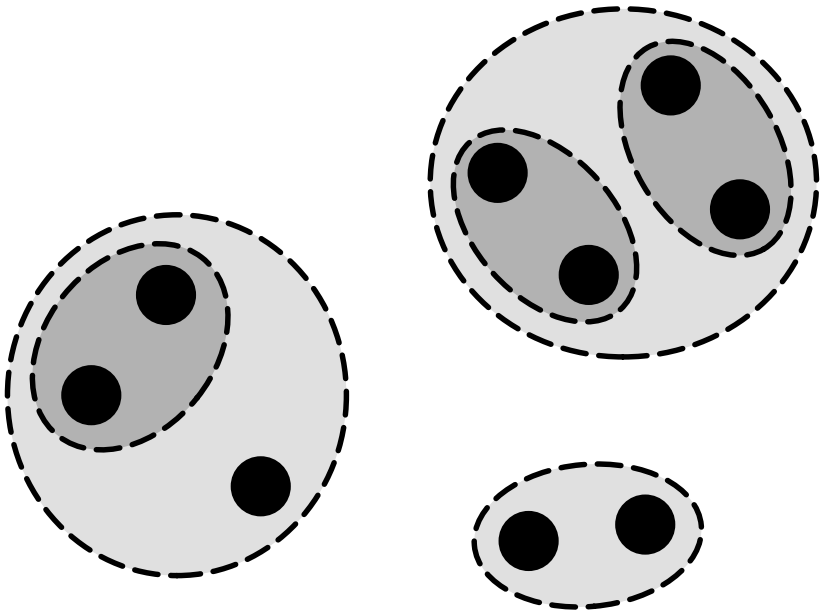
CHALMERS



Large scale news article clustering

Master of Science Thesis
Computer Science: Algorithms, Languages and Logic

MARCUS LÖNNBERG
LOVE YREGÅRD

Chalmers University of Technology
Department of Computer Science and Engineering
Gothenburg, Sweden June 2013

Large scale news article clustering

MARCUS LÖNNBERG
LOVE YREGÅRD

Examiner: PETER DAMASCHKE

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover: Objects in an hierarchical clustering.

Department of Computer Science and Engineering
Gothenburg, Sweden June 2013

Abstract

In this thesis we examined different approaches on how to cluster news articles so that two articles which are covering the same information would belong to the same cluster. We examined already existing algorithms and pre-processing steps as well as developed our own. Our requirements were that the algorithm should be able to handle a vast amount of articles, produce clusters of high quality and do this in a short amount of time.

We managed to come up with an algorithm which was quite fast and could produce clusters of high quality. We also developed two different optimization methods in order to speed up the clustering algorithms even more. We found that these methods improved the runtime performance greatly for two of the algorithms while the cluster quality was not significantly affected.

Sammanfattning

I denna rapport undersökte vi olika angreppssätt på hur man skulle kunna klustra ihop nyheter så att två artiklar som förmedlar samma information skulle tillhöra samma kluster. Vi undersökte redan existerande algoritmer och förbehandlingssteg samt utvecklade egna metoder. Våra krav var att algoritmen skulle kunna hantera en stor mängd artiklar, producera kluster av hög kvalitet och kunna göra detta på kort tid.

Vi lyckades utveckla en egen algoritm som var någorlunda snabb och kunde producera kluster av hög kvalitet. Dessutom utvecklade vi två olika optimeringsmetoder för att ytterligare öka klustringhastigheten. Vi fann att dessa metoder kraftigt förbättrar tidsprestandan för två av algoritmerna medan klustringkvaliteten inte påverkades nämnvärt.

Acknowledgements

We would like to thank all employees at Squeed who have supported us during the project. In particular we want to thank our supervisor Peter Lindh who have been a helpful hand during the entire project. We also would like to thank our examiner and supervisor Peter Damaschke for taking us on and having helpful discussions with us.

Contents

List of Figures

List of Tables

List of Algorithms

1

Introduction

In our modern society we deal with a lot of information from different sources on the Internet. News articles is a common source of information, which can be found on different news sources such as online newspapers, blogs or other types of news websites. However, several news sources may cover the same news category and consequently publish similar articles, i.e. they cover the same information.

For a person who frequently reads the articles from at least two news sources it would be convenient if all those sources' articles could be read at a single location. It would also be preferable in the case with similar articles that only one of them are presented to the reader. Furthermore, it would also be desirable if the reader could access all similar articles easily. This would be beneficial e.g. for source criticism and finding further information about the same topic.

To better describe what we mean with similar articles we consider two cases. When it comes to news articles, they should cover the same event and describe it with similar information, i.e. their information should not differ strongly. When articles do not cover a certain event, for example a review, an opinion or a discussion, then articles should be considered as similar if they share a very similar viewpoint about the same topic.

The objective of this project is to create a system where a user can read news articles from multiple sources and where similar articles are grouped together. The system should only present one article from each group, but at the users request it should also present the other articles from a group.

The system we intend to build should support many users, where each user can specify which news sources the user wants to follow. An advantage with this approach is that all users will contribute with their own sources which will help create larger groups of similar articles, and thereby users may find similar articles from sources they are not following.

In order for users to rely on the system it must produce groups of articles that are flawless. Therefore it is important that the system creates groups of articles that follow our definition of similar articles and do so consistently.

Since the system should handle many users there will also be many sources and consequently many articles. It is therefore important that our system can process a vast amount of articles. It is also crucial that this can be done at a high rate so there is only a short delay between the time of publication and the time the articles are grouped in the system.

The concept of grouping text documents (in our case news articles) together belongs to the field of document clustering. There are several different ways to cluster documents and a few of them will be investigated in this project.

1.1 Project aims

The overall goal with the project is to develop a system that functions as previously described. However, this thesis will only focus on the clustering processes that can be used within the system. Our intention is to examine already existing algorithms and if we deem it necessary and possible we will also develop our own. Most decisions within the project will be taken with the system in consideration.

Regarding the clustering process, our goal is to find a solution that creates clusters with high quality and does so with a good runtime performance. With high quality we mean that the clusters should follow our definition of similar articles to a very high degree and there should be few occurrences of misplaced articles.

For the runtime performance we have a soft goal of being able to cluster 100 000 articles in a short amount of time (less than an hour). This should be doable on a normal computer with inexpensive hardware.

A limitation in this thesis is that we will only work with articles which are written in English.

1.2 Method

This project have been carried out at the company Squeed in Gothenburg. The original idea behind the project originates from a few years ago when Marcus Lönnberg, one of the authors of this thesis, thought that it was both tedious and redundant to see similar articles from different news sources when following many RSS feeds. He thought that it must be possible to build a solution that eliminates this problem. The idea was then both expanded and limited in order to be suitable as a master's thesis.

In the beginning of the project we began studying relevant theory, starting with text representation, similarity measurements and a few clustering algorithms. While the research was performed we implemented methods and algorithms that we thought could be useful for the project. This led to the discovery that some pre-processing was necessary in order to get clusters of high quality. Thus, we also studied various pre-processing methods.

While we were focusing on clustering algorithms we realized that most of the examined algorithms made large amounts of comparisons between articles. This made us

focus on incremental approaches where all articles not have to be available from start but can be added over time.

We examined some incremental clustering algorithms which produced clusterings of good quality. However, they were not as fast as we would like it to be. Therefore, we developed our own algorithm which we named Incremental Clustering Algorithm. Our intentions were that it would be both incremental and simple.

We also started thinking about possible ways to reduce the number of comparisons between the articles. Inspired by centroids from the k-means algorithm and deep knowledge about the Cosine similarity measurement, we tested various ideas for reducing the number of comparisons. This evolved to two different approaches which turned out to work great in combination with the Incremental Clustering Algorithm.

During the entire project, extensive testing was carried out in order to see which methods that worked and which did not. At the end of the project we chose to redo a selection of the tests in order to make a fair comparison of them in this thesis.

1.3 Thesis outline

The rest of this thesis is outlined as follows. Chapter 2 describes how texts can be represented in order for a computer to be able to compare them. In chapter 3 we examine different pre-processing steps which are used to improve the text representation. Chapter 4 covers several existing clustering algorithms and we also present a new algorithm. Chapter 5 presents our reduction methods that is used to speed up the clustering algorithms. Chapter 6 presents our test results for various pre-processing steps and the algorithms with and without our reduction methods. In chapter 7 we draw conclusions about our work and present future work. Finally, appendix A contains the full details of our test results.

2

Representation and similarity measures

In order to be able to divide a set of articles into clusters we need a method which allows us to compare them. Unfortunately two articles can not be compared directly but must be represented in a way so that a computer will be able to do these comparisons. In this chapter we will present how to represent the articles in order for a computer to compare them as well as several comparison methods.

2.1 Representation of articles

There are several ways one can represent a text so that a computer can compare them. Throughout this thesis the texts are represented using the *vector space model* which is a widely used model for representing texts [1]. A text is, with this representation, modeled as an n-dimensional vector, where n is the number of unique terms through the set of texts which is to be clustered [1]. Each component of the vector corresponds to one of the unique terms. As for now, each component contains the number of occurrences of the corresponding term in the text. From here on, the *document vector* created from document d will be denoted as $\vec{W}_d$.

However, there is a flaw with this representation model. By representing a text as a *bag of words* where only the number of occurrences for each unique word is saved, the context in which the word was found is lost. For example, both "The rabbit is faster than the turtle" and "The turtle is faster than the rabbit" will have exactly the same representation even though they do not express the same thing. But as long as two texts do not contain exactly the same terms as well as exactly the same term frequency they will not have the same representation.

2.2 Similarity measures

With the text representation model presented in the previous section, we now want to be able to measure how similar articles are. There exists several methods to do so and a few of them will be presented in the following subsections.

2.2.1 Euclidean distance

The Euclidean distance is the "ordinary" distance between two points [2]. E.g. the distance between two points on a two-dimensional plane or two points in a three-dimensional space. In the case with text clustering one usually deals with more than just three dimensions (if not always, since the number of dimensions is equal to the number of different words in the corpus). Given two document vectors, $\vec{W}_a = (a_{t_1}, a_{t_2},...,a_{t_n})$ and $\vec{W}_b = (b_{t_1}, b_{t_2},...,b_{t_n})$, the Euclidean distance is calculated in the following way [2]:

$$D_E(\vec{W}_a,\vec{W}_b) = \sqrt{\sum_{i=1}^{n}(a_{t_i} - b_{t_i})^2}$$

where t_1 to t_n represent all terms in the corpus and a_{t_i} and b_{t_i} represent the weight for term t_i in $\vec{W}_a$ and $\vec{W}_b$ respectively. The Euclidean distance is a non-negative number, with 0 representing that the vectors are equal and a number greater than 0 represents the level of dissimilarity.

Euclidean distance is a simple similarity measure but performs poorly in comparison to Cosine similarity and extended Jaccard similarity [3] which will be described in the coming sections.

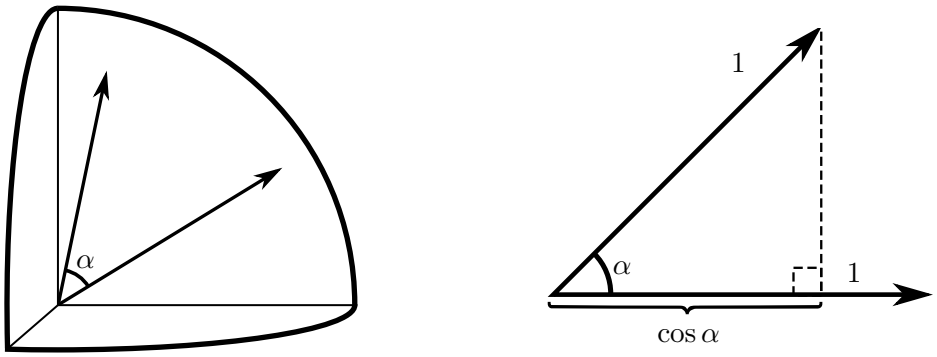
2.2.2 Cosine similarity

When using the Cosine similarity measure one assumes that the direction of the document vectors is more important than their length and the distance between them [4]. Also Cosine similarity is one of the most commonly used similarity measures for text clustering [2] and have been shown to be (together with the extended Jaccard similarity measure) the best similarity measure to "capture human categorization behavior" for high-dimensional data [3].

