Zurich University
of Applied Sciences

School of
Engineering
InIT Institute of Applied
Information Technology

Bachelor thesis Computer Science

Dynamic Event Detection in Data Streams

Author	Daniel Milenkovic David Pacassi Torrico
Main supervisor	Dr. Andreas Weiler
Sub supervisor	Prof. Dr. Kurt Stockinger
Date	07.06.2019

Zurich University
of Applied Sciences

School of Engineering

DECLARATION OF ORIGINALITY

Bachelor's Thesis at the School of Engineering

By submitting this Bachelor's thesis, the undersigned student confirms that this thesis is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the Bachelor thesis have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date: Signature:

....

.....

.....

The original signed and dated document (no copies) must be included after the title sheet in the ZHAW version of all Bachelor thesis submitted.

Zurich University
of Applied Sciences

Abstract

Detecting events in data streams can be difficult, especially if the definition, content, or properties of an event change over time.

This bachelor thesis focuses on the development and evaluation of an online clustering solution in which events are defined either as changes in existing clusters or as the formation of new clusters. The solution is a text mining software, which receives new news articles over a data stream and processes them. Articles are assigned to different clusters due to their similarity to other articles. The assumption is that very similar articles write about the same news story. In addition, the evaluation of the clustering quality is measured with a custom scoring function.

The first part of this work consists of determining a suitable data set, which will be the subject of the clustering and provides the ground truth for evaluating the results. The implemented solution uses HDBSCAN as the clustering method and compares it with the state-of-the-art method k-means. It turned out that the use of HDBSCAN has advantages over k-means in terms of both performance and precision. Furthermore, various text preprocessing methods and vector space models are evaluated, with Text Lemmatization and tf-idf providing the most promising results. Once applied in a simulated online setting, the final evaluation found that the noise rate in the overall clustering reduces the precision in the event detection.

The resulting precision of the clustering is 72% with a standard deviation of 12%. The precision for detecting new events results in 62% with a standard deviation of 43%. Detecting changes in existing events results in a precision 69% with a standard deviation of 16%. A continuation of this work should focus on improving the overall clustering to increase the precision of the event detection.

Preface

The following bachelor thesis *Dynamic Event Detection in Data Streams* was written as part of our computer science studies at the ZHAW Zurich University of Applied Sciences.

After our lectures on artificial intelligence, we realized that we wanted to deepen our knowledge in this area. This thesis was the perfect opportunity to increase our expertise on topics such as natural language processing and cluster analysis.

Special thanks go to our two supervisors, Dr. Andreas Weiler and Prof. Dr. Kurt Stockinger, for their ongoing and effective support during the writing of this thesis.

We would also like to thank our two lecturers Prof. Dr. Thilo Stadelmann and Prof. Dr. Mark Cieliebak for their lectures on artificial intelligence.

At last but not least, we would like to thank our fellow students and the entire ZHAW staff for our great time at ZHAW.

Contents

1 Introduction

1.1 Problem Formulation

While searching for events in a data stream, the definition of an event is not always given. Providing static definitions, as most approaches do, does not suffice for dynamic data streams, which change over time. Additionally, the behaviour of the data stream is an important factor in itself, since blockages of overflows in the system have to be prevented.

An example for a dynamic data stream can be found in a stream of news articles, which are published in irregular time intervals and different quantities over time. Detecting events based on an incoming stream of news articles is therefore a challenging task.

The goal is to develop and evaluate a methodology to detect events in a dynamic data stream of news articles.

1.2 Motivation

Today's environment is rapidly changing. Most news articles are not spread via published media anymore but digital. With the overwhelming amount of news, finding relevant news is difficult.

As reading all news articles is no option, an automated methodology becomes necessary. This is where our work becomes relevant. We want to detect events from a news data stream, in particular identifying new headlines.

Since our solution is based on text data, any data in text form is applicable. With technologies such as speech recognition, the data could also initially be acoustic and converted to text before being entered into our application.

This would open up use cases with smart speakers such as *Amazon Echo* or *Google Assistant*.

2 Theoretical Basics

In order to fully understand our work, it is important to ensure a few basics in the area of Natural Language Processing (NLP) and clustering methods. In this section we will explain how the most important techniques used in our thesis work.

2.1 Text Preprocessing

When working with text data, many algorithms and methods will need to distinguish words from another. In most cases this is done by creating a dictionary of all known words and *vectorizing* (see Section 2.2) the text data according to the dictionary.

While this works well in theory, we deal with tremendous amount of data in real world applications. This results in huge directories with many vector dimensions and does not only take up more disk space but also more text processing *(computation and comparison)* time.

What does that mean? Consider the following words:

1. switzerland
2. Switzerland
3. SWITZERLAND

Anyone of us will be able to extract the same information out of these three words: The country *Switzerland*. However, for machines the terms are different to another because they are written differently. That is why in most cases it makes sense to lowercase all text data before processing it. That way we can ensure that the above words all share the same meaning for a machine and thus reduce the dictionary size.

Exceptions There are a few use cases where lowercasing a text is not desired. For example, when trying to detect the writer's sentiment. Someone who would write a few or all words of a sentence in all uppercase, might be angrier than someone who does not.

Reducing the dictionary size Lowercasing text is just the beginning though. Depending on the size of a document, it might make sense to not use all text data inside a document. A good example for this would be books. Vectorizing books would take too much computational power and time to process. In such cases, the dictionary size needs to be reduced. This can be accomplished by processing a summary of the book instead of the book itself or by extracting the most relevant words from the whole book. The latter can be done with keyphrase extraction or with Named Entity Recognition (NER).

Normalizing text If we are not dealing with books but with articles or papers, there are also other alternatives to keyphrase extraction and NER. Using Text Stemming (Section 2.1.1) or Text Lemmatization (Section 2.1.2), we can simplify terms and group them together to reduce the dictionary size.

2.1.1 Text Stemming

Text Stemming is a form of Text Normalization which aims to simplify words by reducing the inflectional forms of each word into their word stems. For example, the words *connected*, *connecting* and *connection* share a similar meaning and could therefore be simplified to the base term **connect**.

The first paper describing a stemming algorithm was written by Julie Beth Lovins[1] as early as in 1968. In her algorithm she used an ordered list of 294 suffixes to strip them out and then applies one

of 29 associated application rules followed by a set of 35 rules to check if the remaining stem has to be modified further.

Lovins' stemming algorithm was very successful but got mostly replaced by M.F. Porters stemming algorithm[2] published in 1980. In his paper, M.F. Porter was able to process his suffix stripping algorithm in 6,370 out of 10,000 words and thereby reducing the vocabulary size by **one third**. The algorithm simply follows 5 steps with replacement and removal rules and is therefore very easy and efficient.

M.F. Porter improved his stemming algorithm even further by publishing the Porter2 stemming algorithm in 2002 which is widely known as the *Snowball stemming algorithm*[3].

There are even more stemming algoriths. Very well known are the Lancester stemming algorithm[4] and the WordNet stemming algorithm[5]. Since M.F. Porter's snowball stemming algorithm is the most widely used one, we decided to go with his stemming algorithm.

2.1.2 Text Lemmatization

Similar to *Text Stemming*, Text Lemmatization has the same goal to group together the inflected forms of a word but follows a different approach. Instead of processing terms with fixed steps and defined rules, Text Lemmatization normally includes a dictionary lookup for the words and also takes in consideration to which part of a sentence a term belongs to. This results in more accurate root terms but also requires for more computational power than Text Stemming. See Table 1 for a comparison.

#	**Original word**	**Stemmed**	**Lemmatized**
1	written	written	write
2	greatest	greatest	great
3	best	best	best
4	fastest	fastest	fastest
5	highest	highest	high
6	compute	comput	compute
7	computer	comput	computer
8	computed	comput	compute
9	computing	comput	compute
10	studies	studi	study
11	studying	studi	study
12	university	univers	university
13	universities	univers	university
14	universe	univers	universe
15	universal	univers	universal

Table 1: Comparison of Text Stemming and Text Lemmatization.

2.1.3 Keyphrase Extraction

When having to describe data, one common approach is to tag the data with keyphrases. Two of the most well-known tagging methods in social media are tagging content with keyphrases marked with hash tags and user names marked with at signs. Let us check the following Tweet from the *European Space Agency*[6]:

> This walking and hopping #**robot** is currently being tested in ESA's Mars Yard at our **@ESA_Tech** centre in the Netherlands. SpaceBok is a quadruped robot designed by a Swiss student team from **@ETH** and **@ZHAW**.

If we only read the hash tags and user names *#robot*, *@ESA_Tech*, *@ETH* and *@ZHAW*, we do not retrieve all information but we can already think of what the Tweet is about. These words would be our *keyphrases*.

Now, in above example the keyphrases were defined manually by a user. But in our data set (and most data sources), there are no keyphrases defined. Luckily there are different approaches on how to extract keyphrases from text data automatically.

We are using SGRank[7] in our thesis as it is currently one of the most used keyphrase extraction algorithms. Our goal is to check if working with keyphrases alone is more, less or equally accurate than with working the whole text.

2.1.4 Named Entity Recognition

Similar to keyphrase extraction, NER extracts relevant terms from a given text. However, it does not only extract terms but also states what kind of term it is. Consider following sentence:

> CERN in Geneva pays tribute to Murray Gell-Mann, who won the Nobel Prize in Physics in 1969.

When using spaCy's NER model, which is based on a transition-based Convolutional neural network (CNN)[8], we retrieve following entities:

- CERN (Organisation)
- Geneva (Location)
- Murray Gell-Mann (Person)
- the Nobel Prize in Physics (Work of art)
- 1969 (Date)

For comparison, when extracting the keyphrases automatically, we receive following terms:

- Nobel Prize
- Murray Gell
- tribute
- Mann
- Geneva
- Physics
- CERN

Comparing the numbers of the extracted terms solely, keyphrase extraction seems to have delivered a better job. However, when evaluating the results, we can see that NER *understood* the terms and their relations better.

Not only was it able to correctly keep the subject's name, *Murray Gell-Mann*, but also the type of Nobel prize.

2.2 Vector Space Model

Comparing text documents with each other is not a straightforward task. This is the reason why the Vector space model (VSM) was developed.

The VSM transforms text data to a numerical vector. This makes it possible to calculate the distances between different documents, which is essential for clustering.

2.2.1 Term Frequency

The most basic Vectorizer is the Term Frequency Vectorizer. After creating the dictionary of all used terms, it simply sums up the occurrences for each term. Consider following three sentences:

- Rosetta space probe scopes out landing zone.
- Landing site search for Rosetta narrows.
- Major Bank Shake-up At Bank of England.

After removing Stop words, we receive 13 unique terms over all sentences. As the Term Frequency Vectorizer simply sums the occurrences of each term up, the VSM appears as in Table 2.

Sentence	bank	england	landing	major	narrows	probe	rosetta	scopes	search	shake	site	space	zone
1	0.000	0.000	1.000	0.000	0.000	1.000	1.000	1.000	0.000	0.000	0.000	1.000	1.000
2	0.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000	0.000
3	2.000	1.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000

Table 2: Term frequency VSM.

We can quickly see that the first two sentences share a few terms while they do not share any terms with the third sentence. At the same time we see that the term *bank* has been used twice in the third sentence.

2.2.2 tf-idf

Relying on the Term Frequency alone is a good start but can surely be improved. Consider following scenario: We have a document *d1* with 100 terms, 10 of them (10%) belong to one specific term *t1*. In a second document *d2* with 1,000 terms, 10 of them (1%) belong to the same term *t1*. When using term frequency as a vectorizer, both documents will receive for the term *t1* a value of 10. However, in the first document *d1* the term should have received a higher value than in document *d2* as it covered a bigger percentage of the document.

This is what tf-idf tries to fix. Tf-idf was first introduced in 1975 by G. Salton, A. Wong and C. S. Yang[9] and defines the equation as displayed in Equation 1.

$$w_{x,y} = tf_{x,y} \cdot log(\frac{N}{df_x}) \tag{1}$$

where $tf_{x,y}$ is the term frequency of x in y, N the total number of documents and df_x the documents containing x.

However, it is important to note that nowadays there are different tf-idf implementations. As we are using the scikit-learn[10] library, the tf-idf implementation[11] of the library is slightly different, see Equation 2.

$$w_{x,y} = tf_{x,y} \cdot (log(\frac{1+N}{1+df_x}) + 1) \tag{2}$$

where $tf_{x,y}$ is defined as the equal number of times that term $_x$ occurs in document $_y$.

If we now calculate the tf-idf value for the term *bank* inside the third sentence, we receive following value: 3.386. What scikit-learn now does, is to normalize this value using the Euclidean norm:

$$v_{norm} = \frac{v}{||v||_2} = \frac{v}{\sqrt{{v_1}^2 + {v_2}^2 + \ldots + {v_n}^2}} \tag{3}$$

The resulting VSM is shown in Table 3.

Sentence	bank	england	landing	major	narrows	probe	rosetta	scopes	search	shake	site	space	zone
1	0.000	0.000	0.335	0.000	0.000	0.440	0.335	0.440	0.000	0.000	0.000	0.440	0.440
2	0.000	0.000	0.373	0.000	0.490	0.000	0.373	0.000	0.490	0.000	0.490	0.000	0.000
3	0.756	0.378	0.000	0.378	0.000	0.000	0.000	0.000	0.000	0.378	0.000	0.000	0.000

Table 3: tf-idf VSM.

As we can see in Table 3, the term *bank* has a rather high value in *sentence 3*. This is because the term is not present in *sentence 1* or *sentence 2* but appears twice in *sentence 3*. We can also observe that the term *rosetta* has slightly different values for *sentence 1* and *sentence 2*. The reason for this is that *sentence 2* has one term less than *sentence 1*, therefore the significance of one term is weighted heavier in *sentence 2*.

2.3 Clustering

Clustering finds similarities in different documents based on their content and groups them together. The challenge now arises to find an appropriate clustering method, which is able to work with data of varying densities, high dimensionality and data noise.

2.3.1 *k*-means Clustering

k-means clustering is an iterative clustering method which assigns all data points in a given data set into k clusters, where k is a predefined number of clusters in the data set.

How does *k*-means clustering work At the very beginning, k-means creates k centroids at random locations. It then repeats following instructions until reaching convergence:

- For each data point: Find the nearest centroid.
- Assign the data point to the nearest centroid (cluster).
- For each cluster: Compute a new cluster centroid with all assigned data points.

Advantages

- Very simple and easy to understand algorithm.

Disadvantages

- Initial (random) centroids have a strong impact on the results.
- The number of clusters (k) has to be known beforehand.
- Unable to handle noise (all data points will be assigned to a cluster).

2.3.2 DBSCAN

DBSCAN stands for *Density-Based Spatial Clustering of Applications with Noise* and is a density based clustering algorithm.

How DBSCAN works DBSCAN requires two parameters in order to work:

1. *epsilon* - The maximum distance between two data points for them to be considered as in the same cluster.
2. *min samples* - The number of data points a neighbourhood has to contain in order to be considered as a cluster.

Having these two parameters defined, DBSCAN will iterate through the data points and try to assign them to clusters if the provided parameters match. If a data point can not be assigned to a cluster, it will be marked as a noise point.

Data points that belong to a cluster but do not dense themselves are known as **border points**. Some border points could theoretically belong to two or more clusters if the distance from the point to the clusters do not differ.

Advantages

- Does not need to know the number of clusters beforehand.
- Is able to find shaped clusters.
- Is able to handle noise points.

