International Hellenic University

Department of Information and Electronic Engineering

Data Reduction through DBSCAN Clustering

Author:
Athina Drakopoulou

Supervisors:
Dr. Stefanos Ougiaroglou

June 17, 2020

Thesis Title: Data Reduction through DBSCAN Clustering
Thesis Code: 20121
Student: Athina Drakopoulou
Supervisor: Stefanos Ougiaroglou
Assignment Date: 27th March, 2020
Submission Date: June 17, 2020

I hereby declare that I am the author of this dissertation and that any help I received in order to prepare and complete it is properly acknowledged and cited. All the sources from which I used data, ideas, images and text are included, either the material was used as it is or paraphrased. In addition, I declare that the present assignment was entirely written by me as a dissertation for the Department of Information and Electronic Engineering of IHU.

The approval of the dissertation by the the Department of Information and Electronic Engineering of the International Hellenic University of Greece does not necessarily imply acceptance of the views of the author on behalf of the Department.

Τίτλος Π.Ε: Μείωση του πληθυσμού των δεδομένων μέσω του αλγόριθμου συσταδοποίησης DBSCAN
Κωδικός Π.Ε: 20121
Ονοματεπώνυμο φοιτητή: Αθηνά Δρακοπούλου
Ονοματεπώνυμο εισηγητή: Στέφανος Ουγιάρογλου
Ημερομηνία ανάληψης Π.Ε: 27 Μαρτίου 2020
Ημερομηνία περάτωσης Π.Ε: 17 Ιουνίου 2020

Βεβαιώνω ότι είμαι η συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια η οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως πτυχιακή εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Contents

1 Introduction **10**
1.1 k-Nearest Neighbors (kNN) Classifier . . . 10
1.2 Data Reduction Techniques . . . 14
1.3 Motivation and Contribution . . . 17
1.4 Thesis Organization . . . 18

2 Background Knowledge **20**
2.1 Prototype selection through data condensation . . . 20
2.1.1 Condensing Nearest Neighbour(CNN) . . . 20
2.1.2 IB2 . . . 22
2.2 Prototype Abstraction . . . 23
2.2.1 The algorithm of Chen and Jozwik . . . 23
2.2.2 Reduction by Space Partitioning algorithms . . . 25
2.3 Data Clustering . . . 27
2.3.1 Density Based Spatial Clustering of Applications with Noise . . . 30
2.3.2 Determination of Eps and MinPts . . . 35
2.3.3 OPTICS: Ordering Points To Identify the Clustering Structure . . . 36

3 Prototype Selection via DBSCAN Clustering **43**

4 Experimental Study **47**
4.1 Experimental Setup . . . 47
4.1.1 Datasets . . . 47
4.1.2 Parameters . . . 53
4.2 Experimental Results . . . 54

5 Conclusions and Future Work **58**

Appendices **66**

A PSDBSCAN Source Code **66**

B DBSCAN Parameters' Estimators **78**

List of Figures

1.1 k Nearest Neighbours classification process with k = 3 and k = 5.[1] 11
1.2 Hierarchical taxonomy of Data Reduction Techniques categories. 14
1.3 Prototype Selection Map. [2] 15
1.4 Prototype Generation Map. [3] 16
1.5 k-NN through data reduction.[1] 17
2.1 Initial training data and close=to-class-border data.[1] 20
2.2 Example of centroid-based clustering.[4] 28
2.3 Example of distribution-based clustering.[4] 29
2.4 Example of data set with irregular or intertwined clusters -formed based on their density.[5] 30
2.5 For $MinPts$ = 6, point p is directly density reachable from c and density reachable from q.[6] 32
2.6 Point p is density connected to point q through o 32
2.7 Sorted 4-dist graph for sample database.[7] 36
2.8 Clusters wrt. different density parameters.[8] 37
2.9 Illustration of core-distance and reachability-distances, MinPts=5. [9] . . . 38
2.10 Example of a data set with clusters of different densities and its reachability plot.[9] 42
3.1 (a) the initial data set in clusters and (b) the condensing set that can be extracted. 43
3.2 (a) the initial data set in clusters, (b) the condensing set of border items and outliers and (c) the condensing set of border items only. 45
4.1 Example of condensing set from figure 3.1. Circled items do not play a significant role during classification, even though they are border items. . . 55

List of Tables

4.1 Dataset description. 47
4.2 Landsat Satellite Classes. 49
4.3 Texture Classes. 50
4.4 Ecoli Classes. 51
4.5 Yeaststst Classes. 52
4.6 Parametric values. 53
4.7 Experimental results: Reduction Rate (%) 56
4.8 Experimental results: Accuracy (%) 56
4.9 Experimental results: Preprocessing Cost (millions of distance computations) 57

List of Algorithms

1 CNN-rule . . . 21
2 IB2 . . . 22
3 Chen's Algorithm . . . 24
4 RSP3 . . . 26
5 DBSCAN . . . 33
6 expandCluster . . . 34
7 OPTICS . . . 39
8 expandClusterOrder . . . 39
9 *OrderSeeds*.update . . . 40
10 extractDBSCAN-Clustering . . . 41
11 DBSCANcores . . . 44
12 DBSCANborders . . . 45

Abstract

Data classification is a popular and interesting field of research for both data mining and machine learning. For this reason, a quite big part of the available literature examines ways of minimizing the disadvantages of classification algorithms so that they can achieve the best possible accuracy with minimum computational cost and reasonable usage and management of resources. One of these ways is using data reduction techniques prior to the execution of the classifier on data sets of large size. These techniques intend to create a condensing set that is based on and represents the original set of data. Reducing the total number of items in the set automatically makes them easier to manage by the classifier, while leaving accuracy unaffected.

