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

How can events be recognized in a data stream? And what is the definition of an event? Current event recognition procedures define possible events (or event types) statically when the system starts. However, in data streams, the definition should develop dynamically over time as the data and their focus changes over time.

The first goal of this bachelor thesis is to define what an event is and how events can be recognized from static data. Depending on the definition, a suitable data set has to be found and chosen in order to create a system which is capable of recognizing events. In the second part, the events should be recognized from a data stream and therefore support dynamic event recognition.

This work uses news API's as data streams and tries to assign the received news articles to different events. An event is defined as a list of different news articles writing about the same story. A possible event could be "Brexit" or an upcoming election. It's important to keep in mind that new events can and should be created over time.

Assigning news articles to an event can be done by clustering the news articles. In order to achieve best possible results, a comparison between different clustering techniques has been completed. It was revealed that HDBSCAN is a promising candidate for our use case as it delivered the best results.

Unfortunately HDBSCAN was not made for data coming from data streams but this thesis will show one possible solution on how to deal with this. This work was able to process up to 20'000 news articles with HDBSCAN. A possible continuation of this work could try to extend this number to process more data at once.

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 the application in 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

Todo.

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

Todo.

3.1 Text preprocessing

Todo.

3.1.1 Keyterm extraction

Todo.

3.1.2 Entity extraction

Todo.

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[9] 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[10] 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[11] 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[12] and the WordNet stemming algorithm[13]. 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:

3.2 Data Representation

Todo.

#	**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.

3.2.1 Word Frequency

Todo.

3.2.2 tf-idf

Todo.

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't be assigned to a cluster, it will be marked as noise point.

Data points that belong to a cluster but don't 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 don't 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.

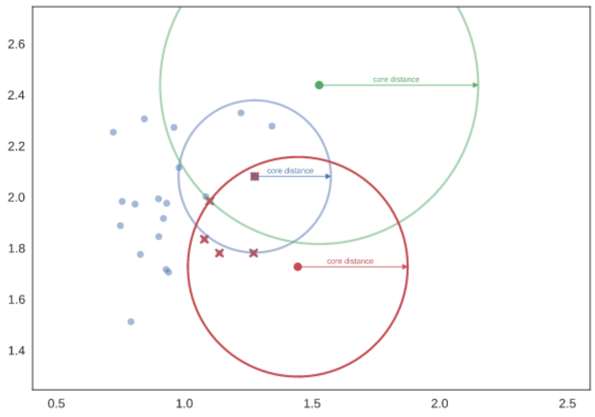

Figure 1: The core distances. TODO give more detailed explanation.

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, 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

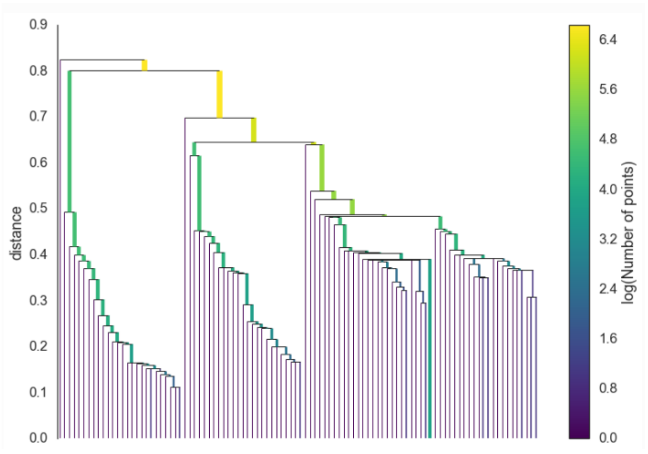

Figure 2: The cluster hierarchy shown as a dendogram

optimal values. This is were HDBSCAN improves upon DBSCAN, by taking additional steps for finding relevant clusters.

