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.

Abstract

While searching for events in a data stream, the definition of an event is not always clear. Providing static definitions, as most approaches do, does not function well for dynamic data streams, which may change the definition of an event over time.

This thesis focuses on developing and evaluating a methodology based on online clustering, where events can be considered either as changes in clusters over time or as the creation of new clusters. The methodology will be applied in the domain of text mining, with data streams consisting of incoming news articles. This allows news articles to be clustered based on their similarity, as similar news articles are considered to be 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 is in determining a suitable data set, which will be the subject of the clustering and at the same time provide the ground truth for evaluating the results. The evaluation focuses on HDBSCAN as the clustering method and compares it with k-means, where HDBSCAN is both faster and more accurate. Moreover, different text preprocessing methods and vector space models are evaluated, with text lemmatisation and tf-idf providing the most promising results. Once applied into an simulated online setting, inaccuracies in the overall clustering have a larger impact on event detection. This results in a significant error rate in the detection of new events, while the detection in changes of existing events shows better results. Overall the error rate in event detection is considered to be too high for real world applications and a possible continuation of this work could be improving the clustering to decrease the error rate in detecting events.

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.

Daniel Milenkovic and David Pacassi Torrico

Contents

1 Introduction

1.1 Problem formulation

How can events be recognized in a data stream? While searching for events in a data stream, the definition of an event is not always clear. 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. Thus detecting events based on an incoming stream of news articles is a challenging task.

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

1.2 Motivation

Today's environment is rapidly changing. With more devices being digitalized and connected to the internet, we are starting to have incredible amounts of data. Every smartphone, smartwatch and many other Internet of things (IoT) devices start tracking every sensor data they record.

There is and will be no way to process all this data manually. This is were our work becomes relevant. We will try to detect events from a data stream, even when new and unknown events arise.

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

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

2 Related Work

Text based event detection is a diverse field and with increasingly large amounts of available information online a compelling topic for research. With the popularity of social media a lot of research around event detection has been done on micro blogs[1] such as Twitter[2][3]. However 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[4] or by using a modified version of DBSCAN to account for its sensitivity for differences in cluster densities[5]. Based on the promising results with DBSCAN we want to further explore text clustering using its successor HDBSCAN[6] and apply it in an online setting. Regarding the clustering validation there has already been research into recognising biases of different scoring functions [7] and developing custom scoring functions as a result[8].

3 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). In this chapter we will explain how the most important techniques used in our thesis work.

3.1 Text preprocessing

When working with text data, many algorithms and processes will need to distinguish words from another. In most cases this is done by creating a dictionary of all known words and/or by *vectorising* text data in order to make them comparable. While this works in theory, in reality we deal with a tremendous amount of data. This results in huge directories or many vectors and does not only take up more disk space but also text processing (computation and comparison) will take more time to process.

What does that mean? Consider the following words:

- switzerland
- Switzerland
- SWITZERLAND

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

Lowercasing text can just be the beginning though. Depending on the size of a document, it might make sense to not use all text inside a document. A good example for this would be books. Vectorising books would take too much computational power and time to process. In such cases, the text size would have to be reduced. This could 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. Later could be done with *Keyphrase extraction* or with *Entity extraction*.

If we are not dealing with books but with articles or papers, there are also other alternatives to Keyphrase and Entity extraction. Using *Text Stemming* or *Text Lemmatisation* we can simplify terms and group them together to reduce the dictionary size. More about this in the sub sections below.

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 uppercase might be angrier than someone who does not.

3.1.1 Keyphrase extraction

When describing data, a popular thing to do is tagging. The most well-known tagging procedures are probably hash tags and user names in social media. Let us check following Tweet from the *European Space Agency*[9]:

> 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. Luckily there are different approaches on how to retrieve Keyphrases from text data.

We are using SGRank[10] 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 equal accurate than with the whole text.

Since the dictionary size would be extremely smaller, the data processing time could be drastically improved.

3.1.2 Named Entity Recognition

Similar to *Keyphrase extraction*, Named Entity Recognition (NER) extracts relevant terms out 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)[11], we retrieve following entities:

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

For comparison, the same sentence would extract following *Keyphrases*:

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

Comparing the 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. It extracts more accurate information compared to Keyphrase extraction but does this also result in improved clustering results?

3.1.3 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*, *connection* share a similar meaning and could therefor be simplified to the base term **connect**.

The first paper describing a stemming algorithm was written by Julie Beth Lovins[12] 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 furtherly.

Lovins' stemming algorithm was very successful but got mostly replaced by M.F. Porters stemming algorithm[13] 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/or removal rules and is therefor very easy and efficient.

M.F. Porter improved his stemming algorithm even further by publishing the Porter2 stemming algorithm[14] in 2002 which is widely known as the *Snowball stemming algorithm*.

There are even more stemming algoriths, very well known are the Lancester stemming algorithm[15] and the WordNet stemming algorithm[16]. Since M.F. Porter's snowball stemming algorithm is the most widely used one, we decided to go with his algorithm.

3.1.4 Text Lemmatisation

Similar to *Text Stemming*, Text Lemmatisation has the same goal to group together the inflected forms of a word but follows a different approach to do so. Instead of processing terms with fixed steps and defined rules, Text Lemmatisation 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 asks for more computational power than Text Stemming. See following table for a comparison:

#	**Original word**	**Stemmed**	**Lemmatised**
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 Lemmatisation.

3.2 Vector Space Model

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

When working with a VSM, text documents get vectorised. This is succeeded by assigning each unique term over all documents to one dimension.

When we now want to compare two documents with each other, we simply compare their vectors with each other.

But how do we vectorise each term of each document to a numerical value? Let us have a look at the two most used text vectorisers.

3.2.1 Term Frequency

The easiest vectoriser is the Term Frequency Vectoriser. 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 documents. As the Term Frequency Vectoriser simply sums the occurences of each term up, the VSM looks as in Table 2.

bank	england	landing	major	narrows	probe	rosetta	scopes	search	shake	site	space	zone
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
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
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.

3.2.2 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 Vectoriser, 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 [17] 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[18] library, the tf-idf implementation[19] of the library is slightly different, see 2.

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

bank	england	landing	major	narrows	probe	rosetta	scopes	search	shake	site	space	zone
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
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
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.

3.3 Clustering

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

3.3.1 *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)

3.3.2 DBSCAN

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

A big advantage of DBSCAN is that it is able to sort data into clusters of different shapes.

How does DBSCAN work 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. minPoints - 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 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.

3.3.3 HDBSCAN

HDBSCAN is a hierarchical density-based clustering algorithm [6], based on DBSCAN and improves its sensitivity for clusters of varying densities. Therefore defining an epsilon parameter, which acts as a threshold for finding clusters, is no longer necessary. This makes the algorithm more stable and flexible for different applications.

How HDBSCAN works Since HDBSCAN is the focus for this thesis, we want to give a more detailed explanation of its inner workings, than for k-means or DBSCAN.

