Efficiently Clustering Documents with Committees

Patrick Pantel and Dekang Lin

Department of Computing Science
University of Alberta
Edmonton, Alberta T6H 2E1 Canada
{ppantel, lindek}@cs.ualberta.ca

Abstract. The general goal of clustering is to group data elements such that the intra-group similarities are high and the inter-group similarities are low. We present a clustering algorithm called CBC (Clustering By Committee) that is shown to produce higher quality clusters in document clustering tasks as compared to several well known clustering algorithms. It initially discovers a set of tight clusters (high intra-group similarity), called committees, that are well scattered in the similarity space (low inter-group similarity). The union of the committees is but a subset of all elements. The algorithm proceeds by assigning elements to their most similar committee. Evaluating cluster quality has always been a difficult task. We present a new evaluation methodology based on the editing distance between output clusters and manually constructed classes (the answer key). This evaluation measure is more intuitive and easier to interpret than previous evaluation measures.

1 Introduction

Document clustering was initially proposed for improving the precision and recall of information retrieval systems [14]. Because clustering is often too slow for large corpora and has indifferent performance [7], document clustering has been used more recently in document browsing [3], to improve the organization and viewing of retrieval results [5], to accelerate nearest-neighbor search [1] and to generate Yahoo-like hierarchies [10].

In this paper, we propose a clustering algorithm, CBC (Clustering By Committee), which produces higher quality clusters in document clustering tasks as compared to several well known clustering algorithms. Many clustering algorithms represent a cluster by the centroid of all of its members (e.g., K-means) [11] or by a representative element (e.g., K-medoids) [9]. When averaging over all elements in a cluster, the centroid of a cluster may be unduly influenced by elements that only marginally belong to the cluster or by elements that also belong to other clusters. Using a single representative from a cluster may be problematic too because each individual element has its own idiosyncrasies that may not be shared by other members of the cluster.

CBC constructs the centroid of a cluster by averaging the feature vectors of a subset of the cluster members. The subset is viewed as a committee that determines which other elements belong to the cluster. By carefully choosing committee members, the features of the centroid tend to be the more typical features of the target class.

We introduce a new evaluation methodology that is based on the editing distance between clustering results and manually constructed classes (the answer key).

2 Related Work

Generally, clustering algorithms can be categorized as hierarchical and partitional. In hierarchical agglomerative algorithms, clusters are constructed by iteratively merging the most similar clusters. These algorithms differ in how they compute cluster similarity. In single-link clustering, the similarity between two clusters is the similarity between their most similar members while complete-link clustering uses the similarity between their least similar members. Average-link clustering computes this similarity as the average similarity between all pairs of elements across clusters. The complexity of these algorithms is $O(n^2 \log n)$, where n is the number of elements to be clustered [6]. These algorithms are too inefficient for document clustering tasks that deal with large numbers of documents. In our experiments, one of the corpora we used is small enough (2745 documents) to allow us to compare CBC with these hierarchical algorithms.

Chameleon is a hierarchical algorithm that employs dynamic modeling to improve clustering quality [8]. When merging two clusters, one might consider the sum of the similarities between pairs of elements across the clusters (e.g. average-link clustering). A drawback of this approach is that the existence of a single pair of very similar elements might unduly cause the merger of two clusters. An alternative considers the number of pairs of elements whose similarity exceeds a certain threshold [4]. However, this may cause undesirable mergers when there are a large number of pairs whose similarities barely exceed the threshold. Chameleon clustering combines the two approaches.

Most often, document clustering employs K-means clustering since its complexity is linear in n, the number of elements to be clustered. K-means is a family of partitional clustering algorithms that iteratively assigns each element to one of K clusters according to the centroid closest to it and recomputes the centroid of each cluster as the average of the cluster's elements. Because the initial centroids are randomly selected, the resulting clusters vary in quality. Some sets of initial centroids lead to poor convergence rates or poor cluster quality.