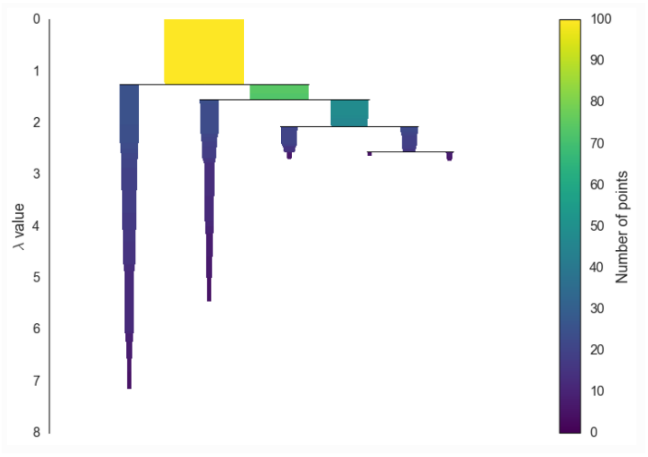

Figure 3: Condensed cluster hierarchy

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

$$\sum_{p \in \mathrm{C}}^{|C|} (\lambda_p - \lambda_{birth}) \tag{1}$$

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 Results

4.1 Clustering Evaluation

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

4.1.1 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 **??** 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 three times. Lower numbers of stories allow for more repetitions due to lower processing times.

4.1.2 Evaluation

The first run is done with 60 stories, which results in approximately 2000 news articles, over five repetitions. Table 2 shows the resulting accuracy 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 accuracy scores for different min cluster sizes. The lowest min cluster size results in the lowest accuracy, while increasing this parameter leads to an increasingly better score. The highest accuracy is reached with a min cluster size of six, while increasing it further reduces the score again. The large difference in accuracy 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 4, with clusters containing as little as two news articles. Based on this distribution we expected the min size cluster size to be low. 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	0.289	0.265	0.223	**0.305**	0.297	0.286	0.268	0.26	0.296	0.3
min_size: 2, metric: euclidean	0.101	0.093	0.110	0.101	0.106	0.301	0.170	0.241	**0.306**	0.301
min_size: 3, metric: cosine	0.488	0.456	0.465	0.48	0.487	0.472	0.446	0.457	**0.493**	0.478
min_size: 3, metric: euclidean	0.172	0.129	0.176	0.174	0.182	0.472	0.306	0.464	**0.500**	0.478
min_size: 4, metric: cosine	0.630	0.555	0.625	0.552	0.577	0.577	0.586	**0.646**	0.589	0.581
min_size: 4, metric: euclidean	0.320	0.182	0.214	0.315	0.332	0.611	0.416	0.559	0.613	**0.615**
min_size: 5, metric: cosine	0.716	0.652	0.656	**0.718**	0.718	0.688	0.664	0.632	0.686	0.692
min_size: 5, metric: euclidean	0.355	0.217	0.266	0.41	0.389	**0.703**	0.512	0.607	0.686	0.692
min_size: 6, metric: cosine	0.693	0.715	0.608	0.701	0.708	0.738	0.729	0.613	**0.751**	0.747
min_size: 6, metric: euclidean	0.179	0.280	0.292	0.202	0.164	0.738	0.408	0.622	**0.778**	0.763
min_size: 7, metric: cosine	0.631	0.611	0.552	0.643	0.634	0.689	0.685	0.553	0.718	**0.722**
min_size: 7, metric: euclidean	0.122	0.392	0.307	0.073	0.099	0.689	0.336	0.539	0.718	**0.722**
min_size: 8, metric: cosine	0.571	0.603	0.514	0.592	0.574	0.685	0.647	0.531	**0.711**	0.695
min_size: 8, metric: euclidean	0.056	0.339	0.338	0.025	0.057	0.685	0.286	0.522	**0.711**	0.695
min_size: 9, metric: cosine	0.542	0.569	0.476	0.544	0.541	0.602	0.614	0.499	0.637	**0.640**
min_size: 9, metric: euclidean	0.065	0.236	0.310	0.025	0.033	0.602	0.216	0.475	0.637	**0.640**
K-Means										
n_cluster: n_true	0.531	0.588	0.514	0.536	0.536	**0.713**	0.653	0.584	0.672	0.693

Table 2: Accuracy for combinations of parameter and preprocessing with a sample size of 60 stories (approx. 2000 articles)

Comparing the two vector models, shows the majority of best scores per parameter achieved by tf-idf. Additionally the different metrics show a significant difference when using the vector model based on word count. With tf-idf the difference between both metrics is often negligible.

