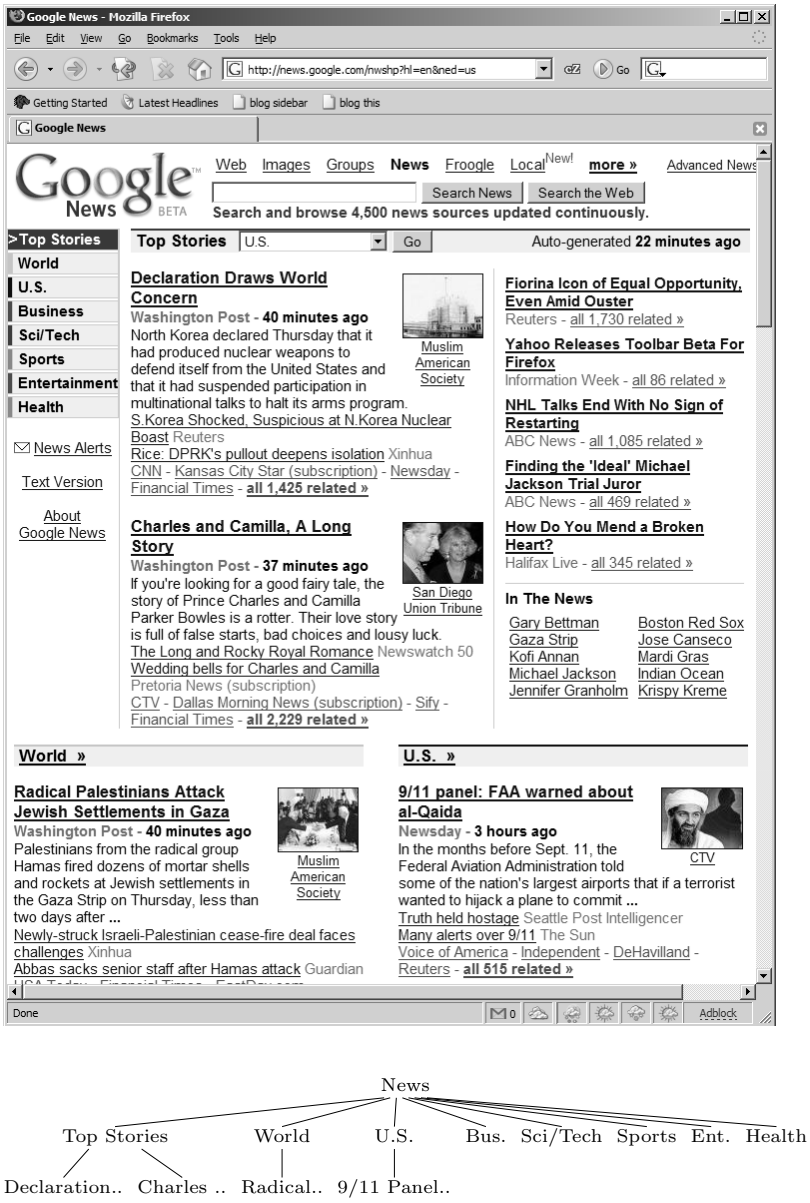


Figure 2. Google News: snapshot and hierarchy sketch.

related groups. Such an arrangement would allow the user to browse only interesting groups of documents. Another benefit of clustering documents into related groups is that when documents are clustered this way they are easier to assimilate because the structure within the broad topic from which the documents are drawn becomes apparent. The structure within the broad topic becomes more apparent when one organizes the clusters as broad and narrow ones, and establishes superset-subset relationship among them. This is called hierarchical clustering.

The reason for using a hierarchy of clusters instead of clusters residing at the same level of granularity is twofold. First, by describing the relationship between groups of documents one makes it possible to quickly browse to the specific topic of interest. The second reason is a technical one. Finding the right number of clusters in a set of documents is an ill-formed problem when one does not know the information needs of the end user. In the Google News example we can group the news articles into seven broad categories or we can group them into individual events, which would produce a much larger number of clusters. The first set of clusters would be helpful for a reader who is interested only in one topic such as ‘‘Sci/Tech’’, while the second set of clusters would be helpful for someone who wants to know about all the events that occurred last week. Clearly, the number of clusters produced in each case is different and neither one is the *only* correct solution.

The different number of clusters are the result of different user's desire to browse documents at different topic specificities. As a solution to this we can present the user with a topic hierarchy populated with documents, which she can browse at her desired level of specificity. Thus we would circumvent the problem of finding the right number of clusters while generating a solution that would satisfy users with different needs.

News articles—and often Usenet postings and blogs—are time sensitive: a fact most apparent in financial services, where, transactions are needed to be carried out based on breaking news. So, quick access to relevant news documents is important. This requires that we process incoming documents as soon as they arrive and present the user with a continuously updated cluster hierarchy. We need an on-line (incremental) hierarchical clustering algorithm for such processing.

Web pages being collected using a web-crawler

Web crawlers are used to collect web pages by following hypertext links in the collected documents to fetch newer documents. Such crawls are done in an unrestricted manner by general purpose web search engines such as Google and AltaVista to collect and index as many documents as possible. But, such unrestricted crawls of the World Wide Web are expensive due to the high storage cost and the bandwidth cost they entail. Another factor that should be kept in mind while designing a crawler is how to keep the collected documents up-to-date. It is important for a crawler to periodically revisit indexed pages and obtain a newer copy if it has changed so that the index is not filled with obsolete documents[12]. But, time spent in revisiting the already indexed pages further limits the amount of web pages a crawler can acquire.

An alternative strategy is to use a topic specific crawler that collects documents belonging to a given topic. Topic specific crawlers predict the relevance of a new hypertext link based on a number of factors, such as relevance of the document that contains the link, presence of topic keywords near the the link, relevance of sibling links in the constructed hyper-link graph, etc., and follow the most relevant link[6].

One can cluster the documents collected by a web crawler into topic hierarchies and use this learned structure in the documents to guide the crawler so that the topic of interest is explored in a desirable manner. However, given the rate at which a crawler collects documents, re-clustering the entire set of collected documents to accommodate each new document in the cluster would be extremely time consuming while resulting in a cluster hierarchy that is similar to the old one. This calls for an incremental clustering solution that can assign a new document to the appropriate node of the existing tree and carry out any necessary *local* reorientation of the tree.

Such application scenarios have motivated this research. Although, work has been done in clustering text documents into topic hierarchies in a batch manner, little research has been done in incrementally clustering text documents into hierarchies of topics. Our work provides—and evaluates using existing datasets—a set of algorithms for incremental clustering of text documents. It is based on an existing incremental clustering algorithm called COBWEB [10].

1.2 Literature review

Clustering is a widely studied problem in the Machine Learning literature [22]. The prevalent clustering algorithms have been categorized in different ways depending on different criteria, such as hierarchical vs. non-hierarchical, partitional vs. agglomerative algorithms, deterministic vs. probabilistic algorithms, incremental vs. batch algorithms, etc. Hierarchical clustering algorithms and non hierarchical clustering algorithms are categorized based on whether they produce a cluster hierarchy or a set of clusters all belonging to the same level. Different hierarchical and non-hierarchical clustering algorithms for text documents have been discussed by Manning and Schutze[28]. Clustering algorithms can be partitional or agglomerative in nature. In a partitional algorithm one starts with one large cluster containing all the documents in the dataset and divides it into smaller clusters. On the other hand, an agglomerative clustering algorithm starts with all documents belonging to their individual clusters and combines the most similar clusters until the desired number of clusters are obtained. Deterministic clustering algorithms assign each document to only one cluster, while probabilistic clustering algorithms produce the probabilities of each item belonging to each cluster. The former is said to make ‘‘hard" assignment while the later is said

to make "soft" assignments. Incremental clustering algorithms make one or very few passes over the entire dataset and they decide the cluster of an item as they see it. But, the batch clustering algorithms iterate over the entire dataset many times and gradually change the assignments of the items to the cluster so that a clustering criterion function is improved. One such criterion function is the average similarity among documents inside the clusters formed. Another criterion function is the average similarity between a document in a cluster and documents outside the cluster. The first criterion is called *average internal similarity* and the second criterion is called *average external similarity*. In a clustering solution we would want high average internal similarity, because that would mean that our clusters are composed of similar items. We would also want low average external similarity because that would mean our clusters are dissimilar, i.e., they do not overlap. The final set of clusters is produced after many iterations when no further improvement of the cluster assignment is possible.

Scatter/Gather

Cutting et al.[13] is one of the first to suggest a cluster aided approach, called Scatter/Gather, to browse large document collections. It describes two fast routines named Buckshot and Fractionation to find the centroids of the clusters to be formed. Then it assigns the documents in the collection to the nearest centroid and recomputes the centroids iteratively until very little or no improvement observed. The last step is similar to the Simple K-means clustering except that in Simple K-means initially one randomly assigns k items as centroids[28]. Note that k is a fixed number here.

Buckshot finds the k centers in the document datasets by drawing a sample of $\sqrt{k\,n}$ documents and clustering them into k clusters using an agglomerative hierarchical clustering routine. The agglomerative hierarchical clustering algorithms have a time complexity of $O\left(n^2\right)$. By drawing a random sample of size $\sqrt{k\,n}$, the time complexity is reduced to $O(k\,n)$. Fractionation, on the other hand, finds k centroids in the following manner. It divides the set of documents into buckets of size m, where $m > k$. Then it clusters each bucket into $\rho\,m$ clusters, where $\rho < 1$ and is a constant. Then it repeats the process of partitioning the data and clustering them treating each of the formed cluster as a one data item, until k clusters are obtained. Cutting et al. have shown that Fractionation has a time complexity of $O(m\,n)$. The center of the clusters formed by the two methods are returned as the starting points for the Simple K-means clustering routine.

With the help of these two routines they have proposed a cluster aided approach to browse document collections in which the program presents the user with a set of clusters for the document dataset (*Scatter*) along with their descriptive labels. Then the user can select the clusters which interest her and submit them to the program. The program merges the documents contained in those clusters (*Gather*) and clusters them again. This process is repeated until the user's information need is met or the user decides to stop the process.

The recursive clustering idea proposed in Scatter/Gather can be effective in browsing large document sets, especially when one does not know enough about the documents to query a deployed search engine using key words. This concept loosely parallels the idea of organizing documents into a hierarchy of topics and subtopics (subsection 1.1), except that the organization in this case is guided by the user and executed by a clustering routine. However, Scatter/Gather has its limitations. It is a batch clustering routine, hence it cannot be used in some important scenarios as described in subsection 1.1. Another limitation that Scatter/Gather shares with many other clustering algorithms is that it requires the input of k, the number of clusters to present the user. A value of k different from the number of subtopics in the collection might lead to meaningless clusters.

Right number of clusters

Finding the right number of clusters in a non-hierarchical clustering exercise is often a difficult problem [30]. The approaches suggested in the literature can, in general, be divided into two groups [10]. The first approach is a multi-fold cross validation one with likelihood as the objective function, in which one fits a series of mixture models with different numbers of components to a subset of the data called *training data* and computes the likelihood of each model given the remaining subset of the data called *testing data*. The model that results in the highest likelihood is selected.

