PROJECT REPORT

Clustering Twitter Trajectories

submitted to the

Marshallplan-Jubiläumsstiftung
Austrian Marshall Plan Foundation
Fostering Transatlantic Excellence

submitted by:

Thomas Weiß, BSc

FH Salzburg

THE UNIVERSITY OF CHICAGO

Supervisor SUAS:	Prof. (FH) Univ.-Doz. Mag. Dr. Stefan Wegenkittl
Supervisor UChicago:	Dr. Rebecca Willett

Salzburg, December 6, 2020

Abstract

The steadily increasing amount of data generated by the Online Social Network Twitter calls for automated methods that can process data for use in identifying trends, determining user groups, performing sentiment analysis and detecting spatial hot spots, among others. Since traditional Natural Language Processing methodologies are often based on the text content only, they can suffer from the short length of tweets and the subtleties of the used language. Therefore, this thesis focuses on the approach of representing each tweet via its retweet network information and subsequently perform tweet clustering based on that data. In detail, three different clustering algorithms, the standard k-means with Euclidean distance, the k-means with a follower distance, and Sparse Subspace Clustering are used. The performance of each clustering algorithm based on the purity measure is conducted on three experiments considering different data sets. The results present that the standard k-means is not suitable to cluster tweets accordingly. In contrast, k-means with the follower distance as well as Sparse Subspace Clustering can deliver appropriate clusters, if participating users can be partitioned into distinct groups.

Table of contents

Abbreviations

API	Application Programming Interface
NLP	Natural Language Processing
NN	Neural Network
ONB	Orthonormal Basis
OSN	Online Social Network
PCA	Principal Component Analysis
REST	Representational State Transfer
SVM	Support Vector Machine
SSC	Sparse Subspace Clustering
t-SNE	t-distributed Stochastic Neighbor Embedding

List of Figures

List of Tables

1 Introduction

Since Twitter was founded in 2006, the number of users has increased steadily to around 330 million active users per month. This massive number of participants produced around 6000 tweets per second on average in 2019, which means that approximately 500 million tweets were generated every day [1]. In general, there exists a wide variety of reasons for people to post a short message, such as expressing one's prevailing mood [2], reporting about a current disaster [3], or commenting on contemporary events. However, since this massive amount of data is generated in an unstructured way, manual processing, like identifying trends, can result in an unmanageable effort. Therefore, this calls for automated methods that can structure tweets concerning their topic affiliation. These resulting topics can subsequently be used to determine users' interests, for example, to generate revenue based on a targeted advertisement or to create an invisible information layer to support any task like disaster management procedures [3].

Many topic clustering approaches consider the textual information of tweets to perform partitioning of topics. Since this data often provides some restrictions like limited text length or usage of slang or abbreviations, clustering methods can suffer from [4]. However, suppose a tweet with the following content: ”Congratulations to the winners! You've made us all proud!” Is this tweet talking about a sports event or commenting on a parliamentary election? Determining the topic affiliation (”sports” or ”politics”) based only on the text content itself can be a very challenging task in social network analysis. Therefore, adding context information can be a promising method to overcome this challenge. Mostly, available additional information on single participants in a social network (such as age, gender, location) is currently used. Also, time-correlation with major known events, such as a final match or a public vote, can be exploited. Besides, the trajectory of a tweet might give another possible source of context as it is retweeted[1]. Since Twitter only provides information about who is involved in retweeting a particular tweet, without explicitly mentioning who retweeted whom, the users who participated in retweeting a tweet could also provide an excellent additional source of information classifying tweets. However, using this non-text-based content, one can suffer from the typical short length of such trajectories compared to the enormous number of possible network participants. Consequently, representing each tweet via its retweet network will cause high-dimensional data. Therefore, this approach calls for a robust method that allows representing the dynamical behavior in a way, which enables tweet clustering based on the retweet information.

The project report starts with introducing the fundamentals of unsupervised learning in Section 2. In detail dimensionality reduction and clustering are described. Section 3 outlines the used data sets. In Section 4 feature extraction and exploratory data analysis based on the data sets as well as methods for clustering data and three different experiments are explained. Subsequently, the results of the experiments are shown in Section 5.

[1] In Twitter, retweeting describes the propagation mechanism to spread and share tweets in the social network. For further details, refer to https://help.twitter.com/en/using-twitter/retweet-faqs.

2 Unsupervised Learning

In this chapter, unsupervised learning approaches are described. In the beginning, the challenges of unsupervised learning and dimensionality reduction are depicted. Furthermore, clustering methods like the k-means, spectral clustering, and Sparse Subspace Clustering (SSC) are explained in detail. At the end of this chapter, model evaluation is introduced. All information in this chapter is derived from [5–7].

In the past, it was common practice to implement rule-based approaches along which data was processed. Thus, the increasing amount of information, the constant variation of data, and the resulting unmanageable effort to maintain handwritten decision rules call for automated methods of data analysis. This field is covered by machine learning, which is a collection of methods that can automatically detect patterns in data and use this information to perform other kinds of decision-making or predict future data. In principle, these methods can be divided into two types, depending on whether they are trained under supervision or not. Nevertheless, there exist approaches and architectures which bridge both types.

The first type, supervised learning, aims to learn a mapping from inputs x_i to outputs y_i considering that each input has a related label. In general, in supervised learning, one can differentiate between the methods classification and regression [6]. For example, a spam filter can be defined as a classification task for analyzing emails based on the text content if they are important or malicious. Before performing a classification task, it is necessary to define and train an appropriate model. For the training process, a data set consisting of samples (emails) and related labels (spam or not) is required. Then, the model is trained to detect if an email is relevant or not by minimizing a loss function. After the training, the model can be used to make predictions based on new samples, which results in a label for each new sample. Well-known examples for supervised learning algorithms are called k-nearest neighbor, linear or logistic regression, Support Vector Machines (SVMs), and Neural Networks (NNs) [2].

The second type, unsupervised learning, aims to find patterns in data, without any preliminary annotation like labeled data. Compared to the previous example, the task of detecting malicious emails using unsupervised learning is less clearly defined and more demanding in solving the problem. In principle an unsupervised learning model tries to find common patterns in the underlying structure of some data. Due to the disclosure of the underlying structures, clusters can appear, which correspond to groups like business, news, and shopping and do not have any common ground truth with the intended task of detecting spam emails. Therefore, if handled well, the unsupervised approach can be more potent than supervised learning because it might be possible to extract additional information then initially designed. Methods related to unsupervised learning are clustering, anomaly detection, anomaly visualization, and dimension-

[2] Even though Neural Networks are covered in the field of supervised learning, there exist architectures that can be trained unsupervised, like autoencoders.

ality reduction. Among others, well-known representatives of unsupervised learning algorithms are k-means and one-class SVM.

2.1 Challenges

As described in the introduction of this chapter, unsupervised learning aims at finding patterns in data without any prior information. In the absence of labels, it is difficult to measure the quality of the outcome: do the found patterns correspond to information, the user is interested in? How can we measure the quality of the clusters and compare different outcomes of such a methodology?

For example, consider the spam filter introduced before, where the desired model output is to highlight malicious emails. As labels are missing, the unsupervised model tries to find patterns in the samples without human supervision. The challenge is that the discovered underlying structures do not necessarily correspond to the expected output of being a spam email or not. Figure 2.1a shows an artificial email data set consisting of two classes, standard email (blue) and spam email (orange). A supervised learning algorithm would be able to learn a linear decision boundary considering the given labels to classify a new email as important or malevolent. In contrast, Figure 2.1b illustrates the output of an unsupervised clustering algorithm, which aims to separate the whole data set into two clusters. It is also apparent that the found clusters do not correspond to the expected labeling in Figure 2.1a. Therefore, the used clustering algorithm is not able to contribute to the task of classifying emails. The strength of this method in this synthetic example is visible in Figure 2.1c. Applying the algorithm to the data set with an optimal selection of parameters can uncover some hidden patterns, where each cluster found corresponds to different email topics.

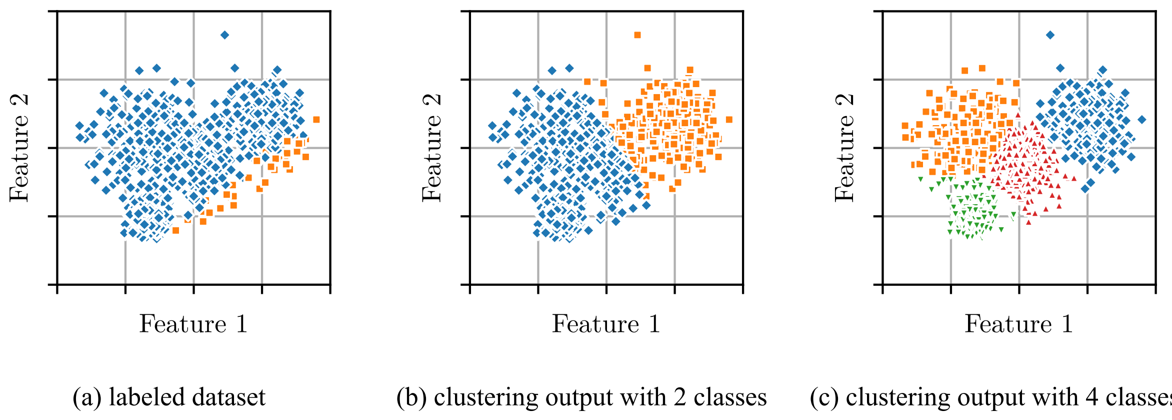


Figure 2.1: Visualization of an artificial email data set: (a) shows a labeled data set, where blue relates to standard emails and orange to malicious emails. (b) displays a possible output of an unsupervised clustering algorithm with two clusters, whereby the intersection of equal labels in (a) and (b) is limited to a few samples. (c) illustrates an optimal solution for a clustering algorithm with four clusters, where each cluster might correspond e.g. to a specific email topic like news, private, business, and sport.

In conclusion, a supervised approach can provide adequate and repeatable results if a task and the expected output are well defined, and sufficient labeled data is available and does not change significantly over time. Since unsupervised learning has the ability to extract underlying structures, it can deal with situations where patterns are unknown or continuously changing. With this intention, unsupervised learning can complement a supervised learning approach by providing stable features based on hidden patterns. Therefore, unsupervised learning can help at finding solutions to previously unsolvable problems, as it is more flexible in exploring hidden patterns in different data sets. Similarly, unsupervised learning is often used in exploratory data analysis, for instance, as a visualization technique.

2.2 Dimensionality Reduction

This section gives an overview of dimensionality reduction, where the following information is derived from [5, 6]. Dimensionality reduction is defined as the process of reducing the number of features in a given set of data by projecting the data onto a lower-dimensional subspace such that essential properties are captured. For example, if one measures the length, the width, and the height of a rectangular cuboid, then these variables are called features and specify the number of dimensions of the data sample, which equals three in this case. In comparison, images are located in high dimensional spaces because the number of dimensions is equal with the number of pixels and consequently varies with the image size. For instance, the mnist data set [8] consists of handwritten digits from 0 to 9 with size 28×28 pixels which results in a 784-dimensional feature space. Although this data set can be found in a high dimensional environment there may exist a smaller number of dimensions thereby preserving this aspect of image content in lower dimensionality. Finally, reducing the number of features will cause less computational effort and a decreasing amount of samples required for training supervised classifiers. This will be explained in Section 2.2.1.

In general, dimensionality reduction can be divided into two types, feature selection, and feature extraction. Feature selection is defined as selecting relevant features to decrease the number of dimensions. For example, if a task only uses a rectangular cuboid's length and width, selecting only those both variables and removing the height is called feature selection. In contrast, feature extraction is the process of aggregating features or finding subspaces to decrease the number of dimensions. For example, reducing the number of features of a rectangular cuboid can be established by aggregating width and length to the feature ground area and removing width and length. This process reduces the number of features but keeps the information about both removed variables. Depending on the application, this reduction of dimensionality can help or hinder the use of the features for designing efficient machine learning.

A disadvantage, which has to be considered, is that dimensionality reduction typically removes some information. Thus, dimensionality reduction has only to be performed concerning a given task. In terms of exploratory data analysis, reducing the number of features, for example, from ten to three to visualize data, can cause information loss such that expected results are not visible anymore. Nevertheless, this does not imply that the data consisting of all dimensions does not contain the necessary information in general.

2.2.1 Curse of Dimensionality

Serious challenges in selecting an appropriate model for a specific task can occur if data lies in a high-dimensional space. Those difficulties are based on the fact that data tends to be located at the surface of the high dimensional space if the number of features increases, as explained in [5]. Further, this implies that distances between data points will also grow exponentially with their dimensionality, such that distances between data points will get almost equal. This influence is demonstrated as follows. Depending on this effect, some algorithms will deliver unsatisfactory results caused by the sparsity of the data points. Another drawback of high-dimensional data is that apart from the computational effort, the necessary number of training samples increases exponentially. For example, if different regions in a high-dimensional space might refer to different data classes, each corner in the hypercube would have to be represented by samples independently of each other. This results in a exponential increase of necessary training samples in terms of the number of features.

To illustrate the influence of increasing dimensions the volume of a sphere is used as an example. Denote by

$$V_D(r) = K_D r^D \tag{2.1}$$

the volume of a sphere located in D dimensions, where r is the radius, and K_D specifies a constant, which only depends on the dimensionality D. In the case of a three-dimensional sphere, K_D will be equal $\frac{4}{3}\pi$. Further, specify a function

$$f(D, \epsilon) = \frac{V_D(1) - V_D(1-\epsilon)}{V_D(1)} = 1 - (1-\epsilon)^D \tag{2.2}$$

which calculates the fraction of the volume of a sphere being between radius $r = 1$ and $r = 1-\epsilon$ [5]. In Figure 2.2, the comparison between four selected dimensions 1, 2, 5, and 20 is visualized. $D = 20$ shows a massive increase of the volume fraction such that more than 98 percent of the volume is located in the outer twenty percent of the sphere. As dimensions grow, this effect will continue exponentially.