As for the optimal preprocessing, text lemmatisation appears to provide the highest accuracy in general or at least being fairly close to the highest score. This is to be expected, since text lemmatisation reduces 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 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.

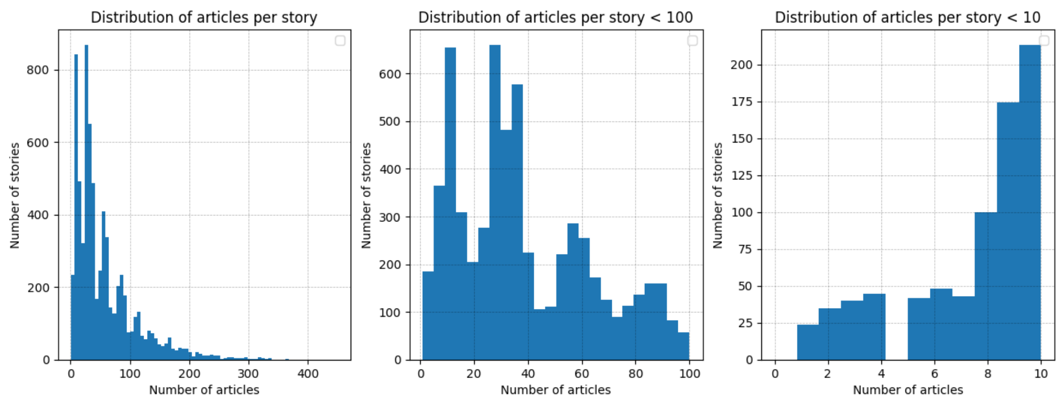

Figure 4: Distribution of cluster sizes.

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 5 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*, even if *euclidean* is occasionally more accurate.

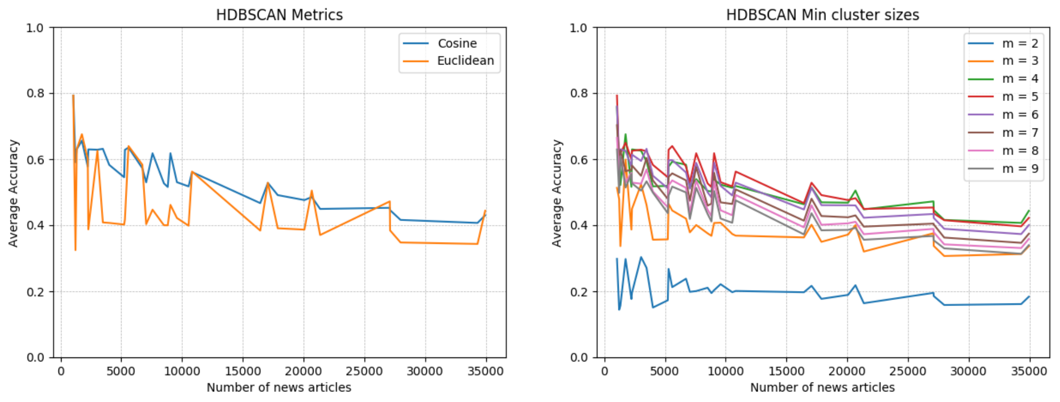

Figure 5: Accuracy for different parameters

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 6 is higher, than we would expect it to be based on our test data. A variety of factors play into the high noise ratio. One major influence is due to the used *min_cluster_size*. Each news article belonging to a cluster ignored due to a size too small, will be counted as noise. In addition to the false positives due to the min cluster size, the test data does still contain noisy data, even after our efforts in cleaning up the data as good as possible. Nonetheless the expected noise ratio based on the test data is less than 10%, nowhere close to the 20% of the current evaluation. Decreasing the noise ratio is certainly an important part in future improvements.