The second approach also fits a mixture model to the data and computes the likelihood of the model given the *entire* dataset using different number of clusters, but it penalizes a model with a higher number of clusters for increased complexity. Observe that a higher number of clusters can be made to fit any dataset better than a lower number of clusters. Hence, by penalizing a clustering solution for its complexity one can achieve a trade off between fitness, or likelihood, of the model and its complexity, which is optimized at the right number of clusters. One such work has been done by Cheeseman and Stutz in their AUTOCLASS algorithm[11]. Other such works include Bayesian Information Criteria and Minimum Descriptor Length criteria [15].

A different approach has been suggested in Liu et al.[13] for clustering text documents. It uses stability of clustering solutions over multiple runs at each of a set of cluster counts to decide the right number of clusters for the document dataset.

Even when the "right" number of clusters can be determined by an algorithm based on some criterion, human observers often differ about the clusters existing in the dataset and what should be the right number of clusters. One alternative solution is to generate a hierarchy of clusters, also called a *dendrogram*, with all the documents belonging to a single cluster at the top of the hierarchy, each document in its individual cluster at the lowest level of the hierarchy and intermediate number of clusters at levels between the two. Thus, the user can look at the desired level in the hierarchy and find a number of clusters that meets her requirement ([28],[22]).

Incremental document clustering

As part of Topic Detection and Tracking (TDT) initiative ([7], [32], [18] and [14]) some experiments have been done in incrementally clustering text documents. The TDT initiative is a DARPA sponsored project started to study and advance the state of the art in detection and tracking of new events in stream of news broadcast and intelligence reports. The identified tasks of TDT are Story Segmentation, Retrospective Topic Detection, On-line New Event Detection, Topic Tracking and Link Detection. The Story Segmentation task involves breaking a stream of text or audio data without story delimiters into its constituent stories. Retrospective topic detection involves detecting new events in the already collected set of documents. On-line new event detection involves identifying a new event, e.g., an earthquake or a road accident, in a new document. Tracking involves keeping track of evolution of an event by assigning the incoming news stories to their corresponding events. Among these tasks the on-line new event detection task involves incremental clustering. In this task a decision is made, after observing a new item, whether it belongs to one of the existing clusters, or it belongs to a new cluster of its own.

The TDT team at the Carnegie Mellon University (CMU) uses a threshold-based rule to decide whether a new document is another story of one of the detected events or it belongs to a new event of its own. If the maximum similarity between the new document and any of the existing clusters is more than a threshold (t_c) the new document is said to belong to the cluster to which it is most similar and it is merged to the cluster. If the maximum similarity is less than t_c but more than another threshold, t_n, then the document is assumed to be an old story but it is not merged to any cluster. If the maximum similarity is less than t_n, then the document is accepted to be about a new event and a new cluster is formed. They have also investigated adding a time component to the incremental clustering. In this experiment, similarities of a new document to each of the past m documents are computed but they are weighted down linearly depending on how old the past documents are. If the similarity scores computed in this manner are less than a preset threshold then the new document is presumed to be about a new event. This work finds that use of time component improves the performance of new event detection task.

TDT team at the University of Massachusetts Amherst (UMASS) takes a variable thresholding approach to the on line event detection task[7]. For each document that initiates a new cluster the top n words are extracted and called a *query vector*. The similarity of the query vector to the document from which the query was extracted defines an upper bound on the threshold required to be met by a document to match the query. A time dependent component is also used in the variable threshold that makes it harder for a new documents to match an older

query. When a new document d_j is compared to a past query q_i the threshold is computed as $0.4 + p \times (\text{sim}(q_i, d_i) - 0.4) + \text{tp} \times (j - i)$, where $0 < p < 1$ and tp, a time penalty factor, are tunable parameters. q_i is the query generated from document d_i. Such threshold is computed for all existing queries q_is. If the similarity of the new document d_j does not exceed any of the thresholds then the document is assigned to a new cluster and a query is computed for the document, else it is added to the clusters assigned to the queries it triggers. The newly generated cluster is said to have detected a new news event.

Outside the TDT initiative, Zhang and Liu in a recent study have proposed a competitive learning algorithm, which is incremental in nature and does not need to be supplied with the correct number of clusters [33]. The algorithm, called *Self Splitting Competitive Learning*, starts with a prototype vector that is a property of the only cluster present initially. During the execution of the algorithm the prototype vector is *split* and updated to approximate the centroids of the clusters in the dataset. The update of the property vector is controlled, i.e., when a new data point is added to the cluster the prototype vector is updated only if the data point is near enough to the prototype. This determined by another *property* vector that starts away from the prototype and zeroes on to it as more and more data points are added. Time for splitting the cluster associated with the prototype is determined based on a threshold condition. When there are more than one prototype a new data point is added to the prototype nearest to it. They have demonstrated their algorithm over text snippets returned from search engines as a response to a query. However, the success of this algorithm on datasets with longer text documents is yet to be demonstrated.

Yet another on-line algorithm called *frequency sensitive competitive learning* has been proposed and evaluated on text datasets by Banerjee and Ghosh[8], which is designed to produce clusters of items of approximately equal sizes. In this work a version of the K-means clustering algorithm called *spherical K-means* has been modified so that the dispersion of the distributions associated with the clusters reduces as more and more data points are added to them. This makes larger clusters less likely candidates for a new data point than the smaller clusters. Thus, the algorithm is tailored to produce clusters which are more or less equal in size.

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

Incremental Hierarchical Clustering: Nominal Attributes

Methods have been proposed in the non-text domain to cluster items in an incremental manner into hierarchies. Most notable among them is the COBWEB algorithm by Fisher [16] and its derivative CLASSIT [20]. COBWEB is an algorithm to incrementally cluster data points with nominal attributes into cluster hierarchies.

At the heart of COBWEB is a cluster quality measure called Category Utility.

Let C_1, ..., C_K be the child clusters of a cluster C_p. The Category Utility of C_1, ..., C_K is computed as

$$CU_p[C_1, ..., C_K] = \frac{\sum_{k=1}^{K} P(C_k) \sum_i \sum_j [P(A_i = V_{ij} \mid C_k)^2 - P(A_i = V_{ij} \mid C_p)^2]}{K}, \quad (1)$$

where,

$P(C_k)$ = Probability of a document belonging to the parent cluster C_p belongs to the child cluster C_k.

A_i = The ith attribute of the items being clustered (say $A_1 \in \{\text{male, female}\}$, $A_2 \in \{\text{Red, Green, Blue}\}$; assumed to be a multinomial variable),

V_{ij} = jth value of the ith attribute (say, V_{12} indicates "female"),

$P(C_k)$ = the probability of a document belonging to cluster k, given that it belongs to the parent cluster p.

The $P(A_i = V_{ij} \mid C_k)^2$ is the expected number of times we can correctly guess of the value of multinomial variable A_i to be V_{ij} for an item in the cluster k when one follows a probability matching guessing strategy. For example, if we have a variable that takes values A, B and C with probabilities 0.3, 0.5 and 0.2, and we randomly predict that the variable takes value A 0.3 fraction of the time, B 0.5 fraction of the time and C 0.2 fraction of the time, we would be correct in predicting A $0.3 \times 0.3 = 0.09$ fraction of the time, B 0.25 fraction of the time and C 0.04 fraction of the time. A good cluster, in which the attributes of the items take similar values, will have high $P(A_i = V_{ij} | C_k)$ values, hence high score of $\sum_j P(A_i = V_{ij} \mid C_k)^2$. COBWEB maximizes sum of $P(A_i = V_{ij} \mid C_k)^2$ scores over all possible assignment of a document to children clusters. When the algorithm assigns a new item to a child node of the node p, it assigns the item in such a manner that the total *gain* in expected number of correct guesses by moving an item from p to its child node, $\sum_i \sum_j [P(A_i = V_{ij} \mid C_k)^2 - P(A_i = V_{ij} \mid C_p)^2]$, is maximized. In this manner the algorithm maximizes the utility function for each node to which a new item is added.

The COBWEB control structure is shown in Fig 3.

Algorithm `CobWeb` (Adapted from Fisher's original work [16])
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 nodes if it leads to improved Category Utility and call COBWEB(item, Merged Node)
3 Split 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 3. COBWEB control structure.

An illustration of the clustering process is given in Figure 4.

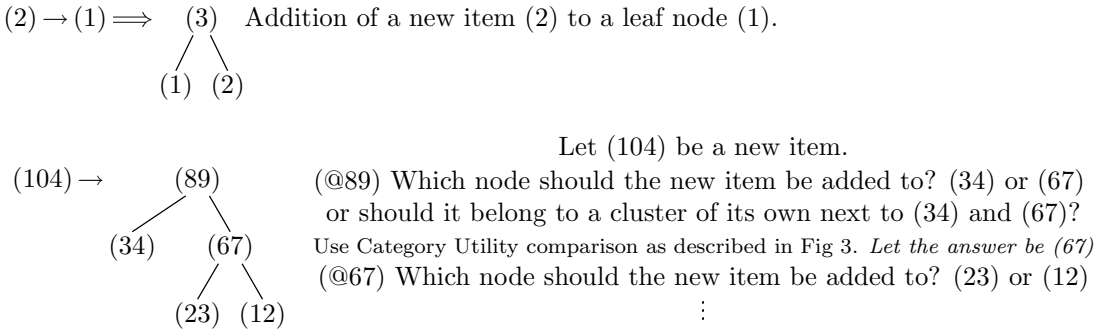


Figure 4. COBWEB illustrated

Assume that there is only one attribute of interest called t and it takes values in $\{A, B, C\}$. Also assume that we have three items a, b and c with t value A, B and C respectively. Further assume that the objects are presented in the order specified, i.e. first a followed by b which is followed by c.

After the first two items are presented the following cluster configuration is arrived without any computation of category utility (First part of Figure 4).

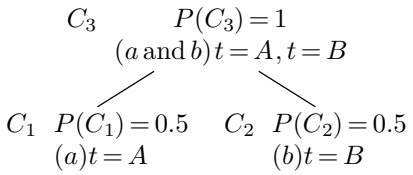
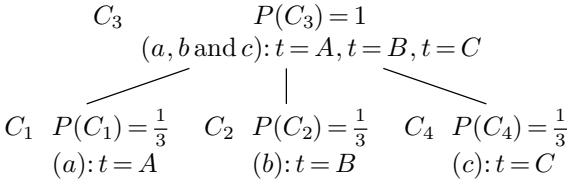


Figure 5. After first two items are added.

C_3 is the root cluster and C_1 and C_2 are two child clusters each containing one item. $P(C_1)$ is the probability that a document randomly picked from its parent cluster of C_1, i.e., C_3, belongs to C_1. Similarly for C_2.

Let's add the third item c to the root node. We can add it at the level of C_1 and C_2 (level 2) as another cluster C_3, or we can add it *in* C_1 or C_2 that will delegate the item c to the third (a new) level. So, our options are (omitting the c within (b, c) configuration that is analogous to the c within (a, c) configuration described below):



or

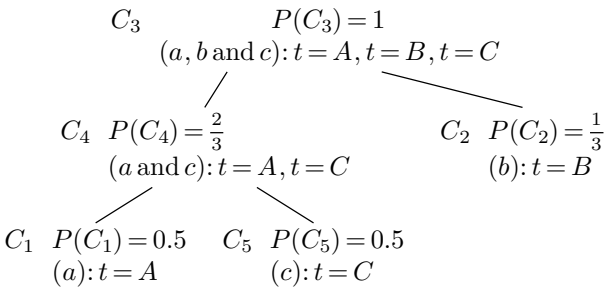


Figure 6. Two partitions of the root cluster.

