Evaluating Clustering Algorithms and Autoencoders for Segmenting Customers in the Tourism Domain

Author
Mark Riezebos
450121

Bachelor Thesis Econometrics and Operations Research*

Supervisor
Utku Karaca

Second assessor
Anoek Castelein

Erasmus University Rotterdam
Erasmus School of Economics

July 7, 2019

*The views stated in this thesis are those of the author and not necessarily those of Erasmus School of Economics or Erasmus University Rotterdam.

Abstract

Nowadays, recommender systems are used to give customers personalized recommendations on where to go on vacation, based on social media data. However, the collaborative filtering algorithms which are used by these recommender systems face scalability problems because of the exponential growth in the amount of social media data. Therefore, in this thesis we compare different clustering algorithms in order to segment customers in the tourism domain. We also use autoencoders as a dimensionality reduction technique for the considered data sets, with the aim of filtering out noise in the data and improving the performance of the clustering algorithms. The results show that the k-means algorithm, the k-medoids algorithm, and the hierarchical clustering algorithm with either complete or Ward linkage are the clustering algorithms which in general perform the best. Which of these algorithms to use depends on the data set. The results also show that autoencoders are able to improve the performance of the clustering algorithms, especially for large data sets.

Contents

1 Introduction

Tourism has globally grown above average at approximately 4% per year for eight straight years since 2010 [1]. Also, the world's top 5 tourism spending countries together spent around $581 billion on tourism in 2017. Therefore, tourism can be considered as an important source of income for many people worldwide. However, in many cases tourists are faced by the following problem.

Suppose one is planning to go on vacation, but (s)he does not yet know the destination of the trip. One option is to gather information from social media and the internet on multiple destinations, and make a decision based on this information. However, this approach is very time consuming and therefore not very appealing to customers. This is where recommender systems come into the picture. Based on the historical behaviour of customers on social media and the internet, recommender systems are able to give customers personalized recommendations on where to go on vacation.

These recommender systems are able to process big volumes of data, oftentimes making use of collaborative filtering algorithms [2, 3]. However, social media and the internet have caused an exponential growth in the amount of data to be processed, making collaborative filtering recommender systems computationally expensive, thus leading to scalability problems [4]. The proposed solution by the authors of [5] to these scalability problems is to make use of clustering algorithms. These clustering algorithms are incorporated in recommender systems by only making personalized recommendations within relatively small clusters. To be specific, the authors of [5] use partitioning clustering algorithms for segmenting customers in the tourism domain, based on social media data.

We expand the work of [5] in this thesis by comparing the performance of the proposed partitioning clustering algorithms to hierarchical clustering algorithms as well as density-based clustering algorithms. However, these clustering algorithms may suffer from scalability problems due to the high dimensions of some of the data sets encountered in social media data. Therefore, we employ deep learning and autoencoders as a dimensionality reduction technique to further boost the performance of the clustering algorithms used in this thesis. This will also help to deal with the presence of noise and outliers in large data sets, as it is likely that the encoding of a data set will filter out the noise.

From the previous, it follows that the main research question that we investigate in this thesis is:

"How to devise a clustering algorithm for segmenting customers in the tourism domain, based on social media data?"

In order to answer this main research question, we investigate the following sub-questions:

- *"What type of clustering algorithm should be used for effectively segmenting customers in the tourism domain, based on social media data?"*
- *"How do dimensionality reduction techniques affect the performance of clustering algorithms in the tourism domain?"*

The results in section 5 show that the clustering algorithms which perform the best in general are the k-means algorithm, the k-medoids algorithm, and the hierarchical clustering algorithm with either complete or Ward linkage. Also, it shows that, especially for large data sets, it is possible to improve the performance of the clustering algorithms by using autoencoders to reduce the dimensionality of the data.

This thesis is structured as follows. An overview of existing related literature is given in section 2. In section 3, we give a description of the data used. The methods used in order to answer the research question are discussed in section 4. Section 5 presents the results obtained by the clustering algorithms and autoencoders. We draw some final conclusions in section 6. There, we also discuss limitations of the research being done in this thesis, and we give some suggestions for future research.

2 Related work

A lot of research has been done with respect to (collaborative filtering) recommender systems [6], which are also used in various other applications than tourism, such as e-commerce and movies [7, 8, 9]. There has even been a competition called the "Netflix Prize", with the target of developing the best collaborative filtering algorithm for predicting user ratings of movies [10]. The winner of this competition, a team called "BellKor's Pragmatic Chaos", was awarded a sum of $1,000,000 [11]. Recommender systems have also been developed using other algorithms than collaborative filtering, such as recommender systems based on matrix factorization models [12], content-based recommendation, knowledge-based recommendation, or hybrid models which combine collaborative filtering and knowledge-based recommendation [13].

Clustering algorithms can be divided into two types: partitioning clustering algorithms and hierarchical clustering algorithms [14]. In the field of partitioning clustering algorithms, methods have been developed for effectively clustering documents with high dimensional feature spaces [15]. These clustering techniques are based on generalizations of graph partitioning. Recent developments on hierarchical clustering have shown that a proper objective function, combined with a simple recursive sparsest-cut based approach for similarity-based hierarchical clustering, and a classic average-linkage approach for dissimilarity-based hierarchical clustering, yields relatively efficient results [16].

More novel clustering algorithms, compared to the standard partitioning and hierarchical clustering algorithms, make use of incorporated dimensionality reduction techniques. One of these algorithms is the spectral clustering algorithm [17]. This algorithm derives matrices from the data under consideration, and uses eigenvectors of the matrices in order to perform dimensionality reduction techniques before the actual clustering is done. Another solution to the scalability problems that are oftentimes encountered in clustering, is called canopy clustering [18]. Its central idea is that it efficiently divides the data into overlapping subsets, called canopies, by using a cheap, approximate distance measure.

Next to dimensionality reduction techniques which are incorporated into clustering algorithms, the field of dimensionality reduction techniques is a well-studied research topic on its own as well. One of the most traditional dimensionality reduction techniques is principal component analysis [19]. The main idea of principal component analysis is to transform the data into a

new set of variables (called the principal components). These principal components should then be uncorrelated, and ordered in such a way that most of the variation present in the entire data set is captured by the first few variables. An extension of principal component analysis, called kernel principal component analysis, which is proposed in [20], makes use of an integral operator kernel function to efficiently compute the principal components. When the situation occurs in which the data consists of non-negative signals only, it has been shown that non-negative matrix factorization is a useful dimensionality reduction technique [21]. Non-negative matrix factorization decomposes the non-negative data matrix into the product of two other non-negative data matrices. The autoencoder technique which we use in this thesis for dimensionality reduction, has also been used in other applications like noise reduction and speech enhancement [22]. For an overview of nonlinear dimensionality reduction techniques, one could consider the overview provided by [23].

3 Data description

The data used to evaluate the algorithms described in section 4 consists of three different data sets of varying sizes. These are the same data sets as the ones which are used in [5].

The first data set we are presented with is the so called `BuddyMove`[1] data set, which contains the number of reviews for 6 different destination categories across South India per reviewer. These numbers of reviews per reviewer are collected from more than 1500 reviews originating from HolidayIQ.com, during a period which lasted until October 2014. The `BuddyMove` data set contains 249 observations on 7 attributes (the first attribute is a unique user ID).

The second data set is collected from TripAdvisor, and it contains average user reviews on 10 different destination categories across East Asia. Each of the single review ratings is either "Terrible" (0), "Poor" (1), "Average" (2), "Very Good" (3), or "Excellent" (4). We will refer to this data set as the `TripAdvisor`[2] data set, which consists of 980 observations on 11 attributes (the first attribute represents a unique user ID).

The third and largest data set we consider contains average user ratings from Google reviews on attractions from 24 different categories across Europe. The individual user ratings range from 1 ("Terrible") to 5 ("Excellent"). This third data set will be referred to as the `Google`[3] data set, containing 5456 observations on 25 attributes (again the first attribute is a unique user ID).

A summary of the three data sets and information on the individual attributes are provided in Table A1 (see Appendix A).

4 Methodology

In this section, we describe the methods used in order to answer the main research question and its sub-questions. In section 4.1, we show how we deal with missing values in the data, and why we normalize the data before clustering. The clustering algorithms which serve to segment customers of the data sets, are discussed in section 4.2. In section 4.3, we present the

[1]The `BuddyMove` data set is available at `https://archive.ics.uci.edu/ml/datasets/BuddyMove+Data+Set`.
[2]The `TripAdvisor` data set is available at `https://archive.ics.uci.edu/ml/datasets/Travel+Reviews`.
[3]The `Google` data set is available at `https://archive.ics.uci.edu/ml/datasets/Tarvel+Review+Ratings`.

autoencoders which we use to deal with the high dimensions of some of the data sets encountered in social media data. We also need to evaluate how well our methods perform, therefore in section 4.4 we show which evaluation measures we use for this.

4.1 Data cleaning and normalizing

The data we work with in this thesis may contain missing values. Therefore, we have to come up with a way to deal with those potential missing values. In our case, we do not have any missing values for the `BuddyMove` and `Tripadvisor` data sets, and we only have two observations with a missing value for the `Google` data set (out of the 5456 observations). As we thus have very few observations with missing values, we simply remove the observations that contain missing values. However, one may encounter a data set in which the number of observations with a missing value is substantially higher. In this case, it may be better to replace a missing value by the mean of the attribute for which the missing value occurs.

