Clustering and Summarization of Large Document Sets

Undergraduate Thesis

Andrew Hollis

May, 2018

Abstract

The intelligence community is awash in large quantities of data, and much of this data is text data. The need for tools that can help analysts quickly separate useful texts from less useful texts is becoming ever more critical. The goal of this paper is to explore a variety of clustering and summarization algorithms that can be used together to create systems that allow analysts to quickly get an idea of the major groups in a document collection and the issues discussed within these document groups. In order to evaluate the effectiveness of these algorithms we compare their performance to clustering and summarization results generated by human analysts.

1 Introduction

As our ability to collect and store data has increased in the digital age, the problem across many sectors of society has become how to make sense of the large quantities of data now available. One area of modern society where this need is particularly critical is in the national security intelligence community.

In order to effectively identify the threats facing our nation, intelligence officers have to be able to process large quantities of data quickly. They must have tools that allow them solve the "needle in the haystack" problem, the problem of finding the relevant information quickly when it is hidden in vast troves of data. Many statistical and machine learning techniques can be used in building these tools for intelligence officers. [5]

One of the most common forms data takes in the intelligence community is that of text. Thus, creating tools that can help intelligence officers find relevant textual information within a large set of text documents is of critical importance. Creating systems that give analysts a sense of what is in a document collection can reduce the amount of time an analyst must spend evaluating information and can improve an analysts ability to find critical information more efficiently.

With this motivation in mind, this thesis attempts to explore various text clustering and text summarization algorithms that can be used to create a system for making the process of analyzing sets of text documents more efficient.

In section 2, we explore how to represent text in vector formats that facilitate analysis and discuss how dimensionality reduction may improve the

clustering analysis. In section 3, we discuss three different text clustering algorithms and associated internal clustering evaluation metrics. In sectin 4, we discuss three multi-document summarization techniques that can be used to give analysts an overview of each text cluster. In section 5, we compare the results of automatic clustering and summarization with clustering and summarization performed by human analysts in order to determine if our system could mimic low-level human analysis and thus free up analysts to do more critical high-level analysis. In section 6, we briefly discuss a framework that uses the clustering and summarization techniques to create a system for interactively exploring the contents of a document set. In section 7, we end with some concluding remarks and possible direction for future research.

2 Data Cleaning and Transformation

2.1 The Data Set

In order to demonstrate and evaluate the use of the clustering and summarization algorithms we are using a declassified collection of texts from the Central Intelligence Agency that document the Soviet Invasion of Czechoslovakia in October 1968. The corpus contains 475 documents of various types. Some are short bulletins describing single events, some are longer analytical reports, others are raw transcripts from meetings or speeches. The documents also vary greatly in length. This is useful as it will allow us to observe how the clustering and summarization algorithms behave on a corpus containing many types of documents.

These documents were taken from the Internet Archive which has the documents available in text file format, which is the easiest format for reading in the text data and processing it.

2.2 Text Vectorization: The Bag of Words Model

In any clustering task, it is important to extract features from the data that can be used to judge the similarity between two objects. Most clustering algorithms will use these features to determine which data objects, in this case texts, should be in the same cluster. Good feature selection is critical for accurate clustering. [1]

One of the simplest and most popular methods for feature selection in text analytics is to use the bag of words model along with a feature selection metric. [21] The bag of words model breaks down a piece of text into its constituent words, thus creating a set or "bag" of words from the text. The principle disadvantage of this method is that we lose information about word order which is often very important to the overall meaning of the text. We have also lost information about potentially important multi-word combinations such as "United States" or "Soviet Union". These concepts are not presented as single data units in the bag of words model. Despite these disadvantages, the method is very simple and is used in many applications.

Once the text has been transformed into a bag of words, we transform this bag of words into an $n-$dimensional feature vector representation of the text. Let $\mathbf{X}$ be the bag of words model, and let f_i be some function that calculates the ith feature from the data, $\mathbf{X}$. We seek to transform the data into a vector of the form $[f_1(\mathbf{X}),\ f_2(\mathbf{X}),\ ...,\ f_N(\mathbf{X})]$.

One very simple choice for the feature functions is term frequency. We create a vocabulary of n terms from the data and then we define $f_i(\mathbf{X}) := \text{freq}(t_i)$, where $\text{freq}(t_i)$ is the number of times t_i, the ith term in the selected vocabulary, appears in the text. Figure 1 illustrates the idea behind this approach.

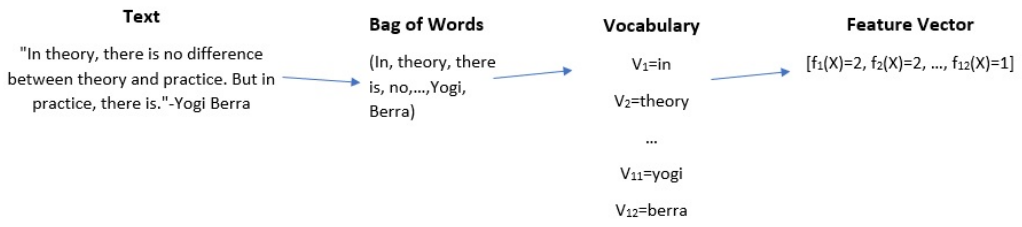

Figure 1: Bag of Words-Term Frequency Model

In this case the vocabulary was the total set of unique words in the quotation. Notice that the vocabulary is not a as long as the total number of words in the quotation, which is 16. It is 12, because certain words are repeated. This can be seen in the feature vector, which shows that the first word in the vocabulary, "in", and the second word in the vocabulary, "theory" both have a frequency of two.

Keep in mind that the goal of a feature is to help the clustering algorithms discriminate between data objects. For feature functions like the one above that correspond to words, the goal is often to assign weights to words from the text in accordance with their "importance" in the text and their ability to differentiate the text from which they originate from other texts. [20] On first inspection, it may appear that the term frequency approach seems to meet this criterion. For instance, if you have some documents that talk about economics and some documents that talk about cybersecurity, terms like "supply", "demand", or "production" will probably have high term frequency among the economics documents and low frequency among cybersecurity documents. In this case, term frequency would be useful for clustering the two types of documents.

2.3 Data Cleaning

Although raw term frequency is a good start, there are several ways in which the feature selection can be improved upon. To begin with, it is common practice to remove words that are common to all documents; these are usually referred to as stop words. [1]. Such a list of stop words might include "the", "and", etc. Because these words appear many times in every document, they do not help discriminate between documents, they are just "noise". Thus, they are commonly removed before further analysis is performed. For instance, in the quotation example in figure 1, we might choose to remove "in" or "and" from the set of features since these word are so common in so many documents.

It is also important to avoid creating features from words that really should not be considered as separate features. For instance, if a word is misspelled in the text. We do not want this data anomaly to be treated as a feature used for discrimination. Thus, it is recommended to have some way of correcting all words which are deemed to not be correctly spelled words before continuing with analysis. We use Peter Norvig's spell checker [17], which is considered a good base line spelling correction algorithm. The algorithm works by considering all "words" that have an edit distance of 1 or 2 from the word of interest. Suppose we are considering the word "werk". We can edit this word by deleting a letter, flipping the order of two adjacent letters, replacing one letter with another letter, or inserting a whole new letter. A word that has an edit distance of 1 from "werk" would be a word that only involves one of the above editing techniques. For instance "wrk", "wrek", "work", and "werko" all have an edit distance of 1 from "werk". Words that have an edit distance of

two from "werk" just involve two of the edits mentioned above. For instance, "wreck" (transposition and insertion) has an edit distance of 2 from "werk". After collecting all "words" that have an edit distance of 1 or 2 from the word of interest we use a very large English text corpus to determine which of the generated "words" are actually valid English words; these become our candidate spelling corrections. We then use the corpus to calculate the frequency of each of these candidates; the word with the highest frequency with an edit distance of 1 is set as the spelling correction for the word of interest. If there is no valid candidate with an edit distance of 1, the most frequent candidate with an edit distance of 2 is chosen. If there are no candidates with an edit distance of 1 or 2, the incorrectly spelled word is left alone and not considered further in the analysis.

We perform three other data cleaning operations to improve feature selection. When text is read in, all words are transformed to their lower case equivalents so that words are treated as the same feature whether they begin with an upper case letter or not. For instance, "war", "War", and "WAR" would all be treated as the same feature. We also perform an operation called lemmatization which maps different forms of a word into a single form. For instance, "soldiers" become "soldier", "invaded" becomes "invade", and so on. This makes it so that words that are conceptually the same but of a different form are treated as being representatives of the same feature. Finally, based on the content of the texts, we extracted several entities that are made up of several words and may go by several different names but should be treated as a single unit. For instance, "Soviet Union" and "USSR" should be treated as the same entity, and "Soviet Union" should not be broken up into "Soviet" and "Union"; it should be treated as a single object. Thus, every instance of "Soviet Union" or "USSR" is treated as a representative of the same feature. This same thing is done for other known proper entities in the text like "West Germany" (German Federal Republic), "United Nations", etc.

With only the relevant words left to be considered as features, we also have to consider how we will construct a finite vocabulary that can be applied to all documents in the corpus. In order to do this we chose to extract vocabulary from all central intelligence bulletins, weekly summaries, and intelligence memorandums. This vocabulary covered at least 75% of every document in the corpus. About 87% of the documents are more than 90% covered by the vocabulary. Only about 3% of all documents are less than 85% covered by the vocabulary. Overall, this is good coverage. We use a limited vocabulary instead of a vocabulary built from all the documents for data quality reasons.

The original intelligence documents were put through an optical character recognition (OCR) system in order to transform them into text files from which data can be easily extracted. The central intelligence bulletins, weekly summaries, and intelligence memorandums had the clearest OCR rendering and thus were less susceptible to misspellings or illegibility due to bad OCR rendering. The cleaner vocabulary built from the central intelligence bulletins, weekly summaries, and intelligence memorandums can be used to help recognize incorrectly rendered words in the other documents and correct them. If we used all the documents, the badly rendered words in the other documents would be included in the vocabulary and would never be corrected.

2.4 The Term Frequency Inverse Document Frequency (TFIDF) Transform

Once the data has been properly pre-processed, we can calculate feature scores and turn the text into vector form. The question then becomes: what is a good feature score? As mentioned previously, a very simple feature score would just be term frequency, but this method has certain inadequacies.

There are two major considerations to keep in mind when constructing a good feature score for text analytics applications. [20] These two considerations are recall and precision. Recall refers to the ability of a feature score to find relevant features or words. If a word is of particular relevance or importance in a document it should appear many times. Thus, term-frequency is a good metric for recall. The other consideration, precision, has to do with a feature score's ability to lessen the importance of irrelevant features in the analysis. Term frequency by itself is not a very good metric for precision [20].

To better understand this, consider the data set used in our study. It is a collection of documents about the Soviet invasion of Czechoslovakia. Thus, we expect words like "Soviet" and "Czechoslovakia" to show up many times in every document, but these terms are not helpful in distinguishing one document from another since they are quite common to all documents. If we have a subset of documents referring to the economic costs of the invasion and a set of documents about the reaction of youths in Prague to the invasion, we want words like "economy", "production", "protests", "youth", etc. to have a higher weight than "Soviet" and "Czechoslovakia" because such words are useful for distinguishing between documents that talk about the economic situation and documents that talk about political unrest.