At this point Category Utilities of the two configurations let us decide which configuration to choose. Note that we need to compute category utility of the two partitions of the root clusters. They can be computed using expression (1) as described below.

For the first configuration in Figure 6 the parent cluster is C_3 and the child clusters are C_1, C_2 and C_4. The category utility of this configuration is:

$$
\begin{aligned}
\text{CU}^1 &= \frac{\sum_{k=\{1,2,4\}} P(C_k)\left[\sum_{A_i=t}\sum_{t=\{A,B,C\}} P(t|C_k)^2 - \sum_{A_i=t}\sum_{t=\{A,B,C\}} P(t|C_3)^2\right]}{3} \\
&= \left[\frac{1}{3}\left\{1^2 - \left(\left(\frac{1}{3}\right)^2 + \left(\frac{1}{3}\right)^2 + \left(\frac{1}{3}\right)^2\right)\right\}\right. \\
&\quad + \frac{1}{3}\left\{1^2 - \left(\left(\frac{1}{3}\right)^2 + \left(\frac{1}{3}\right)^2 + \left(\frac{1}{3}\right)^2\right)\right\} \\
&\quad \left. + \frac{1}{3}\left\{1^2 - \left(\left(\frac{1}{3}\right)^2 + \left(\frac{1}{3}\right)^2 + \left(\frac{1}{3}\right)^2\right)\right\}\right] \Big/ 3 \\
&= \frac{2}{9}
\end{aligned}
$$

For the second configuration in Figure 6 the parent cluster is C_3 and the child clusters are C_4 and C_2.

$$\begin{aligned}
\text{CU}^2 &= \frac{\sum_{k=\{4,2\}} P(C_k)\left[\sum_{A_i=t}\sum_{t=\{A,B,C\}} P(t|C_k)^2 - \sum_{A_i=t}\sum_{t=\{A,B,C\}} P(t|C_3)^2\right]}{2} \\
&= \left[\frac{2}{3}\left\{\left(\left(\frac{1}{2}\right)^2+\left(\frac{1}{2}\right)^2\right)-\left(\left(\frac{1}{3}\right)^2+\left(\frac{1}{3}\right)^2+\left(\frac{1}{3}\right)^2\right)\right\}\right. \\
&\quad \left.+\frac{1}{3}\left\{1^2-\left(\left(\frac{1}{3}\right)^2+\left(\frac{1}{3}\right)^2+\left(\frac{1}{3}\right)^2\right)\right\}\right]\Big/2 \\
&= \frac{1}{6}
\end{aligned}$$

Since, $\text{CU}^1 > \text{CU}^2$ we select configuration 1 over configuration 2. Looking at the Figure 6, it is intuitive to make a new cluster for the third item, because, it has an attribute value not seen in any of the existing categories.

There is one more possible configuration, where c is added below C_2 instead of C_1, but that is symmetrical to the second configuration in Figure 6. So, the analysis will be identical to the one shown in previous paragraph.

Incremental clustering algorithms, such as COBWEB, are sensitive to the order in which items are presented [16]. COBWEB makes use of *split* and *merge* operations to correct this problem. In the merge operation the child nodes with highest and second highest Category Utility are removed from the original node and made child nodes of a new node, which takes their place under the parent node. In the split operation the best node is removed and its child nodes are made children of the parent of the removed node. Merge and split operations are only carried out if they lead to a better Category Utility than obtainable by either assigning the item to existing best node or to a new cluster of its own. By using these two operators, the algorithm remains flexible on the face of change in property of data items in the subsequent observations.

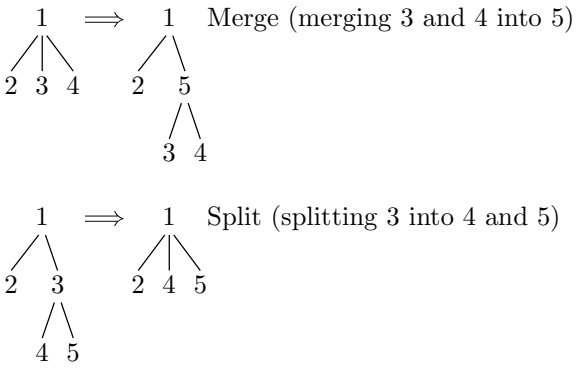


Figure 7. Merge and split operations illustrated.

Incremental Hierarchical Clustering: Numerical Attributes

We now consider an extension of the COBWEB from nominal attributes to numerical attributes. Gennari et al.[20] has shown that in order to use COBWEB for data items with numeric, rather than nominal, attribute values we need to make some assumption about the distribution of attribute values. When the values of each attribute follow a normal distribution, they have shown that the Category Utility function can be written as

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

where,

σ_{ip} = standard deviation of the value of the attribute i in parent node p, and

σ_{ik} = standard deviation of the value of the attribute i in the child node k.

This algorithm is known as the CLASSIT algorithm.

We have not seen any prior application of either of these algorithms to clustering text documents. Hence, their performance on text document data was uncertain at the time of this work. Some properties of text suggest that Classit, in its current form, is unsuitable for this purpose. Word occurrence counts, attributes of text documents that are commonly used in clustering, are far from Normally distributed (see Figure 8). Also, Normal distribution assumes that the attributes take values on the Real line. However, word occurrences, being counts, take values in the set of nonnegative integers. A more suitable distribution for such count data is Zero Inflated Poisson or Negative Binomial or Katz's distribution[23].

Our work proposes to improve upon the original Cobweb algorithm using distributional assumptions that are more appropriate for word count data.

1.3 Contribution of this research

In this paper we demonstrate methods to carry out incremental hierarchical clustering of text documents. Specifically, the contributions of this work are:

1. A Cobweb-based algorithms for text documents where word occurrence attributes follow Katz's distribution.
2. Evaluation of the algorithms on existing text document datasets, using both external and internal cluster quality measures (See Section 4 for a discussion of the cluster quality measures used).

We do not attempt to find the ''correct" number of clusters in a dataset in this work, but we do recognize that this is an important research question. We shall leave this to be explored in future research.

1.4 Organization of this report

In Section 2 we describe key concepts related to text document clustering, which are central to this work. In Section 3 we explain the contributions of our work. In Section 4 we describe the cluster quality metrics that we have used to evaluate the results obtained. In Section 5 we explain the setup of the experiment and discuss the results. In Section 6 we conclude with scope for future research.

2 Text Document clustering and word distributions

2.1 Clustering

A complete exposition of clustering algorithms is beyond the scope of this report. Readers interested in learning more about different clustering algorithms are referred to the review paper by Jain et al[22]. In this section we describe only those concepts of text clustering that will help the reader better understand our work on incremental hierarchical clustering of text.

Clustering is grouping together of similar items. This is also called *unsupervised learning* because no training data is available to help in deciding which group an item should belong to. A clustering algorithm requires

- **A representation of the items.** It is not always obvious what should be the attributes of the items and what should be their values. For example, for text documents, often words contained in them are used as the attributes, but, consecutive sequence alphabets in the document or consecutive sequence of two words, called *bigrams*, have also been used. In selecting the words, one might decide to drop the words that contain numbers or the words that *are* numbers. Many such decisions are often required to come up with a set of attributes to use to represent documents. Similarly decisions need to be taken in assigning values to the attributes. For example, when one uses words as attributes, one might want to use the number of times a word occurs in a document as the value of that attribute, or one might weigh these values depending on how informative they are. A scheme to find the attributes to use and to assign values to those attributes specify the representation of the items.

- **A criterion function to be optimized.** This is usually driven by the property we would like to see in the clusters obtained. An example of a criterion could be ‘‘maximize the average internal similarity of the clusters formed”, because, high average similarity would suggest that items belonging to a cluster are similar to each other. Similarly, another criterion function could be to minimize the similarity between documents in different clusters, because, a lower value of inter-cluster similarity would mean that clusters are dissimilar or separate from each other. Experimental evidence suggests that some of these criterion functions work better than the other. A comparison of several criterion functions is given in [34].

2.2 Text document representation

Text, as we commonly know it, is generally available in the form of relatively unstructured documents. Before we can use it for classification or clustering, we need to convert it to items with attributes and values. A common way of converting the document to such a form 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 forgoes information contained in the grammatical structure in the document. 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.

2.2.1 Feature selection

Use of words as attributes and their occurrence as attribute values leads to large number of attributes, also called features. A large number of attributes in a dataset can pose problems to any machine learning algorithm. It can increase the time required to classify or cluster the documents and it can cause the algorithm to learn from noisy attributes thereby degrading the accuracy of the algorithm. The later problem is known as *over fitting*[29]. Several approaches are available to reduce the set of attributes to a smaller set of more informative attributes. They can be broadly divided into two categories, discussed below.

Feature selection using class label When categories (often called *label*s) of training instances are available one tries to learn the features in the data items that are the most informative. This can be determined by observing the distribution of the attribute values in different classes. An attribute that takes different values among items in different classes is more ‘‘informative” than an attribute that takes all possible values in every class. Lets consider an example of a document dataset that contains documents about digital cameras and automobiles. As we discussed, words are often used as attributes while representing text documents. Words such as *camera* and *focus* occur only in documents about digital cameras while words such as *cylinder* and *suspension* occur only in documents about automobiles. On the other hand words such as *price* and *good* occur in documents of both the categories. Such association between words and topics can be used to select words which are associated with only one class, thus, helpful in predicting the class of a document.

Here we use the word ‘‘informative” to mean the utility of an attribute in predicting the class of a data item. Based on this concept several metrics have been proposed to measure the utility of attributes. This can be used to select the most useful of the attributes for the particular task. Some such feature selection metrics include scores based on Information Gain, Chi-Square and Mutual Information [31]. George Forman has done an extensive comparison of 12 feature selection methods for the text classification task[17]. Yang and Pedersen have done a comparative study of several feature selection methods including the ones that use class labels, such as Information Gain, Mutual Information, Chi-Square and those that do not use class labels such as Document Frequency and Term Strength [31].

1. Through out this paper we shall use *word* and *term* interchangeably to refer to the same thing, i.e., a contiguous sequence of alphanumeric characters delimited by non-alphanumeric character(s). E.g. the first *word* or *term* in this footnote is ‘‘Through”.

Feature selection without class label In many of the practical scenarios class labels of the data points are not usually available, e.g., while clustering documents, which are being generated by a web crawler. In fact in any kind of *truly* unsupervised learning exercise, a training set will not be available. In such a situation one has to come up with ways to select features using the distribution of attribute values in the dataset.

Term selection using Inverse Document Frequency (*idf*) Inverse Document Frequency of a term is defined as[2]:

$$\text{idf} \;=\; \log\left(\frac{N}{\text{df}}\right)$$

Where, N is the number of documents present in the dataset and df is the number of documents in the data set that contain the term. Hence, non-informative words like "the", "an" that occur in almost all documents will get a low idf score and the words that occur in only a few documents will get a high idf score. The *idf* score of a term is used to capture the information content of the term.

Based on the relationship between informativeness of a term and its *idf* score, we have used the $cf \times idf$ score of terms to select a small but informative subset for clustering exercise, where, cf is the number of times the word occurs in the entire collection. Our feature selection procedure is described in the algorithm below.