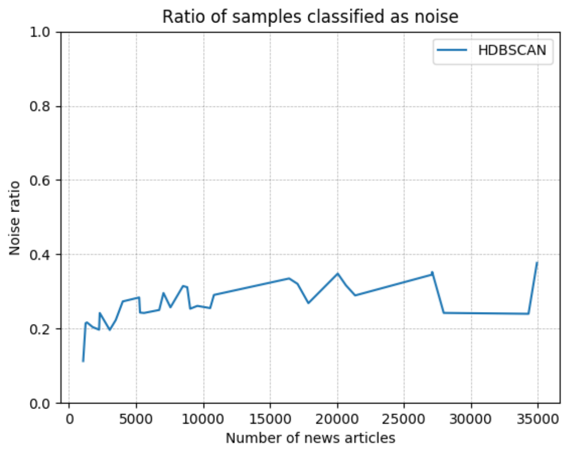

Figure 6: Number of news articles classified as noise.

Having found the optimal settings to run HDBSCAN with, we can start comparing the overall performance with K-Means. Figure 7 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.

Increasing the sample size results for both HDBSCAN and K-means in a small loss regarding the accuracy as can be seen in Figure 7. However the accuracy seems to stabilize around the 0.7 mark.

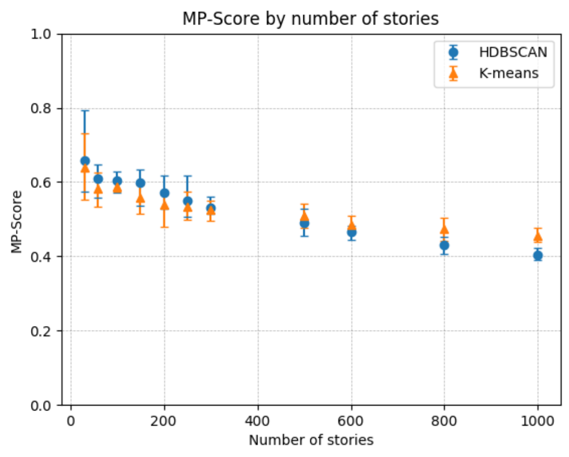

Figure 7: Comparison of the average accuracy between K-means and HDBSCAN

While HDBSCAN and K-means provide a similar accuracy, 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 demonstrated by Figure 8. 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.

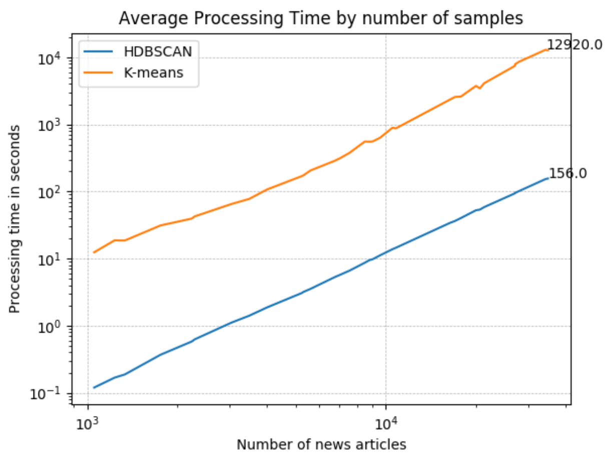

Figure 8: Processing time in seconds

Figure 9 shows, that the difference between predicted over the true number of clusters is fairly low and appears to be roughly linear with the overall number of clusters.

As a final note, 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. Based on this data, we can assume HDBSCAN to be a good candidate for our use case.

