CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Towards Topic Detection Using Minimal New Sets

Master's Thesis in Computer Systems and Networks & Computer Science - Algorithms Languages and Logic

MADELEINE APPERT
LISA STENBERG

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017

Towards Topic Detection Using Minimal New Sets

Master's thesis in Computer Science and Engineering

MADELEINE APPERT
LISA STENBERG

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Towards Topic Detection Using Minimal New Sets
MADELEINE APPERT
LISA STENBERG

Supervisors:
Alexander Schliep, Department of Computer Science and Engineering
Birgit Grohe, Department of Computer Science and Engineering
John Wiedenhoeft, Department of Computer Science and Engineering
Francesco Bruno, DEVnet
Rupert Hughes, DEVnet

Examiner: Graham Kemp, Department of Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in $\LaTeX$
Printed by [Name of printing company]
Gothenburg, Sweden 2017

Towards Topic Detection Using Minimal New Sets
Madeleine Appert
Lisa Stenberg
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

A common way of detecting topics in a collection of articles is to use Hierarchical Clustering. This has shown to be a successful way of clustering texts, and thereby detecting topics. However, this is computationally expensive since the similarities between all articles are compared pairwise.

This thesis aims to examine if smaller amounts of data could be used to detect topics. Building on the work of Damaschke [9] and Guðjónsson [13], we processed articles as a sequence of chronologically ordered documents, and represented each document by the previously unseen word combinations, more formally known as minimal new sets of words. Based on the words that each article now is represented by, we selected articles with a word or word combination. We compared this selection to a ground truth created with Hierarchical Clustering, to see if the minimal new sets can be used to approximate clustering in a streaming setting.

We performed three experiments and evaluated their results. In the first we selected articles based on one given word. In the second we selected articles based on a given two-word combination. In the third we built on the second experiment, but separated the selected articles if two consecutive articles were not published within a given time limit.

Out of the experiments that we performed we found that tracking a pair of words gave the best result. Additionally, we found that the Jaccard index of the word combinations impacted the result, where words appearing more often together gave better results.

The results indicate that minimal new sets can be used to detect topics. Our model shows significantly better results than the corresponding random model. However, we still do not consider our model to hold up against established methods. Therefore, we do not think that our current method is suitable for a topic detecting system, but rather that it could be possible to build on our methods.

Keywords: Minimal New Sets, Topic Detection, Text Mining

Acknowledgements

This project has been a time full of hard work, laughs and self reflection. We would like to express our gratitude for taking us on as a master thesis project. We know that our circumstances were somewhat unique. We so appreciate the knowledge, assistance and insights that have been shared so generously throughout the course of this project. Thank you for all the late night peer reviews and all the extra energy supplied through chocolate, brainstorming and pep talks. Special thanks, too, for all the support and hugs when we have been feeling down, for all the high fives and popped bottles for every milestone reached. Lastly, the sheer patience, flexibility and understanding you have shown us has made this long-distance arrangement work wonderfully - both in a personal and professional capacity! Vielen Dank!

Madeleine Appert & Lisa Stenberg, Munich, November 2017

Contents

List of Figures

List of Tables

1 Introduction

With the rapid growth of technology in the past few decades, the amount of information available has increased. Unlike previous generations, we have instant access to news in digital form from all around the world, as soon as they have been published. Being well-informed is a necessity within a business as it enables leaders to make strategic decisions. However, processing the vast amount of information available to us has become a time-consuming activity. This motivates the need for tools that gather relevant information and optimize information absorption.

Topic Detection and Tracking (TDT) is a research program which studies event-based organization in broadcast news [4]. TDT mainly studies story segmentation, identifying the stories that are the first to discuss a new event and tracking stories about a specific event or a specific topic. We present a more formal definition of a topic in Section 2.2, but for now we can consider a topic as several events that are related to each other. A common way of detecting topics in a collection of articles is to use Hierarchical Clustering [32]. An article is often transformed into some vector representation, where the vectors are compared to determine how to form the clusters. Even though this has shown to be a successful way of clustering articles, it is computationally expensive, since all vectors are compared pairwise. Therefore we want to investigate if smaller amounts of data can be used to detect topics.

Guðjónsson [13] presents a new approach to On-line New Event Detection which builds upon identifying minimal new sets of words from a sequence of articles and using those to determine if the text referred to a new event. We aim to build on their findings and use the minimal new sets from an article to create a smaller representation of the article, and investigate if this representation can be used to approximate clustering in a streaming setting. We call this model the MNS Chaining model. In order to evaluate the correlation between features of minimal new sets and topics, we define ground truth topics by clustering. Our goal is not to develop an algorithm to detect topics or trends but rather to study to what degree minimal new sets and trending topics correlate.

This project is carried out at DEVnet which is a consultancy company based in Munich, Germany. DEVnet is in need of a tool that can segment articles into topics and from that find the most relevant topics at a point in time. Based on our results we aim to make recommendations of relevant articles from the given stream and supply DEVnet with a tool that can be used within Business Intelligence.

2
Background

In this chapter, we introduce basic terminology and concepts that we use in our thesis. We present a formal definition of both minimal new sets and definitions of events and topics. Finally, we discuss previous work within the field and how it relates to our work.

2.1 Definition of Minimal New Sets

The formal definition of minimal new sets, MNS, that we use in the scope of our studies is originally presented by Damaschke [9].

Definition 1. *Let $B_0, B_1, B_2, ...B_{m-1}$ be a sequence of sets that we call bags. For another bag $B := B_m$ we call a subset $X \subseteq B$ new at m, if X was not already a subset of an earlier bag : $\forall i < m : X \setminus B_i \neq \emptyset$. Otherwise X is said to be old at m. We call $X \subseteq B$ minimal new at m if X is new and also minimal (with respect to inclusion) with this property.*

Consider one article, and a number *k*. Let k define the maximum size of a *set* of words. This set of words is a subset of words that occur in a close neighborhood within the article. A close neighborhood can be e.g. a few sentences at a time. Let a *bag* for such an article be a multiset consisting of all such sets, meaning that each word keeps its multiplicity. The definition of a *new set* is a set that is not a subset of any of the previously seen bags (of previous articles). *Minimal sets* are sets that are not supersets to any other sets. A *minimal new set* fulfills both of these criteria [9].

As an example, suppose $k = 2$, and we have already processed three articles providing us with the bags:
B_0 = `{{'quick'}, {'dog'}, {'jump'}}`
B_1 = `{{'brown'}, {'bear'}}`
B_2 = `{{'brown', 'dog'}, {'jump', 'bear'}, {'brown', 'jump'}, {'dog', 'bear'}}`

We now want to create the bag B_3 which will correspond to article A_3, where A_3 = `"The quick brown fox jumps over the lazy dog."`. After the text has been preprocessed, which we expand on in 3.1.1, the following text remains: `"quick brown fox jump lazy dog"`. The only words that have not been seen previously are `'fox'`, and `'lazy'`. Therefore, these words are added to B_3 as MNS of size 1. Any set of size 2 containing either of these words would be new but not minimal.

The following words remain, and can not be added as MNS of size 1 since they are not new: `'quick'`, `'brown'`, `'jump'`, and `'dog'`. There exist six possible sets of size 2 that could be created from these words. Some of these have been seen in previous bags and would not be considered to be new:
Seen in B_0: `{'jump', 'dog'}, {'quick', 'dog'}, {'quick', 'jump'}`
Seen in B_2: `{'brown', 'dog'}, {'brown', 'jump'}`

The only set that has not been seen before is `{'quick', 'brown'}`, which is added to B_3. We now have the bag for article A_3, which looks as follows:
$B_3 =$ `{{'fox'}, {'lazy'}, {'quick', 'brown'}}`

2.2 Definitions of Events and Topics

Events can be defined in several ways. For example, an event can be the earthquake that took place in Japan 2011, another event can be the tsunami that follows the earthquake. Some definitions of events leave it unclear if these are counted as one single event or two separate events. We define an event in the same way as the research program Topic Detection and Tracking (TDT) defines it [8], which is: "a specific thing that happens at a specific time and place along with all necessary preconditions and unavoidable consequences". This means that the earthquake and the tsunami are the same event. We use the TDT definition of a topic which is “a seminal event or activity, along with directly related events and activities” [3].

2.3 Related Work

Guðjónsson [13] believed that one is likely to find new combinations of words within an article written about a new event. Guðjónsson created minimal sets using an overlapping sliding window on the documents, considering three sentences at a time. This was used to limit the amount of data, and also built on the idea that words in close succession are more likely to relate to each other. Important words in close succession were therefore joined into minimal sets. The sets found within a window were added to a temporary sub-bag while a text was being processed. In the end, the union of all sub-bags made the final bag that represents a text. Guðjónsson limits the maximum size of his sets to 2, based on the likelihood that small MNS would prove more informative than larger ones, as discussed by Damaschke [9]. Damaschke showed that the use of MNS was a good indication when finding new events.

Fung et al. [10] worked on a problem called ‘hot bursty events detection’ where they applied an algorithm to a sequence of chronologically ordered articles. They defined a "hot bursty event" as a minimal set of bursty features that occur together in a certain time window. One way to detect bursty features in a text is to use the term frequency-inverse document frequency (tf-idf) schema [10], which is one method used in [13]. This is described in more detail in section 3.1.2.

A common way to detect topics is to use hierarchical clustering. Garcia et al. [11] introduced a hierarchical agglomerative algorithm that can be used to cluster both static and dynamic data sets. The clusters are obtained by calculating the similarity measure between clusters, and creating graphs for each level in the tree. Their algorithm, however, creates much fewer levels and clusters than other methods such as Average-link [11], Complete-link [16] and Bisecting K-Means [11]. Their results show that their algorithm performs well for dynamic data sets, requiring less computational time and achieving a comparable or even better clustering quality than standard methods.

Li et al. [22] introduce two new text clustering algorithms which both are based on the idea that word sequences in documents play an important role to deliver the correct meaning. They have two requirements for their algorithms. One is associating a label to each cluster while running, which is time-consuming to do afterwards. This gives a good description of the clusters and makes it possible to browse through them quickly. Also, it should not be necessary to define the number of clusters that the algorithm creates as they state that it's difficult to know the number of clusters in advance.

3 Methods

This chapter consists of two main sections. The first part describes the methods and theory behind our model, which we call the MNS Chaining model, and the second part presents the ground truth, i.e. what we compare our model to.

3.1 The MNS Chaining Model

In this section, we present methods used for preprocessing the text, how the minimal new sets are enumerated, and how we find them. We also describe how we link articles together based on the MNS.

3.1.1 Preprocessing

Gathering useful information from text is a process also known as *text mining.* Preprocessing is an important first step to text mining, as it results in higher quality data [36]. Articles are segmented into sentences and each sentence is tokenized, i.e. represented as a list of words. Text is converted to lowercase, and punctuation is removed along with stop words [5]. Stop words are the most common and generally unimportant words within a language. A few examples of stop words in the English language are 'and', 'if', and 'the'. All remaining words are stemmed using a Porter2 stemmer [29]. Stemming removes various suffixes allowing variations of a word to be recognized as the same word, thus saving time and memory space [36]. The result after stemming is not always an actual word itself. For example, while 'argument' and 'arguments' both are stemmed to 'argument', the stemmed form of the words 'argue', 'argued', 'argues', and 'arguing is 'argu'.

Recognizing and classifying entities such as people, organizations, locations, and dates is an important task of Natural Language Processing (NLP) known as Named Entity Recognition (NER) [17]. This is also included as a part of our tokenization process. Introducing Named Entity Recognition to our preprocessing allows an entity such as `'The United States of America'` to be identified and treated as one token rather than a list of tokens, i.e. `['The', 'United', 'States', 'of', 'America']` . By introducing Named Entity Recognition `'The United States of America'` can be used as one part of an MNS, rather than using a part of the entity name. This means that we could avoid finding MNS such as `{'United', 'States'}` or `{'New', 'York'}` but rather `{'The United States of America', '...'}` and `{'New York', '...'}`.

We test three different libraries to perform Named Entity Recognition: Natural Language Toolkit (NLTK) [5], spaCy [14], and Stanford NER [12]. A comparison of the results can be seen in Listing 3.1, which also includes the runtime for each library. The only library that recognizes a date is spaCy, as can be seen in Sentence 1. NLTK is not able to identify a person by their full name if the sentence begins with the name, which can be seen in Sentence 4 and Sentence 5. While comparing these two sentences, another interesting finding with Stanford NER is the inability to identify 'the Museum of Natural History' as a named entity when the order of the two people being mentioned is switched. In Sentence 2, we see that Stanford NER can not separate two succeeding named entities, whereas spaCy can not find the second one.

While Stanford shows promising results, it is far too slow to be a viable candidate. We choose to continue performing Named Entity Recognition using spaCy as it was both the fastest performing library and provided the best results.

3.1.2 Calculating the Importance of Words

Within Natural Language Processing the importance for each word is often calculated and used either as a weight or a filter. There are different methods of quantifying the importance of a word for a text. Intuitively, one could argue that a word mentioned many times in one article is likely to be more important. However, a word that is mentioned often in a large collection of texts is likely to be less important to one specific text. Based on these ideas, there are a few metrics that are commonly used to determine the importance of words [23].

- *Term frequency* ($\text{tf}_{t,d}$) - The number of times a term t appears within a given document d.
- *Document frequency* (df_t) - The number of documents in the collection that contains the term t.
- *Inverse document frequency* (idf_t) - A weighted value of document frequency scaled to the number of total documents, N, in the corpus. See Equation 3.1. The logarithm is used in the formula to decrease the impact of high document frequencies [18].
$$\text{idf}_t = \log \frac{N}{\text{df}_t} \tag{3.1}$$
- *tf-idf* - A combined value of term frequency and inverse document frequency. See Equation 3.2. This value will increase if the given term is mentioned frequently within an article or if it exists within a small part of the documents. Similarly, the value will decrease if the given term is seldom mentioned within an article or if it exists within a large part of the collection's documents.

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t \tag{3.2}$$

Guðjónsson [13] used an incremental tf-idf model, meaning that the document frequency for each term was recalculated for every new document processed. He did not set up an initial document frequency. As a consequence, all terms received too little weight in the beginning since they appeared in such a large part of the seen documents. Another option is to use a static model, where a training set is used to establish the document frequencies. However, if the model is never updated, a term that did not appear in the training set will always be considered new and will therefore receive too much weight over time [6]. For these reasons, we choose to use a combination of the two approaches. We use an initial six months of articles for each data set as a training set to establish initial document frequency. We then update the document frequencies incrementally, similar to Guðjónsson.

```
Sentence 1: Rami Eid studies at Stony Brook University in New York until May 2018
Expected: ['Rami Eid', 'Stony Brook University', 'New York', 'May 2018']

NLTK:     ['Rami', 'Eid', 'Stony Brook University', 'New York']
STANFORD: ['Rami Eid', 'Stony Brook University', 'New York']
SPACY:    ['Rami Eid', 'Stony Brook University', 'New York', 'May 2018']
---------------------------------------------
Sentence 2: Mary Shelley's Frankenstein is a great movie.
Expected: ['Mary Shelley's', 'Frankenstein']

NLTK:     ['Mary', 'Shelley', 'Frankenstein']
STANFORD: ["Mary␣Shelley's␣Frankenstein"]
SPACY:    ["Mary␣Shelley's"]
---------------------------------------------
Sentence 3: I like The Bike Shop, it's better than the bike shop next door.
Expected: ['The Bike Shop']

NLTK:     ['Bike Shop']
STANFORD: []
SPACY:    ['The Bike Shop']
---------------------------------------------
Sentence 4: Alice Andersson and Emma went to the Museum of Natural History.
Expected: ['Alice Andersson', 'Emma', 'the Museum of Natural History']

NLTK:     ['Andersson', 'Emma', 'Museum', 'Natural History']
STANFORD: ['Alice Andersson', 'Emma', 'Museum of Natural History.']
SPACY:    ['Alice Andersson', 'Emma', 'the Museum of Natural History']
---------------------------------------------
Sentence 5: Emma and Alice Anderson went to the Museum of Natural History.
Expected: ['Emma', 'Alice Anderson', 'the Museum of Natural History']

NLTK:     ['Emma', 'Alice Anderson', 'Museum', 'Natural History']
STANFORD: ['Emma', 'Alice Anderson']
SPACY:    ['Emma', 'Alice Anderson', 'the Museum of Natural History']
---------------------------------------------

Runtime:
NLTK:     0:00:00.462305
STANFORD: 0:00:20.160481
SPACY:    0:00:00.027018
```

Listing 3.1: A comparison of three libraries used for named entity recognition, namely Natural Language Toolkit (NLTK) [5], spaCy [14], and Stanford NER [12].

3.1.3 Finding Minimal New Sets

The minimal new sets are identified partly using the same methods as Guðjónsson [13]. In this section we explain the theory behind that method, along with our decisions and improvements.

3.1.3.1 Enumeration of Minimal New Sets

Guðjónsson's implementation builds on the efficient algorithm presented by Damaschke [9].

Consider that articles are processed sequentially and the MNS of the nth article are stored in bag B_n. At time m we have the enumerations of previously processed articles stored in the bags $B_0, B_1, B_2, ...B_{m-1}$, and the current task is to find all minimal new sets of bag B_m. Candidate sets $X \subset B_m$ are generated of increasing size until all candidates are supersets of minimal new sets that have been discovered in article m. Initially all sets of size 1, i.e. words, that are new at m are located and ignored from the upcoming enumerations of a larger size. From the remaining words, all possible sets of size 2, i.e. pairs, are generated and checked if they are new at m. All sets of size 2 that are not new at m will be used to find sets of size 3 etc.

The way to determine if a candidate set is new is what makes the algorithm efficient. Consider that we are processing candidate sets of B_m and we want to know if X is new at m. Guðjónsson defined a function f to check where a set was first found. The function f is defined as: $f(X) := \min\{i | X \subseteq B_i\}$. If a value for $f(X)$ $(= i)$ exists, then X was new at i and will be considered old at any subsequent index $j > i$. However, if X is new at m, thus not a subset of any bag B_i, then $f(X)$ is undefined. If X is new at m, we set $f(X) = m$.

To determine if a set of size 1 is new at m, we check if it has been seen in any previous bag. Since we are looking for sets that are both minimal and new, we must consider that for any MNS $X \subset B_m$, all candidate sets $Y \subset B_m$ where $X \subsetneq Y$, will also be considered new at m, but not minimal.

Therefore, for a set Y where $|Y| > 1$, it is not always enough to check if $f(Y)$ is defined, to know if it is new. If a value for $f(Y)$ exists, it is trivial to tell that Y is old. However, if no such value exists, Y could still be old. To check if Y is a new set, we must therefore check $f(X)$ for each $X \subsetneq Y$. If $f(X) =$ i is stored, we check if $Y \subset B_i$. If the latter is true, this implies that Y was new at i but not minimal since it is a superset of X, and therefore Y is not new at m.

Based on Guðjónsson's findings, we limit the enumerations to only consider subsets of size $k \leq 2$. Additionally, Damaschke [9] discusses that minimal new pairs are easy to find, while sets with $k > 2$ are rare. The differences between Guðjónsson's method to find sets and our own are explained further in Section 3.1.3.3.

3.1.3.2 Overlapping Sliding Window

Guðjónsson used an overlapping sliding window to process three sentences at a time. He found that this approach achieved good results and a significant increase in performance [13]. Therefore, we choose to process our text using the same approach. However, if every window is processed separately, without taking the actual overlap into account, some sets will be found multiple times.

For example, a set which is found in a sentence s_n will be found in each window that includes that sentence. If the sliding window stretches across three sentences, the exact same set would be found three times: once in each of the windows $[s_{n-2}, s_n]$, $[s_{n-1}, s_{n+1}]$, and $[s_n, s_{n+2}]$, as can be seen in Figure 3.1. It is undesirable to check if this set is a MNS each time the window slides, as this is computationally more expensive than necessary.

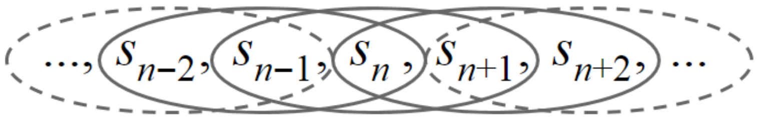


Figure 3.1: Sets found within one sentence could be found multiple times as the window slides.

3.1.3.3 Improvements

As mentioned in the previous section, using Guðjónsson's approach to find MNS leads to an unnecessary amount of computations. Every time the window slides, the new window covers a previously unseen sentence, s_n. In our approach, we first search for MNS using the same technique as Guðjónsson introduced, but we only look at sentence s_n. All new words, meaning MNS of size 1, are found during this step. Therefore, when we continue to look for MNS across sentences, we can focus on only searching for sets of size 2. Then, only previously seen words will be used to make candidate sets. For each sentence that precedes s_n within the current window $s_{n-x} : 1 \leq x <$ `window_size`, we search for new sets that contain one element belonging to the old words of s_n and one element belonging to the old words of s_{n-x}.

