Incremental Hierarchical Clustering of Text Documents

by Nachiketa Sahoo

Adviser: Jamie Callan

May 5, 2006

Abstract

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

1 Introduction

Document clustering is an effective tool to manage information overload. By grouping similar documents together, we enable a human observer to quickly browse large document collections[6], make it possible to easily grasp the distinct topics and subtopics (concept hierarchies) in them, allow search engines to efficiently query large document collections [16] among many other applications. Hence, it has been widely studied as a part of the broad literature of data clustering. One such survey of existing clustering literature can be found in Jain et. al[13].

The often studied document clustering algorithms are batch clustering algorithms, which require all the documents to be present at the start of the exercise and cluster the document collection by making multiple iterations over them. But, with the advent of online publishing in the World Wide Web, the number of documents being generated everyday has increased considerably. Popular sources of informational text documents such as Newswire and Blogs are continuous in nature. To organize such documents naively using existing batch clustering algorithms one might attempt to perform clustering on the documents collected so far. But, this is extremely time consuming, if not impossible, due to the sheer volume of documents. One might be tempted to convert the existing batch clustering algorithms into incremental clustering algorithms by performing batch clustering on periodically collected small batches of documents and then merge the generated clusters. However, ignoring for the moment the problem of deciding on an appropriate time window to collect documents, there will always be a wait time before a newly generated document can appear in the cluster hierarchy. This delay would be unacceptable in several important scenarios, e.g., financial services, where trading decisions depend on breaking news, and quick access to appropriately classified news documents is important. A clustering algorithm in such a setting needs to process the documents as soon as they arrive. This calls for the use of an *incremental* clustering algorithm.

There has been some work in incremental clustering of text documents as a part of Topic Detection and Tracking initiative ([1], [19], [10] and [7]) to detect a new event from a stream of news articles. But, the clusters generated by this task are not hierarchical in nature. Although, that was adequate for the purpose of new event detection, we believe this is a limitation. The benefits of 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. But, if we present the user with a topic hierarchy populated with documents, which she can browse at her desired level of specificity, we would circumvent the problem of finding the right number of clusters while generating a solution that would satisfy users with different needs.

In spite of potential benefits of an incremental algorithm that can cluster text documents as they arrive into a informative cluster hierarchy, this is a relatively unexplored area in text document clustering literature. In this work we examine a well known incremental hierarchical clustering algorithm COBWEB that has been used in non-text domain and its variant CLASSIT. We discuss why they are not suitable to be directly applied to text clustering and propose a variant of these algorithm that is based on the properties of text document data. Then we evaluate both the algorithm using real world data and show the gains obtained by our proposed algorithm.

1.1 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 algorithm for text document clustering where word occurrence attributes follow Katz's distribution.
2. Evaluation of the existing algorithms and our proposed algorithm on large real world document datasets.

In Section 34 we briefly review the text clustering literature. In Section 3 we describe the COBWEB and CLASSIT algorithms. In Section 3 we describe key properties of text documents that are central to this work. In Section 4 we explain the contributions of our work. In Section 5 we describe the cluster quality metrics that we have used to evaluate the results obtained. In Section 6 we explain the setup of the experiment and discuss the results. In Section 7 we conclude with scope for future research.

2 Literature review

Clustering is a widely studied problem in the Machine Learning literature [13]. 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[17]. 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.

Clustering to browser large document collections (Scatter/Gather)

Cutting et al.[6] 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 is 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 of k clusters [17]. Note that k is a fixed user provided number. 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, 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. 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 [18]. The approaches suggested in the literature can, in general, be divided into two groups [4]. 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[5]. Other such works include Bayesian Information Criteria and Minimum Descriptor Length criteria [8]. A different approach has been suggested in Liu et al.[16] 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 from each other 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 ([17],[13]).

Incremental document clustering

As part of Topic Detection and Tracking (TDT) initiative ([1], [19], [10] and [7]) 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[1]. 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 $0.4 + p \times (\mathrm{sim}(q_i, d_i) - 0.4) + \mathrm{tp} \times (j - i)$, 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 [20]. 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[2], 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, which foregoes the opportunity to use clustering as an aid to detect topic and subtopic structure within a large document collection. Also, TDT experiments effectively exploit the information in the time stamp available with news stories, i.e., assumes that news stories that describe the same event will occur within a brief span of time. Such information may not always be available.

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 [9] and its derivative Classit [12]. 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

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

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 $A_1 \in$ {male, female}, $A_2 \in$ {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 [9])

