Online Clustering of Linguistic Data

Lev Reyzin
Princeton University Class of 2005, BSE
Junior Independent Work
Advised by Professor Moses Charikar

Abstract

Clustering text data online as it comes in is a difficult problem. It is both hard to capture a meaningful notion of linguistic similarity and to cluster large amounts of data in a single pass. This problem is especially challenging because most known algorithms that ensure tight clusterings are inefficient on large datasets. While significant work has been done on text clustering, it has not been fully explored. In this paper, we discuss previous methods in text clustering and then develop a single-pass text clustering algorithm designed specifically for clustering news stories, (but more widely applicable) and examine its empirical behavior. We then analyze some of its key design features and compare them to possible alternative methods. Finally, we discuss possibilities for further improvement of our algorithm.

1 Introduction

The problem of clustering involves grouping data into clusters by some measure of similarity defined for every pair of data points. The objective is to cluster the data in such a way as to minimize the intra-cluster data point distances while maximizing inter-cluster distances. The problem of separating data into disjoint sets such that pair wise distances within sets are bounded by a constant is NP complete [6], and since most formulations of the clustering problem are NP hard, it is hard to find optimal clusterings of data under most criteria.

Clustering linguistic data calls for some meaningful measure of linguistic similarity. A distance function needs to be defined that not only is mathematically meaningful, but also approximates what we humans believe to be a good notion of linguistic similarity. The specific instance of linguistic data we want to focus on is news stories. News stories are interesting to study because clustering them involves changing clusters over time to reflect the most current news and adding them to appropriate clusters as they come in. Given the large size of possible datasets, it is not possible to store all of the data, and given the hardness of the problem, we only attempt to find and approximate solution that will give us an intuitively good clustering. Since data comes in over time, this clustering must *fade out* old clusters as time passes and give more weight to the newest data, an idea previously studied [12]. We must also rank the clusters so that we know which news stories are most relevant.

Solving this problem would further advance methods in new event detection [2] as well as tracking important news stories over time. Google News, `news.google.com`, has started doing this and is still in the beta stage and already uses great popularity. Google claims that if a computer, instead of a person, computes a view of the most pertinent news stories, the results will be more trustworthy because human bias will be eliminated. Google, however, does not do its clustering in a truly online form, updating its site only periodically, though often. Furthermore, Google ignores all news over thirty days old even if it is still pertinent. Google does not make its algorithms publicly available, so we do not know exactly what they do. However, our goal is to be able to cluster news very quickly using a standard desktop machine. Finding a good way to do computerized clustering would allow for many applications including helping people file text data on computer, find patterns in email [12], keep track of the most important news, etc.

The key to the approach we develop in this paper is to use simple representations of documents and clusters as vectors in high dimensional vector spaces and to compute cosine distances between them. Using these distances, we cluster the articles in a single pass, which gives certain running time guarantees, and we increase cluster weights (or importances) when new articles are added and reduce the weights over time to simulate news events becoming less relevant.

In this paper, we will present an overview of clustering and its development over time. We will then present the details of our clustering approach and our implementation choices and briefly talk about approaches we tried that failed. We then examine the results of tests run on sample data and comment on future work to be done in this area.

2 Previous Work

Clustering is in essence classifying and any systematic classification. The best-known such system, taxonomy, was invented by Linnaeus in 1753. The first important work on numerical classification was by Sokal and Sneath in 1963 and a mathematical analysis was done by Jardine and Sibson in 1971. The first important application to linguistics was made by Dyen et al in 1967 measured similarities of languages by comparing distances of over a hundred select words in various languages [7]. More recently, clustering techniques have been used for pattern recognition by Anderberg in 1973, information retrieval by Rsmussen in 1992, and in a variety of other applications [10].

2.1 Clustering Algorithms

Specific methods that have been developed that are commonly used for clustering are divided into two categories: hierarchical and non-hierarchical methods. Hierarchical methods involve producing hierarchies of classification of all the data points. Non-hierarchical methods simply divide the data into clusters. The latter is the result we will work to obtain, so we will focus on two examples of previous work in non-hierarchical clustering. These examples of past research are drawn on heavily in our design.