As we will see in Section 4.2, for the clustering algorithms we have to compare observations from a given data set based on the differences in their attribute values. However, for some attributes the differences between observations may be higher than for other attributes, which would lead to some attributes having a higher influence than others when comparing observations. This is not desirable, as we would like to have that each attribute has the same amount of influence when comparing observations. Therefore, we have to normalize the data before we start clustering. We do this in the following manner. For each of the attribute values, we subtract the mean of that attribute. Then, we divide each of the attribute values by the standard deviation of that attribute. In this way, each attribute approximately gets a mean of zero and a standard deviation of one.

4.2 Clustering algorithms

In general, clustering is a subfield of machine learning which focuses on the task of grouping a set of objects in such a way that objects from the same cluster are more similar than objects from different clusters. Clustering is an unsupervised approach, which means that the data we work with is not categorized. Thus, instead of having to predict the labels of objects, clustering, and in general unsupervised machine learning algorithms, try to find hidden patterns in the data. Through finding these hidden patterns, clustering algorithms tend to expose relevant groups within data sets.

However, in order to be able to form a clustering, we need to define how to measure similarity between objects. We consider the similarity between two objects to be the distance between those objects. This means that opting for a clustering with high intra-cluster similarity and low inter-cluster similarity, requires a clustering in which objects from the same cluster are relatively close to each other, and objects from different clusters are relatively far away from each other. In the literature, several methods have been used to measure distance, including Euclidian distance, Manhattan distance, Cosine similarity, the Jaccard coefficient, Minkowski distance, and Pearson's correlation coefficient [5, 24]. In this thesis, we use Euclidean distance to measure similarity between objects, as this is the most commonly used distance measure.

As mentioned before, we can make a distinction between two types of clustering algorithms: partitioning clustering algorithms and hierarchical clustering algorithms, which are discussed in sections 4.2.1 and 4.2.2, respectively. The difference between partitioning and hierarchical clustering algorithms is that for partitioning clustering algorithms, the objects are clustered in such a way that we end up with a single partition, whereas for hierarchical clustering algorithms, the objects are clustered in such a way that we get a nested sequence of partitions. On the other hand we have density-based clustering algorithms, which are discussed in section 4.2.3. We discuss density-based clustering algorithms separately from partitioning and hierarchical clustering algorithms, since some density-based clustering algorithms create a clustering consisting of a single partition, and some create a clustering consisting of a nested sequence of partitions.

4.2.1 Partitioning clustering

The partitioning clustering algorithms we consider in this thesis are the same algorithms as in [5]. The most traditional of these algorithms are k-means clustering, which was introduced by [25], and k-medoids clustering, which is equivalent to Partitioning Around Mediods (PAM) [26, 27]. Furthermore, we use Clustering for Large Applications (CLARA) [27, 28, 29] and fuzzy c-means clustering [30, 31]. As this thesis is an extension of [5], we will not discuss the partitioning clustering algorithms in detail. Instead, we briefly explain each of the algorithms and advise to consult [5] for thorough explanations of the algorithms.

4.2.1.1 K-means The k-means algorithm partitions a given data set into k clusters. Each of these clusters has a centroid, which is the mean of the objects in that cluster. As such, the centroid of a cluster does not have to be an actual object from the data set. The algorithm starts by selecting k initial centroids, which are obtained by randomly sampling objects from the data set. Then, for each object in the data set, the distance to each of the centroids is calculated, and the object is assigned to the cluster corresponding to the closest centroid. After all of the objects have been assigned to clusters, the centroids are updated by calculating the mean of each cluster. This process of assigning objects to clusters and updating centroids is iteratively repeated until the clustering assignments do not change, or until a maximum number of iterations is reached.

The main advantage of the k-means algorithm is that it is relatively easy to understand and implement. However, disadvantages are the sensitivity to outliers, and the scalability problems that occur for large data sets. We also face the challenges of having to to define the number of clusters beforehand, and having to choose how to select initial centroids. The implementation of the k-means algorithm used in this thesis, is the one by Hartigan and Wong [32]. This implementation is an efficient version of the original k-means algorithm.

4.2.1.2 K-medoids The k-medoids algorithm is very similar to the k-means algorithm. The main difference however is that for the k-medoids algorithm, the centroid of a cluster is called the medoid, which is the object of that cluster for which the average distance to all other objects in that cluster is minimal. As a result, the k-medoids algorithm works as follows. By randomly sampling objects from the data set, k initial medoids are selected. We then calculate the distance to each of the medoids for each object in the data set. Thereafter each object is assigned to the

cluster corresponding to the nearest medoid. When each of the objects has been assigned to a cluster, we update the medoids in the following way. For each of the clusters, we search for an object within that cluster that has a lower average distance to all other objects in that cluster than the current medoid. If we find such an object for a particular cluster, we make that object the medoid of that cluster. Similarly to the k-means algorithm, the process of assigning objects to clusters and updating medoids is iteratively repeated until the clustering assignments stay the same, or until we reach a maximum number of iterations.

The k-medoids algorithm suffers from the same problems as the k-means algorithm, except that the k-medoids algorithm is more robust to outliers. This is because the k-medoids algorithm uses medoids as centroids instead of means, which results in centroids that do not adjust to outliers. We use the Partitioning Around Medoids (PAM) [27] implementation of the k-medoids algorithm in this thesis.

4.2.1.3 Clustering for Large Applications (CLARA) The CLARA algorithm has been developed in order to resolve the scalability problems faced by the PAM algorithm. As such, the CLARA algorithm extends the PAM algorithm in the following manner. The data set is split up into different subsets by means of random sampling, and to each of these subsets, we apply the PAM algorithm. This yields k medoids for each subset, and we obtain a clustering for the entire data set by assigning each object of the data set to the nearest medoid. We can then evaluate the quality of the obtained clustering for each subset by calculating the average dissimilarity of the objects in the data set to their closest medoid. Therefore, for each subset, we end up with a clustering for the entire data set, and we select the clustering with the highest quality (thus the lowest average dissimilarity). In this thesis, we use the CLARA algorithm provided by [27].

4.2.1.4 Fuzzy c-means (FCM) In fuzzy clustering, each object of the data set belongs to each cluster to a certain degree, as opposed to non-fuzzy clustering in which each object of the data set belongs to exactly one cluster. The extent to which object x_i belongs to cluster j, is denoted by the degree of membership $\mu_j(x_i)$, and it must hold that $\mu_j(x_i)\ \epsilon\ [0,1]$ for all objects x_i and for all clusters j. Each object will have a relatively high degree of membership for clusters that are nearby, and a relatively low degree of membership for clusters that are further away (less similar). We therefore also impose the restriction that the degrees of membership of each object must sum to 1, i.e., $\sum_{j=1}^{c} \mu_j(x_i) = 1$ for all objects x_i, where c is the total number of clusters. The FCM algorithm serves to minimize the objective function $\sum_{j=1}^{c} \sum_{i=1}^{n} \mu_j(x_i)^m d^2(x_i, C_j)$, where n is the number of objects, m is the fuzziness coefficient, and $d^2(x_i, C_j)$ is the squared distance between object x_i and cluster j's centroid C_j. The fuzziness coefficient m lies in the range $[1, \infty]$, and as m increases, the fuzziness of the obtained clustering by fuzzy c-means increases as well. It is most common to set m equal to 2 [33]. In order to minimize the objective function, the following procedure is applied.

We first have to determine c, the number of clusters. For each of these clusters, we randomly sample an object from the data set, such that we obtain c initial centroids. Based on these

centroids, we calculate the degree of membership for each object x_i to each cluster j as

$$\mu_j(x_i) = \left(\sum_{l=1}^{c} \left(\frac{d^2(x_i, C_j)}{d^2(x_l, C_j)} \right)^{1/(m-1)} \right)^{-1}. \tag{1}$$

We can then update the centroid of each cluster j by

$$C_j = \frac{\sum_{i=1}^{n} \mu_j(x_i)^m x_i}{\sum_{i=1}^{n} \mu_j(x_i)^m}. \tag{2}$$

The procedure of updating membership degrees by (1) and centroids by (2) is iteratively repeated until the membership degrees stay the same, or until a maximum number of iterations is reached. In this way, we minimize the objective function. Eventually, to obtain a single partition, each object x_i is assigned to the cluster for which the degree of membership is highest. The fuzzy c-means implementation used in this thesis stems from [30].