HDBSCAN only requires one parameter to be set beforehand:

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

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

1. Transforming the space At its core HDBSCAN is a single linkage clustering, which are typically rather sensitive to noise. A single noise point between clusters could act a bridge, which would result in both clusters to be seen as one. To reduce this issue, the first step is to increase the distances of lower density points. 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. TODO describe example in Figure 1.

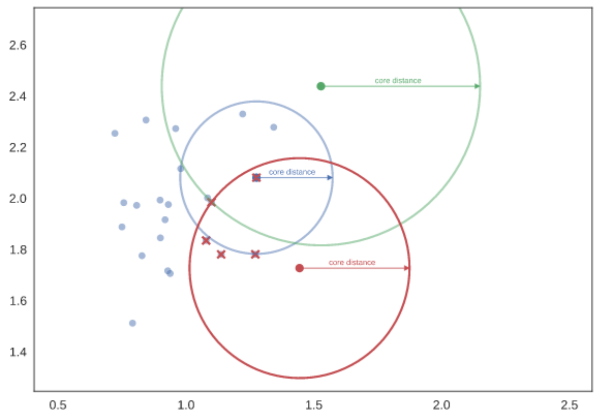


Figure 1: The core distances for three points shown as circles. Source[20]

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. Therefore 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.

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, by sorting edges of the tree by distance and iterate through, creating a new merged 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 current data set. DBSCAN simply cuts through the hierarchy using a fixed parameter,

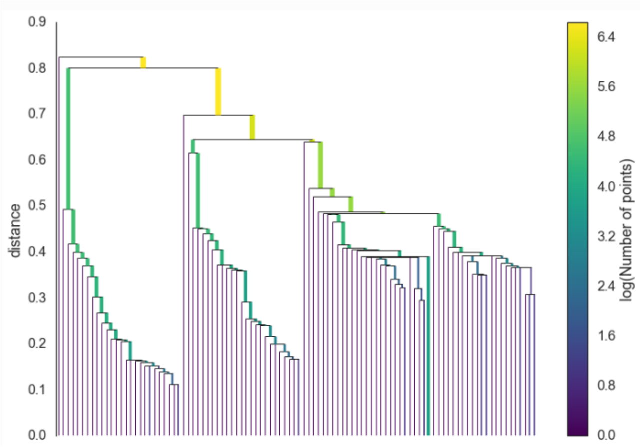


Figure 2: The cluster hierarchy shown as a dendogram. Source[20]

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.

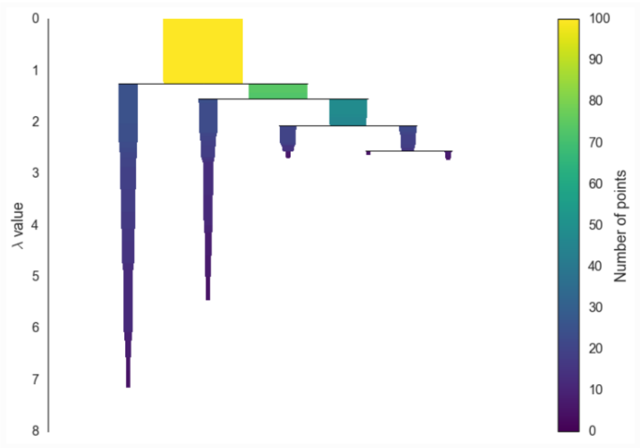


Figure 3: Condensed cluster hierarchy. Source[20]

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. This is repeated until no more splits can be made.

5. Extract the clusters The extraction of the final clusters from the condensed tree is based on the stability per cluster and once it is selected, none of its subclusters can be chosen. 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

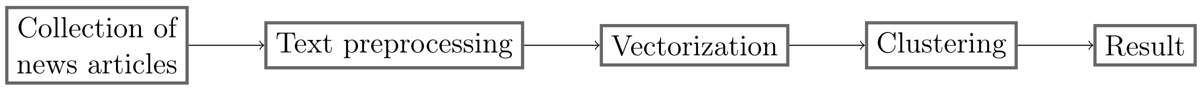
$$\sum_{p \in \mathrm{C}}^{|C|} (\lambda_p - \lambda_{birth}) \tag{3}$$

where λ_p describes when point p fell out of the cluster and λ_{birth} describes when the cluster was created. Now 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 the sum of stabilities of its children. If the sum 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. Based on this approach the final clusters will be selected, with regards to varying densities and noise.

4 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. Once the data is available, we evaluate HDBSCAN and determine the settings, which lead to optimal results regarding our specific use case. The final part applies the results obtained from the previous evaluation in an online setting.

4.1 Data flow



TODO describe

4.2 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.

4.2.1 Data set candidates

As our goal is to detect events in data streams, we've evaluated different data sets and their possibilities to extract events from their data themselves.

Data set	**Number of rows**	**Description**
GDELT 2.0	575'000'000+	Print and web news from around the world.
ChallengeNetwork	4'449'294	Network packages including anomalies.
One Million Posts Corpus	1'011'773	User comments to news articles.
Online Retail Data Set	541'909	Customer retail purchases of one year.
News Aggregator Dataset	422'937	Clustered news articles.
Dodgers Loop Sensor Data Set	50'400	Number of cars driven through a ramp.
10k German News Articles	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 behavior 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 events:

1. Dodgers Loop Sensor Data Set
   - 81 labeled events.
2. News Aggregator Dataset
   - 422'937 labeled events.

As we didn't want to lose too much time in manually clustering data, we've decided to go with one of these two. Regarding our two options, our choice was simple:
We went for the **The News Aggregator Dataset** 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.

4.2.2 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 articles specifically. We decided to use Newspaper3k[21], a Python3 library that allows us to retrieve the text from news articles easily.

The library only requires an URL to download and extract the news article from a website, see following example:

```
from newspaper import Article

url = 'http://fox13now.com/2013/12/30/new-year-new-laws-obamacare-pot-guns-and-drones/'
article = Article(url)
article.download()
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've loaded the data set into a database and run 8 concurrent processes which retrieved the news articles content from the web in different batches.

4.2.3 Data cleansing

The data set contains news articles collected from March 10th to August 10th of 2014. Five years later, many resources are not online anymore or are not accessible from Europe due to GDPR. We've used following SQL query to filter out news articles that were most likely corrupt:

```
SELECT *
FROM news_article
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.

4.3 Clustering Evaluation

4.3.1 Design

The goal of the clustering evaluation is to find the optimal parameters and preprocessing methods for applying HDBSCAN in an online setting. Therefore the clustering evaluation is designed to run HDBSCAN on our test data, using a combination of different text processing methods, vectorizers and parameters. Additionally each evaluation includes k-means as well, to provide a benchmark to compare HDBSCAN to. Once a clustering has been performed, the result is measured based on the ground truth and stored in a database for later analysis.

Another important consideration is the variety of samples to use for a clustering run. Using only a single 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 a certain number of times and each repetition will load a new sample set. New sets of samples are chosen linearly instead of randomly. For example if we define the number of repetitions as two with a sample size of 1000, the evaluation will first be done on the first 1000 samples with all possible settings and the second run will load the next 1000 samples, thus containing sample with indices ranging from 1001 to 2000. 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.

