Fast and Effective Text Mining Using Linear-time Document Clustering

Bjornar Larsen and Chinatsu Aone
SRA International, Inc.
4300 Fair Lakes Court
Fairfax, VA 22033
{bjornar_larsen, aonec}@sra.com

ABSTRACT

Clustering is a powerful technique for large-scale topic discovery from text. It involves two phases: first, feature extraction maps each document or record to a point in high-dimensional space, then clustering algorithms automatically group the points into a hierarchy of clusters. We describe an unsupervised, near-linear time text clustering system that offers a number of algorithm choices for each phase. We introduce a methodology for measuring the quality of a cluster hierarchy in terms of F-Measure, and present the results of experiments comparing different algorithms. The evaluation considers some feature selection parameters (*tfidf* and feature vector length) but focuses on the clustering algorithms, namely techniques from Scatter/Gather (buckshot, fractionation, and split/join) and *k*-means. Our experiments suggest that continuous center adjustment contributes more to cluster quality than seed selection does. It follows that using a simpler seed selection algorithm gives a better time/quality tradeoff. We describe a refinement to center adjustment, "vector average damping," that further improves cluster quality. We also compare the near-linear time algorithms to a group average greedy agglomerative clustering algorithm to demonstrate the time/quality tradeoff quantitatively.

Keywords

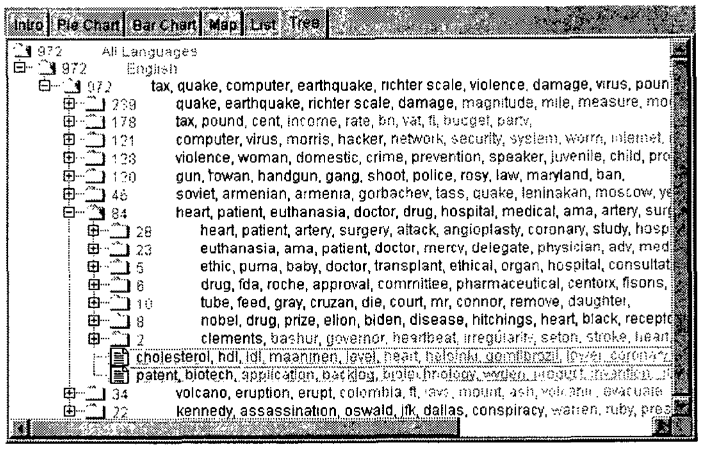
Clustering, text mining, multi-document summarization

1. INTRODUCTION

The information age is surrounding us with increasingly overwhelming quantities of electronic data. Users need software tools that can help them rapidly explore the most frequent form of data, collections of text. Hand-built directories of web content such as Yahoo! offer one solution to the problem, but unfortunately creating and maintaining such directories requires enormous amounts of human effort. Routing/categorization systems can help automate the assignment of documents to a topic hierarchy, but they require training and prior knowledge of the topics in a corpus. For many situations, a more practical solution is to discover and approximate these topic hierarchies using unsupervised clustering methods.

Document clustering helps tackle the information overload problem in several ways. One is exploration; the top level of a cluster hierarchy summarizes at a glance the contents of a document collection, enabling users to selectively drill deeper to explore specific topics of interest without reading every document (e.g., [7], [9], [13]). Used in retrieval (e.g., [6], [10], [14]), clustering organizes search results by topic similarity and potentially helps users find relevant documents more quickly. Some clustering algorithms excel at quickly and accurately grouping duplicate or near-duplicate documents. Though each of these uses is important, the focus of our paper is the first use, i.e., multi-document summarization through the discovery of topic hierarchies.