The purpose of the present study is to test whether an algorithm based on the DBSCAN clustering algorithm can be used as a Data Reduction Technique. DBSCAN is predominantly used for grouping data based on their close characteristics and is based on the density of spatial records as well as their labeling into core items, border items and outliers or noise. The purpose is to determine whether a condensing set containing only the elements that DBSCAN considers as border items is sufficient enough for the above algorithm to be considered a successful alternative data reduction technique. This is especially significant, since it would allow researchers and users of data mining and machine learning tools (such as Weka, MathWorks, etc) to do data reduction with pre-existing libraries and implementations. The sufficiency of the algorithm can be easily examined by comparing its results to the basic criteria for evaluating data reduction techniques (reduction rate, accuracy, preprocessing cost) with the corresponding results of other widely used data reduction algorithms.

Περίληψη

Η κατηγοριοποίηση δεδομένων αποτελεί ένα δημοφιλές και ενδιαφέρον κομμάτι έρευνας τόσο στον τομέα της εξόρυξης δεδομένων όσο και σε αυτόν της μηχανικής μάθησης. Για το λόγο αυτό, μια μεγάλη μερίδα της διαθέσιμης βιβλιογραφίας εξετάζει τρόπους με τους οποίους γίνεται εφικτή η ελαχιστοποίηση των μειονεκτημάτων των αλγορίθμων κατηγοριοποίησης έτσι ώστε αυτοί να πετυχαίνουν την καλύτερη δυνατή ακρίβεια με ελάχιστο υπολογιστικό κόστος και χρήση διαθέσιμων πόρων. Ένας από αυτούς τους τρόπους είναι και οι τεχνικές μείωσης δεδομένων μεγάλου όγκου προτού αυτά εξεταστούν από κάποιον κατηγοριοποιητή. Οι τεχνικές αυτές αποσκοπούν στην δημιουργία ενός συμπυκνωμένου συνόλου δεδομένων που βασίζεται και αντιπροσωπεύει το αρχικό σύνολο προς κατηγοριοποίηση. Η μείωση του πλήθους των εγγραφών-αντικειμένων του συνόλου τα κάνει αυτομάτως πιο εύκολα διαχειρίσιμα από τον κατηγοριοποιητή, ενώ παράλληλα αφήνει ανεπηρέαστη την ακρίβεια.

Η παρούσα εργασία έχει σκοπό την υλοποίηση ενός αλγορίθμου που βασίζεται στην λογική του αλγορίθμου συσταδοποίησης DBSCAN. Ο συγκεκριμένος αλγόριθμος χρησιμοποιείται κατ' εξοχήν για την ομαδοποίηση δεδομένων με βάση κοντινά τους χαρακτηριστικά (clustering) και βασίζεται στην πυκνότητα της κατανομής των εγγραφών στο χώρο και στον διαχωρισμό τους σε στιγμιότυπα πυρήνα, ορίων και θορύβου. Στόχος είναι το να εξακριβωθεί το αν ένα συμπυκνωμένο σύνολο που περιλαμβάνει μόνο τα στοιχεία που ο DBSCAN θεωρεί οριακά είναι επαρκές ώστε ο παραπάνω αλγόριθμος να θεωρηθεί επιτυχής εναλλακτική τεχνική μείωσης των δεδομένων. Αν κάτι τέτοιο είναι πράγματι εφικτό, επιτρέπει σε ερευνητές και χρήστες διάφορων εργαλείων εξόρυξης δεδομένων και μηχανικής μάθησης (όπως Weka, MathWorks, κλπ) να πραγματοποιούν μείωση δεδομένων χρησιμοποιώντας ήδη υπάρχουσες υλοποιήσεις και βιβλιοθήκες. Η αποτελεσματικότητα της προτεινόμενης τεχνικής μπορεί να εξεταστεί εύκολα συγκρίνοντας τα αποτελέσματα της στα βασικά κριτήρια αξιολόγησης τεχνικών μείωσης δεδομένων (βαθμός συμπύκνωσης δεδομένων, ακρίβεια, κόστος προεπεξεργασίας) με τα αντίστοιχα αποτελέσματα άλλων, ευρέως διαδεδομένων στην βιβλιογραφία, αλγορίθμων μείωσης δεδομένων.

1 Introduction

Classification, also known as supervised learning, has always been a popular subject of study among both data scientists and machine learning engineers. Classification algorithms, often referred to as classifiers, [10] are meant to classify previously unknown data items into a set of classes depending on available data sets consisting of already classified items. The use of such algorithms can be met in simple but convenient processes, such as email spam filtering or recommendation systems, as well as in more complex ones, such as image recognition for medical purposes. Thus, the interest around their workings and efficiency is no surprise.

