# Document Clustering using Tweets - Final Report

Marius Kempf, Simon Richebaecher, Daniel Helmrich

2019-11-17

# Contents

# 1 Introduction

The internet has become a platform that most people use everyday throughout a significant period of time. On websites, social networks, blogs, online shops, rating portals and much more, large amounts of data in the form of written text are created every day. These texts are usually written by real people and contain information about their opinion or mood towards certain things. This can be opinions about companies, politicians, celebrities, books, hotels, events or new technologies. However, in order to analyse large amounts of data techniques are needed that can carry out an analysis automatically. This is particularly relevant in the context of text data from the internet, due to the almost unlimited quantity. The area for analyzing data in text form is called text mining. One specific part of it is clustering or classification of text files depending on their content. This allows, for example, the analysis of the sentiment or opinion of people. As these large amounts of data from the internet are usually not labelled, unsupervised learning procedures like clustering are playing an important role for analyzing. Furthermore text data can not be processed by the corresponding algorithms in its pure form. Numerical methods need numerical input, thus a different form of representation is needed. The common approach is to use vector representations generated by proceedings called Word2Vector or Document2Vector. Moreover written text is usually not homogeneous nor consistent. Therefore some preparation steps are needed.

As part of a clustering project a text mining task is addressed: An unsupervised clustering analysis is applied on a big data set containing tweets from Twitter. In addition, the data set contains labels that correspond to the sentiment within each tweet. Thereby a post-clustering comparison and analysis of the labels with the found clusters is applied. The necessary steps from preprocessing to the final clustering results and their evaluation are implemented according to the given methodology. With the help of clustering the expectation is to gain insights about the data and identify distinct clusters. By testing different clustering parameters and evaluations metrics, it is aimed to find an as optimal as possible implementation for initiating sentiment analysis for text data - more specific microblogging context like tweets.

In the following, unsupervised classification and its related types are described. Furthermore the objective of this work is explained as well as the used data set. Chapter 2 contains a short overview of the project design. Afterwards the data preprocessing and the implemented clustering algorithm are explained in detail within chapter 3. Chapter 4 describes our evaluation concept and analyzes its results. Chapter 5 provides the conclusion, findings and lessons learnt during the project.

### 1.1 Definition: Unsupervised classification

Generally numerous machine learning algorithms can be distinguished. Since a clustering algorithm is applied, this work focuses on two main approaches which are shortly explained in the following: Supervised and unsupervised classification or learning. Supervised classification has the objective of finding a way to identify combinations of instance characteristics to assign an observation to one of the given classes. For that the correct or real class must be given as a label for each instance. Given that label, the algorithm can "learn" how to decide, based of the corresponding features if a newly instance belongs to a class or not. In contrast to that, unsupervised classification, which is also called clustering, does not use any labels or target classes. Rather, it is based on finding patterns within the unlabeled data from which an cluster assignment of instances can be derived Witten et al. (2016) Bandyopadhyay & Saha (2012*a*). A more formal description is as follows: Assuming is a set of n observations, where $\mathrm{X} = (x_1, ..., x_n)$ is a set of $n$ observations, where $x_i \in \chi$ for all $i \in [n] := \{1, ..., n\}$. Usually it is assumed that the observations are pulled independently and identically distributed from a distribution $\chi$. Typically a matrix $\mathrm{X} = (x_i^T)_{i\in[n]}^T$ is defined which contains the data points as its rows. Unsupervised learning has the goal find interesting structures and patterns in $X$ Chapelle et al. (2009).

One major challenge in applying clustering algorithm is the evaluation: Since there is no clear definition of true or false prediction, it can be difficult to decide if the model performance is good or not. The application of a metric depends significantly on the type of domain the clustering algorithm is applied on. For that reason there are different metrics which have to be taken into account for evaluation Bandyopadhyay & Saha (2012*b*). The metrics used in this work will be described in more detail in chapter 4. Furthermore the decision how many clusters should be derived is challenging. Especially when it is not known or at least roughly foreseeable, which means it is not possible to make at least an estimation about how many clusters are hidden within the data. In Duda et al. (2001) the authors describe five basic reasons why unsupervised learning can be interesting and useful. First, labeled data can be very costly. Collecting and labeling data, for instance spoken speeches, is very time-consuming and needs a lot of effort. A classifier can be trained in a crudely way on a small sample of labeled data and afterwords "tuned up" by learning unsupervised on the larger part of the unlabeled part of the data. This approach would save a lot of effort. The second point mentioned, describes the same procedure in the reverse direction: An large "cheap" unlabeled data set can be used to initially train a first model. The smaller and labeled data set is used to label the found groupings or clusters in a supervised way. This approach can be useful for large data mining applications such as problems where a lot of data with previously unknown content is investigated. Third, characteristics of the pattern within data can slowly change over time. This pattern shifts need to be tracked by a classifier to improve the model performance. Furthermore unsupervised learning methods can be used to find certain "features" which might be useful for a following categorization. One can think of methods to provide some kind of data-dependent "smart-preprocessing" or "smart features extraction". As fifth unsupervised learning or clustering can help to gain first insights about the previously unknown data. For instance unexpected

patterns or sub classes could be found. This can help to rearrange assumptions made for the model designing Duda et al. (2001).

In addition to the two mentioned types there is another very interesting one: Semi-supervised learning (SSL). SSL can be classified halfway between supervised and unsupervised learning. Additionally to unlabeled data, the algorithms are using some supervision information. This is often implemented by using a partly labeled data set where only a small subset of observations is labeled. There are many forms of partial supervision approaches available which can be found in the provided literature Chapelle et al. (2009). For instance, a literature survey on semi-supervised learning is provided by Zhu (2008).

## 1.2 Objective

In the presented work a clustering algorithm is applied to a data set containing tweets to probably identify different clusters within these. A k-means clustering algorithm is used to identify distinct sentiment clusters within the used Twitter data.