Disadvantages

- DBSCAN is not entirely deterministic.
- Defining the right epsilon value can be difficult.
- Unable to cluster data sets with large differences in densities.

2.3.3 HDBSCAN

HDBSCAN is a hierarchical density-based clustering algorithm[12], based on DBSCAN and improves its sensitivity for clusters of varying densities. Defining an epsilon parameter, which acts as a threshold for finding clusters, is therefore no longer necessary. This makes the algorithm more stable and flexible for different applications.

How HDBSCAN works Since HDBSCAN is the focus of this thesis, we want to deliver a more detailed explanation of its inner workings, than for the previous clustering methods. The explanation is based on the documentation of the library providing the implementation for HDBSCAN[13].

HDBSCAN only requires one parameter to be set beforehand:

1. min cluster size - The number of data points a neighbourhood has to contain in order to be considered as a cluster.

The algorithm consists of five steps, which are as follows:

1. Transforming the space At its core HDBSCAN is a single linkage clustering, which is typically rather sensitive to noise. A single noise point between clusters could act as a bridge, which would result in separate clusters to be seen as one. To reduce this issue, the first step is to increase the distances of lower density points. Points with lower densities are therefore spread further apart and points with higher densities remain close to each other. This is achieved by to comparing the core distances between two points with the original distance to get the the mutual reachability distance. The core distance $core_k(x)$ is defined as the radius of a circle around point x, so that k neighbours are

contained within this circle. Figure 1 shows an example for the core distances of three points marked as green, blue and red.

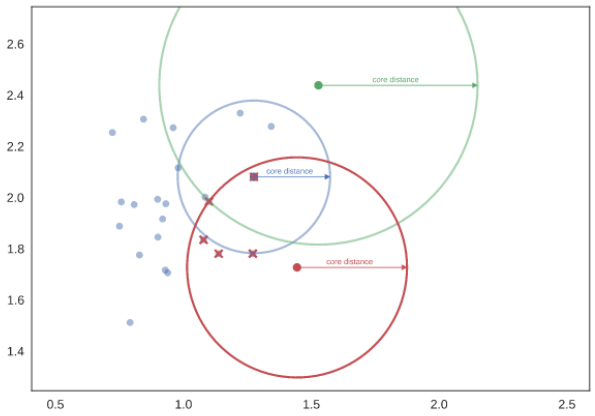

Figure 1: The core distances for three points shown as circles. Source[13]

Once the core distances are known, the mutual reachability distance between two points is defined as follows:

$$d_{mreach}(a,b) = max\{core_k(a), core_k(b), d(a,b)\}$$

where $d(a,b)$ is the original distance between a and b. If two points are close together, but the density around one point is rather low, the core distance will be greater than the original distance and thus the two points appear to be less close together when considering the mutual reachability distance. Considering the example from Figure 1, the mutual reachability distance between green and blue would be equal to the green core distance, since the blue core distance and the original distance are both smaller. The mutual reachability distance between green and red equals the original distance between these two points.

2. Build the minimum spanning tree Based on the mutual reachability distances, the next step is to find points close to each other. This is done by creating a minimum spanning tree, where edges are weighted according to the mutual reachability distance and a point is represented by a vertex. The minimum spanning tree is created one edge at a time, always choosing the lowest distance to a vertex not yet in the tree. This is done until each vertex is connected, which results in the minimal set of edges, such that dropping any edge will cause the disconnect of one or more vertices from the tree.

3. Build the cluster hierarchy Once the minimum spanning tree is complete, it is converted into a hierarchy of connected clusters. The conversion is done by sorting the edges of the tree by distance and creating a new cluster for each edge. The dendogram in Figure 2 shows a possible cluster hierarchy.

At this stage we have to flatten the hierarchy to get the final clusters, which provide the best representation of the data set. DBSCAN simply cuts through the hierarchy using a fixed parameter, usually called epsilon, to get the final clusters. This approach does not work well with clusters of varying densities and the epsilon parameter itself is unintuitive, requiring further exploration to find optimal values. This is were HDBSCAN improves upon DBSCAN, by taking additional steps for finding relevant clusters.

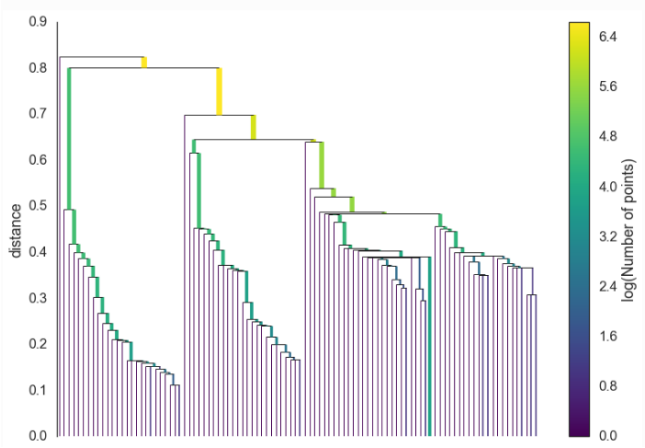

Figure 2: The cluster hierarchy shown as a dendogram. Source[13]

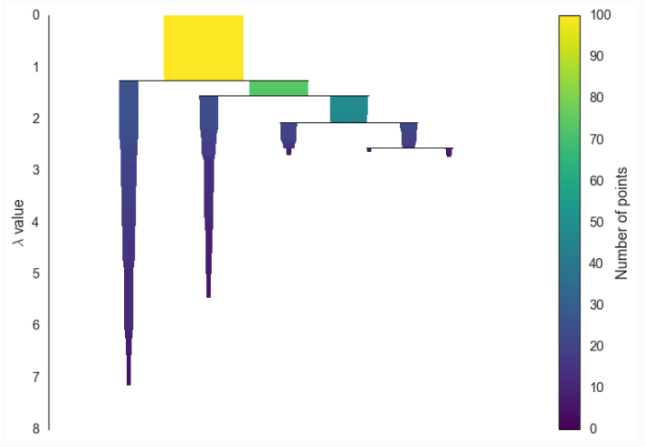

Figure 3: Condensed cluster hierarchy. Source[13]

4. Condense the cluster tree The fourth step consists of condensing the previously built cluster hierarchy into a smaller tree. The process starts at the top where all vertices still belong to the same cluster. Iterating through the hierarchy, for each split the two resulting clusters are compared against a predefined minimum cluster size. If the size of a cluster is below the minimum, its points will be discarded, while the other cluster remains in the parent cluster. If both cluster sizes are above or equal the minimum, the clusters are considered as true clusters and remain in the tree. This is repeated until no more splits can be made. Points discarded by this process are regarded as noise.

5. Extract the clusters The extraction of the final clusters from the condensed tree is based on the stability per cluster. The stability is based on the persistence of a cluster, which is measured by $\lambda = \frac{1}{distance}$. The stability for a cluster C is defined as

$$\sum_{p \in \mathrm{C}}^{|C|} (\lambda_p - \lambda_{birth}) \tag{4}$$

where λ_p describes when point p fell out of the cluster and λ_{birth} describes when the cluster was created. Calculating the stability for each cluster starts at the leaf nodes and ends when the root is reached. A cluster is selected if its stability is larger than the sum of stabilities of its children. Once it is selected, none of its subclusters can be chosen. If the sum of child stabilities is larger than that of its parent, the parent stability will be set to the value of the sum of its children, but no selection will be done. Once

the root node is reached all selected clusters are returned as the final clusters. As a result selected clusters can come from different levels in the hierarchy, which resemble different cluster densities.

3 Design and Implementation

The methodology consist of three parts, where each part builds upon the results from the previous one. Initially, the test data is created, which will be used for all evaluations. Secondly, we evaluate HDBSCAN and determine the optimal settings for our use case. The final part applies the results obtained from the previous evaluation in an online setting.

3.1 Dataflow

The dataflow, as shown in Figure 4, starts by receiving new news articles. The articles content will then be preprocessed (see Section 2.1) and vectorized. With the computed vectors, the articles will then be clustered to existing or new news stories. The cluster changes represent our detected events.

Figure 4: The dataflow of our application.

3.2 Data Set

Before any clustering method can be implemented or evaluated, it is important to rely on the right data set for training and evaluation.

3.2.1 Data Set Candidates

As our goal is to detect events in data streams, we have evaluated different data sets and their possibilities to extract events from their data themselves. Table 4 lists the evaluated data set candidates.

Data set	Number of rows	Description
GDELT 2.0[14]	over 575,000,000	Print and web news from around the world.
ChallengeNetwork[15]	4,449,294	Network packages including anomalies.
One Million Posts Corpus[16]	1,011,773	User comments to news articles.
Online Retail Data Set[17]	541,909	Customer retail purchases of one year.
News Aggregator Data Set[18]	422,937	Clustered news articles.
Dodgers Loop Sensor Data Set[19]	50,400	Number of cars driven through a ramp.
10k German News Articles[20]	10,273	German news articles.

Table 4: Evaluated data set candidates ordered by data set size.

We could extract events from all data sets mentioned in Table 4.
The extracted events could be as follows:

- Network packages
  - Cyber attacks depending on suspicious packets.
- User comments
  - Change of public opinion during time.
- Retail purchases
  - Change of purchasing behaviour based on product choices.
- Traffic
  - Traffic changes due to baseball games.

- News articles
  - Development of a certain news story.

However, from above data sets only two contained prelabeled clusters:

1. Dodgers Loop Sensor Data Set
   - 50,400 sensor activities belonging to 81 games.
2. News Aggregator Data Set
   - 422,937 news articles belonging to 7,231 news stories.

As we did not want to lose too much time in manually labeling data, we decided to go with one of these two. Regarding our two options, our choice was simple:
We went for the **The News Aggregator Data Set** since it not only provided more data, but our work built on the news articles use case could later be continued with real live data. The **GDELT 2.0** data set for example, provides around 1,000 to 2,000 new news articles every 15 minutes.

3.2.2 Data Retrieval

Unfortunately the data set did not contain the news articles themselves but rather only the URL's to the news articles. This was done so due to copyright restrictions on the content. Fortunately there are web scraping tools designed to retrieve the content from news portals specifically. We decided to use Newspaper3k[21], a Python3 library that allows us to retrieve the news article texts from the news portals easily.

The library only requires an URL to download and extract the news article from a website, see Listing 1.

```
1    from newspaper import Article
2
3    url = 'http://fox13now.com/2013/12/30/new-year-new-laws-obamacare-pot-
     guns-and-drones/'
4    article = Article(url)
5    article.download()
6    article.text # Contains the article's text.
```

Listing 1: Retrieve the news article from an URL.

All we had to do now, is to run this code for all news articles. To speed this process up, we loaded the data set into a database and ran 8 concurrent processes which retrieved the news articles content from the web portals in different batches.

3.2.3 Data Cleansing

The data set contains news articles collected from March 10th to August 10th 2014. Five years later, many resources are not online anymore or are not accessible from Europe due to Europe's General Data Protection Regulation (GDPR). We have used the SQL query in Listing 2 to filter out news articles that were most likely corrupt.

```
1    SELECT *
2    FROM news_article
3    WHERE
```

```
        newspaper_text IS NOT NULL
        AND TRIM(COALESCE(newspaper_text, '')) != ''
        AND hostname NOT IN ('newsledge.com', 'www.newsledge.com')
        AND newspaper_text NOT LIKE '%GDPR%'
        AND newspaper_text NOT LIKE '%javascript%'
        AND newspaper_text NOT LIKE '%404%'
        AND newspaper_text NOT LIKE '%cookie%'
        AND newspaper_keywords NOT LIKE '%GDPR%'
        AND newspaper_keywords NOT LIKE '%javascript%'
        AND newspaper_keywords NOT LIKE '%404%'
        AND newspaper_keywords NOT LIKE '%cookie%'
        AND title_keywords_intersection = 1
```

Listing 2: Retrieve valid news articles.

From the original 422,937 news articles, 235,070 were still accessible to us. They belong to 7'183 different news stories.

3.2.4 News Article Example

It is important to check and verify the data before working with it. See following news article we have scraped from thelocal.ch[22]:

> The surrealist painter, sculptor and set designer, known as H.R. Giger, died in hospital following a fall, according to the report.
>
> A native of the canton of Graubünden, he was best known for his design of Alien, an American science-fiction horror film from 1979 directed by Ridley Scott, and was part of the special effects team that won an Academy Award the following year.
>
> Giger's design for the film was inspired by his painting Necronom IV.
>
> He was known for his airbrushed silver and grey canvasses depicting nightmarish dreamscapes and "biomechanical" human figures linked to machines.
>
> Born in Chur in 1940, he studied architecture and industrial art at Zurich's University of Applied Sciences (ZHAW).
>
> A friend of Timothy Leary who was influcenced by surrealist painter Salvador Dali, he reportedly suffered from nightmares that he said inspired some of his work.
>
> Books of his paintings and use of his art for music albums and in publications, such as the American science and science fiction Omni, led to his rise in international fame.
>
> His artwork adorned the cover of albums by artists such as Emerson Lake & Palmer, the Dead Kennedys and French singer Mylène Farmer.
>
> In addition, to Alien, Giger was also involved in films such as Poltergeist II (1986), Alien III (1992) and Species (1995).
>
> In 1998, Giger acquired the Château St. Germain in Gruyères, a village in the canton of Fribourg, to house his work in the H.R. Giger Museum.
>
> He lived and worked in Zurich-Seebach, SRF said.
>
> For all his fame, Giger said he did not gain "big money" from his work in the movies, Zurich newspaper Tages Anzeiger reported.
>
> Others, he said, cashed in on his creative work.

> "My design was done and changed," Giger is quoted as having said. "The film business is a gangster business."

The news article was scraped successfully without any data noise which would not belong to the article.

Text Stemming The stemmed version of the text is:

> the surrealist painter, sculptor set designer, known h r giger, die hospit follow fall, accord report a nativ canton graubünden, best known design alien, american science-fict horror film 1979 direct ridley scott, part special effect team academi award follow year giger design film inspir paint necronom iv he known airbrush silver grey canvass depict nightmarish dreamscap “biomechanical” human figur link machin born chur 1940, studi architectur industri art zurich univers appli scienc (zhaw) a friend timothi leari influcenc surrealist painter salvador dali, report suffer nightmar said inspir work book paint use art music album publications, american scienc scienc fiction omni, led rise intern fame his artwork adorn cover album artist emerson lake & palmer, dead kennedi french singer mylèn farmer in addition, alien, giger also involv film poltergeist ii (1986), alien iii (1992) speci (1995) in 1998, giger acquir château st germain gruyères, villag canton fribourg, hous work h r giger museum he live work zurich-seebach, srf said for fame, giger said gain "big money" work movies, zurich newspap tage anzeig report others, said, cash creativ work "mi design done changed," giger quot said "the film busi gangster busi "

Text Lemmatization The lemmatized version of the text is:

> the surrealist painter , sculptor and set designer , know as h.r.giger , die in hospital follow a fall , accord to the report. a native of the canton of graubünden , he was best known for his design of alien , an american science - fiction horror film from 1979 direct by ridley scott , and was part of the special effect team that win an academy award the following year. giger 's design for the film was inspire by his painting necronom iv. he was know for his airbrushed silver and grey canvass depict nightmarish dreamscape and " biomechanical " human figure link to machine. bear in chur in 1940 , he study architecture and industrial art at zurich 's university of applied sciences (zhaw). a friend of timothy leary who was influcence by surrealist painter salvador dali , he reportedly suffer from nightmare that he say inspire some of his work. book of his painting and use of his art for music album and in publication , such as the american science and science fiction omni , lead to his rise in international fame. his artwork adorn the cover of album by artist such as emerson lake & palmer , the dead kennedys and french singer mylène farmer. in addition , to alien , giger was also involve in film such as poltergeist ii (1986) , alien iii (1992) and species (1995).in 1998 , giger acquire the château st.germain in gruyères , a village in the canton of fribourg , to house his work in the h.r.giger museum. he live and work in zurich - seebach , srf say. for all his fame , giger say he did not gain " big money " from his work in the movie , zurich newspaper tages anzeiger report.others , he say , cash in on his creative work." my design was done and change , " giger is quote as having say." the film business is a gangster business."