4.2.2 Hierarchical clustering

Here we explain the hierarchical clustering algorithm considered in this thesis. We use agglomerative hierarchical clustering, which has a bottom up approach: each object starts as its own cluster, and we iteratively select two clusters that are merged to form one larger cluster, until all objects are in one single cluster [14]. As such, we have to decide on each iteration which two clusters are going to be merged. This is done by taking the two clusters which are the most similar (or the closest to each other in terms of distance). There are several methods to determine the distance between two clusters. In this thesis, we first consider the perhaps simplest distance measures, which are single linkage, complete linkage, and average linkage. Single linkage measures the distance between two clusters as the minimum distance between any two objects of the clusters, whereas complete linkage measures the distance between two clusters as the maximum distance between any two objects of the cluster. As one would expect, the distance between two clusters is measured as the average distance between any two objects of the clusters in the case of average linkage. A mathematical representation of the distance measures is given below, with A and B being two clusters containing objects, thus $d(x_i, x_j)$ represents the distance between object x_i from cluster A and object x_j from cluster B. We use $|.|$ to denote the cardinality of a cluster.

$$\text{Single linkage: } d(A, B) = \min_{x_i \epsilon A, x_j \epsilon B} \{d(x_i, x_j)\} \tag{3}$$

$$\text{Complete linkage: } d(A, B) = \max_{x_i \epsilon A, x_j \epsilon B} \{d(x_i, x_j)\} \tag{4}$$

$$\text{Average linkage: } d(A, B) = \frac{1}{|A||B|} \sum_{x_i \epsilon A} \sum_{x_j \epsilon B} d(x_i, x_j) \tag{5}$$

Besides single, complete, and average linkage, we also consider the more advanced distance measure called Ward linkage [34, 35]. This distance measure is given by (6), with $||.||$ the Euclidean norm, and C_A and C_B the centroids of clusters A and B, respectively.

$$\text{Ward linkage: } d(A,B) = \frac{|A||B|}{|A|+|B|}||C_A - C_B||^2 \tag{6}$$

However, we have a problem as it is desired to end up with a single partition of the data set, and applying the agglomerative hierarchical clustering algorithm does not give us such a single partition explicitly. Instead, the algorithm gives us a hierarchical clustering from which we can derive a single partition. In order to be able to obtain such a single partition, we have to specify the number of clusters, just as for the partitioning clustering algorithms. Having specified this number of clusters, we can obtain a single partition by moving down the hierarchy of the hierarchical clustering. Namely, we start with one cluster that contains all objects, and each time we take a step down in the hierarchy, one of the clusters is split into two clusters. Therefore, we can just take steps down in the hierarchy until we end up with the desired number of clusters. Note that this process of taking steps down in the hierarchy is exactly the opposite of the agglomeration which is done in the agglomerative hierarchical clustering algorithm.

4.2.3 Density-based clustering

Next to the partitioning and hierarchical clustering algorithms described above, we consider the density-based clustering algorithms DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [36, 37], and OPTICS (Ordering Points To Identify the Clustering Structure) [38], which is an extension to the DBSCAN algorithm.

4.2.3.1 DBSCAN The DBSCAN algorithm is based on the notion that clusters may have any arbitrary shape. Therefore we identify clusters by only considering the density of a certain region in the space of the input data. More precisely, we classify each object of the data set as one of the following: a core point, a border point, or an outlier. The main idea is that a core point lies in the core of a cluster, a border point belongs to a cluster but not to its core, and an outlier does not belong to any cluster. The advantage of this approach is that we do not have to specify the number of clusters beforehand, as opposed to partitioning and hierarchical clustering algorithms. Instead, we have to specify two other parameters: $minPts$ and eps. After specifying these parameters, we can identify an object P as:

- a core point, if the number of objects for which the distance to P is less than or equal to eps, is at least $minPts$;
- a border point, if the number of objects for which the distance to P is less than or equal to eps, is less than $minPts$, but there is a core point for which the distance to P is less than or equal to eps;
- an outlier, if the number of objects for which the distance to P is less than or equal to eps, is less than $minPts$, and there is no core point for which the distance to P is less than or equal to eps.

When identifying an object as a core point or as a border point, we also make sure to store for which cluster the object is a core point or border point, such that we end up with a single

partition.

In Algorithm B1, the pseudocode of the DBSCAN algorithm is given (see Appendix B). The algorithm makes use of two functions, which are called `dist` and `RangeQuery`, respectively. For the former function, we simply apply the Euclidean distance in this thesis, whereas the latter function returns a list containing all neighbors of a given object P. An object is considered as a neighbor of P if its distance to P is less than or equal to *eps*. The pseudocode of the `RangeQuery` function can be found in Algorithm B2 (see Appendix B).

Having specified the functions needed, we can discuss the DBSCAN algorithm in detail. What the algorithm does, is that it iterates over all objects in the database. When an object has already been assigned to a cluster, we move onto the next object. If not, we check whether the object is an outlier. If the object is identified as an outlier, we label it as "noise", and proceed to the next object. However, if the object is not identified as an outlier, we identify it as a core point and create a new cluster for that object. We then iterate over all neighbors of the core point, and if a neighbor was identified as an outlier earlier on, we now identify that neighbor as a border point of the current cluster. If a neighbor has already been assigned to another cluster, we skip that neighbor and go to the next one. Finally, if a neighbor has not been labeled yet, we assign that neighbor to the current cluster, and we check whether that neighbor is a core point of the cluster as well. If this is the case, we extend the list of neighbors with all neighbors of the newly found core point.

4.2.3.2 OPTICS As earlier mentioned, the OPTICS algorithm is an extension to the DBSCAN algorithm, and it does so by addressing one of the disadvantages of the DBSCAN algorithm. This disadvantage is that the DBSCAN algorithm does not work well for data consisting of clusters with varying density, since we can not adjust the values of the parameters $minPts$ and *eps* for each cluster separately. The way the OPTICS algorithm tackles this disadvantage, is by linearly ordering the objects of the data set according to the density structure of the objects. This means that objects in a dense cluster are listed closer together than objects in a less dense cluster. Since the OPTICS algorithm does not provide an explicit clustering, we have to extract a clustering from the outputted list of ordered objects. Now, in order to provide an ordering for the objects, we need to know two things for each object P:

- The core-distance of P. If P is a core point, then the core-distance of P is defined as the smallest value of *eps* for which the number of objects within a distance of *eps* from P is greater than or equal to $minPts$. However, if P is not a core point, then the core-distance of P is undefined.

- The reachability-distance to P from a core point O. If there is a core point O for which the distance to P is smaller than or equal to *eps*, the reachability-distance from O to P is defined as the maximum of the core-distance of O and the distance from P to O. In the case that such a core point O does not exist, the reachability-distance to P is undefined.

This core-distance and reachability-distance of each object are then used by the OPTICS algorithm, which is displayed in Algorithm B3 (see Appendix B). There, one can also find how the core-distance is implemented as a function. Next to this function, the OPTICS algorithm

makes use of the functions `dist`, `RangeQuery`, and `updateSeeds`. The `dist` and `RangeQuery` functions are exactly the same as for the DBSCAN algorithm. The `updateSeeds` function is described in Algorithm B4 (see Appendix B).

Having explained the core concepts of the OPTICS algorithm, we are able to discuss it in detail. First, an empty list *OrderSeeds* is created to store the ordered objects. The objects will eventually be in ascending order, according to their reachability-distance from the closest core point. Also, *OrderSeeds* will eventually contain multiple groups of ascending objects (in terms of the reachability-distance), since the objects will be grouped within the list such that their reachability-distances correspond to core points from the same cluster. We initialize the reachability-distance of each of the objects as "undefined". Then, we iterate over all unprocessed objects from the database, and for each object we mark it as processed and add it to the *OrderSeeds* list. If the unprocessed object is not a core point, we simply move on to the next unprocessed object. However, if we encounter an unprocessed object P which does happen to be a core point, we update the reachability-distance of all objects which are neighbors of P. Each of these neighbors is temporarily stored in ascending order (based on the reachability-distance) in the priority queue *Seeds*. In this way, we can find out whether some of these neighbors are core points as well, and if this is the case, we add the neighbors of those core points to *Seeds* as well. The objects in *Seeds* will eventually be ordered like we need it for the *OrderSeeds* list. Therefore, all objects which are present in *Seeds* are then dequeued one by one, and added to the *OrderSeeds* list. In the end, all objects that come from the same *Seeds* queue, will be assigned to the same cluster.

4.3 Autoencoders

In this section, we explain the autoencoders which we use as a dimensionality reduction technique in order to deal with the high dimensions of some of the data sets encountered in social media data. Autoencoders are specific types of neural networks [39, 40]. Therefore we will first explain neural networks, and then discuss how autoencoders work.

Neural networks stem from the field of machine learning, and are among the most popular techniques in this field nowadays. A neural network is a model which tries to learn the features of a data set, usually with the purpose of classification. A visual representation of the architecture of a neural network is shown in Figure 1, which we will use to explain how a neural network works.