To train and test our k-means clustering, the widely known data set Sentiment140[1] is used. It contains 1.6 million tweets collected between April 2009 and June 2009. The tweets are binary labeled with positive and negative sentiment. Collecting and labelling the data is done by the researchers Go et al. (2009). Their approach was to use emoticons in the collected tweets to derive a noisy label. In practical terms, this means that tweets containing smiling emoticons were labeled with positive and those containing sad emoticons were labeled with negative sentiment. Every entry or tweet contains the following variables Go et al. (2009):

1. target: the polarity of the tweet (0 = negative, 4 = positive)
2. id: the id of the tweet (2087)
3. date: the date of the tweet (Sat May 16 23:58:44 UTC 2009)
4. flag: the query (lyx) - If there is no query, then this value is NO_QUERY.
5. user: the user that tweeted (robotickilldozr)
6. text: the text of the tweet („what is the answer to the ultimate question of life the universe and everything“)

Since a clustering algorithm is applied on text data, only the actual tweet text is used and further processed. Which means the clustering decision depends only on this single attribute.

As most machine learning models or algorithms work with numerical data, a transformation from written text to a numerical representation is needed. There are multiple approaches which can be applied. Probably the most basic ones are predefined libraries were words are saved with some

[1] https://www.kaggle.com/kazanova/sentiment140

additional information assigned. More advanced approaches are the so called Word2Vector (Word2Vec) or Document2Vector (Doc2Vec) models. These are unsupervised learning procedures which learn how words can be optimally represented by vectors. The great advantage is that these models learn the context and meaning of the words during the training. As a result, this knowledge is stored within the derived word vectors. Since Document2Vector is based on the idea of Word2Vec this must be introduced first.

Word2Vec is a computationally efficient prediction model to derive numerical vector representations from raw text data. Learning these vector representations itself is unsupervised. Thus the model is generated without external input some form of a label. This means huge amounts of text data can be used to train such a model. The concept Word2Vec does not represent an own algorithm itself. A distinction can be made between two different architectural models: “Continuous Bag of Words” (CBWO) and “Skip-Gram”. CBOW predicts a word based on its context. Skip-gram models work in the opposite direction and predict the context based on a word. Assuming the sentence “We love programming” serves as an example, then CBOW would predict the word “we love” based on the words “we” and “programming”. Skip-gram predicts with high probability the other two words based on “love”. The model is trained using a neural network by adjusting the model weights in form of vectors. For a deeper theoretical background, reference is made to the work of the approach‘s developers Mikolov et al. (2013).

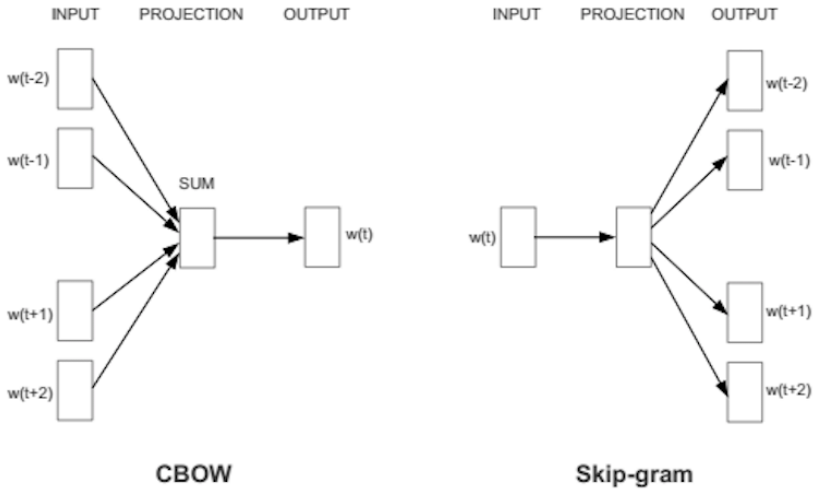


**Figure 1:** Word2Vector model architectures. The CBOW architecture predicts the current word based on the given context, and the Skip-gram predicts surrounding words given the current word Mikolov et al. (2013)

Doc2Vec is based on the introduced Word2Vec approach. The main difference is that Doc2Vec trains vectors for the whole document and the individual words contained in the document. Additionally to the word vectors, a unique vector is created for the respective document or in this case for the whole tweet. This vector is represented in the following illustration by the color orange and the name “Paragraph Id.”

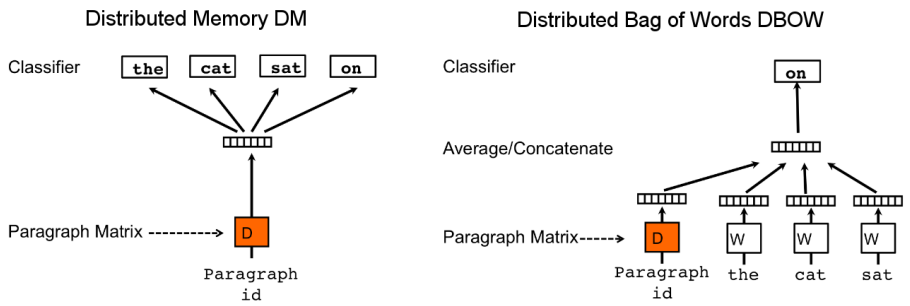


**Figure 2:** Document2Vector model architectures. The DM architectures uses three word and the additional paragraph vectors to predict the fourth word. In the DBOW version the paragraph vector is used to predict the corresponding words Le & Mikolov (2014)

While the approach “Distributed Bag of Words” (DBOW) can be assigned to the principle of the Skip-Gram model in Word2Vec, “Distributed Memory” (DM) corresponds to the CBOW model in Word2Vec. DBOW considers the document vector as well as a random word vector of a document and determines probabilities for the further words of the document. The DM model determines a missing word from the context of the surrounding word vectors and the document vector. The vectors are either concatenated (Distributed Memory Concatenated - DM-C) or the average values are used (Distributed Memory Mean - DM-M). For a more in-depth theoretical understanding, reference is made to the papers of Le & Mikolov (2014) and Rong (2014). The decision to use Document2Vector for the current work is based on the fact that it is the more advanced approach and generally outperforming Word2Vec according to Lau & Baldwin (2016).