Algorithm `idf based feature selection`

1 For each word compute the *idf* using its corpus document frequency (df)
2 Compute the $cf \cdot idf$ score of each word by multiplying the corpus frequency with *idf*
3 Sort the words in the decreasing order of their $cf \cdot idf$ score
4 Select the top N words.

The step 2 of the algorithm was carried to make sure that our list of selected words is not primarily composed of words that occur accidentally, such as spelling mistakes, or occasional mention of proper nouns, etc.

2.3 Word distribution in the corpus

Zipf and Mandelbrot When one looks at the frequency of different words in a large text collection, one observes certain properties of these word frequencies that are common across collections.

George K. Zipf [35] was among the first to characterize the distribution of such word frequencies in a functional form. Zipf established the following relationship,

$$f \propto \frac{1}{r}\,, \tag{2}$$

where,

$f =$ frequency of a word in the collection

$r =$ rank of the word, when all words are sorted in a decreasing frequency order.

Mandelbrot observed that although Zipf's law gives the overall shape of the curve, it is not as good in predicting the details[26]. In order to fit the data better he has proposed the following relation between frequency and rank of terms.

$$f = \frac{1}{(r+V)^B} \tag{3}$$

Where,

V, B are constants for a given topic.

Zipf and Mandelbrot have cast light on the distribution of word frequencies in a corpus. But this does not suggest anything about the occurrence of individual words across different text documents. Characterization of word occurrences across documents is interesting because of several reasons described next.

2. There are several formulations of *idf*. However, the formulation shown here is one of the most popular ones and has been used in our experiments.

2.4 Word distributions across documents

Often we want to characterize the distribution of individual words over text units, such as documents. This is useful in judging the information content of a word. For instance a word that occurs in every document of the corpus, e.g., ‘‘the” is not as informative as a word that occurs in a only a few, e.g., ‘‘Zipf”[23].

Occurrence statistics 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 the documents into related groups, as is done in this work. Manning and Schutze have discussed several models to characterize the occurrence of words across different documents [28].

2.4.1 Models based on Poisson distribution

Poisson The Poisson distribution is often used to model incidence counts. The probability density function of the Poisson distribution with rate parameter λ is given by

$$P(k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{4}$$

From empirical observations, it has been found that Poisson distribution tends to over estimate the frequency of informative words (content words) [28].

Two Poisson Model There have been attempts to characterize the occurrence of a word across documents using a mixture of Poisson distributions. One such attempts uses two Poisson distributions to model the probability of a word occurring in a document. One of the distributions captures the rate of the word occurrence when the word occurs due to the fact that the word 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, e.g. because of some random cause. This mixture of two probability distributions has the probability density function:

$$P(k) = \alpha \frac{\lambda_1^k e^{-\lambda_1}}{k!} + (1-\alpha) \frac{\lambda_2^k e^{-\lambda_2}}{k!} \tag{5}$$

where, α is the probability of the word being topically relevant and $1-\alpha$ is the probability of the word being topically unrelated to the document.

It has been empirically observed that, although the two Poisson model fits the data better than single Poisson model[9], a spurious drop is seen for the probability of a word occurring twice in a document[23]. The fitted distribution has lower probability for a word occurring twice in a document than it occurring three times. In fact, 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 8).

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[19]. The probability density functions of a Negative Binomial distribution is given below,

$$P(k) = \binom{k+r-1}{r-1} p^r (1-p)^k, \tag{6}$$

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[28]. This has been confirmed in our analysis (see Expressions (30) and (31) in Section 3.2).

Zero inflated Poisson When we observe the word occurrence counts in documents, we find that most words occurs in only a few documents in the corpus. So, for most of the words, the count of documents where they occur zero times is very large (see Figure 8). Looking at the shape of the empirical probability density function we have attempted to model the occurrence counts using a Zero Inflated Poisson distribution, which assigns a large probability mass at the variable value 0 and distributes the remaining probability mass over rest of the occurrence counts according to a Poisson distribution.

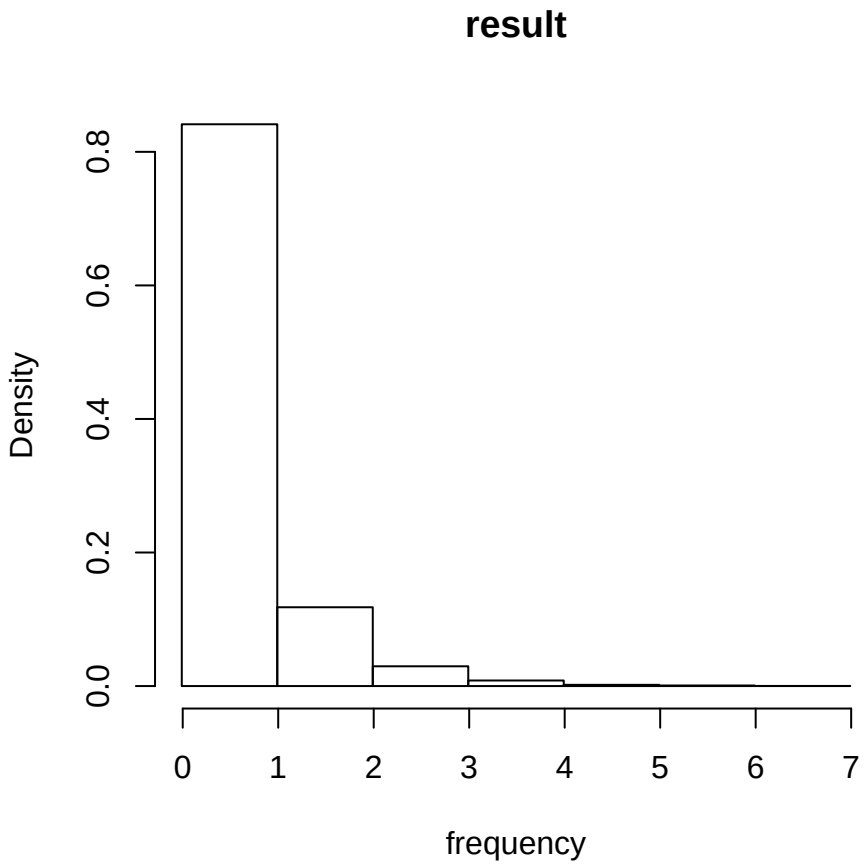


Figure 8. The occurrence of the word ‘‘result” across different documents in our test collection.

The probability density function of Zero Inflated Poisson distribution is given by

$$P(k) = (1-\alpha)\,\delta_k + \alpha\,\frac{\lambda^k e^{\lambda}}{\lambda!}, k = 0, 1, 2... \tag{7}$$

where,

$$\delta_k = \begin{cases} 1, \text{iff } k = 0 \\ 0, \text{otherwise} \end{cases}$$

As we shall demonstrate in Section 2.4.3, this distribution does not fit our test data as well as the Negative Binomial or the Katz's distribution.

2.4.2 Katz's K-mixture model

This distribution, proposed by Katz[23], although simple to work with, 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[28]. Katz's distribution assigns the following probability to the event that word i occurs k times in a document[3].

$$P(k) = (1-\alpha)\delta_k + \frac{\alpha}{\beta+1}\left(\frac{\beta}{\beta+1}\right)^k \tag{8}$$

$\delta_k = 1$ iff $k = 0$ and 0 otherwise.

3. 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.

The MLE estimates of parameters α and β are:

$$\beta = \frac{\text{cf} - \text{df}}{\text{df}} \tag{9}$$

$$\alpha = \frac{1}{\beta} \times \frac{\text{cf}}{N} \tag{10}$$

cf = *collection frequency* = number of times word i occurred in the document collection obtained by adding up the times the word occurred in each document. Here, a collection can be whatever we deem our universe of documents to be. It can be the entire corpus of documents or a subset of it.

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

From (8) it follows that

$$\begin{aligned} P(0) &= 1 - \alpha + \frac{\alpha}{\beta + 1} \\ &= 1 - \frac{\text{df}}{N} \\ &= 1 - \Pr(\text{the word occurs in a document}) \\ &= \Pr(\text{the word does not occur in a document}) \end{aligned} \tag{11}$$

Also, it follows that

$$P(k) = \frac{\alpha}{\beta + 1} \left(\frac{\beta}{\beta + 1} \right)^k, k = 1, 2, ... \tag{12}$$

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

$$P(k) = \alpha \, (1 - p) \, p^k \tag{13}$$

Let's define a parameter p_0 as

$$p_0 = P(0) \tag{14}$$

using (9) we find that

$$\begin{aligned} p &= \frac{\frac{\text{cf} - \text{df}}{\text{df}}}{\frac{\text{cf}}{\text{df}}} \\ &= \frac{\text{cf} - \text{df}}{\text{cf}} \\ &= \frac{\Pr(\text{the word repeats in a document})}{\Pr(\text{the word occurs in a document})} \\ &= \frac{\Pr(\text{the word repeats} \bigcap \text{the word occurs})}{\Pr(\text{the word occurs})} \\ &= \Pr(\text{the word repeats} \mid \text{the word occurs}) \end{aligned} \tag{15}$$

Hence, $1 - p$ can be interpreted as the probability of the word occurring only once. Or, it can be thought of as a scaling factor used to make (13) and (14) together a valid probability density function.

We can write Expression (8) 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

$$\begin{aligned} p_0 &= 1 - \alpha \, p \\ \Rightarrow \alpha \, p &= 1 - p_0 \\ \Rightarrow \alpha &= \frac{1 - p_0}{p} \end{aligned} \tag{16}$$

Expression (13) can now be written as

$$P(k) = (1 - p_0) \, (1 - p) \, p^{k-1} \tag{17}$$

when $k > 0$.

Using Expressions (14) and (17), we can fully specify the Katz's distribution. The two parameters are p_0 and p, which can be estimated as (see Expressions 11 and 15)

$$\hat{p_0} = 1 - \frac{\text{df}}{N} \tag{18}$$

and

$$\hat{p} = \frac{\text{cf} - \text{df}}{\text{cf}} \tag{19}$$

It can be shown that if a distribution is defined by Expressions (14) and (17), then the estimates (18) and (19) are the MLE of the parameters p_0 and p (see Appendix B).

2.4.3 Fitness comparison

We estimated the parameters of Zero Inflated Poisson and Negative Binomial using the method of moment, and parameters for Katz's distribution using the Maximum Likelihood Estimate (MLE) method. The reason for using the method of moments and not the MLE is that for the Negative Binomial and the Zero Inflated Poisson distributions the MLE can only be found numerically, which is computationally complex for our task of incremental clustering. One can still use numerical methods to determine MLEs of the parameters of the distribution, which admittedly have better properties, if one is willing to pay the cost in terms of delay. In this work we shall limit ourselves to the estimates that have closed form expressions and can be computed efficiently, because our goal is to carry out the incremental document clustering in real time.