Keyphrase extraction The extracted keyphrases from this article are:

> Zurich newspaper Tages Anzeiger, french singer Mylène Farmer, surrealist painter Salvador Dali, painting Necronom IV, Zurich 's University, special effect team, american science, science fiction Omni, fiction horror film, H.R. Giger, Giger, Château St. Germain, H.R. Giger Museum, Ridley Scott, following year, human figure, nightmarish dreamscape, Academy

Award, canton, grey canvass, Applied Sciences industrial art, Timothy Leary, airbrushed silver, design, Alien, music album, international fame, Dead Kennedys, Emerson Lake, Alien III, Poltergeist II, film, Graubünden, work, known, native, fall, report, hospital, Chur, architecture, Born,biomechanical, machine, big money, friend, ZHAW, album, fame, Books, nightmare, publication, artist, cover, artwork, Palmer, rise, addition, creative work, specie, Fribourg, village, Gruyères, SRF, Seebach, designer, movie, film business, gangster business, sculptor

Named Entity Recognition The recognized named entities are:

H.R., Alien, Ridley, Necronom, Zurich, ZHAW, Timothy, Salvador, Omni, Emerson, Dead, Mylène, Alien, Alien, Species, Gruyères, H.R., SRF, Giger, Tages

Conclusion Every text preprocessing method worked as expected. Text Lemmatization correctly converted the terms to their dictionary base roots which results in quite readable text. As Text Stemming simply cuts off term endings based on a static rule set, the result is less good readable than the lemmatized text. Keyphrase extraction was able to extract 71 terms, while only 20 Named Entities were recognized.

3.3 Clustering Evaluation

3.3.1 Design

The clustering evaluation will focus on HDBSCAN as the main clustering method. The properties of HDBSCAN are a good fit for our use case of clustering news articles in an online setting. It is able to work with clusters of varying densities, does not require to know the number of clusters in advance and is efficient with a time complexity of $O(nlog(n))$. The density requirement is based on our test data, which contains clusters with sizes ranging from 2 to over 400. In addition, we do not know the number of clusters in the data stream and can therefore not provide an approximation beforehand. This already disqualifies a wide range of clustering methods such as k-means. Another feature of HDBSCAN is its ability to discard samples as noise. This can be a useful feature once applied to an online setting, since noisy data is to be expected in real world data streams.

The goal of the clustering evaluation is to find the optimal parameters and preprocessing methods for running HDBSCAN on our test data and to measure the overall performance. The clustering evaluation is designed to run HDBSCAN using a combination of different text preprocessing methods, vectorizers and parameters. k-means is used to provide a benchmark for the HDBSCAN evaluation. Once a clustering has been performed, the result is measured based on the ground truth and stored in a database for later analysis.

An important consideration is the variety of samples to use for a single clustering. Using one set of samples might bias the score against this specific set of samples and some methods might perform better or worse depending on the samples. To introduce variability, while still retaining repeatability, an evaluation run will be repeated multiple times. Each repetition will load a new set of samples, iterating linearly through the data set. For example, if we define the number of repetitions as 2 with a sample size of 1,000, the evaluation will be done on the first 1,000 samples with all possible settings and the second run will load the next 1,000 samples, thus containing samples with indices ranging from 1,001 to 2,000. The reason we do not load random sets of samples is repeatability. If we make any changes in the implementation or the scoring function, we want to be able to compare the new results with the previous ones in a deterministic manner.

Furthermore, news articles are loaded based on their stories. An evaluation with 30 stories will actually load all news articles belonging to these stories. This is done to prevent stories from being cut off,

resulting in potential noise.

3.3.2 Scoring Function

The scoring function is essential for evaluating the result of a clustering method. The score should reflect the quality of the individual clusters and of the clustering as a whole. The number of existing measures for clustering is vast and can be split into three main categories:

- Internal measures determine the score based on criteria derived from the data itself.
- External measures depend on criteria non-existent in the data itself such as class labels.
- Relative measures compare different clusterings with each other.

Since the ground truth is known in our test data, we are going to apply an external measure.

Initially we used Normalized Mutual Information (NMI) as our primary scoring function. NMI is an entropy-based measure and tries to quantify the amount of shared information between predicted clusters and the ground truth. The score proved to work well for our initial evaluations, but upon closer inspection certain anomalies were found. An example is given in Table 5, where k-means achieved a rather high score, regardless of the large difference between the true amount of clusters and the approximated number of clusters using $\sqrt{n}$. We were not able to explain this behaviour, although there are multiple papers about the selection bias of NMI for higher numbers of clusters[23], [24]. Other scoring functions such as V-Measure or the Adjusted Rand Index (ARI) showed similar unexpected results with different clusterings. Therefore, we decided to develop our own scoring function based on the ideas of Maximum Matching[25] and the Jaccard Index, which we call MP-Score.

Algorithm	**Sample Size**	**NMI**	$\mathbf{n_{true}}$	$\mid \mathbf{n_{true}} - \mathbf{n_{predicted}} \mid$
k-means	19255	0.754	600	457
HDBSCAN	19255	0.742	600	2

Table 5: k-means has a higher NMI score than HDBSCAN, while having a much larger difference in number of clusters.

Calculating the score The scoring function first calculates the similarity between pairs of clusters, where each cluster belongs to a different clustering. We use the Jaccard Index to measure the similarity, which is defined as

$$\frac{|A \cap B|}{|A \cup B|} \tag{5}$$

To illustrate the process we start with an example. We define T and C as our clusterings, where T is the ground truth and C is the predicted clustering. The clusterings are defined as follows:

$$T = \{\{1,2,3\},\{4,5,6,7\},\{8,9\}\}$$
$$C = \{\{1,2\},\{3,4,5,6\},\{7\},\{8,9\}\}$$

We calculate the similarity as defined in Equation 5, for each possible pair of clusters between T and C starting with $t_1 = \{1,2,3\}$ and $c_1 = \{1,2\}$:

$$similarity(t_1, c_1) = \frac{|t_1 \cap c_1|}{|t_1 \cup c_1|} = \frac{|\{1,2\}|}{|\{1,2,3\}|} = \frac{2}{3} = 0.667$$

After doing this for each possible pair, we get the similarity matrix A:

$$A = \begin{pmatrix} similarity(t_1, c_1) & \dots & \dots & similarity(t_1, c_4) \\ \vdots & \vdots & \vdots & \vdots \\ similarity(t_3, c_3) & \dots & \dots & similarity(t_3, c_4) \end{pmatrix} = \begin{pmatrix} 0.667 & 0.167 & 0 & 0 \\ 0 & 0.6 & 0.25 & 0.4 \\ 0 & 0 & 0 & 1.0 \end{pmatrix}$$

As a next step, we have to select the most relevant similarity values from each row of the similarity matrix.

Finding relevant values in the similarity matrix is non-trivial, since clusters do not share labels across different clusterings. To solve this, we make two assumptions based on the principle of Maximum Matching:

1. The higher the similarity between two clusters, the more likely it is that both clusters are describing the same group of documents.
2. Each cluster can be associated with a cluster from another clustering only once.

Based on these assumptions we select the highest similarity value per row, whose column is not already associated with another row. Applying this selection function f to our previously calculated similarity matrix A, results in the set containing the most relevant similarity values.

$$f(A) = \begin{pmatrix} \mathbf{0.667} & 0.167 & 0 & 0 \\ 0 & \mathbf{0.6} & 0.25 & 0.4 \\ 0 & 0 & 0 & \mathbf{1.0} \end{pmatrix} = \{0.667, 0.6, 1\}$$

As we can see, there were no collisions between columns and we simply get the highest value per row. Consider the following example with a different similarity matrix B, which does contain a collision:

$$f(B) = \begin{pmatrix} \mathbf{0.75} & 0.375 & 0.427 & 0.375 \\ 0.4 & \mathbf{0.667} & 0.571 & 0.8 \\ 0.333 & 0.25 & 0.4 & \mathbf{1.0} \end{pmatrix} = \{0.75, 0.667, 1\}$$

The selected similarity for the second row is 0.667 instead of 0.8. This is because the fourth column is already associated with the third row, while having a similarity greater than 0.8. Based on our assumption that clusters cannot be associated twice, the second highest similarity is used for the second column. In case no association could be found, the value would be set to zero.

In a third step we have to calculate the weights to be used for the weighted average. The weight is based on the number of elements inside the cluster and necessary to represent differences in predicted and true number of clusters in the final score. It is defined as follows:

$$w_{ij} = \frac{|t_i| + |c_j|}{|T| + |C|} \tag{6}$$

where T is the ground truth with $t_i \in T$ and C the predicted clustering with $c_j \in C$. The weight for a pairing $t_i c_j$ includes both the size of the true cluster and the size of the predicted cluster. The reason both sizes are used, is that we want to reflect the difference between the number of predicted clusters and the ground truth. Using only the true number of elements as the weight, would affect the score if $|C| < |T|$, but not $|C| > |T|$. Hence, the number of predicted elements has to be included into the weight.

To continue our example, we have to calculate the weights based on the coordinates of the selected values from our similarity matrix A. The coordinates are $(1,1),(2,2),(3,4)$. As a result we calculate the following weights:

$$w_{1,1} = \frac{|t_1| + |c_1|}{|T| + |C|} = \frac{3+2}{9+9} = \frac{5}{18} = 0.278$$
$$w_{2,2} = \frac{|t_2| + |c_2|}{|T| + |C|} = \frac{4+4}{9+9} = \frac{8}{18} = 0.444$$
$$w_{3,4} = \frac{|t_3| + |c_4|}{|T| + |C|} = \frac{2+2}{9+9} = \frac{4}{18} = 0.222$$

In the fourth and final step, we calculate the weighted average

$$\text{MP-Score} = \sum_{i=0}^{|S|} w_i s_i \{w_i \in W \wedge s_i \in S\} \tag{7}$$

where S contains the selected values from the similarity matrix with $s_i \in S$ and W is the set of weights with $w_i \in W$. Using our previously selected similarity values $S = f(A) = \{0.667, 0.6, 1\}$ and the corresponding weights $W = \{0.278, 0.444, 0.222\}$, the the final average is calculated as follows:

$$\text{MP-Score} = (0.278 \cdot 0.667) + (0.444 \cdot 0.6) + (0.222 \cdot 1) = \mathbf{0.674}$$

The final score for the evaluation of the predicted clustering C with the true clustering T is 0.674. The complete implementation of the scoring function can be found in the appendix as Listing 6.

Comparison against other measures The test scenarios in Table 6 show the resulting scores of our MP-Score, NMI and ARI. The second scenario results in a ARI score of 0.308, while the NMI with 0.564 and the MP-Score with 0.637 are both higher. Intuitively we would assume the higher score to better represent the predicted clustering, since the number of clusters is correct and only two out of nine elements are assigned to the wrong cluster. Scenario six shows a similar case. Scenarios four and especially five show the previously mentioned bias of NMI with regards to higher numbers of clusters. The seventh scenario gives an interesting result, where both NMI and ARI result in zero while the MP-Score results in 0.321. The relatively high MP-Score is because a true cluster with four elements is matched with the predicted cluster containing nine elements. This results in a similarity of 0.444, which is then lowered by the weight. The NMI does not infer any entropy, since every element ends up in the same cluster independently from the ground truth. The ARI results in 0 because of its

adjustment for chance. Overall, the MP-Score behaves rather intuitively, although it is not corrected for randomness. This means even a completely random clustering would result in a MP-Score greater than 0, while an adjusted measure such as the ARI would be 0 in such case.

Test scenarios with ground truth $T = \{\{1,2,3\},\{4,5,6,7\},\{8,9\}\}$				
Nr.	Predicted Clustering C	NMI	ARI	MP-Score
1	$C = \{\{1,2,3\},\{4,5,6,7\},\{8,9\}\}$	1.0	1.0	1.0
2	$C = \{\{1,2\},\{3,4,5,6\},\{7,8,9\}\}$	0.564	0.308	0.637
3	$C = \{\{1,2,3\},\{4,5,6\},\{7\},\{8,9\}\}$	0.895	0.771	0.847
4	$C = \{\{1,2,3\},\{4,5\},\{6,7\},\{8\},\{9\}\}$	0.821	0.591	0.583
5	$C = \{\{1\},\{2\},\{3\},\{4\},\{5\},\{6\},\{7\},\{8\},\{9\}\}$	0.651	0	0.227
6	$C = \{\{1,2,3,4,5\},\{6,7,8,9\}\}$	0.434	0.182	0.433
7	$C = \{\{1,2,3,4,5,6,7,8,9\}\}$	0.0	0	0.321
8	$C = \{\{7,2,4\},\{8,9,6,3\},\{1,5\}\}$	0.219	-0.108	0.392

Table 6: Direct comparison of different scoring functions.

We repeated the evaluation shown in Table 5 a second time using the MP-Score. The score (Table 7) for k-means is now considerably lower than HDBSCAN. This reflects what we would expect based on the difference in the number of predicted clusters.

Algorithm	**Sample Size**	**MP-Score**	$\mathbf{n_{true}}$	$\mid \mathbf{n_{true}} - \mathbf{n_{predicted}} \mid$
k-means	19255	0.137	600	457
HDBSCAN	19255	0.605	600	2

Table 7: The MP-Score reflects the difference in number of predicted clusters.

3.3.3 Implementation

The evaluation process is done with our own evaluation framework. The framework allows for automated and repeatable evaluation runs. Results are stored in a database for later analysis. The main features include:

- Defining the number of stories to run the evaluation with and load all news articles from those stories.
- Repeating evaluation runs with different sets of data.
- Providing different vectorizers for converting the textual data into a VSM.
- Defining a range for each parameter of a clustering method and running it with each possible combination of those parameters.
- Storing the result in a database and creating relations between news articles, clusters and evaluation runs. This allows for manual inspection and analysis of individual articles inside a predicted cluster.

The implementation was done with Python. Clustering methods and vectorizers are provided by the scikit-learn library[10], while the specific HDBSCAN implementation is provided by a scikit-learn-contrib package[12]. scikit-learn-contrib is a collection of high quality third-party projects compatible with scikit-learn. We decided to use scikit-learn because of its rich documentation, the wide range of tools and algorithms it provides for clustering and our previous experience with it.

The framework runs in a fully Dockerized environment, which includes the database. This allows the framework to run independently from the underlying host, as long as the host supports Docker. This

principle is useful for developing and testing the framework in a local environment and deploying it on a remote server for long running evaluations, without having to worry about setting up and installing all dependencies.