## 2 Project design and structure

The software developed for this project is available in a Github repository [2]. It allows to:

- Pre-process the data, which includes cleaning and splitting the tweets
- Training a Doc2Vec model for a partition of the tweets and applying the other part on that model
- Cluster the Doc2Vec vectors with the k-means algorithm
- Evaluate the k-means clustering results with different metrics, including a cluster-to-class comparison with the sentiment label that is included in the raw data set
- Introduce new tweets to a k-means clustering result and output the closest cluster including sample tweets from that cluster
- Generate word clouds and a HTML visualization of the clustering results

All installation and usage instructions can be found in the README.

For the clustering, the cloud computing service Google Cloud [3] was used. The rented server offered 4 CPUs with 2.2 Ghz per core.

[2] https://github.com/dhelmr/data-mining-upv
[3] https://cloud.google.com/

# 3 Implementation details

The following section shows and explains the most relevant part of the implemented software used for the clustering. These steps are Data Preparation, Doc2Vec implementation and the implemented k-means algorithm. For the latter, important parts of the algorithm are described. Additionally its complexity in terms of space consumption and runtime is discussed, as well as further optimizations and our test concept.

## 3.1 Data preparation

The original data set contains 1.6 million tweets which are at first splitted into two sets. The large data set contains 70 % of the tweets and is used for training the Doc2Vec model. The second set, containing 30 % of the tweets, serves as base for the application of a clustering algorithm. This division ensures the independence between vector representation model and clustering which is important for the validity. Although Doc2Vec is an unsupervised procedure it might occur that there is a small bias if all of the available data is used. Therefore this division is carried out. Due to limited resources and the general conditions within an university project, an extension of the data used for building the Doc2Vec model and the corresponding vocabulary is not performed. In general, it is possible to use theoretically as many texts or documents as possible from whatever source and domain as base for the word embedding learning. In a further step, the data set containing 30 % of available data is divided a second time. The bigger part is used for training the clustering algorithm while the smaller set, containing 100.000 samples, serves as test data set. These are used so simulate newly incoming text which are to be assigned to the previously found clusters. The data splitting is visualized by the figure below.

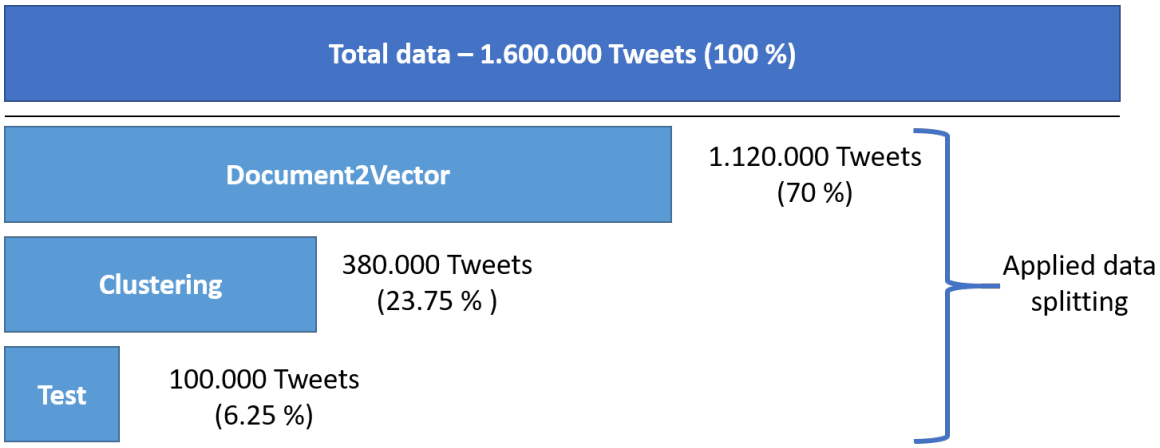


**Figure 3:** Caption: Applied data splitting on the 1.6 Million available tweets. Three sets are derived: Doc2Vec Training, Clustering Algorithm, Testing

To apply a meaningful clustering of text documents it has to be taken into account how twitter posts look like. After checking the raw data it is obvious that tweets also contain content which is not useful or meaningful in terms of understanding what it is about. For that reason the texts need to be cleaned before further processing. All data sets go through the same data cleaning process which is explained in the following: First a data frame is loaded and since the positively labelled tweets are labeled with "4", this is changed to "1". As a matter of fact, these labels are usually not necessary for an unsupervised procedure. Since the found cluster will be compared with the available labels later on, they will still be kept. Afterwards every tweet goes through these steps: lowering the words, removing user names, removing web-addresses, removing, expanding contractions, word stemming, stopwords removal.

```
tweets <- raw_data
for tweet in tweets:
    tweet.lowercase()
    tweet.remove(user names)
    tweet.remove(website links)
    tweet.remove()
    tweet.expand_contractions()
    tweet.remove(everything except letters, i.e. '!', '.', '^', ...)
    tweet.stopword_removal()
    tweet.word_stemming()
    tweet.remove_large_whitespaces()

data_clean <- tweets_cleaned, label
data_clean.drop_empty_entries()

data_clean.save_to_file()
```

### 3.2 Document2Vector implementation

Based on the work of Lau & Baldwin (2016), a hyper-parameter combination for the Doc2Vec model is selected. The authors derived the following optimal parameter for their task: A vector size of 300, window size of five, minimum count between one and five, negative sample of five and training epochs between 600 and 1.000. The authors used an approach and a data set prepared by Hoogeveen et al. (2015). This data set contains questions and answers from Stackexchange forums. The question used as task for the performance evaluation is, if an asked question is a duplicate within the corresponding sub-forums or not. Assuming forum questions have a similar length like tweets, the hyper parameter selection provides a valid basis for the current question in this work. Accordingly, the final model is also trained for 600 epochs. Since the hardware used for the current university project is limited, the size of the vectors is limited to 200 to avoid occurring memory problems. The window size and negative size is the same like in the work of Lau & Baldwin (2016). The minimum count is selected to two.