3.1.3.4 Filtering on Input or Output

We apply Guðjónssons method of filtering, meaning that the five words with the highest ranking tf-idf score within an article are considered important, and the MNS are filtered depending on if they contain one of the important words or not. Additionally, it is important to consider at which point filtering is applied. One way to filter is on input, where only the important words are used to construct sets. Another way is to filter on output. Filtering on output means that sets are constructed using the unfiltered text. Once filtering is applied, all sets that do not include one of the important words are discarded.

For example, consider an article where the word `crisis` is deemed important and the word `financial` is not. If filtering is performed on input, the word `financial` will never be considered to make sets, and the set `{crisis, financial}` could never be found. If filtering is performed on output however, `{crisis, financial}` could be created and since it contains an important word, the set will be kept post-filtering. Hence, filtering on input gives fewer sets, and the impact of using different filters could be important for our results. Guðjónsson [13] concludes that filtering on input gives too few sets, therefore we chose to filter on output.

3.1.4 Linking Articles

Initially, the MNS are found in the data set. For each article, we create a bag which consists of the union of all MNS found in the article. We select articles based on a *feature*. A feature can specify that the bag of an article must contain one or a set of words. We call the resulting selected set of articles a *chain*, which therefore can be seen as a set that contains all articles with a certain feature.

To clarify the concept feature, a feature can specify that the word w should be an element in an article's bag. Then, only the articles that satisfy this criteria will be selected. A feature can additionally specify that articles have to be found within a certain time span to be linked together. To include a time limitation does not affect which articles are selected, only how they are linked. If the time difference between two consecutive articles is not within the defined range, the chain of articles will be separated into two chains, as can be seen in Figure 3.2.

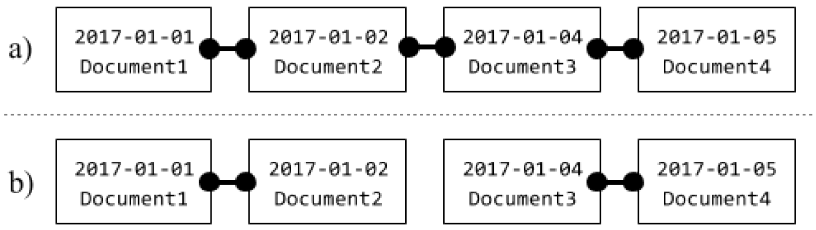


Figure 3.2: Four documents are linked in two different ways. In a), no time limitation is applied. In b) however, a time limit of less than two days is used, which means `Document2` will be separated from `Document3`, and as a result we get two chains instead of one.

3.2 Defining the Ground Truth

In this section we present the theory behind our ground truth and the methods we have used to implement it. The purpose of this model is to have something to compare our model to, which practically means to determine how many of the articles in a chain belong to the same topic.

We define our ground truth of a topic by clustering, where a cluster of articles represents a topic. Steinbach et al. [34] compare three techniques for text clustering: k-Means, Hierarchical Clustering, and Bisecting k-Means. They conclude that although Hierarchical Clustering often is portrayed as the best one regarding accuracy, they find that Bisecting k-Means performs as good or better than Hierarchical Clustering with lower running time.

However, many algorithms that are considered scalable and fast, such as Bisecting k-Means, require the number of clusters, k, to be defined in advance. This is problematic to specify for a data set consisting of articles since the number of topics that should be produced is usually not known in advance. Hierarchical Clustering Algorithms however, do not require k to be defined. Therefore we use a Hierarchical Clustering Algorithm implementation by SciPy [19].

3.2.1 Hierarchical Clustering

A Hierarchical Clustering Algorithm creates a hierarchy of clusters, producing a tree. Each leaf in the tree is a singleton cluster that contains a single document, whereas the root cluster contains all documents in the corpus. Therefore, clusters at different levels in the tree offer information of varying levels of abstraction. While clusters close to the root are more general, clusters closer to leaves are specific [37]. This raises a question of where to cut the tree, i.e. which clusters should represent topics. The result of clustering with a Hierarchical Clustering Algorithm is often visually represented using a dendrogram, as can be seen in Figure 3.3.

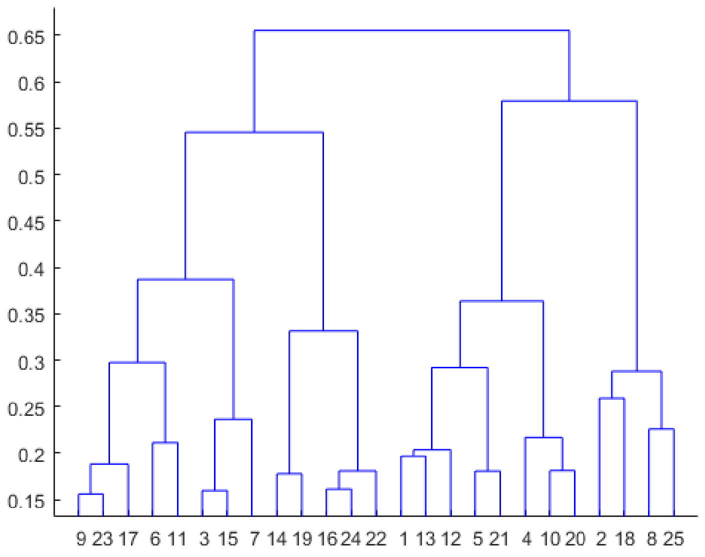


Figure 3.3: A dendrogram is a visual representation of a classification scheme, e.g. the result of hierarchical clustering. A full dendrogram would mean that each leaf in the graph is represented by an individual item, in our case an article. The leaves in a truncated dendrogram represent subclusters, i.e. groups of articles. This means that the labels on the x-axis in the figure show the number of items in each subcluster. The distances between clusters are shown on the y-axis [24].

A Hierarchical Clustering Algorithm is either agglomerative or divisive. In an agglomerative, i.e. bottom-up clustering algorithm, each document is initially a singleton cluster and the closest clusters are merged until one cluster, the root, remains. The opposite is done using a divisive, or top-down approach. Initially, all documents belong to the root cluster which is divided until each leaf is a singleton cluster. We choose bottom-up clustering because we search for small clusters and therefore expect less error propagation than a top-down approach would provide.

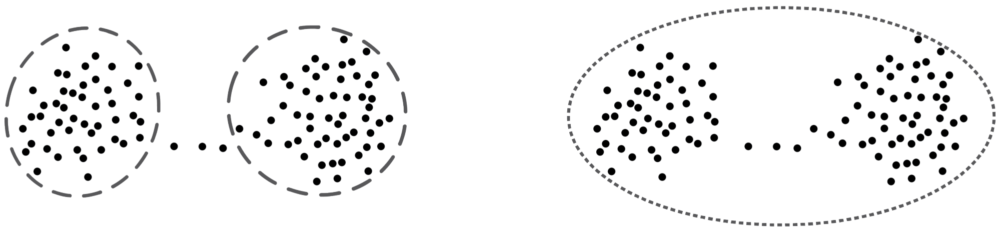
There are multiple ways to measure the distance between clusters, i.e. to determine the linkage criterion. We use two different linkage methods which are described in more detail in Section 3.2.1.1 and 3.2.1.2. We decide to use UPGMA since it is the most common method used within text clustering. Additionally, we decide to use single linkage because the result will likely differ from using UPGMA.

3.2.1.1 Single Linkage

Using single linkage, two clusters are merged based on the shortest distance between them. This means that the elements across the clusters are compared pairwise, and the shortest distance determines the linkage criterion. The distance between two clusters, C_i and C_j, using single linkage is defined as Equation 3.3.

$$D(C_i, C_j) = \min_{x \in C_i, y \in C_j} D(\overrightarrow{V_x}, \overrightarrow{V_y}) \tag{3.3}$$

One well-known problem with single linkage is the so called *single-link effect* or *chaining effect*. It means that if there is a chain of points between two clusters, the clusters might not be separated [1], see Figure 3.4. This also means that the distance between elements within a cluster can be large.



(a) Using UPGMA. **(b)** Using single linkage.

Figure 3.4: Two natural clusters connected by a chain of points and clustered using two different linkage methods. If single linkage is used, the two clusters may not be separated due to the chain of points between them.

3.2.1.2 Unweighted Pair Group Method with Arithmetic Mean

Unweighted Pair Group Method with Arithmetic Mean (UPGMA), also called average linkage, merges two clusters based on the average distance between all objects in the clusters. The distance between two clusters using UPGMA is given by

$$D(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{y \in C_j} D(\overrightarrow{V_x}, \overrightarrow{V_y}). \tag{3.4}$$

Since all objects are used to calculate the distance, and not only individual elements as in single-link, a chaining effect is avoided.

3.2.2 Making Documents Comparable

In order to cluster documents, they need to be comparable. A common approach is to represent each document in the *vector space model* [27, p. 45], which means that each document is represented as an n-dimensional vector. *Bag-of-words* is an approach where n is the number of unique words that exist within the corpus, and each component in a vector represents the number of occurrences of one specific word in that document [2, p. 167].

We explore three different ways to represent a document: using spaCy [14], CountVectorizer [28], and TfidfVectorizer [28]. The method that spaCy uses is a technique known as *word embedding* that represents each word by a 300-dimensional vector [26]. Each document is then also represented as a 300-dimensional vector, based on the document's composition and the vectors of the words it contains. While this technique is known to be very effective to estimate similarities between documents, we find that it is not possible to find specific topics if documents are represented this way. We believe that this can be due to word embeddings not functioning well within the TDT definition of an event, see Section 2.1. For example, the word vectors of Rome and Paris are very similar since both are capitals of European countries. However, there is a distinct difference between an event occurring in Rome and one occurring in Paris.

CountVectorizer uses a bag-of-words model [33] and TfidfVectorizer is a modified version of that. In both techniques, each component in a vector represents a word. While CountVectorizer uses the number of appearances for each word in a given document, TfidfVectorizer uses the tf-idf-value for the word. The later one shows best result and therefore, we choose to represent documents using TfidfVectorizer.

Once the documents are represented as vectors, the documents can be compared using a similarity measurement, which we expand on in the next section.

3.2.3 Measuring Similarity between Document Vectors

The similarity, or distance, is calculated pairwise between all document vectors. While a higher similarity value denotes that two documents are more similar, a higher distance value denotes that they are less similar. There exists a wide range of measurements that can be applied, where some of the best performing approaches within document clustering are cosine similarity and extended Jaccard [35].

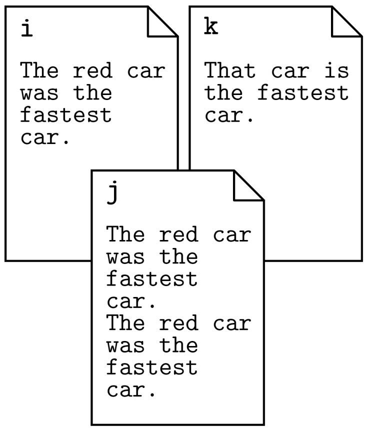
Extended Jaccard builds upon the Jaccard index, which is commonly used to compare sets. Cosine similarity is the most common approach within text clustering [2, p. 89]. Due to a lack of time we therefore decided to only continue with cosine similarity for our experiments and not to use extended Jaccard. However, we will use the basic version of the Jaccard index as a filter in one of our experiments, see Section 4.4.

We denote the vector that represents document d by $\overrightarrow{V_d}$. The cosine similarity between two documents i and j is measured as the cosine of the angle between their corresponding vectors, $\overrightarrow{V_i}$ and $\overrightarrow{V_j}$. It is calculated using the Equation 3.5 [15].

$$sim_C(\overrightarrow{V_i}, \overrightarrow{V_j}) = \frac{\overrightarrow{V_i} \bullet \overrightarrow{V_j}}{\|\overrightarrow{V_i}\|\|\overrightarrow{V_j}\|} \tag{3.5}$$

When comparing vectors that represent documents, it has been widely observed that the direction of the vectors is more important than their length [38]. Therefore, cosine similarity has become a popular metric within document clustering as the length of a document vector has no effect on the result. An example to visualize this is given in Figure 3.5.

In the figure, we consider three documents i, j, and, k, where j is the combination of two copies of document i. Although j has twice the length of i, these documents have the same composition since each word that appears in i will appear twice as often in j. Document k has a different composition, as it does not contain the word *red*. As can be seen in Figure 3.5c, the vectors i and j have the same direction, and the angle between them is 0°. The cosine similarity between i and j is 1 and the two documents are considered identical. The angle between k and either of the other vectors is $\frac{\sqrt{30}}{6} \approx 24°$, which means that the cosine similarity would be roughly 0.91.



```
i

The red car
was the
fastest
car.
```

```
k

That car is
the fastest
car.
```

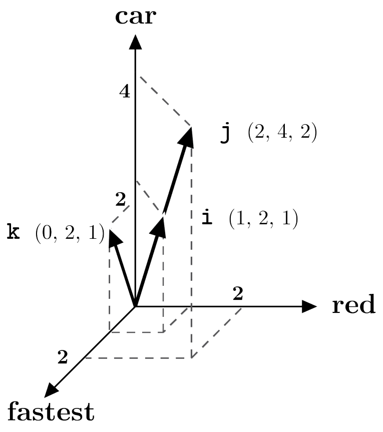
```
j

The red car
was the
fastest
car.
The red car
was the
fastest
car.
```

(a) The three documents i, j, and k.

	i	j	k
red	1	2	0
car	2	4	2
fastest	1	2	1

(b) Representing i, j, and k as vectors.



(c) Visualizing i, j, and k in a 3-dimensional space.

Figure 3.5: Three example documents represented as vectors in the vector space model.

The range of cosine similarity is [0,1], where 1 indicates that the vectors are pointing in the same direction and 0 means that they do not have any words in common.

4
Experiments

We define some features that we want to explore, and run these through our framework. The experiments that we run are described below.

4.1 Data Sets

We apply our methods to the data set Reuters Corpus Volume I (RCV1) [21]. It contains over 800,000 manually categorized news articles in English, from the years 1996 and 1997. The articles were made available by Reuters, Ltd. for research purposes. Every article is marked with an industry, region and category code (called topic code within the Reuters data set). This enables testing on a more narrow data set.

We choose three categories to use as data sets which are presented in Table 4.1, where N is the number of articles that we use for our experiments. These are chosen based on the possibility to find and track actual topics within them. For example, the category 'New products/services' proves to be a weak candidate as it does not seem to contain topics, but rather single articles about new releases.

Although RCV1 contains a year's worth of articles, we only perform our experiments on the articles from the the last six month of that year. We refer to the articles from the first six months as N_0, and they are used for preprocessing purposes to improve the quality of the tf-idf scores, as mentioned in Section 3.1.1. They are also used to find MNS so that the first articles that we use in our experiments do not have a disproportionately large amount of MNS.

Data set	**Category code**	**Category**	N	N_0
D_1	GDIS	Disasters and Accidents	4614	4040
D_2	GENV	Environment and Natural World	3467	2794
D_3	GCRIM	Crime, Law Enforcement	17064	15155

Table 4.1: The three data sets used for our experiments.

A large part of research within the field of text mining uses micro-text, which means to use social media to retrieve data, Twitter commonly being one of the main sources [20] [2] [27]. One advantage with using mainstream media as a source is the higher quality of text. There are fewer misspelled words, the grammar is better, and there are fewer additions such as emojis.

To find the number of clusters for each category, we manually examine different values for max distance (*max_d*) between clusters. It shows that we need to use different *max_d*-values for each combination of data set and linkage method, if we want each cluster in every data set to represent a specific topic. However, they are quite similar, see Table 4.2.

Data set	**Linkage method**	**Metric**	*max_d*	**Clusters**	**Biggest cluster**
D_1	UPGMA	Cosine	0.8	776	140
D_1	Single linkage	Cosine	0.4	2738	107
D_2	UPGMA	Cosine	0.75	1004	88
D_2	Single linkage	Cosine	0.4	2343	68
D_3	UPGMA	Cosine	0.75	3836	299
D_3	Single linkage	Cosine	0.4	8771	278

Table 4.2: Different max_d values for different combinations of data set, linkage method and metric.

4.2 Framework for Experiments

In order to conduct our experiments, we build the following framework: first we define features of the MNS that we want to explore, see Section 3.1.4. We then link articles based on a given feature. Lastly, we use the ground truth (see section 3.2) to check how many of the articles belong to the same topic. The flow of experiments in the framework are visualized in the Figure 4.1.

4.3 Tracking One Word

We start with the simplest tracking which is to link articles based on one word at a time. For each word w we link articles where w is an element in an article's bag. This creates one chain of articles for each word that is used for linking.

4.4 Tracking Two Words

The second experiment examines if two words frequently appearing together in bags is a good indication that they belong to the same topic. We start by removing all combinations which only occur once, because a vast amount of these combinations exist and they do not provide interesting data. This is further explained in Section 4.6.

As can be seen in Table 4.3, there exist millions of word-combinations that could be tracked for each data set. To determine how often a combination of two words appears, we calculate the Jaccard index of the occurrences of the two words. The

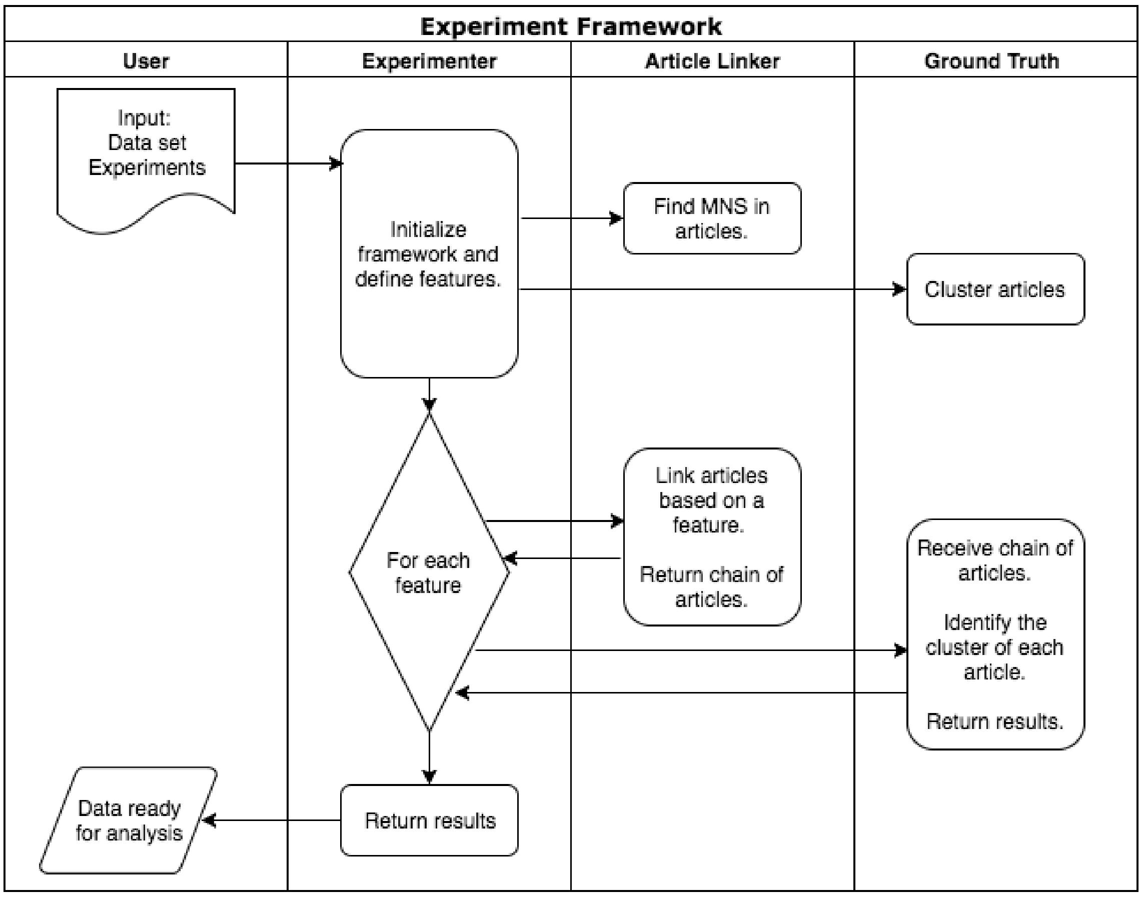

Experiment Framework
User
Experimenter
Article Linker
Ground Truth
Input: Data set Experiments
Initialize framework and define features.
Find MNS in articles.
Cluster articles
For each feature
Link articles based on a feature.
Return chain of articles.
Receive chain of articles.
Identify the cluster of each article.
Return results.
Return results
Data ready for analysis

Figure 4.1: A visualization of the framework we build to perform experiments. Based on the data set and the experiment to run, defined by the user, the Experimenter creates a set of Features and one by one sends these to the ArticleLinker to retrieve their corresponding chains. These chains are sent to the GroundTruth which returns the cluster affiliation for each article in a chain. The results of all features are saved for analysis.

