Zurich University
of Applied Sciences

School of
Engineering
InIT Institute of Applied
Information Technology

Bachelor thesis Computer Science

Dynamic Event Detection in Data Streams

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

Zurich University
of Applied Sciences

School of Engineering

DECLARATION OF ORIGINALITY

Bachelor's Thesis at the School of Engineering

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

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

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

City, Date: Signature:

....

.....

.....

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

Zurich University
of Applied Sciences

Abstract

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 dataset 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

Why do we do this? Who might benefit from our work?

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 microblogs[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

3.1 Text preprocessing

3.1.1 Keyterm extraction

Todo

3.1.2 Entity extraction

Todo

3.1.3 Text lemmatization

Todo

3.1.4 Text stemming

Todo

3.2 Data Representation

3.2.1 Word Frequency

3.2.2 tf-idf

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 araises to find an appropriate clustering method, which is able to work with data of varying densities and of high dimensionality.

TODO why hdbscan

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 dataset.

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

- Very simple and easy to understand algorithm

- 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's able to sort data into clusters of different shapes.

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.

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

- 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 extension of DBSCAN that converts DBSCAN into a hierarchical clustering algorithm. Therefore it only requires one parameter to be set: 1. minPoints - The number of data points a neighbourhood has to contain in order to be considered as a cluster.

1. Transform the space according to the density/sparsity. 2. Build the minimum spanning tree of the distance weighted graph. 3. Construct a cluster hierarchy of connected components. 4. Condense the cluster hierarchy based on minimum cluster size. 5. Extract the stable clusters from the condensed tree.

One reason why HDBSCAN works so well, is that it is aware of noise. This is very important as real life data is often messy and sometimes even corrupt.

Because of this, the very first thing HDBSCAN does, is to remove such data noise. In order to identify noise, the **mutual reachability distance** score between points is calculated. The mutual reachability distance score is defined as

TODO: Formula

whereas corek(x) stands for the core distance for a point x and a defined parameter k.

TODO.

4 Methodology

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

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 1: Evaluated data set candidates ordered by data set size.

We could extract events from all data sets mentioned in Table 1.
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 options, our choice was simple:
The News Aggregator Dataset not only provided more data but our work built on the news articles use case could be continued by using it on different data sets, e.g. the GDELT 2.0 data set.

4.2.2 Preprocessing

4.3 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 a 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 2, where K-means achieved a rather high score, regardless of the significant difference between the true amount of clusters and the approximation using $\sqrt{n}$.

TODO add number of estimated clusters

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

Table 2: K-Means has a higher NMI score than HDBSCAN, while having a much larger difference in number of clusters.

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 and the Jaccard index.

TODO find citations

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

$$\frac{|A \cap B|}{|A \cup B|} \tag{1}$$

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 1, 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:

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

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. The full algorithm for the selection process can be found in the appendix as listing 1.

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

$$w_{ij} = \frac{|t_i| + |c_j|}{|T| + |C|} \tag{2}$$

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

$$\text{MP-Score} = \sum_{i=0}^{|S|} w_i s_i \{w_i \in W \wedge s_i \in S\} \tag{3}$$

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:

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

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 3 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: $C = [1, 1, 1, 2, 2, 2, 2, 3, 3]$.

As a final note, repeating the evaluation shown in table 2 a second time using the MP-Score, the score (Table 4) 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.

4.3.1 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.

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 3: Direct comparison of different scoring functions

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

- 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[9]. 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 running the final evaluations, without worrying about setting up and installing all dependencies again.

4.4 Online Clustering

Detecting events in a stream of news articles will be achieved by using an online clustering approach. The events of interest for this application are the discovery of new topics and the extension of existing topics. Thus we define our two events as follows:

- Topic added: A new cluster of news articles appears in the data stream.
- Topic extended: An existing topic 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 datasets, 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.

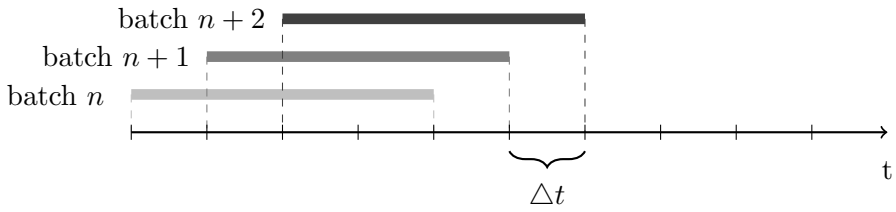

Figure 1: Timeline showing the sliding window approach

Finding pairs clusters 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 topic. 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 1. If the similarity is above a certain threshold, both clusters are seen as describing the same topic.

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.

4.4.1 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 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.