To give an example, neural networks are widely used to classify handwritten digits, with the goal of trying to predict which digit has been written in a certain image. This is done in the following manner. A neural network consists of an input layer, one or more hidden layers, and an output layer. Each of these layers consist of neurons, which all hold a single value. These neurons are represented by the grey circles in Figure 1. The number of neurons for the input layer is equal to k, the dimension of the input data, while the number of neurons for the output layer is equal to m, the number of classes which can be predicted. For example, if we try to classify handwritten digits which are represented by images of 28×28 pixels, then the input layer consists of 784 neurons, and the output layer consists of 10 neurons (since we can predict the numbers 0 to 9). In between the input layer and the output layer, we have h hidden layers,

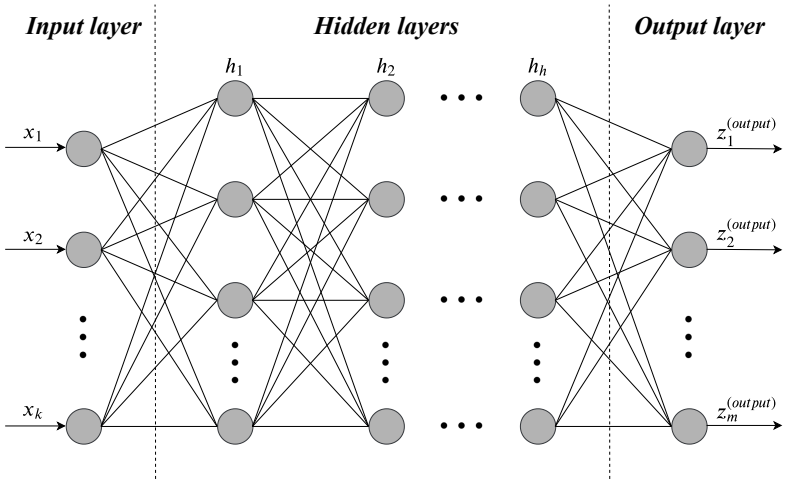

Figure 1: Architecture of a neural network.

with $h \geq 0$. The amount of hidden layers, and the number of neurons per hidden layer are not predefined. All the neurons are then, layer by layer, connected to each other through weights. These weights are visualized in Figure 1 by the lines which connect the neurons. The value which is held by a single neuron is then the weighted sum of all neurons in the previous layer, plus a bias term. Additionally, this value is transformed by applying an activation function. Mathematically, the value held by the j-th neuron in the l-th layer can thus be represented as

$$z_j^{(l)} = \sigma\Big(w_j^{(l)} z^{(l-1)} + b_j^{(l)}\Big). \tag{7}$$

Here, $\sigma(\cdot)$ is the activation function, $w_j^{(l)}$ is a row vector containing the weights from layer $l-1$ to the j-th neuron of layer l, $z^{(l-1)}$ is a column vector containing the values held by the neurons in layer $l-1$, and $b_j^{(l)}$ is the bias term for the j-th neuron of layer l. The neural network we have described thus far is known as a feedforward neural network, since this neural network takes the input data and feeds it forward layer by layer, until we reach the output layer. If we go back to the example of classifying handwritten digits, then the values which are held by the neurons in the output layer represent the probabilities that the input image corresponds to those digits (since each neuron in the output layer represents one of the digits). The digit which is then predicted for that input image, is the one which corresponds to the neuron in the output layer with the highest probability.

The knowledge of a neural network is thus fully stored in the weights and biases, and training the network comes down to adjusting these weights and biases. More precisely, training a neural network consists of three parts: a feedforward part, a backpropagation part, and a part in which the weights and biases are updated.

First, we apply the feedforward part which works as we described above: it takes the input data and feeds it forward layer by layer, until we reach the output layer. Then we can evaluate how well the neural network is performing by comparing the output to the desired output. In the case of classifying handwritten digits, the desired output would be a probability of 1 for the digit that is written in the input image, and a probability of 0 for all other digits. In this way, we can use a loss function which uses the output and the desired output to compute a loss value

for the neural network. The lower this loss value is, the better the neural network is performing.

Second, using the loss function, we can apply the backpropagation part of training the neural network. This part applies gradient descent in order to minimize the loss function. It does so by taking the output of the loss function, and based on this output, the gradient descent algorithm computes partial derivatives for all weights and biases. The partial derivative of the weight that connects the i-th neuron of layer $l-1$ to the j-th neuron of layer l is given by (8), and the partial derivative of the bias term for the j-th neuron of layer l is given by (9). In both equations, L is the loss function.

$$\Delta w_{ij}^{(l)} = \frac{\partial L}{\partial w_{ij}^{(l)}} \tag{8}$$

$$\Delta b_{j}^{(l)} = \frac{\partial L}{\partial b_{j}^{(l)}} \tag{9}$$

Third and last, we update the weights and biases of the neural network. For this part, we use the partial derivatives of the weights and biases that are computed during the backpropagation. The weights and biases are then updated by equations (10) and (11), respectively. Here, α is the learning rate, which determines how much the weights and biases change. The value of α lies in the range $(0, 1]$.

$$w_{ij}^{(l)} := w_{ij}^{(l)} - \alpha * \Delta w_{ij}^{(l)} \tag{10}$$

$$b_{j}^{(l)} := b_{j}^{(l)} - \alpha * \Delta b_{j}^{(l)} \tag{11}$$

Above, we have described the three parts of training a neural network. Executing these three parts once is called an epoch, and during each single epoch, the weights and biases get updated once in order to minimize the loss function. Then, to fully train a neural network, we have to iteratively execute multiple epochs.

As earlier mentioned, the amount of hidden layers and the number of neurons per hidden layer are not predefined. These numbers are thus free parameter choices which one can adjust. The disadvantage of this is that there is no way to determine the optimal amount of hidden layers and the number of neurons per layer, since there is no real intuition behind it. The only intuition we have is that each hidden layer tries to capture some features of the data set, and that multiple hidden layers can thus improve our model. However, we must not create too many hidden layers and neurons, since we do not want our model to pick up noise. Deciding on the amount of hidden layers and the number of neurons per hidden layer thus comes down to a process of trial and error with parameter settings that seem reasonable.

Having explained neural networks, we can now discuss autoencoders and how they relate to neural networks. An autoencoder is a neural network which consists of two parts: an encoder and a decoder. The encoder serves to encode the most important features of the data to a lower dimension. Then, the decoder tries to decode the encoded data back to the original data. Therefore, the desired output of an autoencoder is exactly the data that was given as input, and the input layer and output layer thus have exactly the same number of neurons (if we go back to

the neural network of Figure 1, we have that $k = m$). A visual representation of an autoencoder is given in Figure 2.

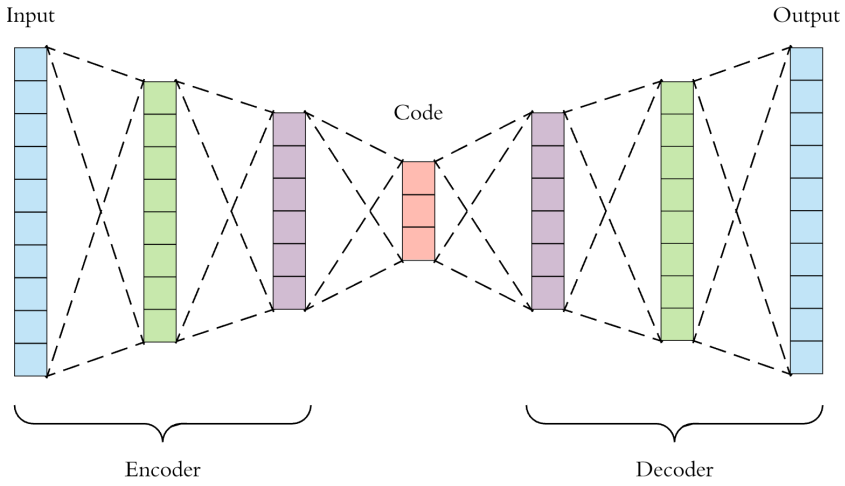

Figure 2: Visual representation of an autoencoder.

For the hidden layers, there is one middle layer which represents the encoded data. This is the layer which is presented in red in Figure 2. The number of neurons in this middle layer is thus equal to the lower dimension to which the data is encoded (we call this the encoding dimension). The input layer, the hidden layers in between the input layer and the middle layer, and the middle layer together form the encoder. The decoder consists of the middle layer, the hidden layers in between the middle layer and the output layer, and the output layer.

When creating the autoencoders for the data sets in this thesis, there are quite some parameter choices that need to made. For most of these parameter choices, it is not the case that there is one choice that is particularly the best, as was also explained for the number of hidden layers and the number of neurons per hidden layer. Therefore, the parameter choices that are made in this thesis for the autoencoders are the result of what seems logical, and finding out what works best by trial and error, unless otherwise stated.

The first decision to make is which activation function $\sigma(\cdot)$ to use. One possible activation function is the rectified linear unit (ReLu) function [41], which is given by $\sigma(z) = max\{0, z\}$. However, this activation function suffers from the so called "dying ReLu" problem. This is the problem that whenever the input of a neuron gets non-positive, we get zero as the value being held by that neuron. This neuron is then said to be "dead", and this may cause other neurons in further layers to become "dead" as well. A proposed solution to this problem by the authors of [42], is the leaky ReLu function. This function is given by $\sigma(z) = max\{\gamma z, z\}$, with $\gamma \epsilon$ (0,1). As one is able to observe, replacing 0 by γz prevents the neurons from becoming "dead". Now, for the encoder part of the autoencoder, we apply the leaky ReLu function with $\gamma = 0.3$. However, for the decoder part of the autoencoder, we simply apply the linear activation function, which is given by $\sigma(z) = z$. Note that this means that for the neurons of the middle layer, the leaky ReLu function is applied, as computing the values which are held by the neurons in the middle layer is part of encoding the data.

We then have to decide which loss function to use for the autoencoders in this thesis, and for this decision, we choose to use the Mean Squared Error (MSE) loss function. This function

is given by

$$L = \frac{1}{n} \sum_{i=1}^{n} \left(Z_i^{(output)} - X_i \right)^2, \tag{12}$$

with n the number of observations in the data set, $Z_i^{(output)}$ a column vector containing the values which are held by the neurons of the output layer for the i-th observation, and X_i a column vector containing the attribute values of the i-th observation.