This method involves, as the name reveals, the cosine of the angle between the two term vectors. A three-dimensional case is shown in figure 2.1a where each axis represents the weight of a unique term and the vectors are displayed according to their term weights. The angle α between the vectors is then calculated and the cosine function is applied to α as shown in figure 2.1b.

Given two document vectors, $\vec{W}_a$ and $\vec{W}_b$, the Cosine similarity is calculated as follows [2]:

$$sim_C(\vec{W}_a, \vec{W}_b) = \frac{\vec{W}_a \bullet \vec{W}_b}{\|\vec{W}_a\|\|\vec{W}_b\|}$$



(a) Two document vectors. **(b)** The cosine value of α. The vectors are of length 1.

Figure 2.1: Cosine similarity in three dimensions.

where the numerator is the dot product of the vectors and the denominator is the product of the Euclidean norm of each vector. The Euclidean norm for $\vec{W}_d$ is defined as:

$$\|\vec{W}_d\| = \sqrt{\vec{W}_d \bullet \vec{W}_d}$$

where the inner product is the dot product.

The value of similarity is bounded by [0,1], with 1 meaning that the vectors are pointing the exact same direction and 0 means that they are totally independent of each other, that is, they do not have any words in common.

2.2.3 Extended Jaccard similarity

To better understand the extended Jaccard similarity we start by presenting the binary Jaccard coefficient. The input to this similarity measure is two sets, S_a and S_b, and the binary Jaccard coefficient measures the proportion that overlaps between these sets [5]. The binary Jaccard similarity is defined as [6]:

$$sim_{BJ} = \frac{|S_a \cap S_b|}{|S_a \cup S_b|}$$

That is, the number of shared element in the sets divided by the total number of different elements.

But in the case of text clustering one will most probably not have binary vector components in the document vectors. The similarity measure must be modified in order to accept vectors with non-negative real numbered vector components. This similarity measure is known as the extended Jaccard similarity measure and is calculated in the following way [2]:

$$sim_J(\vec{W}_a, \vec{W}_b) = \frac{\vec{W}_a \bullet \vec{W}_b}{\|\vec{W}_a\|^2 + \|\vec{W}_b\|^2 - \vec{W}_a \bullet \vec{W}_b}$$

The extended Jaccard similarity method compares the weight of shared terms against the weight of terms which are present in only one of the articles [2]. Note that the modified similarity measure will produce the same result as sim_{BJ} when the input vectors only consist of binary vector entries [5].

As in the case with Cosine similarity the result from sim_J is a value bound by [0,1], with 1 meaning that the vectors are equal and 0 that they have no terms at all in common [2].

2.2.4 Same source penalty

To our knowledge it is very uncommon that a news source publishes two or more articles about the same event with the same information, i.e. duplicate articles. If this is assumed to generally be true, then articles that seem similar and are from a single source is either from two separate events or less likely is the same event where one of the articles is updated.

Based on this, a penalty can be added to the similarity measure between two articles from the same source. The advantage with this is that the possibility that such articles are clustered together is reduced. However, the penalty should be low enough so that articles that are either corrections or only slightly updated can be clustered together.

2.2.5 Publication date difference penalty

Too further distinguish between events that are similarly described in articles we have successfully tried a method where we penalize similarity measurements between two articles that are published long time apart. The idea behind this is that events are usually covered by media just within a few days after an event has occurred.

Based on this, we have constructed a penalty that increases the longer time it is between two articles publishing dates. The penalty is not active until the difference is more than 2 days and after that increases linearly until a 40 % reduction in the similarity measure is reached after 14 days where it flattens out.

If media mentions the same event again in an article it is typically with more information which we should distinguish from the already clustered articles.

2.3 Normalization

In section 2.2.2 the Cosine similarity measure was described. As it only makes use of the direction of the vector, we will in the following subsection describe how to normalize the vectors in order to speed up the similarity calculations.

2.3.1 Normalization when using Cosine similarity

As mentioned in section 2.2.2, when using the Cosine similarity method one assume that the direction of the document vectors is more important than their length [4]. To speed up the calculation of this method one can normalize the document vectors and represent

them as *unit vectors* (i.e. vectors of length one). The document vector for document d, $\vec{W}_d$, is normalized by dividing each vector component with the value of its Euclidean norm. In short, the vector is normalized using the following calculation:

$$\vec{W}_{nd} = \frac{\vec{W}_d}{\|\vec{W}_d\|}$$

where $\vec{W}_d$ represents the original vector and $\vec{W}_{nd}$ represents the corresponding normalized vector.

With normalized input vectors, it is easy to realize that the Cosine similarity can be computed as follows:

$$sim_C(\vec{W}_{na}, \vec{W}_{nb}) = \vec{W}_{na} \bullet \vec{W}_{nb}$$

which does not result in a different value than the original computation, but solely results in a performance boost for the clustering algorithm if similarity measures are calculated frequently.

3

Pre-processing

In the last chapter we introduced how one can represent a text as a vector and how a computer is able to compare them. Unfortunately the representation, as it was presented in section 2.1, is quite poor. Some words may not be as important as other in order to distinguish one text from another. Also, some words may be closely related and should be treated as equal. These and some other problems will be discussed in this chapter.

3.1 Stop words

Stop words are words that are removed before processing texts. In our case we remove words that are frequently occurring in texts and which also not carries any information on their own. Some common stop words in the English language are: *an, and, by, for, from, of, the, to* and *with*.

In document clustering these kind of words are removed from the corpus in order to give both a better and faster clustering [7]. The reason why it gives faster clustering is because the number of dimensions are reduced and consequently the similarity measurements (as described in section 2.2) will have to do fewer calculations. The quality of a clustering can be improved since documents will not be clustered simply because they contain words such as "the" or "to".

One should be aware of that using stop words also can have a negative effect on the clustering. For example when the words "it" and "us" are classified as stop words in documents containing words like "IT specialist" or "US government". In such cases important information could potentially be lost, since both "IT" and "US" are information-bearing words [8].

3.2 Stemming

When working with the vector space model (as described in section 2.1) each text is represented as a *bag of words*. However, the words found in a text often have several morpholigical variants [9] (a noun often have different forms when used in singular and plural and a verb often have different form depending on tense, such as *jump* and *jumped*). Each variant of a word is treated separately even though they can be considered as equal in the case of clustering.

When calculating the similarity between two articles (as described in section 2.2) the algorithm will see e.g. *computer* and *computers* as two completely different words. Consequently these words will not contribute to that two articles (one with many occurrences of *computer* and the other with many occurrences of *computers*) will be treated as similar. We have the same problem with words like *walk* and *walked*, *happy* and *happiness*, and *John* and *John's* (possessive form) as well.

To solve the presented (and other) problems, stemming can be applied to a word. In the English language the stem of a word is written to the left followed by zero or more suffixes [10] (as in the case with with computers). In some cases the stem is also altered when the word changes form (as in the case happy-**happi**ness), but in those cases it usually occurs on the right hand end [10]. The stemming process' task is to remove these suffixes.

A prefix can also be added to a word, but this will often change the meaning of the word [10] (as in happy-**un**happy) so those will not be removed in the stemming process.

The most commonly used stemmer which strips English words of its suffixes is the Porter stemmer [6]. The inventor's paper about the algorithm (Martin Porter's An algorithm for suffix stripping, 1980) has been cited over 6000 times [11].

In this project we have chosen to use another stemmer[1] by Martin Porter which after quick and shallow testing seemed to be better than the original Porter algorithm for our purpose. E.g. the word *nightly* was stemmed to *night* with the alternative stemmer and to *nightli* by the Porter stemmer. Also the following words is stemmed to the same word by the alternative algorithm but not by the Porter stemmer:

$$\left.\begin{array}{l}\text{proceed}\\ \text{proceeded}\\ \text{proceeding}\\ \text{proceedings}\end{array}\right\} \xrightarrow{stem} \text{proceed}$$

The Porter stemmer stemmed *preceed* to *proce* which is clearly not to our advantage. The fact that some stemmed words are not "real" words is not a problem in the case of clustering as long as words with equivalent meaning are stemmed to the same word.

Problems that the stemming process will not solve is to group synonyms (different words that have the same meaning such as *annual* and *yearly*) together. Also by removing the suffixes there is a chance that homonyms (a word which may have different

[1] http://snowball.tartarus.org/

meaning) are introduced [10]. E.g. both *politics* and *polite* are both stemmed to *polit*. Also *computer* and *compute* are stemmed to *comput* even though the original words necessarily not have anything to do with each other.

Another problem is some irregular verbs (such as *go*, *went*, *gone*) which will not be stemmed to the same word. Neither will the plural form of some words be stemmed to the same word as the singular form of the word. An example is *wife* $\xrightarrow{stem}$ *wife* and *wives* $\xrightarrow{stem}$ *wive*.

How the stemming is done will not be covered here as it lies outside the scope of this thesis.

3.3 Pruning of rare words

Pruning of rare words means that if a (stemmed) word appears less frequently than a certain pre-defined threshold throughout the *corpus* (a collection of documents) which should be clustered the term should be removed. The idea behind this is that rare terms does not contribute very much to the similarity, consequently they do not help finding appropriate clusters [12].

It has also been discussed that even if the rare words help distinguish a document from another, the separation would probably be too fine to be useful. This would result in several clusters that only contain a single or two documents [13].

Pruning based on the number of documents a certain word appears in have been shown not to affect the clustering result very much at all [12].

3.4 Weighting of words

Manning et al. [6] discussed that a term which occurs twenty times in an article is most likely not twenty times more significant than a term which is only found once. In order to achieve a good clustering, the words of the articles will need to be weighted. One way to achieve this is by using *tf-idf* (term frequency - inverse document frequency) weighting.

The term frequency, $tf_{t,d}$, of term t in document d can in a very simple case be expressed as the number of occurrences of the term t in d, but in such a case the problem described above will still remain. A common modification is to use the logarithm of the term frequency instead of the original term frequency itself [6]. That is:

$$tf_{t,d} = \begin{cases} 1 + \log(tf'_{t,d}) & \text{if } tf'_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $tf_{t,d}$ is the newly weighted value and $tf'_{t,d}$ is the "true" term frequency.

However, just using the term frequency is not always enough. The problem is that all different terms have the same weight. A specific term which is found in most of the articles in the corpus will probably not help the algorithm to distinguish an article from another. This is very similar to the case of stop words where it is already known that

those terms will not help separating articles from each other. Thus, the significance of terms occurring in many articles should be lower than for other words.

Also, two articles sharing a not very common term tend to be more similar to each other than two articles sharing a more common term. Consequently one can argue that these words are more important and therefore should have a higher weight.

This can be achieved by using the inverse document frequency. It has been suggested that the inverse document frequency for term t can be calculated using the following formula [6]:

$$idf_t = \log \frac{N}{df_t}$$

where N is the total number of documents in the corpus and df_t is the number of documents which contains the term t at least once. The inverse document frequency contributes to that a common term receives a low weight while an uncommon term receives a higher weight.

With known term frequency and inverse document frequency it is now possible to combine them and produce a new weight for each term in each document. The document vectors can by that be reweighted as:

$$tf\text{-}idf_d = (tf_{t_1,d} \times idf_{t_1}, tf_{t_2,d} \times idf_{t_2}, ..., tf_{t_v,d} \times idf_{t_v})$$

where d is a document in the corpus and t_1, t_2, ... , t_v represent the terms found in the corpus. For convenience, in the remainder of this thesis, the document vector for document d will be denoted $\vec{W}_d$ regardless of the selected weighting method.

For the case which is examined in this thesis where articles arrive continuously, there will probably not be possible to set a new weight for all words in all articles as new articles arrive (the inverse document frequency is calculated using all articles). In such a case only the term frequency for each article may be used (as it only depends the corresponding article itself). However, this might not be a problem. Schütze and Silverstein [7] mentions that preliminary studies indicated that the inverse document frequency even could impair the cluster quality.

Another way to weight the terms in an article could be by using the information about where in the article a term is found. The title have a lot of information about the article, thus giving the words found there higher weight would probably be advantageously for the clustering step.