Bisecting K-means [13], a variation of K-means, begins with a set containing one large cluster consisting of every element and iteratively picks the largest cluster in the set, splits it into two clusters and replaces it by the split clusters. Splitting a cluster consists of applying the basic K-means algorithm α times with K=2 and keeping the split that has the highest average element-centroid similarity.

Hybrid clustering algorithms combine hierarchical and partitional algorithms in an attempt to have the high quality of hierarchical algorithms with the efficiency of partitional algorithms. Buckshot [3] addresses the problem of randomly selecting initial centroids in K-means by combining it with average-link clustering. Cutting et al. claim its clusters are comparable in quality to hierarchical algorithms but with a lower complexity. Buckshot first applies average-link to a random sample of $\sqrt{n}$ elements to generate K clusters. It then uses the centroids of the clusters as the initial K centroids of K-means clustering. The complexity of Buckshot is $O(K \times T \times n + n \log n)$. The parameters K and T are usually considered to be small numbers. Since we are dealing with a large number of clusters, Buckshot and K-means become inefficient in practice.

Furthermore, Buckshot is not always suitable. If one wishes to cluster 100,000 documents into 1000 newsgroup topics, Buckshot could generate only 316 initial centroids.

3 Representation

CBC represents elements as feature vectors. The features of a document are the terms (usually stemmed words) that occur within it and the value of a feature is a statistic of the term. For example, the statistic can simply be the term's frequency, *tf*, within the document. In order to discount terms with low discriminating power, *tf* is usually combined with the term's inverse document frequency, *idf*, which is the inverse of the percentage of documents in which the term occurs. This measure is referred to as *tf-idf* [12]:

$$tf\text{-}idf = tf \times \log idf$$

We use the mutual information [2] between an element and its features.

In our algorithm, for each element e, we construct a **frequency count vector** $C(e) = (c_{e1}, c_{e2}, \ldots, c_{em})$, where m is the total number of features and c_{ef} is the frequency count of feature f occurring in element e. In document clustering, e is a document and c_{ef} is the term frequency of f in e. We construct a **mutual information vector** $MI(e) = (mi_{e1}, mi_{e2}, \ldots, mi_{em})$, where mi_{ef} is the mutual information between element e and feature f, which is defined as:

$$mi_{ef} = log \frac{\frac{c_{ef}}{N}}{\frac{\sum_i c_{if}}{N} \times \frac{\sum_j c_{ej}}{N}}$$

where $N = \sum_i \sum_j c_{ij}$ is the total frequency count of all features of all elements.

We compute the similarity between two elements e_i and e_j using the *cosine coefficient* [12] of their mutual information vectors:

$$sim(e_i, e_j) = \frac{\sum_f mi_{e_i f} \times mi_{e_j f}}{\sqrt{\sum_f mi_{e_i f}^{\ 2} \times \sum_f mi_{e_j f}^{\ 2}}}$$

4 Algorithm

CBC consists of three phases. In Phase I, we compute each element's top-k similar elements. In our experiments, we used $k = 20$. In Phase II, we construct a collection of tight clusters, where the elements of each cluster form a **committee**. The algorithm tries to form as many committees as possible on the condition that each newly formed committee is not very similar to any existing committee. If the condition is violated,

Input: A list of elements E to be clustered, a similarity database S from Phase I, thresholds θ_1 and θ_2.

Step 1: For each element $e \in E$

Cluster the top similar elements of e from S using average-link clustering.

For each cluster discovered c compute the following score: $|c| \times$ avgsim(c), where $|c|$ is the number of elements in c and avgsim(c) is the average pairwise similarity between elements in c.

Store the highest-scoring cluster in a list L.

Step 2: Sort the clusters in L in descending order of their scores.

Step 3: Let C be a list of committees, initially empty.

For each cluster $c \in L$ in sorted order

Compute the centroid of c by averaging the frequency vectors of its elements and computing the mutual information vector of the centroid in the same way as we did for individual elements.

If c's similarity to the centroid of each committee previously added to C is below a threshold θ_1, add c to C.

Step 4: If C is empty, we are done and return C.