4.3.2 Scoring Function

The scoring function is essential for measuring 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 two main categories. Internal measures determine the score based on criteria derived from the data itself and external measures depend on criteria non-existent in the data itself such as class labels. 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. The NMI is an entropy-based measure and tries to quantify the amount shared information between the clusterings. 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 significant difference between the true amount of clusters and the approximation using $\sqrt{n}$. One explanation for this result is the bias of NMI for higher numbers of clusters[22].

Other scoring functions such as V-Measure or the Adjusted Rand Index showed similar unexpected results with different clusterings. Therefore we decided to develop our own scoring function based on the ideas of Maximum Matching[23] and the Jaccard Index, which we call MP-Score.

Algorithm	Sample Size	NMI	n_{true}	$\mid n_{true} - n_{predicted} \mid$
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

$$\frac{|A \cap B|}{|A \cup B|} \tag{4}$$

To illustrate the process we start with an example. We use 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 (4), for each possible pair 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 non-trivial, since clusters do not share labels across different clusterings. To solve this, we make two assumptions:

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 those 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 an similarity matrix B, which does contain a collision:

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

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 an similarity greater than 0.8. Therefore 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.

As a third step we have to calculate the weights to be used for the final 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

$$w_{ij} = \frac{|t_i| + |c_j|}{|T| + |C|} \tag{5}$$

where T is the ground truth with $t_i \in T$ and C the predicted clustering with $c_j \in C$. Therefore 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 if the overall number of predicted clusters is different from the ground truth. Using only the true number of elements as the weight, would affect the score if $|C| < |T|$, but not $|C| > |T|$. Therefore the number of predicted elements has to be included as well.

In the fourth and final step we calculate the weighted average

$$\text{MP-Score} = \sum_{i=0}^{|S|} w_i s_i \{w_i \in W \wedge s_i \in S\} \tag{6}$$

where S is the similarity matrix with $s_i \in S$ and w_i the corresponding weight 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 calculation for the final average would be done 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 cluster C with the true cluster is 0.674.

Comparison against other measures The test scenarios in table 6 show the resulting scores of our similarity score, NMI and completeness. It is important to note that NMI and completeness work with cluster labels assigned to each document, instead of considering elements inside a single cluster. This means the clustering will be flattened into one dimension, where each document is assigned the label of the cluster it appears in. The array containing the labels for the first scenario would look as follows: $C = [1, 1, 1, 2, 2, 2, 2, 3, 3]$.

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

As a final note, repeating the evaluation shown in table 5 a second time using the MP-Score, the score (Table 7) for K-means is much lower than HDBSCAN. This reflects what we would expect based on the big difference in the amount of predicted clusters.

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

Table 7: The similarity score reflects the difference in number of predicted clusters.

The full implementation of the scoring function can be found in the appendix as Listing 6.

4.3.3 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 vector space model.
- Defining a range for each parameter of a clustering method and running it with each possible combination of those parameters.
- Storing the result 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 is done with Python. Clustering methods and vectorizers are provided by the scikit-learn library[18], while the specific HDBSCAN implementation is provided by a scikit-learn-contrib package[6]. 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. Additionally 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 was useful for developing and testing the framework in a local environment and deploying it on a remote server for long running evaluations, without worrying about setting up and installing all dependencies again.

Defining cluster parameters 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 two different metrics and eight 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 a number of settings. 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.
data_mining
optional arguments:
  -h, --help            show this help message and exit
  --rows ROWS           number of samples to use for clustering
```

```
                          default: 1000
--stories STORIES         number of stories to load samples from. This parameter
    overrides the rows parameter if set.
--methods METHODS         options: kmeans, hdbscan, meanshift, birch,
    affinity_propagation, spectral_clustering
                          default: all available options
--vectorizers VECTORIZERS
                          options: CountVectorizer, TfidfVectorizer
                          default: all available options
--tokenizers TOKENIZERS
                          options: newspaper_text, text_keyterms, text_entities,
    text_keyterms_and_entities, text_lemmatized_without_stopwords,
    text_stemmed_without_stopwords
                          default: all available options
--runs RUNS               number of runs per clustering method
                          default: 1
```

Listing 4: Command line interface for the evaluation framework

4.4 Online Clustering

4.4.1 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 to the same subject. 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 events as follows:

- New event: A new cluster of news articles appears in the data stream, which describe 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 as discovered in the previous 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.

Since HDBSCAN only supports static data sets, the clustering will be done in batches using a time based sliding window approach. Events are detected by comparing the resulting clusters with the previous ones.

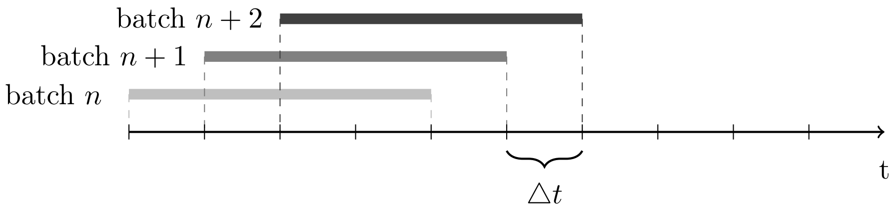


Figure 4: Timeline showing the sliding window approach

Finding pairs clusters Thus we define our two events as follows: To be able to compare clusters of different batches with each other, we have to find pairs of clusters between batches, which describe the same story. This is done by applying the same assumptions as for the scoring function used in the evaluation. Therefore clusters are paired based on their similarity calculated with the Jaccard index as shown in equation 4. If the similarity is above a certain threshold, both clusters are seen as describing the same story.

Sliding window An important consideration for determining existing or new clusters is the overlap of samples between batches. If the overlap is too small, similar clusters will no longer be detected as such, which would result in an increasingly high error rate. Finding optimal values for the step size between batches and the number of samples for each batch is therefore essential for our online clustering approach.

4.4.2 Implementation

The online clustering implemented for this thesis does not operate in a true online setting, but rather it takes our existing test data and simulates a 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 is done with Python and runs in a similar dockerized environment as the evaluation framework.

Comparing clusters To detect events between batches of clusterings, we have find pairs of clusters describing the same story. This problem is solved in the evaluation framework as part of the scoring function, by calculating the similarity for each possible cluster pair. The resulting time complexity is $O(n^2)$, which we deemed acceptable for the static evaluation. However in a dynamic setting such as the online clustering, performance is an important factor, since it restricts the lengths of the time delta between batches and the overall batch size. Thus we decided to use Locality-Sensitive Hashing (LSH)[24] to find similar clusters. This reduces the time complexity to $O(log(n))$. The implementation for LSH is provided by the datasketch library[25].

Detecting Events Once we have found pairs of clusters, which represent the same story, detecting events becomes trivial. For each pair we look for news articles, which are only present in the new cluster. These articles are then summarized in as an extension of an existing event. Clusters from the new batch without a matching cluster from the previous batch are seen as new events.

Measuring the quality of events Since events are themselves clusters of news articles, we apply the MP-Score to measure detected events against events taken from the ground truth. This gives an indication if the detected events contain the same news articles as true events and thus the rate of false positives and false negatives. Calculating the score is $O(n^2)$, but since the application runs on a simulated timeline, time complexity is only a minor concern.

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] [--full-cluster] [--verbose] [--rows ROWS]
                                [--full_rows FULL_ROWS] --date DATE
                                [--run_n_days RUN_N_DAYS] [--threshold THRESHOLD]

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

required arguments:
  --date DATE