When applying backpropagation, we have to specify which optimizer to use. Here, we use the ADADELTA optimizer [43], which is a method that dynamically adapts the learning rate, based on first order information only (see equations (10) and (11) for the learning rate). In this way, we do not have to set the learning rate manually.

What is left are parameter choices that differ per data set. Therefore, we discuss these parameter choices separately for each data set. Note that for each of the data sets, the dimension of the data is equal to the number of attributes minus one, since the first attribute is a unique user ID for each of the data sets.

First, we have the `BuddyMove` data set. This data set only has a dimension of 6, however we still apply an autoencoder to this data set to see if it can improve the results. The encoder part of this autoencoder contains one hidden layer with 5 neurons, the encoding dimension is equal to 3, and the decoder part also contains one hidden layer with 5 neurons. Therefore, we end up with an autoencoder which consists of 5 layers in total, which contain 6, 5, 3, 5, and 6 neurons, respectively. We use 300 epochs to train this autoencoder.

Secondly, we create an autoencoder for the `Tripadvisor` data set, which has a dimension of 10. For this autoencoder, the encoder part contains one hidden layer of 8 neurons, the encoding dimension is equal to 5, and the decoder part also contains one hidden layer of 8 neurons. This autoencoder thus consists of 5 layers in total, which contain 10, 8, 5, 8, and 10 neurons, respectively. Again, we use 300 epochs to train this autoencoder.

Third and last, an autoencoder is created for the `Google` data set. This data set has the highest dimension of the three data sets, which is 24. We therefore have an encoder part which contains two hidden layers of 20 and 15 neurons, respectively. The encoding dimension of this autoencoder is equal to 10, and the decoder part contains two hidden layers of 15 and 20 neurons, respectively. We therefore get an autoencoder which consists of 7 layers in total, which contain 24, 20, 15, 10, 15, 20, and 24 neurons, respectively. In order to train this autoencoder, we use 200 epochs.

4.4 Evaluation measures

After cleaning the data, normalizing the data, possibly encoding the data using an autoencoder, and then applying one of the clustering algorithms of Section 4.2, we end up with a single partition of the data set. In order to evaluate how well our methods perform, we need a way to evaluate the goodness of such a single partition. We do this by considering the same evaluation measures as in [5], except for the entropy measure. The reason for this is that the entropy measure is an external evaluation measure [44], which means that we need labeled data to compute the entropy measure. However, the data we are presented with is unlabeled, which makes it impossible to

compute the entropy measure. This means that the evaluation measures that we do consider are internal evaluation measures. These are the average Silhouette width (`avg.silwidth`) [45], the Dunn index (`dunn`) [46], an index which belongs to the family of Dunn indexes (`dunn2`), and a ratio of average distances (`wb.ratio`). These evaluation measures have the following definitions. The Silhouette width of an object i is defined as

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}, \tag{13}$$

with $a(i)$ the average dissimilarity between i and all other objects of A, which is the cluster to which i belongs. Note that $a(i) = 0$ if i is the only object in A. If we define $d(i, C)$ as the average dissimilarity between i and the objects of cluster C, then $b(i)$ can be defined as $\min_{C \neq A}\{d(i, C)\}$. The average Silhouette width is then calculated as $\frac{1}{n}\sum_{i=1}^{n} s(i)$, with n the number of observations in the data set.

For the Dunn index, we divide the minimum separation between any two clusters by the maximum of the diameters of all clusters. We can write this mathematically as

$$D = \frac{\min_{i=1,\ldots,k;j=i+1,\ldots,k}\{d(c_i, c_j)\}}{\max_{l=1,\ldots,k}\{diam(c_l)\}}. \tag{14}$$

Here, k is the number of clusters, $d(c_i, c_j)$ is the minimum distance between any two objects of clusters i and j, and $diam(c_l)$ is the maximum distance between any two objects of cluster l.

The `dunn2` measure is defined as the minimum average dissimilarity between any two clusters divided by the maximum average within cluster dissimilarity. Mathematically, this translates to

$$D2 = \frac{\min_{i=1,\ldots,k;j=i+1,\ldots,k}\{d_{avg}(c_i, c_j)\}}{\max_{l=1,\ldots,k}\{d_{avg}(c_l)\}}, \tag{15}$$

with k again the number of clusters, $d_{avg}(c_i, c_j)$ the average dissimilarity between the objects of clusters i and j, and $d_{avg}(c_l)$ the average dissimilarity between the objects of cluster l.

The last evaluation measure we consider, `wb.ratio`, can be defined as

$$wb = \frac{avg.within}{avg.between}. \tag{16}$$

In this equation, *avg.within* is the average distance between any two objects from the same cluster, and *avg.between* is the average distance between any two objects from different clusters.

Note that for the interpretation of these evaluation measures in section 5, the average Silhouette width, the Dunn index, and the `dunn2` measure should be as high as possible, while the `wb.ratio` measure should be as low as possible.

5 Results

The results obtained by implementing the methods described in section 4 are presented in this section. In Tables A2 and A3, we give an overview of the programs that are written in the programming languages R and Python, respectively, to obtain the results, including a short explanation for each of the programs (see Appendix A). Also, in Table A4 we provide an overview of the tools and packages that we use in R and Python (see Appendix A).

Before we are able to obtain results for the clustering algorithms, we need to specify the number of clusters for the partitioning clustering algorithms as well as the hierarchical clustering algorithms. We do this by estimating the optimal number of clusters for each of the data sets, using the package `NbClust` in R. This package provides a function which uses 26 indices, of which each proposes an optimal number of clusters. The number of clusters which is then actually used, is the number of clusters which is proposed the most by the 26 indices. We apply this function provided by `NbClust` to the non-encoded as well as the encoded data, and compare these results to the results of [5] in Tables 1, 2, and 3 for the `BuddyMove`, `Tripadvisor`, and `Google` data sets, respectively. In these tables, "Hierarchical single", "Hierarchical complete", "Hierarchical average", and "Hierarchical Ward" refer to hierarchical clustering with single, complete, average, and Ward linkage, respectively.

		Optimal number of clusters based on...		
		[5]	non-encoded data	encoded data
Algorithm(s)	Partitioning clustering	4	2	3
	Hierarchical single	N/A	4	3
	Hierarchical complete	N/A	2	3
	Hierarchical average	N/A	2	2
	Hierarchical Ward	N/A	3	3

Table 1: Optimal number of clusters for the `BuddyMove` data set.

		Optimal number of clusters based on...		
		[5]	non-encoded data	encoded data
Algorithm(s)	Partitioning clustering	8	6	2
	Hierarchical single	N/A	3	2
	Hierarchical complete	N/A	3	2
	Hierarchical average	N/A	2	2
	Hierarchical Ward	N/A	3	3

Table 2: Optimal number of clusters for the `Tripadvisor` data set.

		Optimal number of clusters based on...		
		[5]	non-encoded data	encoded data
Algorithm(s)	Partitioning clustering	3	3	4
	Hierarchical single	N/A	13	2
	Hierarchical complete	N/A	4	5
	Hierarchical average	N/A	2	2
	Hierarchical Ward	N/A	3	3

Table 3: Optimal number of clusters for the `Google` data set.

We observe that the optimal number of clusters for a particular data set and algorithm is dependent on whether we base it on [5], the non-encoded data, or the encoded data. Therefore, for each combination of a data set and algorithm, we apply the algorithm to that data set with each of the proposed number of clusters.

Next to the number of clusters, for the fuzzy c-means algorithm we also need to specify the fuzziness coefficient m. As earlier mentioned, it is most common to set m equal to 2. However,

for the `Tripadvisor` and `Google` data sets, this value for m yields clusterings which are too fuzzy, meaning that $\mu_j(x_i) \approx \frac{1}{c}$ for each object x_i, and each cluster j. Thus, in order to reduce the fuzziness, we decrease m for the `Tripadvisor` and `Google` data sets until the resulting clusterings are not too fuzzy anymore. This results in m being equal to 2, 1.4, and 1.2 for the `BuddyMove`, `Tripadvisor`, and `Google` data sets, respectively.

The only parameters left to specify, are the parameters for the DBSCAN and OPTICS algorithms. For the DBSCAN algorithm, it is common to set $minPts$ equal to the dimensionality of the data plus one, or higher [47]. Therefore, in order to obtain the best results possible, we consider the values equal to the dimensionality of the data plus one and higher for the $minPts$ parameter. For the *eps* parameter, we create a plot of k-nearest neighbor distances. We can then use this plot by setting *eps* equal to the value for which the knee occurs in the plot. The parameters for the OPTICS algorithm are specified in the same way as the parameters for the DBSCAN algorithm.

Having specified all parameters for the clustering algorithms, we are able to show the results of each of the clustering algorithms which are described in section 4.2. In [5], each of the obtained clusterings is visualized by a plot for the clustering itself, and by a Silhouette plot. In this thesis, we do not show these plots as the number of clusterings that we obtain is rather large. However, the plots are available by running the programs which are listed in Table A2.