Jaccard index, also known as the intersection over the union, is defined in Equation 4.1.

$$J(A, B) = \frac{A \cap B}{A \cup B} \tag{4.1}$$

In the equation, consider a combination of words (a, b). For our purposes, A is the set of articles where a exist, and B is the set of articles where b exist. The range of the Jaccard index is [0,1], where 1 means that the two sets have the same content, and 0 means that they do not have any elements in common.

The Jaccard index can also help us filter our word combinations, since it is unfeasible to use all the millions of combination to link articles on. To be able to analyze how the Jaccard index impacts the results, we want to choose a low limit. Originally, we set our lower bound to a Jaccard index greater than 0.05, but this results in too much data to process as one data set had 45000 combinations on average. When we explore to set the lower bound to 0.1 instead, we find that this allows us to filter away uninteresting combinations that appear rarely, while giving us a feasible amount of data to process.

Data set	**Unique words**	**Words that occur more than once**	**Combinations found**	**Combinations existing more than once**	**Combinations with Jaccard index > 0.1**
D_1	17 479	10 700	2 567 821	446 165	11 176
D_2	17 018	10 526	3 376 467	666 860	11 409
D_3	44 027	25 540	8 357 595	1 888 910	18 341

Table 4.3: An overview of how many unique words and how many combinations exist within the article bags of each data set, and how these amounts decrease with filtering.

4.5 Adding a Time Feature

We believe that a word or a combination of words could be important for different topics at different points in time. Therefore, we extend the previous experiments by adding a time feature. With the time feature we study if limiting the allowed time interval between articles could provide better results. We analyze the published dates of articles, both as a whole for each data set, but also within the cluster of each topic. An observation from our Reuters data set is that the article count drops noticeably over the weekend. If the articles within a topic span over more than a week, but have a brief pause over the weekend for example, it does not mean that it is a new wave or that it is a new topic. An important insight from this observation is that the time interval can't be too strict. We define two articles in a chain that are published consecutively within the allowed time interval to appear within *close succession.*

To specify close succession we analyze the time delta between consecutive articles in each ground truth topic. Therefore, we look at the distribution of time deltas for a data set and let the 50th percentile to the 90th percentile in 10 percentage point intervals define close succession. See Table 4.4.

Data set/Percentile	**50**	**60**	**70**	**80**	**90**
D_1	4	8	15	27	51
D_2	7	12	21	34	59
D_3	6	10	17	28	53

Table 4.4: Different percentiles for the distribution of time deltas using UPGMA. The percentiles are shown in days, where for example the 50th percentile, i.e. the median, of D_1 is 4 days.

The chains are initially composed exactly like in the experiments above, but then broken apart into shorter chains depending on the defined close succession. We let a chain of articles be sorted in a chronological order. If two succeeding articles in a chain have a bigger time gap than a defined close succession, the chain is split into two chains. Since trends span over different time periods it is difficult to decide a fixed value for close succession.

4.6 Evaluation of Results

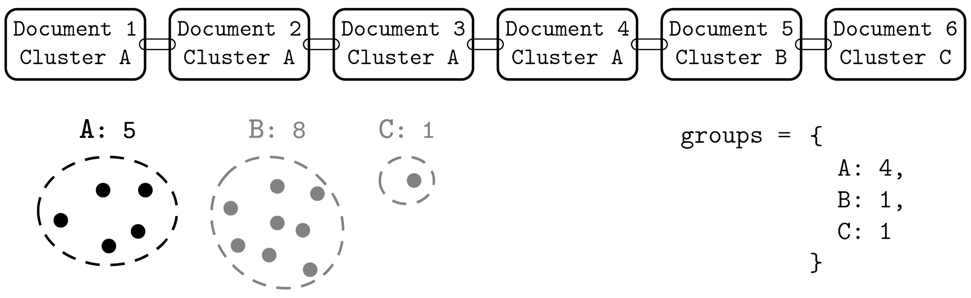
The chains are compared to the ground truth to see how many articles belong to the same cluster. We explore several ways to determine the success of one chain when comparing against the ground truth. A common way to measure similarity between sets is the Jaccard index, defined as Equation 4.1. We considered using this to define the success rate of a chain, where a chain would be compared against each cluster to which articles in the chain belong. The highest Jaccard index would be chosen as the success rate. For example, a chain with articles that belong to cluster Cl_1 and Cl_2 would be compared against both clusters individually. The highest Jaccard index between the two would define the success rate.

However, the Jaccard index is not ideal for our purposes because the cluster sizes affect the results too much. Even if all articles in a chain belong to the same cluster, the success rate could still be low if the cluster size is large. Additionally, let us consider a chain where all except one of the articles belong to the same cluster. If the one article that belongs to a separate cluster represent a singleton-cluster, the success rate would be $1/\ell$. Then, if the other cluster is larger than $(l-1)\cdot\ell$, the success will be determined by the singleton cluster. This would mean that the success of the chain was determined by noise, which we do not want.

Therefore, we choose to define another measurement. Equation 4.2 contains the names of variables which we refer to in this section. The values of these variables are defined for one given chain. The length of the chain is denoted by ℓ. Let the articles in the chain be grouped by which cluster they belong to. Then a *group* of a chain is defined as all articles that belong to the same cluster. The number of groups, which is the number of unique clusters that the articles in the chain belong to, is denoted by g. The size of the largest group is denoted by lgs, and the size of the cluster that its articles belong to is denoted by cl. Figure 4.2 displays an example of this.

$$\boxed{\begin{aligned}\ell &= \text{chain length}\\ g &= \text{number of unique clusters}\\ \text{lgs} &= \text{size of largest group of articles}\\ \text{cl} &= \text{cluster size}\end{aligned}} \tag{4.2}$$

We refer to our first try of a measurement as m_1. Let the articles in a chain be grouped by which cluster they belong to, then each group of a chain is defined by all the articles that belong to the same cluster. The value of m_1 is defined by the largest *group* of a chain and the chain length, see Equation 4.3. Consider a chain



$$\boxed{\ell = 6, \quad lgs = 4, \quad g = 3, \quad cl = 5}$$

Figure 4.2: An example displaying the values that the variables get for one chain. The chain has a length of 6. The largest cluster group is cluster A with 4 articles belonging to it. There are 3 unique cluster that the articles belong to. Cluster A has the size 5.

with five articles, where four of them belong to the same cluster. The largest group of the chain consists of those four articles and the score of that chain is 4/5.

$$m_1 = \frac{\text{lgs}}{\ell} \tag{4.3}$$

However, this approach has some flaws. As can be seen in Figure 4.3 there seems to be a relation between chain length and m_1, where shorter chains generally give better results. This is not surprising since shorter chains have a higher probability of getting a 100% success. Consider drawing random samples of articles to define a chain. Now consider the probability that all articles in the chain are in the same cluster. A chain with only one article in it would automatically have a success rate of 100%, and the probability for a high success rate decreases with the length of the chain. To create a better measurement, we identify two more factors that we find important.

One of the important factors is a weight for chain length, which punishes shorter chains. An observation we made is that the punishment should not be linear. It is common practice to use the logarithm to reduce the impact of high values of a variable. Therefore we define this measurement factor, m_2, as Equation 4.4.

$$m_2 = \ln(\ell) \tag{4.4}$$

Another identified factor, m_3, is the number of unique clusters, g, to which the articles in a chain belong. We consider m_3 to be max if all articles in a chain belong to the same cluster, i.e. if $g = 1$. Furthermore we consider m_3 to be 0 if no articles in a chain belong to the same cluster, i.e when g is equal to the length of the chain. Just as the punishment for chain length, we believe that this measurement should be non-linear. The measurement factor for number of groups is defined in Equation 4.5.

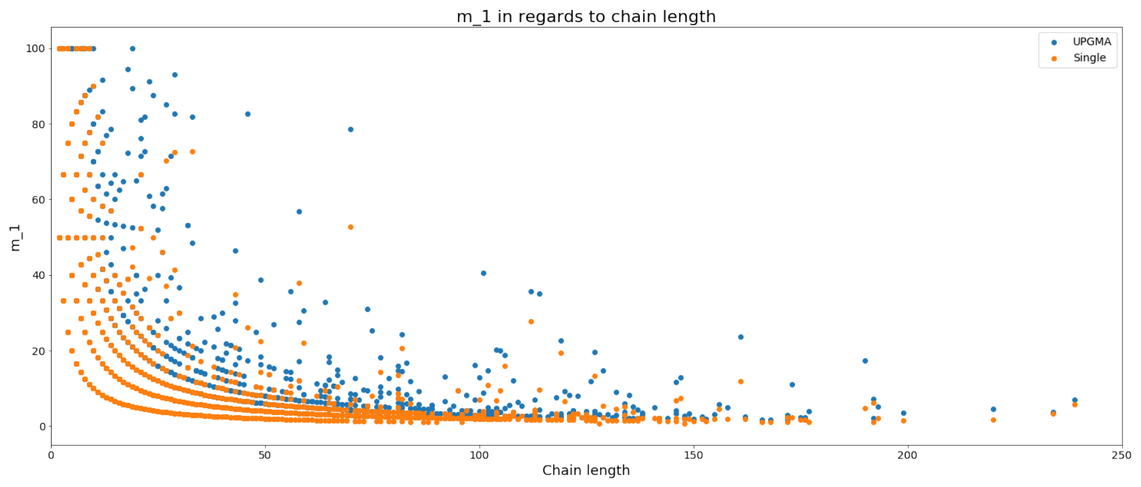


Figure 4.3: The chain lengths of D_2 plotted against m_1.

$$m_3 = \ln\left(\frac{\ell}{g}\right) \tag{4.5}$$

When selecting a random article, the probability of selecting one from a large cluster is greater than selecting one from a small cluster. Therefore, we compensate for cluster size. We choose to compensate according to the cluster which the largest group of articles belongs to, and call this factor m_4. The measurement factor for cluster size is defined in Equation 4.6.

$$m_4 = \text{cl} \tag{4.6}$$

The total measurement, m_{tot}, is defined as a product of the factors, m_1, m_2, m_3 divided by m_4. The formula is defined in Equation 4.7. A suitable property for this measurement is that the success rate of one chain fulfills the behaviour to be zero if no articles in a chain share the same cluster.

$$m_{tot} = \frac{m_1 \cdot m_2 \cdot m_3}{m_4} = \frac{\frac{\text{lgs}}{\ell} \cdot \ln(\ell) \cdot \ln\left(\frac{\ell}{g}\right)}{\text{cl}} \tag{4.7}$$

After evaluating this measurement, we find that the factor m_4 has a much larger impact on the resulting measurement, m_{tot} than the other factors. This is due to m_4 having a much larger range than the other factors. Intuitively, the outcome of the measurement should be at its maximum if the cluster size is as small as possible. Additionally, all articles in a chain need to belong to the same cluster. For this to be possible, the cluster size must be larger than or equal to the chain length. These two requirements imply that cluster size must be equal to the chain length for the final measurement to be maximum. The maximum of our current measurement can be seen in Equation 4.11, where we see that the measurement will actually decrease for larger chains, which is the opposite of what we want.

$$0 \leq m_1 \leq 1 \tag{4.8}$$
$$1 \leq g \leq \ell \implies 0 \leq m_3 \leq \ln(\ell) \tag{4.9}$$
$$\text{if}\,(m_1 = 1 \iff g = 1) \implies \text{cl} \geq \ell \tag{4.10}$$

$$m_{tot} \leq \frac{m_1[\text{lgs} = \ell] \cdot m_2 \cdot m_3[g = 1]}{m_4[\text{cl} = \ell]} = \frac{1 \cdot \ln(\ell) \cdot \ln(\ell)}{\ell} = \frac{\ln^2(\ell)}{\ell} \tag{4.11}$$

Therefore, we define m'_4 as the logarithm of cluster size instead, so that the different factors impact the total measurement with similar weights, see Equation 4.12. But to make sure that we do not divide by zero, we define $m_{tot} = 0$ if $\text{cl} = 1$. The new maximum is therefore $ln(\ell)$, see Equation 4.13.

$$m'_4 = \ln(\text{cl}) \tag{4.12}$$

$$m_{tot} \leq \frac{m_1[\text{lgs} = \ell] \cdot m_2 \cdot m_3[g = 1]}{m'_4[\text{cl} = \ell]} = \frac{1 \cdot \ln(\ell) \cdot \ln(\ell)}{\ln(\ell)} = \ln(\ell) \tag{4.13}$$

Lastly we normalize the final measurement with the maximum value. This gives us a measurement which is bounded to [0,1], see Equation 4.16.

$$\text{if cl} = 1 : m_{tot} = 0$$
$$otherwise : m_{tot} = \frac{m_1 \cdot m_2 \cdot m_3}{m'_4 \cdot \max(m_{tot})} = \frac{\frac{\text{lgs}}{\ell} \cdot \ln(\ell) \cdot \ln\left(\frac{\ell}{g}\right)}{\ln(\text{cl}) \cdot \ln(\ell)} = \frac{\frac{\text{lgs}}{\ell} \cdot \ln\left(\frac{\ell}{g}\right)}{\ln(\text{cl})} \tag{4.14}$$
$$m_{tot} \geq \frac{m_1[\text{lgs} = 1] \cdot m_2 \cdot m_3[p = \ell]}{m'_4} = \frac{\frac{1}{\ell} \cdot \ln(1)}{\ln(\text{cl})} = 0 \tag{4.15}$$
$$m_{tot} \leq \frac{m_1[\text{lgs} = \ell] \cdot m_2 \cdot m_3[g = 1]}{m'_4[\text{cl} = \ell]} = \frac{1 \cdot \ln(\ell)}{\ln(\ell)} = 1 \tag{4.16}$$

4.6.1 Comparing Results

To be able to compare our results we simulate random draws, which means that we create chains by randomly selecting articles. If the results from an experiment are not significantly better than the results from the random model, the experiment is considered unsuccessful.

We begin by calculating a simple empirical probability mass function for chain length, which is based on the actual results of one experiment and one data set. Based on that probability mass function we follow the steps below. Let M be the number of chains to sample. We choose M to be the same number of chains as generated from the experiment.

While one chain should not be able to contain a specific article more than once, an article should be able to appear in multiple chains. Each chain is created independently of chains that have previously been created.

```
1: for $M$ times with replacement do
2:     Generate random number to be chain length, $\ell_r$, from the probability mass function
3:     Randomly draw $\ell_r$ articles from the entire data set without replacement
4: end for
```

The random model is used to calculate a z-score [7], which is a measure of how many standard deviations away from the mean a value is. The z-score of a result x_i is calculated as Equation 4.17, where μ_{rand} and σ_{rand} is the mean and standard deviation of our random model. A negative z-score therefore means that the result is below the mean of our random model, and a positive high z-score means that our model performs much better than the random model.

$$z_i = \frac{x_i - \mu_{rand}}{\sigma_{rand}} \tag{4.17}$$

We also calculate a *p-value* [30] for each chain with a hypergeometric test, to be able to measure the significance of our results. The hypergeometric test is also known as the one-tailed version of Fisher's exact test [25] [31], which uses the hypergeometric distribution to help determine whether an observation is statistically significant or not. In a test for over-representation this means to calculate the probability of drawing k *or more* successes from the data set under the *null hypothesis*: the hypothesis that there is nothing special about the data set. In our case, we compute the probability of drawing k or more articles from a cluster with size K randomly, where the length of the chain is ℓ, and the total number of articles in the data set is N. If this probability, also called p-value, is sufficiently low, we can reject the null hypothesis and say that the result is significant. To decide if the result is significant, the p-value 0.05 is commonly used in statistics as a limit. This means that if the p-value is less than 5%, the result is counted as statistically significant.

The probability mass function of a hypergeometric distribution [30], i.e. the probability of exactly k successes in n draws, is given by Equation 4.18.

$$p_X(k) = P(X = k) = \frac{\binom{K}{k}\binom{N-K}{\ell-k}}{\binom{N}{\ell}} \tag{4.18}$$

4.7 Environment

We perform our experiments on a laptop with the following specifications:

- Processor: Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz 2.30 GHz.
- Installed RAM: 12,0 GB
- System type: 64-bit operating system, x64-based processor

5 Results

This chapter presents the results of each experiment described in Chapter 4.

Table 5.1 describes, for each data set, the lengths of the chains generated by our methods together with a description of the tf-idf-values. We list the minimum and maximum chain length along with the mean and standard deviation. As can be seen the minimum chain length is always 2, which is not surprising since we filter out chains of smaller length. The maximum chain length and other values vary depending on the data set. Both chain length and tf-idf-values are used in the analysis.

Experiment: Tracking one word

Data set	**Measure**	**Chain length**	**Average tf-idf**
D_1	mean	12.66	0.087
	std	27.38	0.055
	min	2	0.016
	max	1523	0.643
D_2	mean	13.53	0.070
	std	25.80	0.056
	min	2	0.011
	max	1337	0.452
D_3	mean	20.60	0.097
	std	59.59	0.075
	min	2	0.012
	max	4019	1.158

Table 5.1: Description of chain length and average tf-idf-value for all data sets when tracking one word. Since a word has different tf-idf-values for different documents, we calculated an average tf-idf for each word.

In the experiment of tracking two words, we list the chain lengths and the Jaccard index. Once again, as can be seen in Table 5.2, the minimum chain length is 2, and the maximum chain length and other values vary between the data sets. The values, however, differ substantially from Table 5.1, where the results of tracking one word are presented. When tracking two words, we include the Jaccard index, see Section 4.4, since we want to analyze if combinations that appear more often together achieve better results.

Experiment: Tracking two words

Data set	**Measure**	**Chain length**	**Jaccard index**
D_1	mean	2.68	0.20
	std	3.77	0.15
	min	2	0.22
	max	210	1.00
D_2	mean	2.64	0.20
	std	2.62	0.14
	min	2	0.14
	max	69	1.00
D_3	mean	3.00	0.26
	std	5.47	0.20
	min	2	0.10
	max	192	1.00

Table 5.2: Description of chain length and Jaccard index for all data sets when tracking two words.

In order to further evaluate the results from our experiments, we compare the MNS Chaining model to a random model. A new random model is created for each combination of data set, linkage method and tracking on one or two words, with or without time feature, see Section 4.6.1.

Data set	Linkage	Experiment	m_{tot}^{avg}	z^{avg}-score	p^{avg}-value
D_1	UPGMA	One word	0.070	6.11	0.0041
D_1	Single	One word	0.049	9.10	0.0029
D_1	UPGMA	Two words	0.196	13.93	0.0015
D_1	Single	Two words	0.176	20.69	0.0047
D_1	UPGMA	Time feature	0.156	12.97	0.00070
D_2	UPGMA	One word	0.056	6.52	0.0046
D_2	Single	One word	0.030	3.85	0.0043
D_2	UPGMA	Two words	0.195	10.42	0.0012
D_2	Single	Two words	0.131	8.07	0.0071
D_2	UPGMA	Time feature	0.141	15.34	0.00059
D_3	UPGMA	One word	0.099	26.07	0.0031
D_3	Single	One word	0.077	30.11	0.0054
D_3	UPGMA	Two words	0.289	40.53	0.00016
D_3	Single	Two words	0.254	48.78	0.00017
D_3	UPGMA	Time feature	0.208	34.38	0.000081

Table 5.3: Our experiments generate values m_{tot}^{avg}. The z-scores indicate the relation of our results compared to the random model, for details see Section 4.6.1. The p-values indicate the significance of our result, see Section 4.6.1.

In Table 5.3 we present the mean success rate m_{tot}^{avg} for various experiments along with corresponding mean z-score, z^{avg}-score, and mean p-value, p^{avg}-value. Since m_{tot} is defined for one chain, we need to define a measure for an entire experiment. Let c denote a chain and let C be all chains in the experiment, then we define m_{tot}^{avg} as follows:

$$m_{tot}^{avg} = \frac{\sum_{c \in C} m_{tot}}{|C|} \tag{5.1}$$

Similarly, z-scores and p-values are defined for each chain. Most z-scores are within the interval $[0, 7]$, but a few can be up to almost 200. For details see Appendix B.1. We want to evaluate our results from an experiment as a whole, and therefore compare our results against a random model by calculating the z-score for each experiment.

To get a z-score for a whole experiment, we define z^{avg}-score as the sum of z-scores over all chains divided by the number of chains. A positive z-score denotes that our model performs better than the random model. For example, for data set D_1, tracking one word and single linkage, the average z-score is 9.1 which means that our result on average is roughly 9 standard deviations away from the mean of the random model.

The p-values indicate the significance of the results from each experiment. P-values are within the interval $[0, 1]$ and low values indicate high significance. A common threshold to say that a result is statistically significant is 0.05.

Note, that since the z^{avg}-scores and p^{avg}-values are calculated from each individual experiment, and thus an individual random model is created for each experiment, the rows cannot be compared in the columns of Table 5.3, see Chapter 6.