Zero Inflated Poisson If the probability density function of a Zero Inflated Poisson distribution is given in the form of Expression (7), then the method of moment estimates of its parameters α and λ are

$$\hat{\lambda} = \frac{\text{Var}(X)}{\overline{X}} + \overline{X} - 1 \tag{20}$$

and

$$\hat{\alpha} = \frac{\overline{X}}{\lambda} \tag{21}$$

Negative Binomial For the Negative Binomial distribution, parameters p and r can be estimated as

$$\hat{r} = \frac{\bar{X}^2}{\text{Var}(X) - \bar{X}} \tag{22}$$

$$\hat{p} = \frac{\bar{X}}{\text{Var}(X)} \tag{23}$$

For the Katz's distribution we used Expressions (18) and (19) to estimate the parameters p_0 and p.

We evaluated how well these three distributions fit the data by computing the likelihood of the estimated parameters on three different datasets[4]. For each dataset we selected the top 100 terms by their $cf \cdot idf$ score. We computed the likelihood of each of these distribution at the respective estimates of the parameters. The distribution that has a higher likelihood than the another can be considered a better fit to the data. For each term a pairwise comparison of fitness of different distributions is carried out in this manner. The results are shown in the form of three dominance matrices in Table 1. Each cell records the number of terms for which distribution for the row has 10% or higher likelihood than the distribution for the column.

	classic				tr41				k1a		
	NB	Katz's	ZIP		NB	Katz's	ZIP		NB	Katz's	ZIP
NB	0	55	92	NB	0	41	98	NB	0	63	98
Katz's	41	0	96	Katz's	58	0	98	Katz's	35	0	98
ZIP	7	4	0	ZIP	2	2	0	ZIP	2	2	0

Table 1. Likelihood comparisons, count of likelihood of row distribution > likelihood of col distribution × 1.1

4. More on the datasets is given in Section 5.1.1.

It can be observed from the table that Katz's distribution, is not only easier to work with as we will see in Section 3, it also fits better than Zero Inflated Poisson (ZIP) and gives fitness comparable to Negative Binomial (NB) distribution.

3 Algorithms for text

3.1 COBWEB: when attribute values follow Katz's distribution

3.1.1 Category utility

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

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

Notice that for each attribute indexed i we need to compute

$$\sum_j \big(P(A_i = V_{i,j}|C_k)^2 - P(A_i = V_{i,j}|C_p)^2\big) \tag{24}$$

where, j is an index of value of the attribute i. Notice that 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 (24) can be written as

$$\mathrm{CU}_{i,k} = \sum_{f=0}^{\infty} P(A_i = f|C_k)^2 \tag{25}$$

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

Substituting Expressions (14) and (17) in Expression (25), we obtain

$$\mathrm{CU}_{i,k} = \sum_{f=0}^{\infty} P(A_i = f|C_k)^2 = \frac{1 - 2\ p_0\,(1 - p_0) - p\,(1 - 2\,p_0)}{1 + p} \tag{26}$$

Substituting estimates of p_0 and p from Expressions (18) and (19) in Expression (26), and simplifying, we get

$$\mathrm{CU}_{i,k} = \sum_{f=0}^{\infty} P(A_i = f|C_k)^2 = 1 - \frac{2 \times \mathrm{df}\Big(N - \frac{\mathrm{cf} \times \mathrm{df}}{2 \times \mathrm{cf} - \mathrm{df}}\Big)}{N^2} \tag{27}$$

where, df, cf, and N are counted in the category k.

Expression (27) 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

$$\mathrm{CU}_p = \sum_k \frac{P(C_k)[\sum_i \mathrm{CU}_{i,k} - \sum_i \mathrm{CU}_{i,p}]}{K} \tag{28}$$

where, $\mathrm{CU}_{i,k}$ is given by Expression (27).

3.2 Cobweb: when attribute values follow Negative Binomial distribution

The probability density function of the Negative Binomial distribution is

$$P(x) = \binom{x + r - 1}{r - 1} p^r\,(1 - p)^x \tag{29}$$

p and r are the parameters of the distribution, which are to be estimated from the data.

3.2.1 Category utility

Substituting Expression (29) in (25), we obtain the contribution of a word in a child cluster towards Category Utility

$$\mathrm{CU}_{i,k} = \sum_{x=0}^{\infty} \left(\frac{(x+r-1)!}{x!\,(r-1)!} p^r (1-p)^{x-1} \right)^2 \tag{30}$$

This expression cannot be reduced to any simpler form, although, it can be written using a hypergeometric function in the following manner.

$$\mathrm{CU}_{i,k} = \frac{p^{2r}\, {}_2F_1\big(r, r, 1, (1-p)^2\big)}{(1-p)^2} \tag{31}$$

One can use a library, such as the one available with Mathematica, to numerically evaluate ${}_2F_1(r, r, 1, (1-p)^2)$. In our experience this computation is three orders of magnitudes more resource intensive than computing (27), the equivalent expression for Katz's distribution. As we described in Section 2.4.3, in this work we shall limit ourselves to the methods that will let us carry out incremental clustering in real time, i.e., in the time available between arrival of two documents. We have given a time study in Appendix A that shows if we use Classit based on Negative Binomial distribution we cannot carry out the clustering in real time for the newswire dataset (Reuters-RCV1) we have used in our experiments to evaluate the clustering algorithms. We expect the real life situation to be close to the Reuters-RCV1, because this dataset is a complete collection of the news documents from Reuters in the year of 1996-97.

For this reason and the reasons cited in Section 2.4.1 and 2.4.3, we shall fully explore only Katz's distribution in our work.

4 Cluster Evaluation Methods

We use two cluster quality metrics to evaluate our algorithms. One quality metric is an extrinsic cluster quality metric, i.e., it uses labels that are not a part of the text document but are attached to the documents by an external entity such as human editor. The other quality metric is an intrinsic cluster quality metric, i.e., it does not make use of any data that is not a part of the text document. It is computed based on the textual similarity of the documents, as described in the following sections. This is unlike the extrinsic cluster quality metric in that it does not make use of the external labels.

4.1 Extrinsic quality measures

Extrinsic quality measures compare the clusters generated by an algorithm to a prespecified cluster solution that is assumed to be the true clusters existing in the data. One commonly used extrinsic quality measure is the purity of clustering solution.

The purity of a clustering solution has been defined in the following manner [34]

$$\begin{aligned} P &= \sum_k P(k) \frac{\max_c \{\mathrm{CF}_k(c)\}}{N_k} \\ &= \sum_k \frac{N_k}{N} \times \frac{\max_c \{\mathrm{CF}_k(c)\}}{N_k} \\ &= \frac{1}{N} \sum_k \max_c \{\mathrm{CF}_k(c)\} \end{aligned} \tag{32}$$

where,

– c is the index of classes

class is a prespecified group of items

– k is the index of clusters

cluster is an algorithm generated group of items

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

$P(k)$ = is the purity of the cluster k. It is greater than 0 and less than 1.

N_k = number of items in class k.

N = total number of documents.

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 at or 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, $\max_c \{\mathrm{CF}_k(c)\}$ is 1 for each $k \in \{1, ..., N\}$ 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.

$$\begin{aligned} R &= \sum_c P(c) \frac{\max_k \{\mathrm{CF}_k(c)\}}{N_c} \\ &= \sum_c \frac{N_c}{N} \times \frac{\max_k \{\mathrm{CF}_k(c)\}}{N_c} \\ &= \frac{1}{N} \sum_c \max_k \{\mathrm{CF}_k(c)\} \end{aligned} \tag{33}$$

where, N_c is the size of the class c. The other variables are as defined for the expression of the purity score (32).

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.

Both these measures have interpretations that parallel the *precision* and *recall* metrics in information retrieval literature. To compute purity, we assign to a cluster a class that contains the most number of the items in the cluster. After the assignment, we compute the purity (or precision) of the cluster assuming that the class contains all the relevant documents and the cluster contains all the retrieved documents. Purity is the ratio of number of relevant documents retrieved and the total number of documents retrieved. The average purity score of the entire solution is the weighted average of the purity scores of each cluster: weights being proportional to the size of the clusters formed. On the other hand, to compute score R we assume the cluster that contains the most number of items in a class, to retrieve the class. The number of class members the cluster contains is the number of items retrieved from the class. So, we can compute the recall of this class-cluster solution by dividing the number of items retrieved by the total number of items that could be retrieved. Recall of the entire solution is the weighted average of the recall of each class; weights being the size of each class. Due to this interpretation of the R score we call this measure the *recall* measure.

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:

$$\frac{1}{F} = \frac{1}{2}\left(\frac{1}{P} + \frac{1}{R}\right) \tag{34}$$

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

Entropy is another extrinsic cluster quality measure that is often used to evaluate the clustering solution. The entropy of a cluster k is computed as follows

$$E(k) = -\sum_c P_k(c) \times \log_2(P_k(c))$$

where, $P_k(c)$ is the probability of a document in cluster k belonging to class c or $\frac{\mathrm{CF}_k(c)}{N_k}$. Note that the lower the entropy, the better the quality of the cluster, because that would mean the cluster is primarily composed of items from one class.

We prefer the purity measure over entropy because, as argued in [27], the purity score of a cluster is closer to perceived quality of the clusters than entropy is. Borrowing from a similar example given in [27], lets consider two candidate clusters and five classes shown in Table 2.

		classes					Entropy	Purity
		1	2	3	4	5		
clusters	1	80%	5%	5%	5%	5%	1.12	0.8
	2	50%	50%	0%	0%	0%	1	0.5

Table 2. Two candidate clusters

Inspection of composition of two clusters would suggest that cluster 1 is better than cluster 2 because it is mostly composed of items from only one class, i.e., class 1, while cluster 2 is composed of equal numbers of items from class 1 and class 2. The entropy of two clusters suggest that cluster 2 is better than cluster 1 as it has a lower entropy. However, the purity scores of the clusters agree with our intuition and indicate that cluster 1, having a higher purity score, is better than cluster 2.

4.2 Intrinsic quality measures

4.2.1 Why does one need an intrinsic measure of cluster quality?

As illustrated in Section 1.1, because human observers can cluster documents according to different criteria, solutions generated by them could have different numbers of clusters. They can also have different cluster memberships, i.e., according to the clusters generated by person A, first document and second document might belong to the same group, but, according to the clusters generated by person B, they might belong to different groups. Thus, validating the clusters generated by clustering algorithms using external item-to-class assignments is difficult, because, it is hard to find objective measure of cluster quality.

This suggests that one cannot compare the machine generated clusters with the human generated solution and find out, conclusively, how good the generated clusters are, because, one human generated cluster need not be the *only* correct solution.

Therefore, we decide to evaluate the generated clusters by using an intrinsic measure in addition to the F score. The intrinsic cluster quality we use consists of *textual similarity* of the documents within a cluster and *textual dissimilarity* among documents belonging to different clusters. Intuitively, it can be observed that a solution in which documents inside a cluster are similar to each other is better than a solution in which documents inside a cluster are dissimilar. Also, it can be observed that a solution in which documents inside each cluster are different from documents *outside* the cluster, is better than a solution in which documents inside each cluster are similar to documents outside the cluster. The former suggests that the document clusters are well separated, while in the later, clusters are overlapping.

In order to compute these two metrics we used the cosine of the angle between vector representation of documents as a measure of document similarity. Two documents with similar word occurrences will have vectors with similar attribute values and hence will have a cosine score close to one. On the other hand, documents with different sets of words will have a zero cosine score between their vectors.

4.2.2 Computing internal similarity

One can find all possible pairs of documents in a cluster and compute the cosine between their vector representations. However, there is a faster way to compute the average internal similarity as explained in [34].