Further is the inverted pyramid story structure a common way to structure an article [14]. When using the inverted pyramid story structure one puts the most newsworthy information of an article in the beginning and the details at the end. With such a story structure it would probably be preferable to give terms located in the beginning of an article higher weight than those located at the end.

3.5 Truncation

As discussed in section 2.2, all terms in two document vectors which shall be compared are used to calculate the similarity. In order to reduce the time the similarity calculation require we can remove the least significant terms from each document vector, or rather, we keep the t most significant terms from each document vector. By removing a term we mean setting the weight for the term to 0 in that specific document vector.

It has been discussed that truncation of document vectors will not give any significant speed boost as the vectors rarely contain a lot of different terms [7]. However, it has also been shown that the clustering quality is not reduced when truncating the vectors. The quality of the clustering was preserved when taking only the 50 most significant terms into account when performing the similarity calculations [7].

3.6 Named entities

The similarity between articles for our problem should in the best case rely on the information the articles contains. But as the vector space model is used, the similarity instead depends on which words are used to relay the information. In an attempt to combine information extraction from articles together with bag of words, we have used named entity recognition (NER).

Named entities are parts of a text that have been classified into predefined categories such as organizations, persons, places, dates, quantities, etc [15]. In the below example entities for a person, an organization and a location have been highlighted.

Barack Obama was previously a member of the **Senate**, but is now the president of the **United States of America**.

We have included Stanford Named Entity Recognizer as a part of our pre-processing. The model we used for the NER system classifies each word as either a regular word or a part of a location, person or organization [16]. Entities of those types are often represented by several words and for this reason does our solution collect all subsequent words of the same entity type in order to create an entity.

Our approach to handling entities is to treat them as if each entity were a regular word, even if the entity consists of several words. The count is adjusted so entities should gain a high weight and therefore have more effect on the similarity. In the case where there are two identical words, where one of them is marked as an entity and the other as a regular word, we distinguish them from each other and they are treated as separate words.

As an example, consider a text where the location *US* and the word *us* appears. Even though we ignore case they will not match, provided that *US* is correctly marked as an entity.

4

Clustering algorithms

In the previous chapters we have learned how to create a good representation of a text and also how to compare these representations in order to find how similar two texts are. However, we are still missing a method how to divide a collection of texts into different clusters. In this chapter we will explain some already known clustering algorithms which performs this task as well as present a new one. The clustering algorithms presented in this chapter is not specific for text clustering, so when cluster objects are mentioned one can think of articles instead.

4.1 k-means

k-means is perhaps the most well known clustering algorithm. This algorithm does not only need the objects which are to be clustered, it also requires a value k which is the number of clusters we want the algorithm to produce. It uses a partitioning approach and starts with initializing k *centroids* by placing them in the term space [6]. A centroid is thus, in this case, just another document vector but with no underlying document, where each vector component is given a weight. This given weight could be a random value which means that the positioning of the centroid in the term space also would be random.

After the initializing of the centroids have been done, a process begins where all documents are compared to all centroids. Each document is then linked to its closest centroid, according to the Euclidean distance (see 2.2.1). When all documents have been linked, all centroids are moved so they are centered with respect to its linked documents by using the following equation:

$$\vec{\mu}(\Omega) = \frac{1}{|\Omega|} \sum_{x \in \Omega} \vec{W}_x$$

where $\vec{\mu}(\Omega)$ represents the new position of centroid $\vec{\mu}$ which is linked to the set of

documents Ω.

This process is repeated until some stopping criteria is met, which typically is when no document links to a new centroid between iterations. That is, each object is compared to each new centroid and linked to the closest one. The centroids position is then once again updated, if no document is linked to a new centroid we are done.

After the last iteration has been done the resulting clusters are formed by the linked documents for each centroid. The pseudocode for k-means is shown in algorithm 1.

Algorithm 1: k-means

Data: A set of clusterable objects and the number of clusters which should be produced, k

Result: k clusters containing the input objects

```
for i ← 1 to k do
    Initialize centroid μ_i with random values for all dimensions of the input objects
end
while stopping criteria is not met do
    For each centroid μ_i create an empty collection Ω_i
    For each object x_n find the closest centroid μ_j that minimizes the Euclidean distance and store the object in Ω_j
    Each centroid is recomputed using μ_i(Ω_i) according to equation 4.1
end
Return all k Ω which is the resulting clusters
```

4.2 Hierarchical Agglomerative Clustering

Hierarchical Agglomerative Clustering (HAC) is a form of hierarchical clustering where a hierarchy of clusters is built in a bottom-up manner [6]. Firstly, the algorithm creates a singleton cluster for every input object. Then it iteratively merges the two most similar clusters into a single cluster. This procedure is repeated until there is only one cluster left. To determine the similarity between two clusters a linkage criterion is used (which soon will be discussed). The result is a tree structure where the root is the last cluster

and the leaves are the input objects. The pseudocode is shown in algorithm 2.

Algorithm 2: Hierarchical Agglomerative Clustering

Data: A collection of clusterable objects
Result: A tree structured hierarchy where the leaves represents the clustered objects

```
1 Place each objects in its own cluster and place the clusters in a list containing
  active clusters
2 while more than one active cluster do
3 |   Find the two most similar clusters according to a linkage criterion
4 |   Create a new cluster containing the two clusters
5 |   Remove the two clusters from the list of active clusters
6 |   Add the new cluster to the list of active clusters
7 end
```

It is possible to prematurely stop the algorithm when the similarity between clusters is too low, this is useful when the top of the whole hierarchy is not needed. In that case the result would be a set of clusters where each cluster is the root of a tree structure and each leaf is a single clusterable object, see example in figure 4.1.

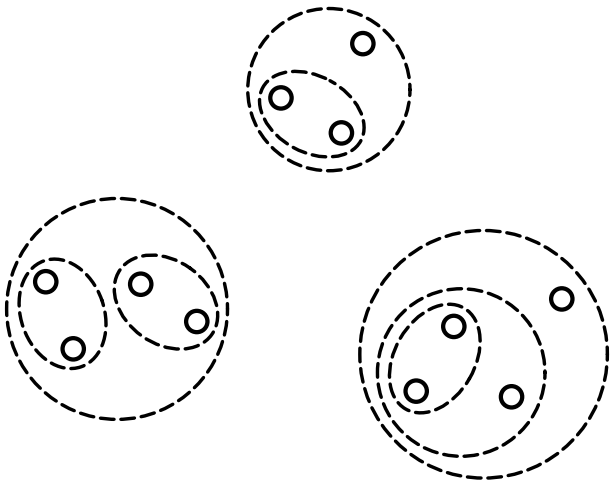


Figure 4.1: Clustering result of HAC that were prematurely stopped when the similarity between clusters were too low. The dashed circles represents clusters.

There exists several known linkage criteria which is used with the HAC algorithm. The following four methods uses different approaches to decide the similarity between two clusters given a similarity measure [1].

Single link

Single link, also known as nearest neighbor, works by finding the most similar pair of objects that are not yet in the same cluster and merging their clusters together [6].

Another way to describe the operation is that when comparing two clusters it only uses the similarity between the two most similar objects, one from each cluster, see figure 4.2.

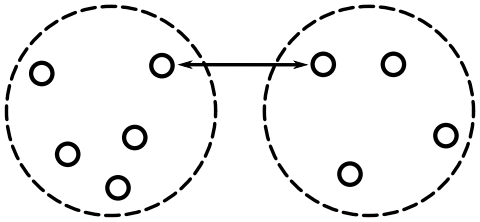


Figure 4.2: Similarity measurement between two clusters using single link.

The following formula shows how the calculations are done. C_i and C_j is clusters to be compared and sim is a similarity measure where a higher value means more similar.

$$sim_{SL}(C_i,\, C_j) = \max_{x \in C_i,\, y \in C_j} sim(\vec{W}_x,\, \vec{W}_y)$$

A possible downside with how single link works is that it can create chains of clusters, see figure 4.3. In such a case one may notice that the objects found at each end of the chain will not be very similar to each other. Therefore single link is unsuitable for isolating spherical clusters [6].

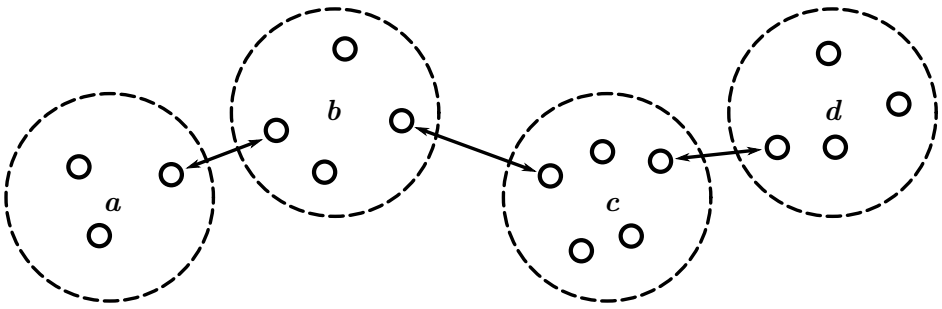


Figure 4.3: Chain created between the clusters a, b, c and d.

Complete link

Complete link works in a similar manner as single link, but instead of finding the maximum similarity between any two objects from different cluster, it instead finds the least similar objects from two clusters (see figure 4.4) and uses that value as similarity measurement [6]. The calculations are done as follows:

$$sim_{CL}(C_i,\, C_j) = \min_{x \in C_i,\, y \in C_j} sim(\vec{W}_x,\, \vec{W}_y)$$

A disadvantage of the complete link method is that a single element that deviates from the other objects in a cluster, an outlier, may have a big impact on the final outcome of a clustering [17].

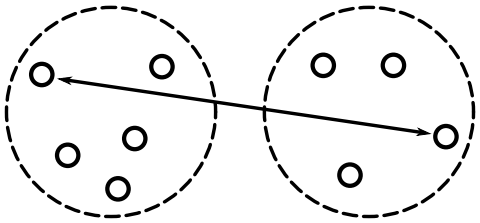


Figure 4.4: Similarity measurement between two clusters using complete link.

UPGMA

Unweighted Pair Group Method with Arithmetic Mean (UPGMA) is an approach where the similarity between two clusters is determined by the average similarity of all cross cluster object measurements [17]. The calculations for this method is shown in the follwing formula:

$$sim_{UPGMA}(C_i,\, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{y \in C_j} sim(\vec{W}_x,\, \vec{W}_y)$$

The most prominent advantage with this method compared to single and complete linkage is that it, as the name hints, takes an average of the similarities between the clusters and not only the similarity of single objects.

Group average

Group average also known as Group-Average Agglomerative Clustering (GAAC) is similar to UPGMA in the way that it computes an averaged value. However, their inner workings is very different. When comparing two clusters it computes the similarity between all objects in the clusters, including similarities between objects that are within the same cluster [6].

GAAC is defined as follows:

$$sim_{GA}(C_i,\, C_j) = \frac{1}{(|C_i| + |C_j|)(|C_i| + |C_j| - 1)} \sum_{x \in C_i \cup C_j} \sum_{y \in C_i \cup C_j,\, x \neq y} \vec{W}_x \cdot \vec{W}_y$$

4.3 Bisecting k-means

Bisecting k-means uses a combination of the two clustering types, hierarchical and partitional clustering [18]. Unlike the Hierarchical Agglomerative Clustering algorithm, the cluster hierarchy is built in a top-down manner when using this algorithm. The algorithm starts with all objects put in a single cluster which is then divided into two new clusters using the k-means algorithm (see 4.1) with $k = 2$. It then continues recursively

dividing clusters until a certain criteria is fulfilled. In the original algorithm the criteria is that only a certain amount of clusters should be created.

However, an augmented version of the algorithm that is tailored for document clustering have been proposed [19]. There are three areas of the algorithm that they have been augmented. A bootstrapping aggregating procedure which performs the k-means algorithm several times to decide the starting positions for the centroids based on the medians from the aggregating runs have been added.

The second area proposes that instead of bisecting the largest cluster it calculates the standard deviation of each cluster to pick the one that is most sparsely packed. The reason for this is because otherwise the algorithm will yield a result where all cluster sizes are very similar, which is not realistic for real world data.

The last modification is a criteria of termination that stops the algorithm when the similarity of clusters objects reaches a certain termination value, γ.