Besides that, the curse of dimensionality causes some relevant issues; there are techniques that can process high dimensional-data. This ability is based on one or both of two properties, which occur in real data. On the one hand, data often lies in a lower-dimensional subspace, enabling

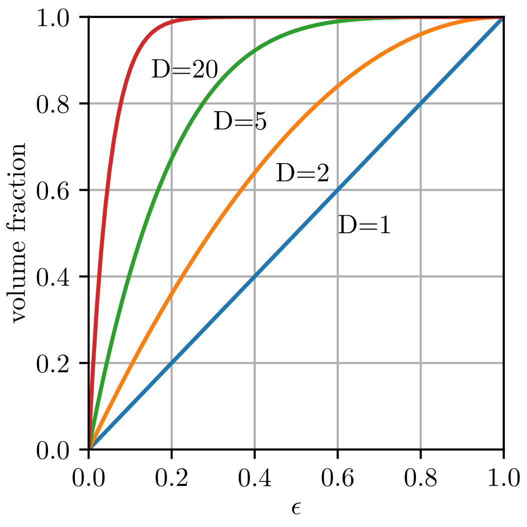


Figure 2.2: Comparision of the fraction of the volume of a D-dimensional unit-sphere lying in the range from $r = 1 - \epsilon$ to $r = 1$ [5].

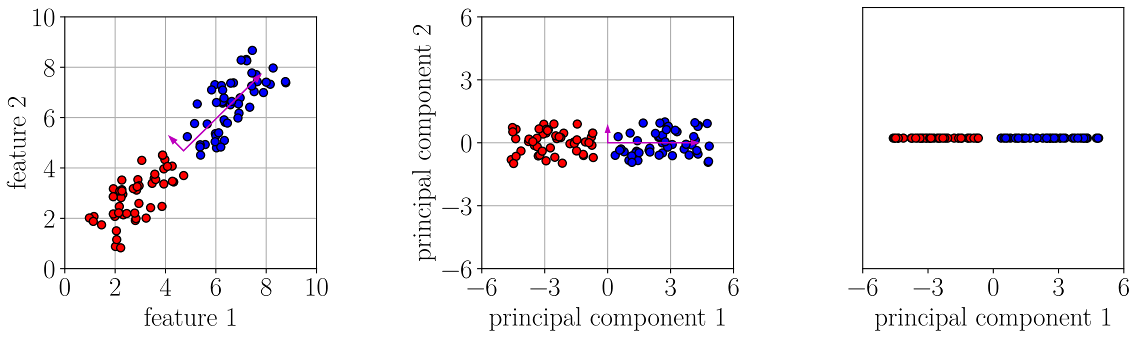
transformation into a lower-dimensional representation without relevant loss of information. On the other hand, one can often make smoothness assumptions such that small changes in the data are unlikely to affect the correct labeling - classes tend to be defined as connected subspaces in the feature space. These characteristics often help to reduce model complexity.

2.2.2 Principal Component Analysis

Principal Component Analysis (PCA) is a widely used technique for dimensionality reduction, feature extraction, and data visualization. PCA can be defined either as the linear projection that minimizes the mean squared distance between data points and their projections or as the orthogonal projection of data onto a lower-dimensional linear space. Although both definitions are equivalent and result in the same algorithm, this section focuses on the orthogonal projection definition where the following information is distilled from [5, 9].

PCA's primary goal considering orthogonal projection is to find an Orthonormal Basis (ONB) of $\mathbb{R}^d$ that transforms the data, such that it is distributed along the orthonormal vectors in decreasing order of the variance. Moreover, the corresponding eigenvalues represent the amount of variance. Thus, depending on the used data set, taking the first coordinates or called ”principal components”, the dimensionality of the original data can be decreased with little loss.

In Figure 2.3a, an input data set is visualized, where data points are spread in two clusters along with two features. PCA now aims to find an ONB that transforms the input data set, considering that each principal component contains the maximum variance in each direction of the data in decreasing order. This means that the first principal component includes the most variance in the data; the second principal component contains the second-most, and so on. Figure 2.3b dis-



(a) input space (b) principal components (c) reduced output space

Figure 2.3: Visualization of PCA on a synthetic data set. (a) displays an artificial data set in the input space. (b) shows the orthogonal projection of the input data onto principal components. (c) presents the reduced output space based on the first principal component.

plays the distribution of the transformed input data over the principal components. Figure 2.3c presents the dimensionality reduction to a single feature by removing the second principal component. Consequently, separating data into two clusters is still possible because only a little information is lost by removing one dimension.

To perform a PCA, first denote $\boldsymbol{x}$ as a single input sample lying in a d-dimensional space $\mathbb{R}^d$, where d specifies the number of dimensions or features. Define

$$\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n\}, \qquad \boldsymbol{X} \in \mathbb{R}^{d \times n} \tag{2.3}$$

as a matrix consisting of samples, where each column refers to a single d-dimensional sample $\boldsymbol{x}$ and n specifies the number of samples. To build the covariance matrix $\boldsymbol{\Sigma}$ first center the input data by calculating the difference between each feature value and the corresponding feature mean. In detail, calculate the mean μ_i for each row and then subtract the result from each value in the corresponding row. This leads to the centered input matrix $\bar{\boldsymbol{X}}$. Then, calculate the symmetric covariance matrix

$$\boldsymbol{\Sigma} = \bar{\boldsymbol{X}} \cdot \bar{\boldsymbol{X}}^T, \qquad \boldsymbol{\Sigma} \in \mathbb{R}^{d \times d} \tag{2.4}$$

by multiplying the centered input matrix $\bar{\boldsymbol{X}}$ and its transpose $\bar{\boldsymbol{X}}^T$. Since $\boldsymbol{\Sigma}$ is a symmetric real-valued matrix ($\boldsymbol{\Sigma} = \boldsymbol{\Sigma}^T$), it is orthogonally diagonalizable having only real eigenvalues [10]. This means, that there exist real numbers $\lambda_1, \lambda_2, \dots, \lambda_d$ called eigenvalues as well as orthogonal non-zero real vectors $\boldsymbol{v_1}, \boldsymbol{v_2}, \dots, \boldsymbol{v_d}$ called eigenvectors such that

$$\boldsymbol{\Sigma} \cdot \boldsymbol{v_i} = \lambda_i \cdot \boldsymbol{v_i}, \qquad i \in \{1, 2, \dots, d\} \tag{2.5}$$

is satisfied. Subsequently, calculate the eigenvalues and eigenvectors of $\boldsymbol{\Sigma}$ and sort the eigenvectors in descending order of the eigenvalues. In this context, the eigenvectors represent the principal components. However, selecting only the first $\widetilde{d}$ eigenvectors, where $\widetilde{d} < d$, causes dimensionality reduction of the data. The percentage of variance covered in the first $\widetilde{d}$ principal components is calculated by

$$V_{\widetilde{d}} = \frac{\sum_{i=1}^{\widetilde{d}} \cdot \lambda_i}{\sum_{i=1}^{d} \cdot \lambda_i} \cdot 100\%. \tag{2.6}$$

Finally, define

$$\boldsymbol{\Lambda} = \{\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_{\widetilde{d}}}\}, \qquad \boldsymbol{\Lambda}^{d \times \widetilde{d}} \tag{2.7}$$

as the ONB consisting of the first $\widetilde{d}$ eigenvectors and transform input data by calculating

$$\widetilde{\boldsymbol{X}} = \boldsymbol{\Lambda}^T \cdot \bar{\boldsymbol{X}}, \qquad \widetilde{\boldsymbol{X}} \in \mathbb{R}^{\widetilde{d} \times n}. \tag{2.8}$$

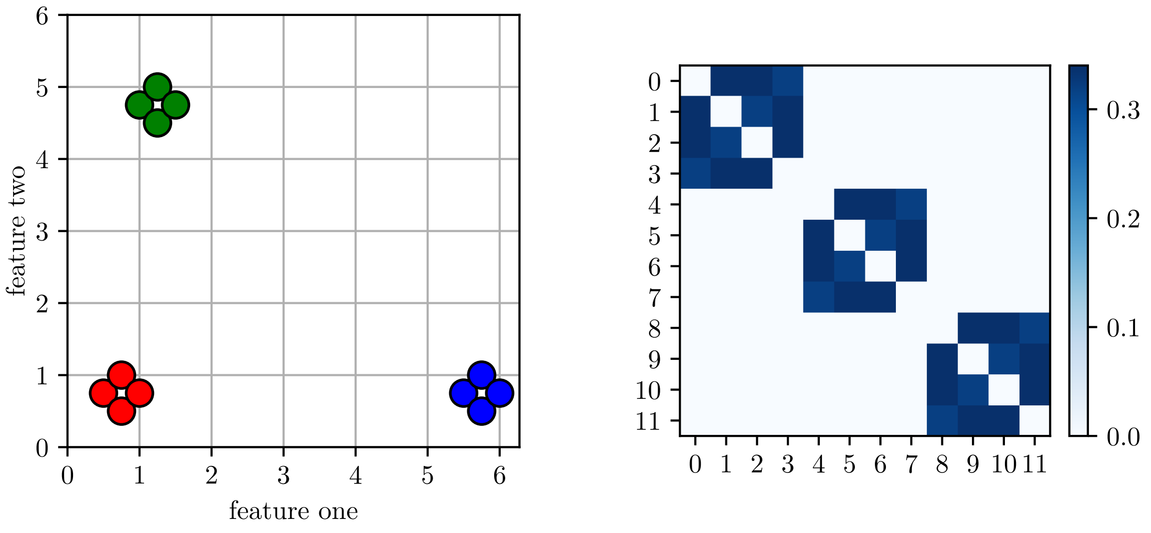
For future calculations, like transforming new data samples, the mean values of the input matrix and the ONB $\boldsymbol{\Lambda}$ have to be stored. As PCA is a powerful method to perform dimensionality reduction on high-dimensional data sets, it can only handle linear dependencies. In detail, dimensionality reduction considering PCA will remove important information if data is lying in a non-linear subspace. Furthermore, PCA is negatively affected if data can only be separated in the direction of the eigenvector with the lowest eigenvalue. Consequently, these characteristics can cause essential information loss, followed by delivering unexpected results.

2.2.3 t-distributed Stochastic Neighbor Embedding

As mentioned in Section 2.2.1, real data often lies in a low-dimensional subspace of an existing high-dimensional data set. Through powerful non-linear transformations, it is possible to transform data into a low-dimensional representation, which can be handy for human visualization purposes. t-SNE is located in the field of non-linear dimensionality reduction or also called manifold learning and can unravel hidden patterns or underlying non-linear structures.

The t-SNE algorithm is be explained by applying it to a simple artificial data set visible in Figure 2.4a. This data set contains three clusters, each consisting of four samples lying in a two-dimensional space. In this example, the data is finally transformed from a two- into a one-dimensional representation. Furthermore, all used content in this section is based on [11].

Denote $\boldsymbol{x_i}$ as the i^{th} data point lying in the original d-dimensional space $\mathbb{R}^d$. Further, define a *map* point $\boldsymbol{y_i}$ as the i^{th} data point lying in the $\widetilde{d}$-dimensional *map* space $\mathbb{R}^{\widetilde{d}}$, where $\widetilde{d} < d$. The *map* space contains the final representation of each data point transformed from the original to



(a) visulization 2D data set

(b) t-SNE similarity matrix input data

Figure 2.4: t-SNE 2D data set and corresponding input similarity matrix visualization

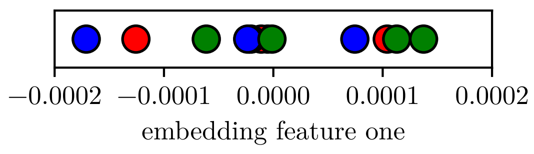
the *map* space. In terms of the synthetic example, a two-dimensional input space ($d = 2$) is transformed into a one-dimensional *map* space ($\widetilde{d} = 1$). Correspondingly, compute the conditional similarity $p_{i|j}$ of $\boldsymbol{x}_i$ picking $\boldsymbol{x}_j$ as its neighbor where neighbors are chosen under consideration of the probability density under a Gaussian distribution centered at $\boldsymbol{x}_i$. Therefore, define

$$p_{i|j} = \frac{e^{-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}}}{\sum_{k \neq i} e^{-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}}}, \qquad p_{i|i} = 0 \tag{2.9}$$

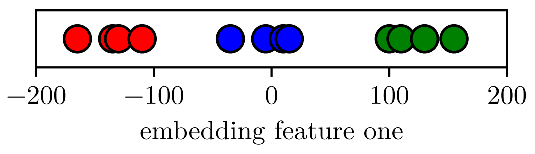
as the conditional property, where $\boldsymbol{\sigma}_i$ specifies the variance of the Gaussian distribution, which is centered on data point $\boldsymbol{x}_i$ and d specifies the distance measure between two points. As $\boldsymbol{\sigma}_i$ varies for each $\boldsymbol{x}_i$ it is automatically optimized by setting the parameter perplexity in the t-SNE algorithm. The optimization results in a small value for $\boldsymbol{\sigma}_i$ if $\boldsymbol{x}_i$ is located in a dense region and in a large value for $\boldsymbol{\sigma}_i$ if $\boldsymbol{x}_i$ is located in a sparse region. Since all $\boldsymbol{\sigma}_i$ are implicitly influenced by the mandatory perplexity parameter, Laurens and Geoffrey [11] recommend using a value between 5.0 and 50.0. After calculating the conditional similarities,

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{2.10}$$

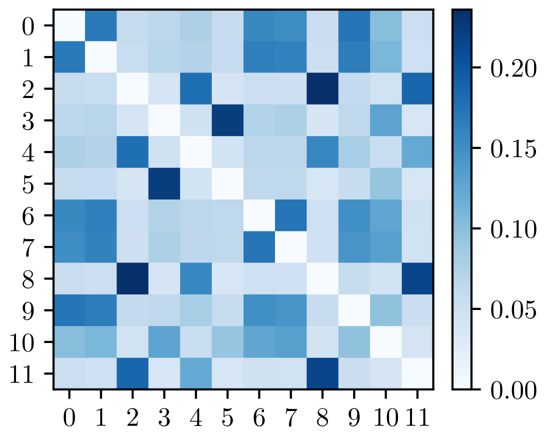
is applied to make the conditional similarities symmetric such that $p_{i|j}$ is equal to $p_{j|i}$. Figure 2.4b visualizes the optimized symmetric input affinity matrix, where dark blue characterizes a high similarity and white corresponds to a low similarity between data points. Subsequently, the data points from the original space are randomly distributed in the *map* space under consideration of



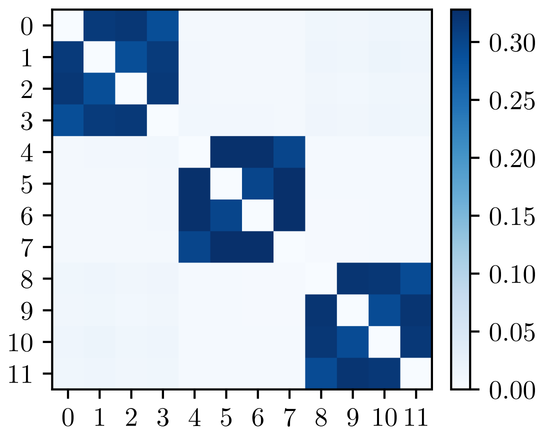
(a) initial position



(b) optimized position



(c) similarity matrix on initial map space positions



(d) similarity matrix on optimized map space positions

Figure 2.5: Comparison between the initial and optimized placement of data points in 1D t-SNE *map* space. (a) shows the initial randomized placement of data points under a Gaussian distribution with parameters $\mu = 0$ and $\sigma = 0.0001$. Additional, (c) visualizes the similarity matrix corresponding to (a) . In contrast (b) displays the final positions or the optimized embedding space. (d) represents the similarity matrix in *map* space concerning data points in (b).