Let $S_1, S_2 ... S_K$ be K clusters. Let d_i be the ith document vector and D_k be the sum of document vectors in cluster k. For this analysis we shall assume that the document vectors are already normalized so that they have Euclidean length equal to one[5].

The average internal similarity of the clustering solution can be computed as

$$\begin{aligned}
\text{isim} &= \frac{1}{|S_.|}\sum_{k=1}^{K}|S_k|\left(\frac{1}{|S_k|^2}\sum_{d_i,d_j\in S_k}\cos(d_i,d_j)\right)\\
&= \frac{1}{|S_.|}\sum_{k=1}^{K}|S_k|\left(\frac{1}{|S_k|^2}\sum_{d_i,d_j\in S_k}d_i\cdot d_j\right)\\
&= \frac{1}{|S_.|}\sum_{k=1}^{K}\frac{1}{|S_k|}\sum_{d_i,d_j\in S_k}d_i\cdot d_j\\
&= \frac{1}{|S_.|}\sum_{k=1}^{K}\frac{1}{|S_k|}D_k\cdot D_k\\
&= \frac{1}{|S_.|}\sum_{k=1}^{K}\frac{1}{|S_k|}\|D_k\|^2
\end{aligned} \tag{35}$$

$|X|$ = Cardinality of the set (cluster) X

$\|Y\|$ = Norm 2 of the vector Y

$S_.$ = The complete document set

In order to compute the internal similarity of the solution, we need to compute the square of the length of the aggregate vector for each cluster, inversely weigh it by it's size and add them up.

A high isim score for a clustering solution would mean that the clusters in the solution have documents which are textually similar to other documents in their clusters. Hence, a clustering solution with high isim score contains cohesive clusters.

4.2.3 Computing external similarity

Average external similarity can be computed as

$$\begin{aligned}
\text{esim} &= \frac{\sum_{k=1}^{K}\text{similarity between documents in cluster } k \text{ and documents outside cluster } k}{\text{total number of pairwise similarities computed}}\\
&= \frac{1}{\sum_{k=1}^{K}|S_k|\times(|S_.|-|S_k|)}\sum_{k=1}^{K}\left(\sum_{d_i\in S_k,d_j\notin S_k}\cos(d_i,d_j)\right)\\
&= \frac{1}{|S_.|^2-\sum_{k=1}^{K}|S_k|^2}\sum_{k=1}^{K}\left(\sum_{d_i\in S_k,d_j\notin S_k}d_i\cdot d_j\right)\\
&= \frac{1}{|S_.|^2-\sum_{k=1}^{K}|S_k|^2}\sum_{k=1}^{K}\left(D_k\cdot(D_.-D_k)\right)\\
&= \frac{1}{|S_.|^2-\sum_{k=1}^{K}|S_k|^2}\sum_{k=1}^{K}\left(D_k\cdot D_.-D_k\cdot D_k\right)
\end{aligned} \tag{36}$$

$D_.$ = Sum of document vectors in all clusters.

A low esim score for a clustering solution would mean that the clusters in the solution contain documents which have little textual similarity to documents outside their clusters. They can be thought of as non-overlapping or disjoint clusters.

A good clustering solution will have high average internal similarity and low average external similarity. In order to capture both of these measures in a single score, we shall use the hybrid H metric as suggested in [34].

$$H = \frac{\text{isim}}{\text{esim}} \tag{37}$$

A clustering solution with a high H score would mean that the clusters in the solution are cohesive and disjoint.

5. Note that we will need to normalize vectors in this manner, either before calculating dot product among them, or, while calculating the dot product if we want to compute the cosine between them.

5 Experiment setup and results

We have measured the performance of our proposed formulation of the Classit algorithm along with the performance of the original Classit algorithm using a set of standard clustering datasets and the Reuters-RCV1[25] text document collection. The first dataset is collected from Zhao and Karypis[34]. The second dataset is based on newswire articles collected from Reuters over a period of one year.

It is worthwhile to mention here that the first set of the datasets have traditionally been used in batch clustering exercises. Hence, the ordering of the documents in the dataset were unimportant. However, in a on-line clustering algorithm the order in which documents are presented to the clustering algorithm is important. Therefore, we have also used the second dataset (Reuters-RCV1) in which documents are ordered by their time stamps to test our algorithms.

5.1 Experiments with standard clustering datasets

5.1.1 Datasets

We used 11 datasets that were used in Zhao and Karypis[34] for the batch clustering exercise.

- The datasets "k1a", "k1b" and "wap" have been collected from the WebACE project[21]. Each document in these datasets correspond to a web page listed in the Yahoo! topic hierarchy. The "k1a" and "k1b" datasets contain exactly the same documents, but, "k1b" has 6 classes[6] while "k1a" has a finer categorization of the dataset containing 20 classes.
- The datasets "re0" and "re1" have been collected from the Reuters-21578 text categorization test collection distribution 1.0 [24]. The classes in the Reuters-21578 test collection were divided into two non-overlapping groups. Among the documents belonging to the classes in these two groups only those documents that have exactly one class label were included in the dataset.
- "tr31" and "tr41" have been collected from TREC-6 and TREC-7 collections [3]. The classes of the documents are the names of the topics they were judged relevant to in TREC relevance judgments.
- The "tr11", "tr12" and "tr23" datasets were constructed from TREC-1, TREC-2 and TREC-5 collections respectively. They contain documents belonging to disjoint subsets of topics used in TREC. As in the previous set, classes of the documents were the topics they were judged relevant to in their respective relevance judgments.
- The "tr45" dataset was collected from the TREC-7 collection and the labels of the documents came from the qrels.trec7.adhoc.part5 relevance judgment file.[7]

Dataset	Source	# documents	# unique words	# classes (k)
k1a	WebACE	2340	21839	20
k1b	WebACE	2340	21839	6
re0	Reuters-21578	1504	2886	13
re1	Reuters-21578	1657	3758	25
tr11	Trec-1	414	6429	9
tr12	Trec-2	313	5804	8
tr23	Trec-5	204	5832	6
tr31	Trec-6	927	10128	7
tr41	Trec-7	878	7454	10
tr45	Trec-7	690	8261	10
wap	WebACE	1560	8460	20

Table 3. Description of the dataset used.

6. We use the term *class* to refer to the groups generated by human observers and the term *cluster* to refer to the groups generated by an algorithm.

7. TREC, acronym for Text REtrieval Conference, was started by National Institute of Standards and Technology and United States Department of Defense in 1992 for the advancement of research in Information Retrieval (http://trec.nist.gov/overview.html).

5.1.2 Experiment Setup

For each dataset we selected a small subset of terms with highest $cf \cdot idf$ scores, as described in Section 2.2.1, to represent the documents. We compare the algorithms at vocabulary sizes of 50 to 400 at steps of 50. For each dataset we created as many clusters as there are detected by human observers.

5.1.3 Results and discussion

Effect of using Katz's distribution (Katz's vs. Normal) We compared the clustering results using both F score and H score (computed as described in Expressions (34) and (37) respectively). The scores at different vocabulary sizes are given in the Table 4 and Table 5[8].

	Vocabulary Sizes															
	Data	50		100		150		200		250		300		350		400	
		K	N	K	N	K	N	K	N	K	N	K	N	K	N	K	N
	k1a	0.40	0.40	*0.48*	*0.41*	0.47	0.43	*0.50*	*0.41*	0.42	0.41	0.36	0.40	*0.35*	*0.40*	*0.35*	*0.46*
	k1b	*0.63*	*0.57*	*0.75*	*0.58*	*0.78*	*0.55*	*0.77*	*0.56*	0.64	0.61	*0.74*	*0.58*	*0.74*	*0.60*	*0.75*	*0.66*
	re0	*0.47*	*0.40*	0.47	0.48	*0.51*	*0.39*	*0.53*	*0.44*	*0.58*	*0.44*	*0.58*	*0.47*	*0.58*	*0.41*	*0.58*	*0.48*
	re1	*0.42*	*0.30*	0.38	0.36	0.38	0.39	*0.37*	*0.30*	*0.38*	*0.34*	0.38	0.34	0.38	0.35	*0.38*	*0.32*
F	tr11	*0.44*	*0.50*	*0.42*	*0.50*	*0.45*	*0.51*	0.49	0.51	0.49	0.50	0.49	0.50	0.49	0.50	0.49	0.50
	tr12	*0.40*	*0.47*	*0.42*	*0.46*	0.43	0.46	0.46	0.46	0.46	0.46	0.43	0.46	0.44	0.46	0.44	0.47
	tr23	*0.48*	*0.57*	*0.51*	*0.59*	0.54	0.59	*0.54*	*0.60*	0.57	0.60	0.57	0.60	0.57	0.60	0.57	0.60
	tr31	0.48	0.49	*0.52*	*0.47*	0.50	0.46	0.50	0.54	*0.49*	*0.54*	*0.47*	*0.54*	0.52	0.54	0.52	0.54
	tr41	*0.50*	*0.43*	*0.40*	*0.46*	0.44	0.43	*0.44*	*0.38*	*0.44*	*0.38*	0.44	0.44	0.44	0.43	0.44	0.42
	tr45	0.43	0.43	0.41	0.38	0.39	0.38	*0.42*	*0.38*	0.39	0.38	0.37	0.38	0.38	0.38	0.40	0.38
	wap	0.37	0.40	0.37	0.41	0.38	0.38	0.38	0.40	0.37	0.38	*0.37*	*0.45*	0.37	0.40	0.37	0.43
	Winner in	4	3	3	4	2	1	6	1	3	1	2	3	2	1	3	2

Table 4. F score for 11 datasets for each of the vocabulary size.

	Vocabulary Sizes															
	Data	50		100		150		200		250		300		350		400	
		K	N	K	N	K	N	K	N	K	N	K	N	K	N	K	N
	k1a	1.24	1.14	*1.29*	*1.15*	1.29	1.23	*1.34*	*1.19*	1.18	1.23	1.17	1.15	1.20	1.17	1.26	1.24
	k1b	1.17	1.11	1.21	1.12	*1.26*	*1.12*	*1.28*	*1.13*	1.13	1.10	1.05	1.11	*0.94*	*1.14*	1.10	1.17
	re0	1.84	1.57	1.48	1.39	1.30	1.36	*1.17*	*1.36*	*1.26*	*1.42*	*1.26*	*1.40*	*1.26*	*1.43*	1.26	1.39
	re1	1.49	1.53	1.57	1.46	1.39	1.47	*1.33*	*1.50*	1.33	1.34	1.33	1.41	1.33	1.39	1.33	1.41
H	tr11	*1.33*	*1.10*	*1.34*	*1.08*	*1.34*	*1.08*	1.07	1.08	1.07	1.08	*1.07*	1.08	1.07	1.08	1.07	1.08
	tr12	1.36	1.30	1.43	1.38	1.46	1.36	*1.52*	*1.35*	*1.50*	*1.36*	1.48	1.36	*1.49*	*1.36*	1.51	1.33
	tr23	*1.99*	*1.40*	*2.06*	*1.42*	*2.14*	*1.44*	*1.79*	*1.38*	*1.86*	*1.42*	*1.86*	*1.42*	*1.80*	*1.42*	*1.79*	*1.37*
	tr31	1.60	1.58	1.55	1.42	1.35	1.27	*1.37*	*1.22*	*1.34*	*1.18*	1.22	1.22	1.23	1.21	1.23	1.21
	tr41	1.42	1.29	*1.17*	*1.30*	*1.06*	*1.25*	*1.06*	*1.55*	*1.06*	*1.83*	*1.06*	*1.39*	*1.06*	*1.94*	*1.06*	*1.91*
	tr45	*1.64*	*1.30*	*1.50*	*1.06*	*1.44*	*1.03*	*1.30*	*1.01*	*1.29*	*1.01*	*1.36*	*1.01*	*1.27*	*1.01*	*1.30*	*1.00*
	wap	*1.02*	*1.20*	*0.96*	*1.21*	*1.05*	*1.20*	*1.02*	*1.17*	*0.94*	*1.15*	*0.98*	*1.24*	*1.01*	*1.16*	*1.01*	*1.21*
	Winner in	4	1	4	2	4	2	6	4	4	3	2	3	3	4	3	2

Table 5. H score for 11 datasets for each of the vocabulary size.