The parameters for each available clustering method are defined beforehand in a dictionary as can be seen in Listing 3. Parameters are defined as a list of possible variations. For example, if we want to run HDBSCAN with two different metrics *cosine* and *euclidean*, we define the metric parameter as `"metric": ["cosine", "euclidean"]`. When running a clustering method, it will be executed with each possible combination of parameters. This means a single evaluation of HDBSCAN, will include 16 different runs, since there are 2 different metrics and 8 different options for *min_cluster_size*. This is important to consider for running clustering methods with long processing times or running evaluations on large sample sizes.

```
    parameters_by_method = {
        self.kmeans: {
            "n_cluster": ["n_square", "n_true"]
        },
        self.hdbscan: {
            "min_cluster_size": range(2, 10),
            "metric": ["cosine", "euclidean"]
        },
        self.meanshift: {"cluster_all": [True, False]},
        self.birch: {
            "branching_factor": range(10, 100, 10),
            "threshold": range(2, 6),
        },
        self.affinity_propagation: {
            "affinity": ["euclidean"],
            "convergence_iter": [15],
            "damping": np.arange(0.5, 0.9, 0.1),
            "max_iter": [50, 100, 200, 500],
        },
        self.spectral_clustering: {
            "affinity": ["rbf"],
            "assign_labels": ["kmeans", "discretize"],
        },
    }
```

Listing 3: Predefined parameters for different clustering methods.

CLI The evaluation framework provides a command line interface to start evaluation runs and specify relevant arguments. Listing 4 shows the full interface.

```
usage: cluster_evaluation_framework.py [-h] [--rows ROWS]
                                       [--stories STORIES]
                                       [--methods METHODS]
                                       [--vectorizers VECTORIZERS]
                                       [--tokenizers TOKENIZERS]
                                       [--runs RUNS]

Run different clustering methods, with a variety of different settings.

optional arguments:
```

```
11  -h, --help            show this help message and exit
12  --rows ROWS           number of samples to use for clustering
13                        default: 1000
14  --stories STORIES     number of stories to load samples from. This
     parameter overrides the rows parameter if set.
15  --methods METHODS     options: kmeans, hdbscan, meanshift, birch,
     affinity_propagation, spectral_clustering
16                        default: all available options
17  --vectorizers VECTORIZERS
18                        options: CountVectorizer, TfidfVectorizer
19                        default: all available options
20  --tokenizers TOKENIZERS
21                        options: newspaper_text, text_keyterms,
     text_entities, text_keyterms_and_entities,
     text_lemmatized_without_stopwords, text_stemmed_without_stopwords
22                        default: all available options
23  --runs RUNS           number of runs per clustering method
24                        default: 1
```

Listing 4: Command line interface for the evaluation framework.

3.4 Online Clustering

3.4.1 Design

Detecting events in a stream of news articles will be achieved by using an online clustering approach. An event is described by the occurrence of multiple news articles about the same story. The events of interest for this application are the discovery of new stories and the extension of existing stories. Thus we define our two types of events as follows:

- New event: A new cluster of news articles appears in the data stream, which write about the same story.
- Event extended: An existing story is extended by additional news articles.

HDBSCAN will be applied as the clustering method, using the optimal settings discovered in the clustering method evaluation. Additional preprocessing of news articles before clustering is going to be explored as part of evaluation as well and will be implemented accordingly for the online clustering.

The clustering will be done in batches over time, since HDBSCAN only supports static data sets. Events are detected by comparing clusters from successive batches. Figure 5 illustrates this with three batches, where the resulting clusters from batch t will be compared with the previous result from batch $t-1$, while batch $t-1$ was previously compared with batch $t-2$.

In this example, each batch only contains samples from a limited time period, where $\triangle t$ stands for the time period between batches. Since a batch does not contain the full set of samples, we have to consider the overlap between batches. The size of the overlap is essential to find similar clusters from different batches. If a similar cluster already exists in the previous batch, the differences between these clusters are detected as a change in an existing event. If no pair exists for a cluster from a current batch, this cluster will be regarded as a new event. The similarity between clusters is based on the same assumptions as for the scoring function described in Section 3.3.2.

The overlap between batches depends on the batch size and the number of new samples in $\triangle t$. A high volume of incoming samples combined with a small batch size would result in an overlap too small to find pairs of clusters. All clusters from the current batch would be detected as new in such case. To

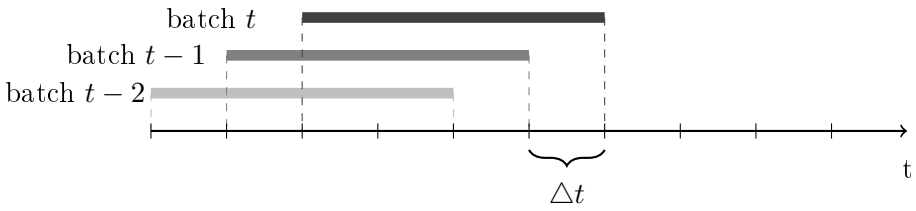

Figure 5: Timeline showing the sliding window approach.

decrease the negative impact on the event detection by peaks in the data stream, we need a batch size which grows accordingly. The ideal batch size provides enough overlap between batches to find pairs of clusters and is small enough to allow efficient processing. We explore three different methods for determining the ideal batch size:

1. **Fixed size**: The first method uses a fixed batch size, where each batch processes the most recent n samples. This makes the clustering unstable against sudden peaks in the volume of incoming data, but we consider this method as an useful benchmark for the dynamic methods.
2. **Size by hours**: This method uses a dynamic batch size by loading the samples from the last n hours. This enables the batch size to increase and decrease with the volume of the data stream. The number of samples will be limited by an upper bound, to keep the space and time consumption of the clustering method reasonable.
3. **Size by incoming data**: This method defines the batch size relative to the incoming stream of data. We count the number of new samples since the last batch and multiply it by a predefined factor. For example, if we want the new samples to be 1% of the overall clustering, we define the factor as 100. The batch size will be limited by a lower and an upper bound. The lower bound prevents the overlap between batches from getting too small. The upper bound is based on the same reasoning as mentioned in the previous method.

The evaluation will use the MP-Score to measure the precision of the event detection, since our model represents events as clusters. True events can be extracted directly from the ground truth based on news articles from two successive batches.

3.4.2 Implementation

The online clustering implemented for this thesis does not operate in a true online setting, but rather in a simulated data stream over time. The simulated approach allows us to directly compare the resulting events with the ground truth and thus evaluate different settings. The implementation was done with Python and runs in a dockerized environment similar to the evaluation framework.

The detection of events relies on comparing clusters between successive batches. We apply Locality Sensitive Hashing (LSH)[26] to find clusters most similar to each other. In its essence, LSH is an efficient way to find similar documents, which does not require to calculate the similarity between each possible pair of documents. The implementation for LSH is provided by the datasketch library[27].

Once we have found pairs of clusters which represent the same story, detecting events becomes trivial. For each pair we subtract the cluster from the previous batch from the current cluster. The resulting set contains all news articles, which are only present in the current cluster. These articles are then summarized as a change of an existing event. If the previous batch did not contain any similar clusters for a cluster from the current batch, the cluster is considered as a new event.

Since events are clusters of news articles themselves, we apply the MP-Score to measure the precision of detected events. Calculating the score has a time complexity of $O(n^2)$, but since the application

runs on a simulated timeline and the scoring is only part of the evaluation, time complexity is a minor concern in this case.

CLI The application provides a command line interface to run the simulation with different parameters such as the start date, number of days to run and the batch size.

```
usage: online_clustering.py [-h] [--verbose] [--persist_in_db]
                            [--rows ROWS] [--hours HOURS]
                            [--factors FACTORS] --date DATE
                            [--run_n_days RUN_N_DAYS]
                            [--threshold THRESHOLD]

Run the batchwise clustering over a simulated stream of news articles.

optional arguments:
-h, --help              show this help message and exit
--verbose               default: False
--persist_in_db         default: False
--rows ROWS             numbers of samples to process per batch
--hours HOURS           numbers of hours to load samples
--factors FACTORS       factor to use for relative batch sizes
--date DATE             start date
--run_n_days RUN_N_DAYS number of days to run the batchwise clustering
                        default: 1
--threshold THRESHOLD   similarity threshold for cluster matching
                        default: 0.75
```

Listing 5: Command line interface for the online clustering.

4 Results

4.1 Clustering Evaluation

The goal of this evaluation is to measure the precision of HDBSCAN with different parameters and preprocessing methods. The most suitable settings will then be used for the online clustering approach to detect events in a stream of news articles. The precision is measured with the MP-Score.

4.1.1 Setup

Text preprocessing The first step in working with text is to apply preprocessing techniques to improve the quality of the data before clustering it. We evaluate five different preprocessing methods as described in Section 2.1. The methods are:

- Full text with stop word removal
- Text Stemming
- Text Lemmatization
- Keyphrase extraction
- NER

Text vectorization Before text can be clustered, it has to be transformed into a VSM. We compare two different models:

- Term frequency
- tf-idf

Parameters HDBSCAN has a range of parameters which can be tuned to fit our data set. We focus on the two primary ones:

- Min cluster size: The minimum size of a cluster. We run the evaluation with a range from 2 to 9 as the $min\_cluster\_size$.
- Metric: The distance measure between points. We apply the metrics *cosine* and *euclidean*.

The primary parameter for k-means is the number of clusters. Since k-means is used as a benchmark to evaluate HDBSCAN, we provide the true number of clusters for each run. Therefore k-means runs with an optimal starting point.

Running evaluations The evaluation is done with different sets of news articles per run. If we define a run to use 30 stories and set it to repeat five times, each repetition will load 30 different stories from the data set. This is done to get a more diverse set of samples. Each run will be repeated at least five times. Lower numbers of stories allow for more repetitions due to lower processing times.

4.1.2 Evaluation

Preprocessing times We have measured the time consumption of each text preprocessing method. As we can see in Table 8, Text Stemming is by far the fastest method of the tested ones.

Text Stemming needs in average 2.3 seconds to process one article of our data set. Text Lemmatization needs in average 75× more time to process a news article and is the slowest evaluated text preprocessing method.

	Minimum time (s)	Average time (s)	Maximum time (s)
Text Stemming	0.1	2.3	4.0
Text Lemmatization	113.0	172.3	1414.1
Keyphrase extraction	8.5	83.5	954.3
NER	8.7	58.2	469.3

Table 8: Comparison of text preprocessing times.

Comparison of clustering settings The first run was done with 60 stories and 20 repetitions which resulted in approximately 2,000 news articles per run. Table 9 shows the resulting MP-Scores for each parameter in combination with each preprocessing method and VSM.

The highest score per parameter is highlighted as bold and the highest score overall is underlined. The first insight we get is the variety of scores for different minimum cluster sizes. The lowest minimum cluster size results in the lowest score, while increasing this parameter leads to an increasingly better score. The highest score is reached with a minimum cluster size of 6. Increasing the minimum cluster size further reduces the score again. The large difference in scores between different minimum cluster sizes, shows the importance this parameter has on the quality of the clustering and requires some knowledge of the data beforehand. In our case, we have a wide range of different cluster sizes as shown in Figure 6, with clusters containing as few as two news articles. Based on this distribution we expected the ideal minimum cluster size to be in a range from 2 to 9, which is why we chose this range initially. The distribution further explains the decrease in the scores after a minimum cluster size of 6, since an increasingly number of clusters are being ignored.

Clustering	**Word Frequency**					**tf-idf**				
HDBSCAN	Full Text	Keyphrases	NER	Lemmatized	Stemmed	Full Text	Keyphrases	NER	Lemmatized	Stemmed
min_size: 2, metric: cosine	0.446	0.456	0.409	0.452	0.451	0.477	0.450	0.398	**0.499**	0.479
min_size: 2, metric: euclidean	0.071	0.068	0.090	0.075	0.073	0.459	0.255	0.444	**0.482**	0.481
min_size: 3, metric: cosine	0.603	0.592	0.558	0.594	0.599	0.624	0.594	0.547	**0.640**	0.63
min_size: 3, metric: euclidean	0.071	0.067	0.090	0.073	0.073	0.595	0.304	0.549	0.609	**0.613**
min_size: 4, metric: cosine	0.656	0.639	0.613	0.647	0.657	0.684	0.654	0.604	**0.691**	0.686
min_size: 4, metric: euclidean	0.062	0.062	0.084	0.064	0.063	0.633	0.310	0.574	0.645	**0.652**
min_size: 5, metric: cosine	0.678	0.668	0.632	0.674	0.681	0.712	0.677	0.630	**0.725**	0.721
min_size: 5, metric: euclidean	0.048	0.057	0.081	0.051	0.051	0.650	0.303	0.578	0.66	**0.674**
min_size: 6, metric: cosine	0.695	0.672	0.636	0.685	0.686	0.731	0.695	0.630	**0.738**	0.735
min_size: 6, metric: euclidean	0.038	0.052	0.074	0.041	0.039	0.651	0.283	0.570	0.638	**0.684**
min_size: 7, metric: cosine	0.690	0.679	0.634	0.687	0.687	0.727	0.685	0.631	**0.737**	0.734
min_size: 7, metric: euclidean	0.032	0.049	0.072	0.034	0.033	0.654	0.269	0.555	0.659	**0.676**
min_size: 8, metric: cosine	0.683	0.669	0.628	0.683	0.685	0.729	0.689	0.626	**0.733**	0.733
min_size: 8, metric: euclidean	0.031	0.042	0.068	0.032	0.031	0.644	0.252	0.540	0.649	**0.668**
min_size: 9, metric: cosine	0.679	0.666	0.622	0.674	0.677	0.723	0.680	0.621	**0.732**	0.726
min_size: 9, metric: euclidean	0.029	0.036	0.064	0.031	0.032	0.640	0.234	0.527	0.648	**0.660**
***k*-means**										
n_cluster: n_true	0.364	0.437	0.289	0.358	0.361	0.643	0.632	0.466	**0.651**	0.649

Table 9: The average MP-Score for combinations of parameters and preprocessing methods with a sample size of 60 stories (approx. 2,000 articles).

Analyzing the two vector space models, we can see that the best scores per parameter were achieved by using tf-idf. Furthermore, the different metrics show a considerable difference when combined with term frequency. The best scores using term frequency are 0.071 for the euclidean metric and 0.695 for the cosine metric. With tf-idf, the difference between both metrics is still notable, but far less drastic than the one based on term frequency. This can be explained by considering that the term frequency model will count every term equally and therefore common terms will have higher weights than less common ones. Based on these weights, the distance calculated by the euclidean metric does not correspond well with the actual similarity between documents. The cosine metric is less affected by these weights, because it considers the direction of vectors instead of the distance. Tf-idf balances out the term frequency with its inverse document frequency and therefore less common terms are weighted higher than less common ones. As a result, the euclidean distance is more affected by infrequent

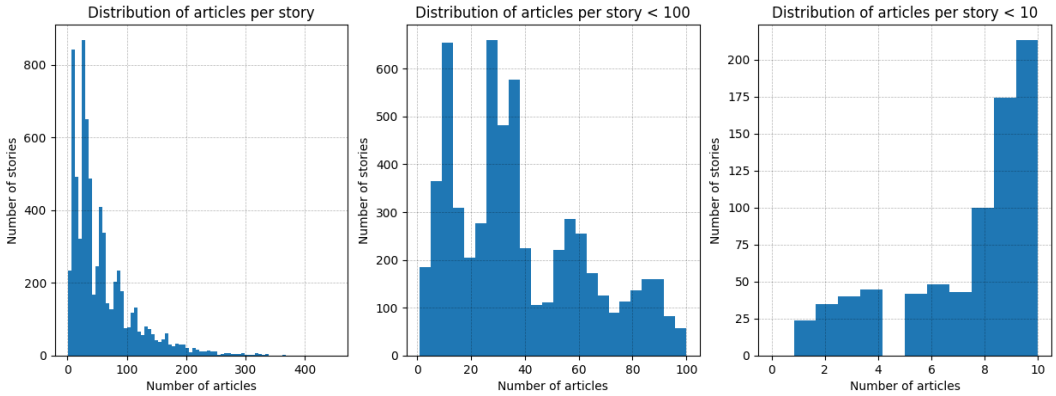

Figure 6: Distribution of cluster sizes.

terms, which are similar for similar documents and hence results in better scores. Although the cosine similarity is still superior in this case. This behaviour has already been studied in the past[28], [29] and is one of the reasons, why the cosine similarity is often preferred over the euclidean distance as a similarity measure in the field of text mining.