We only test the time feature in an experiment where we track two words and use UPGMA as linkage method. We test different percentiles for the distribution of time deltas to define close succession, see Table 4.4. The 90th percentile experiment (between 51 and 59 days) gives the best results, for all data sets 5.4. The numbers in Table 5.3 correspond to the 90th percentile experiment.

5.1 Measurable Properties of the Tracking Words and Their Impact

In this subsection, we aim to display different indicators for our experiments in order to prepare for comparison and analysis of our results. Note that we only present a selection of the plots here and that the corresponding plots for the all data sets can be found in Appendix B.

Figure 5.1 shows the z-scores of D_1 from tracking one word, in regard to each word's tf-idf-value. From the graph, it is difficult to see the individual points and therefore determine which data points are noise. Therefore, we divided the data points into intervals based on their tf-idf-value, and plotted each as a box plot, see Figure 5.2.

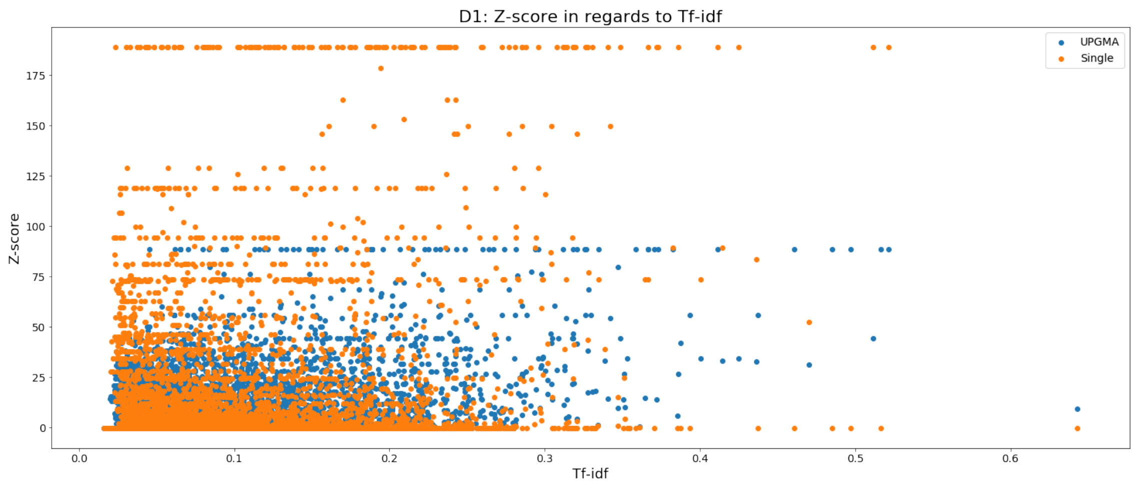


Figure 5.1: The average tf-idf of all words in MNS from D_1 plotted against their z-score.

Box plots show the spread of the data, where the box represent the first and the third quantile, i.e. 50% of the data points are within the box. The whiskers of box plots can represent several alternatives, where one alternative is to represent min and max. The whiskers in our box plots is 1.5% from min and max respectively and the individual dots outside the whiskers represent outliers.

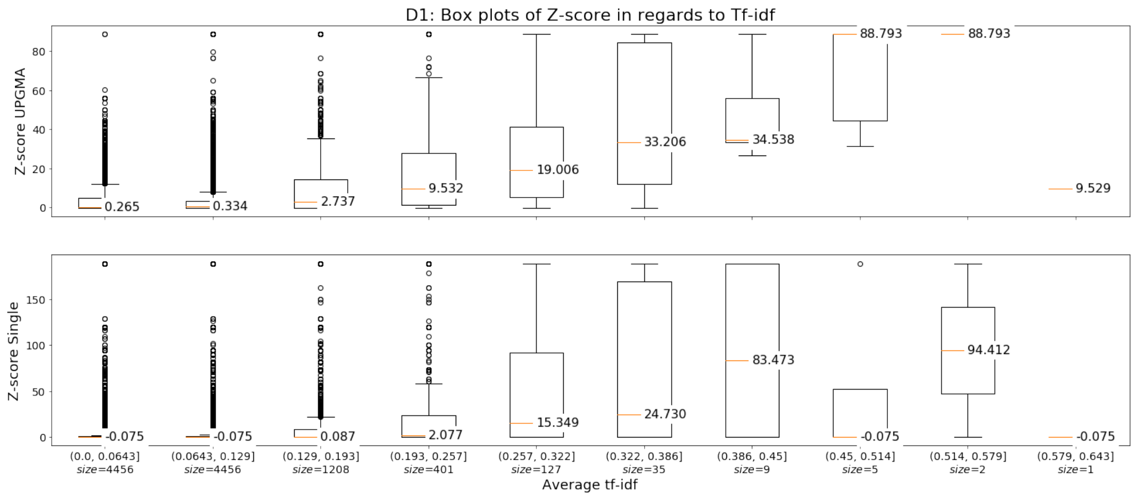


Figure 5.2: Box plots of different intervals for the average tf-idf of all words in MNS from D_1 plotted against their z-score.

Figure 5.3 shows the z-scores of D_2 from tracking two words, in regards to the Jaccard index of the word combination. Similar to the scatter plot for tf-idf, Figure 5.1, it is difficult to tell one point from another in some areas of this graph. We once again divide the data points into intervals based on the Jaccard index and present the data using box plots, as can be seen in Figure 5.4.

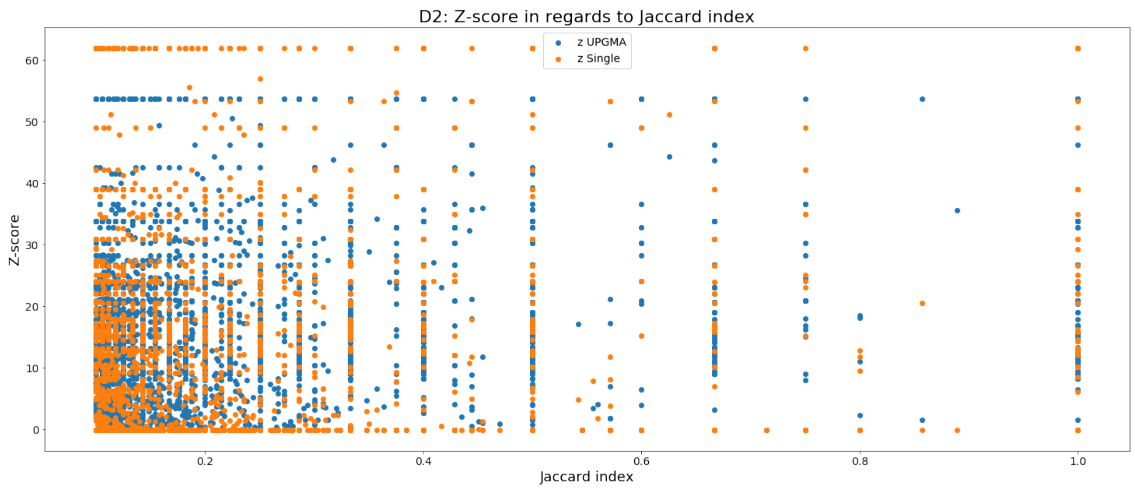


Figure 5.3: The Jaccard index of two words plotted against the z-score. Data set: D_2.

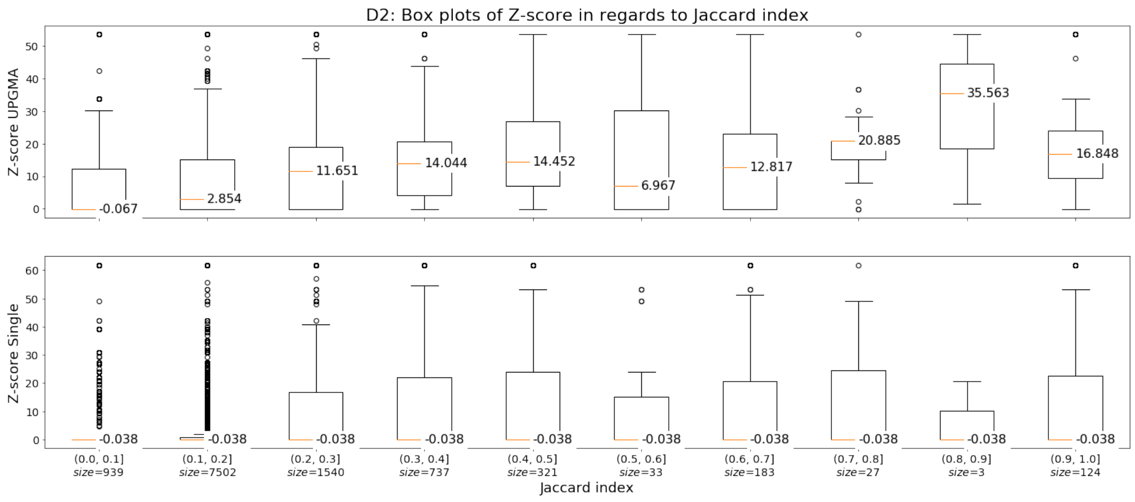


Figure 5.4: Box plots of different intervals of the Jaccard index of two words plotted against the z-score on data set D_2.

5.2 Time Feature

Since tracking two words with UPGMA gives the best average success in our other experiments, as can be seen in Table 5.3, and because of a shortage of time, we decide to add the time feature only to tracking two words and to analyze against the UPGMA-ground truth.

We also analyze if breaking the chains into subchains improves the MNS Chaining model. Consider a chain c where the majority of the articles belong to cluster A. We will simplify the notion of this by saying that the chain c belongs to cluster A. Suppose that c is split by the time feature into two subchains c_1 and c_2. If both subchains still belong to cluster A, dividing the chain was unnecessary. However, if c_1 and c_2 now belong to two different cluster, dividing does make sense.

Data set		**Percentile**				
		50	60	70	80	90
D_1	Mean m_{tot}	0.087	0.103	0.116	0.135	0.157
D_2	Mean m_{tot}	0.063	0.074	0.090	0.110	0.141
D_3	Mean m_{tot}	0.085	0.105	0.130	0.164	0.209

Table 5.4: The mean success of tracking two words with different time deltas. The values for each percentile can be seen in Table 4.4.

We therefore define a successful split as a split where the resulting subchains belong to different clusters. In Equation 5.2 we define a total score for one chain, where the score for one chain is determined by the number of successful splits divided by the number of total splits. The range of this score is [0,1] where 0 means that there where no successful splits, and 1 means that all splits where successful. Table 5.5 represent the mean scores for all chains within the experiment.

$$\text{Successful split rate} = \frac{\text{Successful splits}}{\text{Number of splits}} \tag{5.2}$$

Data set	**Average successful splits per chain**	**Average number of splits per chain**	**Successful split rate**
D_1	0.80	1.02	0.78
D_2	0.81	1.02	0.80
D_3	0.46	1.03	0.45

Table 5.5: All columns represent the mean of each parameter and data set, where the rightmost column is the fraction of the second and third column. The rightmost column tells us the proportion of successful splits.

6
Discussion and Conclusion

In this chapter, we start off by comparing our initial goals and expectations with the outcome of our experiments. We continue with presenting our main challenges and suggest possible future work.

6.1 Analysis of Our Results

In following subsections we discuss the magnitude of the z-scores and p-values in Table 5.3. We also discuss the impact of tf-idf-value, Jaccard index and time feature on the success rate.

6.1.1 Magnitude of z-scores and p-values

A z-score is a measure of how many standard deviations away from the mean of the random model that a result is. In Table 5.3, the z-scores are relatively high compared to z-scores in other experiments in the literature. This is because the standard deviations of our random models are very close to zero, which automatically causes larger z-scores, see Equation 4.17.

The standard deviations of the random model are close to zero due to how our measurement works. A chain's score, m_{tot}, is zero if all articles in the chain belong to different clusters, i.e. if the number of unique clusters, g, is equal to the chain length ℓ. We draw the chain lengths for the random model from the probability mass function of the chain lengths from our experiments. The chains from our experiments are typically very short, see Table 5.1 and Table 5.2. Since it is more likely that all articles in one chain belong to different clusters if the length of the chain is small there are many chains in the random model with success rate zero.

Since Table 5.3 shows that the p-value for all experiments are below 1% we determine the results as significant. We think that this can be explained by the size of N, the number of articles in the data set, and how these are clustered in our ground truths. Since we have so many articles to choose from, from a large amount of clusters, it is not very likely to choose two or more articles from the same cluster by random chance.

6.1.2 Impact of the tf-idf-value on the Success Rate

We expected that words with higher tf-idf, i.e. more important words, would provide better results than common words. However, in Figure 5.1 it initially seems like words with a low tf-idf-value have a higher chance of succeeding than words with a high tf-idf-value. In the box plots in Figure 5.2 the data is presented in intervals, which facilitates the evaluation of the results. Since the last three box plots all include significantly less data points than the previous box plots, we account these plots as noise. Upon evaluation of the remaining box plots, we realize that the median of the success increase for each tf-idf-interval. Therefore, we conclude that our hypothesis is confirmed.

6.1.3 Impact of the Jaccard Index on the Success Rate

Since we believe that one word is not enough to capture a topic, we expected to achieve significantly better results for tracking a combination of words than tracking only one word. This hypothesis can be confirmed if comparing different rows in the column m_{tot}^{avg} in Table 5.3.

We also expected that word combinations with a higher Jaccard index would achieve higher success than those with a lower Jaccard index. We reason that the Jaccard index indicates how strongly two words relate to each other, and if this value is high they seldom appear in the absence of the other word. As explained in Section 5.1, the scatter plots of the Jaccard index were difficult to evaluate due to the high number of data points in each plot. Therefore, we divide the data into intervals depending on their Jaccard index, and evaluate the results with a box plot for each interval instead. Both the scatter plots and the box plots can be found for all data sets in Appendix B.3.

The box plots generally show that the median z-score increases when the Jaccard index increases, this is especially true for the box plots of UPGMA. It is more difficult to make a general assessment of the box plots for Single linkage. While the median is 0 for the first few boxes for D_1 and then increases, the median for D_2 is 0 for all boxes. The same graph for D_3 however, shows that the median increases steadily, more similar to the box plots of UPGMA.

We can see that for some of the intervals, the box plots contradict our hypothesis. In all box plots, the last interval which represents the Jaccard index (0.9, 1], has a median lower than the previous box. We believe that this is due to some word combinations that naturally are probable to occur together, without describing a topic. Examples of this that we have seen are

- An abbreviation along with the full form of the word or phrase, such as `[iiec, the international institute for energy conservation]`
- A name of a person or a location that are important to multiple topics, such as `[helmut, kohl]`
- Sayings, such as `[wreak, havoc]`
- Words that naturally appear together, such as `[divorce, marriage]`

We can also see that there exist other intervals that contradict our hypothesis. However, in the majority of these intervals, the number of data points are relatively small, and therefore these intervals cannot be given the same importance as the other intervals. Our final conclusions regarding the Jaccard index is therefore that our hypothesis is true.

6.1.4 Impact of the Time Feature

Our expectations of adding a time feature was that words in closer succession would give better success, i.e. smaller time deltas would give better results. We expected this since we believe that the same word can be important for different topics at different times. Table 5.4 shows the opposite. In Table 5.3, we can even see that adding a time feature gets worse mean success, m_{tot}^{avg}, than the other experiments. We believe this is due to lower time delta generating smaller chains and therefore it is more likely that a chain gets success rate zero.

If we go beyond our measurement when we analyze the results, we realize something interesting. We looked at how chains were broken up, and if the separation into sub chains made any difference. If a chain is broken into sub chains and all sub chains still belong to the same cluster, breaking the chain apart can be seen as unnecessary work. The result shown in Table 5.5 show that 78-80% of the splits for D_1 and D_2 were successful, but for D_3 the same rate is only 45%. This means that for D_3 more than half of the splits where unnecessary. We think that this indicates that a time feature could improve the model, but it would be necessary to do this experiment on more data sets to be able to draw a conclusion.

6.2 Main Challenges

In these subsections we present the main challenges we encountered during the project.

6.2.1 Defining our Ground Truth

The main challenge we had while defining our ground truth was to determine if the clustering provided us with a suitable model. Since there does not exist a way to automatically decide whether the clustering gives the desired granularity, we had to manually evaluate the results when deciding a maximum distance allowed between clusters, see Chapter 3.2.1. Although we tested multiple distances and manually checked several clusters per distance, we still cannot be entirely sure that our clusters do not contain noise.

In Appendix A, we display the resulting top ten chains from tracking two words to manually analyze the result. For each article in a chain, the title is printed along with the publication date and corresponding cluster id. When we look at the chains generated from D_3, it seems like the articles in one chain belong to an actual topic, judging by the titles. When we do the same comparison for D_1 and D_2 however, we see that the majority of the chains do not really capture topics, even though all articles in each top chain belong to the same cluster.

We consider that this could be due to different kinds of words being important for different data sets. Words that are important for distinction need to be mentioned frequently for the clustering to work properly. Most top features in D_3 contain a name which intuitively seems important for a data set about criminality. For the other data sets, which are about disasters and the environment, a word representing a geographical place is probably essential when tracking on words, but it is unlikely that the location is mentioned repeatedly. This might be the reason why our model performs better on D_3 than on D_1 and D_2. This could have different causes: it could simply be that the ground truth is lacking (at least in the cases for D_1 and D_2), but it could also be that the top chains, by bad luck, happen to represent the worst clusters of the ground truth.

6.2.2 Relevant Chain Lengths

One important aspect we have considered is how small a chain can be to still be interesting for tracking. We definitely know that a chain of just one article is not enough since we think that a topic cannot be defined by one article. But are just two or three articles enough? In our experiments, we get significantly more chains of a small chain length, which means that the higher we set the minimum limit for chain length, the less data we have for analysis. Since we did not want to limit our data too much we set the minimum limit to greater than 1.

6.2.3 Evaluating the Result

Overall a big challenge for us has been to define how we measure success of a chain and of an experiment since our model is very complex. Even though we believe that we succeeded to identify the most important factors of the measurement, it was challenging to know how the different factors relate, and how this could be evaluated further.

For a chain to get the best score possible with our measurement, the chain length, ℓ, should be equal to the cluster size, cl, see Section 4.6. When studying the chain examples in Appendix A we see that all of our top ranking chains are an exact match for a cluster. As we mention in 6.2.1, many of these clusters do not capture topics. Additionally, since many of the chains are very short, this could be an indication that the cluster size still has too much impact on the success rate of a chain.

When we reason about the factors in our final measurement, we also discuss what we expect from the chains and how our measurement reflects that. Consider a few articles that roughly have the same content and that are processed for MNS sequentially. It is then likely that a lot more MNS are found in the article that is processed first in comparison to the one that is processed last. Since we link articles based on MNS it is therefore more likely to link the first few articles within a topic than the last, even if they have a similar composition. Therefore, it could be possible that the chains that we create only captures a subset of a topic. If it seems unreasonable to expect a chain to be an exact match against a cluster, that could further motivate dampening the effect of the cluster size.

The z-scores in Table 5.3 seem to indicate that better results are achieved when single linkage is used instead of UPGMA. However, it is important to note that different rows in the table for z-score and p-value cannot be compared. Since the random model builds on a probability mass function derived from a specific experiment, data set, and linkage method, the z-score and p-value for each experiment will be calculated from a unique random model. The reason for the higher z-scores for single linkage can however be derived from the ground truth clustering, see next paragraph.

As can be seen in Table 4.2, the number of clusters created by single linkage is at least double to the amount of clusters created by UPGMA for all data sets. If two articles are randomly drawn it is therefore more likely to draw from the same cluster if UPGMA is used. This means that the random models for single linkage are very likely to perform much worse than the random models for UPGMA. The mean and standard deviation for single linkage will have smaller values. For this reason the z-scores can be higher when using single linkage compared to UPGMA, even though UPGMA has a higher mean success.

6.3 Future Work

One identified problem, that also Guðjónsson [13] mentions in his thesis, is that the algorithm that finds minimal new sets is not capable of forgetting over time. For example, consider there being an earthquake in California in both 2016 and 2017. Then one wishes the method to identify this as two different events. The terminology used in articles referring to the two events would most likely still have a large overlap. Therefore, it is reasonable to expect that a significantly fewer amount of MNS would be found in the articles referring to the later event since they already appeared in the articles of the first event. Implementing some form of forgetfulness would decrease storage space, improve speed of determining if a candidate set is new or not, and allow finding MNS that were found a long time ago.

We have limited our research to handle articles in one language, namely English. Future work could therefore be to handle multiple languages or test against another language and see if the results would differ.

Expanding finding stop words into also including domain specific stop words would probably result in more relevant sets. In a medical journal the word patient is probably mentioned many times, and should therefore be considered as a stop word.

Another way of attacking our research question could be to investigate if MNS can be used in a different way to detect topics. One idea could be to use the articles' bags of MNS to create a vector representation. These vectors could then be used when measuring similarity between documents, and these comparisons could be the base for clustering. The vectors would then likely be smaller than how we represent documents in the ground truth, which means that one also would investigate if smaller sets of data could be used to detect topics.