In this paper, we first describe our fast, scalable document clustering system. This text mining tool is designed to discover topic hierarchies in gigabytes of documents per day[1], and to present the results in an intuitive GUI (cf. Figure 1). We describe the algorithms we use, for both feature extraction and clustering of extracted features. Then, we evaluate the impact of different algorithms on the quality of the generated cluster hierarchies and



```
972  All Languages
972  English
  972  tax, quake, computer, earthquake, richter scale, violence, damage, virus, poun
    239  quake, earthquake, richter scale, damage, magnitude, mile, measure, mo
    178  tax, pound, cent, income, rate, bn, vat, fl, budget, party,
    121  computer, virus, morris, hacker, network, security, system, worm, internet,
    123  violence, woman, domestic, crime, prevention, speaker, juvenile, child, pro
    120  gun, towan, handgun, gang, shoot, police, rosy, law, maryland, ban,
    46   soviet, armenian, armenia, gorbachev, tass, quake, leninakan, moscow, ye
    84   heart, patient, euthanasia, doctor, drug, hospital, medical, ama, artery, sur
      28   heart, patient, artery, surgery, attack, angioplasty, coronary, study, hosp
      23   euthanasia, ama, patient, doctor, mercy, delegate, physician, adv, med
      5    ethic, puma, baby, doctor, transplant, ethical, organ, hospital, consultat
      6    drug, fda, roche, approval, committee, pharmaceutical, centorx, fisons,
      10   tube, feed, gray, cruzan, die, court, mr, connor, remove, daughter,
      8    nobel, drug, prize, elion, biden, disease, hitchings, heart, black, recept
      2    clements, bashur, governor, heartbeat, irregularity, seton, stroke, hear
      cholesterol, hdl, ldl, maaninen, level, heart, helsinki, gemfibrozil, lower, corona
      patent, biotech, application, backlog, biotechnology, wyden, product, invention,
    34   volcano, eruption, erupt, colombia, ft, lava, mount, ash, volcano, evacuate
    22   kennedy, assassination, oswald, jfk, dallas, conspiracy, warren, ruby, pres
```

Figure 1 - User Interface

[1] Current throughput using the default near-linear algorithms, including all pre-processing (e.g., for building a term frequency baseline) and feature-extraction, is 1.5 hours for 1 gigabyte of text (~200,000 documents) on a Sun ULTRA 1 with 256MB RAM.

on the processing time required to build those hierarchies.

Figure 1 shows the top level clusters for a small corpus (972 documents), a large portion of which are news stories. The first cluster (239 documents) is about "quake, earthquake, richter scale" etc., so we can conclude that it contains documents about earthquakes. Each cluster is labeled with related terms (from the cluster centroid) that convey a coherent topic. The seventh cluster (84 documents) appears to be on medical issues ("heart, patient, euthanasia, doctor, drug, hospital, medical" etc.) We have expanded this cluster to show the nine sub-clusters (which represent medical subtopics) – two of these are documents (i.e., from a cluster of size one), shaded in gray.

2. SYSTEM DESCRIPTION

The primary steps to generating a hierarchy are the extraction of features from documents, followed by clustering or grouping based on those features. Over forty configurable parameters control the algorithms in this process – we will describe the most significant variations.

2.1 Feature Extraction

Feature extraction maps each document into a concise representation of its topic. These extracted features are used to compare documents and to label their content for users. We use the vector space model [11] to represent documents as points in a high dimensional topic space, where each dimension corresponds to a unique word or concept from the corpus. The mapping process extracts a list of unique terms from each document, assigns each a weight, and represents the document using the n highest-weighted terms. We refer to the parameter n as the feature vector length, and our default is 25.

Several parameters control term selection and weighting. By default, we remove stop words. The weight assigned to each remaining term is a function of either the *tf* (term frequency) or *tfidf* (term frequency – inverse document frequency). To use *tfidf*, the system makes an initial pass over the text collection to create a "baseline" – the baseline stores the total number of documents that each unique term occurs in (i.e., the document frequency). Feature extraction consults this baseline to set term weights as a function of both the term's frequency within the document and the number of documents the term occurs in. An important word is one that appears frequently within the current document, but infrequently across the other documents in the collection. For example, the word 'excavation' is uncommon in most corpora, so if it appeared frequently in a particular document, it would be given a high topic weight. Conversely, words like 'say' that might appear in almost every document are assigned a low weight (even if they occur frequently in a given document.)

Because feature extraction produces generic points in space as input to the clustering algorithms, the system is not limited to text; by implementing new feature extraction modules to map data to high-dimensional points, we could cluster any type of data.

2.2 Clustering

The goal of clustering is to group the points in a feature space optimally based on proximity, in order to form a hierarchy of clusters. We unified near-linear time complexity techniques from k-means ([8], [4]) and Scatter/Gather [1]. The techniques are all partitional, meaning that they simply separate a flat collection of items into a single set of "bins." A hierarchy is built by recursively applying a partitional algorithm. The partitional algorithms each run in O(N) with respect to the number of documents, N, so the overall hierarchy is generated in O(N log N) time (assuming a balanced hierarchy.)

Because documents and clusters are represented as points in space, we can compare them using vector cosine. Clusters include a "center" or "centroid" vector that is the weighted average of the documents or clusters they contain. To prevent longer documents from dominating centroid calculations, we normalize all document vectors to unit length. To compare a document to a cluster, we simply calculate the cosine between the document vector and the cluster's centroid vector.