As for the optimal preprocessing method, Text Lemmatization provides the highest overall MP-score with 0.738, although closely followed by Text Stemming. This is to be expected, since both reduce the dimensions by grouping terms into their base form, while still retaining most of the text. In contrast to keyphrase extraction and NER, which both result in a drastic reduction of the dimensions. It is also interesting to see how close the score using the full text is compared to the best score per row. The difference between the overall best score of 0.738 achieved by Text Lemmatization and the score provided by using the full text of 0.731 is only 0.007. This means that text preprocessing has a lesser impact than initially expected. However, it is important to note, that we used pretrained models for keyphrase extraction and NER. Specifically training on a news corpus might improve the performance of both methods, but it was decided to be out of scope for this thesis.

Detailed parameter evaluation After determining the optimal settings for text preprocessing methods and vectorization, we increase the sample sizes for our evaluation runs, to get a deeper insight into the behaviour of HDBSCAN with larger data sets. Figure 7 shows the scores achieved with different parameters over an increasing set of samples. In this Figure we can see that the *cosine* metric tends to be better than the *euclidean* metric and is considerably more stable based on the range of the score. Although the quality of the clustering seems to decrease with larger sample sizes.

Furthermore, the variance with smaller sample sizes can partially be explained with differences in the number of detected clusters, because missing a few clusters has a bigger impact if the overall number of clusters is small. Figure 8 shows the difference between the number of predicted clusters and the number of true clusters. The Figure provides us with an interesting observation: While the minimum cluster size of 6 has given us so far the best scores, the difference in the number of clusters is much smaller with a minimum cluster size of 4. The MP-Score weights the similarity of a pair of clusters with their number of elements. This means ignoring smaller clusters has a lesser impact than ignoring larger clusters. We know our data set contains stories with news articles ranging from 2 up to 400. Based on this knowledge and the workings of the score, we can conclude that using a minimum cluster size of 4 gives us more clusters, but at the same time fragments larger clusters. Therefore, this minimum cluster size of 4, results in a lower score, while having a smaller difference in the number of cluster predictions. This can be validated by analysing our data directly, where we observe the number of predicted clusters to be higher the lower the minimum cluster size is. Using a minimum cluster size of 6 tends to give a lower number compared with the true number of clusters, while a minimum cluster

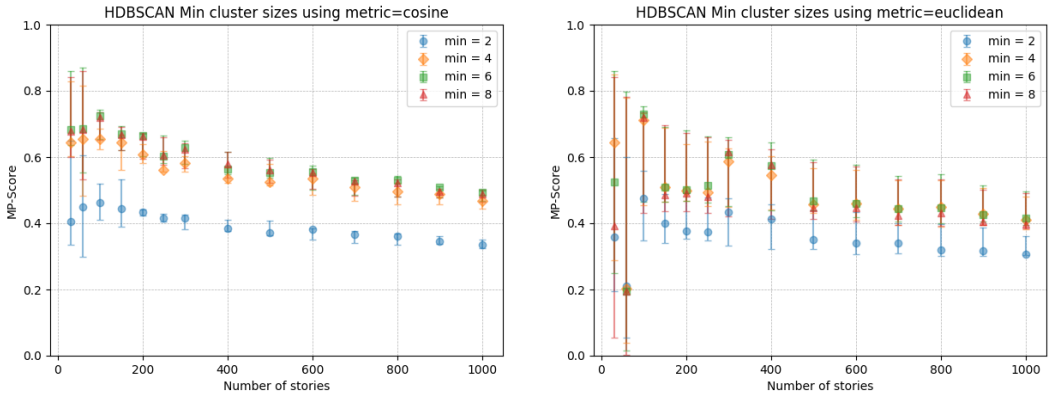

Figure 7: MP-Score for different parameters, where min stands for the minimum cluster size. The marker represents the median, while the vertical line indicates the range between the min and max values. Each run contains at least five repetitions.

size of 3 gives usually a higher number than actual clusters.

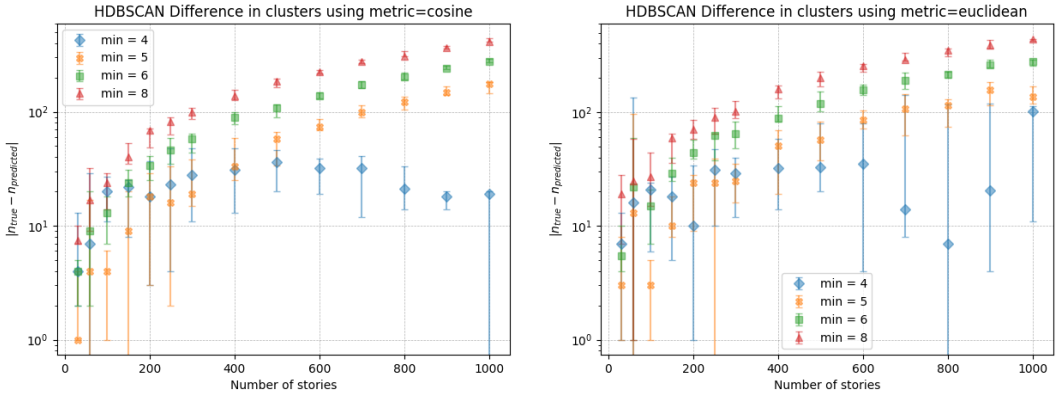

Figure 8: Difference between the predicted and true number of clusters.

Noise rate One of the advantages HDBSCAN has over other clustering algorithms, is the ability to work with noise, since we intent on applying it in an online setting, where noisy data is to be expected. At the same time, the number of articles classified as noise should be kept to a minimum. However the noise ratio shown in Figure 9 is significantly higher, than we would expect it to be based on our test data. A variety of factors play into the high noise ratio. One factor is based on the minimum cluster size. Each news article belonging to a cluster, which has less articles than the minimum cluster size, will be counted as noise. Table 10 lists the calculated percentage of news articles, which would be ignored based on different minimum cluster sizes. Although the percentages show that the impact the minimum cluster size has on the overall noise ratio is very limited. It is reasonably to assume, that the test data still contains a fair amount noisy data, even after cleaning up the data to the best of our efforts. Decreasing the noise ratio is certainly an important part in future improvements.

Let us look closer at the data to get insights behind just the score or noise ratio. The first story we focus on is about the hacking of U.S firms by chinese military hackers. Table 11 shows a number of detected and missed news articles. Based on the text length, we see that the missed articles are generally shorter compared with detected articles with one major exception. Nr. 18 with a character length of 13,980 is nearly twice as the second longest article. The content of Nr. 18 is a collection of different stories, only the first being about the chinese hackers. It seems reasonable for this article

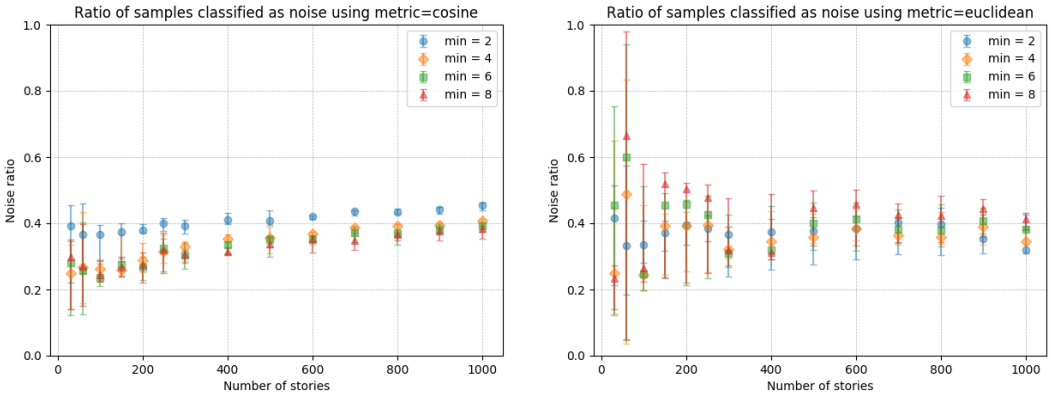

Figure 9: Number of news articles classified as noise.

Min cluster size	Ignored articles
2	0.032%
3	0.126%
4	0.304%
5	0.593%
6	0.985%
7	1.548%
8	2.168%
9	2.712%

Table 10: Percentages of ignored news articles because of their cluster size. The values are calculated directly based on the test data.

to be missed in the clustering. Articles Nr. 17, 19 and 20 all have a significantly lower length than the others, which might already be enough to classify them as noise. Looking at the actual contents reveals that Nr 17. is about the magazine's paywall, Nr. 19 appears to be a short summary of the publisher itself, and Nr. 20 contains only two sentences about the topic, a link to read more and a hint to download Acrobat Reader. Therefore these three news articles are actual noise and ideally should have been removed during the data cleansing. The remaining news articles appear to be valid articles about the story and do not provide any obvious reasons for why they were regarded as noise during the clustering.

To find a possible explanation for the remaining new articles, we take a look at their tf-idf model. Table 12 lists the top 10 keywords per news article based on the tf-idf model. There we see how the keywords from the detected news articles are quite similar, while the keywords from the missing news articles appear to be more varied with minimal overlap to the keywords from detected articles. This seems to give an indication as to why the articles were missing from the cluster. However it is important to note, that the tf-idf model in Table 12 was created from only those 20 articles and used the full text instead of Text Lemmatization for better readability. The top keywords might differ in the model created for the whole clustering. Especially words such as *ap1000*, which is a name of a nuclear power plant, will be weighted higher in the full model. Based on this insight, we expect further work on text preprocessing to lead to considerable improvements in the quality of the clustering.

Comparison with k-means So far we focused on HDBSCAN to determine the optimal settings to run it with. As a next step we start comparing the overall performance with k-means. We use the following settings: Text Lemmatization with tf-idf, cosine as the similarity measure and 6 as the minimum size of clusters. Figure 10 shows a similar behaviour for both clustering methods in value

Detected Articles			
Nr.	Title	Character length	Source
1	What were China's hacker spies after?	3801	CNNMoney
2	Chinese Cyberespionage Crackdown Prompts Look At Intellectual Property Theft	5124	CRN
3	FBI investigator: Many more US firms hit by Chinese military hackers	5585	Tribune-Review
4	State-sponsored business espionage decried	3771	Stars and Stripes
5	Westinghouse Among Companies in Chinese Trade Secret Hacking Case	1364	Nuclear Street
6	US charges on China hackers cap 3-year pressure drive	7668	Thanh Nien Daily
7	#ShotsFired in U.S.-China Cyberwar	7352	Daily Beast
8	Feds claim Chinese hackers hit US firms, including Westinghouse	3746	The Cranberry Eagle
9	America sues China over corporate spying	4278	Telegraph.co.uk
10	How China's army hacked America	3427	Ars Technica

Missed Articles			
Nr.	Title	Character length	Source
11	Other views: China hacking indictments will create waves	2968	Monterey County Herald
12	How much damage has Chinese hacking done to the US government?	785	FederalNewsRadio.com
13	FBI Releases New Details In Cyber Espionage Case	2885	CBS Local
14	How 5 Chinese hackers stole American companies' most closely-guarded secrets	3726	ITProPortal
15	U.S. Charges 5 Chinese Army Members with Economic Spying	1093	Democracy Now
16	U.S. Charges Five Chinese Military Officers with Cyber Espionage	1823	eSecurity Planet
17	Prosecutors: Chinese targeted Western Pa. companies	191	Washington Observer Reporter
18	CNN's GUT CHECK for May 19, 2014	13980	CNN
19	Nuclear Fallout From China's Alleged Espionage	122	Wall Street Journal
20	Charges Of Chinese Cybercrimes To Play Out In American Courts	443	KPBS

Table 11: 10 correctly detected and 10 missed news articles, which all belong to the same story.

and variance of the precision. Although HDBSCAN is generally more accurate than k-means, the difference gets smaller with an increase in the sample size. Additionally an increase in the number of samples results for both HDBSCAN and k-means in a decrease of the precision as can be seen in Figure 10. This has already been observed when analysing HDBSCAN parameters, and it is interesting to k-means behave in the same way.

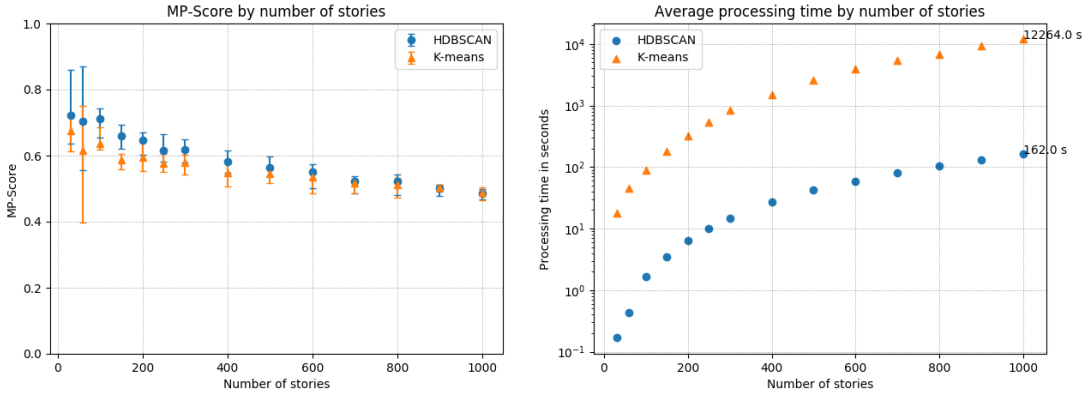

Figure 10: Comparison of the MP-Score and processing time between k-means and HDBSCAN.

While HDBSCAN and k-means provide a similar score, the biggest difference can be noted in the processing time in relation to the number of samples. k-means has a time complexity of $O(n^2)$ in contrast to HDBSCAN with a time complexity of $O(nlog(n))$, which is illustrated by Figure 10. Although running the evaluation has also shown the space complexity for HDBSCAN to be substantially higher for larger amounts of samples than with k-means. Trying to run HDBSCAN with 100,000 news articles caused in a memory error, even with 64GB of RAM, while k-means was able to complete the clustering. The memory issue with large data sets is known and according to the author the current implementation of HDBSCAN is not optimised for memory[30]. This might be another area for further improvements, although it will not help to increase the score on larger data sets.