Comparing clusters To detect events between batches of clusterings, we have find pairs of clusters describing the same topic. 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)[10] to find similar clusters. This reduces the time complexity to $O(log(n))$. The implementation for LSH is provided by the datasketch library[11]. TODO explain LSH?

Detecting Events Once we have found pairs of clusters, which represent the same topic, 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 the event *topic extended* for a specific topic. Clusters from the new batch without a matching cluster from the previous batch are seen as new topics and represented by the event *topic added*.

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 approach will then be used for the online clustering 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 **??** and evaluate each. The methods are:

- Full text with stop word removal
- Keyterms
- Named Entities
- Lemmatization
- Stemmatization

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 dataset. 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 dataset. 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.

5.1.2 Evaluation

The first run is done with 60 stories, which results in approximately 2000 news articles, over five repetitions. Table 5 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 accuracies 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 2, 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	Keyterms	Entities	Lemmatized	Stemmed	Full Text	Keyterms	Entities	Lemmatized	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 5: 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, lemmatization appears to provide the highest accuray in general or at least being fairly close to the highest score. This is to be expected, since lemmatization reduces the dimensions by grouping words into their base form, while still retaining most of the text. In contrast to keyterm 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 keyterm 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 datasets. Figure 3 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.

TODO explain why cosine is generally better than euclidean TODO explain min cluster sizes, but run with lemmatization

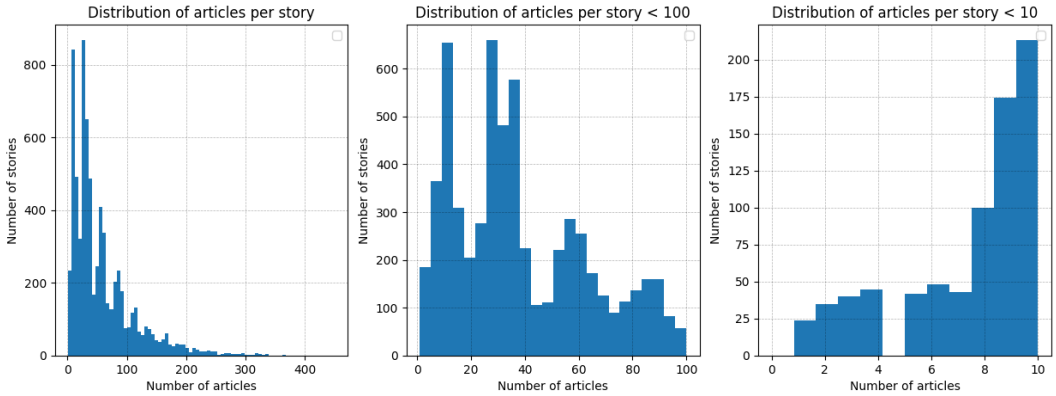

Figure 2: Distribution of cluster sizes.

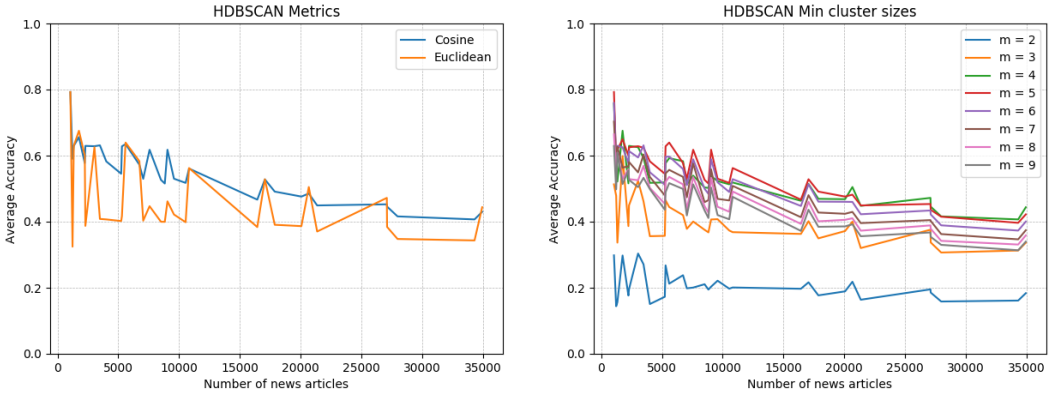

Figure 3: Accuracies for different parameters

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 4 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. Nontheless 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.

TODO calculate expected noise ratio based our min cluster sizes.

Having found the optimal settings to run HDBSCAN with, we can start comparing the overall performance with K-Means. Figure 5 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 5. However the accuracy seems to stabilize around the 0.7 mark.