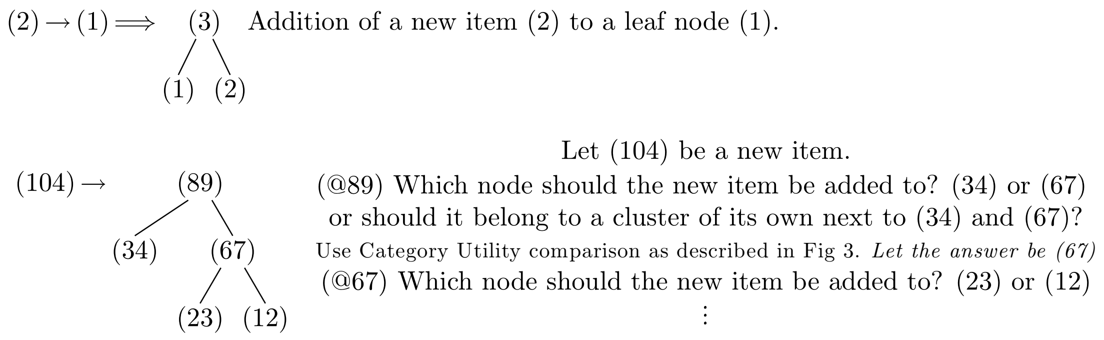
```
function COBWEB(item, root)
Update the attribute value statistics at the root
If root is a leaf node then
  Return the expanded node that accommodates the new Object
else
  Find the best child of the root to host the item and perform the
    qualifying step (if any) among the following:

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 1. COBWEB control structure.

An illustration of the clustering process is given in Figure 2.



Addition of a new item (2) to a leaf node (1).

Let (104) be a new item.
(@89) Which node should the new item be added to? (34) or (67)
or should it belong to a cluster of its own next to (34) and (67)?
Use Category Utility comparison as described in Fig 3. *Let the answer be (67)*
(@67) Which node should the new item be added to? (23) or (12)
⋮

Figure 2. 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 2).

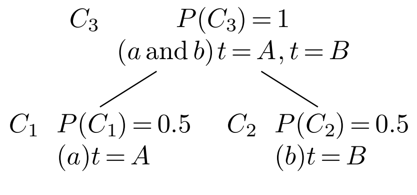


Figure 3. After first two items are added.

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):

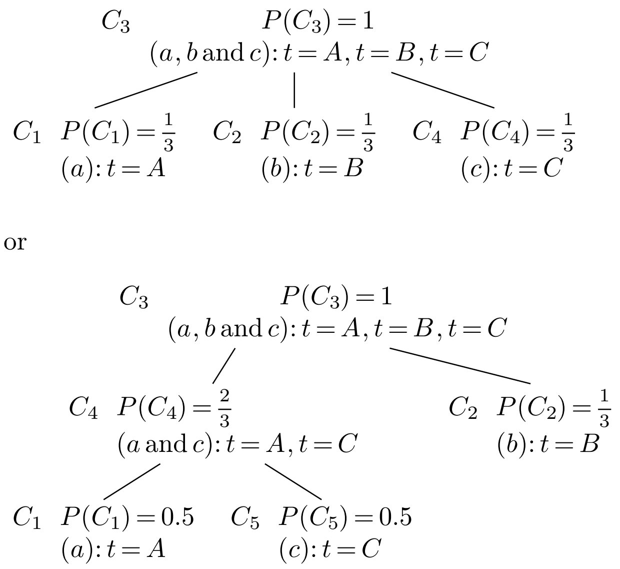

C_3 $P(C_3)=1$
$(a, b \text{ and } c)$: $t=A, t=B, t=C$
C_1 $P(C_1)=\frac{1}{3}$
(a): $t=A$
C_2 $P(C_2)=\frac{1}{3}$
(b): $t=B$
C_4 $P(C_4)=\frac{1}{3}$
(c): $t=C$
or
C_3 $P(C_3)=1$
$(a, b \text{ and } c)$: $t=A, t=B, t=C$
C_4 $P(C_4)=\frac{2}{3}$
$(a \text{ and } c)$: $t=A, t=C$
C_2 $P(C_2)=\frac{1}{3}$
(b): $t=B$
C_1 $P(C_1)=0.5$
(a): $t=A$
C_5 $P(C_5)=0.5$
(c): $t=C$

Figure 4. 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 4 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:

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

For the second configuration in Figure 4 the parent cluster is C_3 and the child clusters are C_4 and C_2.

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

Since, $\mathrm{CU}^1 > \mathrm{CU}^2$ we select configuration 1 over configuration 2. Looking at the Figure 4, 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 4. 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 [9]. 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.

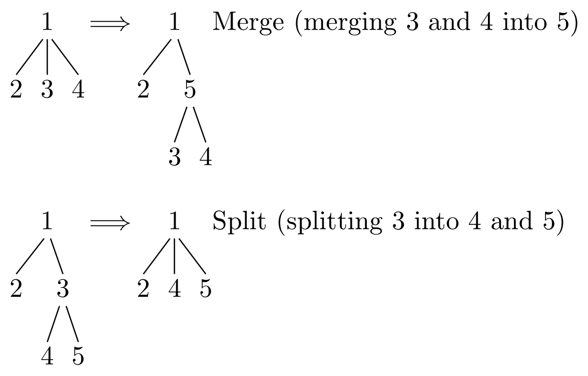

Merge (merging 3 and 4 into 5)
Split (splitting 3 into 4 and 5)

Figure 5. 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.[12] 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 text clustering. Hence, their performance on text document data is uncertain at the time of this work. Further, word occurrence counts, attributes of text documents that are commonly used to represent a document, follow a skewed distribution–-unlike the Normal distribution (Figure 6). Also, Normal distribution assumes that the attributes are Real numbers, but, word occurrence counts are Nonnegative Integers. They can not be treated as nominal attributes either, because the occurrence counts are not contained in a bounded set, which one would have to assume while treating them as nominal attributes. A more suitable distribution for such count data is Negative Binomial, or Katz's distribution [14].

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

3 Text Documents and word distributions

Text, as we commonly know it, is available in the form of unstructured documents. Before we can use such documents for classification or clustering, we need to convert them to items with attributes and values. A popular 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 foregoes information contained in the order of the word. Despite this drawback, the bag-of-words approach is one of the most successful and widely used method of converting text documents into actionable form.

Several attempts has been made to characterize the distribution of words across documents. This is useful in judging the information content of a word. For instance a word that occurs uniformly in every document of the corpus, e.g., “the” is not as informative as a word that occurs frequently in only a few, e.g., “Zipf”.

Occurrence 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 documents of similar topic into same group–- as is done in this work. Manning and Schutze have discussed several models to characterize the occurrence of words across different documents [17].

3.1 Models based on Poisson distribution

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

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

where, λ is a rate parameter. However, from empirical observations, it has been found that Poisson distribution tends to over estimate the frequency of informative words (content words) [17].

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

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