Nr.	Top 10 Keywords
1	['chinese', 'solar', 'steel', 'power', 'firms', 'hackers', 'solarworld', 'theft', 'plants', 'ap1000']
2	['theft', 'said', 'allegedly', 'data', 'security', 'businesses', 'officers', 'intellectual', 'property', 'indictment']
3	['said', 'companies', 'pittsburgh', 'don', 'chinese', 'company', 'security', 'accused', 'computer', 'hackers']
4	['said', 'companies', 'pittsburgh', 'don', 'security', 'know', 'university', 'computer', 'makes', 'chinese']
5	['ap1000', 'state', 'alleging', 'pipe', 'construction', 'design', 'chinese', 'westinghouse', 'china', 'owned']
6	['chinese', 'people', 'snowden', 'companies', 'said', 'china', 'indictment', 'evidence', 'administration', 'officials']
7	['chinese', 'said', 'house', 'white', 'cyber', 'department', 'indictments', 'way', 'defense', 'american']
8	['chinese', 'said', 'officials', 'indictment', 'company', 'mails', 'trade', 'companies', 'pennsylvania', 'stole']
9	['america', 'chinese', 'china', 'know', 'including', 'accused', 'targeted', 'company', 'trade', 'ap1000']
10	['messages', 'access', 'mail', 'according', 'indictment', 'attack', 'mails', 'spear', 'phishing', 'union']
11	['cyberspying', 'united', 'states', 'chinese', 'national', 'security', 'high', 'economic', 'aggressive', 'justice']
12	['report', 'world', 'cyber', 'technologies', 'economic', 'alleging', 'intended', 'links', 'programs', 'says']
13	['pittsburgh', 'cyber', 'allegedly', 'fbi', 'targeted', 'details', 'enforcement', 'officials', 'happens', 'threat']
14	['messages', 'access', 'group', 'attacks', 'like', 'mail', 'spear', 'unit', 'phishing', '61398']
15	['report', 'sponsored', 'state', 'states', 'economic', 'eric', 'holder', 'case', 'united', 'intelligence']
16	['allegedly', 'proprietary', 'sun', '2010', 'information', 'stole', 'market', 'solarworld', 'business', 'owned']
17	['account', 'create', 'log', 'continue', '000', '19', '20', '2008', '2010', '2012']
18	['com', 'leading', 'don', 'obama', 'new', 'house', 'oregon', 'years', 'people', 'state']
19	['news', 'leading', 'media', 'corp', 'network', 'information', 'companies', '000', '19', '20']
20	['alleging', 'order', 'pdf', '2014', 'alleges', 'filed', 'firms', 'chinese', 'documents', 'hacked']

Table 12: Top 10 keywords extracted from the tf-idf model.

Comparison with other clustering methods As a final evaluation, we compare HDBSCAN with six different clustering methods taken from scikit-learn. Each method is run with a variety of parameters and the best scores are shown in Figure 11. HDBSCAN provides the highest precision, while being still being one of the fastest algorithms. We are aware of our bias for HDBSCAN since we invested a significant amount understanding and analysing it, but it is still interesting to see how well it compares with other clustering methods out of the box.

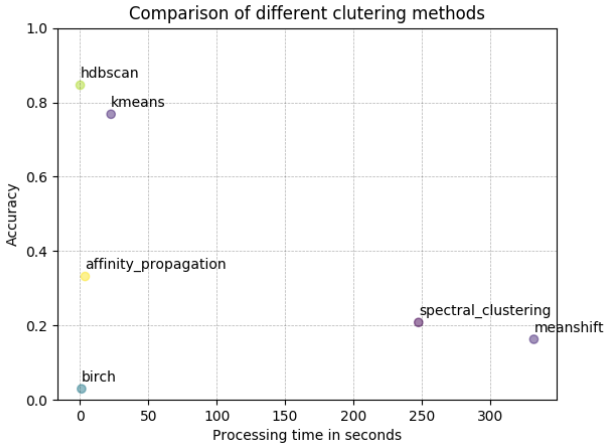

Figure 11: Comparison of different clustering methods with a sample size of approximately 1000 news articles.

4.1.3 Conclusion

The evaluation has shown HDBSCAN to be a good candidate to use for text based clustering. It provides a better precision than k-means, while being significantly faster to process. The predicted number of clusters is consistent with an increasing number of samples and fairly close to the truth. Additionally we have shown the required preprocessing and vectorization steps with the ideal parameters to achieve the most accurate results for our data set. However there is a substantial noise ratio, which causes almost a third of the processed samples to be classified as noise. We have also analysed individual clusters and discovered, that the vector space model can vary substantially between news

articles of the same cluster. Another consideration is the space complexity with larger data sets, where we quickly ran into issues when clustering high number of samples. Overall HDBSCAN provides an acceptable precision, while still leaving room for further improvements.

4.2 Online Clustering

4.2.1 Setup

The online clustering is done on a simulated stream of news articles based on the same data set as used in the clustering evaluation. This allows for direct comparison between the detected events and the ground truth. The settings to run the clustering are as follows:

- Preprocessing: Text Lemmatization
- Vector space model: tf-idf
- Clustering method: HDBSCAN
- Minimum cluster size: 6
- Metric: cosine

The clustering is run over 30 days with a total of 42,916 news articles. The distribution of news articles during this time period is illustrated in Figure 12. The time delta, which is the amount of time between two batches, is set to one hour.

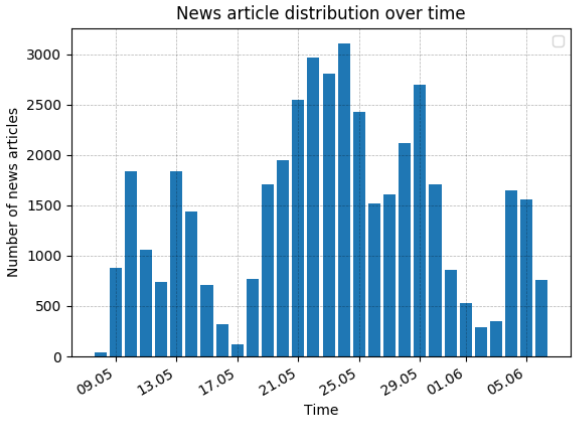

Figure 12: Incoming news articles over 30 days.

4.2.2 Evaluation

The goal of the online clustering is to detect new events in an incoming stream of news articles and changes in existing events.

Static batch sizes We start the evaluation with static batch sizes. Figure 13 shows the differences between the number of detected events and the number of true events for both new and existing topics. Based on this data we see the impact of different batch sizes for the precision in detected events. The difference with a batch size of 5,000 news articles is considerably lower than with a batch size of 1,000. The difference is especially noticeable in the time period between the 21.05 and 25.05. The reason for this spike can be found in the distribution of incoming news articles as shown in Figure 12. During this period we receive up to 3,000 news articles in a single day. This means by using a lower batch size

such as 1,000, the overlap between batches gets too small to reliably detect changes between batches, which causes too many new topics to be detected.

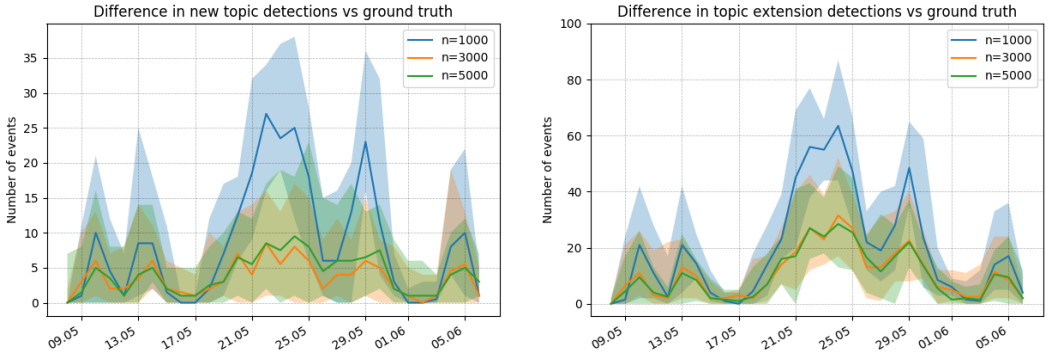

Figure 13: Comparison between the difference in detected and true events. The line represents the median, while the area shows the range from the minimum to the maximum value.

Although a larger batch size does not simply equal a better difference, as can be seen in Figure 13 by comparing the differences using a batch size of 3,000 with a batch size of 5,000. The batch size n=3000 shows a generally lower difference in the detection of new events than with n=5000. The differences between both batch sizes are smaller when detecting changes in existing events.

Based on the overall differences, we do not know the precision of those predictions. If the difference between newly detected events and true events is zero, there is still the possibility, that the events are different from the ground truth, and thus contain false positives. To measure the quality of events, we can look at the collection of events in a single batch as a subset of clusters, where each event is represented by a cluster containing all relevant news articles. Since we now have two clusterings, one containing detected events and the other with events taken from the ground truth, we can apply our MP-Score as a metric to get an insight into the precision of the detected events. Figure 14 shows the MP-Scores for an online clustering using a batch size of 1,000. Since there is quite a large variance, the score is shown as a boxplot, where a single box represents a full day. The large variance is already the first indication, that the quality is rather low. Meaning that there are still many false positives and false negatives.

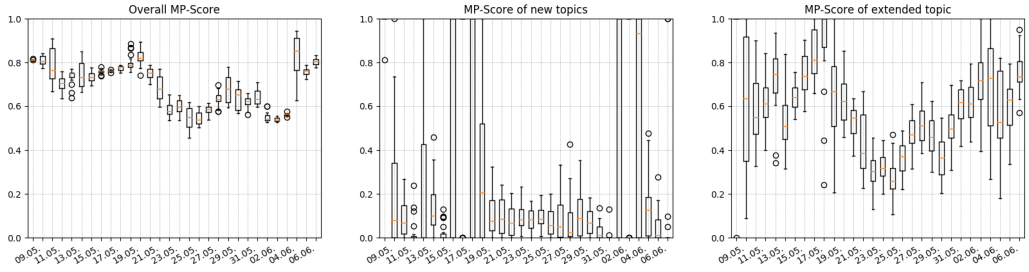

Figure 14: MP-Scores for clusterings using batch size of 1000.

Looking at an increased batch size of 5,000 in Figure 15, we note that there is less variance in the overall score, which compares the full clustering with the ground truth. Although the variance for new and existing event detections is still fairly high. Additionally while the variance is high, the median for new topics is mostly around 0.1. This tells us that most of the newly detected events do not correspond with new events according to the ground truth. The detection of extensions of existing

events is generally more accurate with a median between 0.5 and 0.8 using n=5000, but there is still are wide variance noticeable.

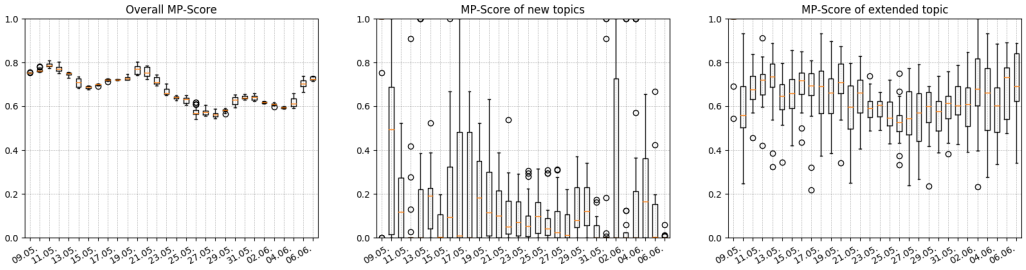

Figure 15: MP-Scores for clusterings using batch size of 5000.

One of the reasons for the difference in the precision of the detection of new events and the extension of events might be explained by the minimum cluster size. In the current setting the minimum cluster size is set to 5, which means if a new event occurs containing only four news articles, it will be discarded as noise. If the second batch contains additional news articles for the same event, it will be detected as a new occurrence, but the ground truth treats it as an existing event. This increases the difference between detections and the ground truth. To see how this affects the result we run the same simulation with a batch size of n=3000 a second time, but only considering new events from the ground truth if the number of news articles is greater or equal to the minimum cluster size.

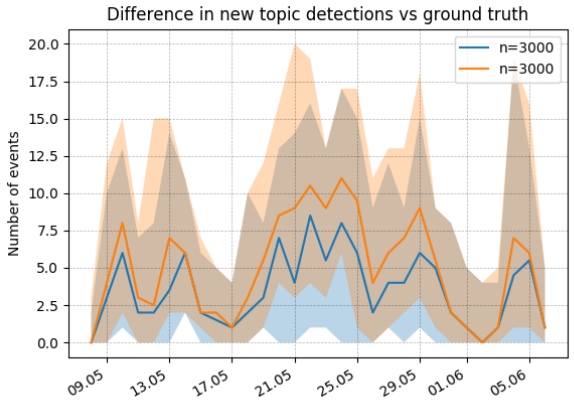

Figure 16: Differences in predictions vs ground truth using batch size of 3000.

Limiting new events in the ground truth based on the minimum cluster size gives the opposite result as initially expected. Figure 16 clearly shows an increase in the difference between predicted events and the adjusted ground truth. This means we already detected more new events than there were present in the ground truth and limiting it based on the minimum cluster size only lowered the true number of events, thus leading to an increase in the difference. A look at the raw data from an initial simulation run in Figure 17 validates this assumption.

The raw data in Figure 17 also shows a direct correlation between the number of detected new events and the number of detected changes in events. The more changes we missed, the more new events are detected. This is to be expected, since the detection of changes depends upon finding similar pairs of clusters in two different batches. If a cluster in the current batch could not be matched to a cluster from the previous batch, the cluster from the current batch will be seen as a new event. Therefore the accuracy in finding pairs of clusters in crucial to a better performance. The online clustering makes

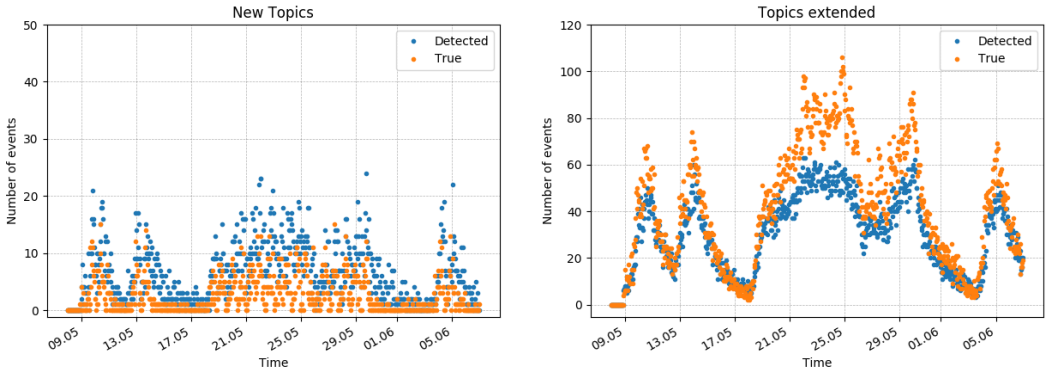

Figure 17: Number of events with a batch size of 3000.

use of LSH to find similar news articles as explained in Section 3.4.2. The current threshold value for determining the similarity is set to 0.75. To see the impact of the similarity threshold, we run the online clustering again with a batch size of n=3000 and different thresholds.

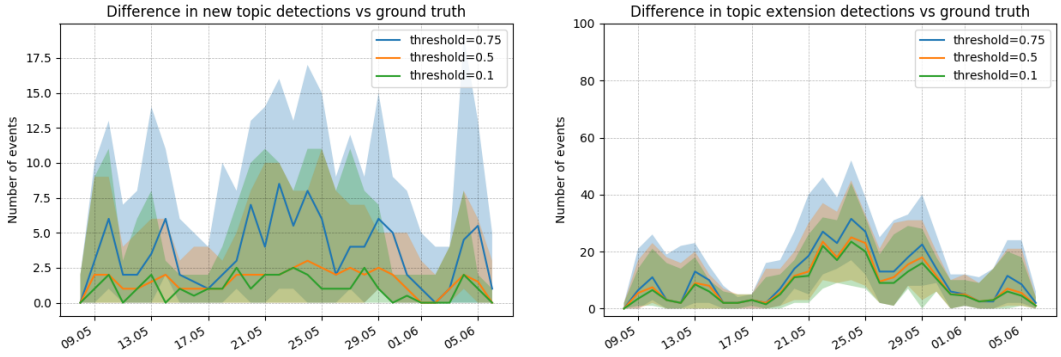

Figure 18: Differences in detected over true events with different thresholds and a batch size of 3000.