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 6. Al-

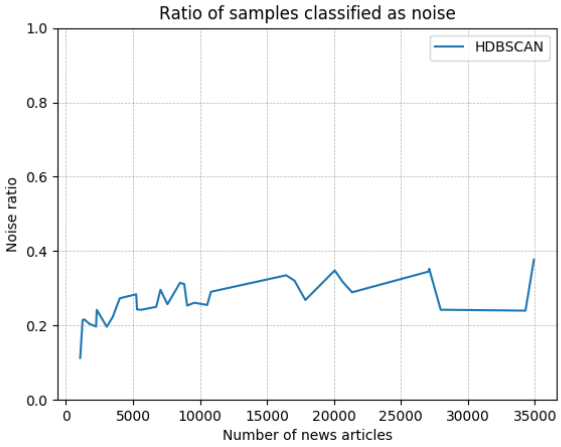

Figure 4: Number of news articles classified as noise.

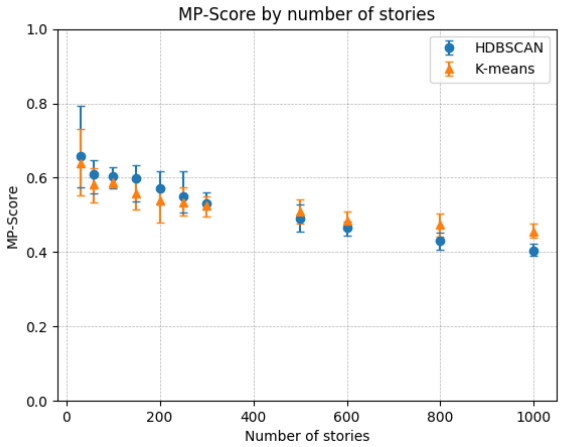

Figure 5: Comparison of the average accuracy between K-means and HDBSCAN

though 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 7 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 8. 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.

5.1.3 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

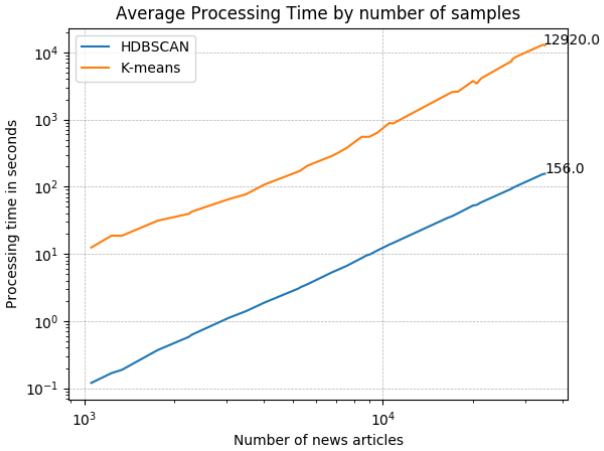

Figure 6: Processing time in seconds

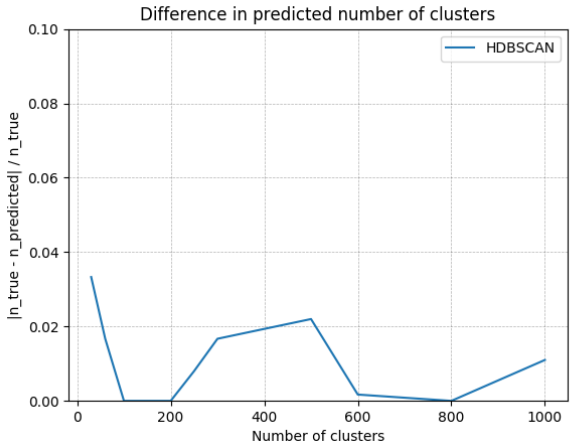

Figure 7: Ratio of difference over predicted with true number of clusters

the most accurate results for our dataset. On the flip side the noise ratio is quite high and the space complexity is problematic with larger datasets. Overall HDBSCAN provides an acceptable accuracy, while still leaving room for further improvements.

5.2 Online clustering

5.2.1 Setup

Start date 2014-05-08 00:00:00 End date 2014-06-06 23:00:00

Number of days 30

5.2.2 Evaluation

TODO evaluate different thresholds

TODO show example with a specific topic e.g. when it first is detected, incoming news articles etc.