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[3], a spurious drop is seen for the probability of a word occurring twice in a document[14]. The fitted distribution has lower probability for a word occurring twice in a document than it occurring three times, i.e., it predicts that there are fewer documents that contain a word twice than there are documents that contain the same word three times. But, empirically it has been observed that document count monotonically decreases for increasing number of occurrences of a word (see Figure 6).

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”.

3.1.3 Negative Binomial A proposed solution to the above problem is to use a mixture of more than two Poisson distributions to model the word occurrences. A natural extension of this idea is to use a Negative Binomial distribution, which is a gamma mixture of infinite number of Poisson distributions[11]. The probability density functions of a Negative Binomial distribution is given below,

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

where p and r are parameters of the distributions.

Although the Negative Binomial distribution fits the word occurrence data very well it can be hard to work with because it often involves computing a large number of coefficients[17]. This has been confirmed in our analysis (see Expressions (28) and (29) in Section 4.2).

3.1.4 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 6). Looking at the shape of the empirical probability density function we attempt 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.

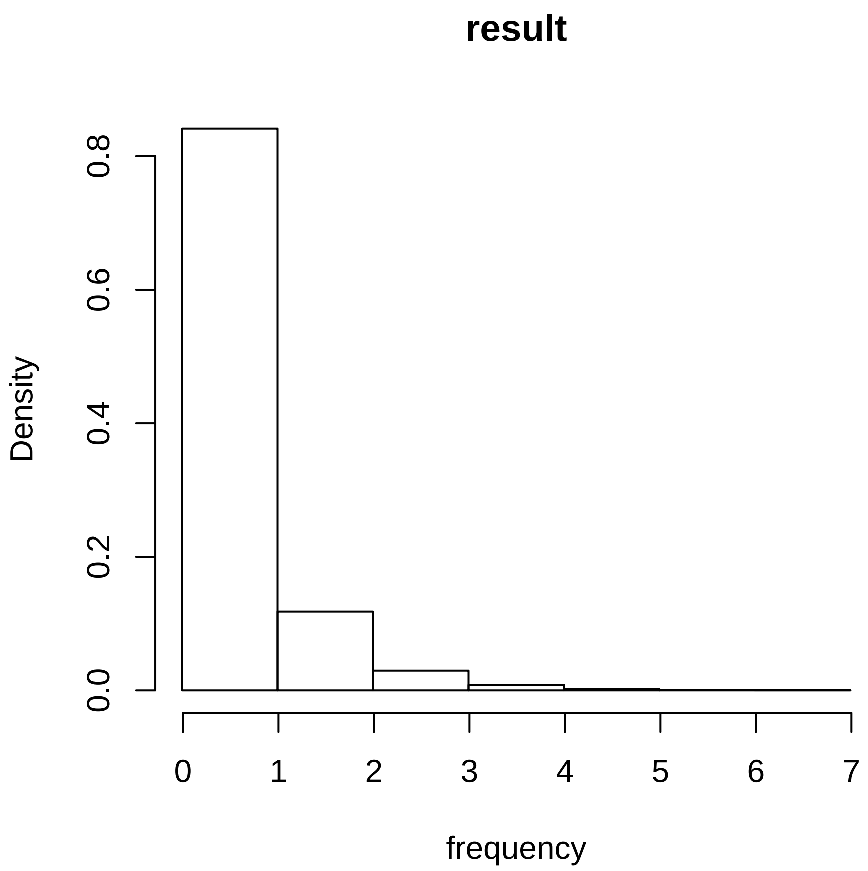


Figure 6. The occurrence of a typical word ("result") across different documents in our test collection.

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

$$P(k) = (1-\alpha)\delta_k + \alpha \frac{\lambda^k e^{\lambda}}{\lambda!}, k = 0, 1, 2... \tag{5}$$

where,

$$\delta_k = \begin{cases} 1, \mathrm{iffk} = 0 \\ 0, \mathrm{otherwise} \end{cases}$$

As we shall demonstrate in Section 3.3, this distribution does not fit text data as well as the Negative Binomial or the Katz's distribution.

3.2 Katz's K-mixture model

This distribution, proposed by Katz[14], 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[17]. Katz's distribution assigns the following probability to the event that word i occurs k times in a document[2].

$$P(k) = (1-\alpha)\delta_k + \frac{\alpha}{\beta+1}\left(\frac{\beta}{\beta+1}\right)^k \tag{6}$$

$\delta_k = 1 \mathrm{iffk} = 0 \mathrm{and} 0 \mathrm{otherwise}$.

The MLE estimates of parameters α and β are:

$$\beta = \frac{\mathrm{cf} - \mathrm{df}}{\mathrm{df}} \tag{7}$$

$$\alpha = \frac{1}{\beta} \times \frac{\mathrm{cf}}{N} \tag{8}$$

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 (6) it follows that

$$\begin{aligned} P(0) &= 1 - \alpha + \frac{\alpha}{\beta+1} \\ &= 1 - \frac{\mathrm{df}}{N} \\ &= 1 - \Pr(\text{the word occurs in a document}) \\ &= \Pr(\text{the word does not occur in a document}) \end{aligned} \tag{9}$$

Also, it follows that

$$P(k) = \frac{\alpha}{\beta+1}\left(\frac{\beta}{\beta+1}\right)^k, k = 1, 2, \ldots \tag{10}$$

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

$$P(k) = \alpha(1-p)p^k \tag{11}$$

Let's define a parameter p_0 as

$$p_0 = P(0) \tag{12}$$

using (7) we find that

$$\begin{aligned} p &= \frac{\frac{\mathrm{cf}-\mathrm{df}}{\mathrm{df}}}{\frac{\mathrm{cf}}{\mathrm{df}}} \\ &= \frac{\mathrm{cf} - \mathrm{df}}{\mathrm{cf}} \\ &= \frac{\Pr(\text{the word repeats in a document})}{\Pr(\text{the word occurs in a document})} \\ &= \frac{\Pr(\text{the word repeats} \bigcap \text{ the word occurs})}{\Pr(\text{the word occurs})} \\ &= \Pr(\text{the word repeats} \mid \text{the word occurs}) \end{aligned} \tag{13}$$

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

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 (11) and (12) together a valid probability density function.

We can write Expression (6) 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

$$\begin{aligned} p_0 &= 1-\alpha p \\ \Rightarrow \alpha p &= 1-p_0 \\ \Rightarrow \alpha &= \frac{1-p_0}{p} \end{aligned} \tag{14}$$

Expression (11) can now be written as

$$P(k)=(1-p_0)(1-p)p^{k-1} \tag{15}$$

when $k>0$.

Using Expressions (12) and (15), we can fully specify the Katz's distribution. The two parameters are p_0 and p, which can be estimated as (see Expressions 9 and 13)

$$\hat{p}_0 = 1-\frac{\mathrm{df}}{N} \tag{16}$$

and

$$\hat{p} = \frac{\mathrm{cf}-\mathrm{df}}{\mathrm{cf}} \tag{17}$$

It can be shown that if a distribution is defined by Expressions (12) and (15), then the estimates (16) and (17) are the MLE of the parameters p_0 and p (see Appendix A).