Despite sharing the same motivation, that being the most accurate class prediction, classifiers can be easily divided into two discreet categories,[10] : (i) the eager classifiers, and, (ii) the lazy or instance based classifiers. Their difference lies on the way they work, though the available training set plays an essential role for the effectiveness in both cases. In particular, an eager classifier identifies unclassified items using a classification model that has been previously built by the algorithm itself after pre-processing the available training set, whereas a lazy classifier uses the training set as a classification model, formed each time a new item arrives to be assigned.

Given the research interest around the problem of the classification, literature is constantly enriched with various eager and lazy classifiers. Some worth to mention variants of eager classifiers are the classification decision trees [11]. Such classifiers process the available training set in order to build a tree structure, later used as their model. Another popular proposal are the artificial neural networks [12, 13], which perform the classification process after being "trained" by examining the classified items of the training set. There are also eager classifiers that build their model relying on probability rules (probabilistic classifiers), the most distinguished of them being the naive Bayes classifier. Also, there is a subcategory of eager classifiers that discover association rules within the available classified data and perform classification based on these rules. As for the lazy classifiers, the most characteristic examples are the well-known k Nearest Neighbours classifier [14, 15] and the case-based reasoning classification methods. The k Nearest Neighbours classifier will be used as a reference point for this study.

1.1 k-Nearest Neighbors (kNN) Classifier

The k-Nearest Neighbours (k-NN) classifier belongs to the category of lazy classifiers, since it does not construct any classification model. It is a simple and easy to implement algorithm and can be easily integrated in many systems, therefore is extensively used in many application domains. It is also easy to use for research purposes as it is analytically tractable and the error rate for the value of k=1 and unlimited items is asymptotically never worse than the Bayes' rate, that being twice the minimum possible [15].

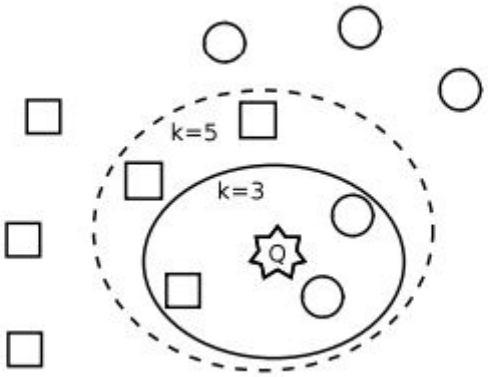

Figure 1.1: k Nearest Neighbours classification process with k = 3 and k = 5.[1]

As a lazy classifier, the k-NN uses the available training set whenever a new, unassigned item arrives in order to be classified. In detail, the algorithm retrieves from the training set the k nearest neighbours (classified items) of the unclassified item according to a previously calculated distance (usually the euclidean distance). The class of the new item is determined by the retrieved neighbours through a procedure known as the nearest neighbours voting; in fact the item is assigned to the class of the majority of them, known as the major class. When the parameter k, which refers to the number of neighbours to be examined, equals to one, the algorithm is known as the nearest neighbour classifier or 1-NN rule.

As illustrated in Figure 1.1, the result of the classification depends on the selected value of k, as the different number of neighbours can result to a different major class. In the example we can assume that our data set consists of items belonging in two different classes, square and circles, and that item Q arrives to be assigned in one of those classes. In the case of k=3, two out of the three nearest neighbours of Q are circles, therefore the item is classified as a circle. However, if we expand the neighbourhood by increasing k to the value of five, three out of the now five nearest neighbours belong to the class of squares, resulting to Q be assigned to their class instead.

Considering the fore mentioned example, the selection of the value of parameter k is critical. However, the actual value that can achieve the highest classification accuracy possible highly depends on the data set and the way the training items are placed. For this reason, the determination of the most appropriate k value requires tuning through costly trial-and-error preprocessing tasks for each data set in study and it is quite common that this value varies for each set. Despite the lack of a general rule to follow while searching for the best k value, it is observed that data sets with noise have better results with larger k values, since in this way the algorithm is allowed to examine larger neighborhoods. However, setting a larger number of neighbors cannot help in defining the borders of distinct classes. On the other hand, lower numbers of neighbors are not recommended

for training sets with noise, as they lead to noise-sensitive and consequently less accurate classifiers. It is also important to mention that the best k value is not necessarily optimal. Even though k-NN classifiers uses the k parameter to examine a neighborhood out of the data set, different k values might form more distinct neighborhoods for different areas of the available data space. In this case, dynamic determination of k can help to improve the final accuracy results of k-NN.

In cases of data sets that include two classes, it is necessary for the parameter k to have an odd value in order to avoid ties during the process of nearest neighbours voting. For non-binary classification problems (data sets with more than two different classes), k can have any value and even though ties can still occur, they are resolved either by selecting one of the most common classes in tie or by selecting the class of the nearest neighbour. Many data mining software tools such as Weka resolve ties through the first way mentioned above, but for this research we adopt the single nearest neighbour classifier solution.

The selection of the most appropriate metric to use for the calculation of distance between the items is also an important issue to consider. This decision mainly depends on the types of the attribute of the training items. The Euclidean distance is quite common for real or integer number attributes, such the ones existing in the data sets used for this study, and for this reason it is adopted in the experiments that follow. In particular, data items described by n attributes are considered as data points (or vectors) in the n-dimensional Euclidean metric space, and the Euclidean distance between points p and q is given by:

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + ... + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

For other types of attributes, there are plenty distance metrics (e.g., Mahalanobis, Manhatan, Minkowski, Chebyshev) [16] which can be adopted as well and there are even similarity measures, specifically proposed to handle attributes of nominal nature (non-metric spaces).