## 3.3 k-means algorithm

In this section applies the following convention:

**instance** refers to individual Doc2Vec representation of a single tweet

**k** number of clusters

**n** dimension of each instance (i.e. the number of features)

**s** number of instances that are chosen for the clustering

**m** parameter for the Minkowski distance

### 3.3.1 General overview

The k-means clustering was first implemented following the algorithm of the lecture, but offering some additional parameters. Later it was enhanced with checks for skipping certain parts of the algorithm in order to reach a performance gain. The python implementation of the algorithm can be found in the file `kmeans/k_means.py`.

The following parameter allow the algorithm to be more flexible than in its classical version:

- A variable distance metric by implementing the Minkowski distance [4] that can be influenced with the parameter m. The two special cases `m=2` and `m=1` yield the Euclidean and Manhattan distances, respectively.
- Two different strategies for the initialization of the cluster distances:
    1. For the first strategy k random instances will be chosen from the set of instances and their coordinates are used as the centroids.
    2. The second strategy (`DOUBLE_K`) will first run the k-means algorithm with the same parameters, but with 2k clusters that are initialized using strategy one. Afterwards for each cluster the mean of the squared distances between its instances and the centroid is calculated and used as a quality measurement for that cluster. The k best centroids (i.e. the ones with the lowest of these intra-cluster distance means) are finally chosen as the initialization points of the actual k-means run, which is executed thereafter.
- Offering the possibility to run the k-means algorithm multiple times with the same parameters and choosing the best result afterwards. The specific results are compared by their squared summed error. This is especially useful, if the random initialization strategy is used to omit the results of poor initialization.

[4] https://en.wikipedia.org/wiki/Minkowski_distance

- Specifying different abort criteria, as specified further below.

The principal part of the k-means algorithm is the main loop. Each run-through of one loop step contains:

1. For each centroid, determine the nearest other centroid. This is used for the skip checks, as described below.
2. Assigning each instance to the closest cluster.
3. Recalculating the cluster centers (centroids) by computing the mean of all instances that belong to one cluster.
4. Updating additional data structures that are used for the skip checks.

At the end of each loop step, the algorithm checks if one of the following abort criteria is reached:

**Fixed number of iterations** This criterion can simply be checked by comparing the current iteration with the parameter `max_iter`, which specifies an upper bound.

**Threshold** If the aggregated changes of centroids fall below a certain value, which can be specified as a parameter, the loop will be aborted. For that, the distance of each centroid change is calculated and the maximum of these values is chosen ( `max_centroids_change`). As this distance is also needed for the Hammerly skip-check in each iteration (described below), it will be calculated for each centroid that has changed. Otherwise it would be possible to implement a further check for skipping the distance calculation in the case that `threshold < max_centroids_change` holds, because then the abort criterion `threshold < max_centroids_change` already would not be satisfied.

**Convergence** As the classical abort criterion, convergence will also terminate the loop. It is detected if there is neither a change of the centroids nor of the cluster membership assignments.

The algorithm stores the following basic data structures:

- A mapping of each instance to the assigned cluster index
- The inverse mapping, containing for each cluster the set of its member instances
- The coordinates of the instances and the centroids

To increase the efficiency of the implementation, the algorithm was extended by using the methods described by Hammerly Hamerly (2010): There are two additional conditions added before re-assigning the cluster memberships. These conditions check quickly if the instance still belongs to the same cluster. In that case, the inner-most loop can be skipped, which contains the computation for determining the nearest centroid.

To achieve that, there are additional data structures implemented in the the k-means algorithm:

- For each cluster the distance by which it was moved lastly. This does not necessarily refer to the last k-means iteration, but instead to the last change in general, that was distinct to zero. (Space: `O(k)`)
- For each centroid the distance to the closest other centroid. (Space: `O(k)`)
- For each instance an upper bound on the distance to the assigned centroid. (Space: `O(s)`)
- For each instance a lower bound to the second closest centroid. `O(s)`)

The bounds are be updated in each iteration of the algorithm. The main idea is to exploit two possible cases:

1. **The upper bound $u(A)$ to the closest center is smaller than the lower bound to the second closest center** $l(A)$ In this case it is safe to say that the cluster assignment cannot be changed: If $u(A) < l(A)$, $distance(A, C1) \leq u(A)$ and $l(A) \leq distance(A, C2)$ hold, then $distance(A, C1) < distance(A, C2)$ can easily be deduced.

   An example can be seen in Figure 4. Supposing that the bounds were be initialized with $l(A) = g$ and $u(A) = h$, the closest centroid C1 was moved away from the data point A, but the second closest centroid C2 moved in direction of A. After these changes, the upper and lower bounds are recalculated respectively: The upper bound on the distance to the closest center ($C1$) is increased with $u(A) := u(A) + P_{C1}$, whereas the lower bound to $C2$ gets reduced with $l(A) := l(A) - P_{C2}$. As the red circle indicates, the distance from A to $C1$ is still lower than from A to $C2$.