A 10%[9] margin is used before calling one algorithm as winner over another. The total number of cases in which each algorithm based on each distribution wins is given in Table 6.

Measure	Normal	Katz's
F	16	25
H	21	30

Table 6. The number of experiments in which one algorithm performed better than the other by 10% or more. Number of clusters extracted was same was the number of clusters created by human observers. These are different for each dataset.

We can see from Table 6 that Katz's distribution based CLASSIT algorithm performs better than the Normal distribution based CLASSIT algorithm in more number of cases. Also, from Table 3, 4 and 5 we can see that Katz-CLASSIT performs better than the Normal-CLASSIT, as demonstrated by the F score, for the larger of the datasets.

8. The K column is the Katz-CLASSIT column and the N column is the Normal-CLASSIT column.

9. The margin of 10% is chosen as it seems a reasonable figure. Had we chosen some other margin, e.g. 5% or 15%, the win counts would change, but, the conclusion drawn would remain unaffected.

Compared to the batch clustering algorithm We compared our algorithm against a batch clustering algorithm using both F and H scores to observe the magnitude of the quality difference between these two types of algorithms.

Algorithm	Repeated Bisection followed by K-way refinement
Similarity Metric	Cosine
Criteria function to Optimize	maximize $\sum_{r=1}^{k} \sum_{d_i \in S_r} \cos(d_i, C_r)$
number of trials	10
number of iterations	10
Term weighting method	tfidf

Table 7. The setup for the Batch clustering experiments.

Table 7 shows the batch clustering parameters used in our experiments. These settings were selected because according to the experiments done in [34], these settings produce superior clusters.

For the incremental algorithms we use 100 words selected by the method described in Section 2.2.1. We compare the results of both original Classit algorithm and our modification of Cobweb using Katz's distribution assumption regarding attribute values.

Dataset	k	Batch	Classit	With Katz's distribution
k1a	20	0.62	0.41	0.48
k1b	6	0.74	0.58	0.75
re0	13	0.52	0.48	0.47
re1	25	0.52	0.36	0.38
tr11	9	0.74	0.50	0.42
tr12	8	0.71	0.46	0.42
tr23	6	0.60	0.59	0.51
tr31	7	0.61	0.47	0.52
tr41	10	0.76	0.46	0.40
tr45	10	0.73	0.38	0.41
wap	20	0.62	0.41	0.37

Table 8. Comparison against batch clustering algorithm using F score.

Dataset	k	Batch Clustering	Classit(Normal)	Classit(Katz)	Human Generated
k1a	20	1.53	1.15	1.29	1.41
k1b	6	1.31	1.12	1.22	1.36
re0	13	3.11	1.39	1.48	2.23
re1	25	3.97	1.46	1.57	3.35
tr11	9	1.73	1.08	1.34	1.62
tr12	8	1.63	1.38	1.43	1.62
tr23	6	3.19	1.36	2.06	1.83
tr31	7	2.48	1.42	1.55	2.13
tr41	10	2.27	1.30	1.17	2.17
tr45	10	2.57	1.07	1.50	2.27
wap	20	1.55	1.22	0.96	1.43

Table 9. Comparison against batch clustering algorithm using H score.

The batch clustering algorithm performs better than the incremental algorithms in both F and H score. This is to be expected due to the incremental nature of the algorithms, where each item is processed only once. On the other hand batch clustering algorithm processes each document multiple time and changes its assignment so that it improves the cluster quality. The clear drawback is that it requires all the documents to be available at the start of the clustering exercise, and this is not possible for the important application to news stories, Usenet postings and blogs that are identified in Section 1.1.

For comparison purpose, we have shown the H scores of human generated clusters in the table 9. Notice that the clusters generated by the batch clustering algorithm have a higher H-score than the human generated clusters, i.e. the clusters are more cohesive and separated from each other than the clusters produced by human subjects.

5.2 Experiments with Newswire articles in time order

The datasets used in Section 5.1 has traditionally been used for testing batch clustering algorithms. Batch clustering algorithms require all data points to be present at the start of the clustering exercise and involve multiple iterations over the dataset, often processing them in different order. Therefore, the ordering of the data points in the datasets does not influence the results of the batch clustering algorithm. Due to this reason we do not observe a natural ordering of the data points in the datasets used in Section 5.1. However, incremental clustering algorithms process the data points once in the order in which they are presented and the order in which data points are present in the dataset influences the clusters produced[10]. Therefore, it is imperative that we test the incremental clustering algorithms with an ordering of data points that is similar to the ordering they are expected to receive during their deployment. As we envision the two algorithms in this work to be used to process streams of text documents from newswire, newsgroups, BLOGs, etc., the natural ordering among the documents is determined by the time at which they are received. Therefore, we need a document dataset in which the time order of the documents is preserved. Reuters-RCV1[25], described more completely in the following section, is one such dataset.

5.2.1 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). These documents have been classified by editors at Reuters simultaneously under three category hierarchies: ‘‘Topic” hierarchy, ‘‘Industry” hierarchy and ‘‘Region” hierarchy. The Topic hierarchy contains four categories at the depth one of the tree, namely ‘‘Corporate/Industrial”, ‘‘Economics”, ‘‘Government/Social” and ‘‘Market”. There are ten such categories in the Industry hierarchy. Some of them are ‘‘Metals and Minerals”, ‘‘Construction”, etc. The Region hierarchy has geographical locations, such as country names, and economic/political groups as categories. There are no finer subcategories in the Region hierarchy.

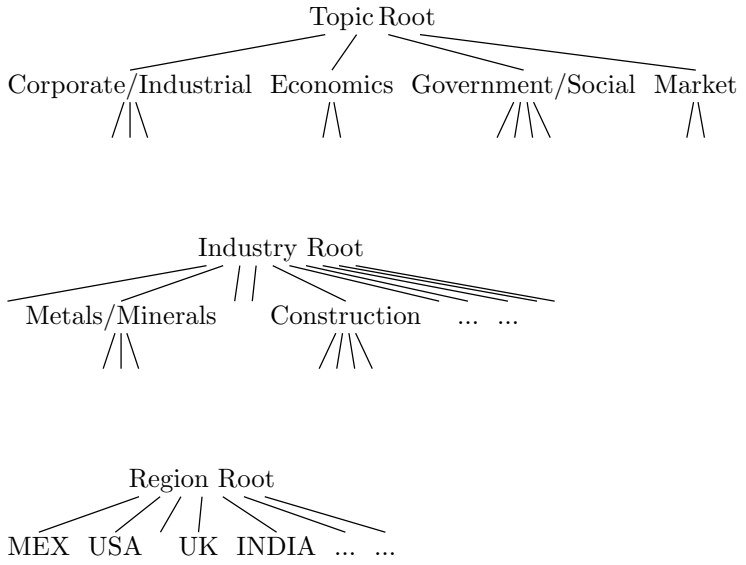


Figure 9. Three classification hierarchies.

10. However, the ideal incremental clustering algorithm is expected to be insensitive to the order in which it encounters the data points. Such, characteristic is partly achieved by the COBWEB algorithm by its *split* and *merge* operators.

The classification policy, also called The Coding Policy, requires that each document must have at least one Topic category and at least one Region category assigned to it. It also requires that each document be assigned to the most specific possible subcategory in a classification hierarchy. A document might be, and it often is, assigned more than one categories from any one of the three category hierarchies. The documents are present in the dataset in the order in time in which they were collected.

5.2.2 Experiment setup

For our experiments articles from the first two days of the Reuters-RCV1 dataset were used. There were 5107 articles. Stop words were removed from the documents and the terms were stemmed. Then the terms with highest $cf \cdot idf$ scores were selected, as described in Section 2.2.1, to represent the documents. We repeated the experiments using 200, 250, 300, 350 and 400 terms.

According to the categorization at the depth two of the Topic hierarchy they were classified into 52 categories. We have extracted 52 clusters from the dendograms constructed by the two algorithms and evaluated them using the Topic category labels attached to the documents.

We have also evaluated the clustering solutions using the region categories. There were 164 region categories present in the selected documents. So, we have extracted 164 clusters from the dendograms constructed by the clustering algorithms and measured their quality using the Region categories of the documents.

5.2.3 Results and Discussion

	Vocabulary Size									
	200		250		300		350		400	
F	K	N	K	N	K	N	K	N	K	N
Topic, k=52	0.60	0.40	0.63	0.40	0.63	0.35	0.62	0.36	0.64	0.37
Region k=164	0.48	0.36	0.49	0.28	0.47	0.30	0.51	0.31	0.50	0.31

Table 10. The extrinsic cluster quality comparison using both Topic and Region codes

	Vocabulary Size									
	200		250		300		350		400	
H	K	N	K	N	K	N	K	N	K	N
Topic, k=52	1.14	1.62	1.14	1.58	1.40	1.62	1.12	1.65	1.09	1.54
Region, k=164	1.73	2.07	1.93	2.12	2.01	2.18	1.50	2.13	1.92	2.08

Table 11. The intrinsic cluster quality comparison using both Topic and Region codes

We can see that Katz's distribution based Classit algorithm dominates Normal distribution based Classit algorithm across varying vocabulary sizes when we compare their F scores. The superior performance of Katz-Classit can be traced to the better fit of the Katz distribution to word occurrence data. We computed the likelihood of the word occurrences being generated by a Normal distribution, as we did for the other distributions in the Section 2.4.3. The likelihood for a word w is computed as:

$$L(w) \;=\; \prod_{i}^{N} \Pr(w \text{ occurs observed number of times in document } d_i)$$

For each word we got ≈ 0 likelihood. This is because one document containing the word relatively large number of times introduces a near zero factor in the likelihood computation and makes the likelihood for the word zero. In summary, Normal distribution is bad at modelling the large occurrences of a word in a relatively few documents. Katz calls it *burstiness* of word occurrence[23]. Katz distribution explicitly models this property of word occurrence and provides a more reasonable probability for large word occurrence values.

When we compare H scores we see that Normal-CLASSIT algorithm seems to perform better than the Katz-CLASSIT. The difference between the extrinsic cluster quality and the intrinsic cluster quality can be understood by observing that often when human observers classify a document as belonging to certain topic they do not take all the words in the document into consideration. Presence of a few keywords in a document is enough to make that document belong to a topic. However, in our similarity computation we take into account all words in the document, except a small list of stop words. Unfortunately, it is unclear at this point how to select words from a document and weight them as human observers would. One can select words that have strongly associated with the human assigned class labels of the documents to select such a subset of words, but, subsequent H scores would not remain intrinsic qualities of the documents as it is dependent on the labels provided by an external source.