3.3 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.

3.3.1 Zero Inflated Poisson If the probability density function of a Zero Inflated Poisson distribution is given in the form of Expression (5), then the method of moment estimates of its parameters α and λ are

$$\hat{\lambda} = \frac{\mathrm{Var}(X)}{X} + \overline{X} - 1 \tag{18}$$

and

$$\hat{\alpha} = \frac{X}{\lambda} \tag{19}$$

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

$$\hat{r} = \frac{\bar{X}^2}{\mathrm{Var}(X)-\bar{X}} \tag{20}$$

$$\hat{p} = \frac{\bar{X}}{\mathrm{Var}(X)} \tag{21}$$

For the Katz's distribution we used Expressions (16) and (17) to estimate the parameters p_0 and p.

We evaluated the fitness of these three distributions by computing the probabilities of the word occurrences using the estimated parameters, on three different datasets. For each dataset we selected the top 100 terms by their cf $\times \log(N/df)$ score. The distribution that has a higher likelihood than 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.

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

Table 1. Likelihood comparisons, count of likelihood of row distribution $>$ likelihood of col distribution $\times$ 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 4, it also fits better than Zero Inflated Poisson (ZIP) and gives fitness comparable to Negative Binomial (NB) distribution.

4 Algorithms for text

4.1 COBWEB: when attribute values follow Katz's distribution

4.1.1 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

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

Notice that for each attribute indexed i we need to compute

$$\sum_j \left(P(A_i = V_{i,j}|C_k)^2 - P(A_i = V_{i,j}|C_p)^2\right) \tag{22}$$

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

Hence, the first part of Expression (22) can be written as

$$\mathrm{CU}_{i,k} = \sum_{f=0}^{\infty} P(A_i = f|C_k)^2 \tag{23}$$

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 (12) and (15) in Expression (23), we obtain

$$\mathrm{CU}_{i,k} = \sum_{f=0}^{\infty} P(A_i = f|C_k)^2 = \frac{1 - 2p_0(1 - p_0) - p(1 - 2p_0)}{1 + p} \tag{24}$$

Substituting estimates of p_0 and p from Expressions (16) and (17) in Expression (24), and simplifying, we get

$$\mathrm{CU}_{i,k} = \sum_{f=0}^{\infty} P(A_i = f|C_k)^2 = 1 - \frac{2 \times \mathrm{df}\left(N - \frac{\mathrm{cf} \times \mathrm{df}}{2 \times \mathrm{cf} - \mathrm{df}}\right)}{N^2} \tag{25}$$

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

Expression (25) 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

$$\mathrm{CU}_p = \frac{1}{K} \sum_k P(C_k) \left[\sum_i \mathrm{CU}_{i,k} - \sum_i \mathrm{CU}_{i,p} \right] \tag{26}$$

where, $\mathrm{CU}_{i,k}$ is given by Expression (25).

4.2 Cobweb: when attribute values follow Negative Binomial distribution

The probability density function of the Negative Binomial distribution is

$$P(x) = \begin{pmatrix} x + r - 1 \\ r - 1 \end{pmatrix} p^r (1 - p)^x \tag{27}$$

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

4.2.1 Category utility

Substituting Expression (27) in (23), we obtain the contribution of a word in a child cluster towards Category Utility

$$\mathrm{CU}_{i,k} = \sum_{x=0}^{\infty} \left(\frac{(x + r - 1)!}{x!(r - 1)!} p^r (1 - p)^{x-1} \right)^2 \tag{28}$$

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

$$\mathrm{CU}_{i,k} = \frac{p_2^{2r} F_1\left(r, r, 1, (1 - p)^2\right)}{(1 - p)^2} \tag{29}$$

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 (25), the equivalent expression for Katz's distribution. As we described in Section 3.3, 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.

For this reason and the reasons cited in Section 3.1 and 3.3, we shall fully explore only Katz's distribution and original CLASSIT algorithm based on Normal distribution in our work.

5 Cluster Evaluation Methods

5.1 Evaluating the clusters

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

$$p_k = \frac{\max_c \{\mathrm{CF}_k(c)\}}{N_k} \tag{30}$$

where,

- c is the index of classes
    - *class* is a pre-specified 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.

N_k = number of items in class k.

Purity of the entire collection of clusters can be found by taking the average of the cluster qualities. Here, there are two kinds of averages one might consider: weighted or unweighted. If we assign a weight to each cluster proportional to the size of the cluster and take the weighted average then it is called *micro average*, since each of the documents get equal weight. If we instead want to give equal weight to each cluster, we compute the arithmetic average instead. This is called *macro average*. The first one is a document level evaluation, while the second one is a cluster level evaluation. Both these purity are greater than 0 and less than 1.