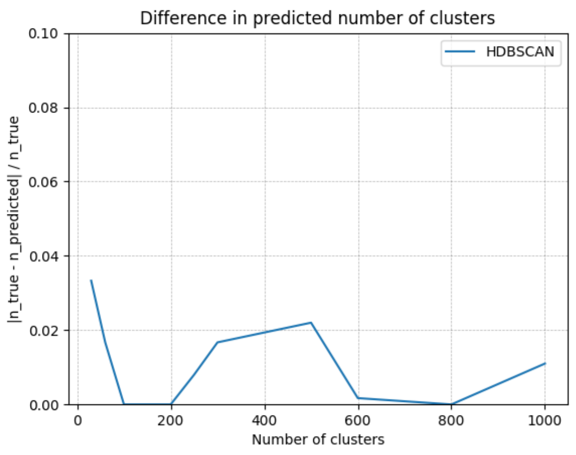

Figure 9: Ratio of difference over predicted with true number of clusters

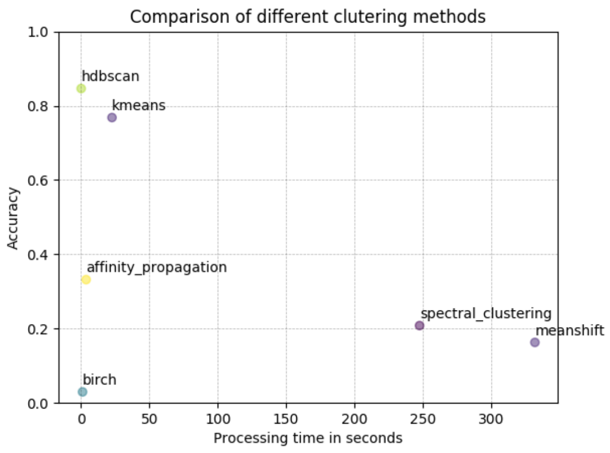

Figure 10: Comparison of different clustering methods with a sample size of approximately 1000 news articles

4.1.3 Conclusion

The evaluation has shown HDBSCAN to be a good candidate to use for news 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. On the flip side the noise ratio is quite high and the space complexity is problematic with larger data sets. Overall HDBSCAN provides an acceptable accuracy, while still leaving room for further improvements.

4.2 Online clustering

4.2.1 Setup

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

- Text Lemmatisation
- Vectorizer: tf-idf

- Clustering method: HDBSCAN
- Min cluster size: 5
- 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.

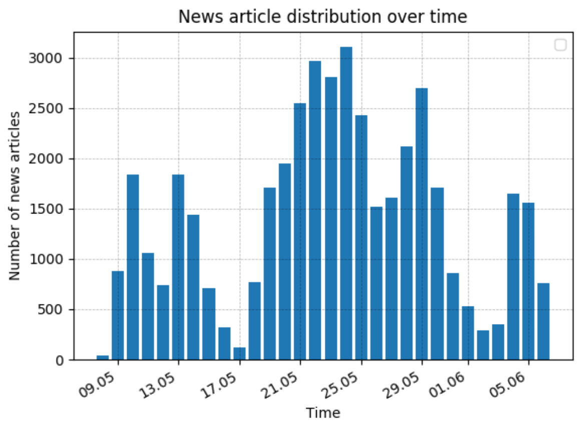

Figure 11: Incoming news articles over 30 days

4.2.2 Evaluation

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

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

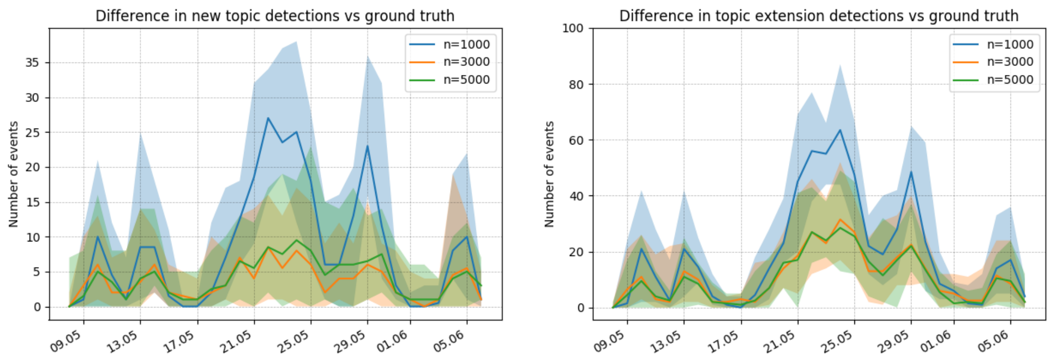

Figure 12: Comparison between the difference in detected and true events. The line represents the median, while the area shows the range from the minimum to the maximum value

large variance is already the first indication, that the quality is rather low. Meaning that there are still many false positives and false negatives.