A feature score with the recall power of term frequency but better precision is the term frequency inverse document frequency (tfidf) score. [20] The tfidf score is calculated as $tfidf(t) = tf(t) \times idf(t)$, where $tf(t)$ is the term frequency of term t in some document and $idf(t)$ is the inverse document frequency of term t across the whole data set of interest. The $idf(t)$ score is calculated as:

$$idf(t) = 1 + \log\left(\frac{C}{1 + df(t)}\right) \tag{1}$$

Where C is the total number of documents in the data set, and $df(t)$ is the the document frequency of t, or number of documents in which the term t is mentioned. Examining this score we can see that it essentially penalizes words like "Czechoslovakia" and "Soviet" that appear in many documents and thus have a high document frequency. Once the feature vectors are computed in this way, the vectors are typically normalized.

We will illustrate the complete TFIDF feature vector construction process with a toy example. Consider the following four "document" data set:

Document 1: the sky is blue

Document 2: sky is blue and sky is beautiful

Document 3: the beautiful sky is so blue

Document 4: i love blue cheese

Ignoring stop words like "the" we have a vocabulary of 8 unique words [sky, is, blue, beautiful, so, i, love, cheese].

The following table shows the feature vector calculation for document 2:

Word	TF	DF	TFIDF
sky	2	3	2
is	2	3	2
blue	1	4	0.7768
beautiful	1	2	1.288
so	0	1	0
i	0	1	0
love	0	1	0
cheese	0	1	0

The final document tfidf vector would be normalized so that documents of very different lengths would still be comparable. Notice that even though "blue" and "beautiful" appear the same number of times in the document they have very different weightings because "blue" shows up in all four documents, thus giving it a higher idf penalty, and "beautiful" only shows up in two documents. Once this tfidf representation has been calculated for all the documents, our data will be put in a term-document matrix, the standard data format for text analytics. The term-document matrix is an $n \times m$ matrix where n is the number of documents and m is the number of words in the vocabulary. Each of the tfidf feature vectors form the rows of this matrix.

2.5 Dimensionality Reduction: Latent Semantic Indexing

There still remains one other way in which to improve the performance of our clustering algorithms. It is possible to reduce the dimensionality of the data set in order to bring out some of the more salient features of the data set and reduce the impact of noisy features. [1] One way of doing this is through a technique know as latent semantic indexing (LSI). LSI is similar to principal components methods in that it uses singular value decomposition of the data matrix to extract the most important features of a data set. LSI was created to deal with the issues of synonymy and polysemy. [7]

Synonymy refers to the problem that arises when a two texts are treated as being dissimilar because they don't have the same words even though they mean essentially the same thing. For instance the sentence "Julius Caesar conquered the native tribes that once inhabited modern day England" would be treated differently from "The Romans invaded and defeated the ancient Britons." because there is no word overlap, but the sentences are talking about the exact same thing and should ideally be treated as very similar. This is a recall problem because there is a tendency to treat as dissimilar texts that are actually very similar.

Polysemy is essentially the opposite of synonymy. Polysemy is the problem that arises when several words overlap, but the texts in question aren't actually very similar in meaning. Thus, two texts are treated as being similar when they actually aren't. For instance the sentence "The thief escaped into the London fog after robbing the shop." is not at all similar in meaning to "I escaped the shop to pick up a London fog latte." Because of the great amount

of word overlap between these sentences, however, they might be treated as being very similar. This is a precision problem as it treats two texts as similar, or relevant to each other, when they are not.

The main lesson here is that word-based features aren't always the best feature choice for doing text analysis. The goal of LSI is to make use of higher order latent structure in the data between terms and documents that might be used to create better features for more accurate clustering. The goal is to use information to about term-document relationships in order to impute new term weights based on semantic similarity. So for instance even though "Julius Caesar conquered the native tribes that once inhabited modern day England" does not contain the words "Roman" or "Briton", a non-zero weight would be assigned to these terms for that sentence because the sentence has words that are semantically related to "Roman" and "Briton".

The way this is done is by performing truncated singular value decomposition on the transposed term-document matrix, $X \rightarrow T_k * S_k * D'_k$, where k is the rank of the approximate matrix $\hat{X} = T_k * S_k * D'_k$. k can also be interpreted as the dimension to which we are reducing the data. Figure 2 is an illustration of the decomposition:

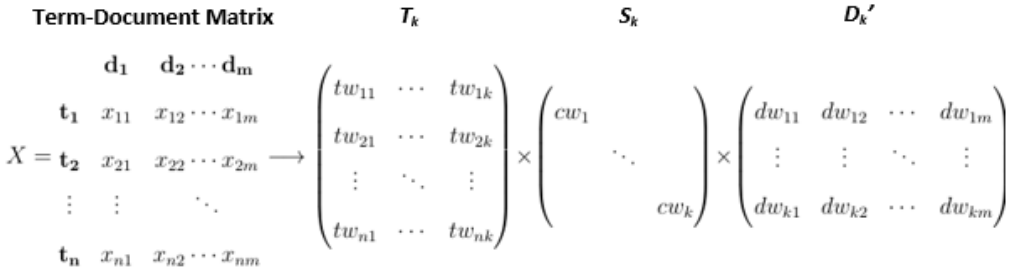

Figure 2: Truncated SVD of Term-Document Matrix

The orthogonal factors produced by this decomposition can be thought of as artificial concepts in the document set. The left singular vectors in matrix T represent the terms of the vocabulary in the abstract concept space, the right singular vectors in matrix D' represent the documents in the abstract concept space. The ijth entry of T, tw_{ij}, represents the level of association between the ith term of the vocabulary and the jth abstract concept with a term weight. Similarly the ijth entry of D', dw_{ij}, represents the level of association between the ith abstract concept and the jth document with a document weight. The diagonal singular values, cw_i, represent the relative

weights of the abstract concepts themselves. [10, 7] Dimension reduction is performed by selecting the top k components and then deleting the vectors of the three component matrices that don't correspond to these top components.

Once this decomposition and dimensionality reduction has been completed, we can treat the matrix SD' as the new reduced document matrix. [7]. We can then perform our analysis on this data matrix instead of the original one. This reduction is supposed to reduce noise brought on by synonymy and polysemy and can be particularly useful for enhancing clustering performance. [1] The main difficulty is in selecting an appropriate k number of components to keep. It has been shown that anything from 50-400 dimensions can be appropriate depending on the document set. [10]

3 Document Clustering

We used three clustering algorithms to cluster the documents: K-Means clustering, average-linkage agglomerative clustering, and fractionation partitional clustering. One of the difficulties of clustering is determining the right number of clusters into which the document set should be partitioned. We can use a number of internal clustering evaluation measures to determine the quality of a particular document clustering. We can use these metrics to determine which k leads to the best quality clustering configuration.

3.1 A Note on Distance Measures

All clustering algorithms rely on some kind of measure of distance between the vector representation of the documents. Such a distance measure can be denoted as $Dist(d_i, d_j)$. One popular measure of distance is Euclidean distance. For documents with vocab size v, the Euclidean distance is:

$$Dist(d_i, d_j)_{euclidean} = \sqrt{\sum_{k=1}^{v}(d_{ik} - d_{jk})^2} \tag{2}$$

Another popular measure of distance is the cosine distance measure which measures the cosine of the angle between the two document vectors in the feature space, and measure of similarity, and then subtracts this from 1. The measure ranges from 0 (identical vectors) to 1 (orthogonal vectors). Thus, a

higher cosine distance measure indicates greater distance between documents. The cosine distance for documents with vocabulary size v is calculated as:

$$Dist(d_i, d_j)_{cosine} = 1 - \frac{\sum_{k=1}^{v} d_{ik} * d_{jk}}{\sqrt{\sum_{k=1}^{v} d_{ik}^2}\sqrt{\sum_{k=1}^{v} d_{jk}^2}} \tag{3}$$

3.2 K-Means

The first clustering algorithm that we will examine is the k-means algorithm. [9]

The k-means algorithm is used to break up a set of objects into k clusters. The algorithm takes as input **K**, the number of clusters, and a set, X of **M** data objects, each with **N** features.

The algorithm begins by randomly selected **K** data objects, these **K** objects will form the initial cluster centers. The other documents are then associated with the cluster whose center is a minimum distance away, using one of the distance metrics described previously. Once this initial clustering is finished, the centroids, which are the mean vector of the cluster, are assigned to be the new cluster centers. For a cluster containing d document with **N** features, the centroid is calculated as $[\sum_{i=1}^{d} \frac{x_{i1}}{d}, \sum_{i=1}^{d} \frac{x_{i2}}{d}, ..., \sum_{i=1}^{d} \frac{x_{i\mathbf{N}}}{d}]$, where x_{ij} is the jth feature score of the ith document. The algorithm repeats for I iterations. The time complexity of the algorithm is $\Theta(\mathbf{NMKI})$ [9].

Figure 3 gives an illustration of how the k-means clustering algorithm works. The three large points represent cluster centers and the smaller points, representing data objects that are assigned to the closest center.

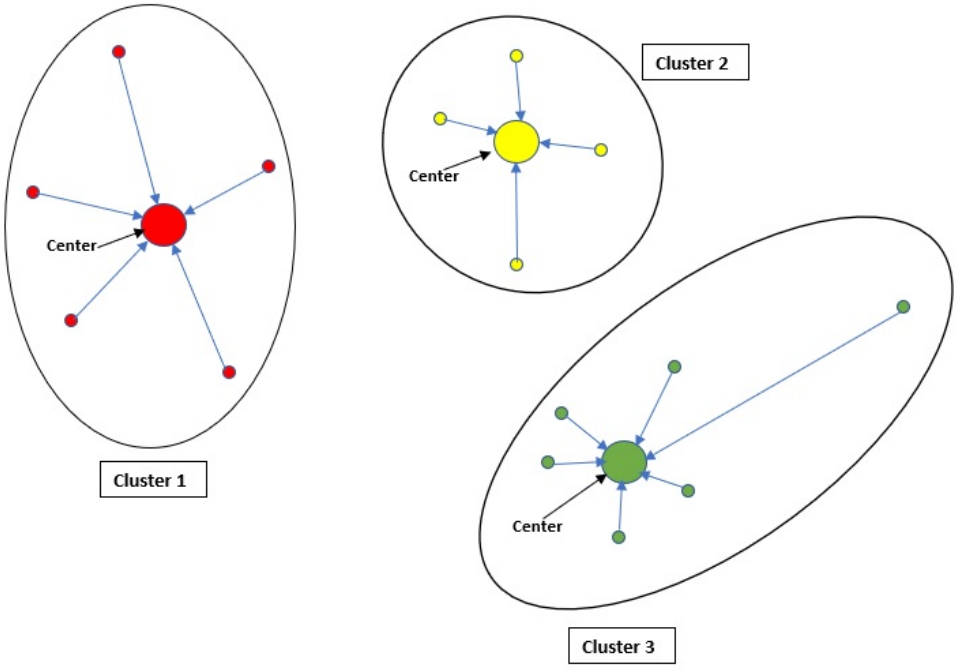

Figure 3: Illustration of K-Means Algorithm

3.3 Average-Link Agglomerative Clustering

The next algorithm we wish to discuss is the average-link agglomerative clustering algorithm. This algorithm comes from a family of agglomerative clustering algorithms.[14] The essential idea of an agglomerative algorithm is to start by treating all data objects as being in their own cluster and then combine the 2 closest clusters (which in this first case are individual objects), the algorithm proceeds by continuing to combine the two closest clusters until only the desired **K** clusters are left. The thing that distinguishes one agglomerative clustering algorithm from another is the way in which cluster "closeness" is defined, this cluster "closeness" is called linkage. There are three measures of cluster linkage: single-linkage, complete-linkage, and average-linkage. Single-linkage clustering assigns the distance between two clusters to be the distance between their two closest points. Complete-linkage clustering, on the other hand, assigns the distance between two clusters to be the distance between their two most distant points. Average-Linkage clustering attempts to use all the information from the two clusters and assigns the distance between the clusters to be the average of the distances between all points in both clusters.