Similarity threshold Figure 18 shows the effect of the threshold on the difference between detected and true events. We see how the initial threshold of 0.75 was set too high, as lower threshold such as 0.1 provide a considerably lower difference. While there is still a substantial variance per day the median of the thresholds 0.1 and 0.5 is generally more stable and lower than with a threshold of 0.75. The reason for the better performance of lower thresholds, is that the overlap between batches decreases with an increase in the volume of news articles. This is clearly visible during the peaks in Figure 18. Thus a high similarity threshold cannot be met, since there exists only an overlap of a few news articles for the same cluster between batches. The MP-Score is also improved for new events as can be seen in Figure 19. While there is more variance than in similar plots from Figure 14 and Figure 15, the median from using threshold=0.1 clearly surpasses any measure from using threshold=0.75. The high variance in the boxplot is due to the fact, that there are only a few new events per hour, if any. This means detecting no events, when there are none leads to a score of 1, while detecting one event, when there is none, leads to a score of 0.

Dynamic batch sizes After having analysed the impact of different batch sizes in detail and the similarity threshold, we want to explore the result of using a dynamic batch size. The first dynamic method loads the number of samples over n hours. The second dynamic method calculates the batch

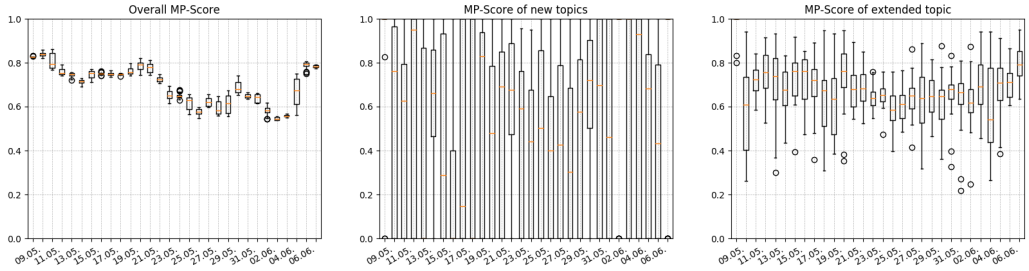

Figure 19: MP-Scores for online clustering with batch size n=3000 and threshold=0.1.

size based on the incoming samples and a predefined factor. The dynamic methods will run with a similarity of 0.1 and an upper limit of 30,000 samples, which is approximately 1,000 stories. The number of the upper limit is based on observations from the clustering method evaluation. Higher number of samples resulted in lower scores, while running into memory issues.

Figure 20 shows the difference in detected events compared with the true number of events using the first dynamic method. The variance is still quite high, especially during the peaks where a lot of news articles are present in the data stream. The best performance is achieved by setting the time window to 24 hours. The average score for the overall clustering is 0.717 with a standard deviation of 0.119. The detection of new events results in an average score of 0.618 with a standard deviation of 0.426 and the detection of changes results in an average score of 0.694 with a standard deviation of 0.16. The resulting scores are better than with using a fixed batch size of 3,000, but not by much. The biggest difference is in the average score of the detection of new events, where this method is better by 0.076. Increasing the time window to 48 or 72 hours results in slightly lower scores.

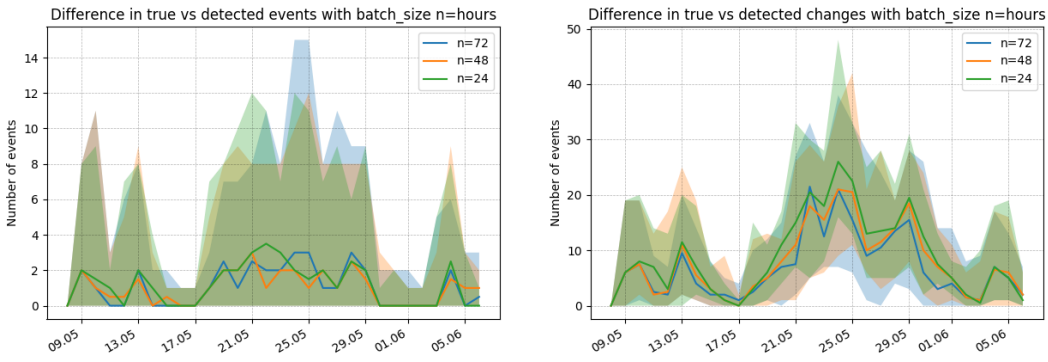

Figure 20: Difference in detected events vs predict events by using a dynamic batch size based on hours.

Since the first dynamic method only showed a marginal improvement over the static method, we move on to the second dynamic method. In addition to an upper bound, this method uses a lower bound to prevent the overlap between batches from becoming too small. The lower bound is set to 3,000, since the evaluation of a fixed batch sizes proved 3,000 to provide the best results. Our initial assumption was, that having a good lower bound would give a good baseline and the relative factor allows the batch size to grow with the increase in the volume of samples. The overall performance should therefore be superior to using only the fixed batch size. The data proved our assumption to be incorrect as can be seen in Figure 20. The best scores are achieved by using a factor of 25. The overall score is 0.692 with a standard deviation of 0.091. The detection of new events results in a score of 0.589 with a standard deviation of 0.429. The score of detecting changes is 0.682 with a standard deviation of 0.136.

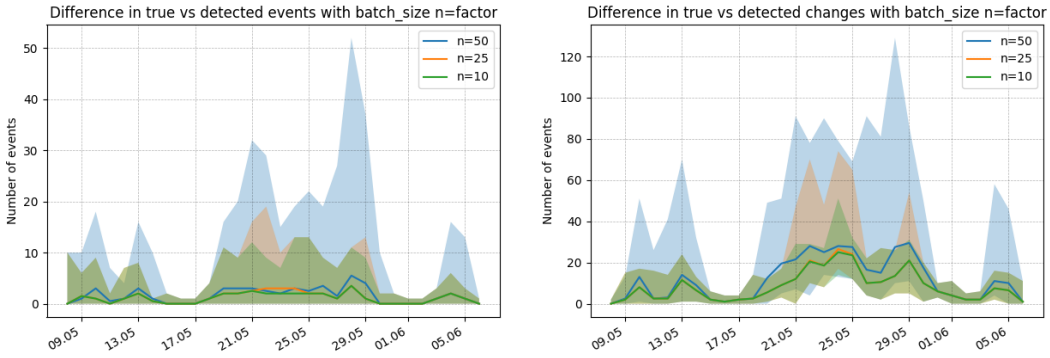

Figure 21: Difference in detected events vs predict events by using a dynamic batch size based on incoming samples.

In addition, the data shows an interesting observation, where using a factor of 50 results in some extreme outliers. The reason can be explained based on our previous clustering evaluation, where we showed the decrease in precision with larger number of samples. Figure 22 shows the maximum number of samples processed in a single batch. The factor 50 reaches the upper limit during most peaks in the data stream. Clustering with 30,000 resulted in an lower average score of roughly 0.24 compared to the best result. Therefore an increasing amount of news articles is classified as noise and stories are further fragmented into multiple clusters. This leads to even bigger decreases in the event detection as this data shows.

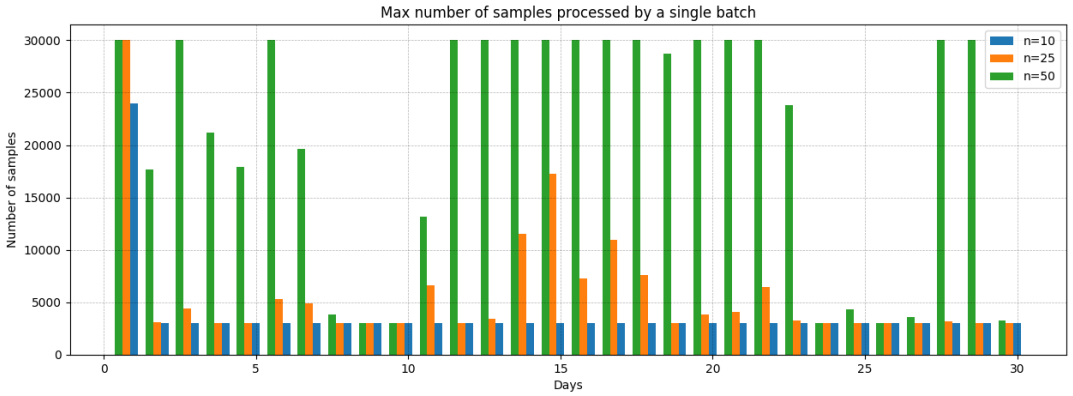

Figure 22: Maximum number of processed samples in a single batch per day and factor.

To conclude the evaluation of different methods for setting the batch size Table 13 shows the best approach per method. The highest scores are highlighted as bold. Based on this data, the time based method seems to provide the best results, although the differences between the methods are minor. The high standard deviation for the detection of new events shows that the detection is quite unstable in its current form. Improving the precision and the stability for the event detection is therefore dependent on improving the overall clustering precision, since the dynamic methods for determining the batch size only show limited impact.

Noise rate One of the major factors in the lower precision of event detection compared with the overall clustering is the noise rate. To demonstrate this let us look at an online clustering over a simulated time period of five days. Overall the time period includes 364 stories with 12,603 news articles. During the online clustering we detected 275 new events and 4,080 changes. On average these

Batch size method	Overall clustering		New events		Changes in existing events	
	Mean	Standard deviation	Mean	Standard deviation	Mean	Standard deviation
Fixed with n=3000	0.701	**0.085**	0.542	0.435	0.683	0.144
Time based with hours=24	**0.717**	0.119	**0.618**	**0.426**	**0.694**	0.16
Relative with factor=25	0.692	0.091	0.589	0.429	0.682	**0.136**

Table 13: Final scores obtained by each method for setting the batch size.

events contain 2 news articles. Of the original 12,603 news articles only 8,626 have been assigned to clusters, with an average cluster size of 26.4. This results in a noise rate of 31.6%. If we now compare the average cluster size of 26.4 with the average event size of 2, we can see how single events are more likely to be missed than clusters due to their size.

During the analysis we also looked at two specific examples "Gmail redesign" and "Bad Neighbours". The first example was detected without errors, while the second example contained multiple news articles discarded as noise and a fragmentation in clusters. The fragmentation means that news articles from the same story got assigned to different clusters during the same batch, resulting in incorrect events.

The difference in the performance of both examples, can be explained based on the contents of their news articles. The first example about the "Gmail redesign" consists mostly of technology focused news sites such as *Ars Technica* or *PC Magazine*. Therefore the contents of the news articles are of good quality and share the same technical vocabulary. The second example with the new release of the film "Bad Neighbours" has a wider variety of sources and types of articles. Some news articles are short summaries, while others are interviews or personal reviews. As a result the vocabulary is more general than compared to the technical articles and varies stronger between articles. This leads to increased differences in the tf-idf model, which we have already explored as part of the clustering evaluation.

4.2.3 Conclusion

We explored different methods for determining the batch size, with one static and two dynamic approaches. The best results were achieved by using the dynamic method, which loads samples from the last 24 hours instead of defining an actual batch size. The resulting precision of the clustering using this method is 72% with a standard deviation of 12%. The precision for detecting new events results in 62% with a standard deviation of 43%. Detecting changes in existing events results in a precision 69% with a standard deviation of 16%. Although the differences in the resulting scores between the methods are marginal and all methods show similar deviations in the event detection. One of the reasons the dynamic methods did not perform better, is based on the overall decrease in precision of HDBSCAN with higher number of samples. Therefore increasing the batch size to react to high volumes of traffic resulted in a score similar or worse than the static batch size. In addition, we compared different thresholds for defining the similarity between samples and found 0.1 to be a good candidate. The noise rate is shown to impact the events even more then the clusters and therefore is a major influence regarding the precision of the event detection. In conclusion the high variance in correctly detecting event shows, that it is quite unstable in its current form and further optimizations should be focused on the overall clustering and on lowering the noise rate.

5 Conclusion

5.1 Summary

We started our work by searching for a suitable data set to create our clustering evaluations with. The primary requirement was to have data points with corresponding cluster labels. Having a labelled data set allows us to apply external measures and evaluate a resulting clustering against the ground truth. After selecting a few data set for closer inspection, we settled on the News Aggregator Data Set, which contains 422,937 labelled news articles, where a label describes the story the news article is about. Since the same story label applies to multiple news articles, we could use this as a cluster descriptor. Unfortunately the data set only contained headlines, which did not contain enough information for our approach. Therefore we collected the full text from each news article based on the provided source url. The content retrieval process turned out to generate a significant amount of noise, due to expired urls, paywalls, parsing errors or wrong redirects. To reduce the noise, we applied different cleansing techniques and ended up with 235,070 usable news articles.

Once the data set was ready, we designed an evaluation framework to automatically run clustering methods with a variety of settings. The focus was to find a combination of text preprocessing methods, vector space model and parameters for the clustering method, which would provide the best clustering. Furthermore we developed a custom scoring function to measure the results of a clustering, since existing measures proved to be unintuitive and biased against certain results, such as the number of clusters. The analysis gave valuable insight into the behaviour of HDBSCAN with different vector space models combined with different preprocessing methods and parameters. We noted the initial good performance and the decrease in the quality of the clustering the larger sample sets. However the amount of news articles proved to be substantial with up to 30%. Possible explanations were explored, such as actual noisy data and different representations of articles belonging to the same cluster with tf-idf. Furthermore we found HDBSCAN to be both faster and more precise than k-means.

Having determined the optimal settings in the HDBSCAN evaluation, we applied them for the event detection using a simulated stream of news articles. The event detection was accomplished by running the clustering in batches over time. We explored three methods for setting the batch size and different similarity thresholds for finding pairs of clusters between batches. Since finding pairs of clusters, requires a large enough overlap in identical news articles, the batch size has to account for this factor with regards to the volume of incoming news articles through the data stream. The best method for determining the batch size turned out to be based on a fixed time period, where we load samples from the past 24 hours for every batch. Thus peaks in volume during this time period will be included and not cut off by a static batch size. Additionally since events are represented as clusters, the sum of events can be regarded as a subclustering of the overall clustering. Although this makes the subclustering more sensitive to the noise rate, due to the smaller size of events. In conclusion we found the precision of the event detection to have a high variance for new events, rendering it rather unstable in its current form. A continuation of this work should focus on improving the overall clustering to increase the precision of the event detection.

5.2 Future Work

The approach in its current state still leaves different areas up for improvement. Further work on NER, might help in drastically reducing the dimensionality of the vector space model and condense a news article into only a few key entities. Using a pretrained model did not result in accurate results, but training a model specifically on a new corpus might improve the NER significantly. Another preprocessing technique, which we did not look at, would be word embeddings. Word embeddings allow for the detection of similar words and therefore reduce the dimensionality of the vector space model substantially more than even Text Lemmatization. Thus leading to a potential improved clustering

and reducing the noise rate.

During our evaluation we did not explore using a dynamic time interval for running the batchwise online clustering. One example could be instead of using a fixed interval of one hour, a clustering would start as soon enough news articles have been collected. Although we do not assume any considerably improvements in the precision of the event detection, since it would still be affected by the high noise rate. Nonetheless once the issue with the noise rate has been solved, this might be an interesting alternative to the dynamic batch sizes.