4.4 Dynamic Hierarchical Compact Algorithm

Just like the Hierarchical Agglomerative Clustering method is the Dynamic Hierarchical Compact Algorithm (DHCA) [20] also both agglomerative and, as the name reveals, hierarchical. This algorithm is, unlike the previously discussed algorithms, also dynamic. This means that the algorithm is incremental (new objects can be added over time) but also that the algorithm will produce the same result regardless which order the objects are added [21]. The other algorithms require all data to be known before the clustering starts.

The original algorithm (without the ability to update the clustering) is just called Hierarchical Compact Algorithm (HCA) [20]. It is based on graphs and makes use of three specific types of graphs called the β-similarity graph, the maximum β-similarity graph and the *U-max-S* graph.

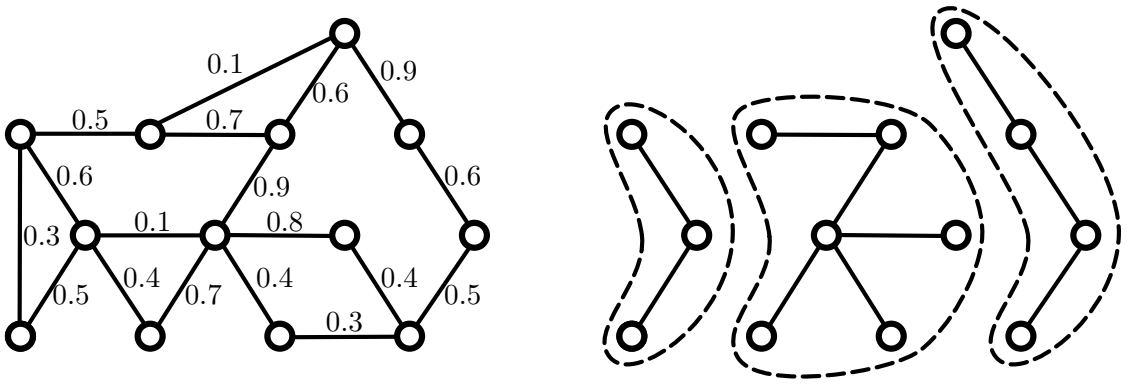
The β-similarity graph is an undirected graph where each vertex represents a cluster and where an edge exists between vertex i and vertex j if the similarity between the corresponding clusters is at least β, where β is a predefined value. The similarity between clusters are calculated using UPGMA linking (explained in section 4.2). A cluster is said to be a *β-isolated cluster* if its similarity to all other clusters is less than β. The maximum β-similarity graph and the *U-max-S* graph will soon be explained.

The algorithm works as follows: start with putting all objects in clusters on their own and create a β-similarity graph. In figure 4.5a one can see the 0.1-similarity graph for a collection of clusters. Following the definition of the 0.1-similarity graph, there are only edges between those vertices where the similarity is at least 0.1.

When the β-similarity graph is created, create a directed graph with the same vertices as the β-similarity graph. For each vertex, create an edge pointing to the vertex which it is most similar to according to the β-similarity graph. That is, the most similar vertex with a similarity measure of at least β. This newly created graph is the maximum β-similarity graph.

With the maximum β-similarity graph one can produce the *U-max-S* graph which is

basically the undirected version of the maximum β-similarity graph (all parallel edges are also removed).



(a) The 0.1-similarity graph with each weight shown.

(b) The *U-max-S* graph with the new clusters shown with dashed lines.

Figure 4.5: Hierarchical Compact Algorithm example.

Find the connected clusters in the *U-max-S* graph and combine them to new clusters (as shown in figure 4.5b). With the newly created clustes, create a new β-similarity graph. If all clusters are β-isolated the clustering is done, else the *U-max-S* graph will have to be produced again in order to cluster more. The pseudocode for this method is shown in algorithm 3 [20]. The clustering complexity of this algorithm is $\mathcal{O}(n^2)$ where n is the number of objects to be clustered [20].

Algorithm 3: Hierarchical Compact Algorithm (HCA)

Data: Clusterable objects
Result: Clusters in a tree structure where the leafs are the objects

```
Place all objects in their own clusters
level ← 0
Create the β-similarity graph, G_0, for the current clusters
while not all vertices in G_level are β-isolated do
    Create the U-max-S graph, U_level, using G_level
    Find the connected elements in U_level and combine them to new clusters
    level ← level + 1
    Create the β-similarity graph, G_level, for the newly created clusters
end
```

Updating the cluster hierarchy

The algorithm presented in algorithm 3 requires all objects, which is to be clustered, to be available in advance. However, it can be extended to handle adding and removing of objects. Given a hierarchy of clusters (calculated using the Hierarchical Compact Algorithm) and an element to be added (or removed), the clusters at all layers of the

hierarchy will need to be updated. The pseudocode for the updating step is presented in algorithm 4 [20].

Algorithm 4: Dynamic Hierarchical Compact Algorithm (DHCA)

Data: Clusterable object to be added (or removed) and a hierarchy of clusters previously built
Result: Clusters in a tree structure where the leafs are the objects

```
1  Place the new object in its own cluster (or remove the cluster in the lowest level)
2  level ← 0
3  Update the β-similarity graph, G_0
4  while not all vertices in G_level are β-isolated do
5  |  Update the U-max-S graph, U_level
6  |  Find the connected elements in U_level and update the clusters if necessary
7  |  level ← level + 1
8  |  Update the β-similarity graph, G_level, with the clusters created from U_level−1
9  end
10 Remove remaining levels greater than level from the hierarchy if such exists
```

The β-similarity graph at the lowest level is updated by adding the new element to a singleton cluster (or by removing the singleton cluster containing the element) and updating the edges. The *U-max-S* graph is then updated using the updated maximum β-similarity graph. Note that if an edge is removed from the *U-max-S* graph a cluster may no longer be connected and should therefore be split. Also if an edge is added and the two endpoints are in different clusters, these clusters should be merged.

With the *U-max-S* graph updated, the β-similarity graph for the next level can be updated using the updated clusters. This process will go on until all vertices in the β-similarity graph is β-isolated. If the process terminates before the top-level of the hierarchy is reached the remaining levels should be removed.

4.5 Dynamic Hierarchical Star

The Dynamic Hierarchical Star Algorithm [21] works in a similar manner as the Dynamic Hierarchical Compact Algorithm but creates clusters which may overlap. Having overlapping clusters means that one or more elements may belong to several clusters which may be desirable in some cases.

As with DHCA, we start with an algorithm which is not able to update the clustering. This algorithms will later be extended in order to handle new objects being added and removed. That algorithm is known as Hierarchical Star Algorithm (HSA) [22]. The pseudocode for HSA looks exactly like the algorithm for HCA with the exception that the combining step at line 6 in algorithm 3 differs. Instead of combining all connected elements in the *U-max-S* graph, HSA makes use of the minimum dominating set of the *U-max-S* graph.

The dominating set of a graph $G = (V,E)$ is a set D such that all vertices $v \in V$ is either in D or is adjacent to a vertex $u \in D$. Thus, the minimum dominating set is a set

D which is as small as possible. But this problem is NP-complete [23], hence we can not expect to find the best solution (in a reasonable time) but merely an approximation.

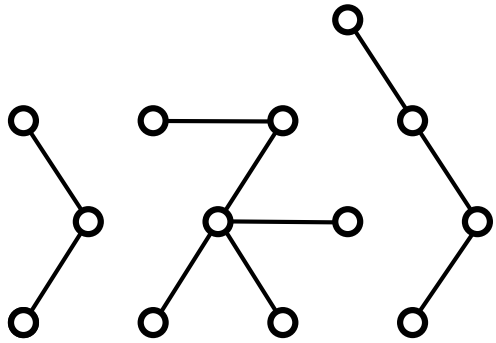
A cluster is created by the elements in the dominating set together with their adjacent vertices. This means that if a vertex is adjacent to more than one element in the dominating set, that vertex will also be a member of multiple clusters. A greedy approximation algorithm for creating as few clusters as possible have been proposed. The pseudocode for this algorithm is found in algorithm 5 [22].

Algorithm 5: Algorithm for creating clusters in HSA

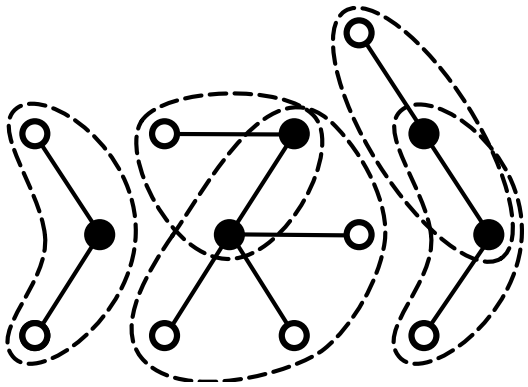
Data: An undirected graph, G
Result: The minimum dominating set of the given graph
while *not all vertices in G are covered* **do**
    Place all vertices in G with the greatest number of uncovered adjacent vertices in a set called M_0
    Put the vertices of M_0 with minimum degree in a set called M
    For each element in M create a cluster containing that element as well as its adjacent vertices. All these vertices are now considered covered
end
Remove all duplicate clusters

In figure 4.6a an illustration of an *U-max-S* graph is presented and the clusters created using the algorithm in algorithm 5 can be seen in figure 4.6b.

As one might see from the proposed algorithm each cluster is composed of a middle vertex (the star) and l vertices connected to it (the star's satellites). Note that l might be 0 in the case when there are no vertices connected to the star. With this structure, the chance that clusters will chain (creating a "long" cluster of elements) will therefore be lower than for other algorithms.



(a) The *U-max-S* graph which should be divided into clusters.



(b) The resulting clusters. The stars of the clusters are shown with black fillcolor.

Figure 4.6: The clustering step of HSA.

Updating the cluster hierarchy

The extension of HSA which is able to add and remove objects from the cluster hierarchy is called Dynamic Hierarchical Star (DHS) [21]. As in the case with DHCA, when new elements are added (or removed) all clusters at all levels of the clustering hierarchy will need to be updated. The pseudocode for this looks exactly like the one presented in algorithm 4 with the exception of line 6.

Instead of finding all connected vertices and updating our clusters (or creating new clusters) we want to update the current star cover which represents our current clusters. In this incremental approach it would be beneficial if this could be done without having to reset all stars in order to find the new stars in the updated *U-max-S* graph. The pseudocode that achieves exactly this is presented in algorithm 6 [21].

Algorithm 6: Updating the stars

Data: The new *U-max-S* graph, $U = (V, E)$, a set N containing added vertices, a set R containing removed vertices as well as two sets, NE and RE, representing added and removed edges respectively. The old stars are still marked in U

Result: The stars for U

```
1  Create a queue Q, Q ← N
2  foreach star s ∈ V do
3  |  if ∃v : (s,v) ∈ RE then
4  |  |  Remove the star marking from s
5  |  |  Add s and all its neighbors to Q. Also add v unless v ∈ R
6  |  end
7  end
8  foreach star s ∈ V do
9  |  if ∃v, v' : (v,v') ∈ NE ∧ (s,v) ∈ E then
10 |  |  Remove the star marking from s
11 |  |  Add s and all its neighbors to Q
12 |  end
13 end
14 while Q ≠ ∅ do
15 |  Extract the vertex, v, with the greatest degree from Q
16 |  if v is not marked as a star ∧ (v does not have any star neighbors ∨ the
   |  degree of v ≥ the degree of all its star neighbors) then
17 |  |  Mark v as a star
18 |  |  foreach star neighbor, s, of v with lower degree than v do
19 |  |  |  Remove star marking from s
20 |  |  |  Put all neighbors of s into Q
21 |  |  end
22 |  end
23 end
```

4.6 Incremental Clustering Algorithm

Incremental Clustering Algorithm (ICA) is our own algorithm that we have developed. The algorithm is very simple although we have not seen the algorithm being described earlier. Our intention behind it was to design an algorithm that was quick and supported incremental data, in other words it had to support adding more articles to the data set after the clustering had been started.

The algorithm processes one object at a time and places it in either an existing cluster or creates a new one. To decide which, it compares the input object with all existing clusters using the UPGMA method (as explained in section 4.2). For the most similar cluster it checks if that similarity is above a predefined threshold value β. If that is the case, it adds the new object into that cluster. Otherwise a new cluster will be created that only contains the new object. For the first object a new cluster will always be created. The pseudocode for this algorithm is found in algorithm 7.