According to the formula stated above, the value of the distance can be reasonably affected by the different attribute ranges, as attributes with wide ranges have higher impact than attributes with narrower ranges, regardless of their significance (which is often the same). Consequently, it is required for the attributes to undergo a process called normalization. Normalization is the process of scaling the attributes of the data items to a predefined interval and easy to control range (e.g., [0, 1]) and some data mining software suites apply it by default. Assume that a data set contains n items and an attribute e should be normalized to [0, 1]. The attribute value of the i-th item, i = 1,..., n is normalized as

follows:

$$normalized(e_i) = \frac{e_i - E_min}{E_min - E_max}$$

where Emin and Emax are the minimum and maximum values for attribute e, respectively.

Even though the k-NN classifier have been proposed in many variations, the most popular one is the distance-weighted k-NN rule. In this variation, k-NN is enriched with a distance-weight function to boost the votes of the closer neighbors and thus "weighting" them as heavier than the further ones. In detail, the nearest neighbor is weighted by one, the furthest of neighbours is weighted by zero and the weights of the rest, in-between neighbors are scale to this range. In this case the major class is the class of the largest sum of weights and not just the class of the majority of the neighbors.

Despite the logic simplicity of the k-NN classifier, some weaknesses related to the way it works might render its use inefficient, thus should be mentioned as well. The most notable of those weaknesses is its high computational cost, either caused by a large number of available items to examine or by the data dimensionality of the data set. The k-NN must calculate all distances between each unknown item and all training items of the available data sets. The larger a data set is or the more attributes each of its items consists of, the more time-consuming and demanding the procedure becomes and even though nowadays systems are equipped with powerful processors, such calculations can be prohibitive in cases of time-constraint environments.

Since k-NN is a lazy classifier, another common weakness it shares with all the other algorithms of this category (in contrast to eager classifiers), is the large storage requirements for the available training sets. The k-NN classifier needs to be executed on computer systems with high storage capability because it does not construct a classification model, like an eager classifier would do. Instead, the training data sets must always be available to the algorithm, thus stored in the system.

Another weakness that should be mentioned is that the k-NN classifier is prone to data sets with noise. Its accuracy relies on the quality of the given data, which might include mislabelled items, outliers or regions of overlaps between two or more classes. Expanding the neighborhood under examination by using a higher k value can partially improve the accuracy in these cases, as mentioned above. However, the determination of the k value requires a high number of trial-and-error executions and this method can give positive results only in cases where we can safely assume that the noise is uniformly distributed.

The reduction of the computational cost of the k-NN is the main goal for many of the most recent research studies, but there is also interest towards other techniques that aim to resolve the rest of the weaknesses of this algorithm. A possible categorization of these methods is:

i Multi-attribute Indexes

ii Data Reduction Techniques

For the purposes of this study, we are going to focus on the Data Reduction Techniques.

1.2 Data Reduction Techniques

Data Reduction Techniques (DRTs) can either be associated with item reduction or dimensionality reduction, though we will focus on the first case. Item reduction techniques are divided into two categories, regarding the origin of the items they result to: (i) prototype selection algorithms [2], and, (ii) prototype abstraction [3] -or prototype generation- algorithms. In prototype selection, representative items known as "prototypes" are selected from the initial data set, while in prototype abstraction, the equivalent prototypes are generated items that represent the summary of similar training items of a specified data area of the multidimensional space.

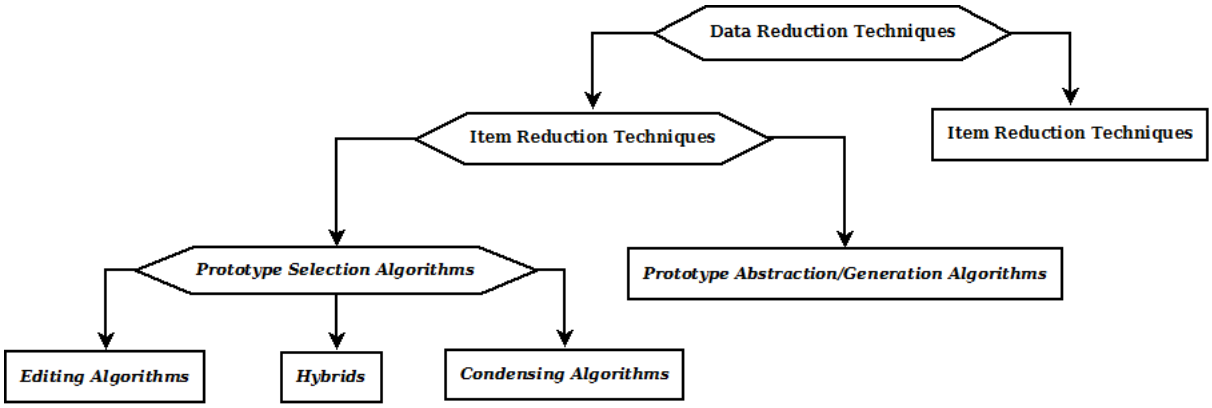

Figure 1.2: Hierarchical taxonomy of Data Reduction Techniques categories.

