Utrecht University

Mining the Evolution of Political Opinion

Master Thesis

First Supervisor

prof. dr. A.P.J.M. Siebes
Algorithmic Data Analysis Group
Department of Information and Computing Science

Second Supervisor

dr. A.J. Feelders
Algorithmic Data Analaysis Group
Department of Information and Computing Science

Submitted By

Everton Lima
Master Thesis
Department of Information and Computing Science

Abstract

This thesis examines political opinion through Reddit comments and pattern set mining. In 2016 the world saw an increasing effect of social media on public opinion. Large internet communities played a major role in disseminating propaganda, information, and misguiding voters. These issues are examined through the extraction and compression of patterns present in word usage and frequency. Moreover, experiments done on synthetic data show that compression is a useful tool in extracting and matching patterns from various datasets. Lastly, the same approach is then used to examine changes in opinion on Reddit's comment data.

Contents

1 Introduction

The 2016 United States Presidential Election brought the influence of social media on public opinion to the forefront of political discussion. The now President Donald Trump, made his use of Twitter notorious through his attacks on other candidates and staunch critic dismissals [Lee and Quealy, 2016]. Targeted advertisement also had a significant role, an important one being Trump's voter suppression operations [Green and Issenberg, 2016]. However, the most notable use of social media has been the crowdsourcing of Internet communities to the production of political propaganda with sites such as Instagram, Facebook, and Reddit playing an important role. Among these communities, the largest crowdsourcing effort has undoubtedly been made by Reddit's *The_Donald* members. The community self-organized to go through the leaked Democratic National Committee emails [reddit,] and produced pro-Trump (and anti-Hillary) image macros and posts, and disseminated them on the internet. Trump supporters collectively named these efforts *The Great Meme War*. In contrast with Trump's mostly volunteer based efforts, candidate Hillary Clinton employed paid staff such as bloggers, former reporters, and designers to resist Trump's online presence [Roller, 2016]. With the rising influence of social media on politics, it has become crucial to understand how voters are affected by political campaigns and internet movements. The novel ways social media is being used by politicians leads to the question: how can political opinion be measured and compared? This thesis addresses this question via pattern set mining through the extraction of Reddit comments and clustering via compression.

Consider now a dataset containing the opinion of a sample of American voters. Assume that such a dataset contains the political views of a large population of electors in the form of textual data. Expectedly the trends present in such a database will differ significantly between election cycles. But consider now the question, how much would voter opinion differ at the end of a election, when compared to its beginning? For example, initial trends may include a negative view of immigrants and increasing concern about health care. As the election approaches its end, voters may become concerned about completely separate issues. However, it is not necessarily the case that the data will completely differ at the end of the election cycle; while new patterns may appear, political parties, the voting population, and candidates are likely to remain relatively the same. Consequently one could expect that trends present will go through a process of change, but are not completely altered.

When considering such a dataset, how could patterns be extracted? One approach is to apply frequent pattern set mining and match co-occurring patterns among sources and across time. In doing so, we could identify how the number of matching patterns increases or diminishes between a pair of datasets. However, this approach is likely to yield undesired results and be computationally expensive. The local descriptions produced by frequent pattern mining are typically too large in quantity and often lack in quality. This issue is further compounded by the difficulty in choosing the minimum frequency threshold leading to an (exponential) explosion in patterns. While there are many approaches for solving the pattern explosion problem, one difficulty remains; due to its quantity and lack of quality, identifying co-occurring patterns quickly becomes a computational costly procedure.

A more direct approach to finding local patterns is through compression. Intuitively, the set of patterns or itemsets that achieves the highest compression is the smallest set of patterns that is well represented in the dataset. If multiple patterns describe the same data, they are likely to lead to a poor compression. One approach to identify patterns that compress is through the Krimp algorithm. Krimp post-processes the set of frequent patterns by selecting those that result in a high compression. The chosen set of patterns is named a code table. Code tables can then be used to compare different datasets by measuring dissimilarity through cross compression.

Social media has become of importance in politics, and consequently the study of its processes. This thesis explores social media data via the extraction of patterns and examines how they change. In doing so, the idea of induction via compression is used through the Krimp algorithm. Additionally, compression is used to compare trends among different datasets and across time. Thus, the evolution of patterns, in the form of political trends, is examined.

This thesis is organized as follows. Sections 2 and 3 provide a brief review of the methods used. Section 4 describes the code table dissimilarity measure. Section 5 further establishes the method through experiments on synthetic data. Section 6 describes the data extraction, preprocessing, and results of using code table dissimilarity in comments extracted from Reddit. Section 7 concludes this thesis by revisiting the methods used and the results produced. Additional figures are shown in Section 8.

2 Background

2.1 Mining Frequent Patterns

Pattern set mining makes it possible to identify anomalies, understand general trends, and obtain a representative set of patterns from transactional datasets that are otherwise too large for manual inspection. Consider a grocery store in which a large quantity of products are stocked. How can we determine which products are typically bought together? This question can be answered by identifying which subsets, of the entire product set, are frequently present in all transactions. A transactional database is defined as a collection of such transactions, where products are either present or absent. A well known formal description of the pattern set mining problem can be read in [Agrawal et al., 1996]. This description is paraphrased here in the context of mining frequent patterns. Given a transaction of databases D, a set of items I, and a minimum frequency threshold *minsup*, compute the set of patterns such that they occur more frequently than *minsup*. In frequent pattern mining the representative, or interesting set of patterns, is defined as the set of transactions that occurs above the *minsup*threshold. In this context, transactions or patterns are also described as itemsets (abbreviation for sets of items). Formally the task of mining frequent itemsets can be described as,

$$FI(D, minsup) = \{X \in I | supp_D(X) \geq minsup\}$$

Where the support of a transaction is the number of supersets present in the database is described as,

$$supp_D(X) := |\{t \in D | X \subseteq t\}|$$

Unfortunately, frequent pattern mining often produces too many patterns or patterns of low quality. If the minimal support threshold is too large then only commonly known patterns will be found. In contrast, setting the minimal support to a small quantity produces too many patterns, making inspection difficult. Moreover, since a single transaction can support multiple patterns (all of its non empty subsets) then it is likely that patterns reflect the same portion of the database. There are well known methods for processing patterns in order to ensure their quality; using more stringent criteria [Pasquier et al., 1999], and selecting representative patterns [Calders and Goethals, 2007] [Webb, 2010] are excellent approaches to reducing the number of frequent itemsets while improving the quality of results.

2.2 Finding Patterns that Compress

Another approach to processing frequent patterns is to use the *minimum description length* (MDL) principle. This approach, applied to itemset data, selects the itemsets that achieve the most compression. As such, MDL prescribes a preference for the smallest frequent set that characterizes the entire database. The equation below summarizes this approach,

$$\hat{H} := argmin_{H \in H} L(H) + L(D|H)$$

Where $\hat{H}$ is the model that minimizes the equation, $L(H)$ is the length in bits of the model description and $L(D|H)$ is the length of the data encoded with H. Thus, in order to apply this principle to transactional databases it is necessary to define models capable of encoding the data. One approach is to make use of code tables, which are a collection of frequent itemsets that achieve a high compression [Vreeken et al., 2011]. The definition below provides a formal description of a code table,

Code Table [Vreeken et al., 2011] *Let I be a set of items and C a set of code words. A code table CT over I and C is a two-column table such that*:

1. *The first column contains itemsets, that is, subsets over I. This column contains at least all singleton itemsets.*
2. *The second column contains elements from C, such that each element of C occurs at most once.*

An item set $X \in P(X)$ occurs in CT , denoted by $X \in CT$ if X occurs in the first column of CT, similarly for a code $V \in \boldsymbol{C}$. For $X \in CT$, code $CT(X)$ denotes its code, i.e., the corresponding element in the second column.

To make the concept of code tables concrete, an example is shown in Figure 1. On the left side you can view the different itemsets sorted by their frequency. The code table frequency signifies the number of times the code is used to compress a transaction in the database.