Figure 4 illustrates the difference between the different agglomerative clustering algorithms. The illustration is taken from the [14].

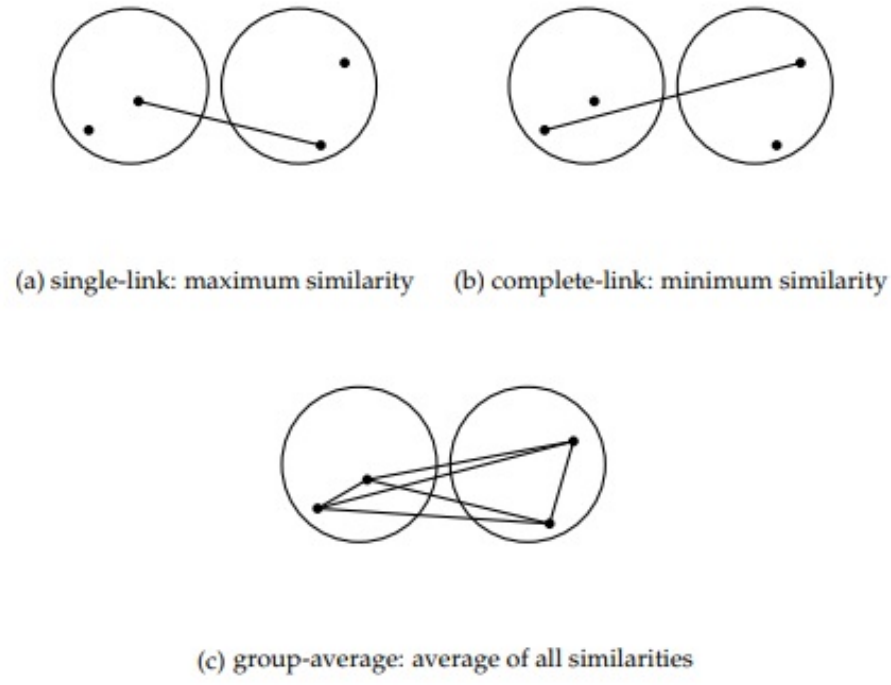

Figure 4: Agglomerative Clustering Algorithms

Linkage is based on similarity, which is really just an inverse concept from distance. An equation for computing the average-linkage between two clusters is as follows [14]:

$$AvgLink(\omega_i, \omega_j) = \frac{1}{(N_i + N_j)(N_i + N_j - 1)} \sum_{d_m \in \omega_i \cup \omega_j} \sum_{d_n \in \omega_i \cup \omega_j, d_n \neq d_m} Sim(d_n, d_m) \tag{4}$$

Where ω_i is the ith cluster, N_i is the size of the ith cluster, d_m is the mth document, and $Sim(d_m, d_n)$ is a measure of similarity between documents.

The average-link clustering algorithm follows the pattern of an agglomerative algorithm. The total cluster set is initiated as being the set of all documents. Clusters are then successively merged according to the average-link

criterion until the desired number of clusters is attained. The time complexity of the algorithm is $\Theta(\mathbf{M}^2 \log(\mathbf{M}))$. [14]

3.4 The Fractionation Algorithm

The final clustering algorithm that we used on the document set is the fractionation partitional algorithm described in [6]. The fractionation algorithm works very similarly to k-means except instead of starting with **K** random data objects to use as the initial cluster centers, it uses a more sophisticated deterministic method to select initial cluster centers and then proceeds with the k-means algorithm.

The algorithm starts by treating the whole document set X as the initial cluster centers. Then the cluster centers (the documents themselves in the first iteration) are sorted according to the feature score of a single feature that has a relatively high feature score across all documents. The set of sorted cluster centers is partitioned into m "buckets" and the average link agglomerative algorithm is used to reduce the number of objects in each bucket to $p * m$ where $0 < p < 1$ is some reduction factor. The centroids of these clusters are calculated and the centroids from all buckets are set to be the new cluster centers. The algorithm repeats until only **K** initial cluster centers are left, and then these initial cluster centers are used in k-means. The time complexity for finding the clusters is $O(m\mathbf{M})$, where once again **M** is the number of total documents in the document set. [6] Figure 5 below shows how the algorithm works:

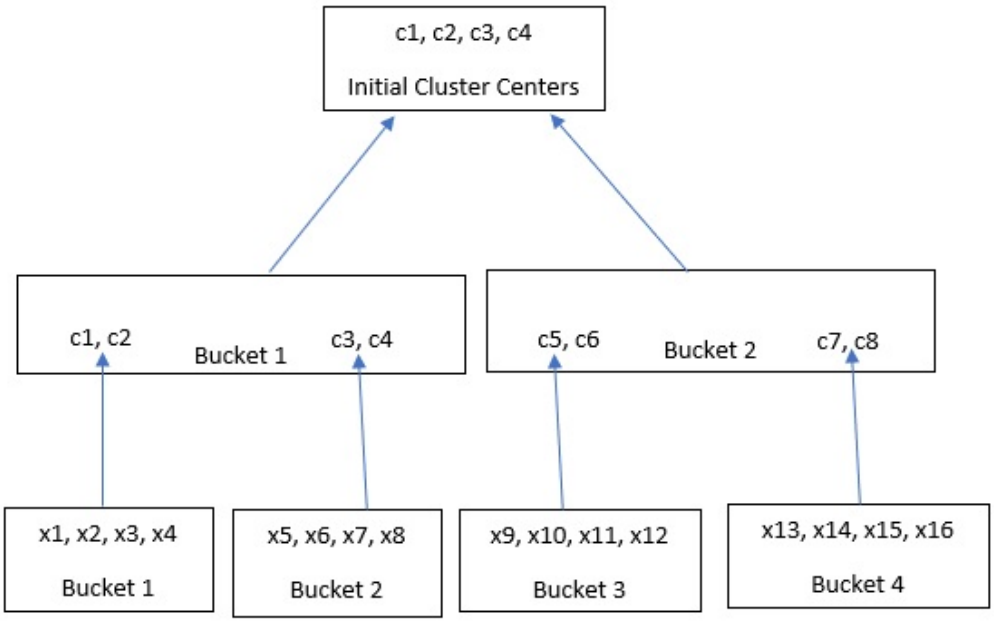

Figure 5: The Fractionation Algorithm

In figure 5, we have **M**=16, **K**=4, $m = 4$, and $p = 0.5$. The algorithm groups documents into buckets of size 4, and then uses an agglomerative algorithm to reduce the number of objects by half until the desired number of cluster centers is achieved.

3.5 Clustering Evaluation Measures

In order to evaluate the quality of different clustering configurations across different clustering algorithms and across different choices for **K**, we use three measures of clustering clustering quality: the silhouette index, the Calinski-Harabasz index, and the SD validity index. For both the silhouette and the Calinski-Harabasz indices, the goal is to maximize the index. For the SD validity index, the goal is to minimize the index. [13]

All three of these measures are internal clustering validation measures which means they use only the data and the clustering structure to evaluate the clustering quality. The measure doesn't have any external standard to which it compares the clustering to determine quality. This kind of metric is helpful when approaching a document set that we know nothing about and

for which we do not have a "correct" clustering to which we compare the automatic clustering methods.

A good internal clustering measure considers characteristics of the clustering configuration, separation and compactness. [13] Separation refers to how well separated, or how distinct the clusters are. A good clustering of the data will lead to well-separated clusters. Compactness refers to how tight the clusters are, or how much variation exists within a cluster. A good clustering will ideally lead to clusters that are grouped tightly around the cluster centroids.

3.5.1 Silhouette Index

We first consider the silhouette index. The way this coefficient is calculated is detailed in [8]. The index is calculated by calculating a coefficient for each point in the data set and then averaging over these coefficients to get a single value for the whole data set. For each point we calculate a_i, a measure of variation within a cluster C_k and b_i, a measure of how close the point is to points in other clusters. The silhouette index for one point in the data set is:

$$s_i = \frac{a_i - b_i}{\max\{a_i, b_i\}} \tag{5}$$

Letting I be the cluster of the ith data point,x_i, and letting n_I be the size of I, and letting $dist$ be a distance function for document pairs, the a_i coefficients are calculated as follows:

$$a_i = \frac{1}{n_I - 1} \sum_{x_i' \in I} dist(x_i, x_{i'}) \tag{6}$$

Thus, a_i is the mean distance between point x_i and all the other points in x_i's cluster.

To calculate the b_i coefficient, we first define the distance between x_i and the other clusters, C_k in the data set:

$$d(C_k, x_i) = \frac{1}{n_{C_k}} \sum_{x_i' \in C_k} dist(x_i, x_i') \tag{7}$$

Thus, $d(C_k, x_i)$ is the mean distance between x_i and every point in a different cluster C_k. The b_i is simply the minimum of the $d(C_k, x_i)$:

$$b_i = \min_k(d(C_k, x_i)) \tag{8}$$

Letting $\mathbf{M}$ be the total number of documents in the whole document set, the final index value for the whole clustering configuration is the mean of the s_i's:

$$s = \frac{1}{\mathbf{M}} \sum_{i=1}^{\mathbf{M}} s_i \tag{9}$$

3.5.2 Calinski-Harabasz Index

The next measure that we use is the Calinski-Harabasz index the details for this calculation are also in [8]. The index is calculated as:

$$CH = \frac{BGSS/(\mathbf{K}-1)}{WGSS/(\mathbf{M}-\mathbf{K})} \tag{10}$$

In this equation, BGSS is the between groups sums of squares. It is a measure of separation. It is calculated as follows:

$$BGSS = \sum_{k=1}^{\mathbf{K}} n_k ||Cent_k - Cent||^2 \tag{11}$$

In this equation n_k is the size of cluster k, $Cent_k$ is the centroid of the kth cluster and $Cent$ is the centroid of the entire data set.