Step 5: For each element $e \in E$

If e's similarity to every committee in C is below threshold θ_2, add e to a list of residues R.

Step 6: If R is empty, we are done and return C.

Otherwise, return the union of C and the output of a recursive call to Phase II using the same input except replacing E with R.

Output: a list of committees.

Figure 1. Phase II of CBC.

the committee is simply discarded. In the final phase of the algorithm, each element is assigned to its most similar cluster.

4.1 Phase I: Find top-similar elements

Computing the complete similarity matrix between pairs of elements is obviously quadratic. However, one can dramatically reduce the running time by taking advantage of the fact that the feature vector is sparse. By indexing the features, one can retrieve the set of elements that have a given feature. To compute the top similar elements of an element e, we first sort the mutual information vector $MI(e)$ and then only consider a subset of the features with highest mutual information. Finally, we compute the pairwise similarity between e and the elements that share a feature from this subset. Since high mutual information features tend not to occur in many elements, we only need to compute a fraction of the possible pairwise combinations. With

18,828 elements, Phase I completes in 38 minutes. Using this heuristic, similar words that share only low mutual information features will be missed by our algorithm. However, in our experiments, this had no visible impact on cluster quality.

4.2 Phase II: Find committees

The second phase of the clustering algorithm recursively finds tight clusters scattered in the similarity space. In each recursive step, the algorithm finds a set of tight clusters, called committees, and identifies residue elements that are not covered by any committee. We say a committee **covers** an element if the element's similarity to the centroid of the committee exceeds some high similarity threshold. The algorithm then recursively attempts to find more committees among the residue elements. The output of the algorithm is the union of all committees found in each recursive step. The details of Phase II are presented in Figure 1.

In Step 1, the score reflects a preference for bigger and tighter clusters. Step 2 gives preference to higher quality clusters in Step 3, where a cluster is only kept if its similarity to all previously kept clusters is below a fixed threshold. In our experiments, we set $\theta_1 = 0.35$. Step 4 terminates the recursion if no committee is found in the previous step. The residue elements are identified in Step 5 and if no residues are found, the algorithm terminates; otherwise, we recursively apply the algorithm to the residue elements. Each committee that is discovered in this phase defines one of the final output clusters of the algorithm.

4.3 Phase III: Assign elements to clusters

In Phase III, every element is assigned to the cluster containing the committee to which it is most similar. This phase resembles *K*-means in that every element is assigned to its closest centroid. Unlike *K*-means, the number of clusters is not fixed and the centroids do not change (i.e. when an element is added to a cluster, it is not added to the committee of the cluster).

5 Evaluation Methodology

Many cluster evaluation schemes have been proposed. They generally fall under two categories:

- comparing cluster outputs with manually generated answer keys (hereon referred to as **classes**); and
- embedding the clusters in an application (e.g. information retrieval) and using its evaluation measure.

One approach considers the average entropy of the clusters, which measures the purity of the clusters [13]. However, maximum purity is trivially achieved when each element forms its own cluster.

Given a partitioned set of n elements, there are $n \times (n - 1) / 2$ pairs of elements that are either in the same partition or not. The partition implies $n \times (n - 1) / 2$ decisions. Another way to evaluate clusters is to compute the percentage of the decisions that are in agreement between the clusters and the classes [15]. This measure sometimes gives unintuitive results. Suppose the answer key consists of 20 equally sized classes with 1000 elements in each. Treating each element as its own cluster gets a misleadingly high score of 95%.

The evaluation of document clustering algorithms in information retrieval often uses the embedded approach [5]. Suppose we cluster the documents returned by a search engine. Assuming the user is able to pick the most relevant cluster, the performance of the clustering algorithm can be measured by the average precision of the chosen cluster. Under this scheme, only the best cluster matters.

The entropy and pairwise decision schemes each measure a specific property of clusters. However, these properties are not directly related to application-level goals of clustering. The information retrieval scheme is goal-oriented, however it measures only the quality of the best cluster. We propose an evaluation methodology that strikes a balance between generality and goal-orientation.