In Tables 4, 5, and 6, the values of the considered evaluation measures are shown for the `BuddyMove`, `Tripadvisor`, and `Google` data sets, respectively. In these tables, "H. complete" and "H. Ward" refer to hierarchical clustering with complete and Ward linkage, respectively. Also, each cell of these tables contains two values for that evaluation measure: one in which we apply the clustering algorithm to the non-encoded data to obtain a clustering (which is shown first), and one in which we apply the clustering algorithm to the encoded data to obtain a clustering (which is shown in brackets). The value which corresponds to the case in which the encoded data is used, is marked with an asterisk if that value is better than the value which corresponds to the case in which the non-encoded data is used. Also, for each evaluation measure, we have highlighted the best result. We thus use the non-encoded as well as the encoded data to obtain clusterings. However, an important thing to mention is that for both cases, we use the non-encoded data to obtain the evaluation measures for a clustering. This is due to the fact that the evaluation measures we consider are internal evaluation measures, which means that we would not be able to compare these evaluation measures if they were based on different data sets.

The first thing to notice in Tables 4, 5, and 6, is that some of the discussed clustering algorithms are missing. These algorithms are the hierarchical clustering algorithm with single as well as average linkage, the DBSCAN algorithm, and the OPTICS algorithm. Also, for hierarchical clustering with complete linkage and the number of clusters set to 2, the results are missing for the `Tripadvisor` data set, which are replaced by "N/A". The reason that these algorithms are missing, is that they are not able to provide a proper clustering. This is because for each of the obtained clusterings by these algorithms, the largest cluster contains at least 86.7% of the objects in the data set, and for 27 of these 31 clusterings, the largest cluster contains at least 97.8% of the objects in the data set.

Second, we notice that our results for the partitioning clustering algorithms in Tables 4, 5,

		Evaluation measure							
		avg.silwidth		dunn		dunn2		wb.ratio	
Algorithm	k-means (k=4)	0.343	(0.272)	0.074	(0.032)	1.320	(1.269)	**0.517**	(0.561)
	k-means (k=2)	0.361	(0.287)	0.073	(0.041)	1.398	*(1.421)	0.609	(0.691)
	k-means (k=3)	0.358	(0.294)	0.064	(0.052)	1.494	(1.382)	0.553	(0.594)
	k-medoids (k=4)	0.318	(0.201)	0.043	(0.032)	1.317	(0.971)	0.530	(0.629)
	k-medoids (k=2)	**0.365**	(0.289)	0.065	(0.051)	1.421	(1.395)	0.602	(0.686)
	k-medoids (k=3)	0.334	(0.301)	0.043	(0.038)	1.389	*(1.393)	0.556	(0.605)
	CLARA (k=4)	0.277	(0.258)	0.075	(0.052)	1.226	*(1.227)	0.536	(0.583)
	CLARA (k=2)	0.343	(0.289)	0.083	(0.041)	1.338	*(1.428)	0.647	(0.689)
	CLARA (k=3)	0.342	(0.277)	0.038	(0.032)	1.461	(1.365)	0.552	(0.612)
	FCM (c=4, m=2)	0.253	(0.244)	0.034	(0.032)	1.108	*(1.240)	0.545	(0.576)
	FCM (c=2, m=2)	0.357	(0.275)	0.036	*(0.062)	1.377	(1.371)	0.614	(0.702)
	FCM (c=3, m=2)	0.342	(0.292)	0.048	(0.038)	1.473	(1.397)	0.551	(0.597)
	H. complete (k=2)	0.328	(0.259)	0.087	*(0.088)	1.350	(1.330)	0.635	(0.745)
	H. complete (k=3)	0.264	*(0.332)	0.091	(0.085)	1.146	*(1.473)	0.627	*(0.575)
	H. Ward (k=3)	0.355	(0.311)	**0.145**	(0.070)	**1.610**	(1.461)	0.565	(0.603)

Table 4: Clustering results for the BuddyMove data set.

		Evaluation measure							
		avg.silwidth		dunn		dunn2		wb.ratio	
Algorithm	k-means (k=8)	0.137	(0.065)	0.036	(0.024)	0.760	*(0.805)	**0.675**	(0.751)
	k-means (k=6)	0.141	(0.072)	0.064	(0.041)	0.807	*(0.823)	0.704	(0.771)
	k-means (k=2)	**0.213**	(0.173)	0.062	(0.045)	1.199	*(1.215)	0.772	(0.821)
	k-medoids (k=8)	0.119	(0.057)	0.024	*(0.025)	0.773	*(0.854)	0.691	(0.763)
	k-medoids (k=6)	0.125	(0.051)	0.045	(0.033)	0.834	*(0.980)	0.720	(0.804)
	k-medoids (k=2)	0.210	(0.169)	0.043	*(0.051)	**1.252**	(1.203)	0.780	(0.824)
	CLARA (k=8)	0.106	(0.062)	0.024	*(0.024)	0.843	*(0.888)	0.715	(0.761)
	CLARA (k=6)	0.099	(0.080)	0.038	(0.021)	0.855	(0.664)	0.770	*(0.759)
	CLARA (k=2)	0.148	*(0.174)	0.035	*(0.042)	1.126	*(1.213)	0.854	*(0.821)
	FCM (c=8, m=1.4)	0.109	(0.046)	0.024	*(0.033)	0.780	*(0.913)	0.712	(0.791)
	FCM (c=6, m=1.4)	0.094	(0.050)	0.028	(0.020)	0.999	*(1.001)	0.754	(0.815)
	FCM (c=2, m=1.4)	0.206	(0.175)	0.052	(0.042)	1.192	*(1.216)	0.780	(0.820)
	H. complete (k=3)	0.190	(0.139)	0.071	(0.046)	1.057	(1.021)	0.743	(0.799)
	H. complete (k=2)	N/A	*(0.148)	N/A	*(0.045)	N/A	*(1.144)	N/A	*(0.866)
	H. Ward (k=3)	0.185	(0.118)	**0.089**	(0.044)	1.009	*(1.088)	0.772	(0.824)

Table 5: Clustering results for the Tripadvisor data set.

and 6 are different than the results in [5]. As we do not have access to the R code that the authors of [5] used in order to obtain their results, it is hard to determine what causes theses differences exactly. However, it is worth to mention that for the BuddyMove data set, if we do not normalize the data, we get exactly the same results for each of the partitioning clustering algorithms with the number of clusters set to 4.

We observe from the results that applying the clustering algorithms to the encoded data does not necessarily yield better clusterings than applying these algorithms to the non-encoded data. However, it does seem to be the case that as the size of the data set increases, we get relatively more instances in which applying the clustering algorithms to the encoded data works better than applying these algorithms to the non-encoded data. This would make sense as dimensionality reduction techniques are needed most for larger data sets, since in general they contain more noise.

		Evaluation measure							
		avg.silwidth		dunn		dunn2		wb.ratio	
Algorithm	k-means (k=3)	**0.145**	(0.139)	0.003	*(0.005)	1.051	(1.032)	0.794	(0.798)
	k-means (k=4)	0.144	(0.129)	0.013	*(0.015)	1.012	*(1.028)	**0.778**	(0.791)
	k-medoids (k=3)	0.122	(0.107)	0.021	(0.003)	0.996	*(1.041)	0.809	(0.830)
	k-medoids (k=4)	0.121	(0.116)	0.006	(0.002)	0.915	*(0.932)	0.807	(0.815)
	CLARA (k=3)	0.067	*(0.099)	0.008	*(0.011)	0.883	*(0.999)	0.881	*(0.870)
	CLARA (k=4)	0.081	*(0.123)	0.010	(0.007)	0.904	*(0.963)	0.847	*(0.799)
	FCM (c=3, m=1.2)	0.135	(0.104)	0.004	*(0.005)	1.003	*(1.043)	0.809	(0.857)
	FCM (c=4, m=1.2)	0.105	*(0.108)	0.005	*(0.006)	0.937	*(1.008)	0.819	*(0.813)
	H. complete (k=4)	0.134	(0.079)	0.109	(0.052)	1.055	(0.952)	0.781	(0.817)
	H. complete (k=5)	0.126	(0.077)	**0.114**	(0.057)	1.055	(0.914)	0.781	(0.802)
	H. Ward (k=3)	0.122	*(0.124)	0.100	(0.086)	1.037	*(**1.146**)	0.816	(0.831)

Table 6: Clustering results for the `Google` data set.

Furthermore, there is not one algorithm in particular that consistently outperforms the other algorithms for all data sets. We can however conclude that the k-means algorithm, the k-medoids algorithm, and the hierarchical clustering algorithm with Ward linkage perform the best in general for the `BuddyMove` and `Tripadvisor` data sets. For the `Google` data set, we can conclude that the k-means algorithm and the hierarchical clustering algorithm with either complete or Ward linkage perform the best in general. Which of these algorithms to choose and whether to apply them to the non-encoded or encoded data, thus depends on which data set is used and which evaluation measure is considered as the most important.

6 Conclusion and future work

In this thesis, we have extended the work of [5] by comparing the performance of partitioning clustering algorithms to hierarchical clustering algorithms and density-based clustering algorithms. These clustering algorithms are used for segmenting customers in the tourism domain, based on social media data. Furthermore, we have used autoencoders to obtain encoded representations of the data sets, in order to reduce its dimensions. We have applied the clustering algorithms to the non-encoded data as well as the encoded data to obtain clusterings. These clusterings are evaluated using four internal evaluation measures.