```

```
optional arguments:
  -h, --help            show this help message and exit
  --full-cluster        run a full clustering once a day
                        default: False
  --verbose
                        default: False
  --rows ROWS           number of samples to process per batch
                        default: 1000
  --full_rows FULL_ROWS
                        number of samples to process for the full clustering
  --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

5 Results

5.1 Clustering Evaluation

The goal of this evaluation is to measure the accuracy of HDBSCAN, with different parameters and preprocessing methods. The most suitable settings will then be used for the online clustering approach to detect changes in a news stream.

5.1.1 Setup

Text Preprocessing The first step in working with text is to apply Natural Language Processing techniques for improving the quality of the data before clustering it. We look the five different preprocessing methods as described in section 3.1 and evaluate each. The methods are:

- Full text with stop word removal
- Key terms
- Named entities
- Text lemmatisation
- Text stemming

Text Vectorization Before the text can be clustered, it has to be transformed into a vector space model. We look at two different models:

- Word 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 two to nine 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 baseline to evaluate HDBSCAN, we provide the true number of clusters for each run. Therefore k-means runs with an optimal starting point.

Running the evaluation The evaluation is done with different sets of news articles per run. This means if we define a run to use 30 stories and set it to repeat five times, each repeat 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.

5.1.2 Evaluation

The first run is done with 60 stories, which results in approximately 2000 news articles, over 20 repetitions. Table 8 shows the resulting MP-Score for each parameter in combination with each preprocessing method and vector space model. The highest score per parameter is highlighted as bold. The first insight we get is the variety in scores for different min cluster sizes. The lowest min cluster size results in the lowest score, while increasing this parameter leads to an increasingly better score. The highest score is reached with a min cluster size of six, while increasing it further reduces the score again. The large difference in scores between different min cluster sizes, shows the importance this parameters 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 5, with clusters containing as few as two news articles. Based on this distribution we expected the ideal min size cluster size to be in a range from two to nine. The distribution also explains the drop in accuracy after a min cluster size of 6, since an increasingly number of clusters are being ignored.

Clustering	**Word Frequency**					**tf-idf**				
HDBSCAN	Full Text	Key terms	Entities	Lemmatised	Stemmed	Full Text	Key terms	Entities	Lemmatised	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 8: The average MP-Score for combinations of parameter and preprocessing with a sample size of 60 stories (approx. 2000 articles)

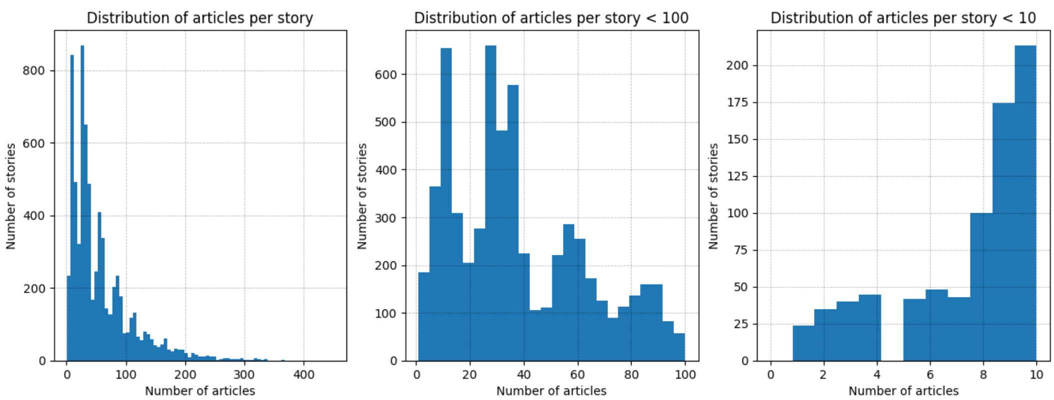


Figure 5: Distribution of cluster sizes.

Comparing the two vector models, shows all of the best scores per parameter have been achieved by using tf-idf. Additionally the different metrics show a significant difference when using the vector model based on word frequency. With tf-idf the difference between both metrics is still notable, but far less drastic than based on word frequency. The overall better performance of the cosine metric over the euclidean distance is due to the high dimensionality and the sparseness of the vector space.

This is behaviour has already been studied in the past[26][27] 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, text lemmatisation provides the highest overall mp-score with 0.738, although closely followed by text stemming. This is to be expected, since both lemmatisation and stemming reduce the dimensions by grouping words into their base form, while still retaining most of the text. In contrast to key term and entity extraction, which both result in a drastic reduction of the dimensions, and therefore less detail. It is also interesting to see how close the score from using the full text is compared to the best score per row. The difference between the overall best score of 0.738 achieved by lemmatisation and the score provided by the full text of 0.731 is only 0.007. This means text preprocessing has a lesser impact than initially expected. However it is important to note, that we used pretrained models for key term and entity extraction. Specifically training on a news corpus might improve the performance of both methods, but it was decided as to be out of scope for this thesis.