Prototype selection algorithms can be condensing, editing or hybrid algorithms, depending on the type of the selection they rely on. The type of selection refers to the type of search that is carried out by algorithms, focusing on the type of points they seek to retain, such as border, central or some other set of points. However, this is not the only sort of discrimination between prototype selection algorithms, as they can be classified depending on the order or the evaluation of search they use. Other factors that can help us to further assort prototype selection techniques may be classifier-dependant. Although we will not focus on these detailed subcategories, their number and different discrimination criteria are only indicative of the total amount of prototype selection algorithms that have been proposed during the previous decades. In Figure 1.3, a map of such algorithms is presented in chronological order, including graphic keys for each subcategory. The Filter and Wrapper shapes refer to the algorithm types depending on the evaluation of search, while the rest refer to the direction of search.

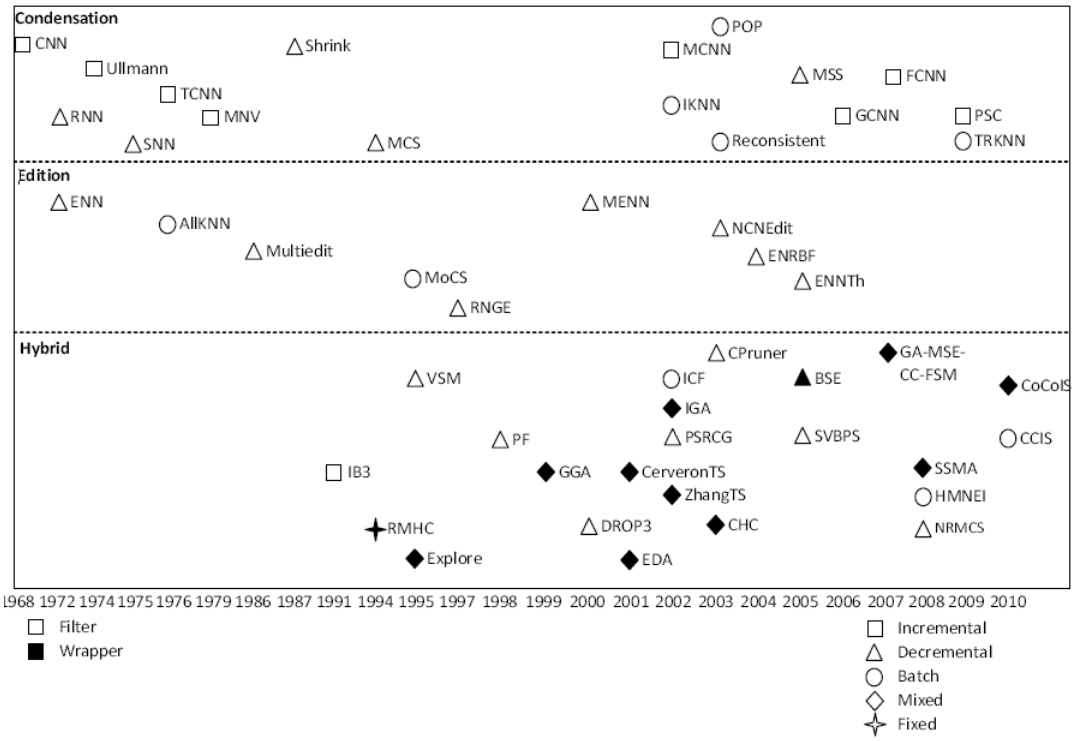

Figure 1.3: Prototype Selection Map. [2]

When it comes to condensing algorithms, they have the same motivation with the prototype abstraction algorithms, which is to built a smaller data set, known as the condensing set, that can accurately represent the original training set. Decreasing the amount of training items benefits the k-NN classification in terms of computational cost and storage requirements while maintaining high classification accuracy. Editing algorithms on the other hand aim to increase the accuracy by improving the quality of the data. This can be achieved by removing misleading items, such as noise and outliers, and by smoothing the borders between classes. Last but not least, hybrid methods, as their name indicates, aim to the removal of either internal or border points by combining the logic of the two previous strategies, trying to maintain the smallest subset S that can be used to preserve or even increase the classification accuracy. Even with a small subset selected as the ideal condensing set, the kNN classifier is found to be highly adaptable to these methods.

Like in prototype selection, prototype abstraction, also known in literature as prototype generation, can also be divided in less known subcategories depending on different factors, some of which are quite similar to the discrimination criteria of prototype selection algorithms. Although there were noticeably fewer prototype abstraction proposals during the past years, they too can be classified depending on (a) the type of reduction -that being anything between Incremental, Decremental, Mixed or Fixed- (b) the evaluation of search -Filter, Semi-Wrapper or Wrapper- (c) the generation mechanisms -namely class relabeling, centroid based mechanisms, methods of space splitting or techniques of positioning adjustment- and (d) the type of the resulting generation set -condensing, editing

or hybrid. Figure 1.4 is the respective chronological map of all prototype abstraction algorithms that have been proposed.