The partitional algorithms have three stages: seed selection, center adjustment, and cluster refinement. Seed selection is the process of choosing k candidate points in the feature space to serve as centers for the partitions[2]. During center adjustment, documents are repeatedly assigned to the nearest center, and the center is recalculated based on the average location of all documents assigned to it, thereby moving it through the feature space. This process may be repeated multiple times. Afterwards, all documents are removed from the centers, and reassigned to the new closest center. Thus, it is important that the centers be distributed effectively enough that they each attract sufficient nearby, topically related documents. Cluster refinement is an optional final step for improving the new partitions.

2.2.1 Seed Selection

Seed selection picks centers to which the system can assign each point in the input set to form a partition. We implemented three seed selection algorithms: random, buckshot, and fractionation. Random is the simplest; it picks k points randomly from the input set as the initial centers.

The second method is buckshot, described by [1]. Buckshot picks $\sqrt{kn}$ points randomly from the input set of n items, and clusters them using a high quality $O(N^2)$ clustering algorithm. The k centroids resulting from this clustering become the initial centers. For the $O(N^2)$ algorithm we use the group average variation of greedy agglomerative clustering, as did [1]. We will also refer to this as the "cluster subroutine."

The third method, fractionation, is also described by [1]. It uses the same cluster subroutine to build a bottom-up hierarchy from the initial input set, clustering fixed-size groups of points at each step to maintain a linear time complexity. The top-level clusters of this hierarchy become the initial seeds.

2.2.2 Center Adjustment

Once k seeds are selected as centers, the system can iteratively assign each point in the input set to the closest center and adjust that center accordingly. If a point's similarity to every center is below the assignment similarity threshold, t, it is not assigned to any center. By default, we use a small non-zero fixed value for t, though we are investigating techniques for setting t dynamically. Continuous k-means [4] consists of following random seed selection with some number of iterations of center adjustment. In

[2] k defaults to 9 in our system.

our implementation, an iteration is an entire pass over a random ordering of the input set. A parameter specifies whether to adjust the centers after each point is assigned (continuous), or only at the end of an iteration when all points have been assigned (non-continuous). Our default is continuous center adjustment.

[4] suggests that for a large data set the centers may stabilize when only a fraction of the points have been considered. We tried having the system stop adjusting centers when they stabilize rather than after a fixed number of iterations, but found that the centers rarely stop moving entirely, and the extra time cost of checking for minimal movement outweighs the advantages gained from knowing exactly when to stop.

[1] describes a step called 'iterated assign-to-nearest' that appears equivalent to *k*-means center adjustment, but it is not clear whether their technique was continuous or non-continuous.

We introduce a modification to *k*-means, called vector average damping, that increases its effectiveness. As each point is assigned to an existing center, that center is normally recalculated as the average of all points that have been assigned to it. We provide a damping parameter for this averaging function that lowers the weight of the existing center relative to the point being averaged in (which has weight '1'). This allows points toward the end of our random ordering of the input set to continue moving the center more than they otherwise could.

When we finish adjusting the centers, we again iterate over the input set and assign each point to the closest center. A new "unmatched" cluster[3] is created to hold the points that do not match any center (i.e., their similarity to each center falls below the threshold *t*). Though their contents may include many differing topics, we recursively partition the unmatched clusters just like the main clusters.

2.2.3 Cluster Refinement

We experimented with cluster refinement algorithms to determine if they can improve the quality of the partitioning. We implemented the basic split/join algorithm described in [1], and then experimented with various modifications to it. This algorithm first breaks each cluster in two (we exclude the "unmatched" cluster) using the cluster subroutine, then rejoins the closest pairs. This split and join process can be repeated any number of times, theoretically improving the quality of the partitioning.

Different criteria may be used to decide which clusters to join. We initially tried combining clusters whose cluster centroids share more than *p* common keywords, where *p* is a fixed parameter [1]. We then tried applying the clustering subroutine to the join problem, and experimented with three different approaches for deciding when to stop joining clusters. First, we tried combining all pairs for which the cosine between them exceeded a fixed parameter. Second, we tried measuring the parent cluster's variance [4], and combining pairs whose cosine similarity exceeded a constant multiple of this variance. Third, we simply applied the greedy cluster subroutine to the 2*k* split clusters, combining each closest pair of clusters until exactly *k* clusters remained.

[3] [6] refers to these as "junk" clusters.

3. EVALUATION

3.1 Evaluation Methodology

Because we use clustering as an exploration tool, our evaluation approach focuses on the overall quality of generated cluster hierarchies (as opposed to, for example, measurements of retrieval effectiveness.) We compare how closely each hierarchy generated by the system matches a set of categories previously assigned to the documents by human judges. This required (1) the preparation of document collections in which each document had been assigned a topic label and (2) the specification of a scoring algorithm. We recognize that our scoring scheme is not perfect (for example, it does not account for the fact that documents often have multiple topics), but in practice it captures the strengths and shortcomings of the clustering algorithms.