Single Pass clustering is the most basic clustering method. In single pass clustering, for each new document, if it is within a threshold distance of a cluster, add it to the cluster. Otherwise, it begins a new cluster. The number of comparisons of single pass clustering for n data points is $O(n^2)$. In the worst case, each data point creates a new cluster so each data point must be compared to every other. With efficient data structures, it is often possible to get the worst-case running time lower for some types of data [11]. If the number of clusters is m, the number of comparisons time is $O(nm)$, and if m is bounded by a constant, $O(n)$. We will use a variant of this algorithm in our design.

K-Means clustering was developed by MacQueen in 1967. The idea of this algorithm is to randomly choose k centers for clusters and to assign each data point to the closest center, iterating and reallocating points to different clusters until the clustering converges. When assigning data points (represented as vectors) to a cluster c, the cluster is represented via its centroid.

$$\vec{\mu}(c) = \frac{1}{|c|} \sum_{\vec{x} \epsilon c} \vec{x} \tag{1}$$

We will see a similar heuristic used to identify the clusters when analyzing our algorithm. The running time of k-means was shown to be at most or more efficient than $O(n^2)$ [11].

Various other clustering methods have been developed including *K-Median* and *K-Center* [3] (which are very similar to the methods described above. Hierarchical clustering methods include *HAC*, or Hierarchical Agglomerative Clustering, which starts with all instances in separate clusters and repeatedly joins most similar clusters until only one remains with complexity $O(n^2)$. *Single-Link Clustering* computes cluster similarities by using maximum similarities of pairs in different clusters $sim(c_i, c_j) = max_{x \epsilon c_i, y \epsilon c_j} sim(x, y)$ and *Complete Link* uses minimum similarities $sim(c_i, c_j) = min_{x \epsilon c_i, y \epsilon c_j} sim(x, y)$. *Group Average Clustering* merges clusters by average across all pairs within a cluster [11]. Other known methods include the Buckshot Algorithm, Expectation Maximization, etc.

2.2 Linguistic Distances and Similarities

The most commonly used distance measures in text clustering are measures we have seen elsewhere. A few examples include *taking the intersection distance* of the two sets of words

$$d(A,B) = 1 - \frac{A \cap B}{A \cup B} \tag{2}$$

converting the data to vectors and taking the *Euclidian distance*

$$d(\vec{x},\vec{y}) = \sum_{i=1}^{m}(x_i - y_i)^2 \tag{3}$$

less common is the *L1 norm*

$$d(\vec{x},\vec{y}) = \sum_{i=1}^{m}|x_i - y_i| \tag{4}$$

and *cosine distance*

$$d(\vec{x},\vec{y}) = 1 - \frac{\vec{x}\cdot\vec{y}}{|\vec{x}|\cdot|\vec{y}|} \tag{5}$$

On top of these basic distance measures, more recent research has included methods for using the word-sets of the articles to determine their similarities. Term Frequency Inverse Document Frequency, or tf.idf, weights each word according to how frequently it is expected to appear in the dataset, the intuition being that more common words should be weighted lower because they contain less information than rare ones. Doing this transformation before applying a known distance function would theoretically make the distance measures more meaningful. Other methods developed by Sable and Hatzivassiloglou only use key significant parts of speech showing that basing similarities simply on key, limited information could yield more accurate results when clustering news articles [8]. Recent studies have included clustering of text, news, and email. A particular study to note focused on detecting bursts in email topic streams. [12]

Additionally tools have been developed to extract grammatical features from text. *Linkit*, developed at Columbia University, uses part of speech information to link phrases like Bill Clinton and President Clinton together so that they may both count as the same word when computing distance calculations. Another tool, *Nominator*, developed at IBM, categorizes proper names into classifications of `person`, `place`, and `organization`, allowing clustering algorithms to extract further information from the dataset [8]. *Wordnet*, developed at Princeton, allows access to linguistic information of English words including their common usage, all possible meanings, and word rarity.

In general, traditional and recent methods involve the creation of a vector or word set based on the words contained in the article and an examination of the features of the words. This includes looking at the meaning, structure, commonness, position, and part of speech of the words. Then distances are computed using appropriate distance functions. [9]

3 Approach

Our goal was to devise both a meaningful measure of similarity and to choose an algorithm that would both cluster news articles and weight the clusters in such a way that would display the most pertinent news first. We chose the realm of standard news articles like in the New York Times or the Economist but made our algorithm general enough to be able to handle a wider range of linguistic data. This first involved a study of the structure of news articles and a determination of what types data the algorithm would have access to and use. Second, we studied the clustering Google News generates to get a sense of what good news clustering is. We assume Google News generates a meaningful clustering given that it is currently a popular source of news.