Algorithm 7: Insertion of a single object with the Incremental Clustering Algorithm

```
Data: A new object o to be added, a set C containing all current clusters and a
      threshold β
Result: A set of clusters
if C is empty then
    Create a new cluster which only contains o and add it to C
else
    Calculate the similarity between o and each cluster c ∈ C using UPGMA
    Find the cluster c ∈ C which o is most similar to
    if the similarity is greater than β then
        Add o to cluster c
    else
        Create a new cluster which only contains o and add it to C
    end
end
return C
```

It is obvious that for each new object that the algorithm processes it has to compare that object to all other objects. The algorithm will consequently perform a lot of comparisons between objects in the long run which is not desirable. In the next chapter we will investigate the possibility to reduce the number of similarity calculations in order to improve the runtime performance for this and other algorithms.

5

Comparison reductions

From previous chapters we have enough information in order to start clustering. However, we still have a problem that running the clustering algorithms takes plenty of time.

From our measurements we concluded e.g. that the Incremental Clustering Algorithm (presented in section 4.6) spent about 95 % of the running time doing similarity measures. In this chapter we will present two (to our knowledge) new methods which will reduce the number of comparisons between text objects and thus obtain a better runtime performance.

5.1 Corner reduction

This method to reduce the number of comparisons between objects (in this case document vectors) requires that the algorithm uses Cosine similarity as its the similarity measure. It is based on the idea that when the document vectors are normalized (as discussed in section 2.3.1) the objects to be clustered can be seen as if they are lying on the surface of a n-dimensional sphere (where n is the number of unique words in the corpus after the pre-processing).

Instead of comparing an object to all other objects on the spheres surface, why not limit the comparisons to the objects in the current object's proximity and assume that all other are not very interesting for now? As described in section 2.2.2, two objects are similar if the angle between the document vectors is small. If the angle is small, the objects are also close to each other on the surface of the sphere. But how can we decide which objects are close to the current object?

The idea is that we divide the surface of the sphere into several areas and check which area or areas an object resides in. A first attempt is to measure the angle between an object and all the vectors which are representing just one term (the *corners*). The corner which the object is closest to will be the area to which the object belongs. The

corner areas will, in the case of three dimensions, look like as presented in figure 5.1. An object should only be compared with the objects in the same area as itself. In this way the number of comparisons will be lowered.

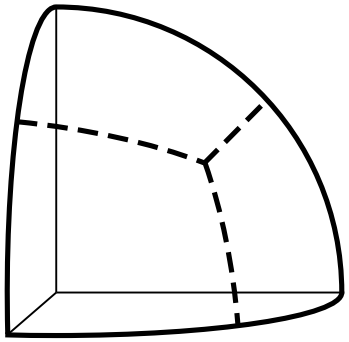


Figure 5.1: The corner areas in the case of three dimensions.

A problem with this method occurs if two object which are very similar to each other are placed in different areas. If that happens those two points will not be compared and thereby will probably not end up in the same cluster (of course this depends on the clustering method).

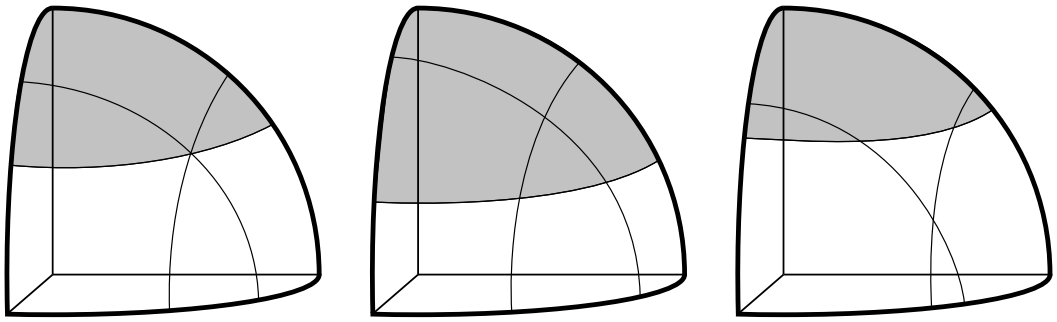
A second attempt is that if the Cosine similarity between an object and a corner is above a certain threshold value the object should belong to that area. With that approach an object may belong to several areas simultaneously. An object should then be compared to all objects in the areas it belongs to. Since an object may belong to several areas at the same time the problem that two very similar objects will not be compared will be a lot lower.

But what is a reasonable value for the threshold? With a vector length of one, the average weighting value for a specific term in the document vector would be $m = \frac{1}{\sqrt{n}}$ where n is the number of dimensions in the document vector. For a document vector with three dimensions and $\frac{1}{\sqrt{3}}$ as threshold value, the different areas would be divided as shown in figure 5.2a. Here we see that an object may belong to one, two and even all three areas (in the case that all weights are exactly evenly distributed).

If a user wants a possibly better clustering quality in exchange for clustering speed one might want to do the overlapping areas larger. This can be done by introducing a new variable γ, which we multiply with m in order to change the threshold value. To alter the threshold to be less strict the γ variable should be less than 1. The areas for a document vector of three dimensions and the threshold value set to $\frac{1}{\sqrt{3}} \cdot \gamma$ (where $\gamma < 1$) would be divided as shown in figure 5.2b.

On the other hand, if the time is important one might want to do fewer comparisons in order to speed up the algorithm. In this case we should multiply m with γ where $\gamma > 1$. The overlapping areas will be smaller and there is also a chance that an object will not be a member of any area if the weights are roughly evenly distributed, as seen in figure 5.2c.

If an object is not a member of any area it will not be compared to any other object



(a) $\gamma = 1$ **(b)** $\gamma < 1$ **(c)** $\gamma > 1$

Figure 5.2: The different corner areas with different values for γ. One of the areas in each figure is shaded for clarification.

and is consequently doomed to create a cluster on its own. To overcome this problem, if an object does not belong to any area it will be assigned an area as if γ was set to 1. With this method, all objects will be member of at least one area and the chance for an object to be compared to other objects is greater.

How to alter the clustering algorithms

To alter an algorithm to benefit from corner reduction is easy. As with the case with the Incremental Clustering Algorithm (presented in section 4.6) all cluster will belong to the corner area in which its member objects belongs to. That is, if the new object to be added, o, is in the same corner area as any of a cluster's, C, members, then o should be compared with C. If no member of a cluster, C, belongs to the same corner area as o they should not be compared and thereby there is no chance for o to be a member of C.

Dynamic Hierarchical Compact (section 4.4) have also been altered to benefit from corner reduction. As with corner reduction in the Incremental Clustering Algorithm a cluster will be a member of the same corner areas as its member objects. For each level in the algorithm, an object (or cluster) is only compared to the objects (or clusters) residing in the same corner area.

The performance and resulting clustering quality of these two altered algorithms will be examined in chapter 6. But corner reduction is of course not restricted to these algorithms but can be implemented to work with e.g. Hierarchical Agglomerative Clustering.

5.2 Word set reduction

Our other optimization approach is Word set reduction, named after its usage of sets of words (dimensions) to approximate the similarity between a document and a cluster.

The way it works is that for each cluster a set of dimensions is created, a Word set, which contains all dimensions of the documents that belongs to the cluster. A dimension

belongs to a cluster if any of the articles in the cluster have a weight over zero. The Word set is later utilized when a new document should be added to a cluster.

As described in the Clustering algorithms chapter there are some algorithms that use group-average or a similar method to compare a cluster with an incoming document inorder to decide their similarity. This is often combined with a threshold β that the similarity needs to fulfill in order to cluster documents together. Instead of performing the full similarity measure which often compares the input document with all documents in the cluster, the Word set is used to make a single comparison between the cluster and the input document using an altered Cosine similarity, see equation 5.1 where S_c is the Word set for cluster c and $\vec{W} = (w_1, w_2, \cdots, w_n)$ is the input documents word vector.

$$sim_C(\vec{W}, S_c) = \frac{\sum_{x \in S_c} w_x^2}{\|\vec{W}\|^2} \tag{5.1}$$

This single comparison is compared to a threshold γ, which says if the cluster should be considered as a candidate or not. If a cluster is considered as a candidate then the real similarity comparison is performed. The idea is that γ should be chosen so that it filters out clusters that should yield a too low similarity when the full similarity is done and compared to β.

If there is only a few number of documents in a cluster then it is usually better to do the full similarity instead of the approximation. In the case when a cluster only contains a single document it will take about equal the amount of time to calculate the real similarity and the approximation and in that case it is obviously better to use the real similarity, which does not need to be recomputed later if it turns out that the cluster is a candidate.

The regular Cosine similarity (section 2.2) measures the similarity between two documents by taking the dot product of the weight vectors. The dot product takes the sum of all shared dimensions with the product of each of the documents weights. Our altered Cosine similarity in equation 5.1 makes a similarity between a document and a Word set. Since the Word set only contains dimensions and not any weights a dot product can not be performed. Instead it takes the sum of the square weight from the document for all shared dimensions between the document and the Word set. The reason why it makes a good approximation is because dimensions with a high weight in the document also have a high impact on the similarity as in the regular Cosine similarity.

6

Results

In this chapter we will evaluate the different pre-processing steps as well as the clustering algorithms and comparison reductions presented in the previous chapters. In order to do so we have created several test data sets which we will compare our clustering results to. We will also present different measurements to evaluate the cluster quality.

6.1 Data sets

To test the pre-processing methods and clustering algorithms, we have constructed two data sets with news articles. One of the data sets is focused on quality where the clusters fully meet our specification. The other set is focused on having a great amount of articles in order to test the performance of the algorithms.

Both data sets have been used to create two smaller subsets, and all these data sets are presented in table 6.1.

Name	Articles	Reference clusters
Quality 1 (Q_1)	2 005	263
Quality 2 (Q_2)	5 011	562
Performance 1 (P_1)	25 686	2 916
Performance 2 (P_2)	44 971	5 685

Table 6.1: Our testing data sets.

In the beginning of our project we looked for clustered data sets of articles which could be used for testing. However, we could not find any data sets that fit our specification of how a cluster should be created (see Introduction). The best we could find was the

Google News website[1] that aggregates news from many news sources and creates clusters of articles that covers the same event. We concluded that sometimes their clusters align with our specification and other times they do not do it fully, but often it was close enough. Therefore we decided to use those clusters as a starting point.

Since Google does not provide a simple way to access their clustering data we had to scrape their site in order to get their clusters of articles. The only information we got from Google about the web articles were their title and URL.

To gather more data we decided to also download articles from RSS (Really Simple Syndication) feeds from some selected sources. RSS feeds usually gives us the title, publication date, brief description and a URL for each article. The purpose of adding these additional articles were to have access to a greater amount of articles and to have some singleton clusters, in order to represent a realistic case.

It is important to note that in order to get the text of an article we download the articles page and use software to extract the article text. The process of extracting article text is not perfect [24] and thus our articles can be faulty. For example, only some parts of the article text is extracted or wrong part of the page is extracted. To minimize the effect of this problem we have performed some automatic and manual cleanup of the articles. The cleanups also removed all non-English articles and articles that had a body shorter than 150 characters.

6.1.1 Quality data set

Since we could not find any reference set that met our definition of how clusters should be defined, we decided to create our own based on the data we had collected. The data set was manually created and checked in order to ensure correctness. To speed up the process we utilized the Incremental Clustering Algorithm (see section 4.6) and then refined the result by hand.

We did some minor filtering of the articles, mostly it was on articles where we could not with a high certainty decide whether an article should belong in a cluster or not. For example, we completely ignored some sports that we had insufficient knowledge of, as well as some economy news, especially related to stocks. There were also some occurrences of articles that were more of a summarization of several articles, such as the most read news a specific day. Those were removed since we did not consider them as "real" articles.

We also tried to make the set realistic with various sizes of the clusters, ranging from singleton clusters up to 59 articles in a cluster, see figure 6.1.

The result was two data sets Q_1 and Q_2, see table 6.1, where Q_1 is a subset of Q_2.