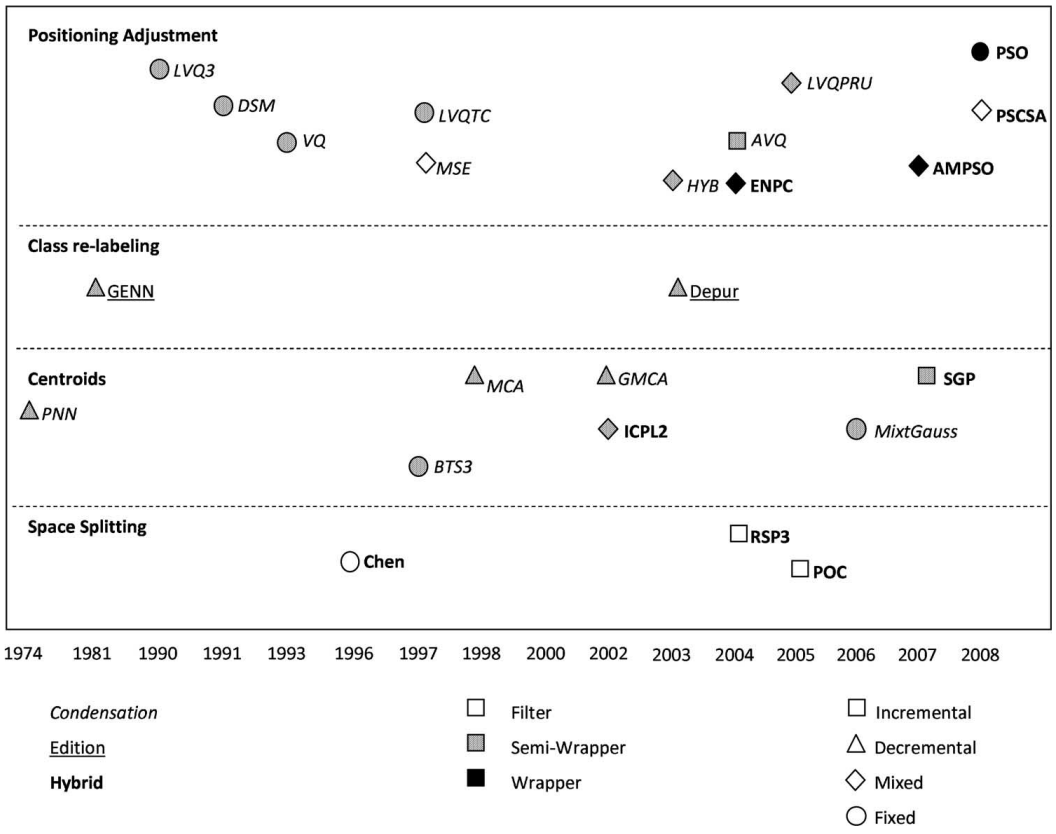

Figure 1.4: Prototype Generation Map. [3]

The efficiency of data reduction techniques can be estimated appropriately according to three criteria:

i Reduction Rate, a number that represents how smaller the size of the final, condensing set is compared to the size of the original training set. It is actually calculated as the ratio of the number of the removed items over the number of the items initially placed in the data set. High reduction rate values result to faster kNN execution, as the algorithm has to deal with lower number of items.

ii Classification Accuracy of kNN classifier when applied to the condensing set.

iii The Computational Cost required to build the condensing set, known as preprocessing cost.

Even though the significance of each of those criteria over the others varies depending on the domain, all of them should satisfy minimum requirements. For example, a higher reduction rate might increase the speed of the classification process, but depending on the data set, after a certain point, it might affect the accuracy of the classifier.

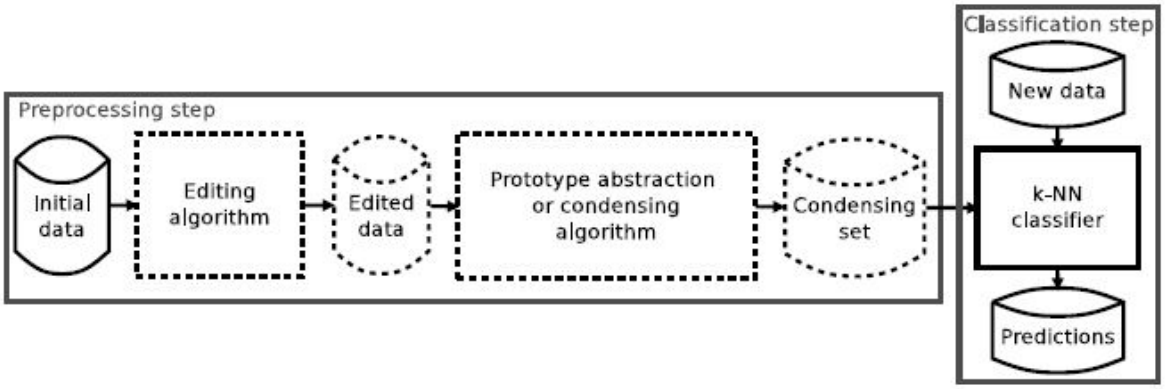

Figure 1.5: k-NN through data reduction.[1]

While condensing algorithms share the same motivation as the prototype abstraction algorithms, the editing algorithms have a different approach as they focus on the quality of the data. However they can be used beforehand in order to improve the performance of condensing or prototype abstraction algorithms, since their produced reduction rate depends on the level of noise of the data. In particular, high level of noise in the training set result to lower reduction rate. Therefore the usage of an editing algorithm is suggested not only for improving the classification accuracy but also for a more effective application of other data reduction techniques.