To reduce the risk that our conclusions might be valid only on a particular corpus, we used two distinct test corpora: a subset of the TREC-5 collection [5], and a subset of the Reuters-21578 collection[4]. For each, we formed two test sets of different sizes, ranging from 5 to 100 megabytes[5], as shown in Table 1. Document topic labels were used only for scoring, i.e., not during feature extraction or clustering.

Test Corpus	# of documents	Size (MB)	# of topics
1 – TREC	5524	103.0	50
2 – TREC	972	8.1	9
3 – Reuters	8654	11.0	65
4 – Reuters	2794	4.7	63

Table 1 - Test corpora

Our evaluation algorithm treats the cluster hierarchy as if it were output from an automatic multi-level routing system. For each hand-labeled topic T in the document set, we assume that a cluster C corresponding to that topic will form automatically somewhere in the hierarchy. This is because topic clusters sometimes form at many levels. For example, the system often built a cluster of documents from the TREC topic "domestic violence" and a cluster from the topic "gun control" under a single unifying parent cluster. To find C, we traverse the hierarchy, calculating precision, recall, and F-Measure for each cluster with respect to the topic in question. For any topic T and cluster X:

N_1 = # of documents judged to be of topic T in cluster X

N_2 = # of documents in cluster X

N_3 = # of documents judged to be of topic T in entire hierarchy

Precision(X, T) = N_1 / N_2

Recall(X, T) = N_1 / N_3

$$F = \frac{2PR}{P+R}$$ where F=F-Measure, P=precision, and R=recall.

[4] The Reuters-21578, Distribution 1.0 test collection is available from David D. Lewis' professional home page, currently http://www.research.att.com/~lewis.

[5] For both corpora the second sub-collection is a subset of the first.

To calculate precision and recall for a cluster, we "flatten" the cluster to also include the documents from all sub-clusters. We consider the cluster with the highest F-Measure to be C, and that F-Measure becomes the system's score for topic T. The *overall* F-Measure for the hierarchy is the weighted average of the F-Measures for each topic T.

$$Overall\ F-Measure = \frac{\sum_{T\in M}(|T| * F(T))}{\sum_{T\in M}|T|},\ where\ M\ is$$

the set of topics, $|T|$ *is the number of documents judged of topic* T, *and* $F(T)$ *is the* $F-Measure$ *for topic* T

The near-linear clustering algorithms we use are nondeterministic due to their use of random seed selection; consequently clustering the same corpus multiple times with the same parameters will produce a different hierarchy each time. This necessitates multiple trials – we repeat each experiment ten times, then calculate the average, high, low, and standard deviation for each measured value (e.g., F-Measure). Though we simplify the presented results by showing only the average value, it should be noted that standard deviation for F-Measure is typically around 0.01, and occasionally as high as 0.02. Thus when we report that certain parameters score higher than others, we are looking at the average over multiple trials and considering the standard deviation.

We discuss the results in terms of F-Measure and processing time. The results of each experiment (except *tfidf*, as we describe below) were consistent for each test corpus, so in most cases we present results for only Corpus 1[6].

3.2 Overall System Performance

System performance for the default parameters is displayed in Table 2. All results are from a 166 MHz single CPU Sparc Ultra 1 with 256 MB of RAM.

Corpus	Size (MB)	Number of Docs	Baseline Creation Time (sec)	Feature Extraction Time (sec)	Clustering Time (sec)	Overall Time
1	103.0	5524	156	178	9.7	5 min 44 sec
2	8.1	972	11	14	1.1	26 sec
3	11.0	8654	14	22	24.1	60 sec
4	4.7	2794	6	8	3.4	17 sec

Table 2: Clustering System Performance Measurements

3.3 Feature Extraction Results

3.3.1 tfidf *Results*

Except for on corpora with a very small number of documents, weighting terms by *tfidf* works better than weighting by *tf* (term frequency). Figure 2 shows that for three of the four test corpora,