After determining the optimal settings for text preprocessing 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 6 shows the scores achieved with different parameters over an increasingly large set of samples. Based on this Figure we see the metric *cosine* to be generally better than *euclidean* and significantly more stable based on the range of the score and the quality of the clustering seems to decrease with larger sample sizes.

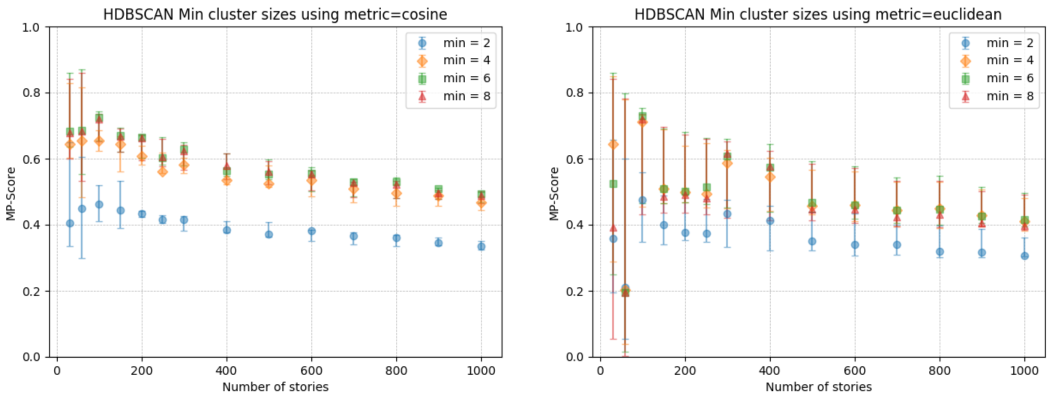


Figure 6: MP-Score for different parameters, where min stands for the min cluster size. The marker represents the median, while the vertical line indicates the range between the min and max values. Each run contains at least five repetitions.

Furthermore the variance with smaller sample sizes can partially be explained through differences in the number of detected clusters, since missing a few clusters has a bigger impact if the overall number of clusters is small. Figure shows the difference between the number of predicted clusters and the number of true clusters. The Figure provides us with an interesting observation: While so far the minimum cluster size of six has given the best scores, the difference in the number of clusters is much smaller with a minimum cluster size of four. 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 one up to 400. Based on this knowledge and the workings of the score, we can conclude that using a minimum cluster size of four gives us more clusters, which are ignored by using a larger minimum cluster size, but at the same time fragmenting larger clusters. Therefore resulting in a lower score, while having a better difference in the number of cluster predictions. This can be validated by analysing our data directly, where we observe the number of predicted cluster to be higher the lower the minimum cluster size is. Using $min\_cluster\_size = 6$ tends to give a lower number compared the the true amount, while $min\_cluster\_size = 3$ gives usually a higher number than there are actual clusters.

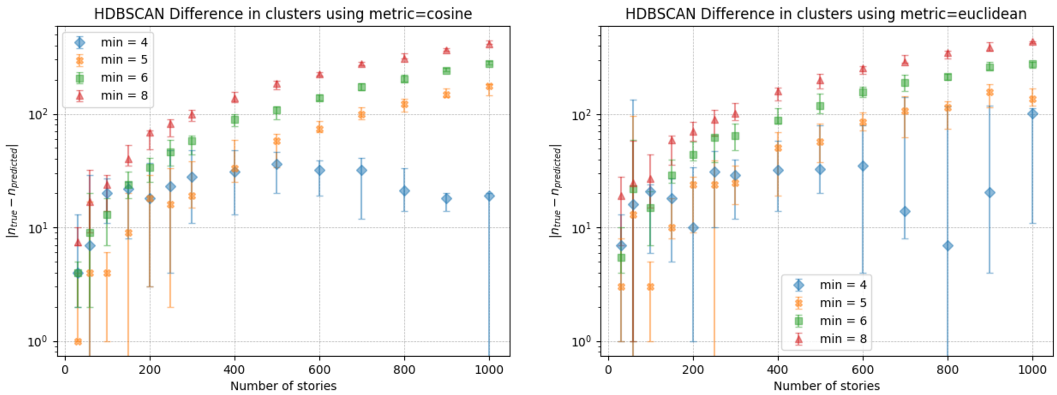


Figure 7: Difference between the predicted and true number of clusters.

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 8 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 influence is due to the $min\_cluster\_size$. Each news article belonging to a cluster, which has less articles than the minimum cluster size, will be counted as noise. Table 9 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.

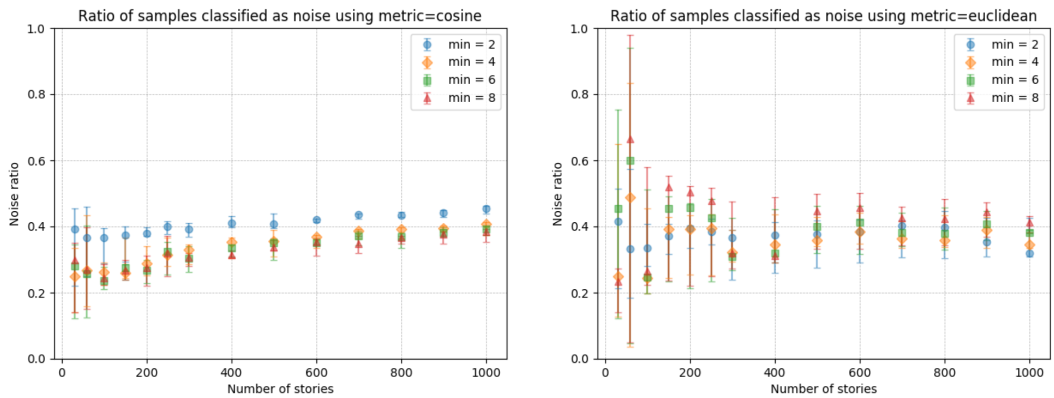


Figure 8: Number of news articles classified as noise.

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 10 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 length of 13980 is nearly twice as long compared with 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 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

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

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.

Detected Articles			
Nr.	Title	Text 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	Text 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 10: 10 correctly detected and 10 missed news articles, which all belong to the same story.

To find a possible explanation for the remaining new articles, we take a look at their tf-idf model. Table 11 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 explain why the articles were missing from the cluster. However it is important to note, that the tf-idf model in Table 11 was created from only those 20 articles and used the full text instead of lemmatisation for better readability. The top keywords might differ in the model created for the whole clustering, but we assume the difference in the weights per keyword to be similar enough to the limited model. This indicates, that further work on the vector space model might lead to considerable improvements in the quality of the clustering.

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

So far we focused on HDBSCAN to determine the optimal settings to run it with. As a next step we can start comparing the overall performance with k-means. We use the follwing settings: Text lemmatisation with tf-idf, cosine as the similarity measure and six as the minimum size of clusters. Figure 9 shows a similar behaviour for both clustering methods in value and variance of the accuracy. Although HDBSCAN is generally more accurate than k-means, the difference gets smaller with an increase in the sample size. Additionally the increase in the number of samples results for both HDBSCAN and k-means in a small loss regarding the accuracy as can be seen in Figure 9 and which we have already observed when analysing HDBSCAN parameters.

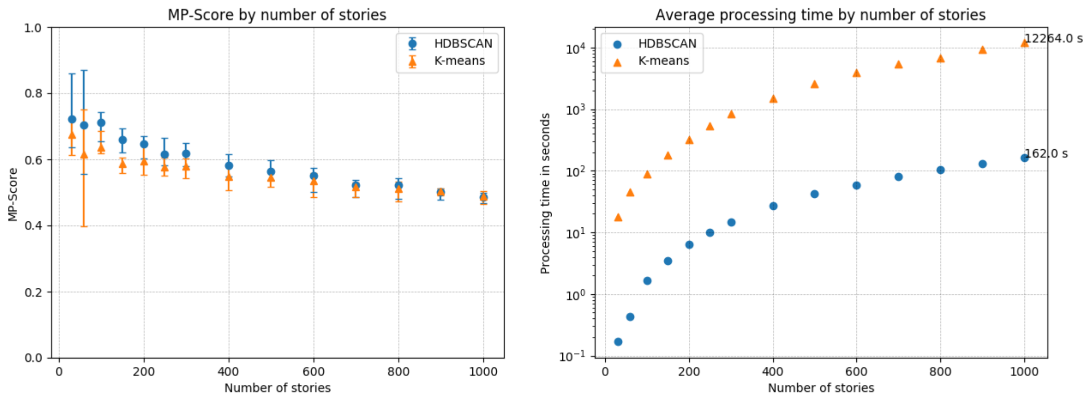


Figure 9: Comparison of the MP-Score and processing time between k-means and HDBSCAN

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 9. 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[28]. This might be another area for further improvements, although it will not help to increase the score on larger data sets.