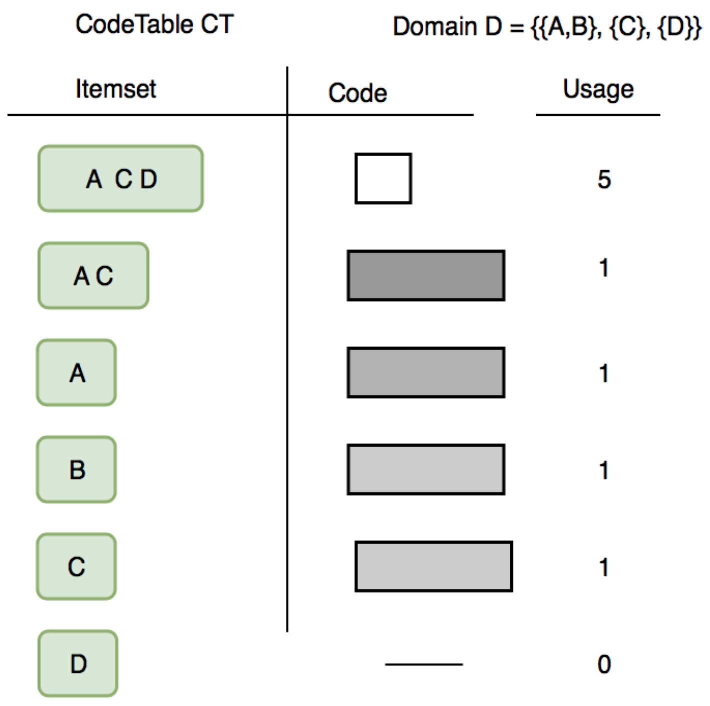

Figure 1: Example of a Code Table over a domain D. Items in the same sub-domain are mutually exclusive.

Furthermore, the COVER algorithm from [Vreeken et al., 2011] describes how transactions are encoded by a code table. In short, each transaction is replaced with the codes of itemsets in its cover. Therefore, the use of code tables and the COVER algorithm allow for the MDL principle to be applied to itemset data. Formally, the problem can be stated as follows,

Minimal Coding Set Problem [Vreeken et al., 2011] *Let I be a set of items and let D be a dataset over I, COVER a cover function, and F a candidate set. Find the smallest coding set $CS \subset F$ such that for the corresponding code table CT the total compressed size, L(D,CT), is minimal.*

The Krimp algorithm provides a greedy approach to the minimal coding set problem [Vreeken et al., 2011] by making use of the COVER function and code tables. In general, it uses the following strategy: start with the singleton itemsets, and iteratively add codes from the candidate set. If they lead to a better compression, keep them. Since considering every possible candidate is infeasible (due to an exponentially large number of itemsets). In order to reduce the search space, Krimp selects all frequent itemsets according to a *minsup* parameter, for its candidate set. Krimp also makes use of pruning in order to avoid being stuck in local minima.

The general approach of clustering by partitioning can be applied to itemset data by utilizing Krimp [van Leeuwen et al., 2009]. In the context of MDL, clustering by partition aims to find clusters (as identified by code tables) of the database such that the total compression is minimized. The result is an algorithm similar to K-means; given an integer K, uniformly split the data into K partitions and apply Krimp to each partition. Thus, K code tables are produced. The next step is to re-assign observations to the code table that compresses it most. This process is repeated until no observation is re-assigned or until a stopping condition is met. Similarly to K-means, the compression improves with each iteration due to the reassignment of observations. The parameter K can be chosen by selecting the number of partitions that yields the maximum compression. In this manner, similar code tables are not likely to be chosen since merging similar clusters will always lead to a better compression. Thus, the algorithm results in the set of code tables that achieve the most compression. Formally it is described as follows,

Krimp Clustering by Partitioning [van Leeuwen et al., 2009] *Let I be a set of items and let db be a bag of transactions over I. Find a partitioning $db_1,...,db_k$ of db and a set of associated code tables $CT_1, ..., CT_k$ such that the total encoded size of d*

$$\sum_{i \in \{1,...,k\}} L(CT_i, db_i)$$

is minimized.

Krimp clustering by partition achieves a higher compression by characterizing the different data generating distributions. For example, in the case of a grocery store, different subgroups of customers exist; customers with children and college students present significant differences in their purchasing behavior. However, it is not the case that they never buy the same product. Instead, it is likely that they present different distributions in the products they buy together.

3 Mining Evolving Patterns

Now consider the question, "How can sets of patterns be compared?" One approach is to compare individual items in each pattern through measures such as Jaccard or edit distance. A better approach is to define similar patterns as the ones that describe the same information. To accomplish this, the code table dissimilarity can be used [van Leeuwen et al., 2009]. This measure compares the compression achieved between code table pairs relative to their respective datasets. The code table dissimilarity is defined as,

$$DS(x,y) = max\left(\frac{CT_y(x) - CT_x(x)}{CT_x(x)}, \frac{CT_x(y) - CT_y(y)}{CT_y(y)}\right)$$

where x and y are different code tables, and $CT_v(k)$ is the compression of v's partition by code table k.

The code table dissimilarity (also referred here as cross compression distance) gives the largest difference between the cross-compression of two code tables. It is then a distance measure of dissimilarity, with similar pairs being close to 0, and dissimilar pairs of code table producing arbitrary large values. Note that the codes need not be equal to produce a low dissimilarity. This ensures robustness as the patterns selected by Krimp may differ between executions and between different datasets, but remain the same in that they describe the same information. Next section examines the results produced by Krimp clustering by partition through the use of the code table dissimilarity.

4 Stability of Clusters

4.1 Generating Synthetic Data

In order to further establish the use of code tables for mining evolving patterns, experiments were done on synthetic data. Data was generated by the procedure described in [Vreeken et al., 2007] with the additional step of also generating code tables. The original objective of this approach is to allow knowledge extraction for a database whilst preserving privacy. With the additional step of sampling a code table, completely synthetic data can be generated. Consequently, this method produces datasets that possess various properties of real datasets, such as mutual exclusion and overlapping. In this section the steps taken to generate data are discussed. Subsequent sections present the resulting analysis.

In Figure 1 you can observe an example of a code table and its itemset domain. As can be readily seen a code table is a set of itemsets and frequency pairs. The frequency of an itemset (or code) signifies the likelihood of a particular code being present in any transaction of the generated data. Note that each itemset is made up of items sampled from differing domains. Thus not all codes can occur in the same transaction.

To create a code table of arbitrary size, simply sample domain elements uniformly for each code in the table while also allowing for the empty set to be sampled. Slightly more formally, given a set D describing the domain of each item, sample n codes by uniformly choosing elements from each of the item domain's. Now sample a frequency value among a range of values. This method is described in detail below, by the function *GenerateCT*. Note that since the empty set can also be sampled, not all codes in the table are mutually exclusive; it is possible that they co-occur in a transaction.

Algorithm 1 Generating a Code Table from a domain of items.

Require: A domain of items D, frequency range R, and the number of codes needed n.

```
function GenerateCT($(R, D, n)$)
    $CT \leftarrow \emptyset$
    $code \leftarrow \emptyset$
    while $|CT| \neq n$ do
        for $d \in D$ do
            $code \leftarrow code \cup Sample(d)$
        $CT \leftarrow CT \cup (code, Sample(R))$
        $code \leftarrow \emptyset$
```

What remains is to generate transactions. This is done as follows; generate a single transaction by sampling codes from the code table with likelihood proportional to their frequencies. Repeat until the transaction is of sufficient size or until all unselected codes overlap with the sampled transaction. The previous steps are repeated until the generated database is of sufficient size. More precisely, given a parameter N indicating the number of rows, while the number of transactions is less than N, repeatedly generate observations by sampling codes from the code table with likelihood proportional to the code's frequencies. Repeat until the transaction is complete (all codes not sampled overlap with the transaction). Additional details of this approach are given in [Vreeken et al., 2007]. Furthermore, to produce data originating from multiple distributions, simply generate data from differing code tables and group all transactions in a single dataset. The data discussed in the following sections was created by this procedure. What results is differing sets of mixture databases.

4.2 Experiments on Synthetic Data

4.2.1 Dissimilarity between Krimp Code Tables

In this subsection the difference between independent executions of Krimp clustering is examined. For this task data was generated by sampling from two code tables (using the previous method discussed) under various sampling proportions ranging from 0 through 1 at 0.1 increment values (See Figure 2). In total 11 databases where generated. Each database contains 50,000 observations and differs in the amount of transactions sampled from a pair of code tables. The sampled proportions range from 0 to 1 by 0.1 increments resulting in a total of 11 datasets. Moreover, in all executions of Krimp clustering the parameter K is set to 2. The candidates considered are all itemsets with minimum support of 10.