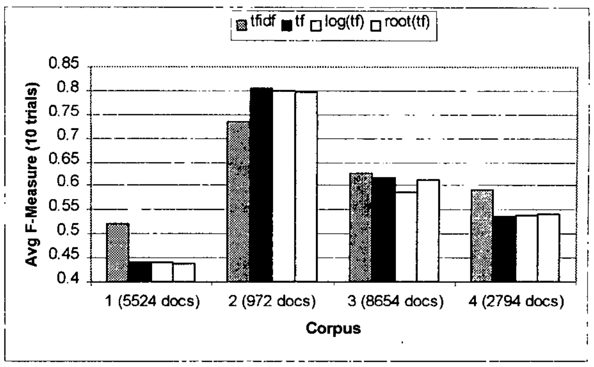


Figure 2 - Differences in clustering quality for *tfidf* and *tf* used in feature extraction

tfidf outperformed *tf*. In Corpus 1, average F-Measure increases from 0.44 to 0.52 with *tfidf*, a difference that significantly exceeds the standard deviation of around 0.01. Damping the frequency using a square root or logarithm function [1] makes no difference. Only in Corpus 2, the one with the fewest documents, did term frequency perform better. However, we noticed that the clarity of the cluster and document labels (viewed in the GUI) was inferior for *tf* as compared to *tfidf*. Corpus 3's results are surprising because the score difference between *tfidf* and *tf* is small, yet this corpus has the most documents. We think this is because Corpus 3 contains many tiny earnings report documents with no sentences, only sets of numbers and acronyms – in fact, we created Corpus 4 by removing these documents. These four test sets may be too small to provide *conclusive* evidence, but our experience suggests that larger document sets consistently benefit from *tfidf*.

3.3.2 *Feature Vector Length Results*

Our experiments varying feature vector length show that longer vectors increase the quality of a topic hierarchy but require additional compute time (cf. Figure 3). For example, using length 250 vectors instead of length 25 vectors increases the F-Measure from 0.53 to 0.60, but increases clustering time from 10 seconds to 110 seconds. Longer vectors also require additional memory. For our purposes, length 25 vectors offer an appropriate default quality/time/memory tradeoff. [12] also measured the effect of truncating feature vectors, concluding that except for radical truncation (length 20), truncation does not affect quality. We attribute the difference in results to different evaluation techniques; their evaluation measured retrieval effectiveness on retrieved subsets of the corpus (using precision), while ours considers the quality of a complete hierarchy (using F-Measure).

3.4 Clustering Results

3.4.1 *Seed Selection and Center Adjustment Results*

Figure 4 shows the results of a series of tests comparing seed selection techniques and center adjustment variations. Each x-axis label includes a letter ("c" for continuous center adjustment or "n" for non-continuous adjustment) and a number that corresponds to the number of iterations of center adjustment that were performed (0, 1, 2, or 4). Without any center adjustment, random seed selection performs very poorly (0.352 F-Measure)

6 We do not present a complete set of results due to space considerations.

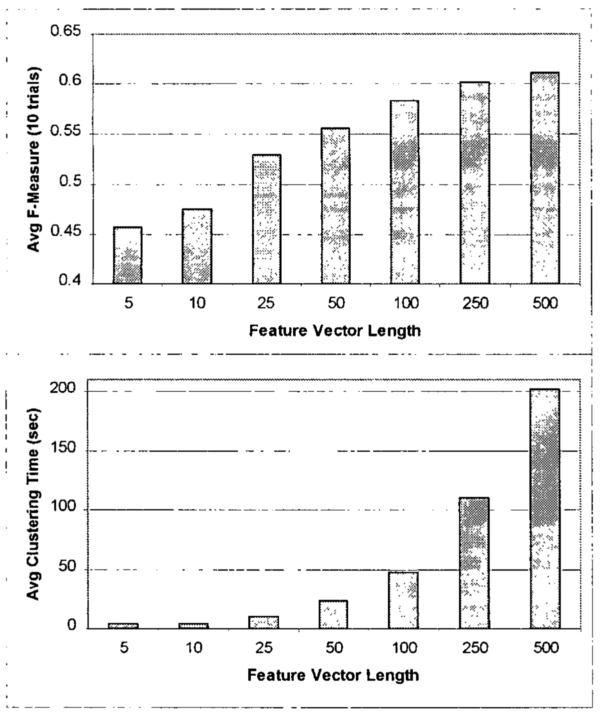


Figure 3 - The effects of feature vector length on clustering quality and time

compared with buckshot and fractionation, as one might expect. Random is significantly faster, however, requiring 1.7 seconds to run compared to buckshot's 11.9 seconds and fractionation's 90.0 seconds. The more significant observation is that a single iteration of continuous center adjustment brings all the scores into a close range, regardless of the seed selection technique used. Additional iterations (c-2, c-4) have almost no additional effect on cluster quality. Adding this single round (c-1) of center adjustment is computationally quite reasonable – clustering times jump to 5.7 seconds for random, 14.3 seconds for buckshot, and 110.8 seconds for fractionation. While all these time differences may seem inconsequential, they become much more significant when clustering gigabytes of text.