6.4 Ethics

For the methods that we use and the applications, we cannot see anything ethically problematic.

However, our method could be used in other contexts. Suppose an email provider or the owner of a social media platform uses the method on a user's inbox to extract information. Our method could then be used to identify important words from the data extracted from the inbox. This could be used to 'classify' users in a legally and socially unacceptable way. What could be even worse, is if the method for would work incorrectly, and the extracted information was used against the user. The user may then risk to be faultily suspended from his/her accounts.

In summary, the method used in a certain context could violate a user's integrity.

6.5 Conclusion

The aim in this project was to investigate if smaller amounts of data can be used to detect topics, in comparison to methods where the entire document is used, which is how topics are commonly detected. We have explored this by creating a model we call MNS Chaining model, where we select articles based on the MNS found in each article and compare this selection to a ground truth created with Hierarchical Clustering.

Of the experiments we performed, we found that linking articles together based on two words gave a significantly better result than linking based on one word. All experiments generated a majority of very short chains. Since we think that longer chains are more interesting to look at, we thought that it was difficult to find a balance between choosing enough data and choosing good data.

We expected that tracking important words and tracking on more common combinations of words would perform better, which showed to be true, as discussed in Section 6.1.1. We also expected that adding a time feature would improve the model. This was however difficult to evaluate. According to our measurement the time feature worsens the outcome of the model. Nonetheless, when we took a closer look, see Section 5.2, it seemed that the results from the splits actually made sense. In general, a big challenge for us has been to evaluate our results. Especially considering that we could not find an already well established evaluation method that could be applied to our model. Instead, we had to create our own measurement by defining how different important factors relate to each other.

Particularly interesting were the high z-scores and low p-values of our model. Although these indicate that the MNS Chaining model performs very well, these values are not very surprising if the results are critically analyzed, see Section 6.1.1. The values of the z-scores are motivated with the results of the random model. The random model is based on an empirical probability mass function for chain length, and since the generated chains in general were very short, the random model also primarily consists of short chains. Since we defined the success rate of a chain to be zero if all articles belong to different clusters, and this is more likely to happen if the chain is short, many of the chains in the random model have a score, m_{tot}, equal to zero.

Although we defined our ground truth using a well established method to achieve a strong base for comparison, we realized that there were flaws in our ground truth. There does not exist a way to automatically check the quality of the results from our clustering, also the data sets are too large to check all clusters manually. Therefore, we cannot say with certainty that all clusters actually represent topics well.

We think that there are indications that MNS can be used to detect topics. The MNS Chaining model definitely shows better results than the random model, and some chains have even been an exact match for a cluster. However, we still do not consider our model to hold up against established methods. Therefore, we do not think that our current method is suitable for a topic detecting system, but rather that it could be possible to build on our methods. A promising future investigation would be to cluster the articles based on their MNS. That is, to represent each document as a vector based on its MNS rather than the entire text as a whole, and cluster on the resulting vectors. This would investigate the same question that we have tried to answer, but in a different way.

Bibliography

[1] Elke Achtert, Christian Böhm, and Peer Kröger. Deli-clu: boosting robustness, completeness, usability, and efficiency of hierarchical clustering by a closest pair ranking. *PAKDD 2006, Advances in knowledge discovery and data mining*, pages 119–128, 2006.

[2] Charu C Aggarwal and ChengXiang Zhai. *Mining text data*. Springer Science & Business Media, 2012.

[3] James Allan. *Topic detection and tracking: event-based information organization*, volume 12. Springer Science & Business Media, 2012.

[4] James Allan, Jaime G Carbonell, George Doddington, Jonathan Yamron, and Yiming Yang. Topic detection and tracking pilot study final report. In *Proceedings of the Broadcast News Transcription and Understanding Workshop (Sponsored by DARPA)*, pages 194–218, 1998.

[5] Steven Bird, Edward Loper, and Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc, 2009.

[6] Thorsten Brants, Francine Chen, and Ayman Farahat. A System for new event detection. *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval - SIGIR '03*, 2002:330–337, 2003.

[7] Susan Rovezzi Carroll and David J Carroll. *Statistics made simple for school leaders: Data-driven decision making*. R&L Education, 2002.

[8] Christopher Cieri, Stephanie Strassel, David Graff, Nii Martey, Kara Rennert, and Mark Liberman. Corpora for topic detection and tracking. In James Allan, editor, *Topic Detection and Tracking: Event-based Information Organization*, pages 33–66. Springer US, Boston, MA, 2002.

[9] Peter Damaschke. Pairs covered by a sequence of sets. In *20th International Symposium on Fundamentals of Computation Theory, FCT, Lecture Notes in Computer Science*, volume 9210, pages 214–226. Springer, 2015.

[10] Gabriel Pui Cheong Fung, Jeffrey Xu Yu, Philip S Yu, and Hongjun Lu. Parameter free bursty events detection in text streams. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 181–192. VLDB Endowment, 2005.

[11] Reynaldo Gil-García, José M. Badía-Contelles, and Aurora Pons-Porrata. *Dynamic Hierarchical Compact Clustering Algorithm*, pages 302–310. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[12] Stanford NLP Group. Stanford named entity recognizer (ner). `https://nlp.stanford.edu/software/CRF-NER.shtml`. (visited on 09/10/2017).

[13] Örn Guðjónsson. On-line new event detection using minimal new sets. Master's thesis, Chalmers University of Technology, 2016.

[14] Matthew Honnibal and Mark Johnson. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[15] Anna Huang. Similarity measures for text document clustering. In *Proceedings of the sixth New Zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, pages 49–56, 2008.

[16] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[17] Ridong Jiang, Rafael E Banchs, and Haizhou Li. Evaluating and combining named entity recognition systems. In *Proceedings of the Sixth Named Entity Workshop, Joint with 54th Association for Computational Linguistics*, pages 21–27, 2016.

[18] Thorsten Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. *The 14th International Conference on Machine Learning (ICML '97)*, 18(1996):143–151, 1997.

[19] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2017-09-06].

[20] Kathy Lee, Diana Palsetia, Ramanathan Narayanan, Md. Mostofa Ali Patwary, Ankit Agrawal, and Alok Choudhary. Twitter Trending Topic Classification. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 251–258. IEEE, December 2011.

[21] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397, December 2004.

[22] Yanjun Li, Soon M. Chung, and John D. Holt. Text document clustering based on frequent word meaning sequences. *Data & Knowledge Engineering*, 64(1):381 – 404, 2008.

[23] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. Scoring, term weighting and the vector space model. In *Introduction to Information Retrieval*, pages 117 – 119. Cambridge University Press, 2008.

[24] MathWorks. Dendrogram. Available at `https://se.mathworks.com/help/stats/dendrogram.html?s_tid=gn_loc_drop`, Accessed: 2017-09-11.

[25] John H. McDonald. *Handbook of Biological Statistics*. Sparky House Publishing, Baltimore, Maryland, 3rd edition, 2014.

[26] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[27] Gary Miner. *Practical text mining and statistical analysis for non-structured text data applications*. Academic Press, 2012.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[29] Martin F Porter, Richard Boulton, and Andrew Macfarlane. The english (porter2) stemming algorithm. `https://tartarus.org/martin/PorterStemmer/`, 2002. (visited on 02/06/2017).

[30] John Rice. *Mathematical statistics and data analysis*. Nelson Education, 2006.

[31] Isabelle Rivals, Léon Personnaz, Lieng Taing, and Marie-Claude Potier. Enrichment or depletion of a GO category within a class of genes: which test? *Bioinformatics*, 23(4):401–407, 2007.

[32] Yoshihide Sato, Harumi Kawashima, Hidenori Okuda, and Masahiro Oku. Trend-based document clustering for sensitive and stable topic detection. In *PACLIC*, pages 331–340, 2008.

[33] scikit-learn - the Bag of Words representation. `http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction`. Accessed: 2017-09-11.

[34] Michael Steinbach, George Karypis, Vipin Kumar, et al. A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 400, pages 525–526. Boston, 2000.

[35] Alexander Strehl, Joydeep Ghosh, and Raymond Mooney. Impact of similarity measures on web-page clustering. In *Workshop on artificial intelligence for web search (AAAI 2000)*, volume 58, page 64, 2000.

[36] S Vijayarani, J Ilamathi, and Nithya. Preprocessing techniques for text mining - an overview. *International Journal of Computer Science & Communication Networks*, 5(1):7–16, 2015.

[37] Yiming Yang, Tom Pierce, and Jaime Carbonell. A study of retrospective and on-line event detection. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 28–36, New York, NY, USA, 1998. ACM.

[38] Shi Zhong. Efficient Online Spherical K-Means Clustering. *IEEE International Joint Conference on Neural Networks*, 5:3180–3185, 2005.

A

Appendix - Top ten chains

The Tables A.1-A.3 show top ten chains in each data set from experiment with two words using UPGMA sorted from highest z-score. Displayed with each article in the chain is also its UPGMA-cluster id and the size index of that cluster. For example "size index 5/3836" means that it is the 5th biggest cluster out of 3836 clusters.

A.1 Data set 1

Feature: AND[soccer,stadium]		Z-score: 71.39	
Size of largest cluster: 7			
published	title	cluster id	size index
1997-04-07	VIETNAM: Building collapse kills two in Vietnam soccer game.	41	136/776
1997-04-07	NIGERIA: Policeman dies in Nigeria soccer stampede.	41	136/776
1997-04-10	GUATEMALA: Guatemala stadium reopening.	41	136/776
1997-04-14	GUATEMALA: Site of tragedy, Guatemala soccer stadium to reopen.	41	136/776
1997-04-15	GUATEMALA: Victims blast reopening of Guatemala stadium.	41	136/776
1997-06-26	UK: UK government looks again at soccer tragedy.	41	136/776
1997-07-28	MOZAMBIQUE: Two die in road chaos after Mozambique soccer game.	41	136/776

Feature: AND[tobago,trinidad]		Z-score: 71.39	
Size of largest cluster: 4			
published	title	cluster id	size index
1997-04-03	TRINIDAD AND TOBAGO: Quake jolts Trinidad and Tobago.	185	268/776
1997-04-22	USA: Quake reported off coast of Trinidad.	185	268/776
1997-04-22	TRINIDAD AND TOBAGO: Quake rocks Caribbean holiday island.	185	268/776
1997-04-23	TRINIDAD AND TOBAGO: Quake rocks Caribbean holiday island, two hurt.	185	268/776

Feature: AND[condol,pope] Z-score: 71.39
Size of largest cluster: 4

published	title	cluster id	size index
1997-04-18	VATICAN: Vatican sends condolences over haj tragedy.	156	263/776
1997-05-12	VATICAN: Pope sends condolences after Iran quake.	156	263/776
1997-07-11	VATICAN: Pope sends condolences for Venezuela earthquake.	156	263/776
1997-08-06	VATICAN: Pope sends condolences to S.Korea for plane crash.	156	263/776

Feature: AND[nickel,smelter] Z-score: 71.39
Size of largest cluster: 3

published	title	cluster id	size index
1997-04-14	AUSTRALIA: WMC says Victor nickel production shut down.	103	325/776
1997-06-30	FINLAND: Outokumpu details smelter blast damages.	103	325/776
1997-06-30	FINLAND: OUTOKUMPU BLAST HALTS OUTPUT,INJURES THREE.	103	325/776

Feature: AND[mobutu sese seko,zair] Z-score: 71.39
Size of largest cluster: 3

published	title	cluster id	size index
1997-04-15	CONGO: Angolan cargo plane crashes in Congo, three dead.	600	331/776
1997-06-08	DEMOCRATIC REPUBLIC OF CONGO: Plane crash in Congo kills 27 - businessmen.	600	331/776
1997-07-13	DEMOCRATIC REPUBLIC OF CONGO: Kabila's Congo bans use of Russian pilots, planes.	600	331/776

Feature: AND[disturb,solar] Z-score: 71.39
Size of largest cluster: 3

published	title	cluster id	size index
1997-04-09	USA: North American power grid braces for solar storm.	75	284/776
1997-04-10	USA: Solar storm slows, weakens as it nears Earth - NOAA.	75	284/776
1997-06-30	UK: SATELLITES IN DANGER FROM SOLAR FLARES - EXPERT.	75	284/776

Feature: AND[disturb,magnet] Z-score: 71.39
Size of largest cluster: 3

published	title	cluster id	size index
1997-04-09	USA: North American power grid braces for solar storm.	75	284/776

1997-04-10	USA: Solar storm slows, weakens as it nears Earth - NOAA.	75	284/776
1997-06-30	UK: SATELLITES IN DANGER FROM SOLAR FLARES - EXPERT.	75	284/776

Feature: AND[smelter,sterlit]		Z-score: 71.39	
Size of largest cluster: 3			
published	title	cluster id	size index
1997-07-08	INDIA: India’s Sterlite denies gas leak - paper.	100	328/776
1997-07-09	INDIA: India’s Sterlite contests copper smelter shutdown.	100	328/776
1997-08-10	INDIA: REPORT ON INDIA’S STERLITE IN 2-3 DAYS.	100	328/776

Feature: AND[magnet,solar]		Z-score: 71.39	
Size of largest cluster: 3			
published	title	cluster id	size index
1997-04-09	USA: North American power grid braces for solar storm.	75	284/776
1997-04-10	USA: Solar storm slows, weakens as it nears Earth - NOAA.	75	284/776
1997-06-30	UK: SATELLITES IN DANGER FROM SOLAR FLARES - EXPERT.	75	284/776

Feature: AND[copper,sterlit]		Z-score: 71.39	
Size of largest cluster: 3			
published	title	cluster id	size index
1997-07-08	INDIA: India’s Sterlite denies gas leak - paper.	100	328/776
1997-07-09	INDIA: India’s Sterlite contests copper smelter shutdown.	100	328/776
1997-08-10	INDIA: REPORT ON INDIA’S STERLITE IN 2-3 DAYS.	100	328/776

Table A.1: Top ten chains in D_1

A.2 Data set 2

Feature: AND[chile,tompkin]		Z-score: 53.66	
Size of largest cluster: 8			
published	title	cluster id	size index
1997-03-21	CHILE: Chile minister slams U.S. magnate over park plans.	820	87/1004
1997-04-11	CHILE: U.S. tycoon’s proposed reserve in Chile under fire.	820	87/1004

1997-05-13	CHILE: U.S. ecologist charges "harassment" by Chileans.	820	87/1004
1997-05-14	CHILE: Chile denies harassment of U.S. eco-millionaire.	820	87/1004
1997-05-14	CHILE: U.S. eco-millionaire charges harassment by Chileans.	820	87/1004
1997-05-16	CHILE: Chilean legislators defend U.S. eco-millionaire.	820	87/1004
1997-05-21	CHILE: U.S. eco-millionaire wants national park in Chile.	820	87/1004
1997-07-10	CHILE: U.S. eco-millionaire splits Chilean opinion-poll.	820	87/1004

Feature: AND[endesa,ralco] Z-score: 53.66
Size of largest cluster: 6

published	title	cluster id	size index
1997-02-20	CHILE: World Bank criticizes Chile's Endesa - report.	819	119/1004
1997-02-20	CHILE: Chile's Endesa rejects World Bank criticism.	819	119/1004
1997-06-06	USA: Greens cry foul on World Bank's Chile dam report.	819	119/1004
1997-06-10	CHILE: Chile approves building of controversial dam.	819	119/1004
1997-06-10	CHILE: Chilean Indians, activists protest licensing of dam.	819	119/1004
1997-06-11	CHILE: Endesa to appeal conditions for Ralco dam.	819	119/1004

Feature: AND[tobago,trinidad] Z-score: 53.66
Size of largest cluster: 4

published	title	cluster id	size index
1997-04-03	TRINIDAD AND TOBAGO: Quake jolts Trinidad and Tobago.	237	184/1004
1997-04-22	USA: Quake reported off coast of Trinidad.	237	184/1004
1997-04-22	TRINIDAD AND TOBAGO: Quake rocks Caribbean holiday island.	237	184/1004
1997-04-23	TRINIDAD AND TOBAGO: Quake rocks Caribbean holiday island, two hurt.	237	184/1004

Feature: AND[diesel,emir] Z-score: 53.66
Size of largest cluster: 3

published	title	cluster id	size index
1997-07-13	DUBAI: UAE oil spill barge towed away.	906	299/1004
1997-07-13	DUBAI: Oil spill threatens UAE water supplies.	906	299/1004

1997-07-16	DUBAI: Diesel spill leaves sour taste in UAE.	906	299/1004

Feature: AND[botswana,the okavango riv]		Z-score: 53.66	
Size of largest cluster: 3			
published	title	cluster id	size index
1997-05-20	LESOTHO: Namibia to eventually use Okavango water, official.	277	311/1004
1997-07-03	NAMIBIA: Nambia suspends immediate plans to drain Okavango.	277	311/1004
1997-07-03	NAMIBIA: Namibia suspends plans to tap Okavango.	277	311/1004

Feature: AND[mmt,royal]		Z-score: 53.66	
Size of largest cluster: 3			
published	title	cluster id	size index
1997-03-20	CANADA: Senate motion to kill Canadian MMT bill fails.	620	295/1004
1997-04-09	CANADA: Canada's ban on MMT passes final hurdle.	620	295/1004
1997-04-25	CANADA: Canada makes tobacco, MMT bills law.	620	295/1004

Feature: AND[persson,seminar]		Z-score: 53.66	
Size of largest cluster: 3			
published	title	cluster id	size index
1997-04-11	SWEDEN: Environment spending to generate jobs - Swedish PM.	458	228/1004
1997-04-11	SWEDEN: Sweden PM says to spend 10 bln Skr on environment.	458	228/1004
1997-04-11	SWEDEN: Sweden hopes to create 50,000 new 'green' jobs.	458	228/1004

Feature: AND[about 600,phalaborwa]		Z-score: 53.66	
Size of largest cluster: 3			
published	title	cluster id	size index
1997-05-08	SOUTH AFRICA: S.AFRICAN GREENS ASSURED ON IRON PLANT PROJECT.	358	278/1004
1997-05-21	SOUTH AFRICA: Public to have input on planned SAfrica iron plant.	358	278/1004
1997-06-20	SOUTH AFRICA: S.Africa govt opposes iron ore pipe through Kruger.	358	278/1004

Feature: AND[maputo,phalaborwa]		Z-score: 53.66	
Size of largest cluster: 3			
published	title	cluster id	size index
1997-05-08	SOUTH AFRICA: S.AFRICAN GREENS ASSURED ON IRON PLANT PROJECT.	358	278/1004
1997-05-21	SOUTH AFRICA: Public to have input on planned SAfrica iron plant.	358	278/1004

1997-06-20	SOUTH AFRICA: S.Africa govt opposes iron ore pipe through Kruger.	358	278/1004

Feature: AND[maputo,mozambiqu] Size of largest cluster: 3		Z-score: 53.66	
published	title	cluster id	size index
1997-05-08	SOUTH AFRICA: S.AFRICAN GREENS ASSURED ON IRON PLANT PROJECT.	358	278/1004
1997-05-21	SOUTH AFRICA: Public to have input on planned SAfrica iron plant.	358	278/1004
1997-06-20	SOUTH AFRICA: S.Africa govt opposes iron ore pipe through Kruger.	358	278/1004

Table A.2: Top ten chains in D_2

A.3 Data set 3

Feature: AND[rosneft,sidanko] Size of largest cluster: 7		Z-score: 140.19	
published	title	cluster id	size index
1997-02-28	RUSSIA: Russian SIDANKO in 3rd fight for key oil producer.	2020	485/3836
1997-03-20	RUSSIA: Russian court postpones Purneft hearing to April 3.	2020	485/3836
1997-04-14	RUSSIA: Moscow court turns down SIDANKO suit vs Rosneft.	2020	485/3836
1997-07-07	RUSSIA: Russian Rosneft oil firm mulls legal setback.	2020	485/3836
1997-07-16	RUSSIA: Russian court halts Purneftegaz ownership change.	2020	485/3836
1997-08-04	RUSSIA: Russia Purneftegaz asks Yeltsin to decide on owner.	2020	485/3836
1997-08-14	RUSSIA: Russian Purneftegaz oil firm court case delayed.	2020	485/3836

Feature: AND[privatis,rosgosstrakh] Size of largest cluster: 7		Z-score: 140.19	
published	title	cluster id	size index
1997-02-26	RUSSIA: Court suspends Russian Rosgosstrakh privatisation.	2024	476/3836
1997-03-24	RUSSIA: Court backs Russian Rosgosstrakh selloff plans.	2024	476/3836
1997-03-25	RUSSIA: Court drops suit against Russian insurer sale.	2024	476/3836