As one of the motivations of using an incremental clustering algorithm is to be able to cluster documents as they are received, a measurement of time taken to cluster a document is needed. The average time required to cluster a document by each of the algorithms is given in Table 12.

	200	250	300	350	400
Katz-CLASSIT	0.25	0.26	0.27	0.37	0.37
Normal-CLASSIT	0.05	0.04	0.05	0.06	0.05

Table 12. Average time taken to cluster a document in seconds.

As the average time between arrival of two documents in the Reuters dataset is 39.42 seconds (800,000 documents collected in one year), we believe that both the algorithms are fast enough to cluster the documents in real time.

6 Conclusion

This is the first attempt of incremental hierarchical clustering of text documents to our knowledge. We have evaluated an incremental hierarchical clustering algorithm, which is often used with non-text datasets, using existing text document datasets and proposed its variation which is more appropriate for text documents.

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[23] and empirically observed in our work. We have demonstrated using 11 existing text clustering datasets that our incremental hierarchical clustering algorithm, which uses Katz's distribution, produces better clusters than the algorithm using Normal distribution, especially among the larger datasets. We have also used a new dataset, Reuters-RCV1, which allows us to carry out the experiments in a scenario very similar to the real life. Using Reuters-RCV1 dataset, when we tested the algorithms by presenting them Newswire articles in time order and have shown that our algorithm performs better than the Normal-CLASSIT algorithm as measured by the F score. This performance is consistent across different vocabulary sizes and across two categorization schemes: Topic categorization and Region Categorization. Both the clustering algorithms process documents fast enough to be deployed in a real life scenario.

This work can be extended by devising a more efficient way to compute the Category Utility when the attributes follow Negative Binomial distribution and by evaluating the resulting algorithm. Another useful extension to this work, one that we are working on, is estimating the parameters of the distributions in a Bayesian manner. This would improve the estimates of the parameters in many small clusters in the formed hierarchy.

Appendix A Time required for computing Negative Binomial based clustering exercise

While clustering Reuters-RCV1 dataset we observed that the Katz-CLASSIT algorithm has to compute Category Utility on an average at 985 ($\approx$ 1000) nodes for every document it routes through the tree. Assuming that we use 200 terms to represent the documents (this is one of the

settings used in our experiments), there will be approximately 200,000 computations of Category Utility components denoted as $\mathrm{CU}_{i,k}$. For category utility based on Negative Binomial It took us 64 seconds to compute these many $\mathrm{CU}_{i,k}$ (see Expression 31) using Mathematica. At the same time it took 0.05 seconds to compute 200,000 $\mathrm{CU}_{i,k}$ components for the Katz's distribution[11]. These are the average time required to compute Category Utilities for clustering one document.

The average time between arrival of two documents in the Reuters dataset is 39.42 seconds (800,000 documents collected in one year). Hence, a Classit algorithm based on Negative Binomial distribution will not be suitable for clustering such a dataset.

Appendix B MLE of Katz's distribution parameters

The Katz's distribution is defined as:

$$\begin{aligned} P(0) &= p_0 \\ P(k) &= (1-p_0)\,(1-p)\,p^{k-1}; \text{when } k>0 \end{aligned} \tag{38}$$

where, p_0 and p are the parameters of the distribution.

Let us discuss about the distribution of only one word or term. The data is the count of occurrences of the word in each document in the text collection. So, if we have N documents in the dataset we have N observations, each of which is a count.

Let us also define n_k to be the number of observations equal to k, i.e., number of documents in which the term occur k times. Let's assume the maximum value of k is K.

Hence,

- document frequency $\mathrm{df} = N - n_0 = \sum_{k=1}^{K} n_k$ and
- collection term frequency $\mathrm{cf} = \sum_{k=1}^{K} k\, n_k$

The likelihood of the parameters given data is

$$\begin{aligned} L(p,p_0) &= \prod_{i=1}^{N} \Pr(\text{the word occurs } x \text{ times in document } i) \\ &= \prod_{i=1}^{N} \left[\delta(x)\,p_0 + (1-\delta_k)\,(1-p_0)\,(1-p)\,p^{x-1}\right]; x \text{ can take values from } 1...K \\ &= p_0^{n_0} \prod_{k=1}^{K} (1-p_0)^{n_k}\,(1-p)^{n_k}\,(p^{k-1})^{n_k} \end{aligned}$$

where, $\delta(\,\cdot\,)$ is the indicator function that is 1 if argument is zero and 0 otherwise.

Log of likelihood is

$$\mathrm{LL}(p,p_0) = n_0 \log(p_0) + \sum_{k=1}^{K} \left[n_k \log(1-p_0) + n_k \log(1-p) + n_k\,(k-1)\log(p)\right]$$

Taking the partial derivative of the log likelihood with respect to p_0 and equating it to 0:

$$\begin{aligned} \frac{\partial \mathrm{LL}}{\partial p_0} &= \frac{n_0}{\hat{p_0}} + \sum_{k=1}^{K} n_k \frac{-1}{1-\hat{p_0}} = 0 \\ \Rightarrow \frac{n_0}{\hat{p_0}} &= \frac{1}{1-\hat{p_0}} \sum_{k=1}^{K} n_k = \frac{1}{1-\hat{p_0}}\,(N-n_0) \\ \Rightarrow \frac{1-\hat{p_0}}{\hat{p_0}} &= \frac{N-n_0}{n_0} \\ \Rightarrow \frac{1}{\hat{p_0}} - 1 &= \frac{N}{n_0} - 1 \\ \Rightarrow \hat{p_0} &= \frac{n_0}{N} = \frac{N-\mathrm{df}}{N} = 1 - \frac{\mathrm{df}}{N}; \text{ returning to the standard IR terminology} \end{aligned} \tag{39}$$

11. These experiments were conducted on a PC with a Pentium 4 (2.8 GHz) processor with 1GB RAM.

We can find the MLE of p in a similar manner.

$$\begin{aligned}
\frac{\partial \mathrm{LL}}{\partial p} &= \sum_{k=1}^{K} n_k \frac{-1}{1-\hat{p}} + \frac{n_k\,(k-1)}{\hat{p}} = 0 \\
\Rightarrow \frac{1}{\hat{p}} \sum_{k=1}^{K} n_k\,(k-1) - \frac{1}{1-\hat{p}} \sum_{k=1}^{K} n_k &= 0 \\
\Rightarrow \frac{1}{\hat{p}} \left(\sum_{k=1}^{K} k\,n_k - \sum_{k=1}^{K} n_k \right) - \frac{1}{1-\hat{p}} \sum_{k=1}^{K} n_k &= 0 \\
\Rightarrow \frac{1}{\hat{p}} (\mathrm{cf} - \mathrm{df}) - \frac{1}{1-\hat{p}}\,\mathrm{df} &= 0;\ \text{returning to the IR terminology} \\
\Rightarrow \frac{1-\hat{p}}{\hat{p}} &= \frac{\mathrm{df}}{\mathrm{cf}-\mathrm{df}} \\
\Rightarrow \frac{1}{\hat{p}} &= \frac{\mathrm{cf}}{\mathrm{cf}-\mathrm{df}} \\
\Rightarrow \hat{p} &= \frac{\mathrm{cf}-\mathrm{df}}{\mathrm{cf}} \qquad (40)
\end{aligned}$$

Expressions (39) and (40) are the MLE of the parameters of Katz's distribution described in Expression (38).

Bibliography

[1] Google News, http://news.google.com/.

[2] ODP - Open Directory Project, http://www.dmoz.org.

[3] Text retrieval conference (TREC).

[4] Vivisimo Clustering Engine, http://vivisimo.com/.

[5] Yahoo!, http://www.yahoo.com.

[6] Charu C. Aggarwal, Fatima Al-Garawi, and Philip S. Yu. On the design of a learning crawler for topical resource discovery. *ACM Trans. Inf. Syst.*, 19(3):286--309, 2001.

[7] James Allan, Ron Papka, and Victor 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.

[8] 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.

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

[10] Soumen Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan-Kauffman, 2002.

[11] P. Cheeseman and J. Stutz. Bayesian classification (AUTOCLASS): Theory and results. *Advances in Knowledge Discovery and Data Mining*, 1996.

[12] Junghoo Cho. *Crawling the web: discovery and maintenance of large-scale web data*. PhD thesis, 2002. Adviser-Hector Garcia-Molina.

[13] Douglass R. Cutting, David R. Karger, Pedersen Pedersen, and John 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.

[14] George Doddington, Jaime Carbonell, James Allan, Jonathan Yamron, Umass Amherst, and Yiming Yang. Topic detection and tracking pilot study final report, Jul 2000.

[15] M. A. T. Figueiredo and A. K. Jain. Unsupervised learning of finite mixture models. *IEEE Trans. on Patt. Analysis and Machine Intell.*, 24(3):381--396, March 2002.

[16] Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139--172, 1987.

[17] George Forman. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.*, 3:1289--1305, 2003.

[18] Martin Franz, Todd Ward, J. Scott McCarley, and Wei-Jing Zhu. Unsupervised and supervised clustering for topic tracking. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 310--317. ACM Press, 2001.

[19] David L. Wallace Frederick Mosteller. *Applied Bayesian and Classical Inference The case of The Federalist Papers*. Springer series in Statistics. Springer-Verlag, 1983.

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

[21] Eui-Hong Han, Daniel Boley, Maria Gini, Robert Gross, Kyle Hastings, George Karypis, Vipin Kumar, Bamshad Mobasher, and Jerome Moore. WebACE: A web agent for document categorization and exploration. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 408--415, New York, May 9--13, 1998. ACM Press.

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

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

[24] David D. Lewis. Reuters-21578,distribution 1.0.

[25] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361--397, 2004.

[26] Benoit Mandelbrot. *Scientific psychology; principles and approaches.*, chapter 29, pages 550--568. New York, Basic Books, 1965.

[27] Bhushan Mandhani, Sachindra Joshi, and Krishna Kummamuru. A matrix density based algorithm to hierarchically co-cluster documents and words. In *WWW '03: Proceedings of the twelfth international conference on World Wide Web*, pages 511--518. ACM Press, 2003.

[28] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, England, 2000.

[29] Tom M. Mitchell. *Machine Learning*, chapter 3, page 67. WCB/McGraw-Hill, 1997.

[30] Padhraic Smyth. Clustering Using Monte Carlo Cross-Validation. In Evangelos Simoudis, Jia Wei Han, and Usama Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, page 126. AAAI Press, 1996.

[31] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In Douglas H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 412--420, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.

[32] Yiming Yang, Tom Pierce, and Jaime Carbonell. A study of retrospective and on-line event detection. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 28--36. ACM Press, 1998.

[33] Ya-Jun Zhang and Zhi-Qiang Liu. Refining web search engine results using incremental clustering. *International journal of intelligent systems*, 19:191--199, 2004.

[34] Ying Zhao and George Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Mach. Learn.*, 55(3):311--331, 2004.

[35] George Kingsley Zipf. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Addison-Wesley Press Inc., Cambridge 42, MA, USA, 1949.