Like the entropy and pairwise decision schemes, we assume that there is an answer key that defines how the elements are supposed to be clustered. Let C be a set of clusters and A be the answer key. We define the editing distance, $dist(C, A)$, as the number of operations required to transform C into A. We allow three editing operations:

- merge two clusters;
- move an element from one cluster to another; and
- copy an element from one cluster to another.

Let B be the baseline clustering where each element is its own cluster. We define the quality of cluster C as follows:

$$1 - \frac{dist(C, A)}{dist(B, A)}$$

This measure can be interpreted as the percentage of savings from using the clustering result to construct the answer key versus constructing it from scratch (i.e. the baseline).

We make the assumption that each element belongs to exactly one cluster. The transformation procedure is as follows:

1. Suppose there are m classes in the answer key. We start with a list of m empty sets, each of which is labeled with a class in the answer key.
2. For each cluster, merge it with the set whose class has the largest number of elements in the cluster (a tie is broken arbitrarily).
3. If an element is in a set whose class is not the same as one of the element's classes, move the element to a set where it belongs.
4. If an element belongs to more than one target class, copy the element to all sets corresponding to the target classes (except the one to which it already belongs).

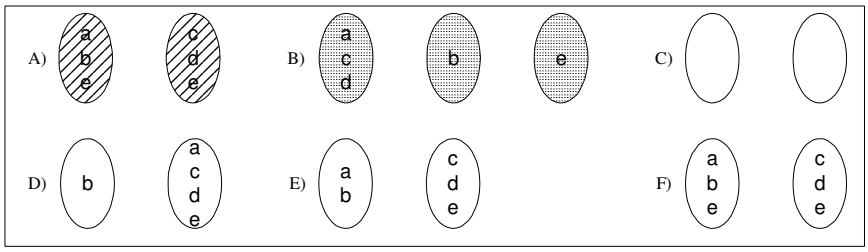

Figure 2. An example of applying the transformation rules to three clusters. A) The classes in the answer key; B) the clusters to be transformed; C) the sets used to reconstruct the classes (Rule 1); D) the sets after three merge operations (Step 2); E) the sets after one move operation (Step 3); F) the sets after one copy operation (Step 4).

Table 1. The number of classes in each test data set and the number of elements in their largest and smallest classes.

DATA SET	*TOTAL DOCS*	*TOTAL CLASSES*	*LARGEST CLASS*	*SMALLEST CLASS*
Reuters	2745	92	1045	1
20-news	18828	20	999	628

dist(*C*, *A*) is the number of operations performed using the above transformation rules on *C*.

Figure 2 shows an example. In *D*) the cluster containing *e* could have been merged with either set (we arbitrarily chose the second). The total number of operations is 5.

6 Experimental Results

6.1 Test Data

We conducted document-clustering experiments with two data sets: Reuters-21578 V1.2[1] and 20news-18828[2] (see Table 1). For the Reuters corpus, we selected documents that:

1. are assigned one or more topics;
2. have the attribute LEWISSPLIT="TEST"; and
3. have <BODY> and </BODY> tags.

There are 2745 such documents. The 20news-18828 data set contains 18828 newsgroup articles partitioned (nearly) evenly across 20 different newsgroups.

[1] http://www.research.att.com/~lewis/reuters21578.html
[2] http://www.ai.mit.edu/people/jrennie/20_newsgroups/

Table 2. Cluster quality (%) of several algorithms on the Reuters and 20-news data sets.

	REUTERS	*20-NEWS*
CBC	65.00	74.18
K-means	62.38	70.04
Buckshot	62.03	65.96
Bisecting *K*-means	60.80	58.52
Chameleon	58.67	n/a
Average-link	63.00	70.43
Complete-link	46.22	64.23
Single-link	31.53	5.30

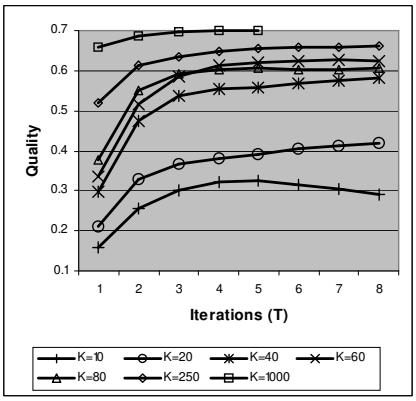

Figure 3. *K*-means cluster quality on the 20-news data set for different values of *K* plotted of over eight iterations.