WGSS is within group sums of squares. It is a measure of compactness. It is calculated as:

$$WGSS = \sum_{k=1}^{\mathbf{K}} \sum_{x_i \in C_k} ||x_i - Cent_k||^2 \tag{12}$$

In this equation, C_k is the kth cluster. We can see from the calculations that the index is a ratio of the separation and compactness of the clustering multiplied by a penalty term that penalizes clusterings with large $\mathbf{K}$.

3.5.3 SD Index

The final cluster evaluation index we made use of was the SD index:

$$SD = \alpha S + D \tag{13}$$

S is calculated as follows:

$$S = \frac{\frac{1}{\mathbf{K}} \sum_{k=1}^{\mathbf{K}} ||V_k||}{||V||} \tag{14}$$

Let $V = [\text{Var}(\mathbf{t}_1), \text{Var}(\mathbf{t}_2), ..., \text{Var}(\mathbf{t_N})]$, where $\text{Var}(\mathbf{t}_i)$ is the variance of the feature score for the ith term across all documents when there are $\mathbf{N}$ total terms in the vocabulary. Let V_k be defined analogously to V except that the variance of the feature score for each term is taken only across the documents in cluster k. This is a measure of the average variation with clusters and is thus a measure of compactness that should be minimized.

D is calculated as:

$$D = \frac{D_{max}}{D_{min}} \sum_{k=1}^{\mathbf{K}} \frac{1}{\sum_{k'=1, k' \neq k}^{\mathbf{K}} ||Cent_k - Cent_{k'}||} \tag{15}$$

$$D_{max} = \max_{k \in \{1,...,\mathbf{K}\}} \{||Cent_k - Cent_{k'}||\}, D_{min} = \min_{k \in \{1,...,\mathbf{K}\}} \{||Cent_k - Cent_{k'}||\} \tag{16}$$

$Cent_k$ is the centroid of the kth cluster. D is essentially a sum of the inverses of the distances between each cluster centroid and all other cluster centroids multiplied by the scaling factor $\frac{D_{max}}{D_{min}}$, thus D is an inverse measure of separation and should be minimized. The α coefficient is the value of D when we are considering the maximum value of $\mathbf{K}$ of interest to the analysis. This is meant to prevent D from carrying too much weight in the calculation of SD.

4 Summarization Algorithms

In this section we discuss three extractive summarization algorithms. An extractive summarization algorithm summarizes the text by picking out salient sentences and then pieces these sentences together to form a summary of the text. We discuss a latent Dirichlet allocation based summarization algorithm, a graph based algorithm, and a centroid based algorithm.

4.1 Latent Dirichlet Allocation Based Summarization

Latent Dirichlet Allocation (LDA) is generative probabilistic topic model proposed in [3]. The model is called generative because it views documents as

being generated by some latent topic structure. The basic idea behind LDA is that there are some topics that exist extrinsic to the documents, and the documents can be seen as being generated by some mixture of the topics. The topics themselves can be viewed as distributions over the words of a fixed vocabulary. Figure 6 illustrates the generation process.

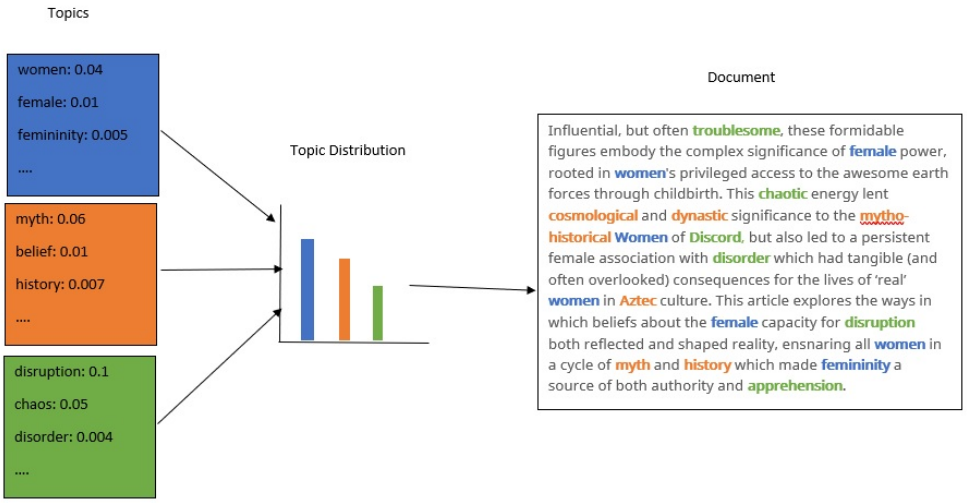

Figure 6: LDA Document Generation

If there are K topics and V words in the vocabulary, the process of document generation begins by drawing a K-long random vector from a Dirichlet distribution over the $K-1$ simplex parameterized by $\boldsymbol{\alpha}$, this will be the distribution of the topics for the document, we denote this K vector as θ. The topics denoted $\boldsymbol{\phi}$ can be seen as K random draws from a Dirichlet distribution over the $V-1$ simplex parameterized by $\boldsymbol{\beta}$. A topic assignment Z is then drawn from a multinomial distribution parameterized by θ, and finally a word W is generated by drawing a word from a multinomial distribution parameterized by the Zth topic.

The generation of a whole collection of documents can be summarized by the following joint distribution with N_j being the number of the words in the jth document, M being the number of documents, and K being the number of topics:

$$P(\mathbf{W}, \mathbf{Z}, \boldsymbol{\phi}, \boldsymbol{\theta}|\boldsymbol{\alpha}, \boldsymbol{\beta}) = \prod_{t=1}^{K} P(\phi_k|\boldsymbol{\beta}) \times \prod_{j=1}^{M} P(\theta_j|\boldsymbol{\alpha}) \times \prod_{i=1}^{N_j} P(z_{j,i}|\theta_j) P(w_{j,i}|\phi_{z_{j,i}}) \tag{17}$$

We are interested in finding the posterior distribution of $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ given the data $\mathbf{W}$, thus, we are interested in finding $P(\boldsymbol{\phi}, \boldsymbol{\theta}|\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{W})$ as this would give us the information needed to get estimates for the topics and the topic distributions for each document. This posterior distribution can be found using Gibbs sampling.

Once we have the topics and topic distributions in hand we can use these to extract summaries from the document collection using an the algorithm described in [2]. The algorithm works by calculating the probability of each sentence in the document collection given a topic for each topic. The conditional probability for a particular sentence S_r from a particular document D_r is calculated as follows:

$$P(S_r|\phi_k) = \frac{P(\phi_k|D_r)P(D_r)}{\sum_{j=1}^{M} P(\phi_k|D_j) * P(D_j)} \times \frac{\sum_{w_i \in S_r} P(w_i|\phi_k)}{length(S_r)} \tag{18}$$

All the $P(\phi|D)$ and $P(w|\phi)$ terms are available to us after performing Gibbs sampling with the LDA model. We assume that the distribution over documents is uniform so $P(D_r) = \frac{1}{M}$ for all i.

Once we have computed all the conditional probabilities of sentences given topics, we use a multinomial distribution to sample a topic from all possible topics. We then select the sentence with the highest probability given the topic and incorporate it into the summary. We repeat this process until the summary is of the desired length. If a sentence is selected more than once, it is not included in the summary more than once, the next most probable sentence is selected instead.

In order to select an appropriate number of topics K, a variety of LDA models are fit with different values of K and the K associated with the model with the greatest log-likelihood is chosen as appropriate K.

4.2 Graph-Based Summarization

Graph-based summarization algorithms use graph ranking algorithms like HITS are PageRank to extract important sentences from a set of documents. [16] For this project, we used the weighted PageRank algorithm [4] as the graph ranking algorithm for extracting the sentences.

The PageRank algorithm works by considering each object to be ranked, such as a webpage or in the case of document summarization, sentences, as a node in the graph. Connections between objects are established and are represented as edges between nodes. If the edges are directed, nodes that proceed a particular node, V_j, are called "in" nodes and are denoted $In(V_j)$, and nodes that succeed V_j are called "out" nodes and are denoted $Out(V_j)$. The algorithm works by calculating a score for each node and ranking the nodes based on this score. The score for each node, using weighted nodes, is calculated as follows:

$$PR(V_i) = (1-d) + d * \sum_{V_j \in In(V_i)} w_{ij} \frac{PR(V_j)}{\sum_{V_k \in Out(V_j)} w_{kj}} \tag{19}$$

The d is a damping factor between 0 and 1. Notice that this score is calculated recursively. The algorithm starts with initial scores for each node and then goes through rounds of recursive calculation until the scores converge within a particular threshold.

In order to use PageRank to perform text summarization, the sentences of the text are treated as nodes in a graph and connections are established between sentences that have words in common. The algorithm is used to rank the sentences in order of importance and then the top sentences are extracted until a desired summary length is reached. For this algorithm similarity between sentences is calculated as the overlap between two sentences normalized by the lengths of the sentences, these overlap scores are treated as the weights used in the weighted PageRank algorithm. Thus, the weight for the edge between two connected sentences, s_i and s_j, is calculated as follows:

$$w_{ij} = \frac{2 * (len(s_i \cap s_j))}{len(s_i) + len(s_j)} \tag{20}$$

Here $len(s_i)$ is the legnth of s_i and $len(s_i \cap s_j)$ is the number of common words for sentences s_i and s_j. Just as PageRank identifies the most relevant webpages for a particular query, the algorithm identifies the sentences that are

most relevant for a set of texts. For multi-document summarization, the algorithm is first run on each document individually and the extracted sentences are combined into a single meta-summary and then the algorithm is run again on the meta-summary. Experimental results have demonstrated that for the PageRank algorithm with multi-document summarization an undirected graph structure performs better than a directed graph. [16] Thus, for this project we use an undirected version of the algorithm. The undirected algorithm calculates scores in the same way except that the in-degree and out-degree of each node is the same. [15] Each undirected edge is replaced by two directed edges going in opposite directions.

4.3 Centroid-Based Summarization

The final summarization method uses the centroid of each cluster, as well as some other simple heuristics, to extract the most relevant sentences. [19]
It is particularly suited to summarizing documents in a cluster.