[1] http://news.google.com

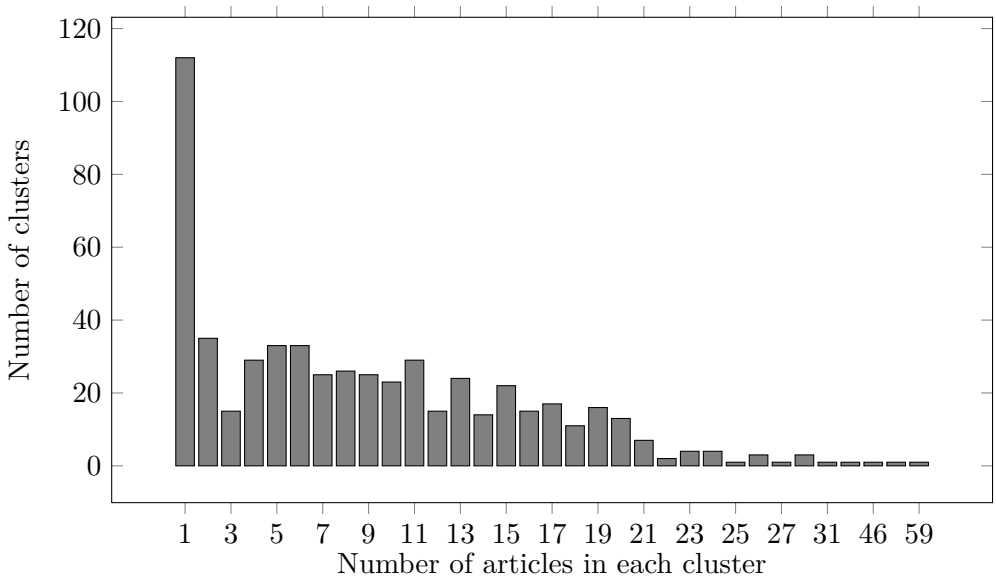


Figure 6.1: Distribution of cluster sizes.

6.1.2 Performance data set

To be able to test how the algorithms scale we decided to construct a data set with a high number of articles. We had access to over 100 000 articles that we had downloaded, but we choose to limit the data set to 45 000 articles. The reasoning behind this was that since we run the clustering single threaded on our own machines we could not allow for too long running times.

Out of the full data set a subset of about 25 000 articles was used. This will be used to see how the running time is affected with a different amount of data.

The data sets can be found in table 6.1 as P_1 and P_2.

6.2 Evaluation of clustering quality

To evaluate the cluster quality of the resulting clusters from the cluster algorithms we will have to introduce some measurements. In this thesis we make use of the *F-measure*, *purity* and *entropy* measures. In order to understand F-measure we will have to discuss *precision* and *recall* which are two of the most frequently used measuring methods in the field of information retrieval [6].

In information retrieval precision (P) is defined as the fraction of the retrieved elements that are relevant and recall (R) is defined as the fraction of the relevant elements which are retrieved [6]. That is:

$$P(c, o) = \frac{|c \cap o|}{|c|} = \frac{\text{Number of relevant elements retrieved}}{\text{Number of retrieved elements}}$$

and

$$R(c,o) = \frac{|c \cap o|}{|o|} = \frac{\text{Number of relevant elements retrieved}}{\text{Number of relevant elements}}$$

where c is the set of the retrieved elements and o is the set of relevant elements.

But there is a trivial way to get a good recall value. Just return all documents for any query to the system [25]. To overcome this problem we introduce the F-measure which combines both recall and precision. The F-measure is defined as:

$$F_1(c,o) = \frac{2 \cdot P(c,o) \cdot R(c,o)}{P(c,o) + R(c,o)}$$

In the equation above the F-measure is denoted F_1 because the recall and precision is weighted equally. That is not always the case, sometimes one might want to weight one or the other higher. This is done in the following way [25]:

$$F_\alpha(c,o) = \frac{(1+\alpha) \cdot P(c,o) \cdot R(c,o)}{\alpha \cdot P(c,o) + R(c,o)}$$

Thus, $F_{0.5}$ means that the precision has higher weight than the recall and F_2 means that the recall is weighted higher than the precision. In the remainder of this thesis we will only be working with the F_1 value.

In order to make this work together with clustering we will have to make use of our reference clusters (presented in section 6.1), which can be seen as our set of relevant elements. We will from here on denote the reference clusters O. Needless to say, we will also have to make use of the calculated clusters from the clustering algorithm (C), which can be seen as our set of retrieved elements. Now the F-measure is calculated in the following way [25]:

$$F_1 = \sum_{o \in O} \frac{|o|}{n} \max_{c \in C} F_1(c,o)$$

where n is the total number of elements in the collection which is clustered.

As mentioned earlier we also make use of the purity and entropy measurements. Purity is the percentage of documents which is in the correct cluster. That is, each computed cluster is associated with the reference cluster from which it has the most number of objects [26]. We sum up the number of correctly placed objects and divide with the total number of objects. The formular looks like this [25, 26]:

$$Purity = \frac{1}{n} \sum_{o \in O} \max_{c \in C} |o \cap c|$$

where n once again is the total number of objects which have been clustered. A purity close to 0 indicates a bad clustering while a perfect clustering has a purity value equal to 1 [26].

Entropy measures the distribution of objects from each reference cluster within a computed cluster [25]. The entropy for a clustering is defined as:

$$Entropy = -\frac{1}{\log k} \sum_{o \in O} \frac{1}{n} \sum_{c \in C} |o \cap c| \cdot \log P(c,o)$$

where k represents the number of reference clusters.

A low entropy value indicates that the computed clusters does not contain objects from a lot of different reference clusters. Thus, we strive for a low entropy value.

6.3 Pre-processing evaluation

In appendix A we have presented the results of our benchmarks with different parameters for our pre-processing steps and we will in this section discuss those results. The benchmarking was done using the Hierarchical Compact Algorithm (presented in section 4.4). When nothing else is stated the following parameters for pre-processing are used:

- Terms which occurs 5 times or less throughout the corpus is removed.
- The document vectors are truncated such that they only contain the 100 most significant terms.
- Terms found in the title will be counted twice.
- Named entity extraction will not be used.
- The term frequency (tf) in the document vectors are weighted using $(1+\log_4(tf_t))\times idf_t$. The idf_t is calculated as:

  $$idf_t = \log_4 \frac{N}{df_t}$$

  where N is the total number of documents in the corpus and df_t is the number of documents which contains the term t at least once.
- The similarity measure used in the benchmarking is Cosine similarity without any penalties if two articles are published by the same source or if there is long time between publications.

The motivation to that we use Hierarchical Compact Algorithm with these pre-processing steps are that after some quick testing it seemed to give a acceptable results with a reasonable runtime performance.

6.3.1 Pruning of rare words

In section A.1 are the results from when the pruning value was alternated. The values we tested ranges from 0 to 30. Our testing does not show any significant improvement in the clustering quality when changing the pruning values. But we see a big dimensionality reduction already at small pruning values, meaning that the majority of the words in the article collection does only occur very few times.

A pruning value of 3 seems to be the best one in the case when clustering 2005 articles and a pruning value of 10 seems to be the best when we clustered 5011 articles. These

values can be explained with that noise (including spelling mistakes) is removed which does not help the algorithm to distinguish one event from another. On the other hand, a too large value may remove words which are important for the articles and consequently impair the cluster quality.

6.3.2 Truncation

The results from the tests where we alternated the truncation value is found in section A.2. We did tests with no truncation at all as well as only keeping the t most significant words. t was given different values ranging from 50 to 150. As with the case with pruning, our tests did not show any significant change in the clustering quality for our different values of t. Even the in parameter β had a quite steady value.

This is most probably because the least significant terms in the document vector does not have any great impact on the similarity calculations between the articles. As long as the most significant term weights for each document vector are kept one can expect that the cluster quality will not be reduced.

The runtime performance was not affected very much by the truncation (not seen in the test result tables). In the case with 5011 articles the runtime performance boost was less than 5 % when comparing no truncation to truncating such that only the 50 most significant terms was kept.

6.3.3 Application of same source- and time difference penalties

In table 6.2 and table 6.3 are the results from when we penalized the similarity measure if the articles were published by the same source or if long time had passed between the article publications. More detailed tables are found in section A.3.

If two articles are published by the same source the similarity measure will receive a 15 % penalty. When it comes to the publication time difference, the similarity measure will not receive a penalty if the difference is at most 2 days. After that the penalty will increase linearly until a 40 % penalty have been reached after 14 days where it flattens out.

To come up with these values we first thought about what would be reasonable values. Then several tests were conducted in order to come up with the final values which showed the best results.

Our tests reveal that by applying the penalties to the similarity measurements the cluster quality will be improved. All our quality measurements are improved when using the penalties. The computed clusters are more pure and does overall contain less articles from different reference clusters.

Penalties	β	F-measure	Purity	Entropy
Off	0.30	0.964	0.976	0.061
On	0.30	0.974	0.987	0.037

Table 6.2: Source and time penalty for the 2005 articles set.

Penalties	β	F-measure	Purity	Entropy
Off	0.295	0.944	0.960	0.091
On	0.290	0.958	0.975	0.064

Table 6.3: Source and time penalty for the 5011 articles set.

6.3.4 Weighting of terms in titles

The results from when we alternated the weight of the terms found in title is presented in section A.4. We tested to not take the title into account at all and then also tests where each term in the title were counted 1, 2, 4, 8 and 16 times. We could observe from our test results that counting each term in the title about 8 times yields the best resulting clustering.

The most likely reason why this values results in good clustering is because a lot of information about the article is found in title. Not taking the title into account at all results in a worse clustering quality as important occurrences of terms are not registered. We could also observe that giving the terms in the title too much weight also results in quality deterioration. A plausible explanation to this is because important terms in the body of the article will lose their meaning when the terms in the title have such a heavy weight.

6.3.5 Weighting of extracted named entities

The extracted named entities will in our representation be added as new words to our document vector. As in the case with the title weights, we tested to count each entity found x times. In our tests we set x to 0, 2, 4 and 8. The results can be found in section A.5.

By inspecting the test results one may see that a greater weight on the entities results in an impaired clustering quality. This might be due to we introduce noise into the document vectors which affect the similarity measures negatively. Another plausible explanation is that articles which for example involves the same person will be more similar to each other even if they do not describe the same event. Consequently may such articles end up in the same cluster even though they should not.

6.3.6 Different term weighting methods

Here we tested four different term weighting methods. The ones we tested were tf, $\sqrt{tf}$, $1 + \log_4(tf)$ and $1 + \log_2(tf)$ where tf is the term frequency. We did all these tests two times. First we combined them with the inverse document frequency and then without it in order to see how much the inverse document frequency affected our clustering quality. The test results are found in section A.6.

Term weighting combined with inverse document frequency

Our tests show that both the $\sqrt{tf}$ and $1 + \log_4(tf)$ weighting methods yields quite similar clustering. But both the tf and $1 + \log_2(tf)$ weighting methods results in better clustering, especially the $1 + \log_2(tf)$ weighting. It seems like the difference in term frequency in an article is more important than we originally thought. However, it appears that some smoothing still is needed to reduce the weight of the most frequently occurring terms.

Term weighting without inverse document frequency

According to our tests it seems like the $1 + \log_2(tf)$ weighting method results in the best clustering quality also in the case when not used with the inverse document frequency. Furthermore, once again the $\sqrt{tf}$ and $1 + \log_4(tf)$ weighting methods performs worst.

One may also notice that the cluster quality is slightly worse in comparison to the cases where also the inverse document frequency was used. It appears that giving terms which occurs infrequently a greater weight than those who appears often is a good approach. However, the quality is still quite good in comparison to the cases where also the inverse document frequency were in use. One may consequently argue that the inverse document frequency should not be used at all, as it will change over time as new articles are are added to the clustering.

6.3.7 Different similarity measure methods

Table 6.4 and table 6.5 shows the test result from our comparisons between the Cosine similarity measure and the Jaccard similarity measure. The results are also found in more detail in section A.7. Our test results show that the clustering algorithm returns clusters of similar clustering quality while using either the Cosine or Jaccard similarity measure.

Similarity measure	F-measure	Purity	Entropy
Cosine	0.964	0.976	0.061
Jaccard	0.962	0.975	0.062

Table 6.4: Different similarity measures for the 2005 articles set.