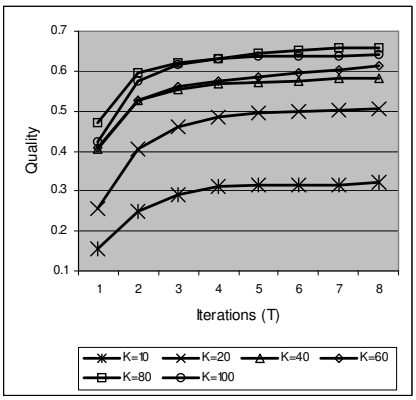

Figure 4. Buckshot cluster quality on the 20-news data set for different values of *K* plotted of over eight iterations.

6.2 Cluster Evaluation

We clustered the data sets using CBC and the clustering algorithms of Section 2 and applied the evaluation methodology from the previous section. Table 2 shows the results. The columns are our editing distance based evaluation measure. CBC outperforms *K*-means with *K*=1000 by 4.14%. On the 20-news data set, our implementation of Chameleon was unable to complete in reasonable time. For the 20-news corpus, CBC spends the vast majority of the time finding the top similar documents (38 minutes) and computing the similarity between documents and committee centroids (119 minutes). The rest of the computation, which includes clustering the top-20 similar documents for every one of the 18828 documents and sorting the clusters, took less than 5 minutes. We used a Pentium III 750MHz processor and 1GB of memory.

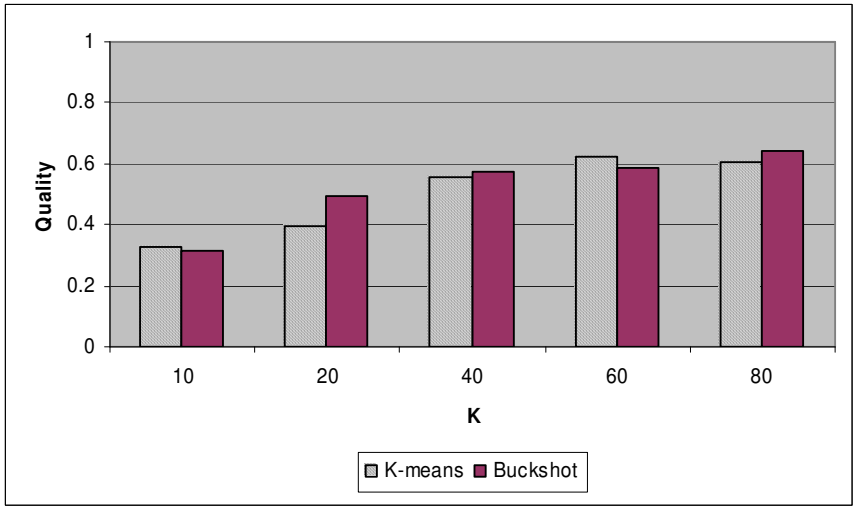

Figure 5. Comparison of cluster quality between *K*-means and Buckshot for different *K* on the 20-news data set.

6.3 *K*-means and Buckshot

Figure 3 and Figure 4 show the cluster quality of different *K*'s on the 20-news data set plotted over eight iterations of the *K*-means and Buckshot algorithms respectively. The cluster quality for *K*-means clearly increases as *K* reaches 1000 although the increase in quality slows down between *K*=60 and *K*=1000.