This leads to a more general issue of what good document clustering is. Even if a clustering is mathematically tight, such a clustering may still not represent what we humans consider a good clustering. The ideal would be to have both mathematical and human indications that a clustering is good in order to be confident. Joachims [11] suggests a good method is to expose the clustering to the internet or a different market to have a representative view of human opinion. However, large psychological and economic studies such as this were outside the scope of this research, so notions of good clustering were limited to mathematical and intuitive methods.

3.1 Data Representation and Proximity Measure

The first task is to find an appropriate data structure for the task. This will largely depend on the data, but even more so on the proximity measure used. For example, if the proximity measure takes the Euclidian distance between two vectors, then the data should be stored as vectors and in some structure that would support that operation efficiently.

Word Filtering: We first decided to use a simplified version of two previously explored ideas (discussed in “Previous Work”, tf.idf and limited word filtering. We wanted to filter out the words in a news story that would not carry any meaning since including them in similarity computations could only add noise to the distances. Hence, we removed all of the common prepositions, linking verbs, pronouns, etc from the articles. Further, we removed all non-character symbols from the articles since we wanted to measure the content carried in the words themselves. This approach does start to focus on news article similarity over other linguistic data, however filtering common words also applies to a variety of other fields.

Simple Intersections: We represented the articles as sets of words and took simple intersection distances between documents (by equation 2) and used those as a linguistic similarity measure. This method had the desirable property satisfying the triangle inequality, which helps with intuitive thinking about similarity. We wanted to see how this compared to our notion of whether two articles were similar. We judged that two articles that have similar content will use the same words, however this method on its own failed to give tight clusterings as will be further discussed in the “Results” section.

Word Repetition: We realized that one reason intersection distances did not work was because we were taking the intersections of the sets of words. Thus, any word repeated multiple times within an article only was counted once in the set computation. So, if a word was mentioned once in one article and a hundred times in another, both would only have the word in the set once. We then modified this approach to count repetitions of the same word as different elements in the word set with the intersection functions matching as many instances of each word the two articles had in common. However, we noticed another key property of this computation of distances, that

$$sim(A,B) \leq \frac{min(|A|,|B|)}{max(|A|,|B|)} \tag{6}$$

This means that no matter how similar in content, a small article and a large article could practically not be similar.

Vectors: We changed the idea behind the distance computation altogether. We could not fix our problem of small-large article distance disparities by staying with our previous approach, so a more traditional method seemed appropriate. We decided to store each article as a vector in high dimensional space. Each possible word would represent a dimension in this vector space. If an article contained a word, it would have 1 added to that dimension. At the end, the vector would be normalized. Hence, articles containing many different words would be comprised of many components but each of small value since the vector would be normalized. Articles with few words would contain few components, but each would have greater value. The distance function we decided to use was the cosine distance (equation 5). Since $|\vec{x}| = |\vec{y}| = 1$, this distance function becomes

$$d(A,B) = 1 - \vec{x} \cdot \vec{y} \tag{7}$$

Hence, when comparing two vectors, each component of one is multiplied by the same component in the other vector. This means that when you are comparing a small document against a larger one, fewer components are multiplied together, but they are multiplied by higher values due to the small dimensioned vector. This mitigates the document size disparity effect.

Logarithmic Scaling: The final approach was keep the previous method but to do logarithmic scaling of the vector dimensions. When adding an extra word a dimension, previously, we would just add one to that dimension. Now, dimension values are updated by the following equation before normalizing the vector:

$$v \leftarrow \begin{cases} 1 & \text{if } v = 0 \\ ln(e^v + 1) & \text{if } v > 0 \end{cases} \tag{8}$$

So, the same word is added, the vector length in that dimension follows the sequence

$$1, 1.31, 1.55, 1.74, 1.90, 2.04, \ldots$$

This intuitively makes sense. Supposing an article mentioning China 100 times is compared to another article mentioning China only once. If one article mentions China more than the other, it should be factored in. However, if one article mentions China 100 times while a second mentions China 50 times, they should be more similar than not. Logarithmic scaling makes that compromise. This is the linguistic similarity function we used.

3.2 Clustering Algorithm

As part of the specification of this research, we focused on clustering news articles as they come in. Given that there could be a large amount of data coming in, it would not have been preferable to store all of the news articles in the program's memory. Hence, we were left with the choice of designing a single pass clustering algorithm that would not store the articles but would only store the clusters. We also wanted to keep the running time down.