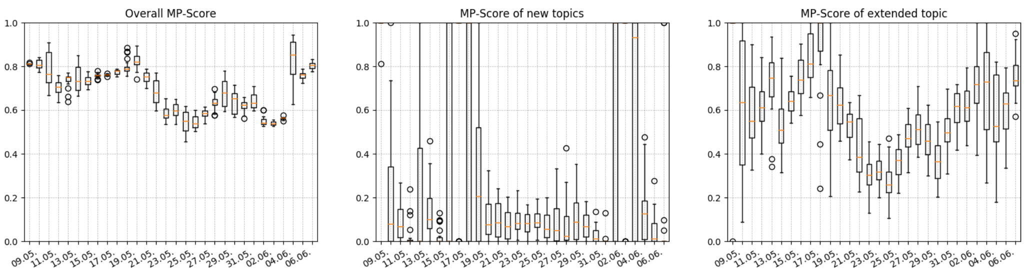

Figure 13: MP-Scores for clusterings using batch size of 1000

Looking at an increased batch size of 5000 in Figure 15, 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.

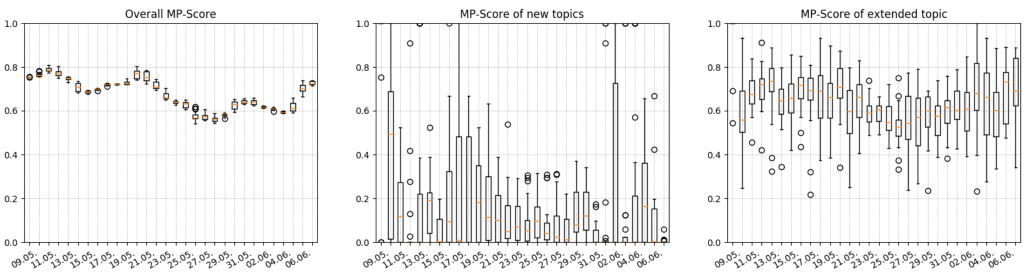

Figure 14: MP-Scores for clusterings using batch size of 5000

One of the reasons for the difference in the accuracy of the detection of new events and the extension of events can 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.

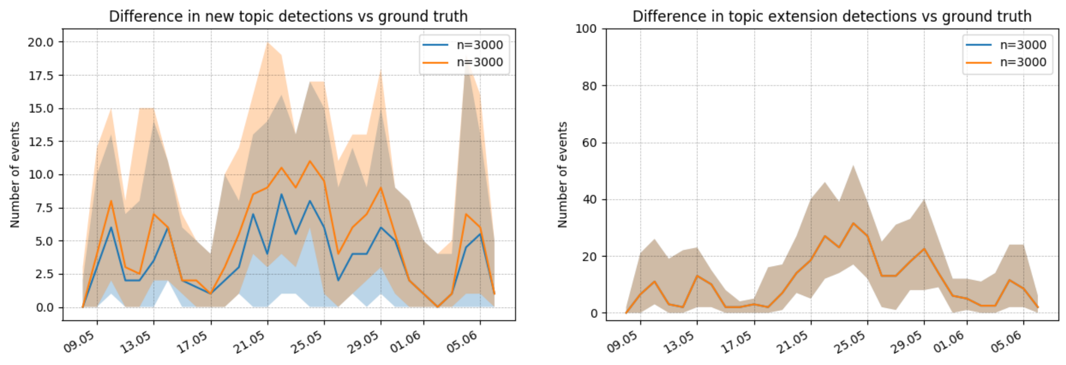

Figure 15: Differences in predictions vs ground truth using batch size of 3000.