1997-05-14	RUSSIA: Russia court to consider suit against insurer sale.	2024	476/3836
1997-05-20	RUSSIA: Russian Audit Chamber slams Rosgosstrakh sale.	2024	476/3836
1997-06-02	RUSSIA: Moscow court postpones Rosgosstrakh sale hearing.	2024	476/3836
1997-08-13	RUSSIA: RUSSIAN ROSGOSSTRAKH COURT HEARING DELAYED AUG 28.	2024	476/3836

Feature: AND[echostar,news corp] Z-score: 140.19
Size of largest cluster: 7

published	title	cluster id	size index
1997-05-09	USA: News Corp to contest EchoStar claim.	190	499/3836
1997-05-09	USA: Gulf widens between Echostar and News Corp..	190	499/3836
1997-05-09	USA: Echostar says files against News Corp.	190	499/3836
1997-05-12	USA: EchoStar files suit against News Corp..	190	499/3836
1997-05-12	USA: News Corp declines comment on Echostar.	190	499/3836
1997-05-12	USA: EchoStar amends suit, seeks financing.	190	499/3836
1997-06-09	USA: News Corp names Echostar in counterclaim.	190	499/3836

Feature: AND[fininvest,telecinco] Z-score: 140.19
Size of largest cluster: 4

published	title	cluster id	size index
1997-07-24	SPAIN: Italy's Berlusconi in Spain legal wrangle.	3108	785/3836
1997-07-25	SPAIN: Germany's Kirch in Spanish legal wrangle.	3108	785/3836
1997-07-29	ITALY: Italian tax police search Fininvest group offices.	3108	785/3836
1997-07-29	ITALY: FOCUS-Italian tax police search Fininvest offices.	3108	785/3836

Feature: AND[o connor,perzigian] Z-score: 140.19
Size of largest cluster: 4

published	title	cluster id	size index
1997-05-05	USA: Judge dismisses actor's suit against drug dealer.	143	964/3836
1997-07-25	USA: Carroll O'Connor cleared in slander suit.	143	964/3836
1997-07-25	USA: Jury deliberates in Carroll O'Connor slander case.	143	964/3836

1997-07-25	USA: Carroll O'Connor found not guilty in slander suit.	143	964/3836

Feature: AND[laser,visx] Size of largest cluster: 4		Z-score: 140.19	
published	title	cluster id	size index
1997-03-26	USA: Pillar Point, LaserSight enter settlement.	3764	957/3836
1997-03-31	USA: VISX, Autonomous in pact, settle suits.	3764	957/3836
1997-05-28	USA: VISX settles patent litigation with LaserSight.	3764	957/3836
1997-06-18	USA: Summit, VISX settle patent dispute.	3764	957/3836

Feature: AND[cinema,perreira] Size of largest cluster: 4		Z-score: 140.19	
published	title	cluster id	size index
1997-06-14	INDIA: Indian police arrest four after cinema fire.	3341	774/3836
1997-06-14	INDIA: Four arrested after India cinema fire kills 57.	3341	774/3836
1997-06-14	INDIA: Indian police accuse cinema managers of homicide.	3341	774/3836
1997-06-14	INDIA: Four arrested after fire kills 57 in India cinema.	3341	774/3836

Feature: AND[mcafe,symantec] Size of largest cluster: 4		Z-score: 140.19	
published	title	cluster id	size index
1997-04-23	USA: Symantec says files McAfee lawsuit.	738	892/3836
1997-05-14	USA: Trend Micro sues McAfee, Symantec for infringement.	738	892/3836
1997-07-21	USA: Symantec adds charges to McAfee suit.	738	892/3836
1997-07-22	USA: McAfee denies Symantec allegations.	738	892/3836

Feature: AND[gorilla,max] Size of largest cluster: 4		Z-score: 140.19	
published	title	cluster id	size index
1997-07-18	SOUTH AFRICA: Gorilla survives gunbattle in S.African zoo.	45	780/3836
1997-07-22	SOUTH AFRICA: GORILLA BECOMES S.AFRICA'S LATEST CULT FIGURE.	45	780/3836
1997-07-22	SOUTH AFRICA: Gorilla becomes South Africa's latest cult figure.	45	780/3836

1997-08-15	SOUTH AFRICA: GORILLA MAX'S ASSAILANT ESCAPES FROM PRISON.	45	780/3836

Feature: AND[azteca,nbc] Size of largest cluster: 4		Z-score: 140.19	
published	title	cluster id	size index
1997-04-30	MEXICO: FOCUS-NBC, Mexico's Azteca in legal battle.	1624	818/3836
1997-04-30	MEXICO: NBC says to take legal action vs TV Azteca.	1624	818/3836
1997-04-30	MEXICO: Mexico's TV Azteca to file lawsuit against NBC.	1624	818/3836
1997-08-14	MEXICO: Legal row with NBC clouds Azteca's IPO.	1624	818/3836

Table A.3: Top ten chains in D_3

B

Appendix - Plots

B.1 Histograms of Z-scores

In Figures B.1-B.3 we have plotted histograms of the z-scores for tracking on one word for each data set and linkage method, to show the distribution of z-scores. Figures B.4-B.6 show the plotted histograms for tracking on two words.

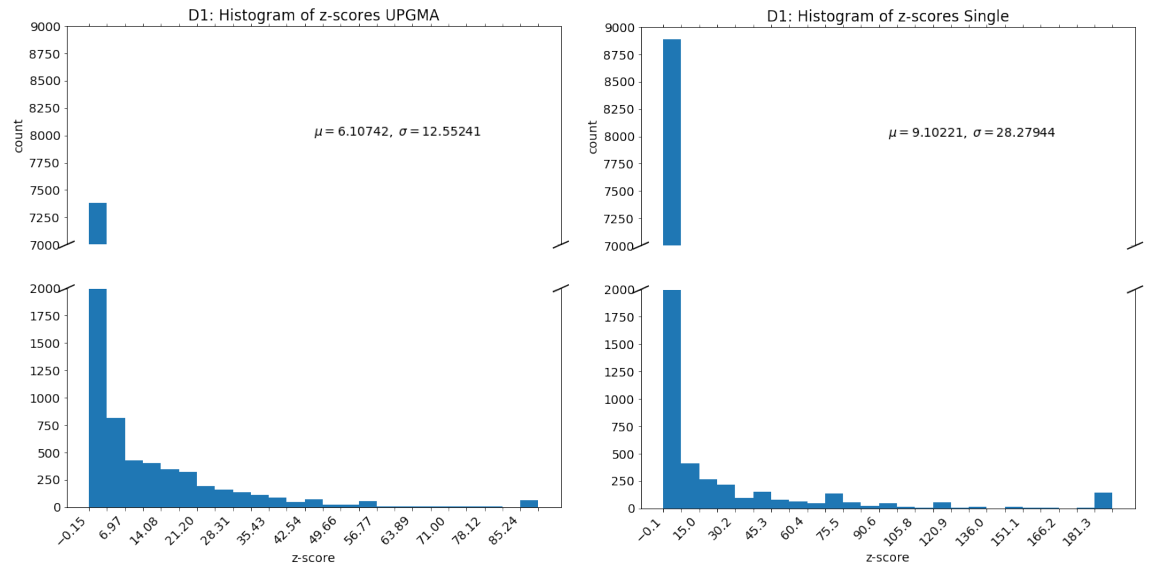


Figure B.1: Histograms of z-scores for tracking one word with data set D_1.

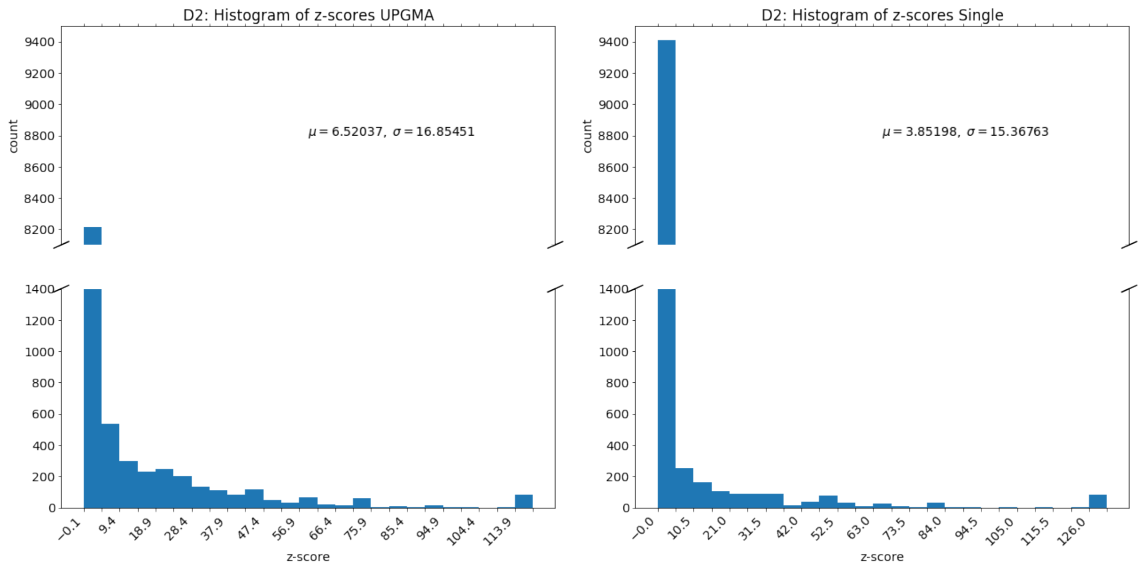


Figure B.2: Histograms of z-scores for tracking one word with data set D_2.

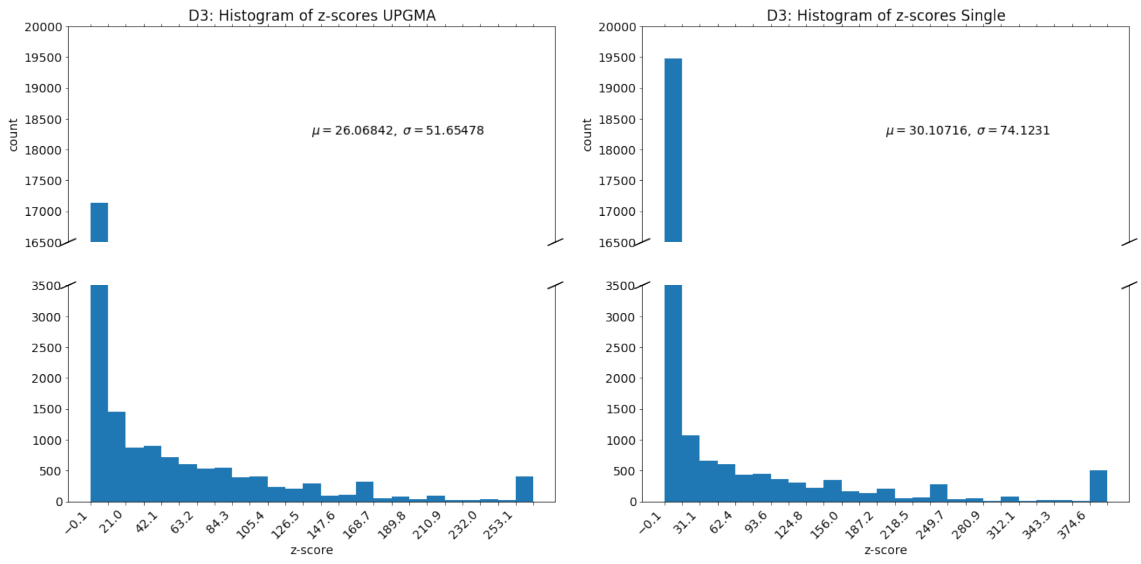


Figure B.3: Histograms of z-scores for tracking one word with data set D_3.

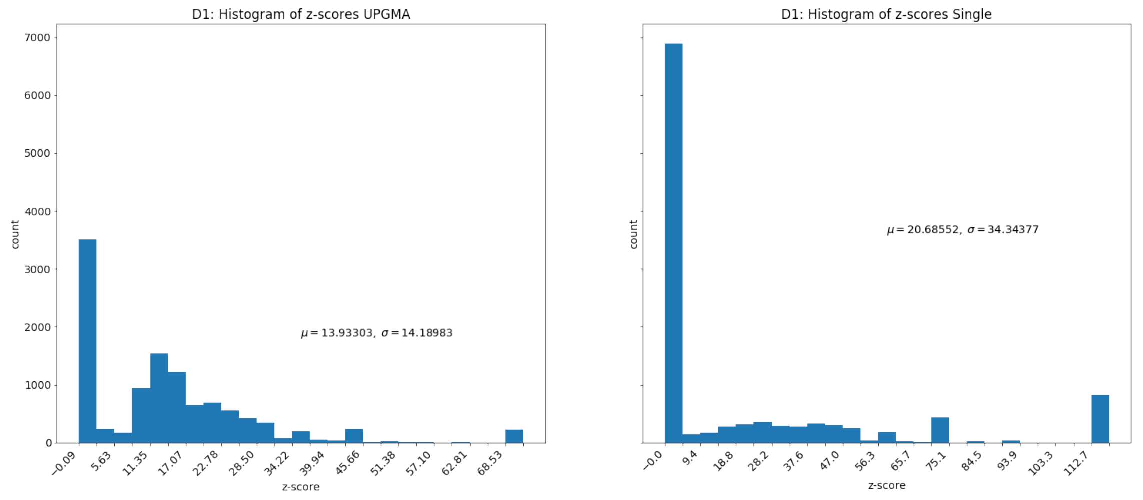


Figure B.4: Histograms of z-scores for tracking two words with data set D_1.

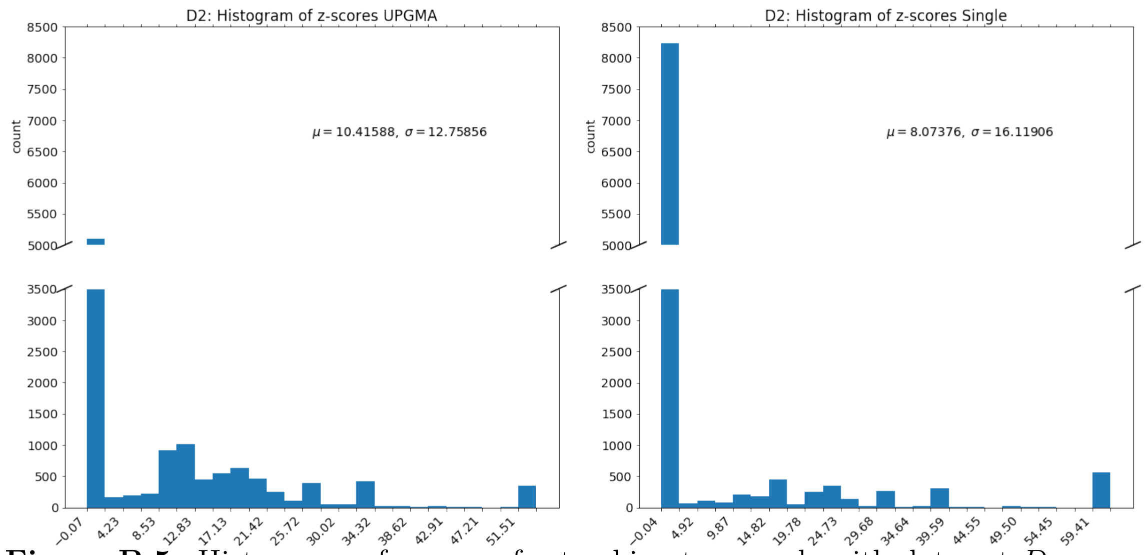


Figure B.5: Histograms of z-scores for tracking two words with data set D_2.

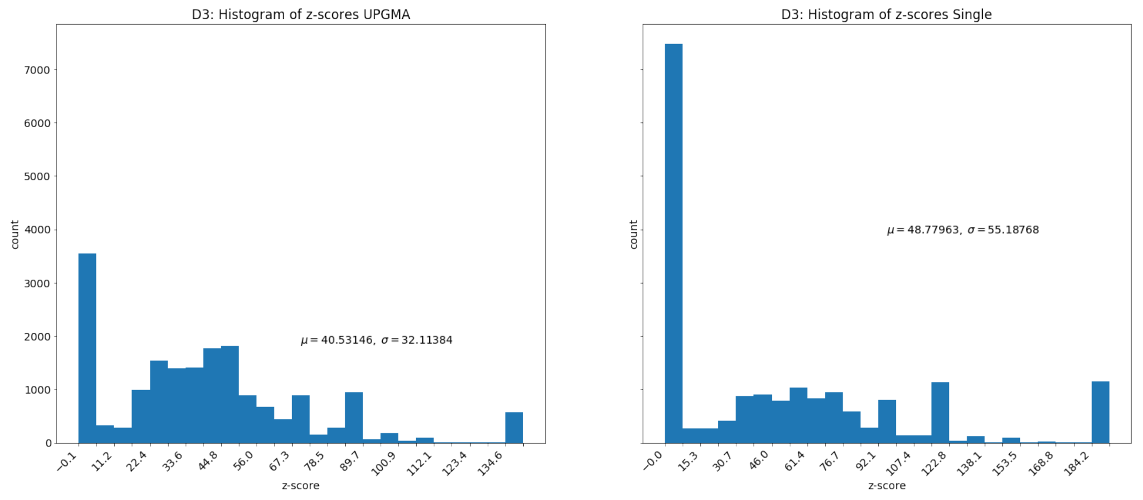


Figure B.6: Histograms of z-scores for tracking two words with data set D_3.

B.2 Z-score in regards to Average tf-idf-value

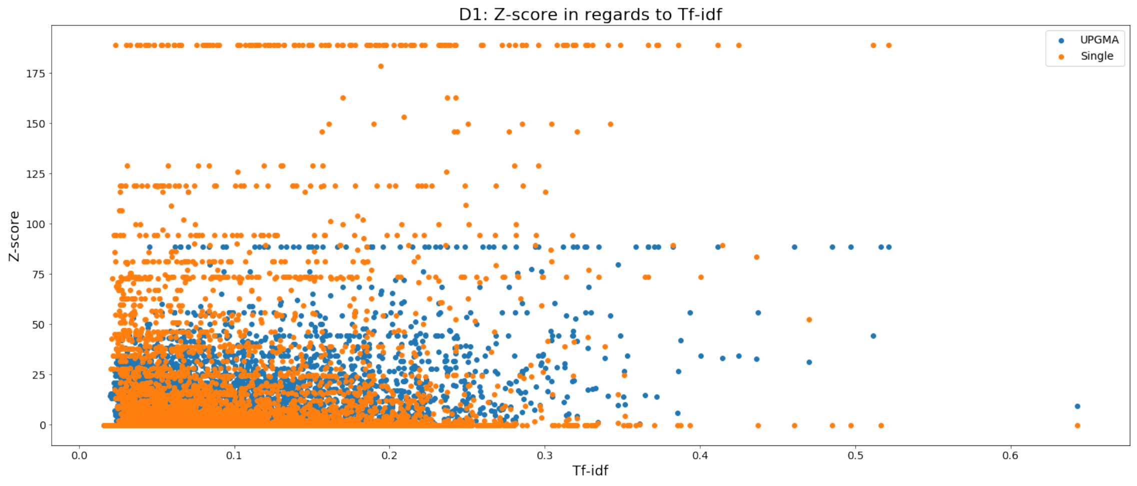


Figure B.7: The average tf-idf of all words in MNS from D_1 plotted against their z-score.

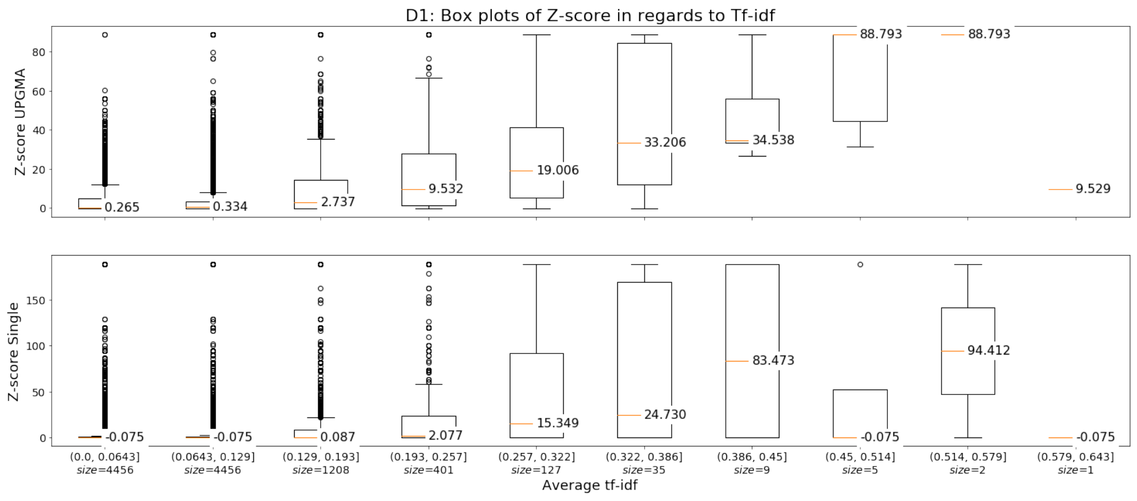


Figure B.8: Box plots of different intervals for the average tf-idf of all words in MNS from D_1 plotted against their z-score.