a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 0.0001$ (see Figure 2.5a). After the initial sampling

$$q_{ij} = \frac{(1 + ||\boldsymbol{y}_i - \boldsymbol{y}_j)||^2)^{-1}}{\sum_{k \neq l}(1 + ||\boldsymbol{y}_k - \boldsymbol{y}_l)||^2)^{-1}}, \tag{2.11}$$

is used to calculate the *map* space similarity matrix (see Figure 2.5c), which uses a t-distribution to calculate the similarities between data points. The t-distribution is used for the reason to prevent clumping in the final *map* space. It is visible that Figure 2.5c and Figure 2.4b highly differ. Thus, the t-SNE algorithm aims to move data points in the *map* space such that the similarity matrix $\boldsymbol{Q}$ in *map* space best approximates the input similarity matrix $\boldsymbol{P}$. In other words, the algorithm tries to keep distances such that points that are close to each other in the input space are close to each other in the *map* space and vice versa. For that reason, a gradient descent approach is used to decrease the total error between the similarity matrix in the input space and *map* space. Therefore, define

$$C = KL(P|Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.12}$$

as the cost function between $\boldsymbol{P}$ and $\boldsymbol{Q}$, which is based on the Kullback-Leibler divergence. Further, define

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij}) g\left(|y_i - y_j|\right) u_{ij} \left(1 + \|y_i - y_j\|^2\right)^{-1} \tag{2.13}$$

as the partial derivative of y_i and move the data point according to the derivative into a direction, such that the total error decreases. The algorithm performs the optimization process by repeating the steps calculating the *map* space similarity matrix, computing the partial derivatives, and moving data points in *map* space until the total error reaches a minimum. In other words, one can imagine the optimization such that data points that are similar in input space attract each other, whereby dissimilar entries repel each other. These steps are continued until all forces cancel each other out. Correspondingly, the output of the t-SNE algorithm highly depends on the initial distribution of data in the *map* space, always resulting in different outcomes for each run. The final converged output is shown in Figure 2.5b. Additional Figure 2.5d visualizes the final similarity matrix, whereby in comparison with Figure 2.4b only minimal differences are visible.

In conclusion, t-SNE is a powerful unsupervised algorithm that is able to transform non-linear subspaces into a lower-dimensional embedding. However, it should be taken into account that t-SNE, on the one hand, tries to keep distances such that clusters will be preserved but, on the other hand, provides different outputs based on the initial randomness in the *map* space. As a result, this algorithm can not map a new data point on a previously calculated output space. For this reason, t-SNE is often used in exploratory data analysis, where high-dimensional data is visualized in two or three dimensions and colored according to relevant aspects. Such visualizations can be investigated by humans and have become a frequently used tool in any kind of data analytics or exploratory data analysis.

2.3 Clustering

Clustering is contained in the field of unsupervised learning. It is defined as the task of grouping similar data points to clusters such that objects in the same group have high similarity, but objects in different clusters are very dissimilar. In contrast to classification, where similarity is based on predefined labels, clustering uses different similarity measures based on the features themselves to determine if objects are near or distant from each other. Depending on the clustering method, similarity can be expressed as either spatial distance or as density- and continuity measure. On the one hand spatial distance measures relying on different metrics like Euclidean or Manhattan distance can often benefit from optimization techniques. On the other hand density- and continuity measures can be used to find clusters of any shape. Therefore, selecting a suitable similarity measure is a fundamental task in the design and development of clustering methods.

In general, clustering methods can be divided into four groups: partitioning, hierarchical, density-based, or grid-based methods. Partitioning takes the approach that a set of n objects or data points are partitioned into k groups or called clusters of the data where $k \leq n$. Such methods are often distance-based and follow an iterative optimization to minimize distances between data points and the corresponding cluster centers. For example, this can be established by moving the cluster centers until the total distance of each data point to its related cluster center is minimized. Hierarchical methods create a hierarchical decomposition of the data by applying either a top-down or bottom-up approach. For instance, the latter starts by assigning each data point to a single cluster and then repeatedly merging two clusters until a single cluster contains all data. In contrast, the topdown approach starts with a single cluster and repeatedly splits data until clusters with only a single data point exist or a termination condition is met. On the contrary to partitioning methods, density-based clustering tries to find clusters by growing a given cluster until the density in a specific area falls below a predefined threshold. In detail, each cluster has to contain at least a predefined number of data points in the neighborhood of a given radius. Grid-based methods build a grid structure by quantizing the data space into a finite number of units. This approach increases performance in terms of processing time and is often used for spatial data mining problems. Since these four approaches define independent clustering algorithms, some methods exist that adopt some of the ideas mentioned earlier.

2.3.1 K-means Clustering

This section introduces the k-means clustering algorithm, where the following content is based on [5]. k-means clustering belongs to the field of partitioning-based clustering methods and follows the approach for grouping a data set into k disjoint clusters. The algorithm aims at assigning a label to each data point in a multidimensional space that refers to a specific cluster number. Since the number of clusters k is not determined automatically, k-means needs this preliminary information to perform the clustering.

To mathematically describe the k-means algorithm first define $\boldsymbol{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ as n data points lying in a d-dimensional Euclidean[3] space. The goal is to partition the data set into a certain number of k distinct clusters in order to minimize the total sum of distances between the data points and the respective cluster center. Therefore, define $\boldsymbol{C} = \{\boldsymbol{C}_1, \dots, \boldsymbol{C}_k\}$ as clusters such that $\boldsymbol{C}_i \subset \boldsymbol{X}$ and $\boldsymbol{C}_i \cap \boldsymbol{C}_j = \emptyset$ for each $i \neq j$ where $i, j = \{1, 2, \dots, k\}$. In other words, each cluster consists of disjoint sub sets $\boldsymbol{C}_i$ of all samples $\boldsymbol{X}$ such that each data point is

[3] The vector space $\mathbb{R}^n$ combined with the metric induced by the scalar product is called n-dimensional Euclidean space [12].

only assigned to a single cluster. Additionally, each group $\boldsymbol{C}_k$ is represented by a d-dimensional vector $\boldsymbol{\mu}_k$ which characterizes the position of a cluster centroid in the data space. Further, define

$$r_{nk} = \begin{cases} 1, & \text{if } k = \arg\min_i \|\boldsymbol{x}_n - \boldsymbol{\mu}_i\|^2 \\ 0, & \text{else} \end{cases} \tag{2.14}$$

as a binary indicator vector of size k, which specifies the affiliation of the i^{th} object $\boldsymbol{x}_i$ to the k^{th} cluster. As specified in (2.14) the distance between a data point $\boldsymbol{x}_n$ and a cluster center $\boldsymbol{\mu}_k$ is usually calculated employing the Euclidean distance. Moreover, define

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2 \tag{2.15}$$

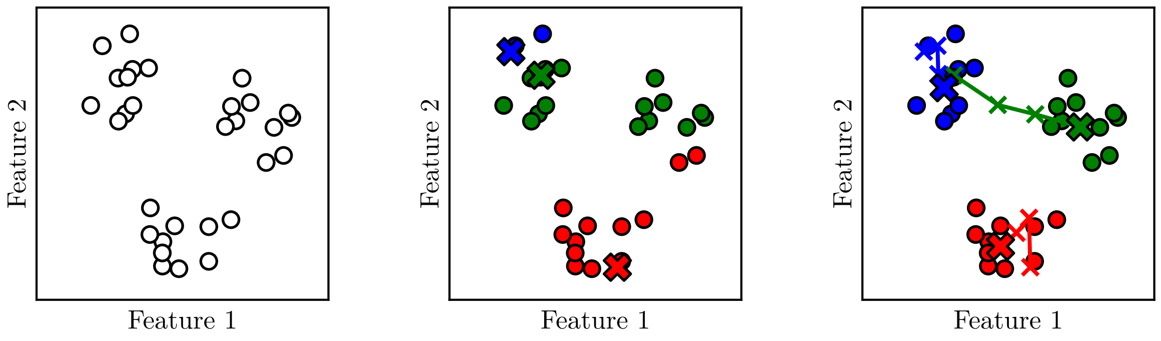
as the objective function, representing the sum of squared distances between each object and its related cluster centroid. Besides, define

$$\boldsymbol{\mu}_k = \frac{1}{\#\boldsymbol{C}_k} \sum_{\boldsymbol{x} \in \boldsymbol{C}_k} \boldsymbol{x} \tag{2.16}$$

as the mean of all data points assigned to a certain cluster.

The k-means algorithm is illustrated using an artificially generated data set visible in Figure 2.6. The data set consists of three distinct clusters which are visualized in Figure 2.6b. Since k-means requires the parameter k number of clusters, it has to be specified in advance. The goal in this example is to partition the data set into three clusters. Therefore k has to be set to 3. The initialization in k-means starts with the randomized distribution of the cluster centers $\boldsymbol{\mu}_k$ in the data space. Thus, all cluster centers are represented by a cross sign in the corresponding figures. This visualization should help to get a sense of how centroids are moving from one to the other optimization steps. The first step in performing k-means clustering is called (re)assigning data points. This procedure covers finding the closest cluster center representative $\boldsymbol{\mu}_k$ and assigning its label to the current data point $\boldsymbol{x}_n$ (see (2.14)). The second step is designated (re)computing cluster centroids. This phase contains the (re)calculation of all values of $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$ under consideration of (2.16). Finally, the steps (re)assigning labels and (re)calculating cluster centers are repeated until no updates for $\boldsymbol{\mu}$ occur. If μ can not be updated anymore, this termination relates to reaching local minimum of the objective function J which results a clustering output of k-means.

To determine if the quality of a k-means clustering output, one can calculate the within-cluster variance (see (2.15)) and compare it with a second and a third run. If results stay stable, the influence of randomness in the initialization step of k-means has less influence on the final result. To get stable clustering outcomes, many implementations of the k-means algorithm use the k-means++ procedure. In contrast to the standard random initialization, k-means++ improves the



(a) data set without labels (b) initial cluster centers (c) final clustering

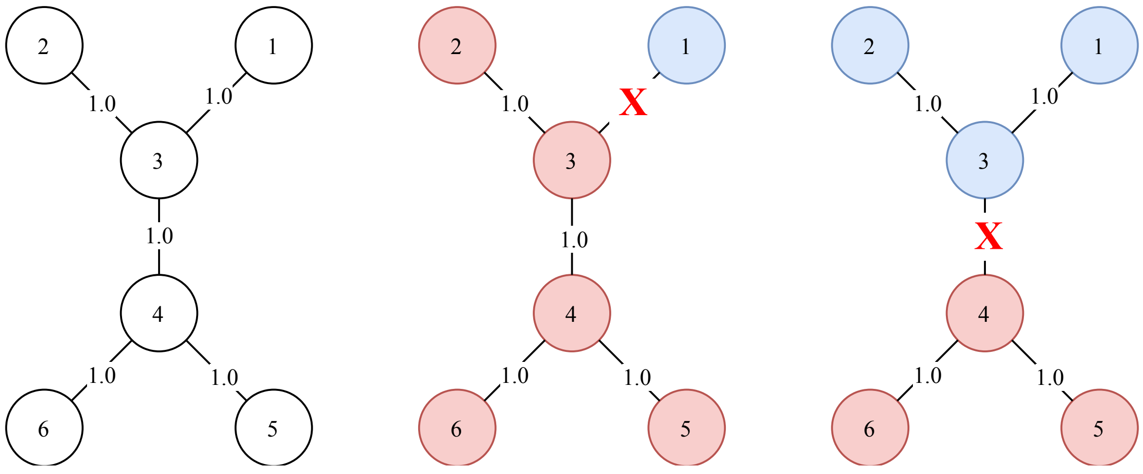
Figure 2.6: Visualization of applying k-means on a synthetic data set: (a) shows a synthetic data set without any labeling. (b) visualizes the initialization step of k-means. Therefore the initial cluster centers are represented with a cross. (c) presents the final clustering output where the intermediate positions of the cluster centers are visualized by a path.

seeding process, which refers to selecting the initial cluster centers. In short, this is achieved by the random selection of the first cluster center and the subsequent choice of all further centroids, taking into account the already selected cluster centers [13].

In conclusion, k-means is a simple, fast, and easy to understand algorithm, which has its advantages if data can be separated into groups in a linear space. Besides, it has been proved that using k-means++ instead of the random cluster initialization positively influences the algorithm's overall performance in terms of speed and accuracy [13]. On the contrary, the clustering can suffer from the random initialization step of selecting cluster centers, which results in different clustering outcomes for each run, also called non-deterministic behavior. Equally important is that k-means based on the use of the Euclidean metric comes with some limitations like the type of data variables that can be taken into account or robustness against outliers. Accordingly, these restrictions can lead to inappropriate clustering results, which can be circumvented using different distance metrics [5].

2.3.2 Spectral Clustering

This section describes spectral clustering, where the following information is derived from [14]. Spectral clustering is a different modern unsupervised learning algorithm, which is located in the field of clustering algorithms. The advantages of this clustering approach are that it is simple to implement, can be efficiently solved by basic linear algebra methods, and often outperforms other approaches like k-means algorithm, which is described in Section 2.3.1. In comparison to k-means, spectral clustering uses an affinity measure instead of the absolute location of data points to determine which sample belongs to the same cluster. This approach has the advantage of being able to cluster any shape of data with the precondition that the similarity between data points in the same cluster is high, and the similarity measure of data points in different clusters is low.



(a) weighted graph (b) 2 clusters cut criterion (c) 2 clusters Ncut criterion

Figure 2.7: Graph cut example: Basically, each graph cut is visualized by a red cross sign in the corresponding figure. (a) displays an undirected weighted graph. (b) shows a clustering output based on the cut criterion. (c) represents a clustering output based on the Ncut criterion.