Figure 2 shows a boxplot for the compression achieved at each sample proportion for all experiments. We can note that there is not a significant difference in the compressed database sizes, between independent executions, at the same proportion of samples. However, we *can* observe that the mean value of compression changes according to the proportion sampled. It is clear that the compression worsens (the total enclosed size increases) as the proportions equalize. This occurs because the number of observations being compressed by either code table decreases, and consequently, a lower compression is achieved.

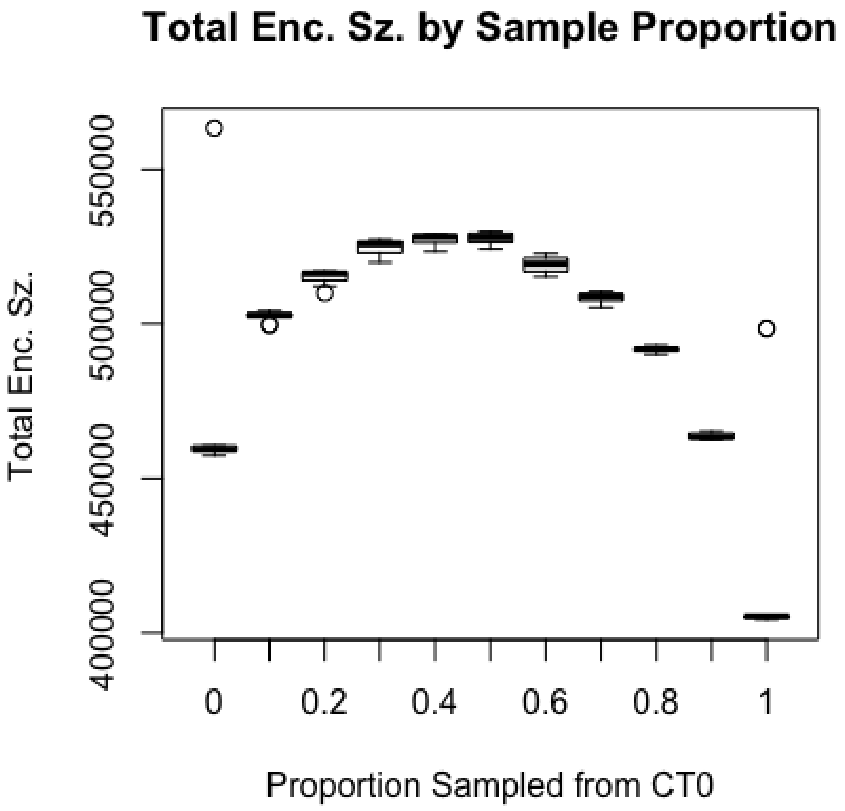

Figure 2: Boxplot of independent Krimp clustering runs, at differing sampling proportions. Each database generated by sampling contains 50 000 rows, with observations being sampled from separate code tables at differing proportions. In total, 10 different independent experiments of Krimp were executed. It is clear that the compression worsens as the proportion equalizes caused by the decreasing number of observations compressed by either code table.

How does the resulting code tables vary between executions of the algorithm? This question can be answered by making use of the code table dissimilarity. In Table 1 you can observe the dissimilarity calculated between a code table pair and itself. What follows is a 2 by 2 dissimilarity matrix with 0 for all entries on the diagonal. These are 0 since there is a complete match between the code tables (since they are the same). In Table 2 you can observe the dissimilarity between differing pairs of code tables. In this case the diagonal entries are not 0. Note that since the algorithm performs a random split the major diagonal does not necessarily correspond to the same portion of the data. Thus, it is necessary to examine both major and minor diagonals to understand the dissimilarity between the two independent executions of the algorithm.

	CT^2_0	CT^2_1
CT^1_0	0	7.97
CT^1_1	7.97	0

Table 1: Dissimilarity between Krimp clustering experiment and itself. Krimp clustering by partition results in two code tables. The dissimilarity distance between a code table and itself is 0. For differing code tables the dissimilarity value indicates by how much larger the compression is achieved (in this case it is more than 7 times larger). The one shown corresponds to the 5th experiment, where the proportion of 0.4 was used.

	CT^2_0	CT^2_1
CT^1_0	5.02	4.47
CT^1_1	4.16	2.69

Table 2: Dissimilarity matrix for differing experiments of Krimp. Each results in two code tables. Due to the random split, code tables do not necessarily represent the same portion of the data (CT^1_0 and CT^2_0 do not necessarily represent the same data). They may be matched or not, meaning that major or maximum diagonal average will indicate the dissimilarity between the two experiments. Similar to Table 1, the matrix shown refers to code table proportion of 0.4.

Averaging the entries in both minor and major diagonals results in the matrices shown in Table 3 and 4. In Table 3, the minimum average diagonal was selected and Table 4 shows the result of selecting the maximum average diagonal for all experiments. Note that both tables correspond to a particular proportion configuration (0.4). In total 22 such tables were produced, each corresponding to a particular sampling proportion where either the maximum or minimum diagonals were averaged. Selecting the lower (or upper) triangular matrices (and ignoring the diagonal) allows us to produce a vector of dissimilarities between independent executions of the Krimp clustering algorithm. Figure 3 summarizes the results of these experiments.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	3.81	3.71	4.065	4.125	3.695	3.88	3.845	3.925	3.53
3.81	0	3.69	3.86	4.215	3.975	3.855	3.93	4.015	3.54
3.71	3.69	0	3.8	3.835	3.795	3.79	3.63	3.745	3.505
4.065	3.86	3.8	0	3.775	3.875	3.99	3.67	3.595	3.365
4.125	4.215	3.835	3.775	0	3.815	4.15	3.92	3.68	3.855
3.695	3.975	3.795	3.875	3.815	0	3.725	3.725	3.82	3.635
3.88	3.855	3.79	3.99	4.15	3.725	0	3.695	3.855	3.85
3.845	3.93	3.63	3.67	3.92	3.725	3.695	0	3.91	3.68
3.925	4.015	3.745	3.595	3.68	3.82	3.855	3.91	0	3.8
3.53	3.54	3.505	3.365	3.855	3.635	3.85	3.68	3.8	0

Table 3: Matrix produced by selecting the smallest diagonal average in all experiments. The results is a symmetric matrix, whose diagonal values are 0. Each entry represents the dissimilarity between a pair of code tables.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
7.97	4.455	4.03	4.15	4.23	4.2	4.12	3.935	4.135	4.23
4.455	8.33	4.39	4.47	4.31	4.225	4.46	4.17	4.275	4.58
4.03	4.39	7.55	4.27	4.33	3.93	4.01	3.985	4.135	4.05
4.15	4.47	4.27	8.22	4.67	4.27	4.255	4.4	4.645	4.735
4.23	4.31	4.33	4.67	8.44	4.49	4.18	4.24	4.655	4.33
4.2	4.225	3.93	4.27	4.49	7.87	4.215	4.045	4.19	4.095
4.12	4.46	4.01	4.255	4.18	4.215	7.79	4.095	4.22	3.965
3.935	4.17	3.985	4.4	4.24	4.045	4.095	7.68	3.965	3.94
4.135	4.275	4.135	4.645	4.655	4.19	4.22	3.965	7.91	4.09
4.23	4.58	4.05	4.735	4.33	4.095	3.965	3.94	4.09	7.45

Table 4: Like Table 3, this matrix represents dissimilarity between code table experiments. However, here the maximum diagonal average was used.

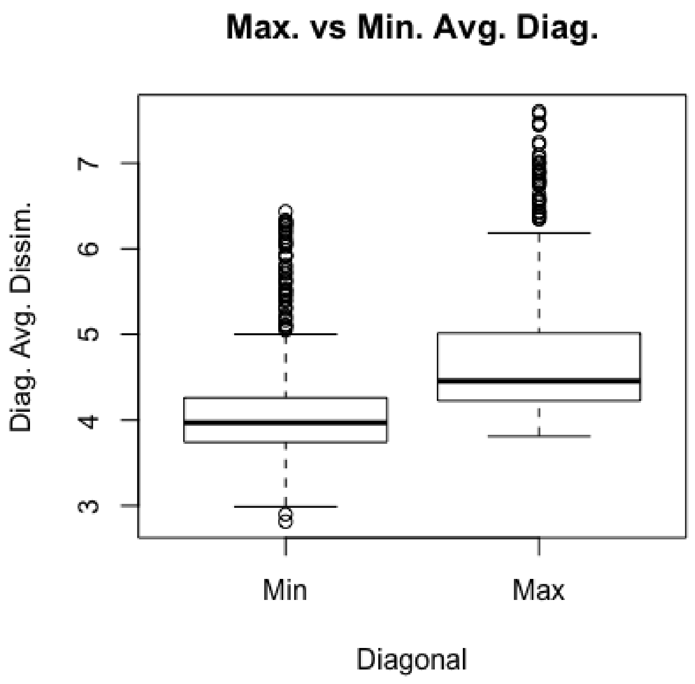

Figure 3: This figure summarizes the previous experiments by showing the overall distribution between the minimum and maximum averages. The differences between code tables among independent experiments is high, with the compression achieved being about four times larger on average.