Non-continuous center adjustment has similar time requirements to continuous, but produces quite different F-Measure scores when random seed selection is used. Multiple iterations of center adjustment are required to bring random seed selection scores anywhere near the scores from buckshot and fractionation. The reason is that the initial cluster centers are not updated until after the entire iteration over the corpus; if the initial seeds are a poor representation of topics in the corpus, few documents will be assigned to them, and they will remain poor.

We found that the choice of k does not affect F-measure – for typical document sets, where the number of documents is much larger than k, setting k to whatever cluster size the user finds easiest to browse should not affect overall quality. However, smaller values of k create deeper hierarchies, because there are fewer items on each level.

Though our experiments yielded similar F-Measures for random, buckshot, and fractionation seed selection (using continuous center adjustment), we found that users' ease of comprehension of the respective output hierarchies differs. Browsing hierarchies created using fractionation, we usually observed individual documents at the top levels of the hierarchy more often than for the other seed selection techniques. This makes it harder to understand at a glance the topical composition of a corpus. Our explanation is related to the fact that most real corpora have a number of outlier documents whose vectors have no similarity to any other document vector. Fractionation examines the entire input set to generate seeds by agglomerating groups of documents together until only k remain. If there are small groups of outliers, the algorithm cannot join them with any other clusters, thus these outliers sometimes contribute to the initial seeds. This may also explain why fractionation, though more thorough, did not consistently outscore random and buckshot seed selection. We believe that the outlier problem could be minimized by implementing a dynamic stopping point in the cluster subroutine, allowing it to stop with more than k items remaining, to avoid including outliers as centers.[7]

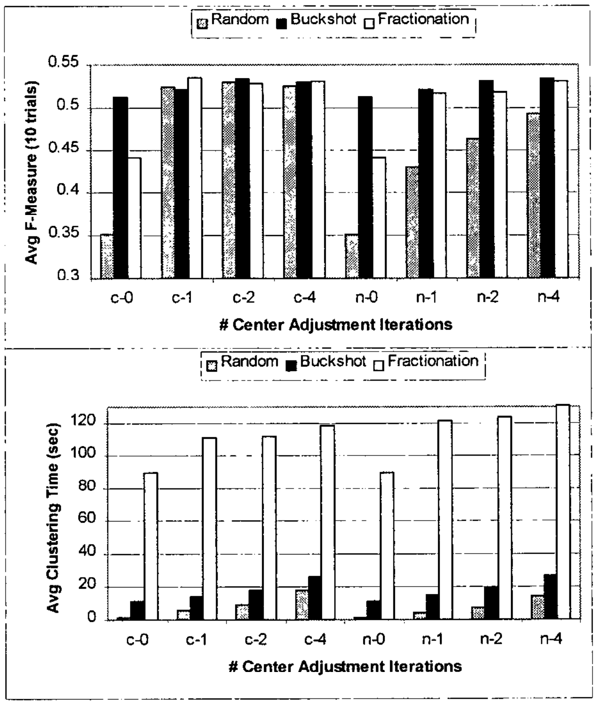


Figure 4 - The effects of seed selection and center adjustment on clustering quality and time

To support the theory that outliers can degrade the performance of fractionation, we experimented on some small document sets generated by hand with a "clean" outlier-free set of distinct topic groups. Fractionation consistently outperformed random and buckshot on these corpora. We plan to investigate this more formally in the future.

[7] Currently, if the set of clusters cannot be agglomerated to less than m clusters, where $m > k$, then the top k largest clusters are chosen.

3.4.2 *Vector Average Damping Results*

Our experiments show that vector average damping yields increased cluster quality without additional computational time. For example, Figure 5 shows that when random seed selection is used with two iterations of continuous center adjustment, adding vector average damping increases the F-Measure from 0.494 to 0.529. It also lowers standard deviation from 0.017 to 0.008, suggesting that this technique can make results more predictable. In this case, a damping value of 0.1 was used, as it seems to