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.

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

where, N_c is the size of the class c. The other variables are as defined for the expression of the purity score in Expression (30). Here, also we can compute the micro average or the macro average to compute the score for the entire solution.

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

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

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

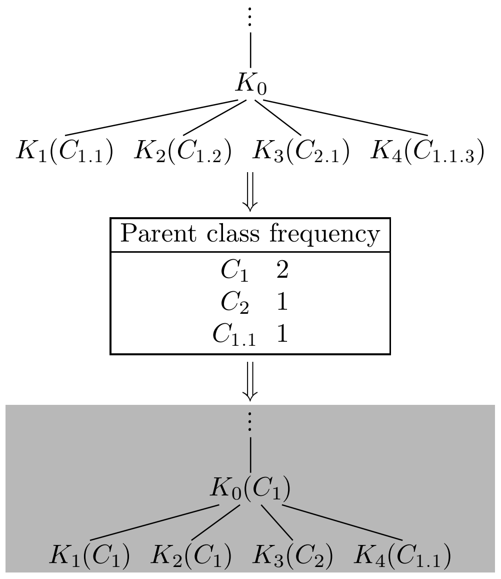
$$\frac{1}{F} = \frac{1}{2}\left(\frac{1}{P} + \frac{1}{R}\right) \tag{31}$$

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

5.2 Evaluating the hierarchy

Another question of interest when evaluating a hierarchical clustering algorithm is "To what extent the generated cluster hierarchy agree with the class hierarchy present in the data?". As we shall describe in Section 6, the datasets we have used in our experiments have a hierarchy of classes and provide us a rare opportunity to evaluate our generated cluster hierarchy for correctness. As a reminder, a *class* is a document category that has been provided to us as a part of the dataset. It is what the documents have been labeled with by an external entity and help us in evaluating how good our algorithm is. On the other hand, a *cluster* is a grouping of documents that our algorithm generates. It does so by grouping together the documents it considers similar.

Matching the generated cluster hierarchy with the existing class hierarchy is a non-trivial task. In stead, in this work we focus on measuring how often the sibling clusters in the generated hierarchy have sibling classes, i.e, how often children clusters of a parent cluster have children classes of the class that is assigned to the parent cluster. For instance, consider the generated cluster subtree shown in Figure 7.



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

Figure 7. A sample subtree with the children nodes. Class labels of the children node are given in parenthesis.

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

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

We needed to make a few decisions while evaluating the hierarchy in this manner. For instance, we used only the internal nodes to compute the precision of any node. This is because, often times leaf nodes co-exist with internal nodes as children of another internal node. In this case if we compute precision based on leaf nodes, i.e., single documents, then we are mixing the precision of the kind we described in Section 5.1 with the precision of the hierarchy and it is not clear how we should interpret the resulting number. Another decision that needed to be made was, what should we do if a child cluster has the broadest class label assigned to it? Since, we can not find a parent class for these classes, we explored the possibility of

i. dropping such child clusters from our evaluation and

ii. treating them as their own parent cluster since, they are the broadest level classes.

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

In our experiments the results do not change much if we take either of these strategy. So, we shall report only the results we got by treating the broadest classes as their own parent classes.

6 Experiment setup and results

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

6.1 Reuters-RCV1

Incremental clustering algorithms process the data points only once and in the order in which they are presented and the order in which data points are present in the dataset influences the clusters produced[4]. Therefore, it is imperative that we test the incremental clustering algorithms with an ordering of data points that is similar to the what 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[15] is one such dataset.

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 sub-categories in the Region hierarchy.

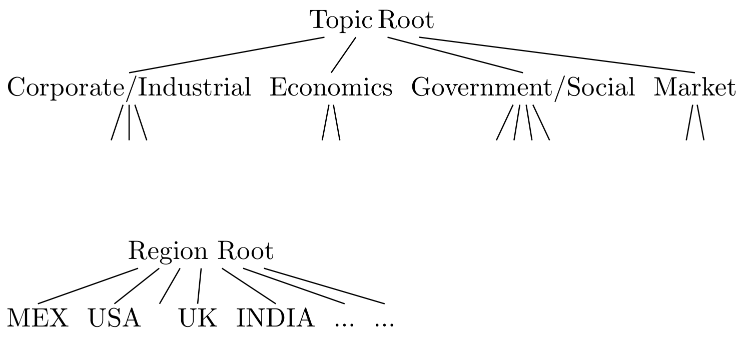


Figure 8. Three classification hierarchies.

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 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.

4. 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.

number of documents	62935
number of unique words	93792
average document length	222
number of classes	259

Table 2. rcv1 dataset (First 30 days). Classes are the region classes