Before diving into some details of spectral clustering, this section introduces the algorithm in terms of graph cuts on a synthetic example visualized in Figure 2.7. In general spectral clustering aims to partition a graph into predefined k clusters $A_1, \dots, A_k$. Therefore, let $\boldsymbol{W}$ be the weighted adjacency matrix of a given similarity graph, as shown in Figure 2.7a. Moreover, denote $W(A,B) = \sum_{i \in A} \sum_{j \in B} w_{ij}$ as the sum of edge weights which are connecting set A with set B. Furthermore, $\bar{A}_k = V \setminus A$ describes the complement of A_k such that $A_k \cap \bar{A}_k = \emptyset$ and $A_k \cup \bar{A}_k = V$ where V is the set of vertices in the graph. An intuitive way of separating the graph into clusters is to minimize the mincut problem

$$cut(A_1, \dots, A_k) = \frac{1}{2} \sum_{i=1}^{k} W(A_i, \bar{A}_i). \tag{2.17}$$

Since this approach is easy to solve for two clusters, it often results in a local minimum separating a single node from the rest of the graph, shown in Figure 2.7b. To improve the clustering output denote

$$Ncut(A_1, \dots, A_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{cut(A_i, \bar{A}_i)}{vol(A_i)} \tag{2.18}$$

as the *normalized cut* where $vol(A_i) = \sum_{j \in A_i} d_j$ and d_j refers to the weighted degree of node j [14]. Minimizing (2.18) aims to find graph cuts that partition the graph such that both graph cuts are minimized, and clusters are requested to being ”reasonable large”. The possible output of calculating the Ncut criterion is represented in Figure 2.7c. Since solving the Ncut problem is

NP-hard[4], spectral clustering aims to approximate the Ncut approach under consideration of the graph Laplacian and its eigenvalues [14]. In principle, spectral clustering can be divided into three main steps:

i) *Pre-processing:* transform the input data into a matrix representation.

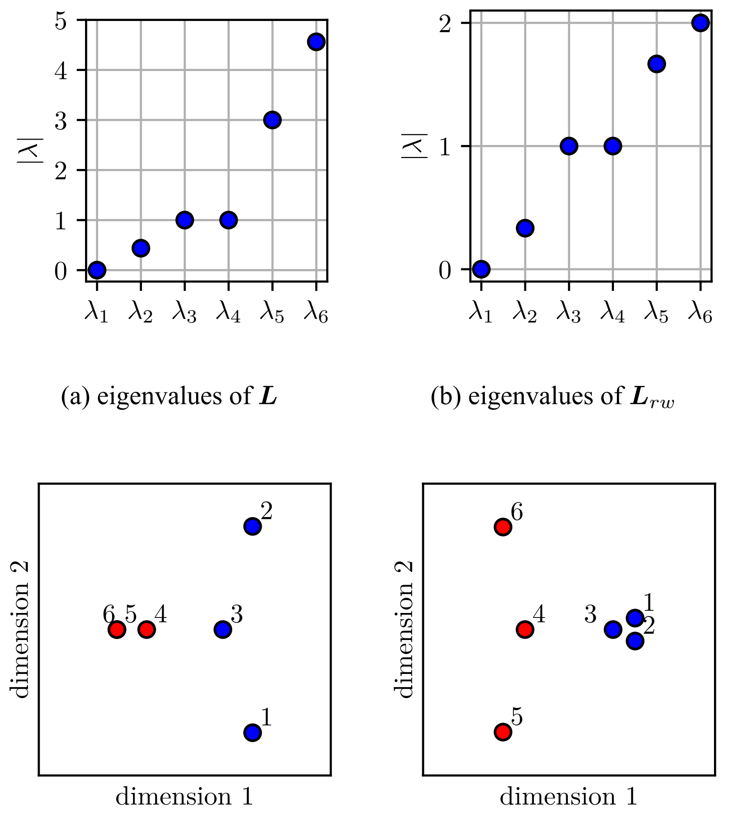
ii) *Decomposition:* calculate the eigenvalues and eigenvectors of the before constructed matrix and map each data point to a lower-dimensional representation based on one or more eigenvectors.

iii) *Grouping:* assign each data point to one of the k clusters concerning the before determined representation.

Spectral clustering algorithm: In general, spectral clustering can handle any arbitrary data where a non-negative symmetric measure expresses similarities between data points. Therefore, define $X = \{x_1, \ldots, x_n\}$ as a set of objects where n specifies the number of objects. To prepare for spectral clustering, transform the data set X into a similarity matrix $\boldsymbol{S} \in \mathbb{R}^{n \times n}$ by calculating the pairwise similarity measures between all data points. First, to perform spectral clustering, create a similarity graph based on one of the methods outlined in [14]. Further, transform the similarity graph into a weighted adjacency matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n}$. Subsequently, calculate the graph Laplacian $\boldsymbol{L} \in \mathbb{R}^{n \times n}$ on the weighted adjacency matrix $\boldsymbol{W}$. Afterwards, calculate the eigenvalues $\lambda_1, \ldots, \lambda_n$ and eigenvectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ of $\boldsymbol{L}$, and sort the eigenvectors in ascending order considering the absolute values of the eigenvalues. In Figure 2.8a the eigenvalues of the unnormalized Laplacian based on the data set shown in Figure 2.7a are visualized. Since the multiplicity of eigenvalues equal to zero defines the number of connected components, Figure 2.8a clearly presents that λ_1 equals to zero. Consequently, this implies that the data set consists of one single connected component. Although spectral clustering can determine the number of connected components by counting the number of eigenvalues equal to zero, the algorithm requires the parameter k number of clusters to perform clustering.

Since the goal of the example is to partition the data set into two clusters construct a matrix $\boldsymbol{U} \in \mathbb{R}^{n \times k}$ where the columns of the matrix comply with the first k eigenvectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$. To separate the data set define $\boldsymbol{y}_i \in \mathbb{R}^k, i = 1, \ldots, n$ as the i-th row of $\boldsymbol{U}$ representing the i-th data point. With this intention, each data point now lies in a two-dimensional embedding space presented in Figure 2.8c. Finally, applying the k-means algorithm on the transformed data points $\boldsymbol{y}_i$ leads to two distinct clusters visualized in red and blue color. In Figure 2.8c it is also apparent that nodes 1, 2 and 3 belong to a cluster as well as 4, 5 and 6. Comparing this result with the output of partitioning the graph with the Ncut approach in Figure 2.7c, it is visible that spectral clustering results in the same clustering output.

[4] NP-hard refers to solving a specific problem which results in non deterministic polynomial complexity [15].



(a) eigenvalues of $\boldsymbol{L}$ (b) eigenvalues of $\boldsymbol{L}_{rw}$

(c) clusters for $\boldsymbol{L}$ embedding (d) clusters for $\boldsymbol{L}_{rw}$ embedding

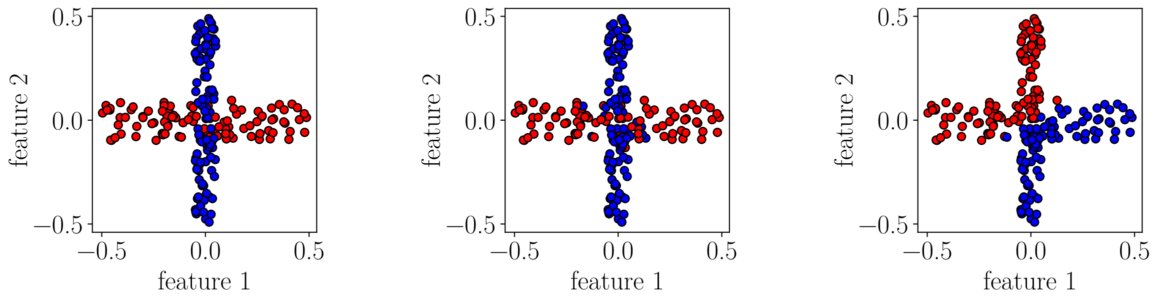
Figure 2.8: Visualization of intermediate values in spectral clustering:

Normalized spectral clustering algorithm In general, the normalized spectral clustering can be performed according to either using the symmetric graph Laplacian $\boldsymbol{L}_{sym}$ [16] or the random walk graph Laplacian $\boldsymbol{L}_{rw}$ [17]. Contrary to that, this section focuses only on the random walk graph Laplacian $\boldsymbol{L}_{rw}$. This decision is based on two properties. On the one hand, compared to $\boldsymbol{L}_{rw}$ the eigenvectors of $\boldsymbol{L}_{sym}$ contain the additional factor $D^{\frac{1}{2}}$, possibly leading to some artifacts. On the other hand, using $\boldsymbol{L}_{sym}$ does not provide any computational advantages [14]. Normalized spectral clustering starts, equally to the unnormalized version, by transforming data into a weighted adjacency matrix $\boldsymbol{W}$ and constructing the normalized graph Laplacian $\boldsymbol{L}$ thereof. Subsequently, the eigenvalues and eigenvectors are calculated by solving the generalized eigenproblem $\boldsymbol{L} \cdot \boldsymbol{u} = \lambda \cdot \boldsymbol{D} \cdot \boldsymbol{u}$. For this reason, the eigenvectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ correspond to the eigenvectors of the normalized graph Laplacian $\boldsymbol{L}_{rw}$. Since the example in Figure 2.8b is based on the previously described data set, it is also visible that only a single eigenvalue equals to zero, which leads to the conclusion of having one single connected component. The remaining steps are essentially the same as in the unnormalized case. The final clustering output of normalized spectral clustering based on the graph Laplacian $\boldsymbol{L}_{rw}$ is visualized in Figure 2.8d and correlates with clusters in Figure 2.8c.

In conclusion, spectral clustering has the advantage that it can handle any arbitrary data where the similarity between objects can be expressed by a similarity measure and does not require the data to be contained in a metric space. Furthermore, due to using graph Laplacian and standard linear algebra spectral clustering is a fast and powerful clustering approach. Since proximity is expressed as a similarity measure it often can succeed on clustering almost arbitrary shapes of data clusters where data in a cluster is somewhat connected by paths along close (w.r.t. the similarity measure) neighbors. For the reason of having a regular graph, meaning that vertices are connected equally, the unnormalized and the normalized approaches of spectral clustering deliver appropriate comparable outputs. Nevertheless, if vertex degrees highly differ, it is recommended to use the normalized version according to [14].

2.3.3 Sparse Subspace Clustering

This section describes the SSC approach, where the following content is based on [18, 19]. SSC is located in the field of clustering methods in the context of unsupervised learning. Albeit high-dimensional data commonly occurs in many real-world phenomena like image processing, the relevant information often lies in a union of low-dimensional subspaces. For example, multiple instances of handwritten digits can be transformed in a way such that varying rotations, translations, and thickness are lying in a union of low-dimensional subspaces. Therefore, subspace clustering follows the approach to unravel those hidden patterns by partitioning data according to their underlying subspaces. Moreover, SSC tries to accomplish finding subspaces by using the so called *self expressiveness* property of the data [18]. In detail, each data point can be expressed as a linear combination of other data points lying in the same subspace. Since this approach results in infinitely many solutions for representing a data point, SSC circumvents this problem by adding the constraint of getting a sparse solution. This approach has the advantage that a data point is represented by a few other data points lying in the same subspace.



(a) example data set (b) final clusters (c) final affine clusters

Figure 2.9: Visualization of Sparse Subspace Clustering example output: (a) shows the input data set. (b) visualizes the final output when performing SSC by solving (2.24). (c) presents the final clustering applying SSC by solving (2.24) with the additional constraint $\mathbf{1}^T \cdot \boldsymbol{C} = \mathbf{1}^T$.

First, to describe the operating principle of SSC, start with a synthetic example visualized in Figure 2.9a. This figure shows a data set consisting of 200 samples related to two different classes, red and blue, where each subset of samples is lying in its individual linear subspace. One can also imagine that each subspace can be represented by a line where each data point is lying on this line with some added noise. So the general goal of SSC is to find those subspaces which enable to divide the data set into two clusters. Therefore, define $\boldsymbol{S} = \{\boldsymbol{S}_1, \ldots, \boldsymbol{S}_l\}$ as an arrangement of l linear subspaces of $\mathbb{R}^D$, where $D = \{d_1, \ldots, d_l\}$ specifies the number of dimensions for each subspace. Further, denote

$$\boldsymbol{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n\}, \qquad \boldsymbol{y} \in \mathbb{R}^d \tag{2.19}$$

as a collection of n data points $\boldsymbol{y}$ lying in a d-dimensional space.

$$\boldsymbol{Y} = [\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_l] \cdot \boldsymbol{\Gamma} \tag{2.20}$$

describes a data matrix, where $\boldsymbol{Y}_l \in \mathbb{R}^{D \times n_l}$ is a rank-d_l matrix of $n_l > d_l$ data points lying in the corresponding subspace S_l. Furthermore, $\boldsymbol{\Gamma} \in \mathbb{R}^{n \times n}$ characterizes an unknown permutation matrix. Hence, the overall goal of any subspace clustering algorithm is to find the number of subspaces, a basis for each subspace, its dimensions, and the partitioning of the data from $\boldsymbol{Y}$. To accomplish this aim SSC uses the *self expressiveness* property, meaning that each data point in a subspace can be expressed as a linear combination of other data points of the same subspace. Therefore,

$$\boldsymbol{y}_i = \boldsymbol{Y} \cdot \boldsymbol{c}_i, \qquad c_{ii} = 0, \tag{2.21}$$

defines that the data point $\boldsymbol{y}_i$ is expressed as a linear combination of all other data points described by the vector $\boldsymbol{c}_i = [c_{i1}, \ldots, c_{iN}]^T$. To prevent the trivial solution of $\boldsymbol{y}_i$ being expressed as itself, the constraint $c_{ii} = 0$ is added. Since this approach still leads to infinitely many solutions, SSC circumvents this problem by minimizing the optimization function

$$min\|\boldsymbol{c}_i\|_1 \qquad s.t. \qquad \boldsymbol{y}_i = \boldsymbol{Y} \cdot \boldsymbol{c}_i, c_{ii} = 0. \tag{2.22}$$

As (2.22) contains the component-wise calculation of $\boldsymbol{c}_i$, this can be transformed into the matrix form

$$min\|\boldsymbol{C}\|_1 \qquad s.t. \qquad \boldsymbol{Y} = \boldsymbol{Y} \cdot \boldsymbol{C}, diag(\boldsymbol{C}) = 0, \tag{2.23}$$

where the i-th column of $\boldsymbol{C} = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n] \in \mathbb{R}^{n \times n}$ corresponds with the solution for i-th data point $\boldsymbol{y}_i$ represented as a linear combination of all other points. Those resulting vectors $\boldsymbol{c}_i$

are sparse, meaning that only a few non-zero values exist. Furthermore, each non-zero value's position expresses the current data point's subspace affiliation represented by the data points $\boldsymbol{y}_i$. Because (2.23) assumes having perfect data without any corruptions like noise or outliers let us introduce the optimization function[5]

$$min\|\boldsymbol{C}\|_1 + \lambda_e\|\boldsymbol{E}\|_1 + \frac{\lambda_z}{2}\|\boldsymbol{Z}\|_F^2 \qquad s.t. \qquad \boldsymbol{Y} = \boldsymbol{Y}\cdot\boldsymbol{C} + \boldsymbol{E} + \boldsymbol{Z}, diag(\boldsymbol{C}) = 0 \tag{2.24}$$

which can handle those nuisances. Accordingly, $\boldsymbol{E}$ corresponds to a matrix of sparse outlying entries, whereas $\boldsymbol{Z}$ characterizes the noise matrix. Additionally, the parameters λ_e and λ_z balance the influence of the three terms in (2.24). Moreover, if having no outliers or noise when solving the optimization problem, the corresponding parameters can be set to zero eliminating the related term. As there exist real-world problems, where data is lying in a union of affine rather than linear subspaces [19], adding the constraint $\boldsymbol{1}^T\cdot\boldsymbol{C} = \boldsymbol{1}^T$ to the optimization function

$$min\|\boldsymbol{C}\|_1 + \lambda_e\|\boldsymbol{E}\|_1 + \frac{\lambda_z}{2}\|\boldsymbol{Z}\|_F^2 \quad s.t. \quad \boldsymbol{Y} = \boldsymbol{Y}\cdot\boldsymbol{C} + \boldsymbol{E} + \boldsymbol{Z}, diag(\boldsymbol{C}) = 0, \boldsymbol{1}^T\cdot\boldsymbol{C} = \boldsymbol{1}^T \tag{2.25}$$

enables to handle affine subspaces. However, because linear subspaces are also affine subspaces adding the constraint in (2.25) can still deal with linear subspaces.