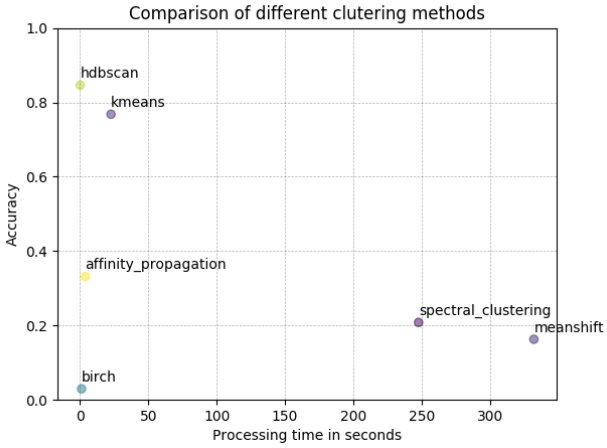

Figure 8: Comparison of different clustering methods with a sample size of approximately 1000 news articles

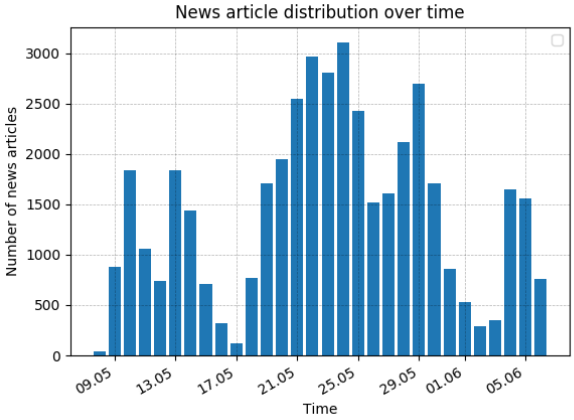

Figure 9: Incoming news articles over 30 days

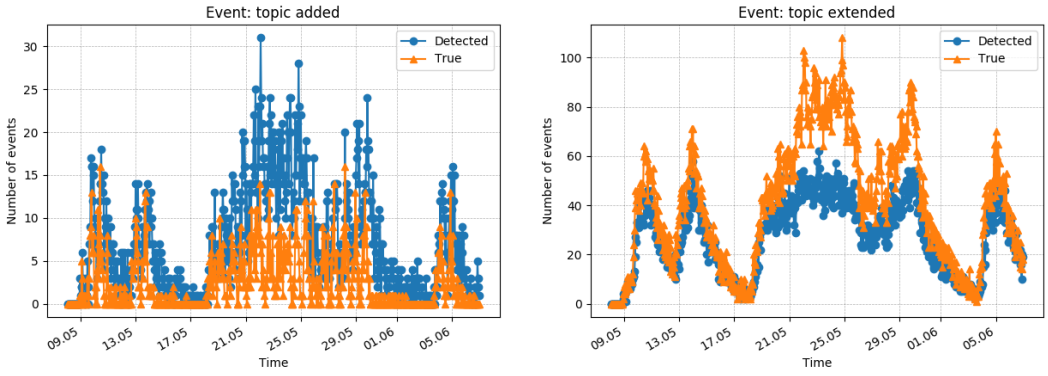

Figure 10: Comparison between detected and true events with batch size of 2000 samples

Figure 11: Plot work in progress

6 Conclusion

6.1 Lessons learned

Todo.

6.2 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 algorighm for the acutal creation of the clusters.

7 Index

7.1 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] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[10] Alexandr Andoni et al. *Practical and Optimal LSH for Angular Distance*. 2015. arXiv: 1509.02897 [cs.DS].

[11] Eric Zhu and Vadim Markovtsev. *ekzhu/datasketch: First stable release*. Feb. 2017. DOI: 10.5281/zenodo.290602. URL: https://doi.org/10.5281/zenodo.290602.

7.2 Glossary

api An application programming interface allowing access to data or features of an application. 25

clustering The task of grouping a set of objects based on their similarity. 25

docker A tool to package the application with all its dependencies as a single deployable unit and run it on any host with docker installed. 25

dockerized An application environment running as a single or a collection of docker containers. 25

vectorizer A vectorizer transform a text into a numeric vector. 25

7.3 List of Figures

7.4 List of Tables

List of Symbols (draft)

Symbol	Description
$\mathbb{N}$	\symnz — Menge aller natuerlichen Zahlen ohne die Null (no)
$\mathbb{N}_0$	\symnzmn — Menge aller natuerlichen Zahlen einschliesslich Null (no)
$\mathbb{Z}$	\GZ — Menge aller ganzen Zahlen (no)
$\mathbb{Q}$	\RatZ — Menge aller rationalen Zahlen (no)
$\mathbb{R}$	\RZ — Menge aller reellen Zahlen (no)
A	\AB — Aufrechter Buchstabe (no)

8 Appendix

8.1 Algorithm for Accuracy Selection

```python
    def select_max_values(self, accuracy_matrix):
        unique_indices = dict()
        row_index = 0
        nrows = len(accuracy_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(accuracy_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
```

Listing 1: Select relevant accuracy values from a accuracy matrix.