Simple Single Pass: We first implemented a straight single pass clustering algorithm as described in section 2.1. For the dataset, we experimentally found an appropriate threshold function to determine whether or not to put a document into a cluster. Any document that passed the threshold for a cluster was included in the cluster. Hence, a document was allowed to be added to more than one cluster. This method, while simple, generated encouraging results. However, as news articles changed in content, they at one point stopped being included in their appropriate clusters. Take the example of wanting to have a clustering such that all of the news regarding grade inflation at Princeton was clustered together. If at first, the news articles talked about the proposal, then talked about the faculty vote, and finally about the reactions of the students to the vote, the cluster would be defined by the first articles about the proposal. Thus, when the articles start to explain student reactions, they might not fall into the cluster because the nature of the news changed over time, but the cluster did not.

Evolving Clusters: The solution to this problem was the following. We kept the same algorithm, but as data were added to clusters, we updated the vectors representing the clusters in the following way. Take N to be the number of articles in the cluster, $\vec{c}$ to be the vector representing the cluster, and $\vec{v}$ to be the vector being added to the cluster.

$$\vec{c} \leftarrow \frac{N}{N+1}\vec{c} + \frac{1}{N+1}\vec{v}$$

$$\vec{c} \leftarrow \frac{N\vec{c} + \vec{v}}{N+1} \tag{9}$$

Then the vector would be normalized again. This is similar to the idea expressed in equation 1. By this scheme, each new component modifies the cluster less and less the more articles the cluster represents. With this scheme, each cluster is represented by all of the articles that were included in it, solving the problem posed in the straight single-pass approach. This however created a new problem. Since the same article sometimes passed the threshold of many clusters, articles put into the same clusters modified all of those clusters to be closer to it. Hence, clusters started looking similar to each other. To fix this, we modified the algorithm to only add each article only to the *closest* cluster if it passes the threshold.[1]

Fading Out: We were now faced with the problem of the clusters needing to die out over time. The previous scheme allowed for clusters to evolve to represent new data, but the clusters also needed to gradually disappear if new data were not being added to them. We decided to use the following scheme: a cluster starts out with weight of one. Every time an article is added to it, its weight increases by 1. After each time-step (however it is defined) the weights of all clusters decrease by one. When a cluster reaches weight 0, it disappears. Using this method would make clusters to which articles are not added at a high enough rate disappear and those to which articles are added often be reinforced. For a cluster of weight W, we modified the cluster update rule to be similar to equation 9

$$\vec{c} \leftarrow \frac{W\vec{c} + \vec{v}}{W+1} \tag{10}$$

so what matters to how much weight a new article has in defining the cluster is not the number of articles in the cluster, but how important it is (which is related to how many articles are in it or have been added recently). This approach worked, but it did not guarantee any running time bounds for the algorithm. If each processed article created a new cluster, then there would be $O(n^2)$ comparisons.

Limiting Clusters: We wanted to maintain no more than 40 clusters (and usually fewer) to guarantee running time bounds and to make it so that the weight decreases are a function of the size of the input. We changed the weighting scheme as follows: a cluster starts out with weight equal to one, and as each new data point is added to a cluster, its weight still increases by one. However, at each time-step the weight of every cluster would decrease by $\frac{1}{40}$ times the number of clusters added at that time-step. An equivalent view for an analysis would be that for every 40 articles added, the weight of each cluster decreases by one. When a cluster reaches weight zero, it disappears.

Using this approach, only 40 clusters are sustainable. To show this, take the potential of the clustering to be equal to the sum of the weights of all of the clusters. After 40 articles are processed, the potential rises by at most 40 and falls by however many clusters exist, call that k. Hence, the increase in potential after 40 operations $= 40 - k$. Hence, the average number of clusters over time is less than k since the potential of a clustering cannot be negative. Hence, this algorithm has to compare n articles, each against at most 40 clusters, making at most $40n$ comparisons, which is $O(n)$ comparisons.