In Figure 3 you can observe that while there is some difference between the two cases, it does not appear significant. Also, even in the minimal diagonal case the dissimilarity between code tables is also rather high; Figure 3 shows that the compression achieved between a pair of code tables is 4 times worse on average. Thus, we can say there is evidence that code tables differ between independent executions of the algorithm. Consequently, in the case of Krimp clustering it is important to run the algorithm multiple times for improved results. Additional plots are given in Section 8 which show the breakdown of dissimilarities for each of the sampled proportions.

4.2.2 Comparing Krimp clustering results with Originating Code Tables

This section examines an additional property of the algorithm; that of the difference between the code tables used to generate the data and the ones produced by independent runs. To accomplish this task 184 databases were generated. Each containing 25,000 rows, originating from three differing code tables. As before, each dataset was generated by sampling at different proportions. In the table shown below, you can observe the dissimilarity between the data generating code tables.

	CT0	CT1	CT2
CT0		9.73	4.16
CT1			9.23
CT2			

Table 5: Dissimilarity between the code tables used to generate data. The code tables CT0 and CT2 are more similar than other possible code table pairs.

The code tables can be compared as follows: for each of the datasets 10 pairs of code tables were produced by independent runs of the Krimp clustering algorithm. Then the code table dissimilarity measure was calculated between each of the code table pairs with the data generating code tables. Moreover, the 10 dissimilarity values were averaged. The result is 3 average dissimilarity values, indicating the average dissimilarity between all the 10 experiments and each of the originating code tables. For clarity the results are summarized in a pair of ternary plots colored according to the most similar and dissimilar according to the dissimilarity measure.

The ternary plots can be read by looking at the triangle points and parallel lines from each of the triangle sides. The placement of the dots in the graph indicate the relative number of observations sampled from each of the three databases (which sum to 1). The horizontal lines indicate the proportion sampled from $CT1$, which decrease with the distance from its labeled triangle edge. Similarly, the parallel lines from each of the other two edges indicate the proportion of observations sampled. With the proportion decreasing as the distance from the edges increase. Additionally, each dataset corresponds to 10 executions of Krimp clustering via partition algorithm, resulting in 1840 independent executions.

Furthermore, each ternary plot was colored by determining which code table is either the most similar or dissimilar with the particular set of databases. Figure 4 was then colored according to which code table possesses the minimum dissimilarity, and Figure 5 was colored according to the maximum dissimilarity average. Examining Figure 4 more closely we observe that the regions are colored as one would expect; the region most similar to $CT1$ contains the datasets with more than 40% of observations. A similar statement can be made about $CT0$ and $CT1$. Moreover, Figure 5 follows suit with the most distant regions being colored as expected. For additional clarity, the regions shown in orange are the ones for which the minimum, or maximum, dissimilarity was smaller than 0.5. In section 7, Figure 11 clarifies the relationship between the code table dissimilarity and proportion of observations sampled.

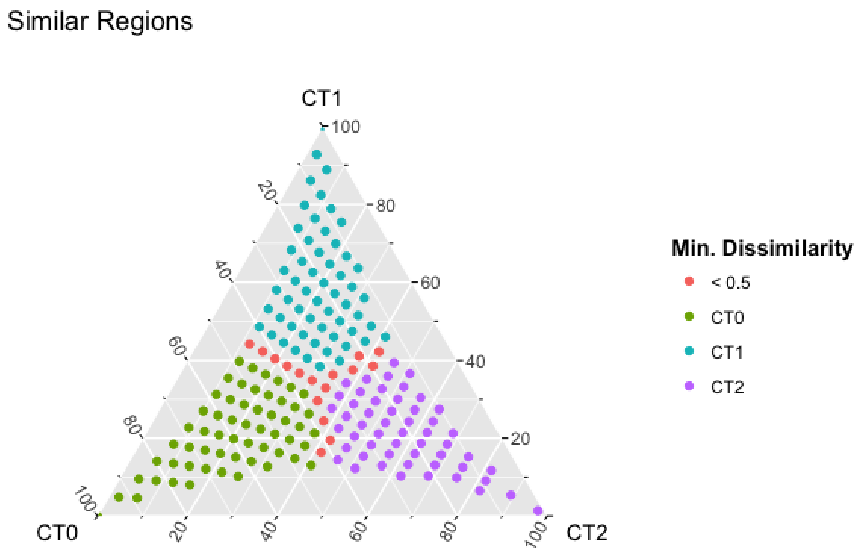

Figure 4: Tenary plot of the generated data, colored according to the most similar originating code tables. Each point correspond to a database, containing a differing proportion sampled from $CT0$, $CT1$ and $CT2$. Moreover, the size of the database before compression is 2,449,425 for all datasets and an average of 80% of reduction in size was achieved.

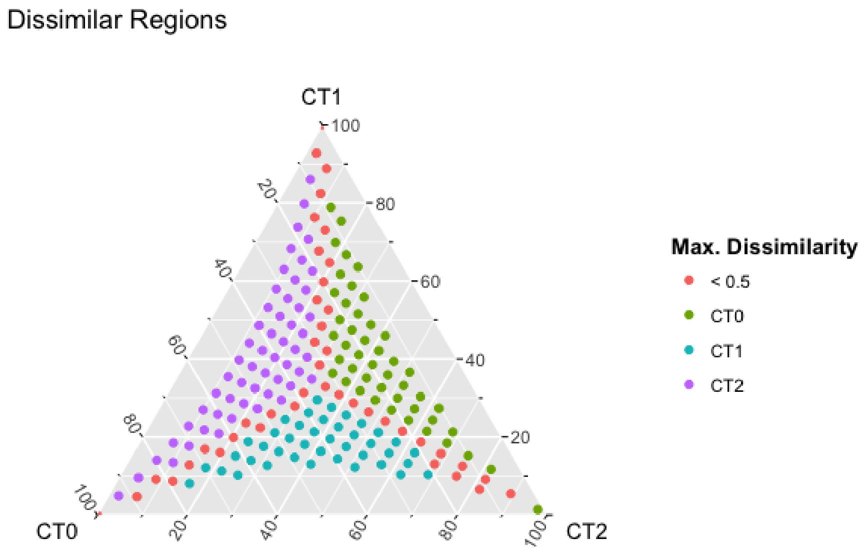

Figure 5: Tenary plot of the generated data, colored according to the most dissimilar originating code tables.

Both Figures 4 and Figures 5 show that the dissimilarity measure can be used to determine the data generating code tables and reflect the proportion of observations sampled. Also, the dividing orange observations indicate a separation boundary between each of the datasets. These results support the claim that Krimp clustering and more generally, compression, is able to explain various relationships present in synthetic data.

4.2.3 Changes in Compression

This section expands on the previous by examining the compression achieved for various values of the K parameter. In Figure 6 shows one path through the ternary plot superimposed on the similarity plot from Figure 4. As before, the points represent the proportions used to generate data. In each of these datasets 10 executions of the Krimp clustering dataset were performed for values of K ranging from 1 (standard Krimp algorithm) to 3. Figure 7 summarizes the results observed.