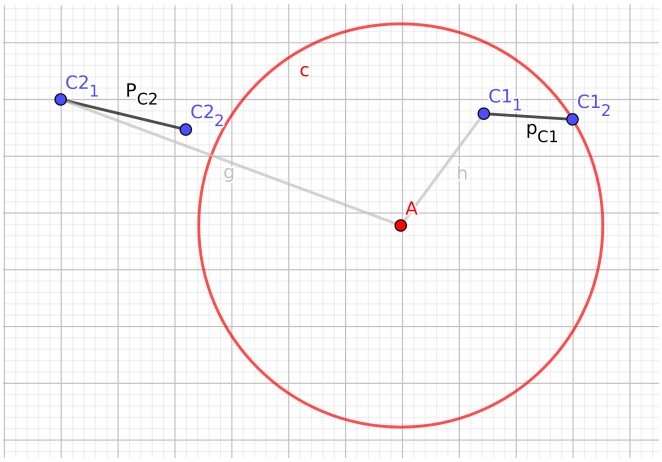


**Figure 4:** Example of the first case. A is the instance point, C1 and C2 are centroids. The lower indices specify to which iteration the points belong. For example, the second centroid was moved by $P_{C2}$ from $C2_1$ to $C2_2$

.

2. $u(A)$ **is lower than the distance** $s(C)$ **from that centroid to the closest other centroid, divided by 2.**

   In that case, it is safe to say that A is within the circle shown in figure 5. As $s_1$ is the smallest distance from $C_1$ to any other centroid, the closest centroid from $A$ must be $C_1$.

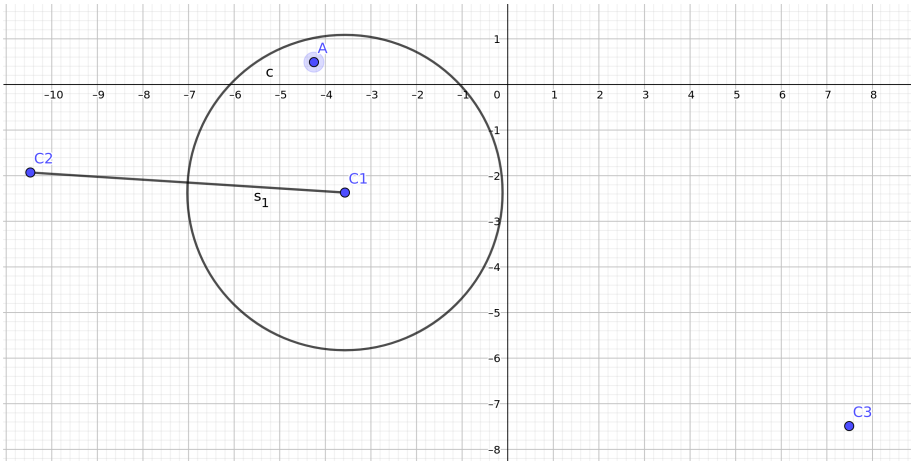


**Figure 5:** Example of the second case. A is the instance point, C1, C2 and C3 are centroids. The circle has radius $s_1/2$

If one of these cases occur, the membership reassignments, which consists in calculating the distance to all other centroids, can be skipped.

### 3.3.2 Complexity

Assuming the worst case, i.e. no skip as described above can be made and the algorithm runs until the maximum number of iterations, max_iter, is reached, the runtime complexity of the algorithm is within $\theta(\text{max\_iter} \times (k^2n + skn + kn + s + k))$. That means that it has quadratic complexity for k, and linear complexity for all other parameters. The specific parts of the algorithm contribute as follows:

**The main loop** It is executed at most *max_iter* times in the worst case. In practice, it could terminate earlier when convergence is detected or the threshold for the centroid changes is reached.

**Update of the centroid distances** Each centroid must be compared to all other centroids, and for each centroid-pair the Minkowski distance must be calculated which is dependent from the size of $n$ (the number of features). Thus, the complexity for this part is within $\theta(k^2n)$.

**Cluster membership reassignments** It iterates of all instances (length s), and determines for each instance the closest and second closest centroid ($\theta(2kn)$). By means of the Hamerly skips (see above), which are determined in constant time, the number of iterations in which these centroids

have to be calculated, can be reduced significantly. The average runtime would therefore be within $\theta(2s'kn) = \theta(s'kn)$, where $0 < s' < s$ and $(s - s')$ is the number of times the centroid calculation is skipped.

**Update of the centroid positions** Here, it must be iterated over all instances, grouped by each cluster, in order to calculate the new centroid. For calculating the distance between the old and new position, and for checking if the old and new position are the same, the runtime is within $\theta(n)$, each time. This yields an overall time complexity within $\theta(2kn + s) = \theta(kn + s)$.

**Updating the bound data structures** It is needed to iterate over the last changes of each cluster and over all instances to update their upper and lower bounds.

The Minkowski parameter m influences the computation of the distances. As it is often needed to compute distances between the centroids or instances or between two centroids, it is worth taking into account how it influences the computation time: For $m = 1$ (Manhattan distance), there is no need to compute roots or exponents which makes the computation faster. For higher values of m, the computational times depend on the python interpreter and the optimizations of numba (see below). Higher computation times were experienced for higher values of.

The space complexity of the algorithm is within $\Theta((k + s)n)$ as for each centroid and for each instance, n instances and a constant amount of additional data structures, like the upper and lower bounds to the closest centroids or the distances between centroids are stored.

### 3.3.3 Further optimizations

Apart from the optimizations achieved by implementing the faster k-means version of Hammerly, numba [5] is used for accelerating the computation at two points:

1. The calculation of the Minkowski distance
2. The calculation of the new cluster centroids

It does not change the algorithm by itself, but instead makes these computations faster by translating them to machine code. In practical terms, this optimization was very important because it made running the clustering feasible for large k and a larger dataset.

### 3.3.4 Testing concept

The correctness of the algorithm was tested by comparing the results to the k-means implementation of scikit-learn. For that, 1000 data points are randomly generated, and both initialization are fed

[5] https://numba.pydata.org/

with different k to both implementations. The centroids of the assigned cluster for each instance are compared subsequently.