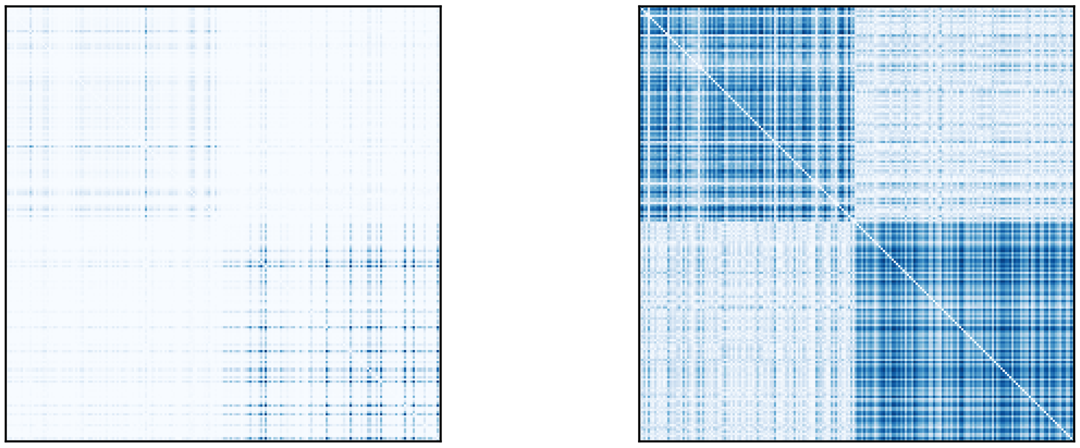
Since the described optimization functions (2.22), (2.23), (2.24), and (2.25) represent the relaxed versions of the corresponding sparse optimization programs as specified in [18, 19], they can be efficiently solved by using convex programming tools [20]. For additional information refer to [18, 19]. Subsequently, solving those convex optimization problems results in the matrix $\boldsymbol{C}$, representing each data point through a few data objects in the same subspace. Nevertheless, this does not provide symmetry for $\boldsymbol{C}$, such that $c_{ij} = c_{ji}$ is satisfied because each sample has its individual representation. As it is required to provide a symmetric affinity matrix $\boldsymbol{W}$ for the subsequent spectral clustering step, the proposed way is to denote

$$\boldsymbol{W} = |\boldsymbol{C}| + |\boldsymbol{C}|^T. \tag{2.26}$$

This results in a symmetric matrix where the weight of an edge connecting node i with node j is the same as connecting node j with node i. An optional step before building the similarity matrix is to consider normalization. Consequently, normalize each row of $\boldsymbol{C}$ before building a

[5] $||\boldsymbol{Z}||_F$ describes the Frobenius norm of the matrix $\boldsymbol{Z}$, which is defined by $||\boldsymbol{Z}||_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n z_{ij}^2} = \sqrt{tr(\boldsymbol{Z}^T\boldsymbol{Z})} = ||\boldsymbol{Z}(:)||_2$ [6].

similarity matrix $\boldsymbol{W}$ such that $\boldsymbol{c}_i \leftarrow \frac{\boldsymbol{c}_i}{\|\boldsymbol{c}_i\|_\infty}$ is satisfied[6]. In general, this normalization leads to higher robustness considering different norms of data points. This means, that data points with a large Euclidean norm represented by data points with small Euclidean norms result in large non-zero coefficients and vice versa. Since the subsequent spectral clustering algorithm focuses on keeping stronger connections in the graph, normalization ensures that the highest edge weights for all vertices are of the same scale [19]. With this intention, comparing Figure 2.10a and Figure 2.10b visualizes the influence of normalization on the similarity matrix $\boldsymbol{W}$, such that in-cluster connections provide equally scaled edge weights, where dark blue relates to high value or similarity and white represents zero or not similar either.



(a) similarity matrix (b) normalized similarity matrix

Figure 2.10: Visualization of similarity matrix in SSC: Both matrices are calculated on the example data set visualized in Figure 2.9a. (a) shows the unnormalized similarity matrix. (b) displays the normalized similarity matrix.

SSC's final step is to use the similarity matrix $\boldsymbol{W}$ to apply apply spectral clustering, which is defined in Section 2.3.2. Since the example data set in Figure 2.9a requires to partition the data set into two clusters, Figure 2.9b visualizes the final clustering where the optimization problem in (2.24) is solved. It is visible that the clustering algorithm can almost entirely find the initially designed clusters. In contrast, Figure 2.9c presents SSC's final output by solving (2.25) for affine subspaces. Consequently, it is apparent that the clustering output does not correspond to the labeling, as intended in Figure 2.9a. This grouping outcome is based on the characteristic that adding the constraint $\mathbf{1}^T \cdot \boldsymbol{C} = \mathbf{1}^T$ in (2.25) forces the connectivity of vertices by increasing large edge weights and decreasing small edge weight. Consequently, this results in two affine subspaces as visualized in Figure 2.9a.

[6] $\|\boldsymbol{v}\|_\infty = \max_i |v_i|$ is the max norm or the Chebyshev norm and defines finding the maximum absolute value in the vector $\boldsymbol{v}$ [21].

In conclusion, SSC is a powerful clustering algorithm that can find clusters in high-dimensional ambient space by unraveling hidden linear or affine low-dimensional subspaces. Since a data point is expressed via a linear combination of other data points lying in the same subspace, this algorithm can process new unseen data points. Accordingly, this enables to *train* SSC on a representative data set by calculating and storing intermediate values and subsequently determining the cluster affiliation of new unseen samples concerning the *trained* data set.

2.4 Model Evaluation

In general, the task of evaluating the output of a clustering model can be very challenging. This difficulty is based on the property that often the used data set does not provide any labeling. Therefore, different metrics follow the approach of matching set or peer-to-peer correlation or information theory [22]. Since this thesis focuses on clustering techniques where the used data set provides labeling, this section introduces a evaluation metric considering the ground truth.

As described in the introduction of Section 2, unsupervised learning generally aims to find hidden patterns and to unravel the underlying structure. In terms of clustering, those hidden patterns are used to partition the data set into a certain number of clusters. Depending on the clustering algorithm, the number of clusters is required as a mandatory input parameter. Consequently, to evaluate varying grouping results, it is necessary to use a metric that can compare different outputs. For example, Figure 2.11 illustrates a clustering output consisting of three clusters containing two different classes, red and blue.

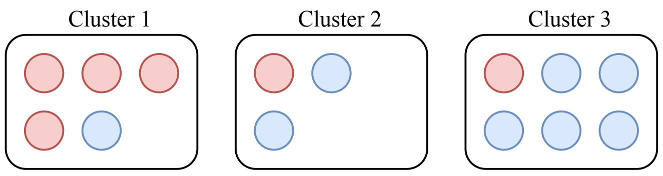


Figure 2.11: Clustering output example consisting of three clusters containing two different classes, red and blue.

Purity: A model evaluation metric that can handle such clustering output permutations is the purity measure, as explained in [6]. This metric results in a real value ranging from 0, which means insufficient purity to 1 corresponding to perfect purity. Therefore, define N as the total number of data points distributed over k clusters and N_i as the total number of objects in the i-th cluster. Furthermore, specify N_{ij} as the number of objects of the dominant class in the i-th cluster and describe the empirical distribution of class labels for the i-th cluster as

$$p_{ij} = \frac{N_{ij}}{N_i}. \tag{2.27}$$

Hence, finding the maximum value of p_i by solving $p_i = \max_j p_{ij}$ leads to the purity of the i-th cluster. Finally, the overall purity measure is denoted as

$$\text{purity} = \sum_{i=1}^{k} \frac{N_i}{N} p_i. \tag{2.28}$$

In other words, the purity measures the pureness of groups in the final clustering output concerning the classes in the ground truth. Referring to the example in Figure 2.11, calculating the purity measure results in $purity = \frac{5}{14} \cdot \frac{4}{5} + \frac{3}{14} \cdot \frac{2}{3} + \frac{6}{14} \cdot \frac{5}{6} = \frac{4+2+5}{14} = 0.786$. A drawback of this measure is that there exists no term which penalizes the number of clusters. For example, in terms of k-means clustering setting the number of clusters equal to the number of objects delivers the purity measure equal to 1.

3 Data Sources

In this section, three different Twitter data sources are presented, which are used in the experiments of this thesis. Since Twitter's Developer Policy[7] permits to share a collection of tweets including other properties than the tweet id, all content in the three downloaded data sets consists of tweet ids and related topics only. Therefore, each tweet id has to be enriched with additional information using the Twitter Representational State Transfer (REST) Application Programming Interface (API)[8]. Also, it has to be considered that all Twitter REST endpoints have some restrictions on the number of queries in a 15 minute window and limited access to the quantity of returned entries. Consequently, these properties can cause in a long time for retrieving information from Twitter and receiving incomplete data. An example, for being limited to the number of returned records is the *GET statuses/retweets/:id* endpoint. Here, only the 100 most recent retweets of a tweet are returned. It also has to be considered that Twitter's data basis is continuously changing, like adding, updating, and deleting users, tweets, and corresponding connections. Thus enriching tweet ids with additional tweet content via the Twitter REST API may fail.

Since all experiments in this thesis require the information of which user participated in retweeting, all corresponding additional information must be retrieved from the Twitter REST API. Apart from the tweet id and the tweet's label, this includes the user and the associated retweets. Furthermore, some algorithms rely on the information of who follows whom, which will be expressed as a follower graph, such that users who follow each other are represented as edges connecting vertices. In contrast to Twitter, where the follower data is modeled as a directed graph, this thesis accompanies designing the follower graph as an undirected graph, meaning that if only one of the two users follows the other, they are connected via an edge.

Since each tweet in the downloaded data sets consists of tweet id and label, the following steps are performed to retrieve additional Twitter data. First, enrich each tweet with the information of its user and the corresponding retweets. Second, use the information of the participating users to construct the follower graph. Third, store all downloaded data into a local database for further processing. With this intention, the following sections describe data set specific properties like the source, the number of tweets, the number of topics or labels, and the involved users.

3.1 Election Data Set

The election data set is a subset of the Twitter data collected during the United States' presidential elections in 2016 [23]. The original data set includes records collected between July 13, 2016, and November 10, 2016, via the Twitter REST API and the Twitter Stream API.

[7] https://developer.twitter.com/en/developer-terms/policy
[8] https://developer.twitter.com/en/docs/api-reference-index

Littman *et al.* [23] accumulated 280 million tweets partitioned in 12 different collections, considering different purposes. In contrast to the original data set, this thesis only uses data from the *democratic-party-timelines.txt* collection and the *republican-party-timelines.txt* collection. As each collection represents a different cluster affiliation, the related tweets are labeled accordingly. Based on this subset, all tweets, the corresponding retweets, and the user follower network is retrieved.

The downloaded files from [23], containing the tweet ids, consist of 21426 Democratic and 21871 Republican tweet ids, which totals 43297. Nevertheless, the total number of fetched tweets from the Twitter REST API on April 15, 2020 was reduced to 23983, as described in the introduction of this section. Accordingly, Table 1 represents the distribution of tweets and retweets via the corresponding labels. Furthermore, 154827 different users participated in the process of tweet creation and tweet propagation.

Topic number	Name	Tweets	Retweets
1	Republican Party	12273	257835
2	Democratic Party	11710	352212
	Total	23983	610047

Table 1: Distribution of tweets and retweets over labels in the Election data set

3.2 Auspol Data Set

Topic number	Name	Tweets	Retweets
1	#nbn	1212	2838
2	#qldpol	3182	9589
3	#uspoli	1380	1218
4	#springst	1227	5137
5	#lnp	2604	5633
6	#marriageequality	1983	4331
7	#insiders	2888	13968
8	#politas	1696	2734
9	#climatechange	800	2078
10	#stopadani	1022	4754
11	#qanda	837	1429
12	#turnbull	815	1133
13	#trump	1142	1602
	Total	20788	56444

Table 2: Distribution of tweets and retweets over labels in the Auspol data set

The Auspol data set is based on the tweet ids and the corresponding labels provided by [4]. All data in this data set has been initially collected between June 13, 2017, and September 2, 2017, from the Twitter streaming API by applying the hashtag filter *#Auspol*. The hashtag *#Auspol* is a frequently used label for any kind of political discussion in Australia. Curiskis *et al.* [4] preprocessed the streaming output by filtering the English language and adding topic labels associated with the tweets' hashtag. In total, they collected 29283 tweets distributed along 13 hashtag-related topics. Fetching additional properties from the Twitter REST API on August 4, 2020, finally reduced the total amount of tweets to 20788. The topic association and the number of tweets and retweets for each topic are presented in Table 2. Furthermore, 11864 different users participated in the process of tweet creation and tweet propagation.