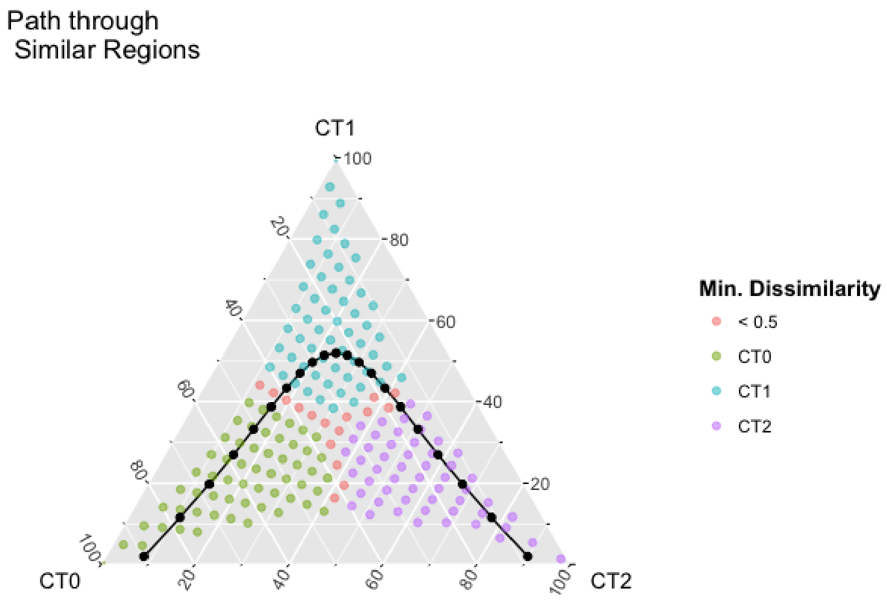

Figure 6: This figure relates to Figure 7 which displays the enclosed or compressed size of each point in the line with the parameter K set to 1, 2, and 3.

In Figure 7 we observe that the worst compression is achieved for the parameter K = 1 for which clusters are not used. Moreover, the compression achieved for K equals 2 and 3 are also shown as averaged values. The results indicate that using two code tables achieves an improvement in performance at the end points of the path. In contrast, using three code tables achieves a better compression for the midpoints of the path. Examining the distribution of values also indicates an increase in variance proportional to K (not shown).

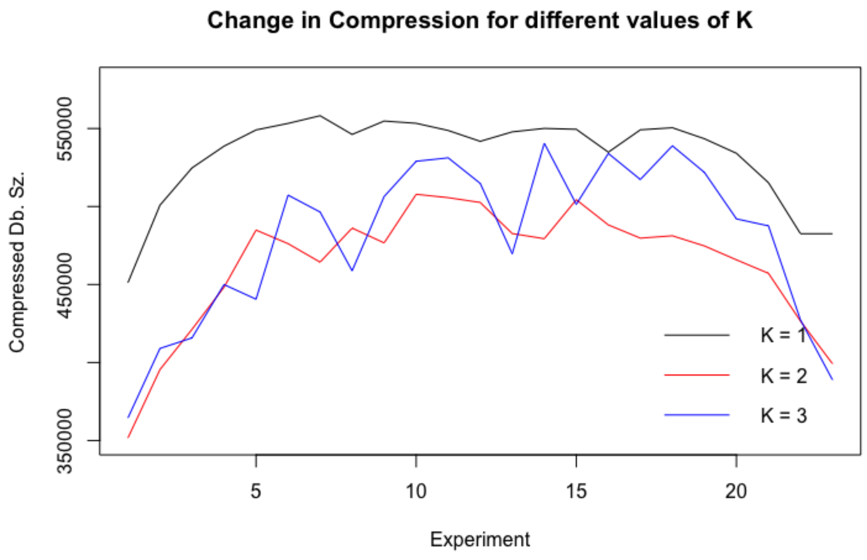

Figure 7: This figure relates to the datasets presented in Figure 7 and show the compressed size of the databases for K values ranging from 1 to 3. The slanting of the lines occur due to the fact that CT2 produces data that compresses worser than CT0.

5 Mining the Evolution of Opinion

In the leading discussion, various properties of Krimp clustering via partition were examined. In this section we return to the objective of identifying subgroups of political opinion, and examining how they change over time. For this task user comments were retrieved from the popular Reddit website, and preprocessed, after which code tables were extracted. The resulting code tables were then compared via the code table dissimilarity measure.

5.1 Extracting and Preprocessing Data

The Reddit comment data was retrieved using PRAW (Python Reddit API Wrapper) [Bryce Boe, 2016] and preprocessed using R's koRpus package [Michalke M. Eik , 2017]. The data consists of a sample of comments from different subreddits as summarized in Table 6. The data was retrieved from the most recent comments, going back to a maximum of 2 years. Comment replies are also included.

Subreddit Data Summary		
Name	Description	$\|Db\|$
The_Donald	Trump's supporters forum.	60,783
Politics	General political discussion forum.	171,855
Esist	Anti-Trump forum.	73,498
WorldNews	World news forum.	156,054

Table 6: Summary of data sources. User comments were extracted going from most recent to a maximum of 2 years back. Comments were preprocessed by removing stopwords, and then lemmatizing terms.

The data was preprocessed as follows. Each Reddit comment was first lowercased, URLs and special characters were removed, then stopwords were removed. Additionally, terms were lemmatized by using koRpus package's *TreeTagger* function. A *Tf-Idf* document term matrix was then created from the entire dataset. Its columns were then summed to produce a *Tf-Idf* score for all the terms in the data. The top 300 terms were chosen to represent the item alphabet and subsequently used to produce itemset data (See Figure 12). The choice of alphabet size was made as balance of minimal transaction size and computational time.

5.2 Changes in Opinion

For each of the code tables, pairs were extracted (parameter K = 2) and subsequently compared. Below you can observe the average code table dissimilarity between each of the datasets. Moreover, Figure 8 shows the overall compression achieved. The average reduction in size for all datasets was 16%. Tables 8 and 9 also show the top non-singleton itemsets for the Politics and WorldNews datasets. Additionally, in the case of the Politics and WorldNews datasets, the data was split into two 3 month groups as shown in Figure 9 and 10. The code table dissimilarity was then calculated between each of these additional datasets and the ones extracted from the Esist and The_Donald dataset to determine changes in political opinion.

In total 10 independent runs of the algorithm were executed with K equaling 2. The choice of parameter was made for the K value that achieved the highest average compression (see figure 13 for an overview of the compression achieved). Overall an average of 16% of compression was achieved. Furthermore, Tables 8 and 9 show the code tables that achieved the highest compression in the Politics and Worldnews datasets. In Table 7 you can observe the pairwise code table dissimilarity. The biggest dissimilarity present is between the pair The_Donald and Esist, which agrees with the expectation that these datasets should present very contrasting opinions. Also, WorldNews is more similar to Esist than The_Donald. The opposite is true for the Politics dataset.

	esist	worldnews	politics	the_donald
worldnews	0.04840			
politics	0.11155	0.1174		
the_donald	0.13735	0.1450	0.07640	

Table 7: Average code table dissimilarity matrix between reddit dataset pairs.

The top 20 codes for the Politics and WorldNews datasets are shown in tables 8 and 9 respectively. In the Politics code table we can observe that groups of terms like (white, house), (putin, russia), (hillary, bernie), and (trump, fake, news) occur together and achieve a relatively high code length. Notably, expletives are grouped together. Together the codes indicate that issues such as climate change, information leaks, national security, and fake news are important in the political discussion in the Reddit's Politics forum. In contrast to the Politics codes, the WorldNews code table does not provide such clear associations. The code that possesses the longest code length is the pair (shit, healthcare) which may indicate that healthcare is an important issue. Other codes like (fake, american) and (good, president) don't present as much direct information when compared to the codes extracted from the Politics dataset.

Codes			Code length
white	house		586.118
change	climate		411.41
security	national		368.367
fuck	shit		326.325
hillary	bernie		322.392
state	unite		312.546
news	fake		301.1086
people	vote	election	292.291
trump	white	house	280.587
trump	supporter		277.964
trump	news	fake	273.509
people	vote		245.893
year	obama		243.249
vote	election		241.745
trump	donald		239.615
president	state	unite	236.288
news	report		227.339
official	intelligence		225.224
information	leak		223.222
russia	putin		222.239

Table 8: Top 20 non-singleton codes of Politics Reddit dataset.

Codes			Code length
shit	healthcare		700.699
fake	american		495.153
president	good		485.566
white	military		448.2483
people	fake		392.1059
american	start		384.1255
big	level		380.434
people	start		338.955
make	american		337.1221
fuck	year		320.416
fake	vote		311.1088
people	make		306.911
people	white	military	305.518
talk	fire		297.296
people	american		297.1062
people	american	start	294.308
happen	man		283.285
show	take		278.277
year	day		275.282
president	vote		274.507

Table 9: Top 20 non-singleton codes of Worldnews Reddit dataset.