Additional reasons for the general low performance 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. The difference in the number of news articles in the overall clustering compared to the total number of news articles, which are part of an event in a single batch.

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.

TODO

4.2.3 Conclusion

While the overall clustering results in good scores, the detection of new events and changes in existing events gives rather poor results. Possible reasons have been explored such as the *min_cluster_size*, the noise rate or the general difference in the number of news articles, but there exist no simple solutions for any of them. 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. TODO elaborate a bit more

5 Conclusion

5.1 Lessons learned

Todo.

5.2 Future work

How can this work be improved further?

- Improving space complexity and noise ratio of HDBSCAN.
- Alternatively, use HDBSCAN as an approximation for the number of clusters and a different clustering algorithm for the actual creation of the clusters.

6 Index

6.1 Bibliography

[1] Ozer Ozdikis, Pinar KARAGOZ, and Halit Oğuztüzün. "Incremental clustering with vector expansion for online event detection in microblogs". In: *Social Network Analysis and Mining* 7 (Dec. 2017). DOI: 10.1007/s13278-017-0476-8.

[2] Farzindar Atefeh and Wael Khreich. "A Survey of Techniques for Event Detection in Twitter". In: *Computational Intelligence* 31.1 (2015), pp. 132–164. DOI: 10.1111/coin.12017. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/coin.12017. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/coin.12017.

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

[4] Seo and; Sycara. "Text Clustering for Topic Detection". In: (Jan. 2004).

[5] Ilias Gialampoukidis, Stefanos Vrochidis, and Ioannis Kompatsiaris. "A Hybrid Framework for News Clustering Based on the DBSCAN-Martingale and LDA". In: vol. 9729. Jan. 2016, pp. 170–184. ISBN: 978-3-319-41919-0. DOI: 10.1007/978-3-319-41920-6_13.

[6] Leland McInnes, John Healy, and Steve Astels. "hdbscan: Hierarchical density based clustering". In: *The Journal of Open Source Software* 2.11 (Mar. 2017). DOI: 10.21105/joss.00205. URL: https://doi.org/10.21105%2Fjoss.00205.

[7] Junjie Wu, Hui Xiong, and Jian Chen. "Adapting the Right Measures for K-means Clustering". In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '09. Paris, France: ACM, 2009, pp. 877–886. ISBN: 978-1-60558-495-9. DOI: 10.1145/1557019.1557115. URL: http://doi.acm.org/10.1145/1557019.1557115.

[8] Alexander J Gates et al. "On comparing clusterings: an element-centric framework unifies overlaps and hierarchy". In: *arXiv preprint arXiv:1706.06136* (2017).

[9] Julie Beth Lovins. "Development of a Stemming Algorithm". In: *Mechanical Translation and Computational Linguistics* 11.1–2 (June 1968), pp. 22–31. URL: http://www.mt-archive.info/MT-1968-Lovins.pdf.

[10] C.J. van Rijsbergen, S.E. Robertson, and M.F. Porter. "New models in probabilistic information retrieval". In: (1980). URL: https://tartarus.org/martin/PorterStemmer/def.txt.

[11] Martin F. Porter. *The English (Porter2) stemming algorithm*. Sept. 2002. URL: http://snowball.tartarus.org/algorithms/english/stemmer.html.

[12] Chris D. Paice. "Another Stemmer". In: *SIGIR Forum* 24.3 (1990), pp. 56–61. DOI: 10.1145/101306.101310. URL: https://doi.org/10.1145/101306.101310.

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

6.2 Glossary

api An application programming interface allowing access to data or features of an application. 23

clustering The task of grouping a set of objects based on their similarity. 23

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

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

vectorizer A vectorizer transform a text into a numeric vector. 23

6.3 List of Figures

6.4 List of Tables

7 Appendix

7.1 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()
    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 1: Calculate the MP-Score between two clusterings.