The algorithm works by calculating an initial score for each sentence in the set of documents in a cluster. The score is a weighted sum of the centroid score of the sentence, the positional score of the sentence, and the first sentence overlap score of the sentence. The centroid score of sentence i is the sum of the centroid values for the words in sentence i. The centroid values for the words are taken from the cluster centroid described previously as the mean vector of all document vectors in the cluster. Thus, letting $C_{w_{ij}}$ be the centroid value for the jth word of the ith sentence, the centroid score for sentence i, C_i is:

$$C_i = \sum_{w_{ij} \in s_i} C_{w_{ij}} \tag{21}$$

The positional value score for the sentence gives higher scores to sentences that are closer to the beginning of the document they are coming from and lower scores to sentences that are further from the beginning. The idea behind this score is that sentences near the beginning of a document will contain more essential summary information than sentences closer to the end and thus should be considered more important for summary purposes. Letting n be the number of sentences in the document containing sentence i and letting C_{max} be the maximum centroid score across sentences in the document containing sentence i, the position score for sentence i is calculated as follows:

$$P_i = \frac{n - i + 1}{n} * C_{max} \tag{22}$$

The final score used to create an initial importance score for each sentence is the first sentence overlap score. The reasoning behind using this score is similar to the reasoning behind using the positional score. The idea is that the first sentence of a document especially when using documents such as news articles or bulletins, the first sentence often gives a fairly good summary of the overall document. Thus, sentences that are similar to the first sentence, that is, sentences that have a lot of overlap with the first sentence should be given a higher score. Letting $len(s_i)$ be the length of s_i and $len(s_i \cap s_j)$ be the number of common words for sentences s_i and s_j, the first sentence overlap score for sentence i is:

$$F_i = \frac{2 * (len(s_i \cap s_j))}{len(s_i) + len(s_j)} * C_{max} \tag{23}$$

The overall sentence importance score is calculated as a weighted average of the three scores described above. Thus, we have the importance score for sentence i:

$$SCORE(s_i) = w_C * C_i + w_P * P_i + w_F * F_i \tag{24}$$

In order to help prevent information from being repeated in the summary, we also introduce a redundancy score for each sentence. After the sentences have been sorted by their initial importance score, we extract the sentences to create a summary. We then calculate the redundancy score for each sentence, subtract it from the original importance score, and then resort. This process is repeated until the sentences making up the summary don't change. We let SUM be the set of sentences in the summary at a given time and we let S be the set of all sentences in the cluster being summarized. The redundancy score for sentence i is:

$$R_i = \max_{s_j \in SUM} \left(\frac{2 * (len(s_i \cap s_j))}{len(s_i) + len(s_j)} \right) * \max_{s_k \in S} (SCORE(s_k)) \tag{25}$$

Once the sentences selected for the summary stabilizes with respect to the redundancy metric, we take the sentences as our final summary.

5 Comparison with Human Analysis

In this section we wish to compare the performance of the clustering and summarization algorithms described with clustering and summarization performed by humans. We compare the human and automatic performance on two different subsets of the original text data. One subset is a set of 25 documents that

were intentionally picked so that each document was similar to some other documents and different from all the other documents in the subset. This was done to create a document set that actually has some reasonably well-defined clusters. The other data set is composed of 25 randomly drawn documents. This represents a very noisy data set that may not have a very clearly defined cluster structure. Throughout the rest of the report we will refer to the document set with intentionally picked documents as the well-defined data set, and we will refer to the document set with randomly selected documents as the random set. Figure 7 shows the 2-D projection of these two document sets as they were clustered by the human analysts. Colors and shapes represent the cluster a document was assigned to by a human analyst. Even at this very low dimension (the original dimension was 5078) the analyst seems to have found a more reasonable structure in the well-defined document set then in the random document set.

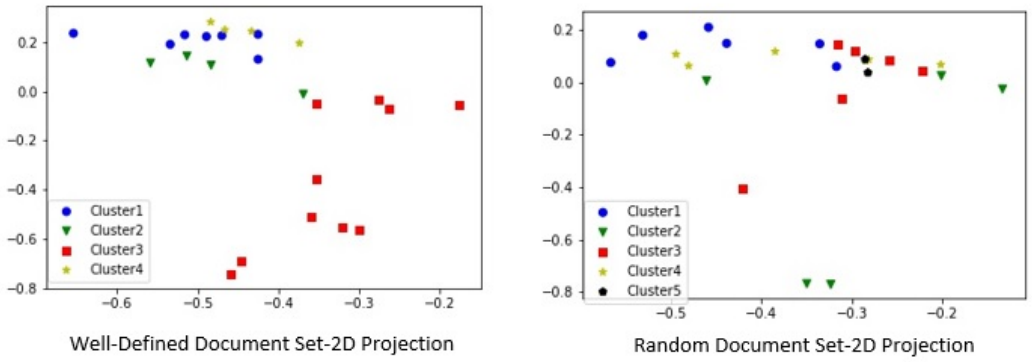

Figure 7: 2-D Projection of Test Document Sets

5.1 External Clustering Measure

To evaluate the quality of the clustering compared with the clustering done by human analysts we use an external clustering metric called the RAND index [14]. The metric is called external as opposed to internal because instead of using the data and the clustering structure to assess quality, it uses another clustering configuration as a reference by which to judge quality. The RAND index essentially measures the level of agreement between two clustering configurations. It is calculated as follows:

$$RAND(Clust_i, Clust_j) = \frac{TP + TN}{TP + TN + FP + FN} \tag{26}$$

$Clust_i$ refers to the ith clustering configuration. TP, TN, FP, and FN refer to the number of true positives, true negatives, false positives, and false negatives respectively. A true positive occurs when both clustering configurations agree that two documents belong in the same cluster; a true negative occurs when both clustering configurations agree that two documents don't belong in the same cluster. A false positive occurs when the two clustering configurations disagree about whether two documents belong in the same cluster, and similarly, a false negative occurs when the two clustering configurations disagree about whether or not two documents belong in the same cluster. Essentially the RAND index is just the fraction of all pairs of documents in the document set on which the two clustering configurations had agreement as to their cluster assignment.

5.2 Summarization Evaluation Metrics

We employ two evaluation metrics to assess the quality of the summaries produced by the computer as compared to the summaries produced by humans. The two metrics each measure a different component of summarization quality. These two components are recall and precision. Recall measures how much of the information in the human generated reference summary is present in the automatic summary, and precision measures how much of the information in the automatic summary is present in the reference summary. An automatic summary with good recall captures most of the information in the reference summary and an automatic summary with good precision captures only that information that is in the reference summary and not information that is not present in the reference summary.

A very simple metric for measuring recall is the ROGUE(Recall-Oriented Understudy for Gisting Evaluation)-1 metric [11]. It is calculated as follows:

$$ROGUE-1(AutoSum, HumSum) = \frac{len(AutoSum \cap HumSum)}{len(HumSum)} \quad (27)$$

Thus, the ROGUE-1 metric measures the proportion of overlapping words between the automatic and reference summary and the total number of words in the reference summary.

A simple metric for measuring precision is found in [18]. It is very similar to the ROGUE-1 metric except we divide by the number of words in the automatic summary:

$$Precision(AutoSum, HumSum) = \frac{len(AutoSum \cap Hum_Sum)}{len(AutoSum)} \tag{28}$$

5.3 Comparison of Automatic Clustering with Human Clustering

For each of the two document sets, well-defined and random, we will use internal cluster validation to find the optimal number of clusters and to determine if LSI improves the performance, we then compare the performance of the three algorithms to the human clustering using the RAND index.

5.3.1 Well-Defined Document Set

The initial clustering results for the three algorithms with the well-defined data set are presented in figure 8:

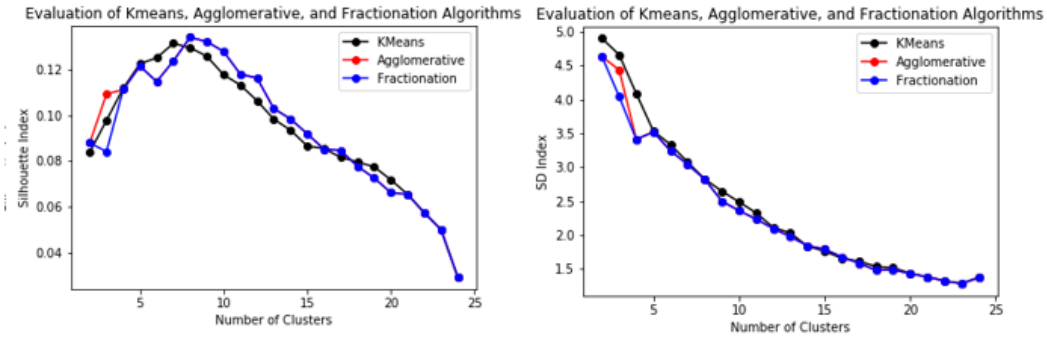

Figure 8: Initial Clustering Results for Well Defined Document Set

The figure on the right shows the silhouette of each clustering algorithm for different values of k, the left plot is similar, but it uses the SD score. We would have used the Calinski-Harabasz index since this is already implemented in Python. This index, however,puts a heavy penalty on clustering configurations with more clusters and is sometimes not ideal for determining the ideal k. It is common to prefer a smaller number of clusters, so we chose the number of clusters where the peak of the silhouette metric begins. The SD metric would seem to suggest going with the maximum number of clusters possible, but again we would prefer to go with fewer clusters. A good number of clusters seems to be 6 according to the silhouette metric. We also perform LSI analysis. We look at projections between 2 and 25. The results are in figure 9:

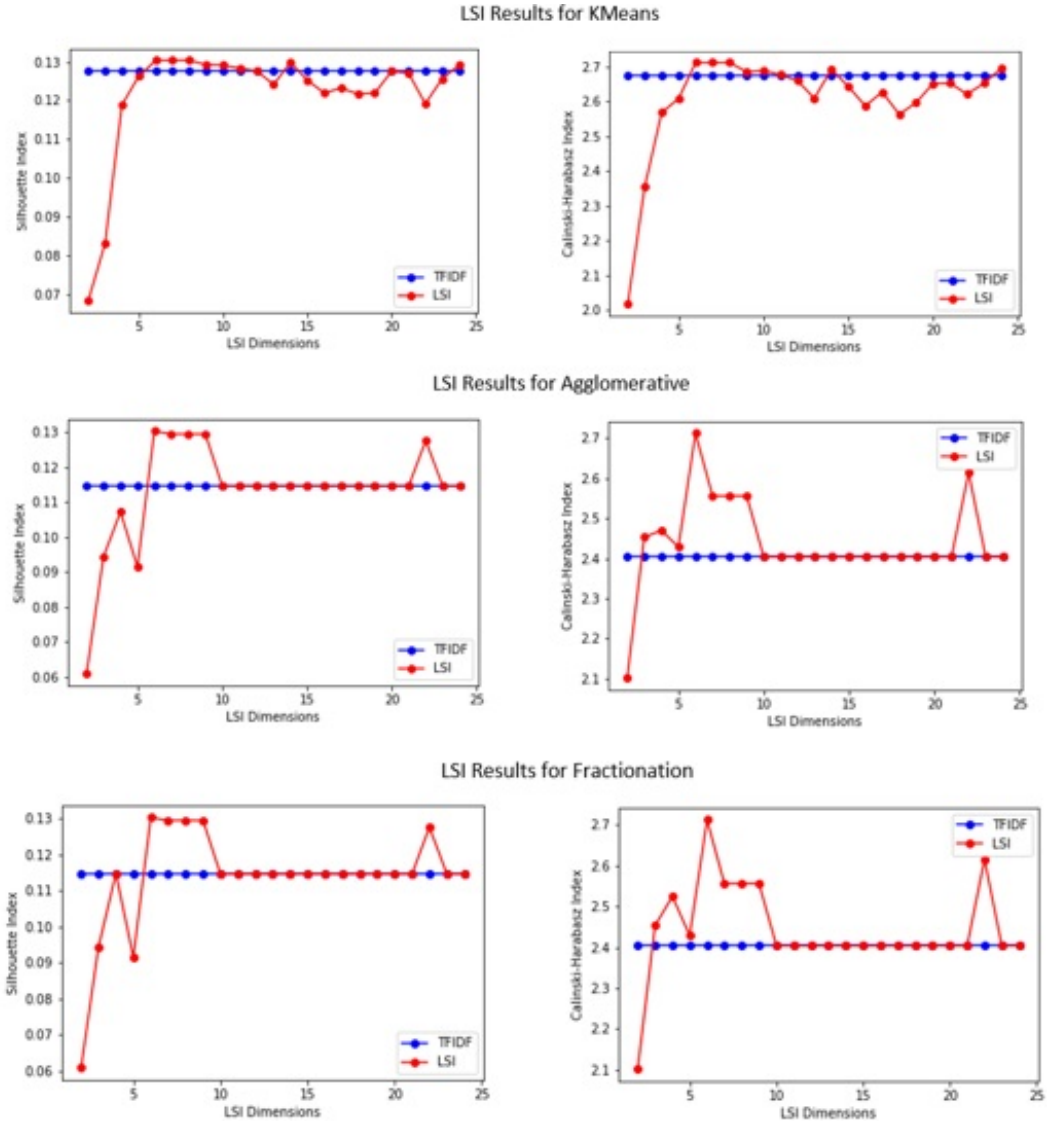

Figure 9: LSI Results for Well Defined Document Set

The plots on the right give the silhouette coefficients; the plots on the right give the Calinski-Harabasz coefficients. All of the plots for each of the algorithms tend to suggest that a projection of dimension 6 would be best as this is the point where the silhouette and Calinski-Harabasz indices seem to be maximized.