Furthermore, Figure 8 and 9 shows how the code table dissimilarity changes in time. In Figure 8 you can observe the change in dissimilarity between the Esist, and The_Donald datasets with a selection of comments from the WorldNews dataset referent to two month periods. The first being the months of January through March, and the second April through June. Figure 8 indicates there isn't a significant change in dissimilarity between these two time periods for both Esist and The_Donald code tables. However, a small increase in dissimilarity is observed for the The_Donald code tables. This small increase may relate to a negative perception of the Trump administration.

Figure 9 compares the Esist and The_Donald code tables to the Politics code tables for the same time periods of Figure 8. A significant change in dissimilarity was also not observed. The distribution of dissimilarities did not significantly differ for both the Esist and The_Donald code tables.

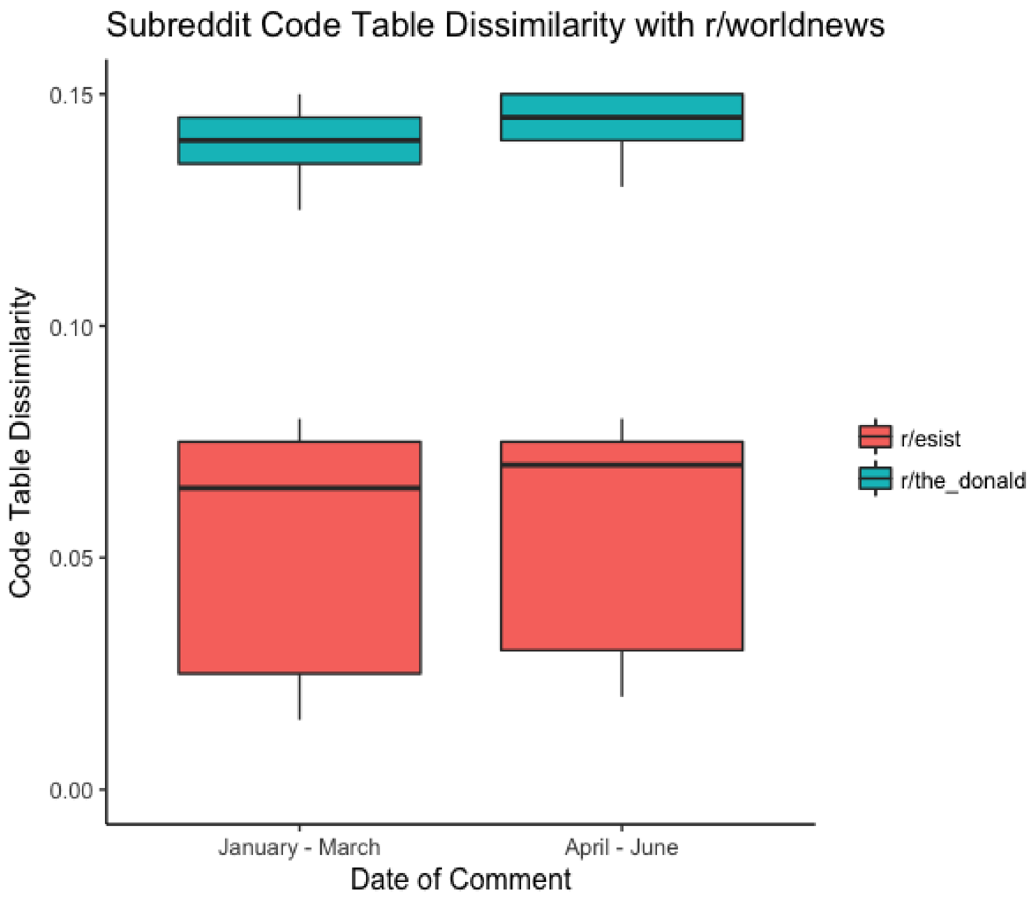

Figure 8: Boxplot of code table dissimilarity between Esist and The_Donald with the Worldnews dataset split into two datasets. The first containing data only from the months of January through March and second April through June. A small increase in dissimilarity can be observed between the two periods in the case of The_Donald dataset.

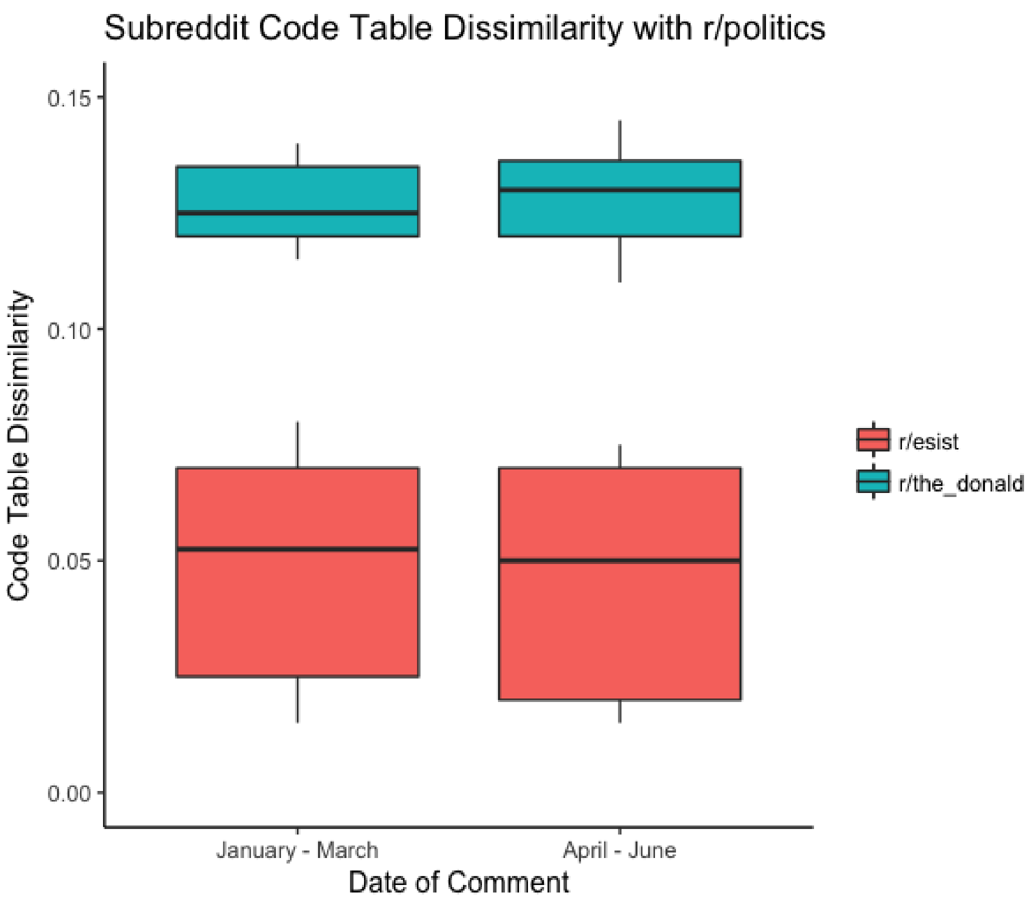

Figure 9: Boxplot of code table dissimilarity between Esist and The_Donald with the Politics dataset split into two datasets. The first containing data only from the months of January through March and second April through June. No significant change in the code table dissimilarity can be observed.

6 Conclusion

This thesis examined political opinion through the extraction of code tables from Reddit comments. In the previous section code tables were extracted from Reddit's Politics and WorldNews forums and subsequently compared with comments from Esist, and The_Donald. The comparison of Esist and The_Donald code tables showed that while a significant difference in dissimilarity was not observed according to time, the use of compression through the Krimp algorithm is able to extract a high level of meaning from unlabeled data. The codes extracted are both interesting on their own, and can be made use of directly through the code table dissimilarity function. However, in this approach a significant difference in regards to time was not observed. This is likely due to the low amount of compression achieved on the data.