3.3 RepLab Data Set

The RepLab data set is also provided by [4] and contains tweet ids and the corresponding labels. The initial data set is based on a competitive evaluation exercise for Online Reputation Management systems [24]. Since the original data set provides 1263 different topics Curiskis *et al.* [4] preprocessed all data resulting in 2657 tweets distributed over 13 topics. Out of the data provided by [4], it was possible to retrieve 2444 tweets from the Twitter REST API on August 5, 2020. Table 3 presents the distribution of tweets and retweets over the corresponding labels. Additionally, 14431 different users participated in the process of tweet creation and tweet propagation.

Topic number	Name	Tweets	Retweets
1	For Sale	339	277
2	Jokes	128	1428
3	User Comments	260	381
4	Spam	117	7
5	MotoGP - User Comments	99	292
6	Ironic Criticism	106	961
7	Criticism	107	831
8	Nice comments from fans	95	401
9	For Sale - Nissan Cars & Parts & Accessories	124	20
10	Suzuki cup	290	2514
11	money laundering / terrorism finance	196	960
12	Sports sponsors	118	746
13	Princeton Offense	126	1055
14	Fan Craze - Beliebers	142	826
15	Record of views on YouTube	197	2057
	Total	2444	12756

Table 3: Distribution of tweets and retweets over labels in the RepLab data set

4 Clustering Twitter Trajectories

Classical Natural Language Processing (NLP) methodology often only uses the text-based content of tweets to perform clustering and/or classification tasks. However, these algorithms can suffer from the short length of tweets and the subtleties of the used language [25]. Therefore, this thesis aims to identify pathways of (re-)tweets in the Online Social Network (OSN) Twitter by representing each tweet via its retweet behavior and building clusters thereof. Since the retweet information is based on the participating users, using this information may reveal some exciting properties.

In detail, this chapter focuses on the methodology to perform experiments based on the retweet information. The key idea is that different types of tweets (e.g. sports tweets or political tweets) are shared differently in the OSN. Therefore, each tweet is represented as a retweet vector considering the users who participated in propagating this tweet. While this implies that tweets can be clustered based on their propagation paths, it is unclear if the resulting clusters serve other purposes than understanding the typical paths of (re-)tweets. Since all used data sets provide corresponding labels (see Section 5), the clustering output is evaluated concerning the given topic affiliation. In particular, the project investigates possible relationships between propagation clusters on the one hand and tweet topics on the other hand. The stronger the relationship, the more the clusters will be helpful in topic analysis, especially in the case of limited or multi-language data.

With this intention this chapter is structured as follows. First, feature extraction describes methods used to represent each tweet as a retweet vector and construct a follower adjacency matrix. Second, exploratory data analysis presents algorithms that are used to get deeper insights into the given data. This enables strong interpretation background to the final partitioning of a data set concerning the different used methods. Third, three clustering approaches are presented, where two algorithms are based on the k-means approach described in Section 2.3.1, and one is based on the SSC approach introduced in Section 2.3.3. Fourth, the evaluation section contains a method to compare clustering output based on a given topic label.

4.1 Feature Extraction

As this thesis aims to cluster data using the retweet information, this section describes the preprocessing steps and usage of an observation model to transform each tweet into a d-dimensional vector representation. Furthermore, creating the follower adjacency matrix based on participating users and their relationship is presented. In general, retweeting is one of the propagation mechanisms in Twitter to share and distribute information over the network. Thus, the main goal is to represent each of n tweets and the tweet/retweet behavior of p users as binary vectors. These vectors contain the information about which accounts were involved in creating or retweeting a tweet.

Preprocessing: In the context of this thesis, preprocessing aims to clean up the data set. Basically, this process can be divided into three steps, determining the top 2 labels, identifying the top 100 user accounts, and filtering the data accordingly. In the first step, the tweets are grouped by their label and the number of (re-)tweets in each group is counted. Subsequently, the labels are sorted according to their (re-)tweet count in descending order. Since there exists a varying number of labels in each data set, only the top 2 labels are considered for further processing. In the second step, the number of (re-)tweets each user participated in is determined and sorted in descending order to the tweet count. As there are also different numbers of user accounts in each data set, only the top 100 users are selected. Besides, the sorting of the top 100 accounts is remembered to construct the retweet vectors and the follower adjacency matrix. In the third step, only (re)tweets are selected, which are related to one of the top 2 labels and the top 100 user accounts. This filtering finally results in the preprocessed data set, which is further used to construct the retweet vector representations.

Tweet Representation: To map tweets into binary vectors, an observation model based on the preprocessed data set is used. In detail, for tweet $i \in \{1, \dots, n\}$ observe if person $j \in \{1, \dots, p\}$ is involved in retweeting or creating tweet i. Since preprocessing delivers only the top 100 users, the parameter p equals 100. Accordingly, data is defined as

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_n \end{bmatrix}, \qquad \boldsymbol{x} \in \{0,1\}^p \tag{4.1}$$

such that

$$x_{ij} = \begin{cases} 1, & \text{if person } j \text{ created or retweeted } i \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

is satisfied. The case $x_{ij} = 1$ represents that person j retweeted or created tweet i, whereas $x_{ij} = 0$ signals that user j does not interact with the i-th tweet. Finally, this representation is used as input in the experiments.

Follower adjacency matrix: As introduced before, a further step in the context of feature extraction is to create a matrix that represents the information of who follows whom. Therefore, use all users, who remain after preprocessing, and construct an undirected graph based on connecting two vertices if at least one of both users follows the other. Then a symmetric matrix

$\boldsymbol{A} \in \{0,1\}^{p \times p}$ representing the undirected graph and considering the order of users is built such that

$$a_{ij} = \begin{cases} 1, & \text{if } i \text{ follows } j \text{ or } j \text{ follows } i \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

is satisfied.

4.2 Exploratory Data Analysis

In this section, some methods in terms of exploratory data analysis are explained, applied, and visualized. All following visualizations are based on the data sets after applying feature extraction. Thus, only retweet vectors are considered, which belong to the top 2 labels and the top 100 user accounts. First, the follower adjacency matrix of the top 100 users of each data set is shown. Second, the number of tweets each of the top 100 users participated grouped by labels is presented. Third, PCA is applied to each data set and visualized concerning each principal component's cumulative percentage of variance. Fourth, the t-SNE algorithm is used to perform a dimensionality reduction onto two dimensions. The t-SNE output is then shown in a two-dimensional plot.

Connectivity of top 100 users via follower graph: In Figure 4.1 three plots are shown which present the connectivity of user accounts considering the follower graph. In other words, dark blue in each figure represents that user i follows user j. Since these plots visualize the top 100 users follower information, each figure has 100 rows and columns. It is visible that the Auspol data set contains the densest follower graph, whereas in the RepLab data set only a few users follow each other.

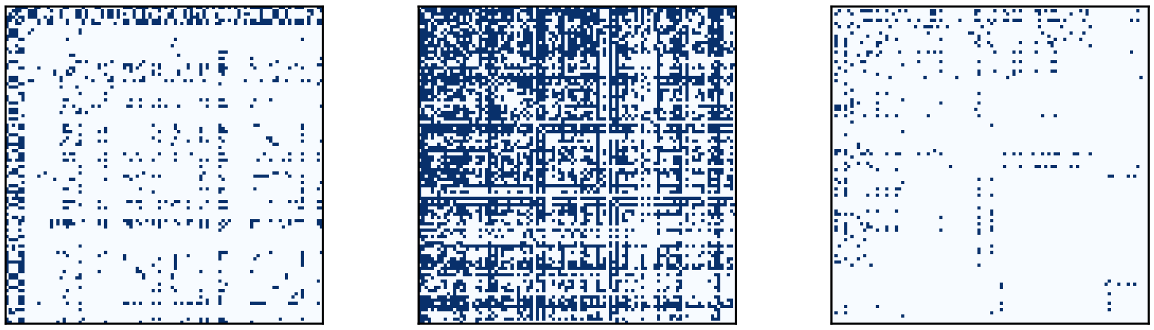
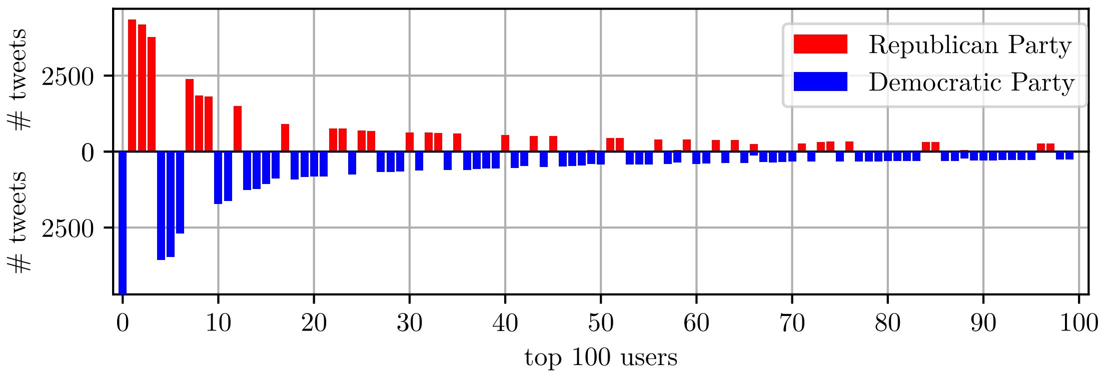
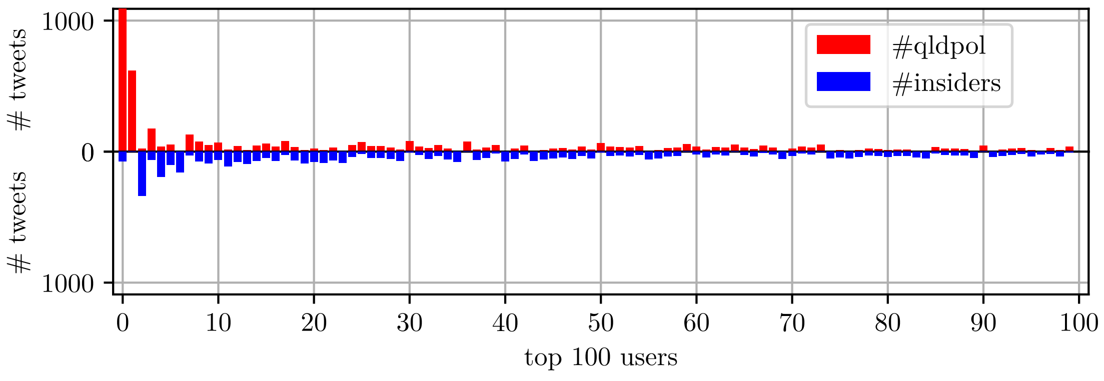

(a) Election data set (b) Auspol data set (c) RepLab data set

Figure 4.1: Follower adjacency matrix visualization of all data sets

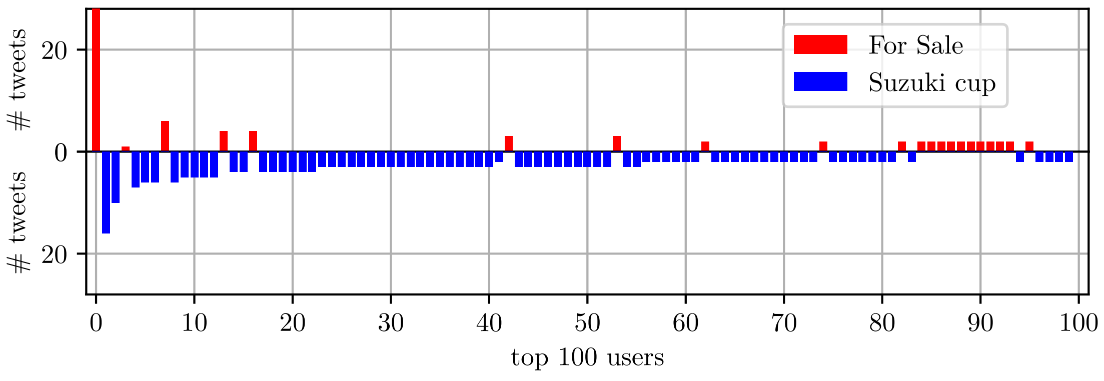
User participation per label Figure 4.2 presents the number of tweets a user participated grouped by labels. Comparing all three data sets, it is evident that in the Election data set and the RepLab data set users mainly interact with tweets belonging to the same label or topic. At the same time, users in the Auspol data set participate in tweeting and retweeting in both topics with a preference for a single topic.



(a) Election data set



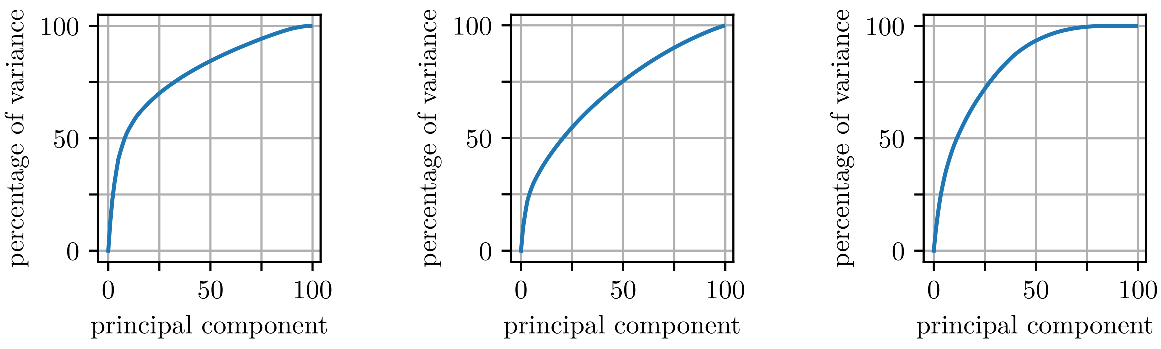
(b) Auspol data set



(c) RepLab data set

Figure 4.2: Visualization of user participation per label

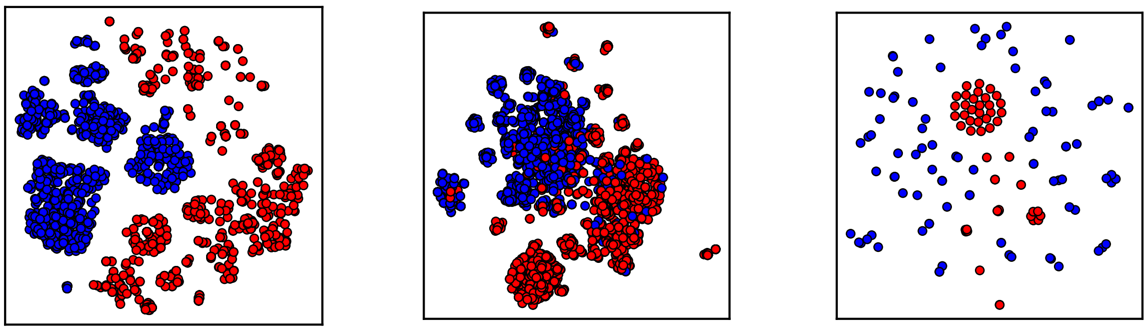
Principal Component Analysis As described in Section 2.2.2, PCA is a method to determine data set specific properties and perform dimensionality reduction under consideration of those characteristics. In the context of this thesis, PCA is used to calculate the percentage of variance in each dimension. With this intention, one can get a sense of how data is distributed over the dimensions, which helps to interpret if dimensionality reduction can improve clustering output. For example, imagine an extreme case where the first 5 of 100 principal components contain 100 percent of the total variance. This implies that reducing dimensions to 5 principal components would have a significant influence on the curse of dimensionality (see Section 2.2.1), and can enable basic clustering algorithms like the k-means algorithm to perform well. In contrast, if each principal component contains almost the same amount of variance, it is very unlikely that dimensionality reduction can help in improving the clustering output. However, PCA is used to calculate the eigenvalues of the covariance matrix of the retweet representations. Since PCA provides the eigenvalues in descending order, plotting the eigenvalues as a curve helps to interpret how data is distributed considering the variance in the principal components. Figure 4.3 visualized the cumulative percentage of variance over the principal components.