After this projection, we run the clustering algorithms again to see if they recommend a different number of clusters. These results are presented in figure

Algorithm	RAND Score
KMeans	0.806666
Agglomerative	0.806666
Fractionation	0.806666
Random	0.64313

Table 1: RAND Scores for Clustering with Well-Defined Document Set

10:

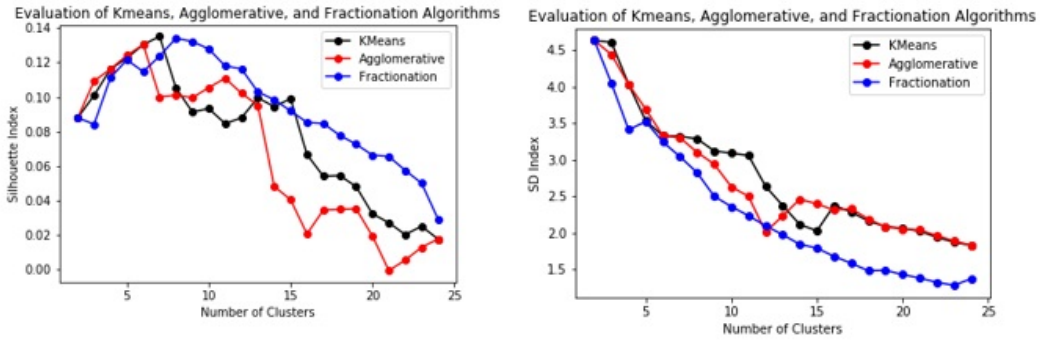

Figure 10: Final Clustering Results for Well Defined Document Set

Again a clustering number of 6 seems to be a good choice according to the silhouette metric.

We can now compare these results against the human clustering results using the RAND index described previously. The human clustering found 4 clusters for this data as opposed to the 6 clusters chosen for the automatic algorithm. We also performed random clustering in which the documents were assigned to 6 clusters randomly. This was used as a benchmark against which to compare the other automatic algorithms. The RAND agreement scores are presented in the table 1 below:

As can be seen from the table, the automatic clustering algorithms perform quite a bit better than the random algorithms for the well-defined data set. I ran the random algorithm 100 times and then created a histogram of the RAND scores for each of these runs and compared these scores with the best automatic clustering score. Figure 11 displays the histogram with the vertical line representing the best automatic clustering score. As can be seen from the figure, the automatic clustering score is significantly better than all the

random clustering scores.

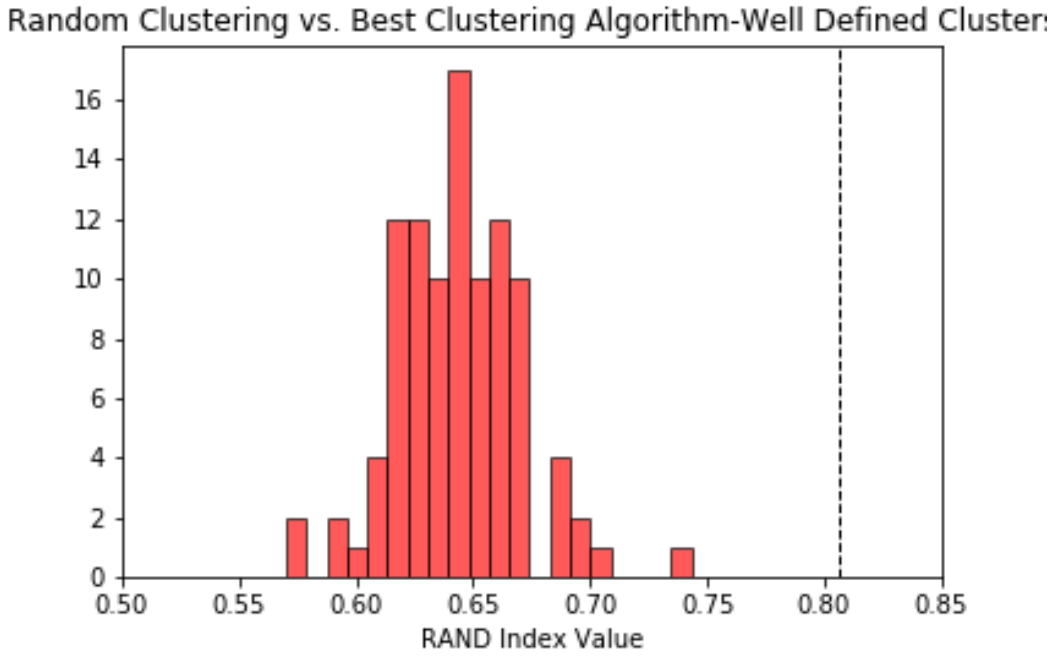

Figure 11: Histogram of Random Clustering Results for Well-Defined Document Set

5.3.2 Random Document Set

The initial clustering results for the random document set are displayed in figure 12:

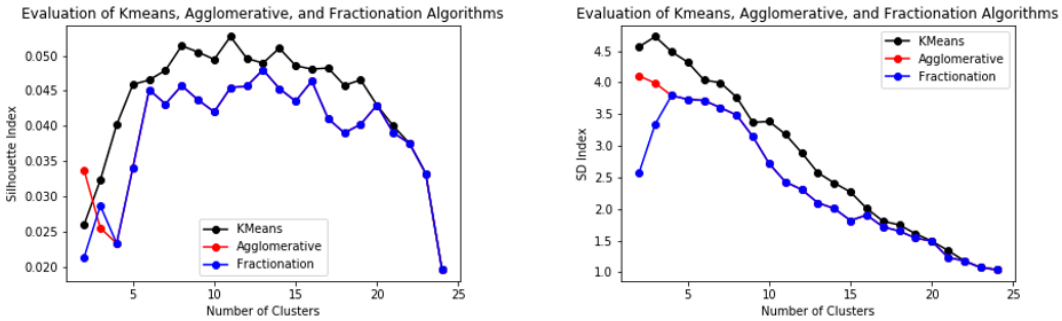

Figure 12: Initial Clustering Results for Random Document Set

Again there is disagreement between the silhouette and SD indices. We will once again, go with the metric that recommends fewer clusters. We determine that 8 clusters seems appropriate by the silhouette metric.

We again perform LSI analysis to determine if any dimension reduction would be useful. The results are presented in figure 13:

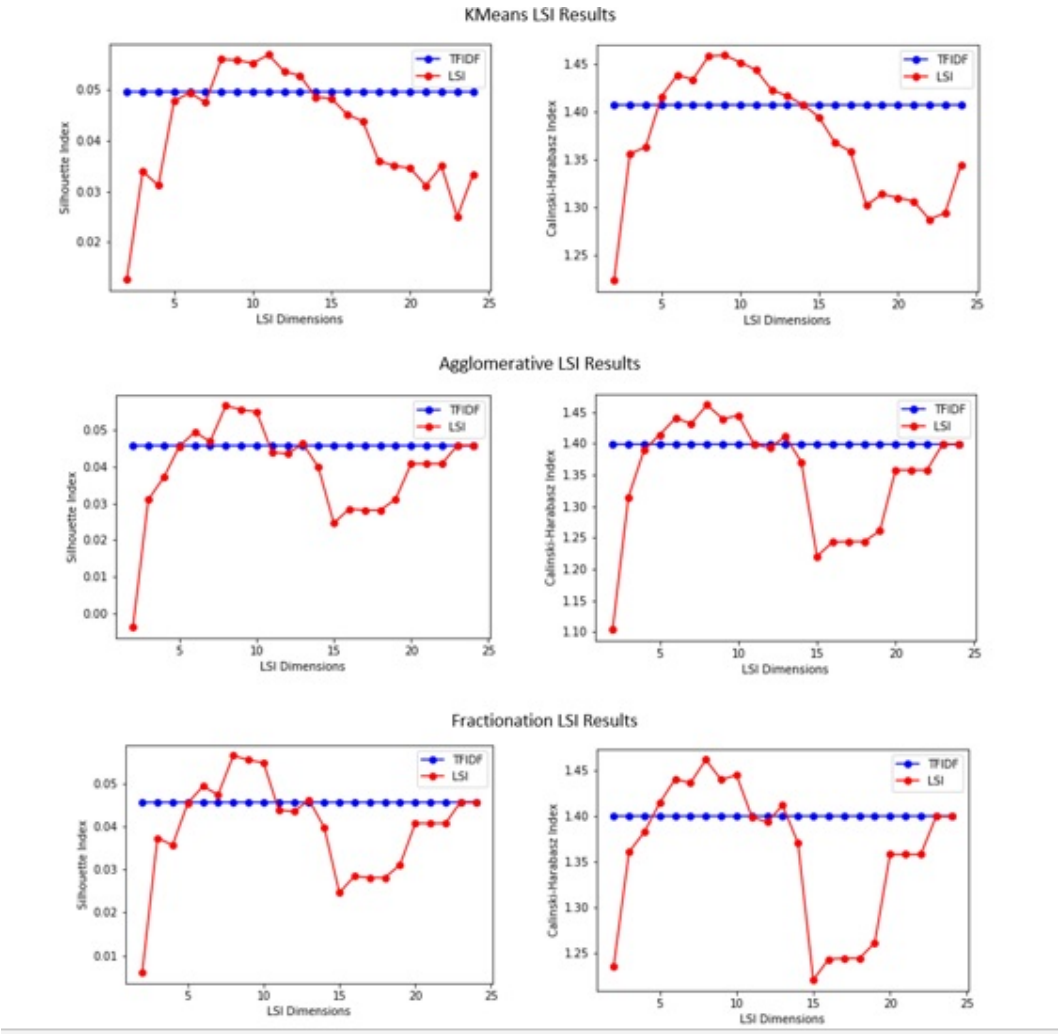

Figure 13: LSI Results for Random Document Set

All the plots suggest that an LSI dimension of 8 would lead to some improvement. Using 8 dimensional data, we redo the clustering analysis with the automatic algorithms to see if a new optimal number of clusters is suggested.