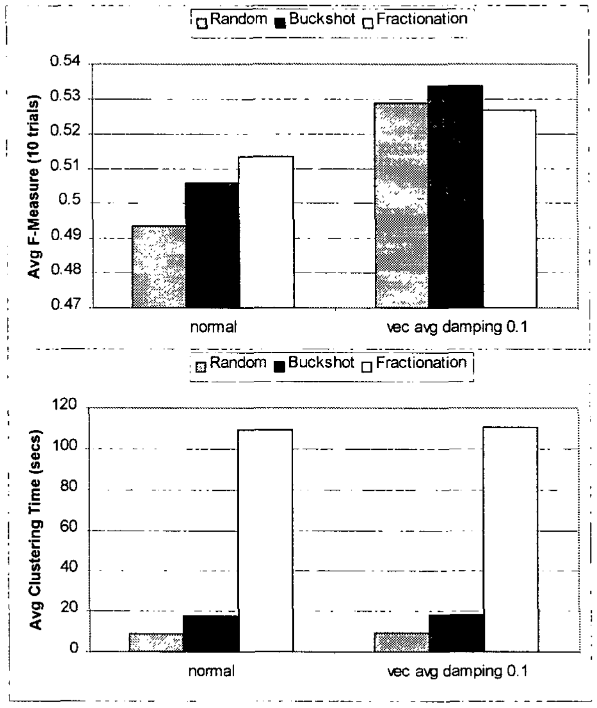


Figure 5 - The effects of vector average damping on clustering quality and time

produce to best results for our system's configuration. Because each cluster center's weight was multiplied by 0.1 when averaging in a new document vector, the new document vectors were able to have more impact on the centroids than they otherwise would; this technique seems to let the algorithm "home in" on the best cluster centers more effectively. To ensure that the effects of this damping were not related to our truncation of feature vectors, we repeated the tests for vectors of length 500. With the longer vectors, we saw the same relative increase in scores, so the technique does not appear to be just correcting a truncation-related loss in quality.

3.4.3 *Cluster Refinement and Quadratic-Time Clustering Results*

We conducted other experiments intended to study the effects of split/join and to compare near linear time clustering algorithms to quadratic-time alternatives. Figure 6 shows the results when using two iterations of continuous center adjustment with vector average damping.

We found that a fixed threshold is ineffective for the join algorithm. Toward the top of the hierarchy, very few clusters are joined because these items tend to be the most dissimilar. Deep in the hierarchy, the documents are more closely related, and the same fixed threshold tends to join too many clusters, often combining them all to form one cluster. We did not arrive at a threshold algorithm based on depth or on cluster variance that was consistently effective. Split/join was most effective when joining clusters until a fixed number, *k*, remained; the results we report are from this technique.

Average scores for random, buckshot, and fractionation center adjustment are all in the 0.53 range. Adding a single iteration of split/join for cluster refinement brings scores to almost 0.56. The disadvantage of split/join is that it adds substantially to clustering time; for the random seed selection case, clustering time jumps from 9.7 seconds to 41.4 seconds. We found that additional split/join iterations did *not* increase scores further.

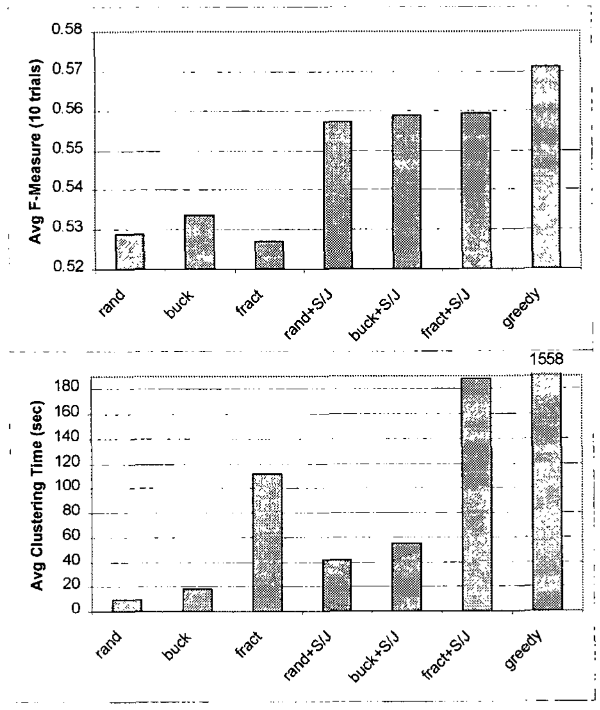


Figure 6 - A Comparison of clustering techniques

Greedy agglomerative clustering applied to the entire document set scores better than any of the other algorithms. However, its cost of 1558 seconds is substantial. Worse still, because this is an $O(N^2)$ algorithm, clustering time will be dramatically higher for larger document sets. This algorithm clearly does not scale to large text mining problems, but may be appropriate for some small corpora.

4. SUMMARY AND FUTURE DIRECTION

We have described our text clustering system, including a number of algorithms and an evaluation of their effectiveness. We used F-Measure (a combination of precision and recall) to gauge the quality of the generated hierarchies, averaging the results over ten trials.