[1] an approach to add articles partly to different clusters did not seem encouraging either

Dynamic Thresholds: The final issue we wanted to tackle was the threshold that determined whether or not an article should be in a cluster. The threshold had to be experimentally determined in our previous approaches for each dataset individually. However, hard-coded thresholds pose a problem. Human experimentation should not be needed to cause an algorithm to work. Further, if the character of the data were to change over time, one would have to find a new threshold manually. We decided to try several techniques to solve this problem. One possibility was to include articles in clusters with probability proportional to their similarities to the closest cluster. This approach failed because once an unrelated article randomly became a part of a cluster that it was not too similar to, it would change the vector for the cluster, and eventually clusters became meaningless. Another idea was to vary the threshold to keep the ratio of articles included in clusters to the number of articles excluded from clusters constant. However, the ratio that worked for the clustering varied from dataset to dataset, so this approach just moved the manual setting of the threshold to the manual setting of the ratio.

The approach we used was to modify our algorithm in the following manner. Suppose we want to have k clusters of n data points. Assuming each cluster has the same number of data points, there are k clusters. We want to compute what fraction of distance computations are intra-cluster distance computations. Within each cluster, there are $\frac{n}{k}$ points. So the number of intra-cluster distances within one cluster is

$$\begin{aligned} & \binom{n/k}{2} \\ = & \frac{\frac{n}{k}!}{2!(\frac{n}{k}!-2)!} \\ = & \frac{1}{2}(\frac{n}{k})(\frac{n}{k}-1) \\ \approx & n^2/2k^2 \end{aligned}$$

So, within k clusters, the number of distances is approximately

$$\begin{aligned} & (n^2/2k^2)k \\ = & n^2/2k \end{aligned} \tag{11}$$

Now the total number of pair wise distances in the entire data set is

$$\begin{aligned} & \binom{n}{2} \\ = & \frac{n!}{2!(n-2)!} \\ = & \frac{n(n-1)}{2} \\ \approx & n^2/2 \end{aligned} \tag{12}$$

So the number of intra-cluster distances over the number of total distances is

$$\begin{aligned} & \frac{n^2/2k}{n^2/2} \\ = & \frac{1}{k} \end{aligned} \tag{13}$$

Hence before the clustering begins, the program samples 1000 random distances and takes the $\frac{1}{k}$th distance to be the threshold. For our analysis before, we used 40 to be the number of clusters. However, since clusters are not all exactly equal in size and because of other factors like weights changing the computation, this approach worked for ≈ 67, and $\frac{1}{k} = .015$. This held for different datasets and removed the need for setting a manual threshold.

4 Methodology

To implement the clustering algorithm, our program was divided into three large components. One component ran the clustering, another component computed similarity, and the final component ran an evaluation of the clustering algorithm. In this section we will describe only our implementations of the final algorithm we used. Our implementation was made to perform the tasks needed and to keep the asymptotic running time as low as possible. Since this research focused more on the algorithms than on the implementation, we will only describe the more interesting parts of the implementation.

The clustering component of the algorithm was implemented in a straighforward manner. Each vector was compared to each other vector and was added to the closest cluster if it passed the threshold. Since clusters were represented as vectors in high dimensional space, it would have been hard to not go through all of them for each new data point. More importantly, since as we have shown that the number of clusters is constant amortized, the number of comparisons remains $O(n)$ nevertheless.

Each article was stored as a vector which we implemented as a hash map, with words as keys and their values as the length of that dimension in the vector. When comparing two articles, for each word in the article of smaller dimension (and thus of fewer words), its value in that article was multiplied by its value in the other article. Each value lookup took constant time given the properties of a hash table, so the lookup speed was linear in the number of words in the article of smaller dimension. We bounded this by limiting articles to reasonable word length and cutting them off if they exceeded it. Normalizing the article required summing up the values of each key in the hash table and dividing each key by the sum, which again took linear time in the number of different words in hash table.

The final part of the implementation was a component that collected data about the clustering. We wanted to have some empirical method of seeing whether the clustering good. This component kept track of the spread of the linguistic similarity measures, the average intra-cluster distances, the average inter-cluster distances, the spread of data inside and outside the clusters, etc. Whenever a data point was put into a cluster or a distance was measured, it was processed by the data collector. Furthermore, the data collector was made so that it would not contribute to increasing the running time. While this was not a problem for most computations, we will look at one instance for which the data collector needed to use some clever methods. The usual method for determining spread involves a variance computation

$$V(X) = E(X - \mu)^2 \tag{14}$$

where

$$E(X) = \frac{\sum_{i=1}^{n}(x_i)}{n} \tag{15}$$

This would force all of the data to be stored because within each sum the computation requires the average of all the data. However, we can also compute the variance through a clever shortcut

$$V(X) = E(X^2) - E(X)^2 \tag{16}$$

This way, we only have to store a running sum of the distance measurements and their squares, as their count to compute their spread. [5]