The results are shown in figure 14:

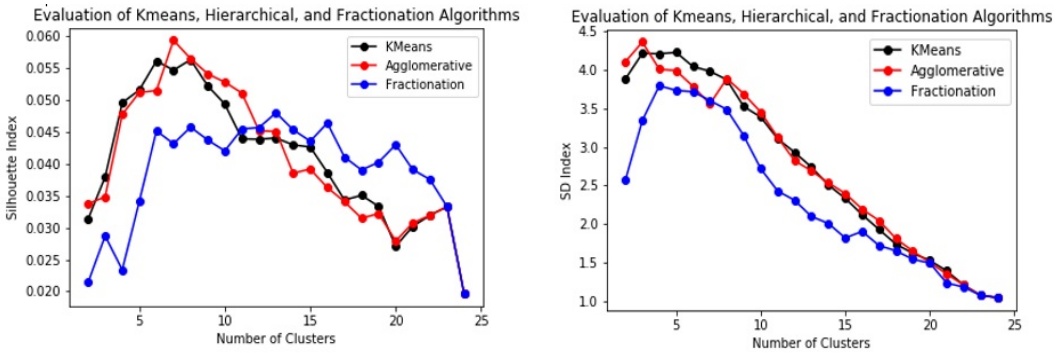

Figure 14: Final Clustering Results for Random Document Set

The silhouette index again suggests the smaller number of clusters, and so by the silhouette index, we determine the best number of clusters to be 7.

As with the well-defined document set, we compare the three automatic algorithms and a random benchmark algorithm with the RAND index. The results are in the table 2 below:

Algorithm	RAND Score
KMeans	0.746666
Agglomerative	0.72
Fractionation	0.72
Random	0.7143

Table 2: RAND Scores for Clustering with Random Document Set

The results show that the automatic clustering algorithms did quite a bit worse when no strong clustering structure is actually present in the data. This is expected. We also see that for the random document set, the automatic algorithms don't do much better than the random benchmark. We produced a histogram of the scores for 100 random clusterings to compare with the best automatic score. The histogram is in figure 15:

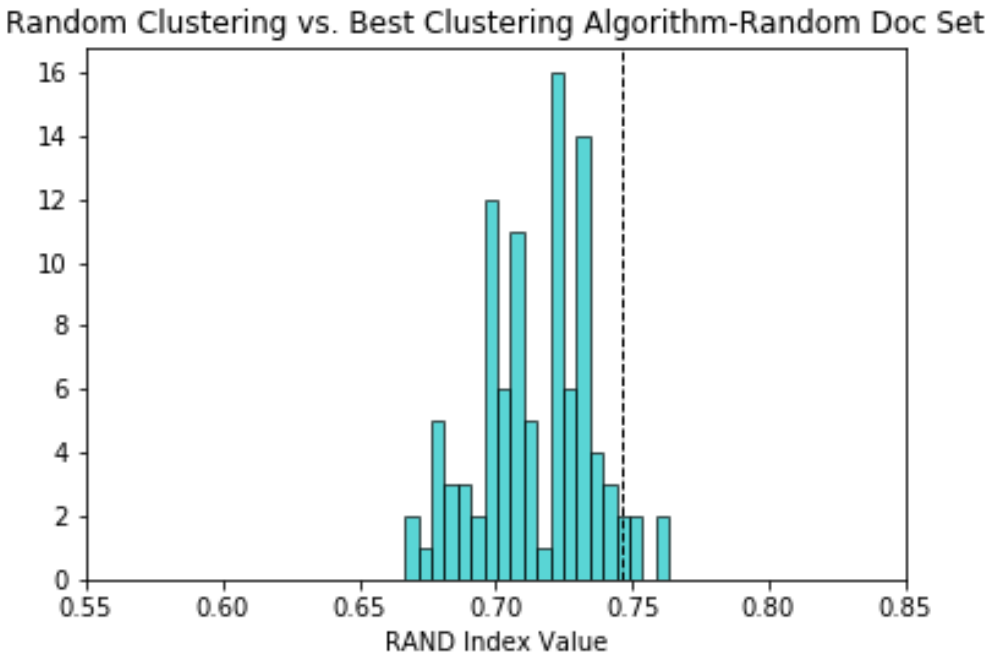

Figure 15: Histogram of Random Clustering Results for Random Document Set

As can be seen from the figure, the automatic clustering algorithms don't do much better than the random clustering algorithms and very occasionally did a bit worse than the random clustering algorithm when no clear clustering structure is present in the data.

5.4 Comparison of Automatic Summarization with Human Summarization

For the well-defined and the the random document set, we had the human analysts construct two summaries. One summary is abstractive, meaning the analyst summarized the contents of each cluster in their own words, and one summary is extractive, meaning the analyst extracted the sentences from the documents in the cluster that he or she felt best captured the contents of the cluster. The extractive summaries are constructed in a similar way to how the automatic summary algorithms construct summaries. For each cluster in each document set, we calculated the ROGUE-1 and precision scores between both the extractive and abstractive human summaries and the automatic summary. We then took the average of the precision and ROGUE-1 scores to create a final

overall score for how the automatic summary compared to each human created summary. A higher overall score indicates an algorithm whose automatic output was closer to the corresponding human summary. As a benchmark for the summarization algorithms' performance, we created a random summarization algorithm that works by randomly selecting sentences from the cluster of documents until the desired summary length is reached. The ROGUE-1 and precision scores for this algorithm and the human summaries is also calculated for comparison.

5.4.1 Well-Defined Document Set

Figure 16 displays two plots summarizing how the three summarization algorithms and the random algorithm compared with the human extractive and abstractive summaries.

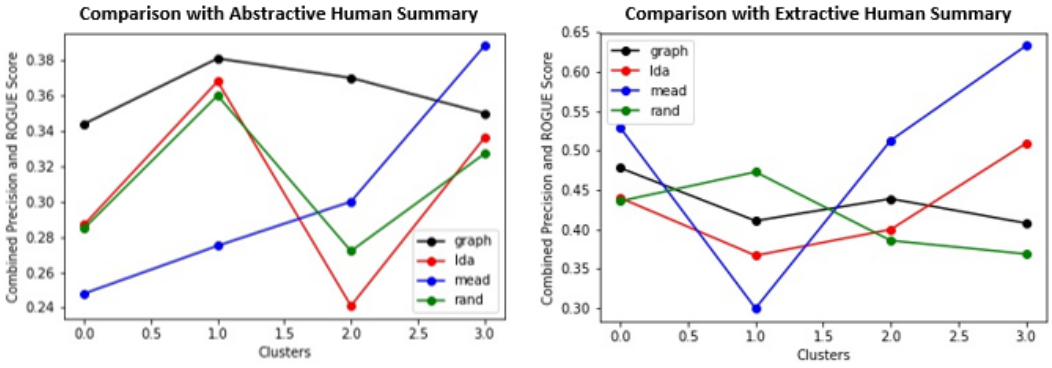

Figure 16: Performance of Automatic Summarization Algorithms for the Well-Defined Document Set

Each point represents the overal score for the summary of one of the 4 clusters with a particular summarization algorithm, different algorithm are represented with different color. The figures show that the random summarziation algorithm produces summaries with greater similarity to the human summary than each automatic summarization algorithm several times. We also note that the scores for all summarization algorithms are higher in when comparing with the extractive human algorithm. This doesn't necessarily mean that the algorithms actually do a lot better when compared with an extractive human summary because the random algorithm also does much better. This jump in scores is due mostly to the fact that the vocabulary of the extractive human summary will be more similar to the extractive automatic algorithms

than the vocabulary of the abstractive human algorithms that don't necessarily use the same terminology and style as the documents being summarized.

In order to get a better idea of which algorithms performed best and whether the algorithms actually do better in the extractive case, for each automatic algorithm, we calculate the mean difference between the random summarization score and the automatic summarization score for each cluster. To get an indication of the performance of the algorithms when compared with the extractive v.s. the abstractive human summary, we look at the mean of the mean difference over the three different algorithms. These results are shown in the table 3 below:

Algorithm	Mean Difference
LDA	-0.003
Graph	0.0502
Centroid	-0.0082
Average	0.0130

Algorithm	Mean Difference
LDA	0.0130
Graph	0.0180
Centroid	0.0778
Average	0.03625

Table 3: Summarized performance with Abstractive Human Summary (Left) and Extractive Human Summary (Right) on Well-Defined Documents.

As can be seen from the above table, the automatic algorithms when compared with the extractive human summary (right table) have a greater average lead than when compared with the abstractive human summary (left table). The exception to this trend is the graph algorithm which has a had a greater mean difference with the random algorithm when compared with the abstractive human summary. The graph and centroid algorithms both do much better than the LDA algorithm on average.

5.4.2 Random Document Set

We now perform a similar analysis for the random document set. Figure 17 displays the results for how the output of the algorithms compare with the human extractive and abstractive algorithms in the same style as above.

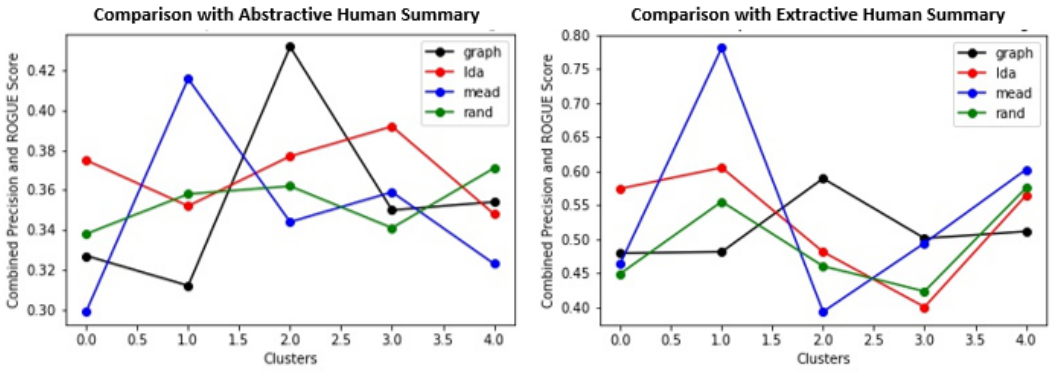

Figure 17: Performance of Automatic Summarization Algorithms for the Random Document Set

The results are similar to the well-defined document set. The random algorithm still beats each of the other algorithms at different points. We present two tables similar to the ones above to summarize information from the plots:

Algorithm	Mean Difference
LDA	0.0148
Graph	0.0010
Centroid	-0.0058
Average	0.0033

Algorithm	Mean Difference
LDA	0.0323
Graph	0.01980
Centroid	0.0544
Average	0.0355

Table 4: Summarized performance with Abstractive Human Summary (Left) and Extractive Human Summary (Right) on Random Documents.