As we have shown, the current implementation of HDBSCAN still leaves room for improvement in regards to space complexity. Finding potential optimizations in memory consumption would not necessarily improve our approach, since the quality of clusters decreases with larger sample sets, but might be a valuable contribution to the community and enable future work with larger data sets.

We focused mainly on HDBSCAN in our analysis, but the evaluation framework allows for many different clustering methods. Finding different methods suitable for text clustering or even a combination of different algorithms might lead to better results. Although we did try out some different variations such as HDBSCAN with LDA, but without any notable results.

Furthermore it would be interesting to see how HDBSCAN would perform using a data set based on a different kind of textual data. A possible alternative data set could be based on computer logs, which would also provide a source for data streams. Improvements in the overall performance of HDBSCAN will also significantly improve the event detection in data streams.

5.3 Lessons Learned

We learned that HDBSCAN is quite a versatile and efficient clustering algorithm and at the same time experienced the challenges in text mining. The wide variety in types of articles and the noisy data caused the noise rate to be quite high. We underestimated at the beginning the effect this would have on the quality of the clustering and moreover on the precision of the event detection.

Another lesson we learned was the importance of understanding and exploring the scoring function in more detail. We did not research the scoring function we used in the beginning enough to be aware of their biases. As a result we had to restart our evaluations, once we found these anomalies in our results and ultimately developed our own function.

Further we learned to question our third-party libraries even well established ones such as scikit-learn. While researching the theory for tf-idf, we could not reproduce the results produced by scikit-learns tf-idf. It turned out scikit-learn uses a slightly different formula from the official theory. Additionally the implementation of HDBSCAN is not optimized for memory consumption, which we found out by looking through the issues on github.

6 Related Work

Text based event detection is a diverse field with an increasing amount of available information online. With the popularity of social media, a lot of research around event detection has been done on micro blogs[31], such as Twitter[32], [33].

Text Preprocessing Text Stemming and Text Lemmatization have been around for decades and both are proven to not only work in theory but also in practice. So it comes as no surprise that many people try to improve them with new methods such as Ultra-stemming[34] or corpus-based stemming algorithms[35].

There are also different papers comparing the preprocessing methods with each other[36], [37].

With that many resources available, it is important to be aware that the different methods score differently for each language and each data set. Having enough time for evaluating potential candidates is crucial.

Clustering This thesis focuses on news articles as the primary data source. Text based clustering as a technique for event detection has already been explored with different approaches such as using custom methods based on neural networks[38] or by using a modified version of DBSCAN to account for its sensitivity for differences in cluster densities[39].

Based on the promising results with DBSCAN, we want to further explore text clustering using its successor HDBSCAN[12] and apply it in an online setting. Regarding the clustering validation, there has already been research into recognizing biases of different scoring functions[40] and developing custom scoring functions as a result[41].

7 Index

7.1 Bibliography

[1] J. Lovins, "Development of a stemming algorithm", *Mechanical Translation and Computational Linguistics*, vol. 11, no. 1–2, pp. 22–31, Jun. 1968. [Online]. Available: http://www.mt-archive.info/MT-1968-Lovins.pdf.

[2] C. van Rijsbergen, S. Robertson, and M. Porter, "New models in probabilistic information retrieval", 1980. [Online]. Available: https://tartarus.org/martin/PorterStemmer/def.txt.

[3] M. Porter, *The english (porter2) stemming algorithm*, Sep. 2002. [Online]. Available: http://snowball.tartarus.org/algorithms/english/stemmer.html.

[4] C. D. Paice, "Another stemmer", *SIGIR Forum*, vol. 24, no. 3, pp. 56–61, 1990. DOI: 10.1145/101306.101310. [Online]. Available: https://doi.org/10.1145/101306.101310.

[5] *Wordnet stemmer*, https://web.archive.org/web/20190516161521/https://www.nltk.org/_modules/nltk/stem/wordnet.html, Accessed: 2019-05-16.

[6] *Esa tweet*, https://twitter.com/esa/status/1067763858310422529, Accessed: 2019-05-30.

[7] T. S. Soheil Danesh and J. H. Martin, "Sgrank: Combining statistical and graphical methods to improve the state of the art in unsupervised keyphrase extraction", pp. 117–126, 2015. [Online]. Available: https://www.aclweb.org/anthology/S15-1013.

[8] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition", pp. 260–270, 2016. [Online]. Available: https://www.aclweb.org/anthology/N16-1030.

[9] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing", *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.446.5101&rep=rep1&type=pdf.

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

[11] *Scikit-learn: Tf–idf term weighting*, https://scikit-learn.org/stable/modules/feature_extraction.html\#tfidf-term-weighting, Accessed: 2019-06-03.

[12] L. McInnes, J. Healy, and S. Astels, "Hdbscan: Hierarchical density based clustering", *The Journal of Open Source Software*, vol. 2, no. 11, Mar. 2017. DOI: 10.21105/joss.00205. [Online]. Available: https://doi.org/10.21105%2Fjoss.00205.

[13] *How hdbscan works*, https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html, Accessed: 2019-05-25.

[14] *The gdelt project*, https://www.gdeltproject.org/, Accessed: 2019-06-05.

[15] *Challengenetwork dataset*, http://odds.cs.stonybrook.edu/challengenetwork-dataset/, Accessed: 2019-06-05.

[16] *One million posts corpus*, https://ofai.github.io/million-post-corpus/, Accessed: 2019-06-05.

[17] *Online retail data set*, http://archive.ics.uci.edu/ml/datasets/Online+Retail, Accessed: 2019-06-05.

[18] *News aggregator data set*, https://archive.ics.uci.edu/ml/datasets/News+Aggregator, Accessed: 2019-06-05.

[19] *Dodgers loop sensor data set*, http://archive.ics.uci.edu/ml/datasets/Dodgers+Loop+Sensor, Accessed: 2019-06-05.

[20] *Ten thousand german news articles dataset*, https://tblock.github.io/10kGNAD/, Accessed: 2019-06-05.

[21] *Newspaper3k: Article scraping & curation*, https://web.archive.org/web/20190312144257/https://newspaper.readthedocs.io/en/latest/, Accessed: 2019-03-12.

[22] *Swiss surrealist designer for alien film dies*, https://www.thelocal.ch/20140513/swiss-surrealist-designer-for-alien-film-dies, Accessed: 2019-06-05.

[23] Y. Lei, J. C. Bezdek, S. Romano, N. X. Vinh, J. Chan, and J. Bailey, "Ground truth bias in external cluster validity indices", *Pattern Recognition*, vol. 65, pp. 58–70, 2017, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2016.12.003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320316303910.

[24] A. Amelio and C. Pizzuti, "Correction for closeness: Adjusting normalized mutual information measure for clustering comparison", *Computational Intelligence*, vol. 33, no. 3, pp. 579–601, 2017.

[25] W. M. J. Mohammed J. Zaki, "Data mining and analysis, Fundamental concepts and algorithms", in. Cambridge University Press, 2014, ch. Chapter 17: Clustering Validation.

[26] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt, *Practical and optimal lsh for angular distance*, 2015. arXiv: 1509.02897 [cs.DS].

[27] E. Zhu and V. Markovtsev, *Ekzhu/datasketch: First stable release*, Feb. 2017. DOI: 10.5281/zenodo.290602. [Online]. Available: https://doi.org/10.5281/zenodo.290602.

[28] A. Strehl, E. Strehl, J. Ghosh, and R. Mooney, "Impact of similarity measures on web-page clustering", in *In Workshop on Artificial Intelligence for Web Search (AAAI 2000*, AAAI, 2000, pp. 58–64.

[29] A. Huang, "Similarity measures for text document clustering", *Proceedings of the 6th New Zealand Computer Science Research Student Conference*, Jan. 2008.

[30] *Optimizing hdbscan for huge datasets*, https://github.com/scikit-learn-contrib/hdbscan/issues/212, Accessed: 2019-06-01.

[31] O. Ozdikis, P. KARAGOZ, and H. Oğuztüzün, "Incremental clustering with vector expansion for online event detection in microblogs", *Social Network Analysis and Mining*, vol. 7, Dec. 2017. DOI: 10.1007/s13278-017-0476-8.

[32] F. Atefeh and W. Khreich, "A survey of techniques for event detection in twitter", *Computational Intelligence*, vol. 31, no. 1, pp. 132–164, 2015. DOI: 10.1111/coin.12017. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/coin.12017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/coin.12017.

[33] A. Nurwidyantoro and E. Winarko, "Event detection in social media: A survey", pp. 1–5, Jun. 2013. DOI: 10.1109/ICTSS.2013.6588106.

[34] J. Torres-Moreno, "Beyond stemming and lemmatization: Ultra-stemming to improve automatic text summarization", *CoRR*, vol. abs/1209.3126, 2012. eprint: 1209.3126.

[35] J. H. Paik, D. Pal, and S. K. Parui, "A novel corpus-based stemming algorithm using co-occurrence statistics", W. Ma, J. Nie, R. A. Baeza-Yates, T. Chua, and W. B. Croft, Eds., pp. 863–872, 2011.

[36] D. U. Suryanarayana, S. M. Hussain, P. Kanakam, and S. Gupta, "Stepping towards a semantic web search engine for accurate outcomes in favor of user queries: Using rdf and ontology technologies", *2015 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pp. 1–6, 2015.

[37] M. Bounabi, K. E. Moutaouakil, and K. Satori, "A comparison of text classification methods method of weighted terms selected by different stemming techniques", M. Lazaar, Y. Tabii, M. Chrayah, and M. A. Achhab, Eds., 43:1–43:9, 2017.

[38] S. and; Sycara, "Text clustering for topic detection", Jan. 2004.

[39] I. Gialampoukidis, S. Vrochidis, and I. Kompatsiaris, "A hybrid framework for news clustering based on the dbscan-martingale and lda", in. Jan. 2016, vol. 9729, pp. 170–184, ISBN: 978-3-319-41919-0. DOI: 10.1007/978-3-319-41920-6_13.

[40] J. Wu, H. Xiong, and J. Chen, "Adapting the right measures for k-means clustering", in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09, Paris, France: ACM, 2009, pp. 877–886, ISBN: 978-1-60558-495-9. DOI: 10.1145/1557019.1557115. [Online]. Available: http://doi.acm.org/10.1145/1557019.1557115.

[41] A. J. Gates, I. B. Wood, W. P. Hetrick, and Y.-Y. Ahn, "On comparing clusterings: An element-centric framework unifies overlaps and hierarchy", *arXiv preprint arXiv:1706.06136*, 2017.

7.2 Glossary

Docker A tool to package the application with all its dependencies as a single deployable unit and run it on independently from the underlying host. 23

Dockerized An application environment running as a single or a collection of docker containers. 23

Stop word A term that is overall so frequently used, that it is ignored in natural language processing. 9

Vectorizer A vectorizer transform a text into a numeric vector. 9

7.3 List of Abbreviations

ARI Adjusted Rand Index. 20, 22, 23

CNN Convolutional neural network. 8

GDPR General Data Protection Regulation. 16

LSH Locality Sensitive Hashing. 26, 38

NER Named Entity Recognition. 6, 8, 28–30, 42

NLP Natural Language Processing. 6

NMI Normalized Mutual Information. 20, 22, 48

VSM Vector space model. 8–10, 23, 28, 29, 48

7.4 List of Figures

7.5 List of Tables

8 Appendix

8.1 Algorithm for the MP-Score

```
import collections


def calculate_mp_score(true_clusters, predicted_clusters):
    """
    Calculate the mp_score of a clustering based on the contents of the clusters and the overall difference in
    predicted over true number of clusters. The calculation is based on three steps:
        1. Create an similarity matrix by calculating the difference between each cluster of both clusterings.
        2. Select the most relevant values from the similarity matrix and make sure no two clusters are being used
           at the same time.
        3. Calculate the weighted average, where the weight is based on the true and predicted amount of elements
           in a cluster.

    Parameters
    ----------
    true_clusters: array[clusters]
        2-dimensional array of true clusters

    predicted_clusters: array[clusters]
        2-dimensional array of predicted clusters
    """

    # If both clusters are empty, they are identical.
    if len(true_clusters) == 0 and len(predicted_clusters) == 0:
        return 1

    similarity_matrix = create_similarity_matrix(true_clusters, predicted_clusters)
    unique_indices = select_max_values(similarity_matrix)
    return calculate_weighted_average(unique_indices, true_clusters, predicted_clusters)


def create_similarity_matrix(true_clusters, predicted_clusters):
    similarity_matrix = []
    for true_cluster in true_clusters:
        true_set = set(true_cluster)
        n_true = float(len(true_set))
        row = []
        for predicted_cluster in predicted_clusters:
            cluster_set = set(predicted_cluster)

            # Calculate the similarity using the jaccard index
            similarity = len(true_set.intersection(cluster_set)) / len(
                true_set.union(cluster_set)
            )
            row.append(similarity)

```

```
        similarity_matrix.append(row)
    return similarity_matrix


def select_max_values(precision_matrix):
    unique_indices = dict()
    row_index = 0
    nrows = len(precision_matrix)

    while row_index < nrows:
        ignore_indices = set()
        max_value_found = False

        while not max_value_found:
            max_value = 0
            column = 0
            for col_index, value in enumerate(precision_matrix[row_index]):
                if value >= max_value and col_index not in ignore_indices:
                    max_value = value
                    column = col_index

            if (
                max_value > 0
                and column in unique_indices
                and unique_indices[column]["row_index"] != row_index
                and unique_indices[column]["max_value"] > 0
            ):
                if unique_indices[column]["max_value"] < max_value:
                    # The column is already used, but we found a better
                    # candidate. We use the new candidate and set the
                    # cursor to the old one to find a new max value.
                    old_row_index = unique_indices[column]["row_index"]
                    unique_indices[column]["row_index"] = row_index
                    row_index = old_row_index
                    unique_indices[column]["max_value"] = max_value
                    max_value_found = True
                else:
                    # The column is already used by a better candidate.
                    ignore_indices.add(column)
            else:
                # If max_value is greater than 0, we store the value as a
                # new candidate. Otherwise either the row does not match
                # any other column or the max_value was low and got
                # overridden by previous tries and no other match is
    available.
                if max_value > 0:
                    # The column is free to use
                    unique_indices[column] = {
                        "row_index": row_index,
                        "max_value": max_value,
                    }
                max_value_found = True
                row_index += 1

    return unique_indices


```

```
def calculate_weighted_average(unique_indices, true_clusters, predicted_clusters):
    mp_score = 0

    elements_per_true_cluster = [len(cluster) for cluster in true_clusters]
    elements_per_predicted_cluster = [len(cluster) for cluster in predicted_clusters]

    total_true_elements = sum(elements_per_true_cluster)
    total_pred_elements = sum(elements_per_predicted_cluster)
    total_elements = total_true_elements + total_pred_elements

    if total_elements > 0:
        for column, value in unique_indices.items():
            # The row of the similarity matrix equals the index of the true cluster, while the column is the index of the predicted cluster
            weight = (
                elements_per_true_cluster[value["row_index"]]
                + elements_per_predicted_cluster[column]
            ) / (total_elements)
            mp_score += value["max_value"] * weight

    return mp_score
```

Listing 6: Calculate the MP-Score between two clusterings.

8.2 Code

The complete source code for this thesis can be accessed at https://github.com/dpacassi/ba2019. The repository contains the applications for the evaluation framework and the online clustering. Furthermore it contains the source of this document and the code to generate the plots and data tables from.

TODO make sure access is available (move to zhaw github?)