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 10. HDBSCAN provides the highest accuracy, 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.

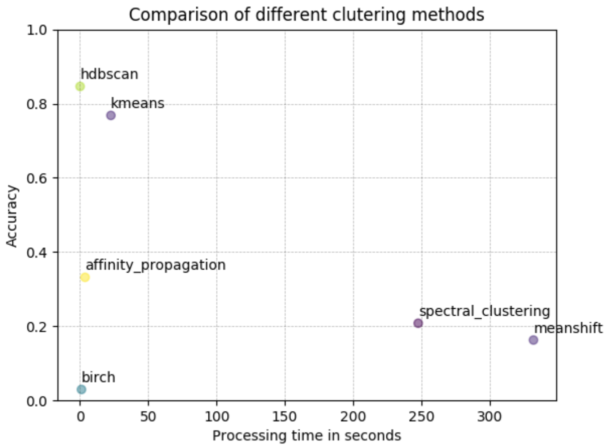


Figure 10: Comparison of different clustering methods with a sample size of approximately 1000 news articles

5.1.3 Conclusion

The evaluation has shown HDBSCAN to be a good candidate to use for text based clustering. It provides an better accuracy 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 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 accuracy, while still leaving room for further improvements.

5.2 Online clustering

5.2.1 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 lemmatisation
- Vector space model: tf-idf
- Clustering method: HDBSCAN
- Minimum cluster size: 6

- Distance metric: cosine

The clustering is run over 30 days with a total of 42'916 news articles. The distribution of news articles this time period is illustrated in Figure 11. The time delta, which is the amount of time between two batches, is set to one hour.

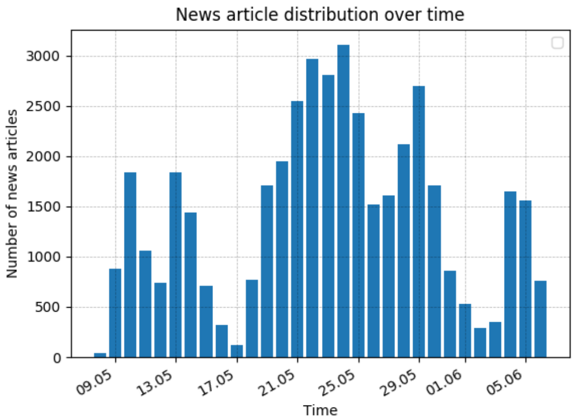


Figure 11: Incoming news articles over 30 days

5.2.2 Evaluation

The goal of the online clustering is to detect new events in an incoming stream of news articles and changes in existing events. This evaluation analyses the results of our simulated test runs with different batch sizes.

Figure 12 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 accuracy in detected events. The difference with a batch size of 5000 news articles is significantly lower than the batch size of 1000. 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 11. During this period we receive up to 3000 news articles in a single day. This means by using a lower batch size such as 1000, the overlap between batches gets too small to reliably detect changes between batches, which causes too many new topics to be detected.

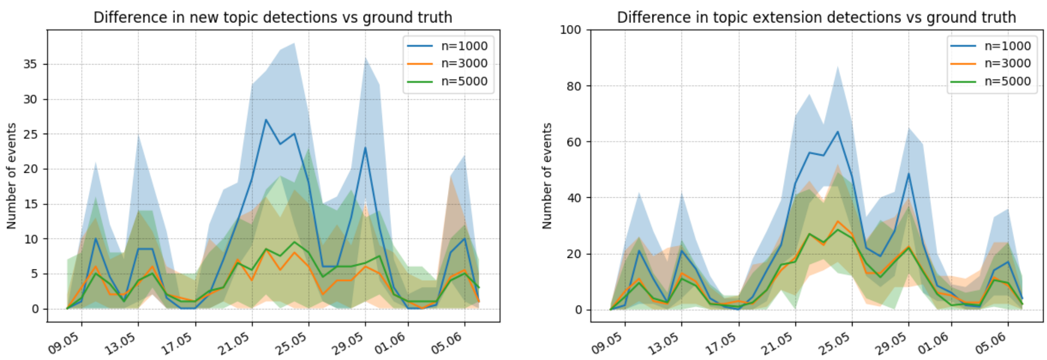


Figure 12: Comparison between the difference in detected and true events. The line represents the median, while the area shows the range from the minimum to the maximum value

Although a larger batch size does not simply equal a better difference, as can be seen in Figure 12 by comparing the differences using a batch size of 3000 with a batch size of 5000. 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 less significant when detecting changes in existing events.

Based on the overall differences, we do not know the accuracy of those predictions. If the difference between newly detected events and true events is zero, there is still the possibility, that the events itself 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 quality of the detected events over the ground truth. Figure 13 shows the MP-Scores for an online clustering using a batch size of 1000. 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.

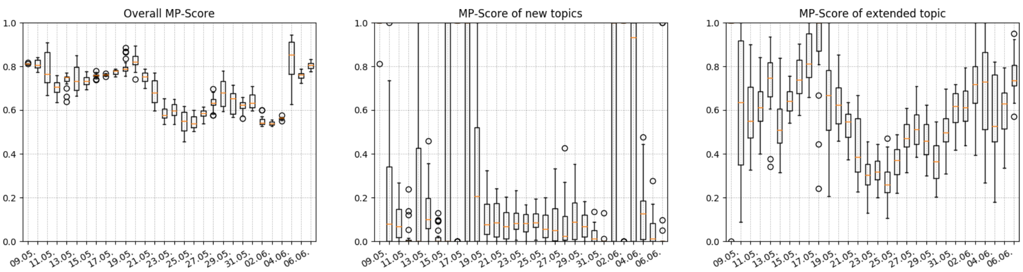


Figure 13: MP-Scores for clusterings using batch size of 1000

Looking at an increased batch size of 5000 in Figure 14, we note that there is less variance in the overall MP-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.

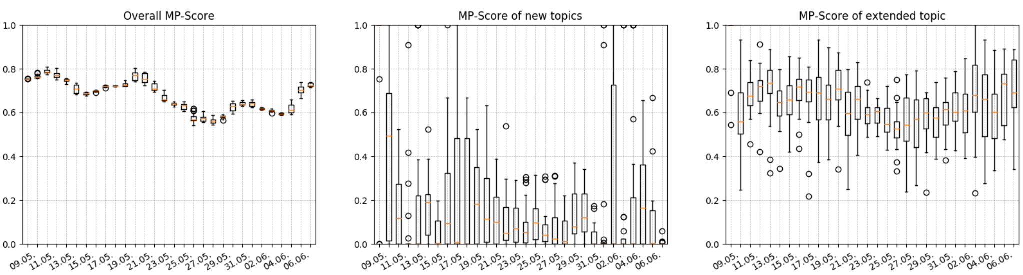


Figure 14: MP-Scores for clusterings using batch size of 5000

One of the reasons for the difference in the accuracy of the detection of new events and the extension of events might be explained by the $min\_cluster\_size$. In the current setting the $min\_cluster\_size$ is set to 5, which means if an event occurs in batch one 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 occurrences, but the ground truth treats it as an existing event. 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 *min_cluster_size*.

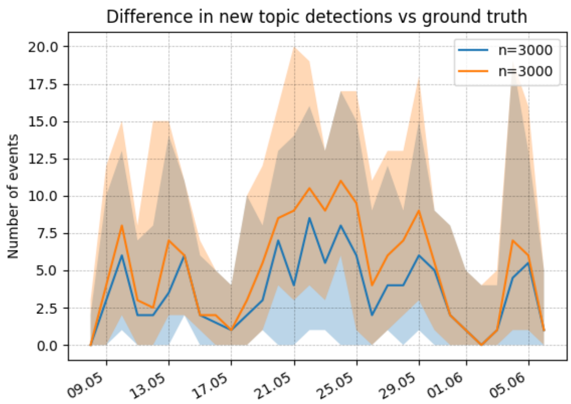


Figure 15: Differences in predictions vs ground truth using batch size of 3000.

Limiting new events in the ground truth based on the *min_cluster_size* gives the opposite result as initially expected. Figure 15 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 *min_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 16 validates this assumption.

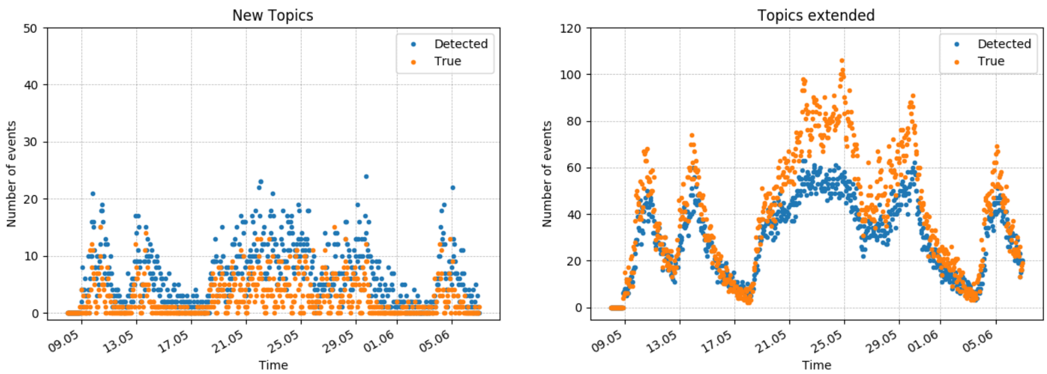


Figure 16: Number of events with a batch size of 3000

The raw data in Figure 16 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 use of Locality Sensitive Hashing as explained in section 4.4.2. The current threshold value for determining the similarity is set to 0.75. To see the impact on the similarity threshold, we run the online clustering again with a batch size of n=3000 and different threshold values.

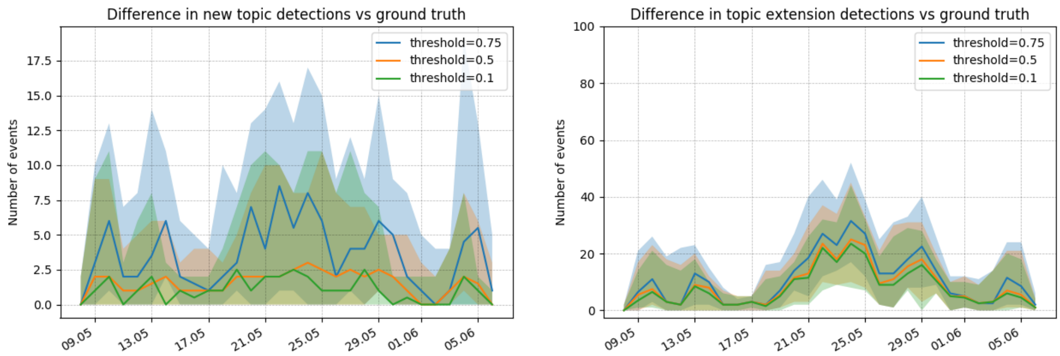


Figure 17: Differences in detected over true events with different thresholds