Similarity measure	F-measure	Purity	Entropy
Cosine	0.944	0.960	0.091
Jaccard	0.949	0.961	0.091

Table 6.5: Different similarity measures for the 5011 articles set.

6.4 Algorithms

In order to see how well the clustering algorithms from chapter 4 performs we have conducted a comparison between them. We have tested their clustering quality and measured their running times on our different data sets.

All test results can be found in A.8, the data that follows in this section is a subset out of those results.

6.4.1 Experimental setup

From the previous section about pre-processing (section 6.3) we have concluded that the highest quality should be achieved when using the following options:

- Titles are counted 8 times.
- Cosine similarity are used along with penalties.
- The terms are weighted using $(1 + \log_2(tf)) \times idf$.
- The pruning value is set to 3.
- The truncation value is set to 100.

The mentioned options have been used in all clustering algorithm tests.

All algorithms have been written in the programming language Scala. Some of our implementations are better optimized than others. However, this should only affect the running time, the evaluation measures should remain unaffected of this.

All tests have been performed on a single computer that used a Intel Core i7 2.7 GHz (4 cores) processor. The tests ran on a single thread in order to limit the optimization differences. The JVM that was used had its heap size limited to 4 GB and was restarted for each test.

Time measurements is measured between the start of a clustering process until it completes, neither the pre-processing steps nor evaluation measurements are included.

Hierarchical Agglomerative Clustering with GAAC linkage have been excluded due to unreasonable running times on the Q_2 data set. For k-means the expected number of clusters were given as input. Since k-means and Bisecting k-means are nondeterministic algorithms, we ran each of these tests five times and selected the test which produced the clustering with maximum F-measure. All other algorithms were run with a β value

which produced roughly the same amount of clusters as reference clusters, with as high F-measure as possible.

6.4.2 Clustering quality

In order to test how well the algorithms clusters articles we have tested them with the data set Q_2. The data set contains a reasonable amount of articles divided into different sizes of clusters (see 6.1.1). Table 6.6 contains the results.

Algorithm	β	γ	Time (s)	F-measure	Purity	Entropy
HAC complete-link	0.20		408.581	0.939	0.964	0.089
HAC single-link	0.34		369.133	0.928	0.929	0.144
HAC UPGMA	0.28		353.762	0.965	0.973	0.063
k-means			334.696	0.883	0.919	0.180
Bisecting k-means		5E−5	23.332	0.588	0.630	0.978
HSA	0.36		259.137	0.966	0.982	0.046
HCA	0.35		113.310	0.964	0.980	0.050
ICA	0.28		10.630	0.962	0.975	0.058
HCA with Corner red.	0.36	1.4	5.100	0.963	0.982	0.045
ICA with Corner red.	0.28	1.4	1.023	0.962	0.975	0.058
ICA with Word set red.	0.28	0.05	1.632	0.961	0.975	0.056

Table 6.6: Different clustering algorithms and reductions on the Q_2 data set with 5011 elements.

Starting with the worst result, Bisecting k-means performed very badly. The reason for this could be that articles that belongs to the same reference clusters is at some point divided between the two centroids and have no possibility to be merged together again.

k-means had some help since we had to give it the number of clusters it should create. It performed quite well, not as good as the other algorithms which all had F-measures above 0.9. The entropy is high and the reason behind that is probably that the centroids are not evenly distributed to cover the reference clusters.

Regarding the Hierarchical Agglomerative Clustering all three linkage criteria shows good results. Single-link have most likely created chains since it has a higher entropy compared to the other two. Out of the three, UPGMA have the best measurements and this is related to that it takes all articles in a cluster into account when it performs similarity measurements.

HSA, HCA and ICA shows almost equally good F-measures.

Over the entire test HSA shows the absolute best result with the highest F-measure and purity as well as the lowest entropy.

As when it comes to the reductions methods (presented in chapter 5) we can see that the quality is quite stable even when those are applied. However, we observed that the resulting clusters are not exactly the same as when the algorithms did not use any reduction method at all. But those differences occurred very rarely throughout the resulting clusters.

6.4.3 Clustering performance

Since we have aimed to build a large scale system that could handle a vast amount of articles, we need a clustering process that can cluster articles at a high rate. In order to compare the algorithms we have used the data sets Q_2, P_1 and P_2.

The result from the Q_2 test can be found in table 6.6. This is the only performance test where all algorithms have been tested.

Test results for P_1 and P_2 can be found in table 6.7 and 6.8. Those tests only include HCA with Corner reduction as well as ICA without any reduction as well as with both applied separately. All other algorithms were too slow to cluster the number of articles present in those data sets.

Algorithm	β	γ	Time (s)
HCA with Corner reduction	0.30	1.4	150.198
ICA	0.23		353.096
ICA with Corner reduction	0.23	1.4	25.723
ICA with Word set reduction	0.23	0.05	53.906

Table 6.7: Algorithms with reduction methods on the P_1 data set with 25 586 elements.

Algorithm	β	γ	Time (s)
HCA with Corner reduction	0.30	1.4	517.328
ICA	0.23		1 127.040
ICA with Corner reduction	0.23	1.4	87.232
ICA with Word set reduction	0.23	0.05	172.269

Table 6.8: Algorithms with reduction methods on the P_2 data set with 44 971 elements.

From table 6.6 one can see that ICA is the absolute fastest algorithm, when not looking at the reduction method results. Bisecting k-means was also very fast, but did not achieve a good clustering quality and its result is therefore not relevant. Thus, HCA was the second fastest algorithm but was, with that data set, ten times as slow as ICA.

The Q_2 data set is 5011 articles, a 1/20 of the amount of articles that we wanted to be able to cluster in less than an hour (see the Introduction). If the algorithms would have

a linear time complexity then it would only be ICA and HCA that could possibly achieve the runtime performance goal, based on the test results from table 6.6. To be clear, the time complexity of the algorithms is not linear to the number of articles. Therefore, we performed larger tests on the P_1 and P_2 data sets.

As seen in table 6.7 and table 6.8, when running ICA with either reduction method the running times have reduced to just a small fraction compared to ICA without any reduction. The Word set reduction in ICA gives about 6.5 times performance boost over the unchanged algorithm. Corner reduction is slightly better and gives between 10-14 times performance boost compared to ICA. We could also observe that the runtime performance of HCA is greatly improved when using Corner reduction. It managed to cluster more than 25 000 articles in 150 seconds while it spent 113 seconds to cluster only 5000 articles without any reduction.

7

Conclusion

In this thesis we examined different approaches on how to cluster news articles so that articles covering the same information would end up in the same cluster. The requirement was that the algorithm should produce clusters of high quality and to do so with good runtime performance. We investigated several pre-processing methods as well as various clustering algorithms to achieve this.

Regarding the pruning pre-processing step, our tests showed that pruning advantageously could be applied. This seemed to remove some noise and consequently made the similarity measures more reliable.

When it comes to the similarity measures used we could see that when using either the Cosine similarity measure or the extended Jaccard similarity measure the algorithm returned clustering results of equal quality. However, when applying penalties to similarity measure that was published by the same source or if long time passes between publications we observed that the quality was greatly improved.

The weighting of words showed us that it was a good idea to count the words found in the title several times as it boosted the quality a bit. We also observed that by extracting named entities and giving weight to them actually degraded the clustering quality. But the perhaps most interesting result when it comes to the weighting is that the inverted document frequency only improved the clustering quality a little. Thus, one may remove it altogether in order to skip reweighting all terms as new articles are added.

All of the examined algorithms showed quite bad running times. Although there seems to be a lot to gain by using our reduction methods which improved the running time greatly.

Our tests also showed that the Incremental Clustering Algorithm (presented in section 4.6) returned clusters of high quality. In combination with the different similarity reduction methods (chapter 5) it could do so in very short time, without significantly affecting the clustering quality. It was much faster than the Hierarchical Compact Algorithm while the cluster quality was just slightly reduced.

Even if we managed to achieve good clusterings during our tests one should be aware that it is hard (if not impossible) to guarantee a flawless result. However, our tests shows that it is possible to cluster news articles with high accuracy and to do so with good runtime performance.

We think that we with the Incremental Clustering Algorithm in combination with our reduction methods have met both the quality and runtime performance requirements for this project.

7.1 Future work

There are lots of additional problems to address regarding this bigger problem. Some of our ideas that we would like to see further investigation are the following.

As for the pre-processing part it would be interesting to find synonyms and combine them to a single word in the document vector. If this could be done with high precision we think that the similarity measures could be more reliable and consequently one can expect better clustering quality.

It would also be interesting if the relationship between words could be used in the similarity measures. For example, the words *ambulance* and *hospital* are not synonyms but both words belongs to the medical care area. If this could be used when performing the similarity measure then the clustering might get better.

The context of the words is something else which could be taken into account. In this thesis we used the vector space model which represents the text as a bag of words. But by actually using the information in the article and not just the words present in it, the similarity measurement might distinguish an event from another even better and thus one might expect a boost in the clustering quality.

We have with good result explored two approaches for reducing the number of comparisions that is done in the clustering algorithms. We think that further developments can be done in this area in order to improve the runtime performance of different algorithms.

Lastly we think that an alternative to the inverted document frequency should be examined. This method should not require all articles term weights to be updated as new articles arrive over time.

Bibliography

[1] W. B. Frakes, R. Baeza-Yates, Information Retrieval: Data Structures & Algorithms, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[2] A. Huang, Similarity measures for text document clustering, in: Proceedings of the Sixth New Zealand Computer Science Research Student Conference (NZCSRSC2008), Christchurch, New Zealand, 2008, pp. 49–56.

[3] A. Strehl, J. Ghosh, R. Mooney, Impact of similarity measures on web-page clustering, in: Proceedings of the AAAI Workshop on AI for Web Search, AAAI, 2000, pp. 58–64.

[4] S. Zhong, Efficient online spherical k-means clustering, in: Proceedings of IEEE Int. Joint Conf. Neural Networks (IJCNN 2005), Vol. 5, 2005, pp. 3180–3185 vol. 5.

[5] A. Strehl, Relationship-based Clustering and Cluster Ensembles for High-dimensional Data Mining, Ph.D. thesis, The University of Texas at Austin (2002).

[6] C. D. Manning, P. Raghavan, H. Schütze, An Introduction to Information Retrieval, Online Edition, Cambridge University Press, 2009.

[7] H. Schütze, C. Silverstein, Projections for efficient document clustering, in: Proceedings of the 20th International ACM SIGIR Conference on Research and Development in Information Retrieval, 1997, pp. 74–81.

[8] L. Dolamic, J. Savoy, When stopword lists make the difference, Journal of the American Society for Information Science and Technology 61 (1) (2010) 200–203.

[9] N. Sandhya, Y. Sri Lalitha, V. Sowmya, K. Anuradha, A. Govardhan, Analysis of stemming algorithm for text clustering, IJCSI International Journal of Computer Science Issues 8 (5) (2011) 352–359.

[10] M. F. Porter, Snowball: A language for stemming algorithms, online; accessed 21st of February 2013 (2001).
URL http://snowball.tartarus.org/texts/introduction.html

[11] Google scholar search, online; accessed 21st of February 2013.
URL http://scholar.google.com/scholar?q=%22Martin+Porter%22

[12] A. Hotho, S. Staab, G. Stumme, Wordnet improves text document clustering, in: In Proc. of the SIGIR 2003 Semantic Web Workshop, 2003.

[13] J. Sedding, D. Kazakov, Wordnet-based text document clustering, in: Proceedings of the 3rd Workshop on RObust Methods in Analysis of Natural Language Data, Association for Computational Linguistics, 2004, pp. 104–113.

[14] K. Blake, Inverted pyramid story format, online; accessed 15th of February 2013.
URL http://kelab.tamu.edu/spb_encyclopedia/data/Inverted%20pyramid%20story%20format.pdf

[15] E. Marsh, D. Perzanowski, MUC-7 Evaluation of IE Technology, in: Message Understanding Conference Proceedings, 1998.
URL http://www-nlpir.nist.gov/related_projects/muc/proceedings/muc_7_proceedings/marsh_slides.pdf

[16] The Stanford Natural Language Processing Group, Stanford Named Entity Recognizer (NER), online; accessed 8th of May 2013.
URL http://www-nlp.stanford.edu/software/CRF-NER.shtml

[17] Y. Zhao, G. Karypis, U. Fayyad, Hierarchical clustering algorithms for document datasets, Data Mining and Knowledge Discovery 10 (2) (2005) 141–168.