Figure 1.5 demonstrates the kNN classification in two discreet phases, preprocessing, which is optional, and the classification itself. In advance, we can distinguish four types of preprocessing:

i no preprocessing which is acceptable for datasets of small size and with no noise,

ii only editing, which is recommended for small data sets with noise,

iii only condensing, appropriate for data sets that are noise-free (thus not in need for editing), but still large in size,

iv and finally, both editing and condensing (as displayed in figure 2), which is the type of preprocessing required for large training sets with noise.

The complete data reduction preprocessing procedure aims to building a condensing set that is noise-free and contains an adequate number of representative prototypes of each class, in other words, an ideal data set for successful kNN classification.

1.3 Motivation and Contribution

Considering the benefits of data reduction techniques in eliminating the weaknesses of classifiers and specifically the ones regarding management of big data, research and pro-

posals of techniques with the same motivation is something quite useful. In particular, this study aims to prove whether or not DBSCAN, a distinguished clustering algorithm, can be used as a prototype selection algorithm with success. DBSCAN (Density Based Spatial Clustering of Applications with Noise) is, as its name indicates, a clustering algorithm that supports and relies on the notion of the density of data, meaning it perceives the elements of the data set not as particular objects but as a substance that fills the data space. Although DBSCAN will be described in detail in Section 2, the main concept is that if we create a condensing set which only consists of the border items generated by DBSCAN, it might be as adequate and accurate in respect to the original data set as a condensing set generated by a typical data reduction algorithm, meaning that DBSCAN is able to generate precise prototypes as well. This assumption relies on the concept that the non-close-to-border items are redundant and can be removed, whereas border items are more crucial and useful to a classifier, thus they should be retained. In a similar manner, items that lie in high density areas, like within the boundaries of a cluster, are not as pivotal during classification, especially in cases of data sets where each cluster reflects a separate class. In any case, the DBSCAN borders or even the outliers that lie between clusters, fit to the criteria of a usual condensing algorithm in order to be retained and then used as a condensing set.

In case of successful results, we can be able to use DBSCAN, for a different purpose, that of data reduction, in addition to the one it was originally developed for, which is clustering. This is quite useful considering DBSCAN, being a relatively old, yet popular algorithm, is already implemented and included in many data mining tools such as Weka, MathWorks, scikit-learn and the R programming language. This is not the case for other data reduction techniques. Therefore, a successful usage of a pre-existing algorithm for data reduction would allow researchers to improve the classification process in cases where condensing is required, by already having an alternative that can work as effectively as a regular prototype selection algorithm.

1.4 Thesis Organization

This has the following layout. Section 2 provides some background knowledge required for the later parts of the study to be comprehensible. In particular, some distinguished data reduction algorithms that will later be used for comparison purposes are presented. Data clustering and DBSCAN will be presented as well, since it is the clustering algorithm in which renders the base of the alternatives we propose as new data reduction techniques. Those alternatives will be shown in Section 3. Section 4 begins with some details about the experimental set up, the data sets that were examined, the parametric values that were chosen for the algorithms we tested and the final results of the experiments in comparison with the results of the fore mentioned data reduction algorithms of Section 2. The results that are to be compared regard all three main data reduction evaluation

criteria: Reduction Rate, Classification Accuracy and Computational Cost. Finally, the conclusions and suggestions for future work are included in Section 6.

2 Background Knowledge

2.1 Prototype selection through data condensation

In prototype selection, the prototypes are selected from the initial data set, so they are actually original training items that have not undergo any sort of modification. Although prototype selection includes condensing, editing and hybrid algorithms, editing algorithms focus more in improving the quality of the data by removing noise and are not recommended solely as a data reduction technique but also as an assisting stage before the execution of condensing or abstraction algorithms. Also hybrid algorithms are characterized by combining both editing and condensing strategies, so they inevitably include some of the logic of an editing algorithm, in terms of noise removal. However, the motivation of condensing algorithms, which is to built a smaller data set through collecting essential, close to class borders items as prototypes, is closer to the motivation of the algorithm that will be proposed and compared with existing techniques later in this study. Therefore, we focus on data condensation. The following two algorithms of this subcategory of prototype selection are presented and used for comparison purposes.

2.1.1 Condensing Nearest Neighbour(CNN)

The Condensing Nearest Neighbour rule [17] is used in many papers as a reference algorithm, for comparison purposes and like many other data reduction techniques, tries to place into the condensing set only items that are essential for a successful classification procedure.