(a) Election data set (b) Auspol data set (c) RepLab data set

Figure 4.3: Visualization of cumulative percentage of variance over principal components

Visualize Data with t-SNE t-SNE represents another technique to get a sense if data contains the information for being separated into a certain number of clusters. Therefore, t-SNE is applied on the retweet vectors to reduce its dimensionality to two dimensions and to visualize the data accordingly. Since each record in the data sets has a related real label, adding a particular label color to each entry in the visualization helps to interpret if visible groups represent a useful clustering. It is worth mentioning that t-SNE, in general, tries to present small distances in the ambient space also as small distances in the output space (see Section 2.2.3). On the one hand, if t-SNE provides good clustering output, this means that the data contains the information to partition records into groups according to their topic affiliation. On the contrary, if there exists no useful clustering such that groups include multiple labels, this does not imply that the data does not contain the information needed to cluster the data accordingly. Figure 4.4 visualizes the output of t-SNE with respect to labels of the samples. The Election data set and the RepLab data set show almost *clean* clusters such that each visible group only contains one color. In contrast the Auspol data set has a strong mixture of different labels. Nevertheless, this does not imply that the clustering of the data into two groups is not possible.



(a) Election data set (b) Auspol data set (c) RepLab data set

Figure 4.4: Clustering output of t-SNE for all data sets

4.3 Methods for Clustering Data

As the goal of this thesis is to cluster data according to the retweet vector representations, this section focuses on three different clustering algorithms. Therefore, this section is structured into three parts, presenting those clustering methods. First, the k-means algorithm with the usually used Euclidean distance is described. Second, a new distance measure considering the follower information is introduced, which replaces the standard Euclidean distance in the k-means algorithm. Third, the sparse subspace clustering algorithm is shown.

4.3.1 Method 1: k-means with euclidean distance

In general, k-means is a very commonly used algorithm in clustering data in an unsupervised manner, as introduced in Section 2.3.1. After the initialization of cluster centers, k-means essentially uses a distance measure to calculate the distance of the data points to its cluster centers and update the cluster centers according to the data points assigned to them. The standard distance measure for k-means is the Euclidean distance which is denoted as

$$d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|. \tag{4.4}$$

This metric has the advantage that it is defined for every positive number of dimensions. However, since this method represents the standard k-means implementation, it is used as a comparative measure for the other algorithms.

4.3.2 Method 2: k-means with follower distance

Based on the standard implementation of k-means as described in Section 2.3.1 this section describes how another distance measure can help to improve clustering results. The goal of the k-means with follower distance is to exchange the standard Euclidean distance with a so-called follower distance. This new distance measure aims to consider the information of who follows whom in the context of calculating the distance between two data points. To describe the intention of this new distance measure, an example is used.

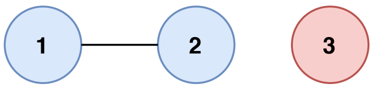


Figure 4.5: Visualization of an example follower graph

In Figure 4.5 an undirected graph with two groups, blue and red, is visualized. Transforming the before mentioned graph into a follower adjacency matrix results in

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

where $a_{ij} = 1$ signals a connection between node i and j and $a_{ij} = 0$ describes that vertex i and j are not connected either. Further, consider the three very sparse vectors in (4.6), which represent a retweet vector where the value 1 signals that user i is involved in retweeting, and a 0 represents no interaction. Those vectors are denoted by

$$\boldsymbol{x} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \boldsymbol{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \boldsymbol{z} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{4.6}$$

However, using the Euclidean metric from (4.4) to calculate the distance between any combination of $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{z}$ results in

$$d(\boldsymbol{x}, \boldsymbol{y}) \approx d(\boldsymbol{x}, \boldsymbol{z}) \approx d(\boldsymbol{y}, \boldsymbol{z}) \approx 1.41. \tag{4.7}$$

It is obvious that the distance between each pair of vectors equals the same value of approximately 1.41. Consequently, if data is very sparse k-means with the Euclidean distance can not find useful clusters. To circumvent this issue the follower adjacency matrix is used for calculating the distance such that vectors have a small distance if their corresponding users are connected in the follower graph. Accordingly, denote the follower distance measure as

$$d(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{A}) = e^{-\boldsymbol{x}^T \cdot \boldsymbol{A} \cdot \boldsymbol{y}} \tag{4.8}$$

which causes values between 0 and 1. Consequently, calculating the follower distance measures between all combinations of $\boldsymbol{x}, \boldsymbol{y}$, and $\boldsymbol{z}$ results in

$$\begin{aligned} d(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{A}) &\approx 0.37, \\ d(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{A}) &= 1.0, \\ d(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{A}) &= 1.0. \end{aligned} \tag{4.9}$$

However, at first glance, it seems that the retweet vectors in (4.6) have nothing in common. But the outcomes in (4.9) clearly show that using the follower graph as additional information can substantially improve the distance between retweets originating from the same cluster.

4.3.3 Method 3: Sparse Subspace Clustering

The third method to cluster retweet vectors is based on the SSC approach described in Section 2.3.3. This algorithm aims to represent each vector as a linear combination of other vectors in the same subspace concerning sparsity. In other words, as few data points in the same subspace as possible are used to represent the current data point as a linear combination. Subsequently, those representations are used by the spectral clustering algorithm to partition the data set into a before defined number of groups. This algorithm has the advantage that it can cluster data that is located in subspaces of any shape.

4.4 Evaluation

To evaluate the different clustering outcomes' performance, the purity measure, as described in Section 2.4 is used. This evaluation metric requires labels to compute each clustering output's purity, which is satisfied for all three data sets used in this thesis. In general, this metric has a maximum value of 1.0, representing a perfect clustering output in terms of purity. In contrast, the minimum value of this metric varies by the number of clusters and the number of labels per cluster.

4.5 Experiments

In this section, experiments based on the data sets introduced in Section 3, and the methods described in Section 4.3 are presented. Therefore, Figure 4.6 visualizes the steps performed for each experiment. The white rectangles present the intermediate data, whereas the blue rectangles signal processing steps such as feature extraction, clustering, and cluster evaluation. The goal of the experiments is to determine the performance of the different clustering algorithms on varying data sets. To get comparable results, each of the three experiments is performed equally, such that the input data is only based on the top 2 labels and the top 100 users. Additionally, the number of clusters for each clustering method is set to $k = 2$, which means to cluster data into two groups. To investigate the stability of each clustering algorithm in terms of getting reproducible clustering outcomes, the whole data set is split into equally sized chunks depending on the number of samples in the used data set. Finally, the purity measure is calculated on each grouping outcome, enabling comparing distinct runs and different clustering algorithms.

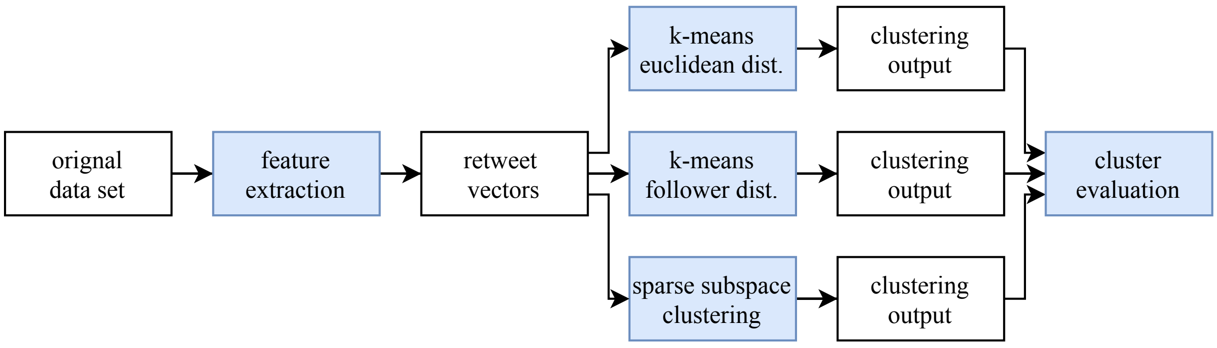


Figure 4.6: Visualization of experiment pipeline for each data set

4.5.1 Experiments on Election data

The first experiment is based on the Election data set. Since this data set contains only the two labels *Republican Party* and *Democratic Party*, preprocessing performs only filtering for (re-)tweet vectors that belong to the top 100 users. Table 4 shows the number of retweet vectors per label after the feature extraction process.

Label	# Retweet vectors
Republican Party	12273
Democratic Party	11710
Total	23983

Table 4: Number of retweets per label in the preprocessed Election data set

Since this data set finally contains 23983 retweet vectors, it is randomly split into 20 equally sized chunks with approximately 1200 samples. Each chunk is then used as input for all three clustering algorithms, visualized in Figure 4.6. Finally, the purity measure is calculated on each clustering outcome and documented in a boxplot.

4.5.2 Experiments on Auspol data

The second experiment is based on the Auspol data set. This data set contains initially 13 different labels, which are reduced to the top 2 labels *#qldpol* and *#insiders* in the feature extraction step. Correspondingly, Table 5 shows the number of retweet vector representations of the top 2 labels.

Label	# Retweet vectors
#qldpol	2244
#insiders	1741
Total	3985

Table 5: Number of retweets per label in the preprocessed Auspol data set

As this data set contains 3982 retweet vectors, it is randomly split into four equally sized chunks with approximately 1000 samples. As described in Section 4.5.1 each chunk is then used as input for all three clustering algorithms. Finally, the purity measure is calculated on each clustering outcome and documented in a boxplot.

4.5.3 Experiments on RepLab data

The final and third experiment is based on the RepLab data set. This data set originally contains 15 different labels which are reduced to the top 2 labels *For Sale* and *Suzuki cup* in the feature extraction step. Accordingly, Table 5 shows the number of retweet vectors of the top 2 labels.

Label	# Retweet vectors
For Sale	53
Suzuki cup	97
Total	150

Table 6: Number of retweets per label in the preprocessed RepLab data set

Since this data set contains only 150 retweet vectors in total, the whole data set is reused four times. In contrast to the experiments in Section 4.5.1 and Section 4.5.1 this procedure determines the stability of the clustering algorithms in terms of random initialization. Since each of the used clustering methods finally uses the k-means algorithm (see Section 2.3.1 and Section 2.3.3) for clustering data, random initialization can play an important role. Finally, the purity measure is calculated on each of the four clustering outcomes and documented in a boxplot accordingly.

5 Results

In this section, all results of the experiments explained in Section 4.5 are visualized and described. Therefore, a boxplot presents the overall performance measures for each clustering approach on a single data set. Moreover, a confusion plot for each clustering method concerning the best clustering output of all data chunks displays grouping specific performance. Therefore, this section is structured into three parts related to experiments on the Election data set, the Auspol data set, and the RepLab data set.

5.1 Clusters on Election data

Figure 5.1 presents the purity measure for each clustering algorithm on 20 different data chunks of approximately 1200 samples of the Election data set. The method k-means with Euclidean distance shows purity values ranging from 0.54 to 0.65, with a mean of 0.58 and a median of 0.57. The k-means algorithm with follower distance and the SSC approach results in purity measures from 0.999 to 1.0. Besides, both methods also have the same mean value of 0.999 and the same median value of 1.0. Consequently, the k-means method with Euclidean distance has a considerably lower purity measure than the others.

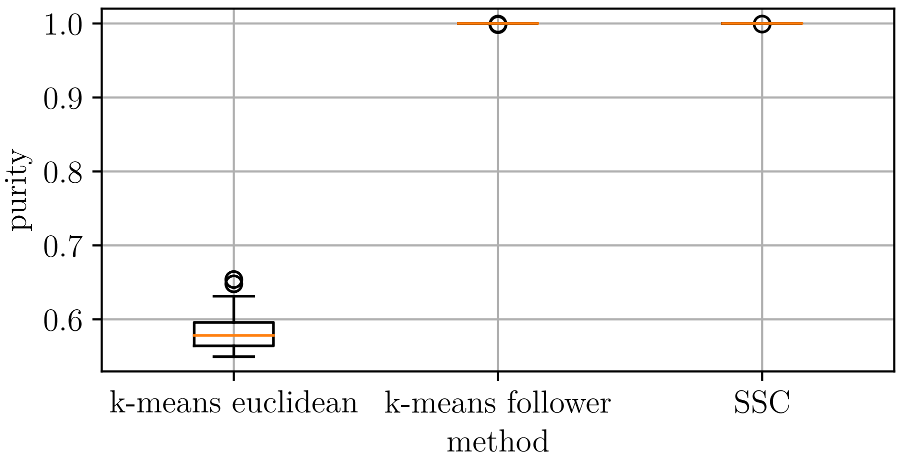
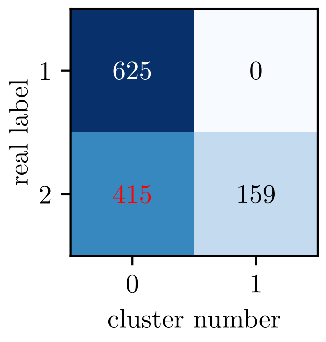
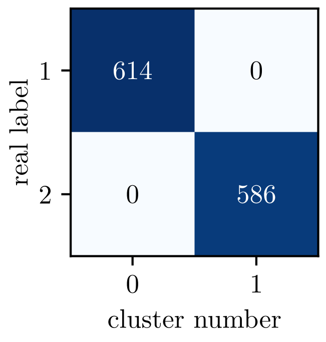


Figure 5.1: Purity measures of the three clustering methods on the Election data set

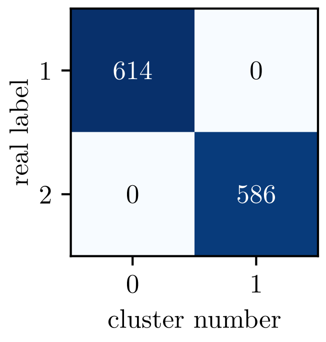
Figure 5.2 shows the confusion plots for each clustering method of the Election data set for the best clustering output. Since the best clustering outputs are based on different data chunks, the number of samples and their cluster affiliation in the confusion plot varies. Figure 5.2a displays that cluster 0 contains 415 misclassified data points, which should belong to cluster 1. In contrast, Figure 5.2b and Figure 5.2c present an optimal clustering output such that all samples which belong to the same label are grouped together.