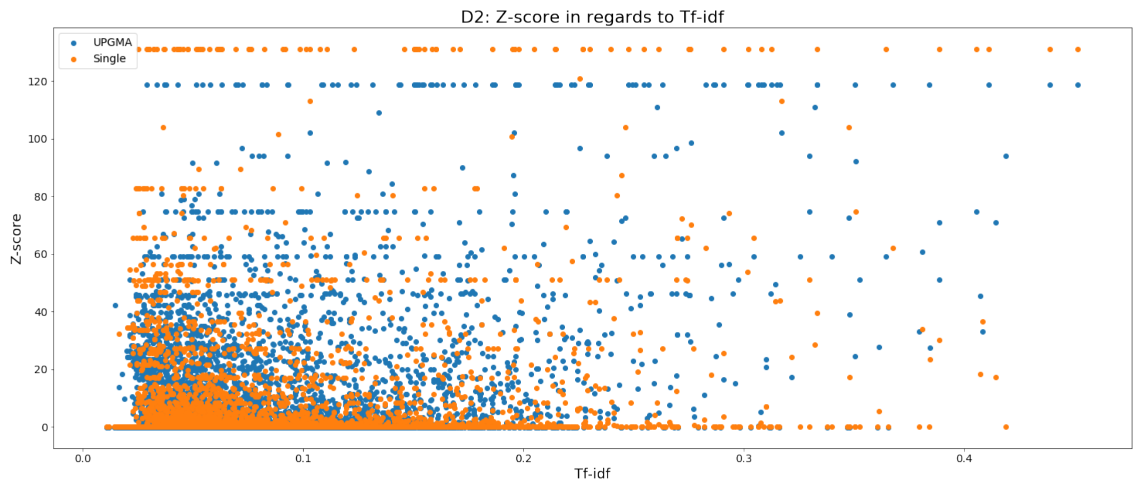


Figure B.9: The average tf-idf of all words in MNS from D_2 plotted against their z-score.

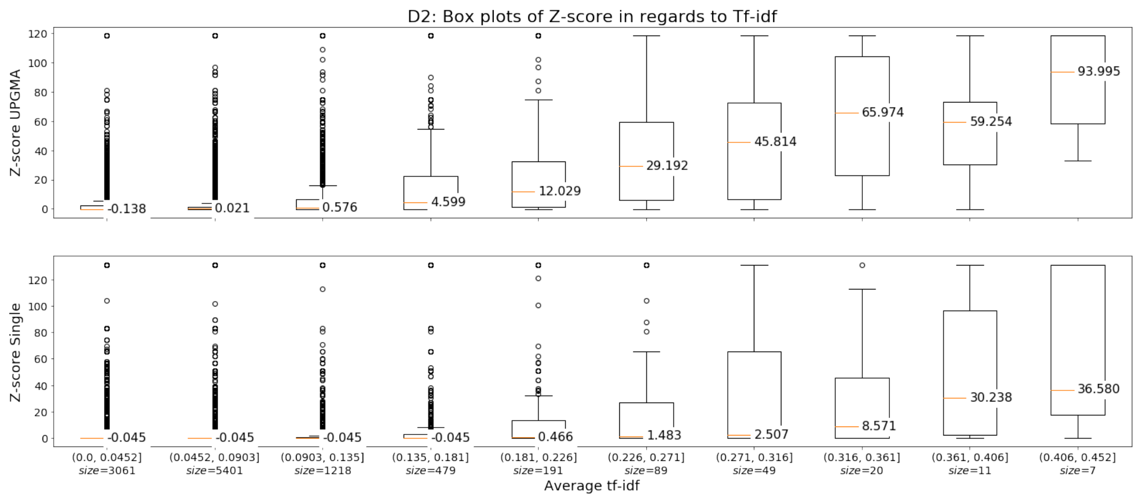


Figure B.10: Box plots of different intervals for the average tf-idf of all words in MNS from D_2 plotted against their z-score.

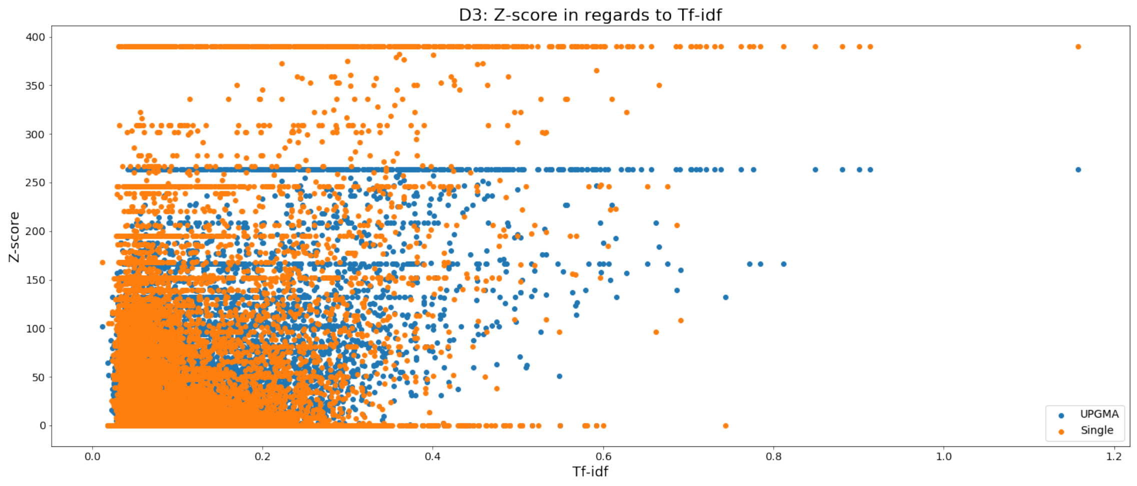


Figure B.11: The average tf-idf of all words in MNS from D_3 plotted against their z-score.

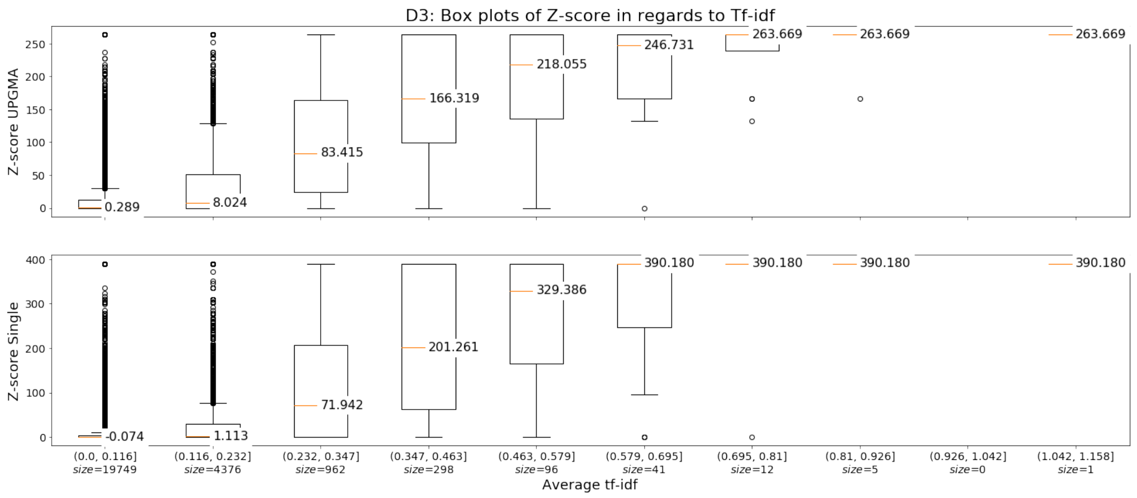


Figure B.12: Box plots of different intervals for the average tf-idf of all words in MNS from D_3 plotted against their z-score.

B.3 Z-score in regards to Jaccard Index

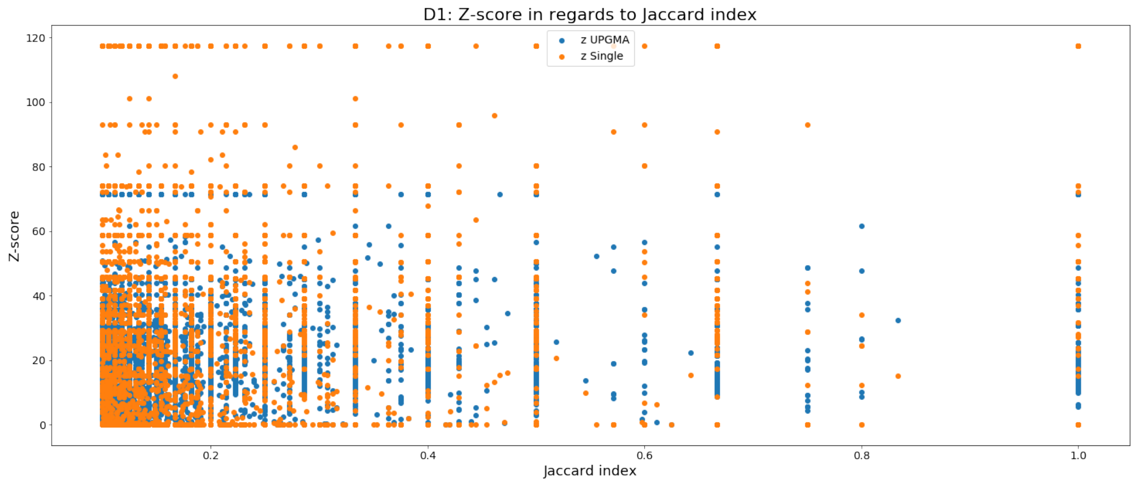


Figure B.13: The Jaccard index of two words plotted against the z-score. Data set: D_1.

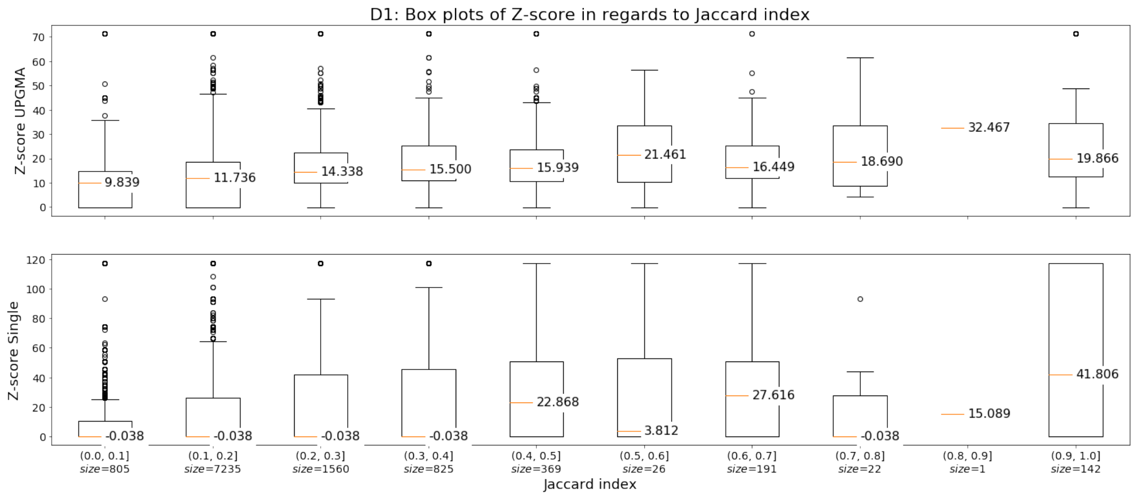


Figure B.14: Box plots of different intervals of the Jaccard index of two words plotted against the z-score on data set D_1.

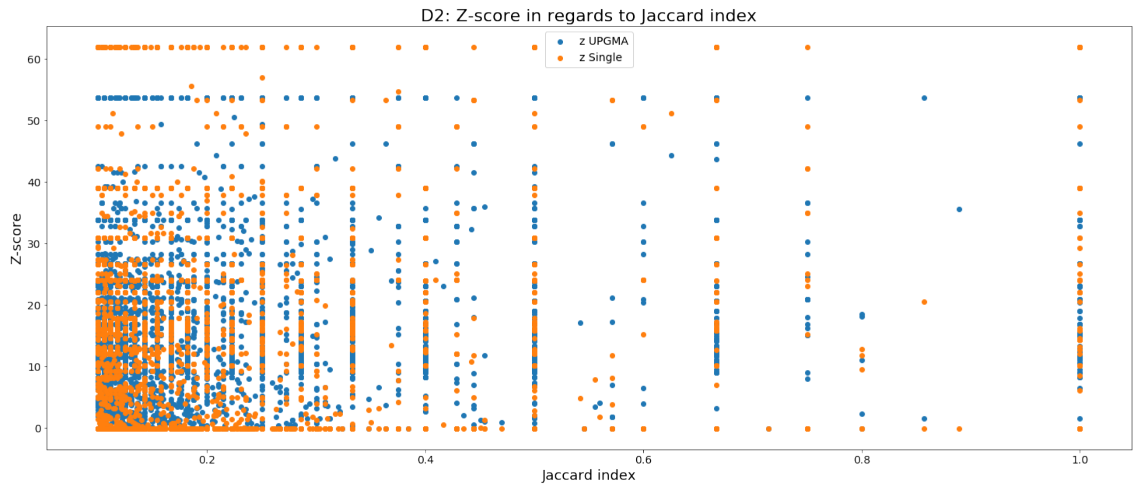


Figure B.15: The Jaccard index of two words plotted against the z-score. Data set: D_2.

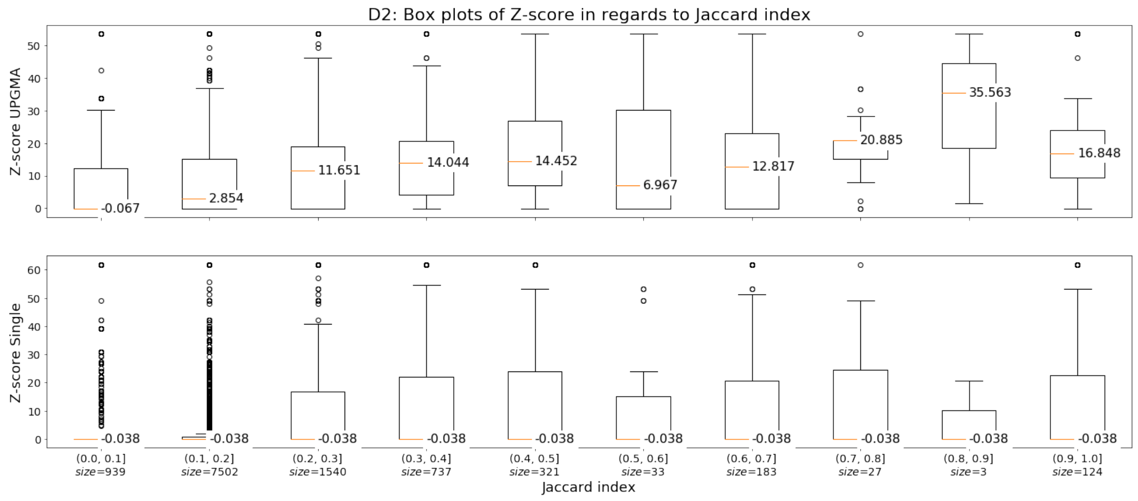


Figure B.16: Box plots of different intervals of the Jaccard index of two words plotted against the z-score on data set D_2.

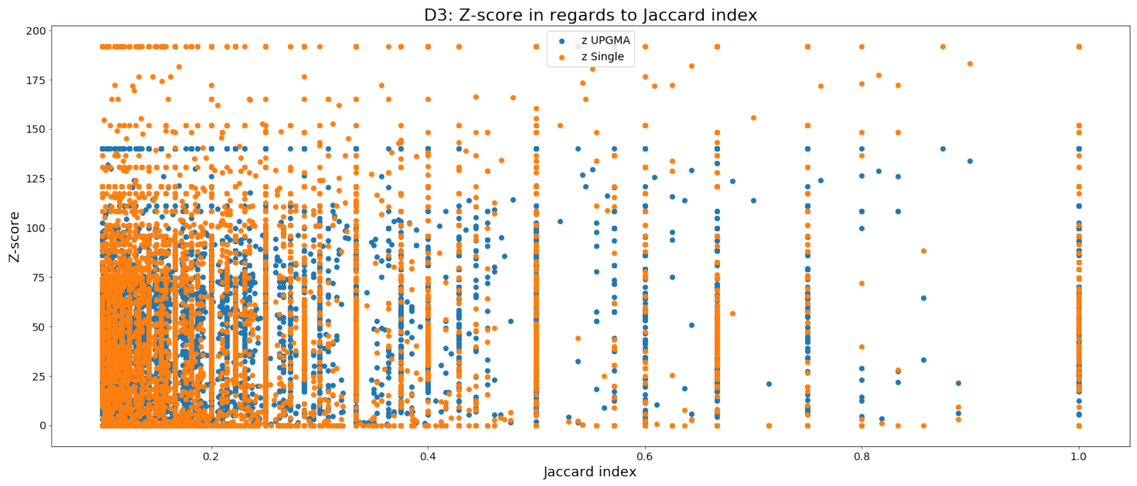


Figure B.17: The Jaccard index of two words plotted against the z-score. Data set: D_3.

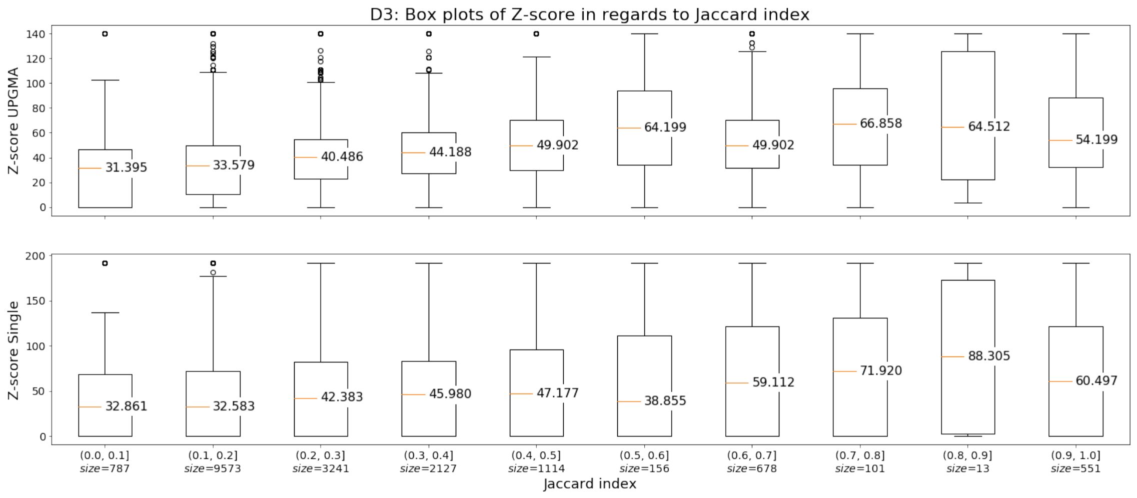


Figure B.18: Box plots of different intervals of the Jaccard index of two words plotted against the z-score on data set D_3.