Figure 17 shows the effect of the threshold on the difference in decided over true events. We see how the initial threshold of 0.75 was set too high, as lower threshold values such as 0.1 provide a significant lower difference. While there is still a substantial variance per day the median of 0.1 and 0.5 is generally more stable and lower than with a threshold value of 0.75. The reason for the better performance of lower threshold values, 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 17. 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 18. While there is more variance than in similar plots from Figure 13 and Figure 14, 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.

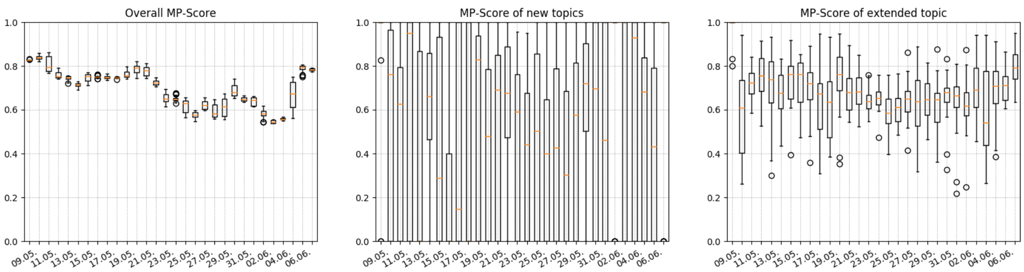


Figure 18: MP-Scores for online clustering with batch size n=3000 and threshold=0.1

Additional reasons for the general low performance in the quality of events might be due to the noise rate and the difference in the number of news articles. As described in the section about the cluster method evaluation, the noise rate for sample sizes between 1000 and 5000 ranges from 20% to 30%. This means a significant amount gets discarded as noise and thus potential new events might not even be detected, because too many of their news articles are discarded. Once an event is detected, detection in changes are more reliable than for new events, but as the variance in the MP-Score shows, there is still a fairly large error rate.

As a final note let us look at two specific examples. One where the detection was mostly accurate and a second where the detection failed. Figure 19 shows the result for the story regarding the "Gmail redesign". Each dot represents one or more news articles, which appear at a certain time in the data stream. All news articles belong to the same story. As we can see after five news articles appearing in the stream, a new event is detected marked as green. Following the new event, additional news articles are clustered and detected as part of the existing event marked as blue. The last cluster does not contain all news articles any more, since the first few articles were not part of the last batch. However there is still enough overlap with the cluster from the previous batch, to match this cluster to the existing event about the "Gmail redesign". This example shows a good example of how the online clustering works in the optimal case.

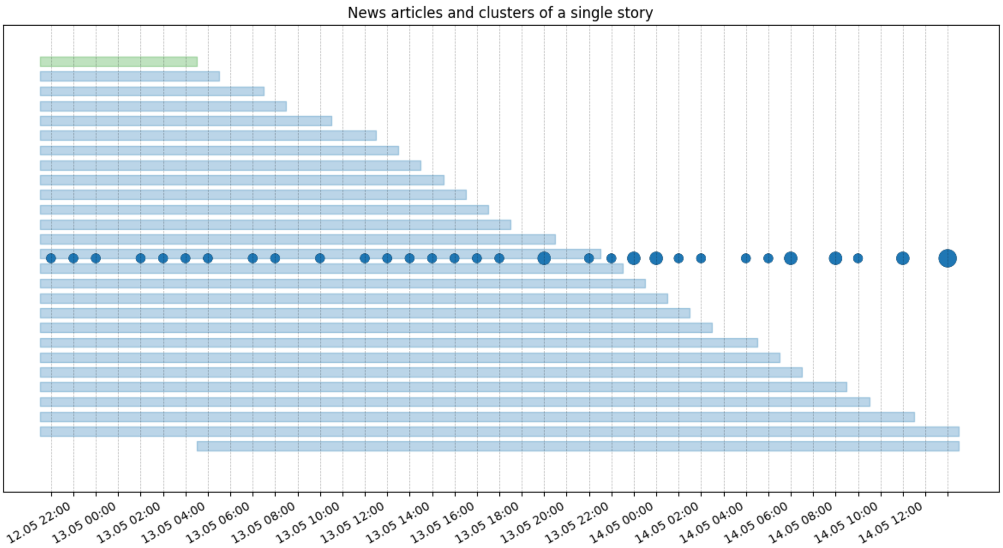


Figure 19: News articles belonging to the story "Gmail redesign" together with clusters, where they appear in. The circles represent incoming news articles at a specific time. The rectangles show which news articles are contained within a cluster. New clusters are marked as green and existing cluster are blue.

The second example shown in Figure 20 illustrates the opposite case, where the events do not match well with the ground truth. All news articles in this example belong to a story about the release of new film called "Bad Neighbours". Black dots indicate news articles, which do not appear in any cluster. The fragmentation of the clusters themselves clearly shows, that many of these news articles have been matched with different stories and therefore in different events.

The difference in the performance of both examples, can be explained based on the news articles themselves. The first example about the "Gmail redesign" consists mostly of press releases, technology focused news sites such as *Ars Technica* or *PC Magazine*. Therefore the contents of the news articles are of high quality and share the same technical vocabulary. The second example with the new release of the movie "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. Additionally the vocabulary is more general than compared to technical articles and varies stronger between articles. This might lead to differences in the tf-idf model, causing the articles to be not considered similar enough to belong to the same cluster, as was shown in the clustering method evaluation with Table 11.

5.2.3 Conclusion

While the overall clustering results in good scores, the detection of new events and changes in existing events is more sensible to smaller inaccuracies and the volume of incoming data. We explored different

Figure 20: News articles belonging to the story "Bad Neighbours" together with clusters, where they appear in. Circles marked as black represent news articles, which do not appear in any cluster.

settings for the batch size, threshold values and looked at two specific examples for different clusters. It turned out our initial similarity threshold value was set too high and lowering the value resulted in a significant improvement detecting new events. Although the overall performance was still very varied. Possible reasons have been explored such as the *min_cluster_size*, the noise rate, the general difference in the number of news articles or the vector space model representing news articles. As a result we conclude that the accuracy of the clustering method used in this approach is insufficient for the detection of events in a news stream, but it would be certainly interesting to see it be applied with different kinds of textual data streams.