6.1.1 Evaluating clusters

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

We have evaluated the clustering solutions for the correctness of assignment of documents to the clusters using the region categories, because (i) in the region class hierarchy all the assigned classes belong to one level and (ii) fewer articles are assigned multiple region class labels than they are assigned other class labels, suggesting that the region classes in the dataset do not overlap a lot. This allows us to evaluate out algorithm on a dataset with well defined classes. There were 259 region categories present in the selected documents. So, we have extracted 259 clusters from the dendrogram constructed by the clustering algorithms and measured their quality using the Region categories of the documents.

Results and Discussion The results of the clustering exercise is given in Table 3. We can see that Katz's distribution based Classit algorithm dominates Normal distribution based Classit algorithm across varying vocabulary sizes in both the micro and macro average of F scores.

			K	N
V	100	micro	0.46	0.31
		macro	0.83	0.60
			K	N
	200	micro	0.45	0.43
		macro	0.81	0.74
			K	N
	300	micro	0.45	0.33
		macro	0.85	0.67
			K	N
	400	micro	0.45	0.42
		macro	0.79	0.74
			K	N
	500	micro	0.45	0.36
		macro	0.84	0.69
			K	N
	600	micro	0.45	0.42
		macro	0.82	0.76
			K	N
	700	micro	0.45	0.39
		macro	0.81	0.74
			K	N
	800	micro	0.45	0.30
		macro	0.83	0.61

Table 3. Cluster quality comparison on RCV1 data

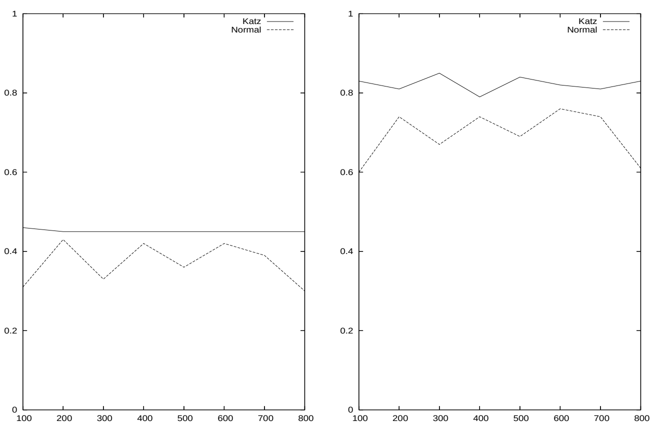


Figure 9. Cluster quality comparison on RCV1 data. The left panel shows the micro average of F-score and the right panel shows the macro average of the F-score.

As we can see Katz based Classit algorithm consistently performs better than the Normal based Classit algorithm on this dataset. However, we are cautious in interpreting the micro averaged-F score. Both of these algorithms produce clusters of widely different sizes, i.e., a few big clusters, a few more clusters of intermediate size and a lot of smaller clusters. The micro-averaged F score, is affected by it. Because, performance over a few good clusters dominates the entire performance metric. This explains the flat nature of the plot of micro averaged F score with Katz based Classit. The larger of the clusters generated by the algorithm do not change much over different vocabulary sizes, so, the micro-averaged F score remains nearly constant. Therefore, we also compute the macro-averaged F score, where each cluster gets equal weight, and find that Katz based Classit performs better than Normal based Classit over a wide range of vocabulary sizes.

6.1.2 Evaluating hierarchy

We evaluate the generated cluster hierarchy using the *topic* hierarchy of classes[5] as our reference. There are 63 different topic codes in the documents we used, where as in the entire topic hierarchy there are 103 topic codes.

We pre-processed the documents using the steps described in the previous section. Evaluated the accuracy of the parent/child cluster configurations as described in Section 5.2. The results are given in Table 4.

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

Table 4. Evaluation of the cluster hierarchy using Rcv1 data

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

5. This can be obtained from [15] Appendix 2.

6.2 OHSUMED (88-91)

The Ohsumed test collection is a set of 348,566 abstracts collected from 270 medical journals over a period of 5 years. Each abstract is annotated with MeSH (Medical Subject Heading) labels by human observers. This indicates the topic of the abstract. Unlike the rcv1 dataset, these documents are not in temporal order. Another property of this dataset is, being from a specific subject area, they contain words from a much smaller vocabulary. Due to the presence of human assigned MeSH keywords over such a large collection, this dataset provides us with an opportunity to evaluate our algorithm over a large dataset and against real topic labels.

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

Table 5. Ohsumed dataset (88-91)

6.2.1 Evaluating clusters

Experiment Setup We used the Ohsumed 88-91 dataset from the TREC-9 filtering track to evaluate our algorithm for the correctness of assignment of documents to the classes. We selected only those articles for which both the MeSH labels and the abstract text were present. There were 196,555 such articles. As with the rcv1 dataset most informative words in the dataset were selected using $\mathrm{cf} \times \log\left(\frac{N}{\mathrm{df}}\right)$ score of the words. We repeated the clustering exercise using 25 to 200 words at a step size of 25. To determine the number of different topics present in this dataset one can look at the unique MeSH labels present in the dataset. But, as there are tens of thousands of such labels present we used fixed number of clusters to evaluate (see Table 6) the algorithms.