[18] M. Steinbach, G. Karypis, V. Kumar, A comparison of document clustering techniques, in: KDD workshop on text mining, 2000, pp. 525–526.

[19] Y. Hatagami, T. Matsuka, Text mining with an augmented version of the bisecting k-means algorithm, in: Proceedings of the 16th International Conference on Neural Information Processing: Part II, Springer-Verlag, 2009, pp. 352–359.

[20] R. Gil-García, J. Badía-Contelles, A. Pons-Porrata, Dynamic hierarchical compact clustering algorithm, in: A. Sanfeliu, M. Cortés (Eds.), Progress in Pattern Recognition, Image Analysis and Applications, Vol. 3773 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005, pp. 302–310.

[21] R. Gil-García, A. Pons-Porrata, Dynamic hierarchical algorithms for document clustering, Pattern Recognition Letters 31 (6) (2010) 469–477.

[22] R. Gil-García, J. Badía-Contelles, A. Pons-Porrata, A general framework for agglomerative hierarchical clustering algorithms, in: Proceedings of the 18th International Conference on Pattern Recognition (ICPR'06), Vol. 2, 2006, pp. 569–572.

[23] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, USA, 1979.

[24] C. Kohlschütter, P. Fankhauser, W. Nejdl, Boilerplate detection using shallow text features, in: Proceedings of the third ACM international conference on Web search and data mining, ACM, 2010, pp. 441–450.

[25] N. O. Andrews, E. A. Fox, Recent developments in document clustering, Technical Report TR-07-35, Computer Science, Virginia Tech. (2007).

[26] M. Deepa, P. Revathy, Validation of document clustering based on purity and entropy measures, International Journal of Advanced Research in Computer and Communication Engineering 1 (3) (2012) 147–152.

A

Benchmarks

In section A.1 through section A.6 we have evaluated the different pre-processing steps. They were evaluated using the Hierarchical Clustering Algorithm. When nothing else is stated the following parameters for pre-processing were used:

- Terms which occurs 5 times or less throughout the corpus is removed.
- The document vectors are truncated such that they only contain the 100 most significant terms.
- Terms found in the title will be counted twice.
- Entity extraction will not be used.
- The term frequency (tf) in the document vectors are weighted using $(1+\log_4(tf_t))\times idf_t$. The idf_t is calculated as:

  $$idf_t = \log_4 \frac{N}{df_t}$$

  where N is the total number of documents in the corpus and df_t is the number of documents which contains the term t at least once.
- The similarity measure used in the benchmarking is Cosine similarity without any penalties if two articles are published by the same source or if there is long time between publications.

In the following tables the *Cluster* column presents how many clusters that were produced and the *Perfect* column presents how many of the produced clusters that were perfect. With a perfect cluster we mean that the cluster contains exactly the same elements as a corresponding reference cluster. The β column represents what value the β in parameter was set to during the clustering.

A.1 Pruning

Pruning	β	Words remaining	F-measure	Purity	Entropy	Clusters	Perfect
0	0.23	19668	0.963	0.974	0.062	270	192
1	0.25	12056	0.964	0.972	0.068	258	191
3	0.28	7925	0.967	0.977	0.058	262	192
5	0.30	6358	0.964	0.976	0.061	266	188
10	0.34	4470	0.959	0.973	0.064	273	184
15	0.37	3613	0.957	0.975	0.063	286	179
30	0.39	2396	0.960	0.975	0.065	275	184

Table A.1: Pruning for the 2005 articles set

Pruning	β	Words remaining	F-measure	Purity	Entropy	Clusters	Perfect
0	0.235	32288	0.949	0.963	0.088	579	372
1	0.255	19195	0.949	0.962	0.090	562	371
3	0.280	12851	0.949	0.963	0.086	570	381
5	0.295	10379	0.944	0.960	0.091	572	375
10	0.315	7522	0.950	0.965	0.084	569	382
15	0.320	6107	0.946	0.961	0.093	566	375
30	0.340	4211	0.946	0.961	0.094	569	358

Table A.2: Pruning for the 5011 articles set

A.2 Truncation

In the following tables the *Mean length* column represents the mean length of the document vectors.

Truncation	β	Mean length	F-measure	Purity	Entropy	Clusters	Perfect
No truncation	0.31	105.2	0.963	0.976	0.061	269	186
150	0.31	97.9	0.961	0.976	0.060	272	183
125	0.31	92.6	0.962	0.977	0.057	274	186
100	0.30	83.4	0.964	0.976	0.061	266	188
75	0.31	69.1	0.962	0.977	0.059	273	185
50	0.31	49.1	0.962	0.976	0.059	270	183

Table A.3: Truncation for the 2005 articles set

Truncation	β	Mean length	F-measure	Purity	Entropy	Clusters	Perfect
No truncation	0.300	106.7	0.949	0.961	0.089	569	374
150	0.300	98.8	0.949	0.962	0.088	564	377
125	0.295	93.0	0.948	0.960	0.090	569	375
100	0.295	83.6	0.944	0.960	0.091	572	375
75	0.290	69.2	0.945	0.960	0.092	567	373
50	0.295	48.0	0.945	0.964	0.089	571	369

Table A.4: Truncation for the 5011 articles set

A.3 Source and time penalties

Penalties	β	F-measure	Purity	Entropy	Clusters	Perfect
Off	0.30	0.964	0.976	0.061	266	188
On	0.30	0.974	0.987	0.037	273	199

Table A.5: Source and time penalty for the 2005 articles set

Penalties	β	F-measure	Purity	Entropy	Clusters	Perfect
Off	0.295	0.944	0.960	0.091	572	375
On	0.290	0.958	0.975	0.064	578	392

Table A.6: Source and time penalty for the 5011 articles set

A.4 Weighting of titles

Title weight	β	F-measure	Purity	Entropy	Clusters	Perfect
0	0.285	0.953	0.969	0.075	273	168
1	0.295	0.963	0.976	0.060	269	185
2	0.300	0.964	0.976	0.061	266	188
4	0.310	0.961	0.974	0.062	267	191
8	0.320	0.968	0.980	0.049	272	202
16	0.320	0.966	0.979	0.053	271	196

Table A.7: Different title weights for the 2005 articles set

Title weight	β	F-measure	Purity	Entropy	Clusters	Perfect
0	0.260	0.945	0.957	0.102	559	344
1	0.285	0.945	0.960	0.094	575	358
2	0.295	0.944	0.960	0.091	572	375
4	0.300	0.948	0.961	0.088	567	381
8	0.305	0.952	0.967	0.078	572	391
16	0.305	0.952	0.964	0.081	556	396

Table A.8: Different title weights for the 5011 articles set

A.5 Clustering with named entities

Entities weight	β	F-measure	Purity	Entropy	Clusters	Perfect
0	0.300	0.964	0.976	0.061	266	188
2	0.310	0.954	0.973	0.070	265	181
4	0.300	0.952	0.972	0.073	268	177
8	0.235	0.937	0.958	0.101	269	166

Table A.9: Different weights on named entities for the 2005 articles set

Entities weight	β	F-measure	Purity	Entropy	Clusters	Perfect
0	0.295	0.944	0.960	0.091	572	375
2	0.290	0.942	0.963	0.094	562	346
4	0.285	0.936	0.963	0.095	577	334
8	0.235	0.923	0.947	0.128	583	302

Table A.10: Different weights on named entities for the 5011 articles set

A.6 Weighting methods

With idf

Penalties	β	F-measure	Purity	Entropy	Clusters	Perfect
$tf \times idf$	0.445	0.962	0.979	0.056	271	186
$\sqrt{tf} \times idf$	0.315	0.964	0.976	0.061	267	188
$(1 + \log_4(tf)) \times idf$	0.300	0.964	0.976	0.061	266	188
$(1 + \log_2(tf)) \times idf$	0.345	0.969	0.978	0.054	258	200

Table A.11: Different tf methods (with idf) for the 2005 articles set

Penalties	β	F-measure	Purity	Entropy	Clusters	Perfect
$tf \times idf$	0.455	0.947	0.966	0.085	566	350
$\sqrt{tf} \times idf$	0.310	0.946	0.962	0.091	571	363
$(1 + \log_4(tf)) \times idf$	0.295	0.944	0.960	0.091	572	375
$(1 + \log_2(tf)) \times idf$	0.350	0.954	0.967	0.081	554	386

Table A.12: Different tf methods (with idf) for the 5011 articles set

Without idf

Penalties	β	F-measure	Purity	Entropy	Clusters	Perfect
tf	0.440	0.956	0.973	0.074	269	179
$\sqrt{tf}$	0.335	0.960	0.973	0.072	265	179
$1 + \log_4(tf)$	0.320	0.954	0.968	0.079	270	177
$1 + \log_2(tf)$	0.370	0.965	0.977	0.060	270	190

Table A.13: Different tf methods (without idf) for the 2005 articles set

Penalties	β	F-measure	Purity	Entropy	Clusters	Perfect
tf	0.465	0.942	0.960	0.107	567	331
$\sqrt{tf}$	0.340	0.954	0.968	0.081	565	355
$1 + \log_4(tf)$	0.325	0.952	0.964	0.088	567	355
$1 + \log_2(tf)$	0.375	0.953	0.966	0.081	564	367

Table A.14: Different tf methods (without idf) for the 5011 articles set

A.7 Similarity measures

Similarity measure	β	F-measure	Purity	Entropy	Clusters	Perfect
Cosine	0.30	0.964	0.976	0.061	266	188
Jaccard	0.17	0.962	0.975	0.062	267	188

Table A.15: Different similarity measures for the 2005 articles set

Similarity measure	β	F-measure	Purity	Entropy	Clusters	Perfect
Cosine	0.295	0.944	0.960	0.091	572	375
Jaccard	0.160	0.949	0.961	0.091	565	370

Table A.16: Different similarity measures for the 5011 articles set

A.8 Algorithms

See 6.4.1 for the setup that was used for the following tests.

Quality tests with data set Q_2

Algorithm	β	γ	Time (s)	F-measure	Purity	Entropy	Clusters	Perfect
HAC complete-link	0.20		408.581	0.939	0.964	0.089	553	336
HAC single-link	0.34		369.133	0.928	0.929	0.144	565	396
HAC UPGMA	0.28		353.762	0.965	0.973	0.063	570	404
k-means			334.696	0.883	0.919	0.180	562	204
Bisecting k-means		5E−5	23.332	0.588	0.630	0.978	448	2
HSA	0.36		259.137	0.966	0.982	0.046	608	405
HCA	0.35		113.310	0.964	0.980	0.050	594	403
HCA with Corner	0.36	1.4	5.100	0.963	0.982	0.045	614	403
ICA	0.28		10.630	0.962	0.975	0.058	586	406
ICA with Corner	0.28	1.4	1.023	0.962	0.975	0.058	587	405
ICA with Word set	0.28	0.05	1.632	0.961	0.975	0.056	597	403

Table A.17: Different clustering algorithms and reductions on the Q_2 data set with 5011 elements.

Performance tests with data set P_1

Algorithm	β	γ	Time (s)	F-measure	Purity	Entropy	Clusters	Perfect
HCA with Corner	0.30	1.4	150.198	0.581	0.623	0.919	2936	307
ICA	0.23		353.096	0.582	0.618	0.917	2878	285
ICA with Corner	0.23	1.4	25.723	0.582	0.618	0.917	2891	286
ICA with Word set	0.23	0.05	53.906	0.590	0.633	0.878	3052	276

Table A.18: Algorithms with reductions on the P_1 data set with 25 586 elements.

Performance tests with data set P_2

Algorithm	β	γ	Time (s)	F-measure	Purity	Entropy	Clusters	Perfect
HCA with Corner	0.30	1.4	517.328	0.539	0.580	1.048	5321	519
ICA	0.23		1 127.040	0.540	0.575	1.045	5195	485
ICA with Corner	0.23	1.4	87.232	0.540	0.576	1.044	5222	481
ICA with Word set	0.23	0.05	172.269	0.549	0.592	0.996	5523	465

Table A.19: Algorithms with reductions on the P_2 data set with 44 971 elements.