6 Conclusion

6.1 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 their corresponding cluster labels. Having a labelled data set allows us to apply external measures and evaluate a resulting clutering against the ground truth. After selecting a few data set for closer inspection, we settled on the News Aggregator Dataset, 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 approximately XXX 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. After having done many clustering evaluations based on our test data, we focused on analysing the collected data for our primary clustering method HDBSCAN and compared it to k-means. The analyse 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 bigger the sample size got. 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.

Having determined the optimal settings in the HDBSCAN evaluation, we applied them in a simulated online setting for event detection. The event detection was accomplished by running the clustering in batches over time. Events are detected by comparing a batch with its predecessor. If a batch contains clusters, which do not appear in the previous batch, than those clusters are considered as new events. If a cluster did exist in the previous batch, we look at the difference in assigned news articles and can therefore detect changes in existing event. The detection of events is measured against the ground truth. We explored different batch sizes and 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. 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 inaccuracies in the overall clustering, which explains the high error rate we observed in detecting new events. In conclusion we found the error rate in detected events to be rather high for our approach and therefore not applicable in a real world scenario in its current state.

6.2 Future work

The approach in its current state still leaves different areas up for improvement. Further work on named entity recognition, 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 named entity recognition 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 lemmatisation. Thus leading to a potential improved clustering and reducing the noise rate.

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.

6.3 Lessons learned

HDBSCAN

preprocessing

knowing your score

Good data set -> noise rate

7 Index

7.1 Bibliography

[1] 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.

[2] 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.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

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

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

[15] 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.

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

[17] 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.

[18] 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.
[19] *Scikit-learn: Tf–idf term weighting*, https://scikit-learn.org/stable/modules/feature_extraction.html\#tfidf-term-weighting, Accessed: 2019-06-03.
[20] *How hdbscan works*, https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html, Accessed: 2019-05-25.
[21] *Newspaper3k: Article scraping & curation*, https://web.archive.org/web/20190312144257/https://newspaper.readthedocs.io/en/latest/, Accessed: 2019-03-12.
[22] 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.
[23] W. M. J. Mohammed J. Zaki, "Data mining and analysis, Fundamental concepts and algorithms", in. Cambridge University Press, 2014, ch. Chapter 17: Clustering Validation.
[24] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt, *Practical and optimal lsh for angular distance*, 2015. arXiv: 1509.02897 [cs.DS].
[25] 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.
[26] 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.
[27] A. Huang, "Similarity measures for text document clustering", *Proceedings of the 6th new zealand computer science research student conference*, Jan. 2008.
[28] *Optimizing hdbscan for huge datasets*, https://github.com/scikit-learn-contrib/hdbscan/issues/212, Accessed: 2019-06-01.

7.2 Glossary

API An application programming interface allowing access to data or features of an application.. 42

Clustering The task of grouping a set of objects based on their similarity.. 42

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.. 42

Dockerized An application environment running as a single or a collection of docker containers.. 42

Stop word A term that is overall so frequently used, that it is ignored in natural language processing.. 11, 42

Vectoriser A vectoriser transform a text into a numeric vector.. 42

7.3 List of Figures

7.4 List of Tables

7.5 List of Abbreviations

CNN Convolutional neural network. 9, 42

IoT Internet of things. 6, 42

8 Appendix

8.1 Algorithm for MP-Score

```python
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()
```

```python
    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.