# 4 Evaluation of experimental results

As unsupervised machine learning techniques try to uncover information about data which was previously unknown, they are of exploratory nature and require an iterative approach. Using clustering to partition data falls into this category and the evaluation of such is therefore accompanied by constant adjustments to the underlying model. The different results that stem from adjusting parameters like number of partitions, used metric and initialization method need to be judged by qualitative criteria. For clustering, those can either be of internal or external nature. While internal criteria utilise structure and information within the data itself, external criteria rely on given benchmarks which represent existing knowledge about the data (Rendón et al. (2011)).

The goal behind the information extraction and the general availability of information influence the decision between internal or external validation. In the case of this project the source provides a great amount of textual data, but no additional attributes except for the classification into positive and negative sentiment state. With such a lack of descriptive information, going for an internal validation is less value-adding, as it only depends on structural analysis. Having a k-means model that optimizes for compactness and separability within partitions has no informational value, if features are missing to describe or interpret the resulting clusters.
Therefore it was opt for an external validation approach, using the two given classes for orientation. The k-means clustering is optimized to partition the data in such a way, that the clusters are pure subdivisions of the external classes as much as possible. With such clusters the available knowledge about the data can be evaluated. An application that is designed to group tweets in such a manner will either reveal weaknesses in the classification or will extend the classification into subdivisions that might represent yet unknown attributes.

## 4.1 Iterative adjustment of clustering models

In order to find the clustering model with the best performance concerning the upcoming validation one has to test different parameter options. Chapter 3.3 introduced the available options for adjusting the k-means clustering. Combined with the different training possibilities of the Doc2Vec model (3.2), this leaves four variables to adjust: number of epochs for Doc2Vec, used metric (m), k-means initialization method, number of clusters (k). To allow for comparison between results, one model gets produced for each combination of the following options:

1. Epoch: 10, 30 and 600 (see 3.2)
2. Minkowski-Metric: (m=) 1, 2, 15
3. Initialization: (init=) 1 - random, 3 - best centroid
4. Number of Clusters: (k=) 1 to 10 regular and higher if needed

After generating those with the computational assistance of the cloud server, the result are saved and hierarchically ordered by the listed options. In order to save storage capacity the saved objects only carry information on the allocation between the index of instances and the cluster membership. The storage intensive instance vectors are stored in a single, unchanging '.vec' file. Those readily accessible clustering results and instance vectors reduce the run-time of the evaluation process enormously.

## 4.2 Clustering performance and evaluation strategy

Before applying common external validation indexes to clustering results, a test for general cluster tendency is executed. The underlying data set is the one used for training the k-means (see 3.1). Testing the affinity of a data set towards clustering can be done with the hopkins statistic. By quantifying the spatial homogeneity of a data set, this method ranks data with a concrete score. Data with a tendency towards uniform distribution has a score of < 0.5, whereas compartmentalised and spatially separated data goes towards 1. A python implementation of the hopkins evaluation which uses an inverted scale (see documentation pyclustertend.hopkins) is applied. The result of 0.249 for the training data therefore translates to a hopkins score of 0.75. It can therefore be assumed that the training data is generally suited for clustering. This allows the evaluation to move on an external validation of partitions generated by different k-means settings.

The main objective of cluster validation with given classes is to examine the agreement in the cluster vs. class comparison. The more purity of class labels can be found within the clusters, the better those represent additional subcategories within the external benchmark. Various indexes like Jaccard and Rand exist to quantify this degree of agreement. After looking into python implementations of such from sklearn, it is discovered that those are not suited for the given case. These implementations solely work with lists of labels and do not offer the necessary class vs cluster comparison based on membership of instances. Consequently a customized comparative evaluation is developed, based on the total number of instances in the intersection between class and cluster. Like a confusion matrix this also allows for transparency on the distribution of instances across all clusters (see figure 6).

| | class_0 | class_1 | match_percentage | cluster_weight |
|---|---|---|---|---|
| 0 | 2144.00000 | 1299.00000 | 0.62271 | 0.00912 |
| 1 | 88139.00000 | 51063.00000 | 0.63317 | 0.36880 |
| 2 | 1675.00000 | 1014.00000 | 0.62291 | 0.00712 |
| 3 | 63189.00000 | 36202.00000 | 0.63576 | 0.26333 |
| 4 | 22558.00000 | 13067.00000 | 0.63321 | 0.09438 |
| 5 | 7513.00000 | 4462.00000 | 0.62739 | 0.03173 |
| 6 | 422.00000 | 217.00000 | 0.66041 | 0.00169 |
| 7 | 1968.00000 | 1092.00000 | 0.64314 | 0.00811 |
| 8 | 44479.00000 | 26757.00000 | 0.62439 | 0.18873 |
| 9 | 6484.00000 | 3700.00000 | 0.63668 | 0.02698 |

**Figure 6:** Exemplary class vs cluster matrix (epoch=600, m=2, init=1, k=10)

Using this summary of membership intersections the percentage of agreement (concerning the representative class-label within a cluster) is calculated. Next the weighted average of all percentage values is calculated, accounting for the different cluster sizes. With the help of this indicator it is possible to test the partitions that result from different k-means settings. In order to visually aid the comparison of such, the percentile agreement is plotted against the different numbers of clusters (k). This helps in recognising possible points of "clustering satisfaction" from which on a higher k will not significantly improve agreement with the classes.