5 Results

Given the difficulty of coming up with criteria for good clustering, it is even more difficult to come up with metrics to evaluate a clustering algorithm. However, there are some objective measures we can consider to be needed in order for a clustering to be good. We will focus on three criteria and evaluate our algorithm based on all three. The first is that the linguistic similarity measure needs to be meaningful. The second is that the average intra-cluster distances have to be significantly smaller than average inter-cluster distances. The third is that clusters should be roughly equal in size. Finally, we should also evaluate the clustering based on our intuition for what a good clustering is.

We ran our algorithms on various news datasets mostly comprised of articles from the Economist (about 200). We also ran our final algorithm on a medical advice database (about 300 entries) and on Edgar Allan Poe's short works to see how it would cluster data of different linguistic sources.

Similarity Meaningfulness: The first metric by which we evaluated our results was on looking at the distances computed in the similarity measure. We took news articles and computed distances between random pairs to see how to make the similarity measure more effective. Using simple intersection distances, similarities between news articles ranged from 0.79 to 0.89. It is preferable for the distance function to use the entire range from 0 to 1, so a function where the difference between distances between similar articles and dissimilar articles differs by only 10 percent of the scale is not too meaningful.

Filtering the common words out made the distances range from about 0.8 to 0.95. While this made articles more distant to each other, it improved the range the similarity function was using. Taking into account multiple instances of words broadened the range even further to 0.75 to 0.95. However, switching to vector representations had the biggest impact. That, together with logarithmic scaling, gave us a range of 0.23 to 0.95 on newspaper articles. The distances are also well distributed through the range. The range for medical database entries was even greater, ranging from .07 to 1.0.

Intra and Inter Cluster Distances: The vector representation with logarithmic scaling also produces encouraging cluster tightness. What we do not want to happen is for the average intra-cluster and inter-cluster distances to be too close to the threshold because it would make the threshold a less meaningful separator of the data. For news sources, experimental results settled on a threshold of a .58 distance, while the average intra-cluster distance was .46 and average inter-cluster distance was .78. Furthermore, the intra-cluster distance spread was less than the inter-cluster spread. These results were duplicated on the trials on the medical database, where the average intra-cluster distance was 0.47 and the average inter-cluster distance was .85. Finally, the dynamically determined thresholds the algorithm found for the news, medical, and short story data were all different, .58, .54, and .28 respectively.

Relative Cluster Sizes: Given that articles do not evenly distribute themselves among all news topics, we do not want to see results showing all clusters to be of the same size. However, we do not want to see only one big cluster and no other ones. When we were adding articles to many clusters, such a result seemed to occur. However with the vector implementation, relative cluster sizes are very threshold sensitive. That was another reason why it was important to have a dynamically determined threshold. If a dynamically determined distance threshold increased by .05 or more, generally the data begins to form into one big cluster. If it is decreased by .05 or more then very few clusters form at all. This means that much useful linguistic similarity information is contained within a small crucial range of 0.1. However, it is a great improvement over the first implementation where the entire range of distances was within .1 on the scale.

Intuition: Even given the metrics, we want to be able to judge for ourselves whether a clustering is good. We were able to see that probabilistic cluster assignment was not working largely out of intuition. Similarly, when we were adding data to many clusters at once, intuitive looking at the clusters helped. We believe that the current model of clustering clusters stories into more appropriate categories. In the medical directory, certain trends of illnesses in clusters were visable. With news, a sample cluster contains the headlines: *Al-Qaeda, National security, The presidency, Searching for John Kerry's economic policy, Lexington, Iraq, Australia, George Bush's credibility, Iraq, Lexington, Science and the Bush administration, Now what?, Iraq, Bagehot, Central European nationalism, Iraq, Iraq's neighbours Iraq, George Bush makes his case.* While there are some elements that clearly do not belong, we could see a clear trend of issues of national security and the war in Iraq in the cluster. Other clusters are more precise: *China and Taiwan, Behind the mask, China's economy, Taiwan, China, Taiwan, Taiwan, The Caribbean and Taiwan*, where topics are almost indisputably clustered together correctly.[2] Finally, this clustering generated many of the same topic categories the Economist provided on its own.