Our results support two feature extraction techniques. First, weighting extracted terms using *tfidf* produces better results than using term frequency alone for all corpora but those with the smallest number of documents. Second, truncating feature vectors saves time (clustering time and vector length appear to be very roughly proportional), at the expense of cluster quality.

We evaluated several algorithms for seed selection, center adjustment, and cluster refinement. For seed selection, random is fastest, closely followed by buckshot, and fractionation is much slower. With no center adjustment or with non-continuous center adjustment, buckshot and fractionation significantly outperform random seed selection. However, continuous center adjustment seems capable of creating equally good partitions regardless of seed selection technique. We also introduced a novel center adjustment technique, vector average damping, that consistently increases cluster quality without costing additional time. Another way to consistently increase scores is performing cluster refinement using split/join, but this algorithm is relatively slow. We conclude that random or buckshot seed selection with a single iteration of continuous center adjustment using vector average damping offers the best time/quality tradeoff.

Another experiment compared these near linear time clustering techniques to greedy agglomerative clustering. Though the latter produces the highest scores of any of our algorithms (for a given feature vector length), it is prohibitively slow and does not scale beyond small corpora.

In the future, we plan to improve upon our evaluation method in several respects. We would like to conduct a user-oriented evaluation in addition to using automatic scoring techniques, in order to confirm that increased scores do directly benefit users. However, due to the large quantity of parameters in our system, such an evaluation would need to focus on only a handful of variables. We might use a similar method to [9], who measured how well users could understand the topic structure presented to them during cluster exploration experiments. We also plan to do scoring on additional test corpora; the system was designed to scale to gigabytes of text, but unfortunately currently available topic-labeled collections are nowhere near that large[8].

Because a single document frequently belongs in more than one topic category, we also plan to enhance the system to handle automatic clustering of a document to multiple topic clusters. Finally, we plan to apply our clustering algorithms to structured data instead of text, and reevaluate their effectiveness in this new type of feature space.

5. REFERENCES

[1] Cutting, D., Karger, D., Pedersen, J., Tukey, J. "Scatter/Gather: a Cluster-based Approach to Browsing Large Document Collections", in Proceedings of the 15th ACM SIGIR, 1992.

[2] Cutting, D., Karger, D., and Pedersen, J. "Constant Interactive-Time Scatter/Gather Browsing of Very Large Document Collections", in Proceedings of the 16th SIGIR, 1993.

[3] El-Hamdouchi, A., and Willett, P. "Hierarchical document clustering using Ward's method," in Proceedings of the 9th ACM SIGIR, 1986.

[4] Faber, V. "Clustering and the Continuous k-Means Algorithm", Los Alamos Science, November 22, 1994.

[5] Harman, D. and Voorhees, E., editors. Proceedings of the Fifth Text Retrieval Conference (TREC-5). National Institute of Standards and Technology, Department of Commerce. 1996.

[6] Hearst, M. and Pedersen, J. "Reexamining the Cluster Hypothesis: Scatter/Gather on Retrieval Results", in Proceedings of the 19th ACM SIGIR, 1996.

[7] Lagus, K., Honkela, T., Kaski, S., and Kohonen, T. "Self-Organizing Maps of Document Collections: A New Approach to Interactive Exploration," in Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, 1996.

[8] MacQueen, J. "Some methods for classification and analysis of multivariate observations," in Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability. Volume I, Statistics. Edited by Lucien M. Le Cam and Jerzy Neyman. University of California Press. 1967.

[9] Pirolli, P., Schank, P., Hearst, M., and Diehl, C. "Scatter/Gather Browsing Communicates the Topic Structure of a Very Large Text Collection," in Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems, 1996.

[10] Sahami, M., Yusufali, S., Bal-donado, M. Q. W. "SONIA: A Service for Organizing Networked Information Autonomously," in Proceedings of Digital Libraries 98, 1998.

[11] Salton, G. Automatic Text Processing – The Transformation, Analysis, and Retrieval of Information by Computer, Addison-Wesley, Reading, MA. 1989.

[12] Schutze, H. and Silverstein, C. "Projections for Efficient Document Clustering," in Proceedings of the 20th ACM SIGIR, 1997.

[13] Yang, Y., Pierce, T., and Carbonell, J. "A Study on Retrospective and On-Line Event Detection," in Proceedings of the 21st ACM SIGIR. 1998.

[14] Zamir, O., Etzioni, O., Madani, O., and Karp, R. "Fast and Intuitive Clustering of Web Documents," in Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, 1997.

[8] We have not found a *research-oriented* topic-labeled corpus larger than Reuters. An alternative might be to generate a large test corpus from Usenet postings, using the Usenet group names as the topic labels.