C

Appendix - Source code

C.1 article_linker.py

Creates chains, i.e. links articles based on a specified feature. Described in more detail in Section 3.1.4.

```python
import pickle_handler
from collections import defaultdict
from features import Feature, FeatureType

__author__ = "Madeleine Appert & Lisa Stenberg"


class ArticleLinker:
    def __init__(self, category):
        self.category = category
        self.articles = pickle_handler.get_pickled_mns(category)
        self.article_count = 0
        self.vocabulary = None
        self.no_wanted_words = 0
        self.jaccard_similarity_combos = {}
        self._set_vocab()

    def get_tracking_words(self, percent_of_words=1):
        """
        Finds the percent_of_words% of words that appear most frequently in
        MNS within the category and returns them.

        :param percent_of_words: Percent of words that should be returned.
        Is by default set to 1 (which is a lot).
        :return: Dictionary where key is the word and value is a list with 2
        values: [total count, document frequency].
        """

        def sort_on_total_count(x): return x[1][0]
        # def sort_on_document_frequency(x): return x[1][1]

        sorted_list = sorted(self.vocabulary.items(),
                             key=sort_on_total_count, reverse=True)

        no_wanted_words = round(percent_of_words/100*len(self.vocabulary))
        self.no_wanted_words = no_wanted_words
        as_dict = dict(sorted_list[:no_wanted_words])
        print("Tracking words from {0} retrieved. {1}% of {2} words - {3} "
              "words in total.".format(self.category,
                                          percent_of_words,
                                          len(self.vocabulary),
                                          no_wanted_words))
        return as_dict

    def get_tracking_tuples(self, percent_of_words=10):
        self.calculate_jaccard()

        def bigger_than_x_percent(x):
            return x[1][0] >= (percent_of_words / 100)

        def sort_on_jaccard(x):
            return x[1]

        above = [t for t in self.jaccard_similarity_combos.items() if
                 bigger_than_x_percent(t)]
        self.no_wanted_words = len(above)
        as_dict = dict(above)
        print("Tracking words from {0} retrieved. Of {2} combinations, "
              "those with above {1}% in jaccard are used - {3} combinations "
              "in total.".format(self.category,
                                 percent_of_words,
                                 len(self.jaccard_similarity_combos),
                                 self.no_wanted_words))
        return as_dict

    def get_chain_tuples(self, feature):
        """
        Provided a Feature object, gets the articles that fit the
        requirements, split up based on the time variable.
        :param feature: A Feature object
        :return: List of tuples. Each tuple contains a list of article IDs
        and a list of corresponding dates.
        """
        chain = self._get_chain(feature)
        if not chain:
            return chain
        sorted_tuple_list = self._get_sorted_list_dates_and_ids(chain)

        # Split the chain of articles based on the time variable and get
        # tuple with ids and dates.
        chains = self.split_chain_by_time(sorted_tuple_list, feature.time)
        return chains

    def get_co_occurrences(self, word):
        return list(self.co_occurrences[word])

    def _get_chain(self, feature):
        """
        Provided a Feature object, gets the articles that fit the content
        based requirements.
        :param feature: A Feature object
        :return: List of article IDs. If no matches are found an empty list
        will be returned.
        """
        chain = []
        if feature.feature_type == FeatureType.CONTENT:
            # 'feature' holds a list of words. Find article IDs based on that
            #  list.
            chain = self._get_chain_for_content(feature.content,
                                                feature.operation_type)
        elif feature.feature_type == FeatureType.FEATURE:
            # 'feature' holds a list of Feature objects. Call this method
            # recursively for each and join the results.
            chains = [self._get_chain(f) for f in feature]
            chain = self._join_lists(chains, feature.operation_type)
        elif feature.feature_type == FeatureType.MIX:
            # 'feature' holds a mix of Feature objects and words. Do like
            # above, for the 2 types, and join the results.
            features = [e for e in feature if isinstance(e, Feature)]
            chains = [self._get_chain(f) for f in features]
            content = [e for e in feature if isinstance(e, str)]
            chains.append(self._get_chain_for_content(content,
                                                      feature.operation_type))
            chain = self._join_lists(chains, feature.operation_type)
        return chain

    @staticmethod
    def _join_lists(elements, operation):
        """
        Takes a list of lists and performs the given operation to create one
        list.
        :param elements: List of lists (containing article IDs).
        :param operation: Function to perform, defined by the variable
        operation_type in a Feature object.
        :return: A list (containing article IDs).
        """
        content_based_matches = elements[0]
        for b in elements:
            content_based_matches = operation(content_based_matches, b)
        return content_based_matches

    def _get_chain_for_content(self, content, operation):
        """
        Will create a list of IDs of the articles that contain all the words
        provided in the list content.
        :param content: A list of words
        :return: A list of article IDs
        """
        content_match_list = []
        for word in content:
            content_match_list.append(self.word_found_in[word])
        return self._join_lists(content_match_list, operation)

    @staticmethod
    def _is_time_within_range(prev_date, date, time_limit):
        """
        Checks if the given parameter date, is within the limit from the
        parameter last_date
        :param prev_date: Date to count from, sets the start of the allowed
        time interval
        :param date: Date that is being checked
        :param time_limit: Timedelta object, sets the end of the allowed time
        interval
        :return: True if prev_date isn't a date (no start of the interval
        exists). True if date is within the allowed interval. Else False.
        """
        return (not prev_date) or date - prev_date < time_limit

    def _get_sorted_list_dates_and_ids(self, id_list):
        """
        Get a sorted list of tuples, that contains dates and the article ids,
        based on the provided list of IDs.
        :param id_list: List of IDs
        :return: Sorted list of dates and IDs.
        Example:
        <class 'list'>: [(Timestamp('1997-03-06 00:00:00'), [424633]),
         (Timestamp('1997-03-09 00:00:00'), [429315])]
        """
        # Create a dict where the key is the published date and the value is
        # a list of IDs.
        date_dict = defaultdict(list)
        for article_id in id_list:
            date = self.articles[self.articles._id == article_id].iloc[
                0].datetime
            date_dict[date].append(article_id)
        return sorted(date_dict.items())

    def split_chain_by_time(self, sorted_tuple_list, time_limit):
        """
        Splits the given list of IDs based on the timestamps of the articles
        and the allowed time_limit.
        :param sorted_tuple_list: Sorted list of tuples where 1st elem is
        date and 2nd is list of IDs for that date.
        :param time_limit: Timedelta object.
        :return: List of tuples. Each tuple contains a list of article IDs
        and a list of corresponding dates.
        """
        chains = []
        current_articles, current_dates = [], []
        prev_date = None
        for date, values in sorted_tuple_list:
            if not time_limit or self._is_time_within_range(prev_date, date,
                                                            time_limit):
                current_articles.extend(values)
                current_dates.extend([date for _ in values])
            else:
                chains.append((current_articles, current_dates))
                current_articles = values
                current_dates = [date for _ in values]
            prev_date = date
        chains.append((current_articles, current_dates))
        return chains

    def _set_vocab(self):
        """
        Goes through all articles and adds the words found in its MNS to the
        vocabulary. Creates the instance variable vocabulary, which is a
        dictionary that keeps track of the total word count and the document
        frequency for each word. Also creates the instance variable
        word_found_in which is a dictionary that keeps track of the articles
        that has the word.
        :return: None
        """
        self.co_occurrences = defaultdict(lambda: defaultdict(int))
        self.vocabulary = {}
        self.article_ids = []
        self.word_found_in = defaultdict(list)
        for index, row in self.articles.iterrows():
            sorted_words = sorted(row['df'].items())
            indexes = range(len(sorted_words))
            self.article_ids.append(row['_id'])
            for (word, (count, tfidf)), i in zip(sorted_words, indexes):
                # TODO Currently not doing anything with tfidf
                for (other_word, _) in sorted_words[i + 1:]:
                    self.co_occurrences[word][other_word] += 1
                current = self.vocabulary.get(word, [0, 0])
                current[0] += count
                current[1] += 1
                self.vocabulary[word] = current
                self.word_found_in[word].append(row['_id'])

        # Remove words that only exist within one article.
        # Note that these words will still exist in the value part of
        # self.co_occurrences.
        print("Vocabulary for {0} has been set up. A total of {1} words "
              "found. Number of combinations: {2}".format(self.category,

                                                         ↪  len(self.vocabulary),
                                                         self.combo_count()))
        only_has_one = [word for word, (total, doc_freq) in
                        self.vocabulary.items() if doc_freq == 1]
        for word in only_has_one:
            del self.vocabulary[word]
            del self.word_found_in[word]
            if word in self.co_occurrences.keys():
                del self.co_occurrences[word]
        print("Removed all words that only exist in 1 document. A total of "
              "{0} words remain.".format(len(self.vocabulary)))

    def combo_count(self):
        """
        Temporary to count total number of combinations that exist, before
        filtering.
        """
        tmp = {}

        def get_document_frequency(x):
            return self.vocabulary[x][1]

        for k, v in self.co_occurrences.items():
            k_value = get_document_frequency(k)
            d = dict((k2, v2) for k2, v2 in v.items())
            for k2, v2 in d.items():
```

```python
                k2_value = get_document_frequency(k2)
                jaccard_similarity = v2 / (k_value + k2_value - v2)
                tmp[(k, k2)] = (jaccard_similarity, k_value, k2_value, v2)
        return len(tmp)

    def calculate_jaccard(self):
        def get_document_frequency(x):
            return self.vocabulary[x][1]

        for k, v in self.co_occurrences.items():
            k_value = get_document_frequency(k)
            d = dict((k2, v2) for k2, v2 in v.items() if v2 > 1)
            for k2, v2 in d.items():
                k2_value = get_document_frequency(k2)
                jaccard_similarity = v2/(k_value+k2_value-v2)
                self.jaccard_similarity_combos[(k, k2)] = (
                    jaccard_similarity, k_value, k2_value, v2)
```

C.2 features.py

A class representing features which we use to link articles. Described in more detail in Section 3.1.4.

```python
from enum import Enum
from datetime import timedelta

__author__ = "Madeleine Appert & Lisa Stenberg"


def intersect(a, b):
    """
    Returns a list which is the intersection of two lists.
    :param a: First list
    :param b: Second list
    :return: A list which is the intersection of two lists.
    """
    return list(set(a).intersection(b))


def unite(a, b):
    """
    Returns a list which is the union of two lists.
    :param a: First list
    :param b: Second list
    :return: A list which is the union of two lists.
    """
    return list(set(a).union(b))


class FeatureType(Enum):
    FEATURE = 1
    CONTENT = 2
    MIX = 3

```

```

class OperationType(Enum):
    """
    Each enum-value is set to a function. Will either intersect or unite 2
↪   lists.
    """
    AND = intersect
    OR = unite


class Feature:
    def __init__(self, content, operation_type=OperationType.AND, time=None):
        self.operation_type = operation_type
        if time and not isinstance(time, timedelta):
            raise ValueError("Param time must be a timedelta!")
        if not content:
            raise ValueError("Content needs to be injected!")
        self.time = time
        self.content = content if isinstance(content, list) else [content]
        self._set_feature_type()
        self._set_time()

    def add_feature(self, new_content):
        self.content.append(new_content)
        self._set_feature_type()
        self._set_time()

    def _set_time(self):
        """
        Sets the variable time to the lowest it can be. Compares this
        instance own time value with the time values of any nested Feature
        objects.
        """
        # Edit the time variable to the lowest one
        for f in [e for e in self.content if isinstance(e, Feature)]:
            self.time = f.time if f.time and (not self.time or f.time <
                                                   self.time) else self.time

    def _set_feature_type(self):
        """
        Sets the enum variable feature_type to what it should be, based on
        the objects found in the variable content.
        """
        if all(isinstance(elem, str) for elem in self.content):
            self.feature_type = FeatureType.CONTENT
        elif all(isinstance(elem, Feature) for elem in self.content):
            self.feature_type = FeatureType.FEATURE
        elif all(isinstance(elem, (str, Feature)) for elem in self.content):
            self.feature_type = FeatureType.MIX
        else:
            raise ValueError("At least one element isn't a string or a "
                             "Feature: ")

    def __iter__(self):
        """
        Enable using the 'for f in feature' syntax
        :return: An element in the content.
        """
        for i in self.content:
            yield i

    def __str__(self):
        """
        Overrides the common str() method. Returns a string containing the
        elements in content and the timedelta value.
        :return: A string containing the elements in content and the
        timedelta value.
        """
        return_string = "AND[{0}]" if self.operation_type == \
                                      OperationType.AND else "OR[{0}]"
        strings = [str(c) for c in self]
        strings = ",".join(strings)
        return_string = return_string.format(strings)
        # Remove any inner timedelta information as it will not be applied
        days = "days"
        while days in return_string:
            i = return_string.index(days)
            for e in reversed(range(i)):
                if not return_string[e].isdigit():
                    start = e
                    break
            end = i + len(days)
            return_string = return_string[:start] + return_string[end:]
        if self.time:
            return_string = return_string + '-' + str(self.time.days) + days
        return return_string
```

C.3 ground_truth_getter.py

Creates a ground truth based on a linkage method and metric. From a chain of article id's the class will return a corresponding list of cluster id's. Described in more detail in Section 3.2.

```
import pickle_handler
import pandas as pd
from scipy.cluster.hierarchy import fcluster

__author__ = "Madeleine Appert & Lisa Stenberg"


class GroundTruthGetter:
    def __init__(self, category, max_d, linkage_method='average',
                 metric='cosine'):
        self.category = category
        self.linkage_method = linkage_method
        self.metric = metric
        self.max_d = max_d

        (_, self.category_df, self.linkage_matrix) = self._load_pickles()

        self.assignments = self._get_assignments()

    def _load_pickles(self):
        linkage_matrix = pickle_handler.get_pickled_linkage_matrix(
            self.category, self.metric, self.linkage_method)
        articles_data = pickle_handler.get_pickled_data_frame(self.category)
        docs = articles_data.get('text').tolist()
        return docs, articles_data, linkage_matrix

    def _get_assignments(self):
        """
        Based on a linkage matrix it returns a DataFrame with articles and
        their corresponding cluster_ids.
        :return: a DataFrame with the assignments
        """
        clusters = fcluster(self.linkage_matrix, self.max_d,
                            criterion='distance')
        assignments = pd.DataFrame({'cluster_id': clusters})

        assignments = assignments.join(self.category_df, lsuffix='_caller',
                                       rsuffix='_other')

        return assignments

    def get_clusters(self, chain):
        """
        Returns the corresponding ground truth cluster id's based on a chain
        of articles
        :param chain: A chain of article id's
        :return: the cluster-ids that the elements in the chain has been
        assigned to
        """

        cluster_ids = []
        for article_id in chain:
            try:
                article_row = self.assignments[self.assignments._id ==
                                               article_id].iloc[0]
                c_id = article_row.cluster_id
            except Exception as e:
                print('Article id {0} does not exist in category {1}'.format(
                    article_id, self.category))
                print(e)
                raise

            cluster_ids.append(c_id)

        return cluster_ids
```

C.4 experimenter.py

The file that runs all our experiments. Can also be seen as the controller of our model. Described in more detail in Section 4.2 and Chapter 4.

```python
import ground_truth_getter
import article_linker
import features
import pickle_handler
import analyzer
import pandas as pd
import datetime
import sys

__author__ = "Madeleine Appert & Lisa Stenberg"


max_ds = {('GENV', 'average', 'cosine'): 0.75,
          ('GENV', 'single', 'cosine'): 0.4,
          ('GDIS', 'average', 'cosine'): 0.8,
          ('GDIS', 'single', 'cosine'): 0.4,
          ('GCRIM', 'average', 'cosine'): 0.75,
          ('GCRIM', 'single', 'cosine'): 0.4}

linkage_methods = ['average', 'single']
metrics = ['cosine']


class Experimenter:

    def __init__(self, category):
        self.category = category

        self.al = article_linker.ArticleLinker(category)
        self.analyzer = analyzer.Analyzer()

        # Will contain data from runs, where one run looks like exp_run
        # exp_run = {(linkage_method, metric, features):
        # pd.DataFrame['article_id', 'cluster_id']}
        self.results = {}

    def run_experiments(self, tracking_method='track_one_word',
     ↪  time_limit=None):
        """
        Runs all experiments
        """
        print('################################')
        print('## Running experiments on category {0}'.format(self.category))

        if tracking_method == 'track_one_word':
            print("## Tracking on ONE word.")
            print('## Timedelta: {0}'.format(time_limit))
            print('################################')

            tracking_words = self.al.get_tracking_words(100)
            self._track_one_word(tracking_words, time_limit)
            pickle_handler.pickle_object(
                "ResultsNER_{0}1Word{1}Words_Timedelta{2}".format(
                    self.category, self.al.no_wanted_words, time_limit.days),
                self.results)

        if tracking_method == 'track_two_words':
            print("## Tracking on TWO words.")
            print('## Timedelta: {0}'.format(time_limit))
            print('##############################')

            tracking_words = self.al.get_tracking_tuples()
            self._track_multiple_words(tracking_words, time_limit)
            pickle_handler.pickle_object(
                "ResultsNER_{0}2Word{1}Words_Timedelta{2}".format(
                    self.category, self.al.no_wanted_words, time_limit.days),
                self.results)
        # Currently doing analysis in total analyser post run.
        # self.analyzer.analyze_experiments(self.results)

    def _compare_chain_to_ground_truths(self, chain_tuple, fts, values,
                                        chain_number):
        """
        Compare a chain to all different kinds of linkage methods and metrics.
        :param chain_tuple: A tuple containing a list of article IDs and a
        list of corresponding dates.
        :param fts: Features used to created chain. Needed to be able to save
        run settings
        :param chain_number: In case chain was split up, int to differentiate
        between them.
        """
        chain, dates = chain_tuple
        for linkage_method in linkage_methods:
            for metric in metrics:
                settings = (linkage_method, metric, (fts, chain_number),
                 ↪  values)

                # Init ground truth getter
                max_d = max_ds[(self.category, linkage_method, metric)]
                gt = ground_truth_getter.GroundTruthGetter(self.category,
                                                           max_d,
                                                           linkage_method,
                                                           metric)

                # Get clusters for chain
                gt_assignments = pd.DataFrame({'article_id': chain,
                                               'cluster_id': gt.get_clusters(
                                                   chain),
                                               'datetime': dates})

                self.results[settings] = gt_assignments

    def _track_one_word(self, tracking_words, timedelta=None):
        """
        Link articles together based on one word.
        """
        list_of_tuples = []
        total = len(tracking_words)
        skip = 0
        for w, values in tracking_words.items():
            fts = features.Feature(content=[w], time=timedelta)
            skip += 1
            sys.stdout.write("Remaining: {0}    Testing feature: {1}".format(
                total - skip, fts))
            sys.stdout.flush()
            sys.stdout.write("\r")

            chain_tuples = self.al.get_chain_tuples(fts)
            list_of_tuples.append((w, values, chain_tuples))
            for chain_tuple, chain_number in zip(chain_tuples, range(len(
                    chain_tuples))):
                self._compare_chain_to_ground_truths(chain_tuple, fts,
                                                     tuple(values),
                                                      ↪   chain_number)
        pickle_handler.pickle_object(
            "Chains{0}1Word{1}Words_Timedelta{2}".format(self.category,

                                                          ↪   self.al.no_wanted_words,
                                                         timedelta.days),
            list_of_tuples)

    def _track_multiple_words(self, tracking_words, timedelta=None):
        """
        Link articles together based on multiple words.
        """
        print("Tracking tuples: {0}".format(len(tracking_words)))
        skip = 0
        start_time = datetime.datetime.now()
        count = len(tracking_words)
        list_of_tuples = []
        for (word1, word2), values in tracking_words.items():
            skip += 1
            fts = features.Feature(content=[word1, word2], time=timedelta)
            sys.stdout.write("Remaining: {0}    Testing feature: {1}
             ↪   ".
                             format(count - skip, fts))
            sys.stdout.flush()
            sys.stdout.write("\r")

            chain_tuples = self.al.get_chain_tuples(fts)
            if not chain_tuples == []:
                list_of_tuples.append((fts, values, chain_tuples))
                for chain_tuple, chain_number in zip(chain_tuples, range(len(
                        chain_tuples))):
                    self._compare_chain_to_ground_truths(chain_tuple, fts,
                                                         values, chain_number)
        end_time = datetime.datetime.now()
        print("Got all chains for 2 words! Took: {0}".format(end_time -
                                                             start_time))
        pickle_handler.pickle_object(
            "Chains{0}2Words{1}Words_Timedelta{2}".format(
                self.category,
                self.al.no_wanted_words,
                timedelta.days),
            list_of_tuples)
```

C.5 random_linker.py

The random model which is described in Section 4.6.1.

```
import numpy as np
import pandas as pd
import pickle_handler

__author__ = "Madeleine Appert & Lisa Stenberg"


class RandomLinker:
    def __init__(self, ids, file_name):
        """
        Firstly we calculate a simple empirical probability mass function for
        chain length, which is based on the actual results of one experiment
        and one data set. Based on that pmf we follow the following steps:

        (1) Let N be the number of chains to sample. We choose N to be the same
        number of chains as generated from the experiment. Repeat step 2 and
        3 with replacement.
        (2) Draw a chain length l using the pmf.
        (3) Randomly draw l articles from the entire data set (ids), without
        replacement.

        :param ids: List of article ids.
        :param file_name: Name of results file
        """
        self.ids = ids
        df = pickle_handler.load_from_pickle(file_name)

        # Get the empirical pmf for chain length
        if 'chain_length' in df.columns:
            # Needed because we have different names for Tracking one/two
            # words and time feature
            chain_lengths = pd.Series.to_frame(df.groupby(['chain_length'])
                                                ['chain_length'].count())
        else:
            chain_lengths = pd.Series.to_frame(df.groupby(['doc_freq'])
                                                ['doc_freq'].count())
        chain_lengths.columns = ['count']
        total_nbr_of_chains = chain_lengths['count'].sum()
        chain_lengths['probability'] = chain_lengths['count'].apply(
            lambda c: c / total_nbr_of_chains)

        # Based on pmf - draw chain lengths
        rand_chain_lengths = np.random.choice(
            list(chain_lengths.index), total_nbr_of_chains,
            p=list(chain_lengths['probability']))

        self.chains = []
        # For every chain, draw len(chain) articles without replacement
        for chain_length in rand_chain_lengths:
            self.chains.append(np.random.choice(self.ids, chain_length,
                                                replace=False))
```