When examining synthetic data, a high compression was achieved and subsequent experiments showed that Krimp and the code table dissimilarities can be used to accurately detect groups and reflect the original distributions present in the data. Additionally, while the code tables are affected by the initial random split, in the case of Krimp clustering by partition, they remain similar to distributions present in the data. The experiments also showed that the underlying changes in distribution directly influenced the compression observed for different values the K parameter. In all, experiments with synthetic data showed the usefulness of code tables but also indicate steps should be taken to minimize the influence of the random initial split. One approach to reduce the variance between independent executions of Krimp is to make the initial assignment of clusters according to a probability through the use of a 'pivot transaction'. A pivot transaction is one that is chosen to represent a cluster. Subsequent transaction assignments can then be done according to a probability proportional to a dissimilarity measure. The choice of measure can be made in many ways, but the code table dissimilarity can also be used when considering all subsets of the chosen pivot transaction as a code table. Both theoretical analysis and experimental results are needed to examine this approach and determine its performance in practice. Additional consideration is also needed when a higher number of clusters is required.

In conclusion, the code tables produced by Krimp capture the underlying information present in data. Therefore it can be concluded that both compression and code table dissimilarity are useful tools to understand changes in such datasets. Through the use of synthetic data, the use of code tables was shown to be effective in practice, with underlying changes in distribution being detected. Moreover, the resulting approach was used in real data to extract and examine changes in comments from Reddits forum, with a focus given to political discussion. The code tables extracted present a significant level of information in the case of Reddit's Politics forum, and while small, some changes were observed when compared to pro-Trump The_Donald and anti-trump Esist forums. The observed changes indicate there was a shift away from pro-Trump discussion. All together, the examination of political opinion in social media presents both technically interesting and socially important challenges due to the increasing use of information to misguide voters and influence opinion through propaganda. Therefore further development of unsupervised learning approaches are becoming increasingly important and likely to achieve improved results.

7 Additional Figures

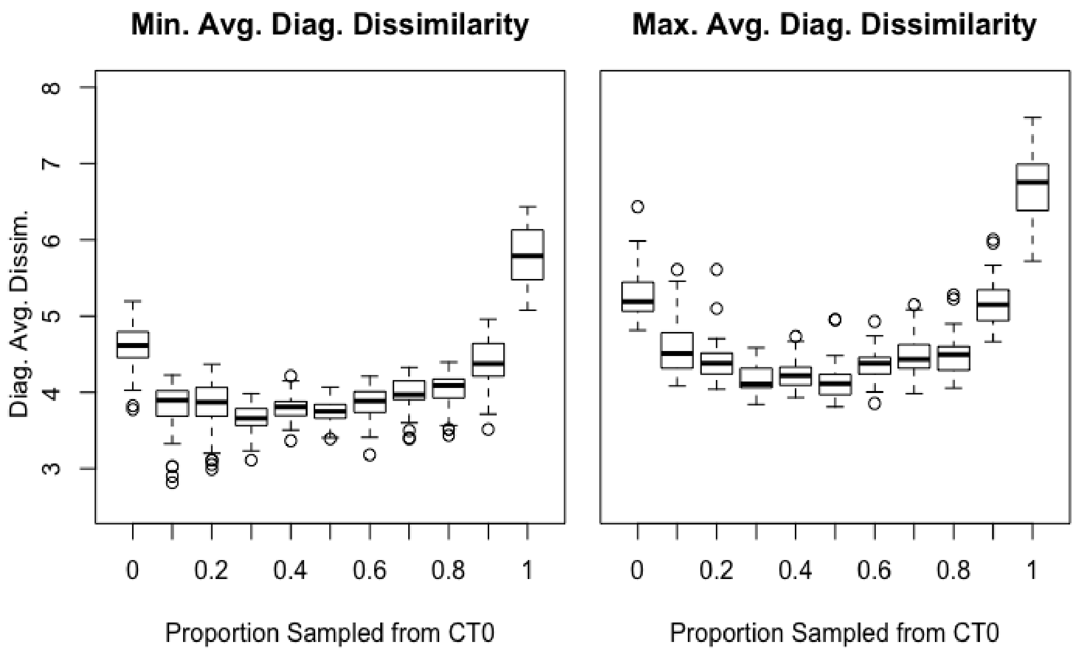

Figure 10: Comparison of minimum and maximum diagonals for the dissimilarity of two code tables. For both the diagonal was ignored, and the lower triangular matrix was used. The result is 45 dissimilarity measures between independent runs of Krimp clustering by partition.

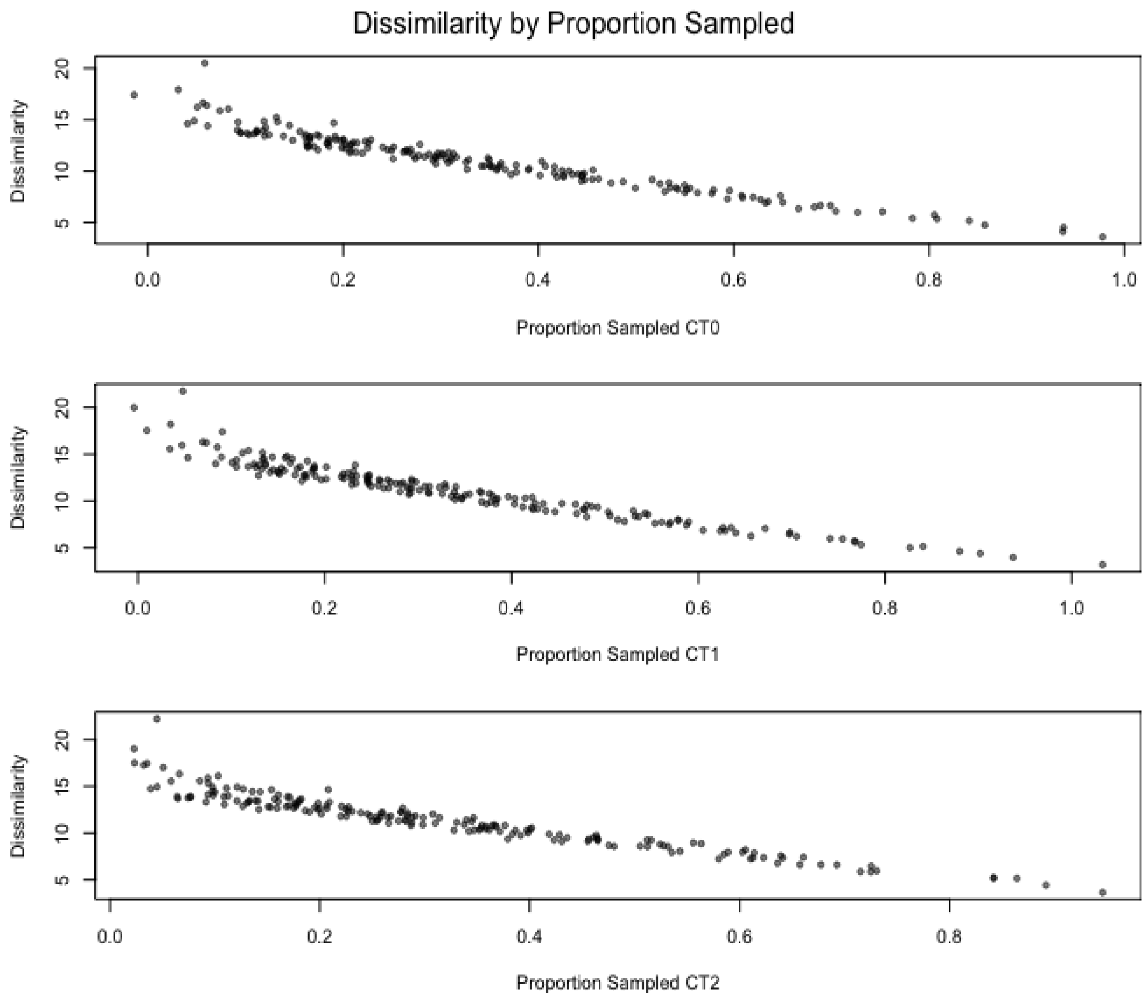

Figure 11: This figure shows dissimilarity as a function of the sampled proportion, for the case of three synthetic code tables. The dissimilarity is measured between each experiment (database) and the code tables used to generate the data. For clarity see Section 5, and Figured 4 and 5.