Results and discussion The F-score results of the experiments are given in Table 6.

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

Table 6. Cluster quality comparison on OHSUMED data at different number of clusters (k) and vocabulary size (V). The figures in the table are F-score × 100. K stands for Katz-Classit, N for the original Classit. μ and M row in the smallest table hold the micro and macro average of the F-score respectively. The cells where Katz-Classit performs better are marked with a ✓, the cells where Normal-Classit performs better are marked with a × and the cells where there is no clear winner are marked with a ?. Best Katz-Classit and the best Normal-Classit have been highlighted by grey cells.

We can see from the table that Normal-Classit is the most competitive when the vocabulary size is small *and* the number of clusters formed is large. For all other settings, i.e., when the size of the vocabulary used is larger or when the number of clusters formed is smaller, Katz-Classit performs better. This shows that the Katz-Classit algorithm is more robust as it performs well across a much larger range of parameter values.

Performances of both the algorithms suffer when we create more number of clusters, which makes sense because, there are fewer features based on which to distinguish between clusters.

6.2.2 Evaluating hierarchy

MeSH labels present in the Ohsumed collection has a hierarchical structure to it[6]. This provides us with another opportunity to evaluate the correctness of our hierarchy. This class hierarchy is much larger than the *topic* hierarchy of Rcv1 dataset. There are 42610 different MeSH labels. Each MeSH label has a code attached to it. The class hierarchy information can be directly read from this code. For instance the first three records of 2005 "ASCII MeSH collection" reads

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

Figure 10. First three lines of MeSH labels file (filename: mtrees2005.bin)

This says that the topic labeled "Abdominal Cavity" (*A01.047*.025) is a child topic of label with code A01.047, which we can find from the file as the topic "Abdomen" (*A01*.047), which in turn is a child topic of a topic with code A01. We can find from the file that this is the code of the label "Body Regions". This "." separated topic codes let us easily find the parent topics by dropping the suffix of the code. Not all the MeSH labels are seen in our dataset. There were only about 14138 different MeSH labels used in document set we used for our experiments.

Documents were pre-processed as described in the previous section. Entire cluster hierarchy was generated and the correctness of the hierarchy was evaluated as described in Section 5.2. The precision values are reported in table

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

Table 7. Evaluation of the cluster hierarchy using Ohsumed data

Here again both the algorithms do reasonably well in grouping classes with common parents under the same cluster with Katz-Classit seems to have an advantage over Normal-Classit

6. The entire collection of MeSH labels can be downloaded from the web-site of National Institute of Health (http://www.nlm.nih.gov). We have used 2005 MeSH label collection for our purpose.

across all vocabulary sizes. But, we must be careful here not to interpret these precision values as closeness of the entire cluster hierarchy to the existing class hierarchy. Instead it is the accuracy of the algorithms in classifying sibling classes under same parent cluster.

We also tracked the sibling precision score at different depths of the generated cluster tree (Figure 11 and 12).

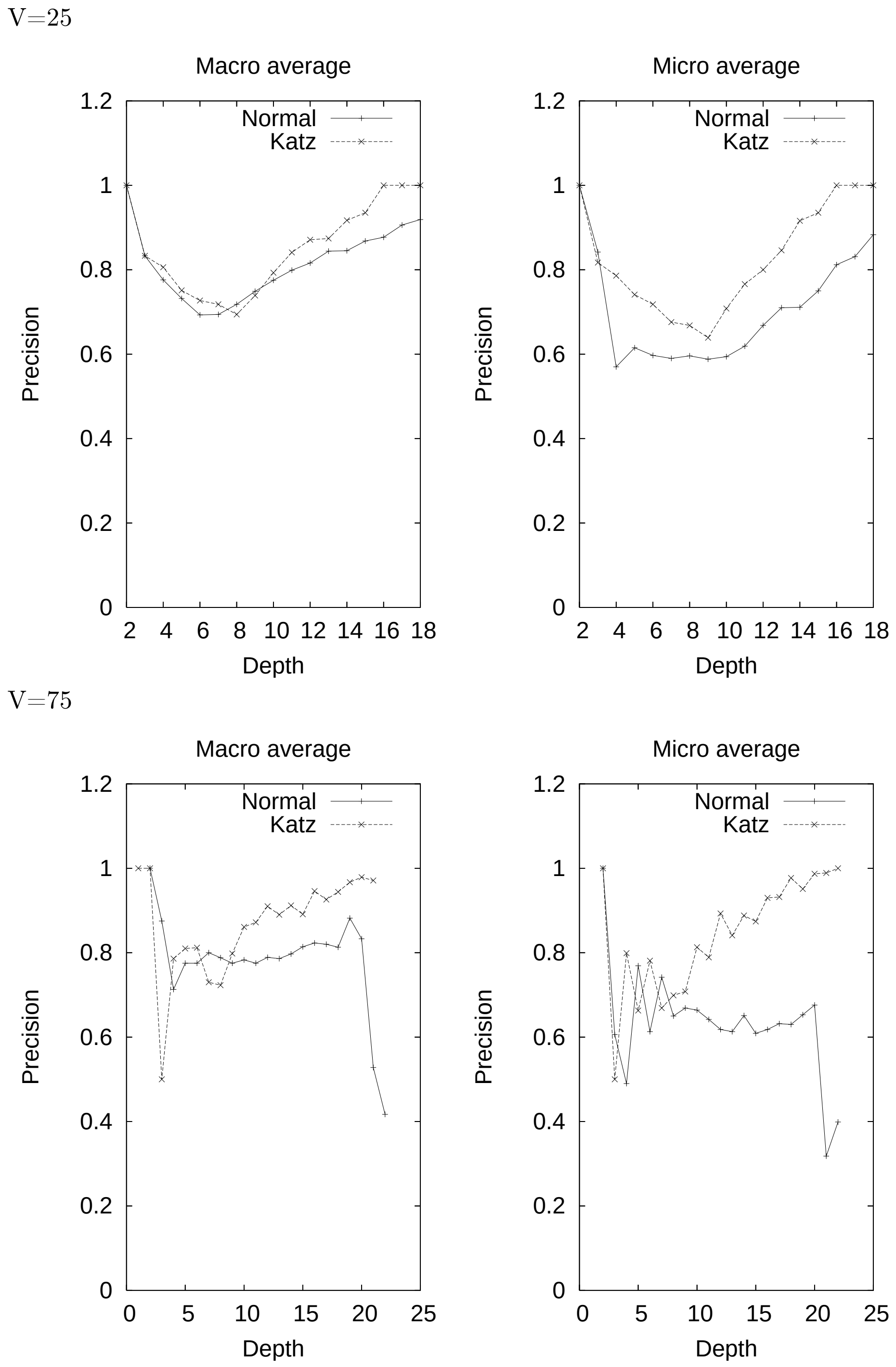


Figure 11. Tracing the sibling precision over the height of the tree. Vocabulary 25 and 75.

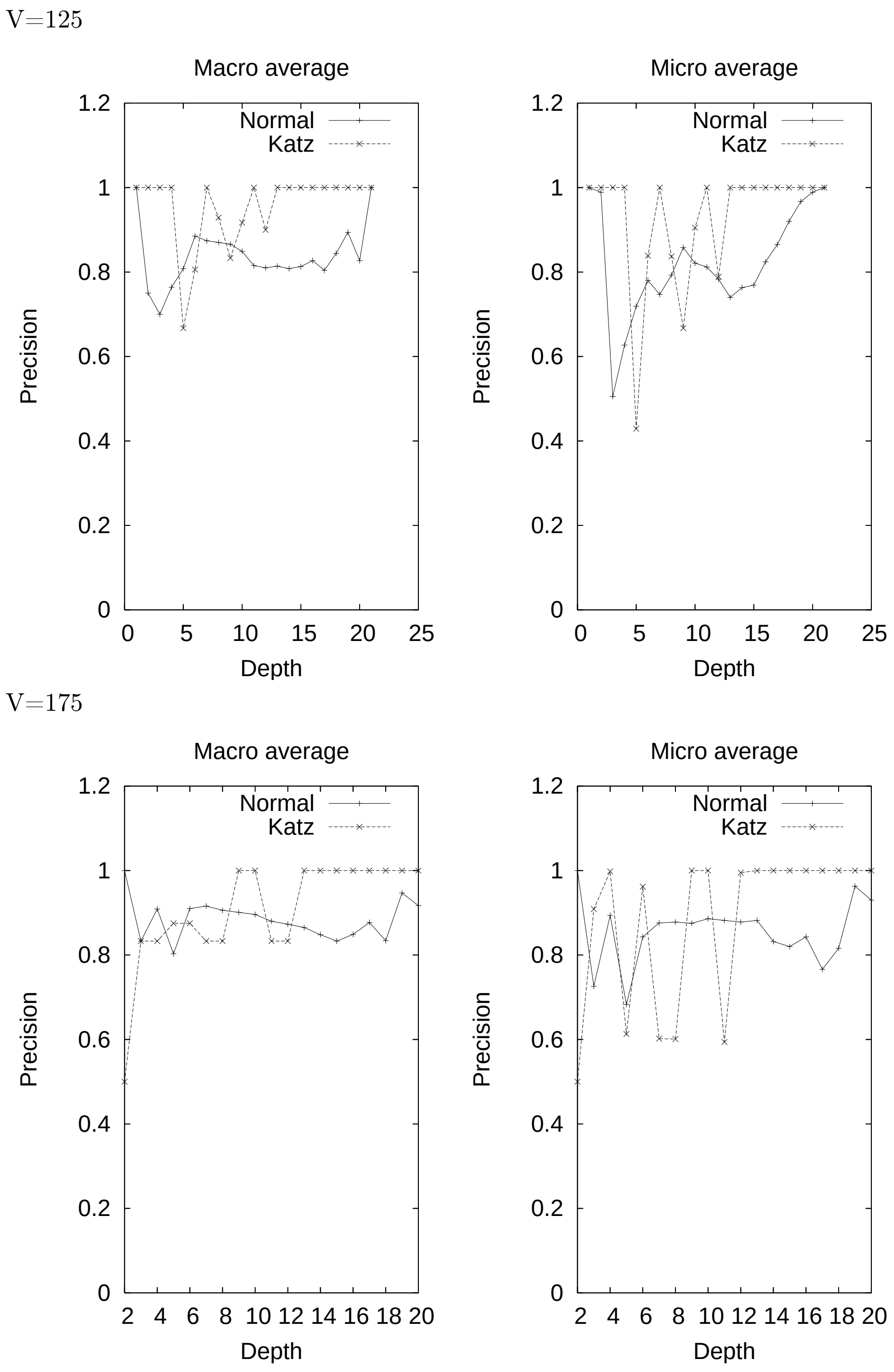


Figure 12. Tracing the sibling precision over the height of the tree. Vocabulary 125 and 175.

These plots show the general trend at different vocabulary sizes. As we can see there is considerable variation in the sibling precision over different depths. Amidst these variation we can observe that the sibling precision is higher and more consistent when we look at the nodes occuring at the lower layers of the tree. Also, we find that on these layers the Katz-Classit usually performs better than the Normal-Classit.