### 4.3 Cluster validation and strategy evaluation

Percentile agreement plots summarize the results from parameter adjustments of different k-means models (4.1). They are compared to find the models with the best class vs cluster agreement. This results in the discovery that all models generate almost the same percentile agreement, despite the efforts to vary the training parameters as much as possible. Only with slight deviations the generated partitions achieve a value of approximately 63%. This is especially surprising as an increase in k results at some point in a high class within cluster purity. Nonetheless, as it is worked with a very large data set this process takes an unfeasible amount of run-time or computational power and generally leads to a high number of small, less expressive clusters. The development plot of the most tested and exemplary model illustrates this issue (see figure 7). To make sure that this is not an implementation issue, the clustering and evaluation process are validated with the classic iris data set. Common clustering results and the display of a known increase of percentile agreement (up to 10 k) prove the correctness of the implementation.

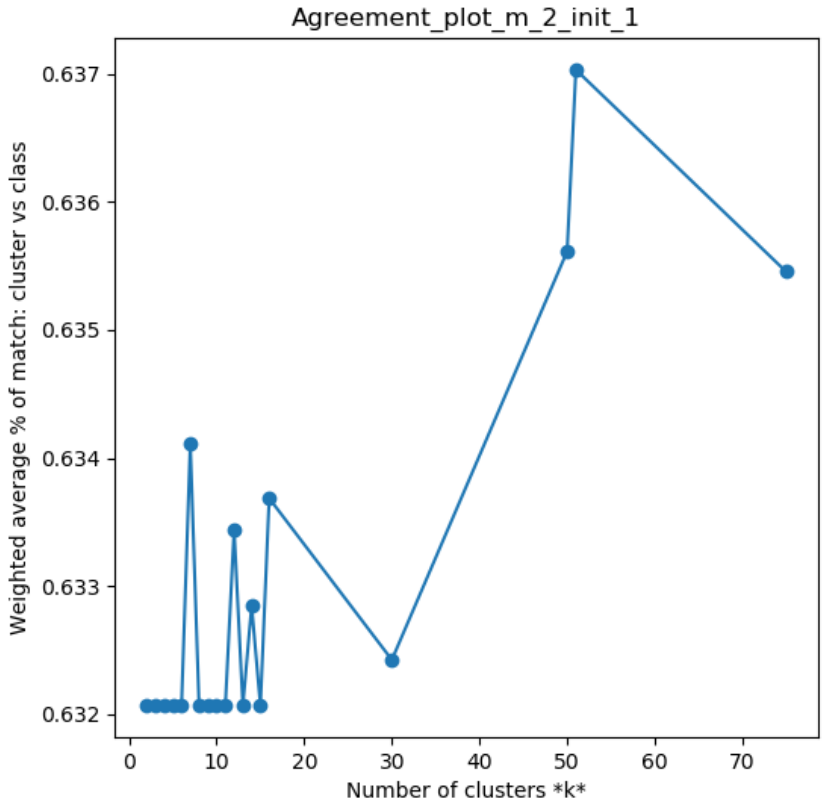


**Figure 7:** Exemplary development of percentile agreement (epoch=600, m=2, init=1, varying k)

In summary these hardly altered results in cluster vs class agreement reveal a very uniform distribution of the two class labels within the data. The clustering might therefore still generate partitions that are correct, but of unknown thematic relevance. A hardly changing influence of parameter adjustments leads to the conclusion that the two classes are not suitable as a benchmark for cluster validation. Due to the great number of instances and the high dimension of the data set, this is to be expected from a simple two class labelling.

The limited scope of this project only allowed for an external validation approach. An additional validation with internal indexes would be a valuable extension of the evaluation. In the delivered software lais a foundation in this direction, implementing functions for indexes like silhouette. A comparison with validation results from such indexes could provide further certainty about the questionable utility of the given external benchmarks.

### 4.4 Exemplary model application

The following shows some samples from the applied comparison from newly assigned tweets from the test data with tweet assigned to the corresponding cluster (m=2, k=75, init_strategy=Random):

Cluster 01 - test tweet: “@DynamicShock yep np lol”

1. “Is cooking lunch. Can’t stand the igcse pressures. Math paper 2 is only 2.5 days away I’m terrified.r math paper day away terrifi”
2. “I wish I was playing Catan.”
3. “I added 6 new adoptables to the site, and some emoticons ”

Cluster 21 - test tweet: “bow chika bow chika bow chika bow chika wow wooooooooow”

1. “@aj99000 pineapple is my faave! I can eat it till my tongue goes numb. LOL! Guess IRS the Hawaiian in me.”
2. “For those of you who were subjected to my not so bubbly tweets - sorry about that, I’m feeling way better now Will @ message you later”
3. “@icklesal I do wish I was born in the 80s! They seem much cooler. Though my mom isn’t that cool, she thinks david tennant is ugly!”

Cluster 41 - test tweet: “missed him..haven’t talk this much to him for about a year already..but still.. it’s so totally over.”

1. “@hepxxx i missed you too x”
2. “Now has Miss Texas hair for the wedding. So glad the lady listened to what I wanted...not.”
3. “@jinnah I missed it. The SC feed broke and never fixed itself so I had to switch to the Alaskan feed.”

Cluster 71 - test tweet: “True, highly subjective of me there. Tombre was actually my favorite character in the book. You got me - http://is.gd/13be0 - Rishabh”

1. “I miss both game 7s today Thank goodness for Tivo :-D If someone ruins the games for me before I see them, there will be hell to pay ;-/”
2. “listening to wat he told george so she knows its not bullshit bye”
3. “count sleep...only got bout 3 hours of sleep and now going to lovely school! oh what joy! ”

There is no obvious connection between the these samples except for cluster 41 where all tweets contain the word “missing” or related forms. However, the sample contains the word “Miss” which might be somehow misunderstood.

To get a quick impression of the topics a cluster is about, it is possible to create wordclouds from all tweets that belong to this cluster. An example for Cluster 41 can be seen in Figure 8.


**Figure 8:** Wordcloud with the tweets of cluster 41.

Another clear thematical correlation between the tweets exists in cluster 47, as its wordcloud in Figure 9 shows.