[2] *Behind the mask* is an article on China

6 Discussion

The approach we took with this clustering was to use a fast clustering algorithm and to incorporate various heuristics and measures to make the clustering work well. We were able to experimentally determine some interesting properties of clustering and to use and improve some known methods. This approach certainly seems to have merit. What is especially encouraging about our approach is that it does not depend on user defined constants, yet appears to work on a variety of different data. Furthermore, there are some running time guarantees, that, coupled with clever implementation, could make for a fast, reliable clustering algorithm.

However, we also cannot claim complete success. To determine how well this algorithm works, it should be tested on larger datasets and with more rigorous analysis, the data for which we did not have the time to gather. Also, we would recommend that future research in this area included a component that tests human evaluation of the clustering by either running an experiment asking people to use the program or by putting it online and seeing responses from a worldwide audience like Google News does.

Furthermore, the sensitivity of the threshold implies that more could be done to make the linguistic similarity function produce even more meaningful results. This could be done by incorporating tf.idf fully or by using already created tools like *Linkit* and *Nominator*, both described in "Previous Work." It may also prove useful to apply machine learning techniques to finding proper threshold values.

If we want to handle larger article sizes, we can instead of limiting the words in the articles, use vector sketches [4]. More improvement could be done to finding more efficient data structures to hold the vectors themselves and the clusters as well.

Many lessons were learned in doing this research including how difficult it even is to evaluate whether results are promising. It was also interesting to see how small changes to an algorithm could have profound impact on its output and how sensitive thresholds could be. Also, it was surprising how much information was available on a topic considered so unexplored and how hard (and rewarding) it is to do research.

7 Conclusion

The task of clustering is a very difficult one but with many applications and uses. We have shown that while there are many barriers to overcome in doing single-pass clustering of linguistic data, it is possible to achieve reasonable results even with simple algorithms. We also analyzed various approaches to solving the clustering problem and have found some improvements that can be made to simple known algorithms to make them more effective. While much of the notion of similarity that humans comprehend can be captured by techniques such as word counting, advances in this task are likely to come with the use of more sophisticated algorithms that try to capture a more semantic view of similarity. Due to a recent emergence of interest in clustering, it is becoming evident that reasonable clustering can be done by computerized means, and our research reinforces this claim.

Acknowledgements

I would like to thank my advisor Moses Charikar for his advice and immeasurable input into this project. He generated many of the ideas I have expressed here including vector representations of the articles, sampling distances for the threshold, efficiently computing spreads, and others. This paper could not have materialized without his help.

References

[1] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Phillip S. Yu. *A Framework for Clustering Evolving Data Streams.* Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

[2] James Allan, Ron Papka, and Victor Lavrenko. *On-line New Event Detection and Tracking.* Center for Intelligent Information Retrieval. Proceedings of the 21st ACM-SIGIR International Conference on Research and Development in Information Retrieval, Melbourne, Australia, August 1998.

[3] Moses Charikar and Liadan O'Callaghan. *Streaming Algorithms for Clustering Problems.* Lecture, Princeton University.

[4] Moses S. Charikar. *Similarity Estimation Techniques from Rounding Algorithms.* STOC 2002, ACM.

[5] Jay L. Devore. *Probability and Statistics for Engineering and the Sciences.* Duxbury Thomson Learning, Pacific Grove CA, USA, 2000.

[6] Paul E. Dunne. *An Annotated List of Selected NP-complete Problems.* COMP202. Department of Computer Science, University of Liverpool, 2002.

[7] John Hartigan. *Clustering Algorithms.* John Wiley and Sons, New York, 1975.

[8] Vasileios Hatzivassiloglou, Luis Gravano, and Ankineedu Maganti. *An Investigation of Linguistic Features and Clustering Algorithms for Topical Document Clustering.* Department of Computer Science, Columbia University.

[9] Vasileios Hatzivassiloglou, Judith L. Klavans, and Eleazar Eskin. *Detecting Text Similarity over Short Passages: Exploring Linguistic Feature Combinations via Machine Learning.* Department of Computer Science and Center for Research on Information Access, Columbia University.

[10] A.K. Jain, M.N. Murty, and P.J. Flynn. *Data Clustering: A Review.* ACM Computing Surveys, Vol. 31, No. 3, September 1999.

[11] Thorsten Joachims. *Text Clustering.* CS630 Representing and Accessing Digital Information, Cornell University.

[12] Jon Kleinberg. *Bursty and Hierarchical Structure in Streams.* Proceedings of the 8th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2002.