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

7 Conclusion

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 text document datasets. We have also proposed a variation of the same that has more desirable properties when used for incremental hierarchical text clustering.

The variation of Cobweb/Classit algorithm that we have demonstrated in this work uses Katz's distribution instead of Normal distribution used in the original formulation of the Classit algorithm. Katz's distribution is more appropriate for the word occurrence data as has been shown in prior work[14] and empirically observed in our work. We have evaluated both the algorithms over Reuters-RCV1 dataset, which allows us to carry out the experiments in a scenario very similar to the real life. We tested the algorithms by presenting them Newswire articles from Reuters-RCV1 dataset in time order and have shown that our algorithm performs consistently better than the Normal based Classit algorithm as measured by both the micro and macro average of the F score over a range vocabulary sizes. We have also evaluated both the algorithms using Ohsumed 88-91 dataset and have found that Katz-Classit performs better except for the narrow range of parameter values with small vocabulary sizes *and* large number of clusters, where results are likely to be unreliable. This shows that the performance of Katz-Classit is more robust across broad parameter settings.

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

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

Bibliography

[1] 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.

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

[3] 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.

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

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

[6] 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.

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

[8] 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.

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

[10] 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.

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

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

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

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

[15] 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.

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

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

[18] 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.

[19] 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.

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

Appendix A

MLE of Katz's distribution parameters

The Katz's distribution is defined as:

$$\begin{aligned} P(0) &= p_0 \\ P(k) &= (1-p_0)(1-p)p^{k-1}; \text{when } k > 0 \end{aligned} \tag{32}$$

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 $L(p, p_0)$ of the parameters given data is

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

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

Log of likelihood is

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

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

$$\begin{aligned}
\frac{\partial \mathrm{LL}}{\partial p_0} &= \frac{n_0}{\hat{p}_0} + \sum_{k=1}^{K} n_k \frac{-1}{1-\hat{p}_0} = 0\\
\Rightarrow \frac{n_0}{\hat{p}_0} &= \frac{1}{1-\hat{p}_0}\sum_{k=1}^{K} n_k = \frac{1}{1-\hat{p}_0}(N-n_0)\\
\Rightarrow \frac{1-\hat{p}_0}{\hat{p}_0} &= \frac{N-n_0}{n_0}\\
\Rightarrow \frac{1}{\hat{p}_0} - 1 &= \frac{N}{n_0} - 1\\
\Rightarrow \hat{p}_0 &= \frac{n_0}{N} = \frac{N-\mathrm{df}}{N} = 1 - \frac{\mathrm{df}}{N} \qquad (33)
\end{aligned}$$

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

$$\begin{aligned}
\frac{\partial \mathrm{LL}}{\partial p} &= \sum_{k=1}^{K} n_k \frac{-1}{1-\hat{p}} + \frac{n_k(k-1)}{\hat{p}} = 0\\
\Rightarrow 0 &= \frac{1}{\hat{p}}\sum_{k=1}^{K} n_k(k-1) - \frac{1}{1-\hat{p}}\sum_{k=1}^{K} n_k\\
\Rightarrow 0 &= \frac{1}{\hat{p}}\left(\sum_{k=1}^{K} k\, n_k - \sum_{k=1}^{K} n_k\right) - \frac{1}{1-\hat{p}}\sum_{k=1}^{K} n_k\\
\Rightarrow 0 &= \frac{1}{\hat{p}}(\mathrm{cf}-\mathrm{df}) - \frac{1}{1-\hat{p}}\mathrm{df}\\
\Rightarrow \frac{1-\hat{p}}{\hat{p}} &= \frac{\mathrm{df}}{\mathrm{cf}-\mathrm{df}}\\
\Rightarrow \frac{1}{\hat{p}} &= \frac{\mathrm{cf}}{\mathrm{cf}-\mathrm{df}}\\
\Rightarrow \hat{p} &= \frac{\mathrm{cf}-\mathrm{df}}{\mathrm{cf}} \qquad (34)
\end{aligned}$$

Expressions (33) and (34) are the MLE of the parameters of Katz's distribution described in Expression