Again, the summarization algorithms have a greater lead over the random summarzier when comparing to the extractive human summary. This is most likely because in this case, the human was constrained to choosing sentences only from the document set. There may even be several summaries that overlap completely between the automatic and human extractive summaries. This constraint is not present for the abstractive summaries. The information about which algorithm is better is not as conclusive from this information, for instance the centroid algorithm performs much better than the random algorithm compared to the extractive summary, but a bit worse than the random algorithm when compared with the abstractive algorithm. Unlike in the previous analysis, the LDA algorithm seems to do well when compared to both summaries.

6 Extending Clustering and Summarization to Large Document Sets

Sometimes for large document sets, a single clustering of the whole document set may not be necessarily useful. For instance, figure 18 below shows the behavior of the silhouette and SD metrics when trying to find an appropriate number of clusters for the entire 475 document set with KMeans.

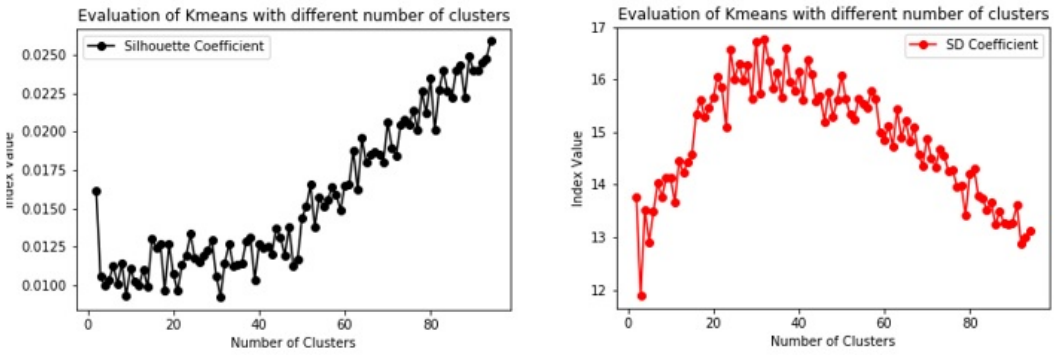

Figure 18: Number of suggested Clusters with K-Means Algorithm

The silhouette coefficient is highest at 94 clusters, it could go even higher if we looked at other clustering configurations. The SD coefficient is lowest at 2 clusters or around 92 clusters. None of these clustering configurations would be particularly useful for the analyst. Having to sort through 2 very large clusters or 95 smaller clusters is still a big job. We want the analyst to be able to find a manageable subset of the document set that is interesting to them. The very large and very small k would also seem to suggest that clustering may not make sense on such a document set. There may not be a well-defined clustering structure, which as we have seen can lead to meaningless results when clustering. Ideally, we would like to combine the clustering and summarization techniques to help an analyst zero in on a small set of documents that is of particular interest to him or her. One way of combining these operations was suggested by Cutting et. al [6]. The technique is called scatter-gather.

To begin, an analyst scatters the document set into k clusters, some kind of summary, known as the cluster digest, is produced for each cluster. For large clusters, it may be difficult to capture enough of the key concepts with one of the extractive summarization algorithms detailed above. A better approach may be to use a keyword extraction method until the cluster size has less than

20 documents. Keyword extraction can capture more of the main keywords of a cluster with less redundancy because the extraction unit is not restricted to be complete sentences. Once the cluster sizes are smaller, more detailed extractive summaries can be more useful and can capture a good amount of the information while remaining reasonably small. Once an analyst has constructed some set of cluster digests, they can use these digests to determine which clusters may contain useful information and then gather these clusters into their own separate subset. The analyst can then repeat this procedure on the gathered subset. In this way, the analyst can drill down to a subset of documents that is of particular interest to them. The idea of how the algorithm works is illustrated in the figure below taken from [6]:

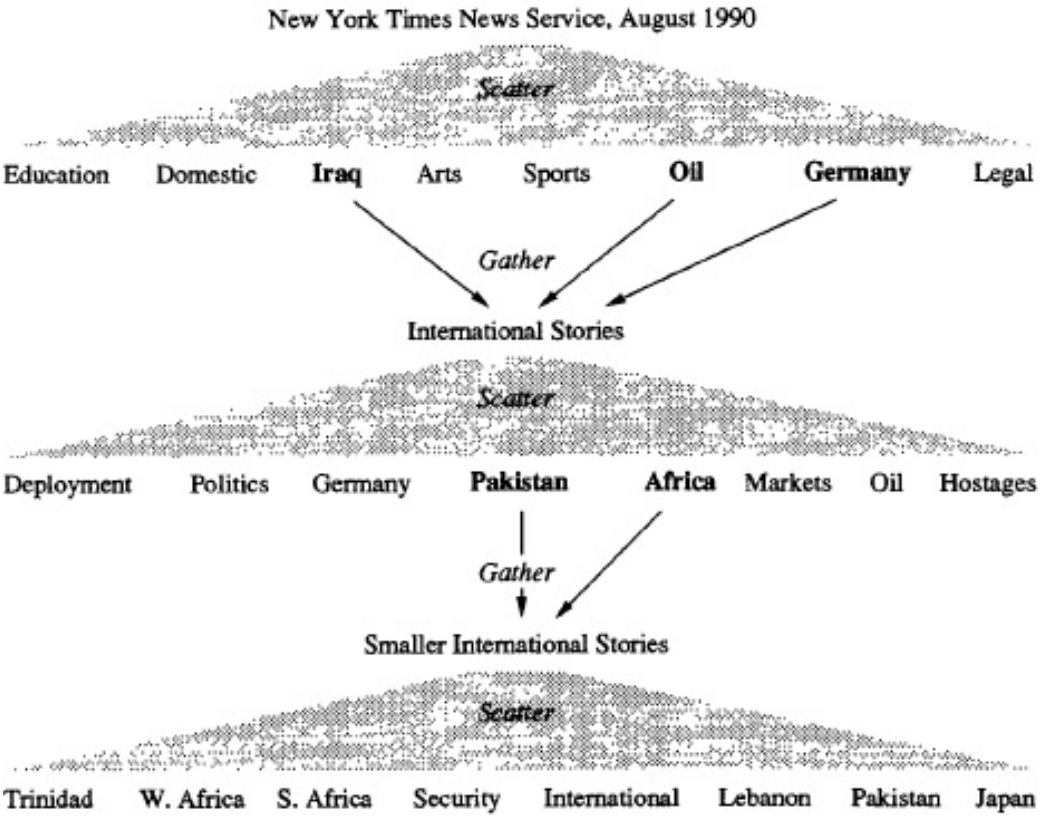

Figure 19: Illustration of Scatter-Gather

The analyst starts with a large set of documents, like all New York Times articles from August of 1990, and then scatters and gathers the set to systematically reduce the document set to some manageable and useful subset.

The single words that describe potential topics of interest stand in as cluster digests. Such a method relies on both clustering and summarization together.

7 Conclusions and Future Work

The goal of this report was to explore algorithmic tools for easing the burden of an intelligence analyst who must make sense of large sets of text documents. We have explored various clustering and summarization algorithms that when used in a frame work like scatter-gather can make the processing of text data more efficient.

We also examined the performance of these methods when compared with human generated clustering configurations and summaries. We found that when a document set has some kind of actual clustering structure, the automatic clustering algorithms and human clustering configurations were quite comparable. When a document set doesn't have such a structure, the human and automatic clusterings aren't as similar.

When comparing the output of the extractive summarization algorithms to the extractive and abstractive human summaries, the automatic summarization algorithms did not perform very consistently and sometimes had less similarity with the human summaries than a random set of sentences extracted from the document. Ideally, the summarization algorithms would consistently produce results similar to the human summaries. Thus, in future studies it would be worthwhile to do further study of summarization algorithms.

Some research has been done on automatic abstractive summarization algorithms [12]. This would be an interesting field of study to pursue. It is also important to note that our algorithms performed inconsistently compared when compared with one human reference summary using two basic metrics. It would be worthwhile to explore how the algorithms perform when compared with multiple human summaries. It could be that while the summarization algorithm may not produce summaries that are similar to individual human summaries, when compared with several human summaries, the algorithms may perform well on average. It is also worth noting that the precision and recall metrics used are somewhat naive. They base similarity purely on overlapping word count. There may be more sophisticated ways of measuring similarity that may show the automatic summarization algorithms to be more similar to the human summaries than they appear to be under the simplistic

precision and recall measure used.

8 Acknowledgments

I would like to thank my mentor Dr. James Degnan for his support and guidance and for allowing me to speak to his class about my work. I would also like to thank the UNM Mathematics and Statistics department for sponsoring my research. I would like to thank Dr. Monika Nitsche in particular for organizing the SUnMaRC conference which gave me an excellent opportunity to present my research to some of my peers. Finally, I would like to thank Mr. Tucker Berry and Ms. Makenna Johnson for doing the hard work of acting as the human analysts who produced the human results for me to compare against the automatic algorithms.

References

[1] Charu C. Aggarwal and ChengXiang Zhai. *A Survey of Text Clustering Algorithms*, pages 77–128. Springer US, Boston, MA, 2012.

[2] Rachit Arora and Balaraman Ravindran. Latent dirichlet allocation based multi-document summarization. In *Proceedings of the second workshop on Analytics for noisy unstructured text data*, pages 91–97. ACM, 2008.

[3] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[4] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[5] Randal E Bryant, Jaime G Carbonell, and Tom Mitchell. From data to knowledge to action: Enabling advanced intelligence and decision-making for america's security. *Computing Community Consortium, Version*, 6, 2010.

[6] Douglass R Cutting, David R Karger, Jan O Pedersen, and John W Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *ACM SIGIR Forum*, volume 51, pages 148–159. ACM, 2017.

[7] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990.

[8] Bernard Desgraupes. Clustering indices. *University of Paris Ouest-Lab ModalX*, 1:34, 2013.

[9] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[10] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[11] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.

[12] Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman Sadeh, and Noah A Smith. Toward abstractive summarization using semantic representations. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1077–1086, 2015.

[13] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. Understanding of internal clustering validation measures. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 911–916. IEEE, 2010.

[14] C.D. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval International Student Edition*. Cambridge University Press, 2008.

[15] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004.

[16] Rada Mihalcea and Paul Tarau. A language independent algorithm for single and multiple document summarization. In *Companion Volume to the Proceedings of Conference including Posters/Demos and tutorial abstracts*, 2005.

[17] Peter Norvig. Natural language corpus data. *Beautiful Data*, pages 219–242, 2009.

[18] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[19] Dragomir R Radev, Hongyan Jing, Małgorzata Styś, and Daniel Tam. Centroid-based summarization of multiple documents. *Information Processing & Management*, 40(6):919–938, 2004.

[20] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[21] ChengXiang Zhai and Sean Massung. *Retrieval Models*, page 88–89. ACM Books, 2016.