The results have shown that the hierarchical clustering algorithm with single as well as average linkage, and density-based clustering algorithms do not provide a proper clustering for the social media data. To answer the research question, we conclude that the k-means algorithm, the k-medoids algorithm, and the hierarchical clustering algorithm with either complete or Ward linkage are the clustering algorithms which in general are the most effective. Whether to apply the clustering algorithms to the non-encoded data or the encoded data, seems to depend on the size of the data set, since applying the clustering algorithms to the encoded data seems to work better as the size of the data set increases. Concluding, the choice of which clustering algorithm to use and whether to apply this clustering algorithm to the non-encoded data or encoded data for segmenting customers in the tourism domain, depends on the data set under consideration, and which evaluation measure one considers to be the most important.

For future work, we suggest to further explore the use of hierarchical clustering algorithms,

as they have shown to perform relatively well under certain circumstances. Since we have only considered agglomerative hierarchical clustering, it might be interesting to consider divisive hierarchical clustering as well. The use of autoencoders has shown to be promising for obtaining better results, especially for larger data sets. Therefore, we think it would be interesting to apply autoencoders in combination with clustering algorithms to larger data sets than the ones considered in this thesis. Also, as there are many parameter choices that need to be made for autoencoders, and the parameter choices that are considered in this thesis are quite limited, one can further explore which parameter choices yield the best results.

References

[1] World Tourism Organization (UNTWO). *UNWTO Annual Report 2017*. World Tourism Organization (UNWTO), Madrid, 2018.

[2] S. Renjith and C. Anjali. A personalized mobile travel recommender system using hybrid algorithm. In *2014 First International Conference on Computational Systems and Communications (ICCSC)*, pages 12–17, Dec 2014.

[3] S. Renjith and C. Anjali. A Personalized Travel Recommender Model Based on Content-based Prediction and Collaborative Recommendation. *International Journal of Computer Science and Mobile Computing*, ICMIC13:66–73, 12 2013.

[4] S. Jiang, X. Qian, T. Mei, and Y. Fu. Personalized travel sequence recommendation on multi-source big social media. *IEEE Transactions on Big Data*, 2(1):43–56, March 2016.

[5] S. Renjith, A. Sreekumar, and M. Jathavedan. Evaluation of Partitioning Clustering Algorithms for Processing Social Media Data in Tourism Domain. In *2018 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, pages 127–131, Dec 2018.

[6] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating Collaborative Filtering Recommender Systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, Jan 2004.

[7] J. B. Schafer, J. Konstan, and J. Riedl. Recommender Systems in e-commerce. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 158–166, New York, NY, USA, 1999. ACM.

[8] B. N. Miller, I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl. MovieLens Unplugged: Experiences with an Occasionally Connected Recommender System. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, IUI '03, pages 263–266, New York, NY, USA, 2003. ACM.

[9] M. S. Pera and Y. Ng. A Group Recommender for Movies Based on Content Similarity and Popularity. *Inf. Process. Manage.*, 49(3):673–687, 2013.

[10] J. Bennett, S. Lanning, and N. Netflix. The Netflix Prize. 01 2009.

[11] A. Töscher and M. Jahrer. The BigChaos Solution to the Netflix Grand Prize. 01 2009.

[12] R. Bell, Y. Koren, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(08):30–37, Aug 2009.

[13] R. Burke. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, Nov 2002.

[14] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[15] D. Boley, M. Gini, R. Gross, E.H. Han, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. Partitioning-based clustering for Web document categorization. *Decision Support Systems*, 27:329–341, 12 1999.

[16] V. Cohen-Addad, V. Kanade, F. Mallmann-Trenn, and C. Mathieu. *Hierarchical Clustering: Objective Functions and Algorithms*, pages 378–397. 2018.

[17] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, Dec 2007.

[18] A. McCallum, K. Nigam, and L. H. Ungar. Efficient Clustering of High-dimensional Data Sets with Application to Reference Matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 169–178, New York, NY, USA, 2000. ACM.

[19] I. Jolliffe. *Principal Component Analysis*, pages 1094–1096. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[20] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In W. Gerstner, A. Germond, M. Hasler, and J. D. Nicoud, editors, *Artificial Neural Networks — ICANN'97*, pages 583–588, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[21] D. D. Lee and H. S. Seung. Algorithms for Non-negative Matrix Factorization. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 556–562. MIT Press, 2001.

[22] X. Lu, Y. Tsao, S. Matsuda, and C. Hori. Speech enhancement based on deep denoising autoencoder. In *INTERSPEECH*, 2013.

[23] L. van der Maaten, E. Postma, and H. Herik. Dimensionality Reduction: A Comparative Review. *Journal of Machine Learning Research - JMLR*, 10, 01 2007.

[24] A. Huang. Similarity Measures for Text Document Clustering, 2008.

[25] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[26] L. Kaufman and P. Rousseeuw. Clustering by Means of Medoids. *Data Analysis based on the L1-Norm and Related Methods*, pages 405–416, 01 1987.

[27] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction To Cluster Analysis*. 01 1990.

[28] H. S. Park and C. H. Jun. A simple and fast algorithm for K-medoids clustering. *Expert Systems with Applications*, 36(2, Part 2):3336–3341, 2009.

[29] C. P. Wei, Y. H. Lee, and C. H. Hsu. Empirical comparison of fast clustering algorithms for large data sets. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 10 pp.–, Jan 2000.

[30] J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1981.

[31] J. C. Bezdek, R. Ehrlich, and W. Full. FCM: The fuzzy c-means clustering algorithm. *Computers Geosciences*, 10(2):191–203, 1984.

[32] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[33] Z. Cebeci, F. Yildiz, A. T. Kavlak, C. Cebeci, and H. Onder. *ppclust: Probabilistic and Possibilistic Cluster Analysis*, 2019. R package version 0.1.2.

[34] J. H. Ward Jr. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[35] F. Murtagh and P. Legendre. Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion? *Journal of Classification*, 31(3):274–295, Oct 2014.

[36] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.

[37] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Trans. Database Syst.*, 42(3):19:1–19:21, Jul 2017.

[38] M. Ankerst, M. M. Breunig, H. P. Kriegel, and J. Sander. OPTICS: Ordering Points to Identify the Clustering Structure. *SIGMOD Rec.*, 28(2):49–60, Jun 1999.

[39] A. Ng. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[40] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, 2006.

[41] X. Glorot, A. Bordes, and Y. Bengio. Deep Sparse Rectifier Neural Networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[42] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. 2013.

[43] M. D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv preprint arXiv:1212.5701*, 2012.

[44] M. Meilă. Comparing clusterings—an information based distance. *Journal of Multivariate Analysis*, 98(5):873–895, 2007.

[45] P. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[46] J. C. Dunn. Well-Separated Clusters and Optimal Fuzzy Partitions. *Journal of Cybernetics*, 4(1):95–104, 1974.

[47] M. Hahsler, M. Piekenbrock, S. Arya, and D. Mount. *dbscan: Density Based Clustering of Applications with Noise (DBSCAN) and Related Algorithms*, 2018. R package version 1.1-3.

Appendix A: Tables

	Data set		
	`BuddyMove`	`TripAdvisor`	`Google`
Number of observations	249	980	5456
Number of attributes	7	11	25
Attribute 1	Unique user ID	Unique user ID	Unique user ID
	Number of reviews on...	*Average user feedback on...*	*Average ratings on...*
Attribute 2	stadiums, sports complexes, etc.	art galleries	churches
Attribute 3	religious institutions	dance clubs	resorts
Attribute 4	beaches, lakes, rivers, etc.	juice bars	beaches
Attribute 5	theaters, exhibitions, etc.	restaurants	parks
Attribute 6	malls, shopping places, etc.	museums	theaters
Attribute 7	parks, picnic spots, etc.	resorts	museums
Attribute 8	N/A	parks/picnic spots	malls
Attribute 9	N/A	beaches	zoos
Attribute 10	N/A	theaters	restaurants
Attribute 11	N/A	religious institutions	pubs/bars
Attribute 12	N/A	N/A	local services
Attribute 13	N/A	N/A	burger/pizza shops
Attribute 14	N/A	N/A	hotels/other lodgings
Attribute 15	N/A	N/A	juice bars
Attribute 16	N/A	N/A	art galleries
Attribute 17	N/A	N/A	dance clubs
Attribute 18	N/A	N/A	swimming pools
Attribute 19	N/A	N/A	gyms
Attribute 20	N/A	N/A	bakeries
Attribute 21	N/A	N/A	beauty & spas
Attribute 22	N/A	N/A	cafes
Attribute 23	N/A	N/A	view points
Attribute 24	N/A	N/A	monuments
Attribute 25	N/A	N/A	gardens

Table A1: Summary of the three data sets and information on the attributes.