Buckshot has similar performance to *K*-means on the Reuters corpus; however it performs much worse on the 20-news corpus. This is because *K*-means performs well on this data set when *K* is large (e.g. *K*=1000) whereas Buckshot cannot have *K* higher than $\sqrt{18828} = 137$. On the Reuters corpus, the best clusters for *K*-means were obtained with *K* = 50, and Buckshot can have *K* as large as $\sqrt{2745} = 52$. However, as *K* approaches 52, Buckshot degenerates to the *K*-means algorithm, which explains why Buckshot has similar performance to *K*-means. Figure 5 compares the cluster quality between *K*-means and Buckshot for different values of *K* on the 20-news data set.

7 Conclusion

Document clustering is an important tool in information retrieval. We presented a clustering algorithm, CBC, which can handle a large number of documents, a large number of output clusters, and a large sparse feature space. It discovers clusters using well-scattered tight clusters called committees. In our experiments on document clustering, we showed that CBC outperforms several well-known hierarchical, partitional,

and hybrid clustering algorithms in cluster quality. For example, in one experiment, CBC outperforms *K*-means by 4.14%.

Evaluating cluster quality has always been a difficult task. We presented a new evaluation methodology that is based on the editing distance between output clusters and manually constructed classes (the answer key). This evaluation measure is more intuitive and easier to interpret than previous evaluation measures.

Acknowledgements

The authors wish to thank the reviewers for their helpful comments. This research was partly supported by Natural Sciences and Engineering Research Council of Canada grant OGP121338 and scholarship PGSB207797.

References

1. Buckley, C. and Lewit, A. F. 1985. Optimization of inverted vector searches. In *Proceedings of SIGIR-85*. pp. 97–110.
2. Church, K. and Hanks, P. 1989. Word association norms, mutual information, and lexicography. In *Proceedings of ACL-89*. pp. 76–83. Vancouver, Canada.
3. Cutting, D. R.; Karger, D.; Pedersen, J.; and Tukey, J. W. 1992. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Proceedings of SIGIR-92*. pp. 318–329. Copenhagen, Denmark.
4. Guha, S.; Rastogi, R.; and Kyuseok, S. 1999. ROCK: A robust clustering algorithm for categorical attributes. In *Proceedings of ICDE'99*. pp. 512–521. Sydney, Australia.
5. Hearst, M. A. and Pedersen, J. O. 1996. Reexamining the cluster hypothesis: Scatter/Gather on retrieval results. In *Proceedings of SIGIR-96*. pp. 76–84. Zurich, Switzerland.
6. Jain, A.K.; Murty, M.N.; and Flynn, P.J. 1999. Data Clustering: A Review. *ACM Computing Surveys* 31(3):264–323.
7. Jardine, N. and van Rijsbergen, C. J. 1971. The use of hierarchical clustering in information retrieval. *Information Storage and Retreival*, 7:217–240.
8. Karypis, G.; Han, E.-H.; and Kumar, V. 1999. Chameleon: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer: Special Issue on Data Analysis and Mining* 32(8):68–75.
9. Kaufmann, L. and Rousseeuw, P. J. 1987. Clustering by means of medoids. In Dodge, Y. (Ed.) *Statistical Data Analysis based on the L1 Norm*. pp. 405–416. Elsevier/North Holland, Amsterdam.
10. Koller, D. and Sahami, M. 1997. Hierarchically classifying documents using very few words. In *Proceedings of ICML-97*. pp. 170–176. Nashville, TN.
11. McQueen, J. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematics, Statistics and Probability*, 1:281-298.
12. Salton, G. and McGill, M. J. 1983. *Introduction to Modern Information Retrieval*. McGraw Hill.
13. Steinbach, M.; Karypis, G.; and Kumar, V. 2000. *A comparison of document clustering techniques*. Technical Report #00-034. Department of Computer Science and Engineering, University of Minnesota.
14. van Rijsbergen, C. J. 1979. *Information Retrieval*, second edition. London: Buttersworth. Available at: http://www.dcs.gla.ac.uk/Keith/Preface.html
15. Wagstaff, K. and Cardie, C. 2000. Clustering with instance-level constraints. In *Proceedings of ICML-2000*. pp. 1103–1110. Palo Alto, CA.