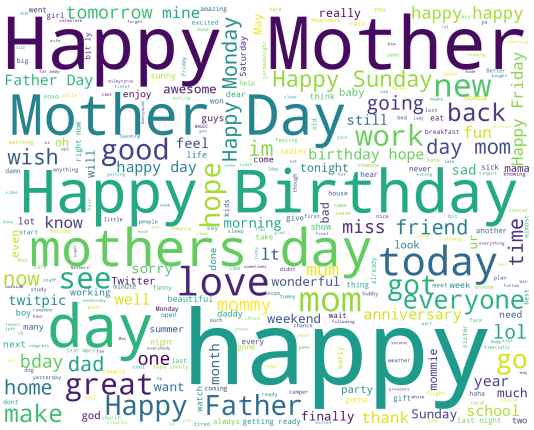


**Figure 9:** Wordcloud with the tweets of cluster 47. Most of the tweets contain congratulations.

### 4.5 Critical analysis and discussion of results

One of the main findings of the evaluation is that k-means clustering seems unsuited for grouping the tweets by their associated sentiment labels. Even the highest agreement rates (around 63%) cannot be called satisfactory, as a classification by chance would yield around 50% agreement. However, one has to take the limited scope of this project into account. With a larger data set for clustering, more k-means iterations, or a broader variation of the parameters m and k, might result in better performances.

An intuitive explanation that justifies the poor performance of the trained k-means results for grouping tweets lies in the nature of how Doc2Vec works. It is more suited for detecting similar context and

thematic relationships between the tweets. Detecting the sentiment of a tweet however goes beyond this, and needs a deeper understanding of the content and its semantics. Therefore it can be doubted that Doc2Vec, even when trained as many epochs as in this project, would even reflect the sentiments contained by the tweets. It does not seem suited for weighting the difference between similar sounding tweets like "I really like school today" and "School could really be better today", in such a way that the vector representations reflect their different emotions properly. In fact, both the tweets are very similar as they cover the same topics (an opinion about school). One can therefore argue that it is justified to group both into the same cluster with the help of k-means.

Moreover, the typical character of twitter tweets can partly explain the bad performance. Due to their limited length, they sometimes do not contain enough content to make a justified guesses about their sentiment. A pure grouping based on a sentiment attribute can therefore hardly exist. The scarcity of information applies even more for the cleaned versions of the tweets.

Furthermore, there might occur a bias within the used data. Since the tweets are labeled by the contained emoticons, the labels are noisy. They do not necessarily represent the actual sentiment expressed by the tweet author. A non-noisy labeled data set might lead to better results applying the introduced clustering workflow. This relates to the general subjective character of sentiment analysis. It is not unambiguous how people interpret sentiments, especially in written text. And it is even more difficult if the examined text data is limited to a very small amount of characters.

# 5 Conclusion

Even if the attempts of using k-means clustering for grouping the tweets by their sentiments did not give satisfactory results, they still lead to a valid conclusion. As described before, the project results can be seen as a hint that a correlation between the Doc2Vec representations and sentiments hardly exist. This project provides empirical findings for this matter concerning the unsatisfactory detection by the k-means algorithm. Nevertheless, the clustering results can be usable for finding related tweets for new instances. By analyzing the wordclouds for each cluster, one can find inspiration concerning possible connections between the tweets and topics of similar nature.

As the cluster-sentiment experiments did not yield a good agreement, it is not advisable to further investigate a possible correlation between them. Instead, future work on this topic could try to focus more on internal evaluation for finding good results that do not depend on the purity of sentiment labels within the clusters.

# 6 Bibliography

## References

Bandyopadhyay, S. & Saha, S. (2012*a*), *Unsupervised classification: similarity measures, classical and metaheuristic approaches, and applications*, Springer Science & Business Media.

Bandyopadhyay, S. & Saha, S. (2012*b*), *Unsupervised classification: similarity measures, classical and metaheuristic approaches, and applications*, Springer Science & Business Media.

Chapelle, O., Scholkopf, B. & Zien, A. (2009), 'Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]', *IEEE Transactions on Neural Networks* **20**(3), 1–3.

Duda, R. O., Hart, P. E. & Stork, D. G. (2001), *Pattern classification*, A Wiley-Interscience publication, 2. ed. edn, Wiley, New York.

Go, A., Bhayani, R. & Huang, L. (2009), 'Twitter sentiment classification using distant supervision', *CS224N Project Report, Stanford* **1**(12), 1–6.

Hamerly, G. (2010), 'Making k-means even faster.', *Proceedings of the 2010 SIAM International Conference on Data Mining* pp. 130–140.

Hoogeveen, D., Verspoor, K. M. & Baldwin, T. (2015), Cqadupstack: A benchmark data set for community question-answering research, *in* 'Proceedings of the 20th Australasian Document Computing Symposium', ACM, pp. 2–4.

Lau, J. H. & Baldwin, T. (2016), 'An empirical evaluation of doc2vec with practical insights into document embedding generation', *arXiv preprint arXiv:1607.05368* pp. 78–86.

Le, Q. & Mikolov, T. (2014), Distributed representations of sentences and documents, *in* 'International conference on machine learning', pp. 1–9.

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013), 'Efficient estimation of word representations in vector space', *arXiv preprint arXiv:1301.3781* pp. 1–12.

Rendón, E., Abundez, I., Arizmendi, A. & Quiroz, E. M. (2011), Internal versus external cluster validation indexes, p. 27.

Rong, X. (2014), 'word2vec parameter learning explained', *arXiv preprint arXiv:1411.2738* pp. 1–21.

Witten, I. H., Frank, E., Hall, M. A. & Pal, C. J. (2016), *Data Mining: Practical machine learning tools and techniques*, Morgan Kaufmann.

Zhu, X. J. (2008), Semi-supervised learning literature survey, Technical report, University of Wisconsin-Madison Department of Computer Sciences.