(a) k-means Euclidean distance



(b) k-means follower distance



(c) SSC

Figure 5.2: Comparison of confusion matrices calculated on the best clustering output for each clustering algorithm on the Election data set: In (a), (b), and (c) the label 1 corresponds to *Republican Party* topic, and the label 2 relates to *Democratic Party* topic.

5.2 Clusters on Auspol data

In Figure 5.3, the purity measure on the Auspol data set is presented. In this data set, each clustering algorithm was applied on four different data chunks of approximately 1000 samples. The method k-means with the Euclidean distance shows purity values ranging from 0.541 to 0.577, a mean of 0.565, and a median of 0.571. The k-means algorithm with follower distance reached a minimum of 0.541, a maximum of 0.575, a mean of 0.563, and a median of 0.568. Also, the SSC delivers almost equal values such that 0.541 represents the minimum, 0.575 is the maximum, 0.563 equals the mean, and 0.568 is the median. With this intention, all three algorithms perform almost equally in terms of the purity measure.

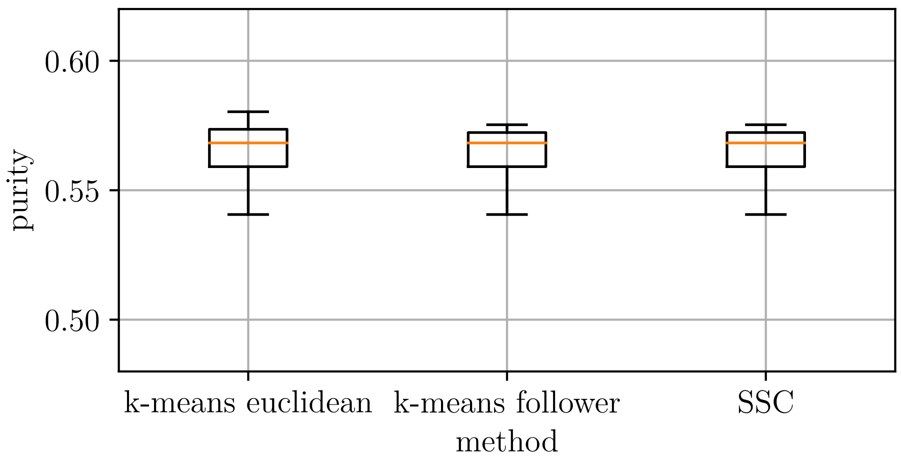
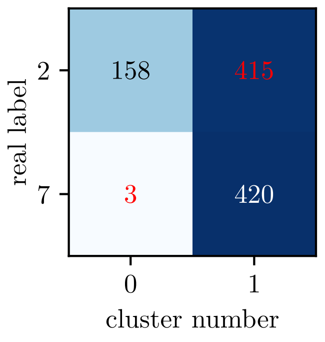


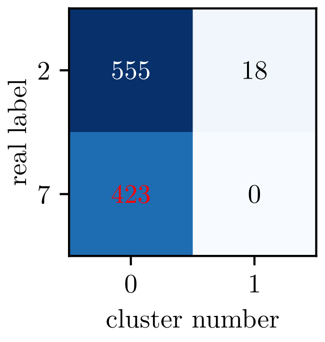
Figure 5.3: Purity measures of the three clustering methods on the Auspol data set

Figure 5.4 visualizes the confusion plots of the best clustering output of each clustering method of the Auspol data set. All three figures show misclassified samples ranging from 418 to 423. These numbers indicate that approximately 45 percent of all samples are partitioned wrong.

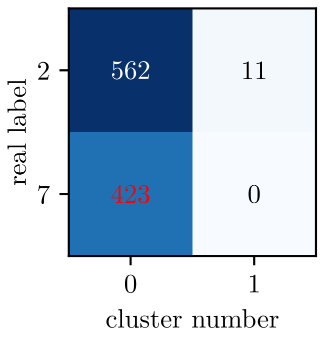
However, Figure 5.4b and Figure 5.4c differ from Figure 5.4a such that almost all samples are contained in cluster 0.



(a) k-means Euclidean distance



(b) k-means follower distance



(c) SSC

Figure 5.4: Comparison of confusion matrices calculated on the best clustering output for each clustering algorithm on the Auspol data set: In (a), (b), and (c) the label 2 corresponds to *#qldpol* topic and the label 7 relates to *#insiders* topic.

5.3 Clusters on RepLab data

Figure 5.5 presents each clustering algorithm's purity measure on four runs of the same RepLab data set containing 150 samples. The method k-means with the Euclidean distance presents a purity measure ranging from 0.64 to 0.66, a mean and a median of 0.65. The k-means algorithm with the follower distance shows a constant value of 0.93 for the minimum, maximum, mean, and median. Similarly, the SSC method has a constant value for the minimum, the maximum, the mean, and the median of 0.83. In Figure 5.5, it is also visible, that k-means with follower distance has the highest purity measure followed by the SSC algorithm and the k-means with the Euclidean distance.

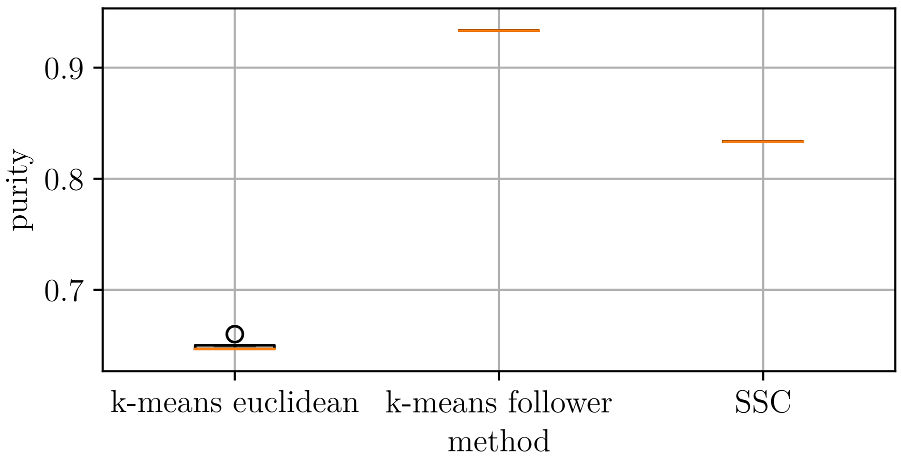
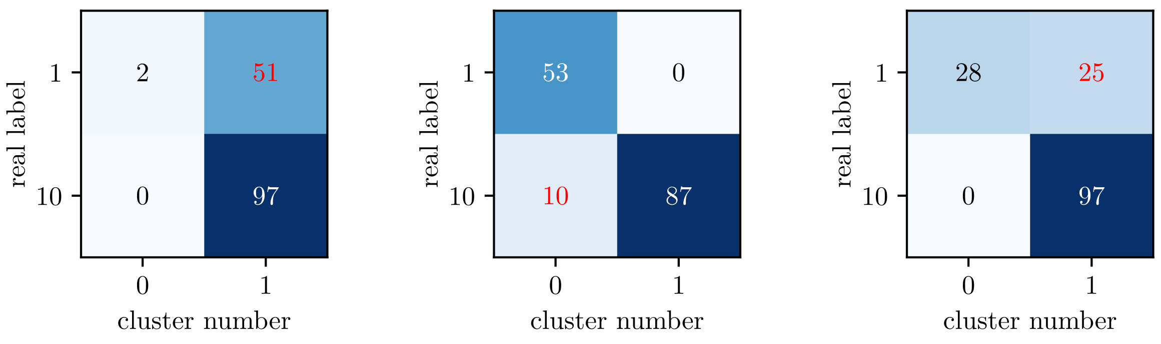


Figure 5.5: Purity measures of the three clustering methods on the RepLab data set

In Figure 5.6 the confusion plots for the best clustering output of each clustering method of the Election data set are presented. It is visible that k-means with the Euclidean distance separates almost all data points into a single cluster shown in Figure 5.6a. On the contrary, since k-means with the follower distance has the highest purity measure, the corresponding confusion plot presents only ten misclassified samples displayed in Figure 5.6b. Consequently, this method correctly found all samples belonging to the topic *For Sale*. In contrast, SSC misclassified 25 data points, whereas all samples belonging to the topic *Suzuki cup* were detected accurately. The corresponding confusion matrix is visualized in Figure 5.6c.



(a) k-means Euclidean distance (b) k-means follower distance (c) SSC

Figure 5.6: Comparison of confusion matrices calculated on the best clustering output for each clustering algorithm on the RepLab data set: In (a), (b), and (c) the label 1 corresponds to *For Sale* topic and the label 10 relates to *Suzuki cup* topic.

6 Conclusion

This report concludes the author's research conducted from March to August 2020 at the Center of Statistics and Computer Science at the University of Chicago. Due to the covid-19 pandemic, the cooperation was carried through as a series of weekly online-meetings. A final presentation and visit is planned once the situation allows travelling. Based on the theoretical background in Section 2, three different clustering approaches are introduced to cluster tweets concerning their retweet vector representations. Furthermore, the performance of the clustering algorithms is investigated under consideration of three different data sets. The experiments exhibit that the clustering algorithm's choice and the underlying data set have a massive influence on the final clustering output. The results present that k-means with the Euclidean distance is not suitable to perform clustering on such sparse data as the retweet vector representation offers. On the contrary, k-means with the follower distance can group tweets via their retweet vectors by calculating the distance measure under consideration of the underlying Twitter follower graph. Equally, SSC delivers suitable clusters using the retweet vector representations by exploiting their self-expressiveness property.

References

[1] R. Nugroho *et al.*, “A survey of recent methods on deriving topics from Twitter: algorithm to evaluation,” *Knowl. Inf. Syst.*, volume 62, number 7, pages 2485–2519, 2020.

[2] M. E. Larsen *et al.*, “We Feel: Mapping Emotion on Twitter,” *IEEE J. Biomed. Heal. Informatics*, volume 19, number 4, pages 1246–1252, 2015.

[3] B. Resch, F. Usländer, and C. Havas, “Combining machine-learning topic models and spatiotemporal analysis of social media data for disaster footprint and damage assessment,” *Cartogr. Geogr. Inf. Sci.*, volume 45, number 4, pages 362–376, 2018.

[4] S. A. Curiskis, B. Drake, T. R. Osborn, and P. J. Kennedy, “An evaluation of document clustering and topic modelling in two online social networks: Twitter and Reddit,” *Inf. Process. Manag.*, volume 57, number 2, page 102 034, 2020.

[5] C. Bishop, *Pattern recognition and machine learning*. New York: Springer, 2006.

[6] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge: MIT Press, 2012.

[7] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd edition. Waltham: Morgan Kaufmann, 2012.

[8] Y. LeCun, C. Cortes, and C. J. Burges. (2010). “MNIST handwritten digit database,” [Online]. Available: `http//yann.lecun.com/exdb/mnist` (visited on 08/04/2020).

[9] J. Shlens, “A Tutorial on Principal Component Analysis,” *CoRR*, volume abs/1404.1, 2014.

[10] L. Papula, *Mathematik für Ingenieure und Naturwissenschaftler Band 2*, 14th edition. Wiesbaden: Springer Fachmedien, 2015.

[11] v. d. M. Laurens and H. Geoffrey, “Visualizing Data using t-SNE,” *J. Mach. Learn. Res.*, volume 9, pages 2579–2605, 2009.

[12] L. Göllmann *et al.*, *Mathematik für Ingenieure: Verstehen – Rechnen – Anwenden*. Berlin, Heidelberg: Springer, 2017.

[13] D. Arthur and S. Vassilvitskii, “K-means++: The advantages of careful seeding,” *Proc. Annu. ACM-SIAM Symp. Discret. Algorithms*, pages 1027–1035, 2007.

[14] U. von Luxburg, “A tutorial on spectral clustering,” *Stat. Comput.*, volume 17, number 4, pages 395–416, 2007.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability - A Guide to the Theory of NP-completeness*. Murray Hill, New Jersey: W. H. Freeman, 1979.

[16] Jianbo Shi and J. Malik, “Normalized cuts and image segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, volume 22, number 8, pages 888–905, 2000.

[17] A. Y. Ng, M. I. Jordan, and Y. Weiss, “On Spectral Clustering: Analysis and an algorithm,” in *Adv. Neural Inf. Process. Syst. 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds., MIT Press, 2002, pages 849–856.

[18] E. Elhamifar and R. Vidal, “Sparse subspace clustering,” in *2009 IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pages 2790–2797.

[19] E. Elhamifar and R. Vidal, “Sparse Subspace Clustering: Algorithm, Theory, and Applications,” *IEEE Trans. Pattern Anal. Mach. Intell.*, volume 35, number 11, pages 2765–2781, 2013.

[20] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge: Cambridge University Press, 2004.

[21] J. E. Gentle, *Matrix Algebra: Theory, Computations, and Applications in Statistics*. New York: Springer, 2007.

[22] J.-O. Palacio-Niño and F. Berzal, “Evaluation Metrics for Unsupervised Learning Algorithms,” 2019.

[23] J. Littman, L. Wrubel, and D. Kerchner, *2016 United States Presidential Election Tweet Ids*, 2016.

[24] E. Amigó *et al.*, “Overview of RepLab 2013: Evaluating Online Reputation Monitoring Systems,” in *Proc. Fourth Int. Conf. CLEF Initiat.*, 2013, pages 333–352.

[25] A. Chinnov *et al.*, “An Overview of Topic Discovery in Twitter Communication through Social Media Analytics,” in *Proc. 21st Am. Conf. Inf. Syst.*, Puerto Rico, 2015.