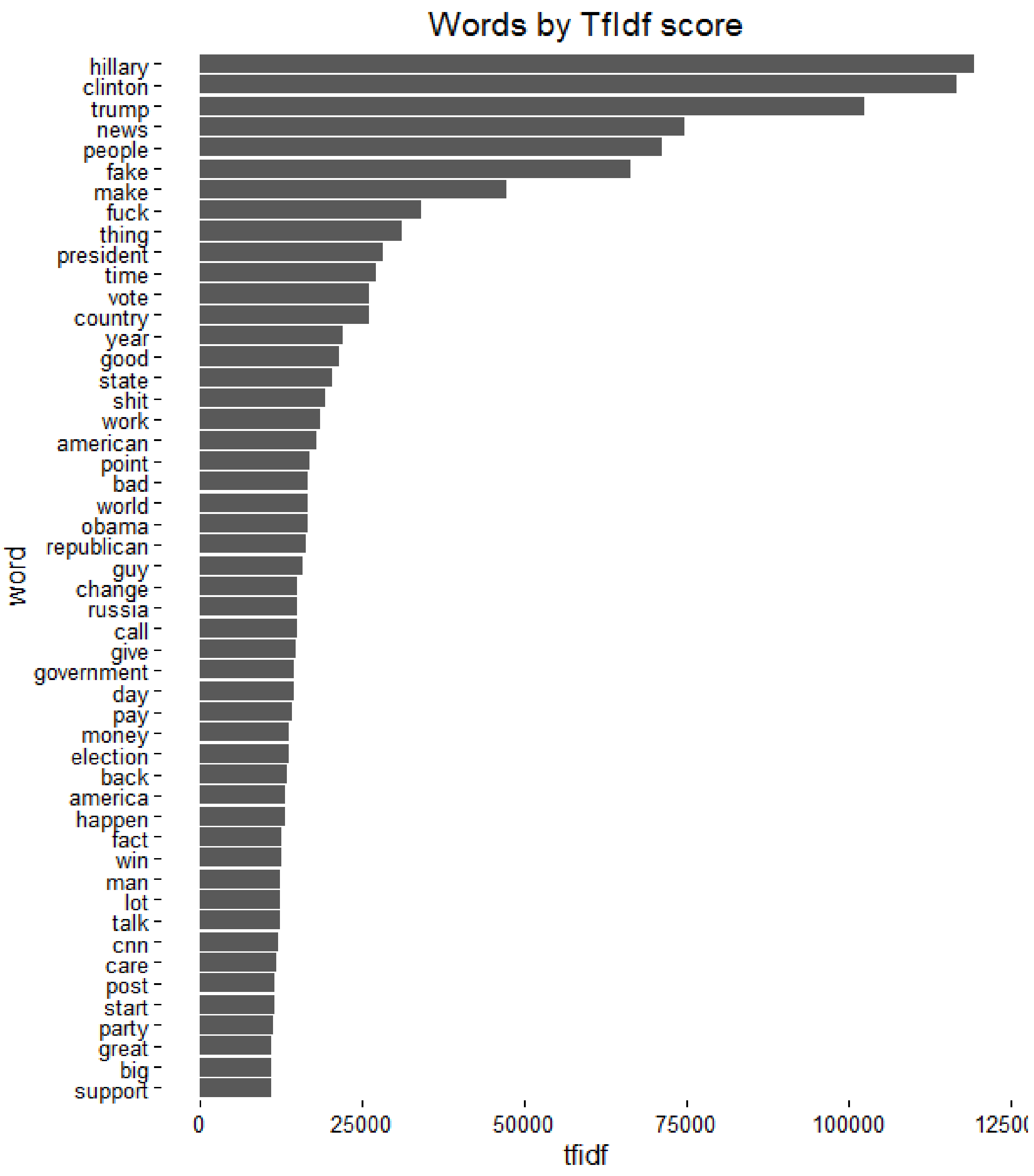

Figure 12: Terms ordered by TfIdf score obtaining from all the reddit datasets.

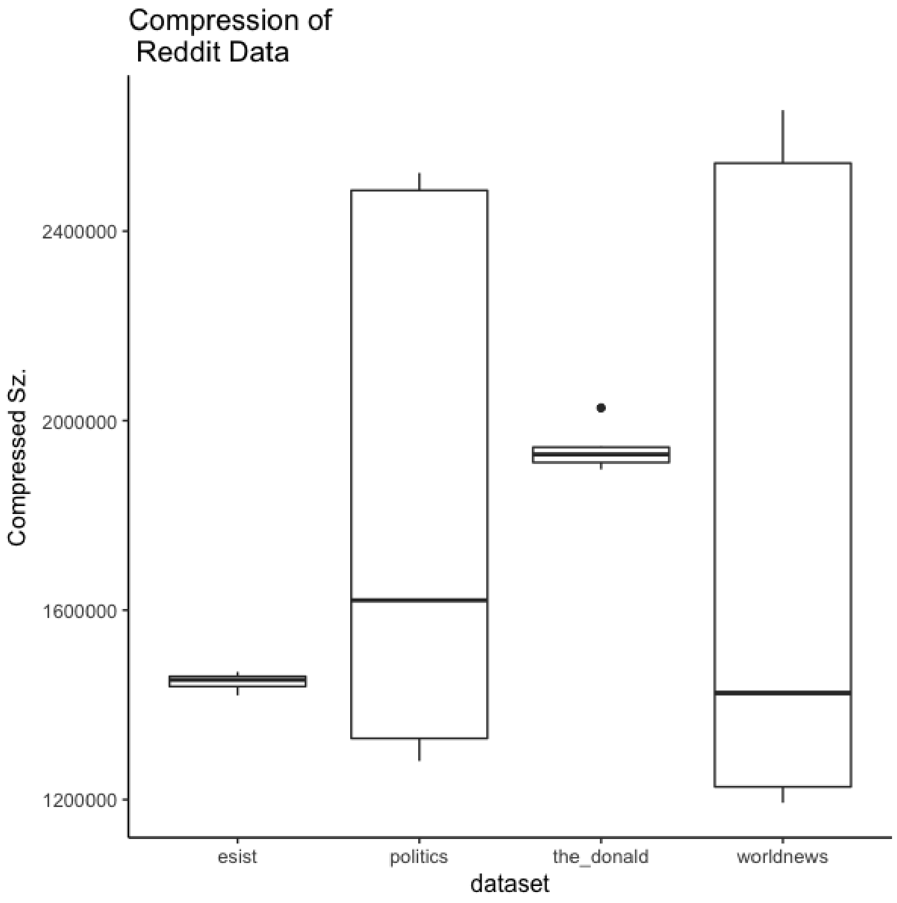

Figure 13: Compression achieved for each of the Reddit datasets for 10 executions and K = 2.

References

[reddit,] Podesta emails magathread r/the_donald. https://www.reddit.com/r/The_Donald/comments/57vefh/podesta_emails_magathread/

[Agrawal et al., 1996] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A. I., et al. (1996). Fast discovery of association rules. *Advances in knowledge discovery and data mining*, 12(1):307–328.

[Michalke M. Eik , 2017] koRpus: An R Package for Text Analysis (Version 0.10-2). Available from https://reaktanz.de/?c=hacking&s=koRpus

[Bryce Boe, 2016] Bryce Boe, 2016. Python reddit api wrapper (PRAW). https://github.com/praw-dev/praw

[Calders and Goethals, 2007] Calders, T. and Goethals, B. (2007). Non-derivable itemset mining. *Data Mining and Knowledge Discovery*, 14(1):171–206.

[Green and Issenberg, 2016] Green, J. and Issenberg, S. (2016). Inside the trump bunker, with 12 days to go.

[Lee and Quealy, 2016] Lee, J. C. and Quealy, K. (2016). The 325 people, places and things donald trump has insulted on twitter: A complete list. *The New York Times*.

[Pasquier et al., 1999] Pasquier, N., Bastide, Y., Taouil, R., and Lakhal, L. (1999). Efficient mining of association rules using closed itemset lattices. *Information systems*, 24(1):25–46.

[Roller, 2016] Roller, E. (2016). Opinion — hillary clinton has sent you a friend request.

[van Leeuwen et al., 2009] van Leeuwen, M., Vreeken, J., and Siebes, A. (2009). Identifying the components. *Data Mining and Knowledge Discovery*, 19(2):176–193.

[Vreeken et al., 2007] Vreeken, J., Van Leeuwen, M., and Siebes, A. (2007). Preserving privacy through data generation. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 685–690. IEEE.

[Vreeken et al., 2011] Vreeken, J., Van Leeuwen, M., and Siebes, A. (2011). Krimp: mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214.

[Webb, 2010] Webb, G. I. (2010). Self-sufficient itemsets: An approach to screening potentially interesting associations between items. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1):3.