Program name	Explanation
CLARA.R	Applies the CLARA algorithm to the non-encoded data.
data_cleaning_and_normalizing.R	Imports, cleans, and normalizes the non-encoded BuddyMove, Tripadvisor, and Google data sets.
DBSCAN.R	Applies the DBSCAN algorithm to the non-encoded data.
encoded_CLARA.R	Applies the CLARA algorithm to the encoded data.
encoded_data_cleaning_and_normalizing.R	Imports, cleans, and normalizes the encoded BuddyMove, Tripadvisor, and Google data sets.
encoded_DBSCAN.R	Applies the DBSCAN algorithm to the encoded data.
encoded_fuzzy_c-means.R	Applies the fuzzy c-means algorithm to the encoded data.
encoded_hierarchical_average.R	Applies the hierarchical clustering algorithm with average linkage to the encoded data.
encoded_hierarchical_complete.R	Applies the hierarchical clustering algorithm with complete linkage to the encoded data.
encoded_hierarchical_single.R	Applies the hierarchical clustering algorithm with single linkage to the encoded data.
encoded_hierarchical_ward.R	Applies the hierarchical clustering algorithm with Ward linkage to the encoded data.
encoded_k-means.R	Applies the k-means algorithm to the encoded data.
encoded_k-medoids.R	Applies the k-medoids algorithm to the encoded data.
encoded_nbclust_hierarchical.R	Determines the optimal number of clusters for the hierarchical clustering algorithms, based on the encoded data.
encoded_nbclust_partitioning.R	Determines the optimal number of clusters for the partitioning clustering algorithms, based on the encoded data.
encoded_OPTICS.R	Applies the OPTICS algorithm to the encoded data.
fuzzy_c-means.R	Applies the fuzzy c-means algorithm to the non-encoded data.
hierarchical_average.R	Applies the hierarchical clustering algorithm with average linkage to the non-encoded data.
hierarchical_complete.R	Applies the hierarchical clustering algorithm with complete linkage to the non-encoded data.
hierarchical_single.R	Applies the hierarchical clustering algorithm with single linkage to the non-encoded data.
hierarchical_ward.R	Applies the hierarchical clustering algorithm with Ward linkage to the non-encoded data.
k-means.R	Applies the k-means algorithm to the non-encoded data.
k-medoids.R	Applies the k-medoids algorithm to the non-encoded data.
nbclust_hierarchical.R	Determines the optimal number of clusters for the hierarchical clustering algorithms, based on the non-encoded data.
nbclust_partitioning.R	Determines the optimal number of clusters for the partitioning clustering algorithms, based on the non-encoded data.
OPTICS.R	Applies the OPTICS algorithm to the non-encoded data.

Table A2: Programs used in R.

Program name	Explanation
autoencoder.py	Creates autoencoders for the BuddyMove, Tripadvisor, and Google data sets to encode these data sets to a lower dimension.
data_cleaning_and_normalizing.py	Imports, cleans, and normalizes the non-encoded BuddyMove, Tripadvisor, and Google data sets.

Table A3: Programs used in Python.

Programming language	Package/tool	Usage
R	`scale` function	Data normalizing
R	`NbClust`	Determination of optimal number of clusters
R	`stats` package	K-means algorithm
R	`cluster`	PAM and CLARA algorithms
R	`ppclust`	Fuzzy c-means algorithm
R	`factoextra`	Hierarchical clustering and data visualization
R	`dbscan`	DBSCAN and OPTICS algorithms
R	`fpc`	Clustering evaluation measures
Python	`Keras`	Autoencoders

Table A4: Packages and tools used.

Appendix B: Algorithms

Algorithm B1: DBSCAN

Required parameters/functions:

- DB: Database containing all objects.
- $minPts$: The minimum number of data points (objects) needed to form a cluster (excluding one of the core points of that cluster).
- eps: Distance threshold for two objects to be considered neighbors.
- `dist`(A, B): Computes the distance between two objects A and B.
- `RangeQuery`(DB, P, eps; `dist`): Returns a list containing all objects from database DB which are within a distance of eps from object P, according to the distance function `dist`. These objects are considered to be neighbors of P. See Algorithm B2.

def `DBSCAN`(DB, $minPts$, eps):

```
C ← 0 // Cluster label
for each object P in DB do
    if label(P) is defined then
        continue // Skip objects which are already labeled
    end if
    Neighbors N ← RangeQuery(DB, P, eps, dist)
    if |N| < minPts then
        label(P) ← Noise // P is identified as an outlier
        continue
    end if
    C ← C + 1 // Increment cluster label
    label(P) ← C // P is identified as a core point
    N ← N \ {P}
    for each object Q in N do
        if label(Q) == Noise then
            label(Q) ← C // Q is identified as a border point
        end if
        if label(Q) is defined then
            // Q is already assigned to another cluster, or it is a border point of the current
            // cluster, therefore we do not have to consider its neighbors
            continue
        end if
        label(Q) ← C
        Neighbors M ← RangeQuery(DB, Q, eps, dist)
        if |M| ≥ minPts then
            // Q is identified as a core point, therefore we also have to consider its neighbors
            N ← N ∪ M
        end if
    end for
end for
```

Algorithm B2: RangeQuery

Required parameters/functions:

- DB: Database containing all objects.
- P: An object from DB.
- *eps*: Epsilon.
- `dist`(A, B): Computes the distance between two objects A and B.

def `RangeQuery`(DB, P, *eps*):

1: Neighbors $N \leftarrow \emptyset$
2: **for each** object Q in DB **do**
3: **if** dist(P, Q) $\leq$ *eps* **then**
4: $N \leftarrow N \cup Q$
5: **end if**
6: **end for**
7: **return** N

Algorithm B3: OPTICS

Required parameters/functions:

- DB: Database containing all objects.
- $minPts$: The minimum number of data points (objects) needed to form a cluster (excluding one of the core points of that cluster).
- eps: Distance threshold for two objects to be considered neighbors.
- `dist`(A, B): Computes the distance between two objects A and B.
- `RangeQuery`(DB, P, eps; `dist`): Returns a list containing all objects from database DB which are within a distance of eps from object P, according to the distance function `dist`. These objects are considered to be neighbors of P. See Algorithm B2.
- `core_dist`(DB, P, $minPts$, eps; `RangeQuery`): First computes the neighbors of object P as $N \leftarrow$ |`RangeQuery`(DB, P, eps)|. If it then holds that $|N| < minPts$, the function returns UNDEFINED. Otherwise (thus if $|N| \geq minPts$), the function returns the distance between P and the $minPts$-th closest neighbor of P.
- `updateSeeds`(DB, P, N, $Seeds$, $minPts$, eps; `core_dist`): Updates the priority queue $Seeds$ by enqueueing all neighbors of P from N which are not marked as processed yet. See Algorithm B4.

def `OPTICS`(DB, $minPts$, eps):

```
1: // List to store the objects, which are ordered according to their reachability-distance
2: OrderSeeds ← empty list
3: for each object P in DB do
4:     P.reachability_dist ← UNDEFINED
5: end for
6: for each unprocessed object P in DB do
7:     Mark P as processed
8:     Add P to orderSeeds
9:     if core_dist(DB, P, minPts, eps) != UNDEFINED then
10:        N ← RangeQuery(DB, P, eps)
11:        Seeds ← empty priority queue
12:        updateSeeds(DB, P, N, Seeds, minPts, eps)
13:        while Seeds is not empty do
14:            Q ← Seeds.dequeue()
15:            O ← RangeQuery(DB, Q, eps)
16:            Mark Q as processed
17:            Add Q to orderSeeds
18:            if core_dist(DB, Q, minPts, eps) != UNDEFINED then
19:                updateSeeds(DB, Q, O, Seeds, minPts, eps)
20:            end if
21:        end while
22:    end if
23: end for
```

Algorithm B4: updateSeeds

Required parameters/functions:

- DB: Database containing all objects.
- P: A core point.
- N: A list containing all neighbors of P.
- *Seeds*: Priority queue in which objects are in ascending order, according to their reachability-distance from the closest core point. The objects in this queue are likely to end up in the same cluster.
- *minPts*: The minimum number of data points (objects) needed to form a cluster (excluding one of the core points of that cluster).
- *eps*: Distance threshold for two objects to be considered neighbors.
- `dist`(A, B): Computes the distance between two objects A and B.
- `RangeQuery`(DB, P, *eps*; `dist`): Returns a list containing all objects from database DB which are within a distance of *eps* from object P, according to the distance function `dist`. These objects are considered to be neighbors of P. See Algorithm B2.
- `core_dist`(DB, P, *eps*, *minPts*; `RangeQuery`): First computes the neighbors of object P as $N \leftarrow |$`RangeQuery`(DB, P, *eps*)$|$. If it then holds that $|N| < minPts$, the function returns UNDEFINED. Otherwise (thus if $|N| \geq minPts$), the function returns the distance between P and the $minPts$-th closest neighbor of P.

def `updateSeeds`(DB, P, N, *Seeds*, *minPts*, *eps*):

```
1:  coredist ← core_dist(DB, P, eps, minPts)
2:  for each object Q in N do
3:      if Q is unprocessed then
4:          new_reachability_dist ← max{coredist, dist(P, Q)}
5:          if Q.reachability_dist == UNDEFINED then
6:              Q.reachability_dist ← new_reachability_dist
7:              Seeds.enqueue(Q, new_reachability_dist)
8:          else if new_reachability_dist < Q.reachability_dist then
9:              Q.reachability_dist ← new_reachability_dist
10:             Seeds.move_up(Q, new_reachability_dist)
11:         end if
12:     end if
13: end for
```