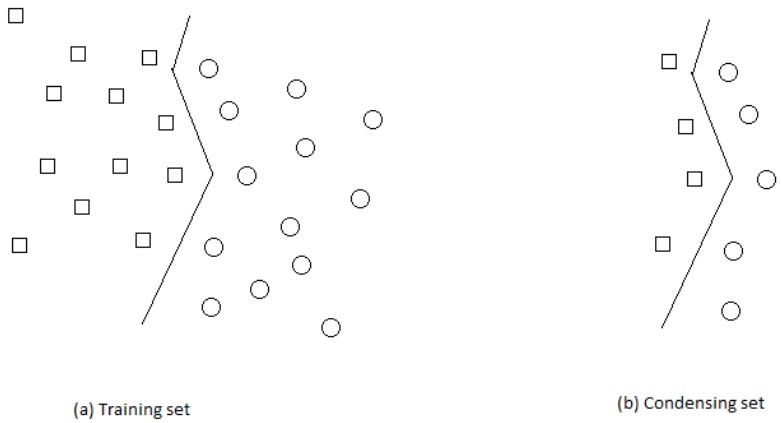

Figure 2.1: Initial training data and close=to-class-border data.[1]

Items lying close to the class decision boundaries of the data space belong to this category, therefore can be useful as prototypes. On the other hand, items that are located in data areas inside those boundaries are useless during the classification phase, thus can be

removed without causing a negative impact in the accuracy of the classifier. Consequently, the classifier will be able to achieve the same accuracy using the condensing set (the usage of whom involves lower computational cost and storage requirements) instead of the original training set. Figure 2.1 displays the clear difference in the number of items of each data set and also makes visually comprehensible which of these items are more needed in the condensing set.

As shown in Algorithm 1, CNN begins accepting the available training set as a parameter and outputs the condensing set. Initially, a random item of the training set is placed to the currently empty condensing set. Then a loop is performed so that all the remaining items of the training set are classified by 1NN (whose way of working was explained in section 1.1) based on the items of the condensing set. In case an item is misclassified according to 1NN it is moved to the condensing set. The algorithm completes the condensing set construction once there are no moves of items from the training set to the newly constructed set during a complete loop execution. The remaining items of the training set are discarded and the condensing set is ready to be used.

Algorithm 1 CNN-rule

Input: $TrainingSet$ **Output:** $CondensingSet$

1: $CondensingSet \leftarrow \varnothing$
2: pick an item of $TrainingSet$ and move it to $CondensingSet$
3: **repeat**
4: $stop \leftarrow TRUE$
5: **for** each $x \in TrainingSet$ **do**
6: $NN \leftarrow$ Nearest Neighbour of x in $CondensingSet$
7: **if** $NN_{class} \neq x_{class}$ **then**
8: $CondensingSet \leftarrow CondensingSet \cup \{x\}$
9: $TrainingSet \leftarrow TrainingSet - \{x\}$
10: $stop \leftarrow FALSE$
11: **end if**
12: **end for**
13: **until** $stop == TRUE$ {no move during a pass of $TrainingSet$}
14: discard $TrainingSet$
15: **return** $CondensingSet$

Although the CNN reasonably assumes that mislabeled items are close to class boundaries areas, items that are actually noise are obviously an exception to this rule and CNN might wrongly move them and their neighborhoods to the condensing set. Therefore it is clear that CNN is prone to high level of noise, in terms of reduction rate and preprocessing cost. Another factor which might also have a negative effect on the reduction rate is the number of discreet classes included in the data set. High number of classes indicates larger areas of boundaries and consequently more prototypes need to be collected.

An important advantage of the CNN rule is that it determines the final number of selected prototypes automatically and works without user-predefined parameters. Also the training items that are not included in the condensing set are classified correctly by 1NN based on the content of the resulting condensing set, achieving 100% accuracy. A weakness of the condensing set however is that it depends on the order of the condensing items in the initial data set. In fact, CNN can result to a different condensing set when it is applied in the very same training set but with different order of items.

It is also worth to mention that the CNN rule is non-incremental, meaning that it cannot update an already constructed condensing set by adding data from a new training set. The CNN rule involves multiple passes over the initially available data in order to form a condensing set but it can not handle new training items to update it later. The only way for the algorithm to lead to an updated condensing set is to be executed anew over the updated training set instead. The updated training set however includes both new and old items which did or did not enter the initial condensing set at first. This means that those items, even if they were originally discarded at the end of the algorithm, must be retained in memory, in case of later updates. For this reason, CNN rule is not recommended for dynamic streaming environments, where new data is gradually available.

2.1.2 IB2

Another prototype selection algorithm for data condensation that belongs to the subcategory of Instance Based Learning algorithms (IBL) [18, 19] is IB2. IB2 is based on the CNN rule but with the major difference that it involves only one pass over the items of the training set in order to build the condensing set. In detail, each item of the training set is classified by 1NN and in case the item is classified correctly, it is instantly discarded from the training set, otherwise is moved to the condensing set.

Algorithm 2 IB2

Input: $TrainingSet$ **Output:** $CondensingSet$

1: $CondensingSet \leftarrow \varnothing$
2: pick an item of $TrainingSet$ and move it to $CondensingSet$
3: **for** each $x \in TrainingSet$ **do**
4: $\quad NN \leftarrow$ Nearest Neighbour of x in $CondensingSet$
5: $\quad$ **if** $NN_{class} \neq x_{class}$ **then**
6: $\quad\quad CondensingSet \leftarrow CondensingSet \cup \{x\}$
7: $\quad$ **end if**
8: $\quad TrainingSet \leftarrow TrainingSet - \{x\}$
9: **end for**
10: **return** $CondensingSet$

Because of the similarities of the CNN rule, IB2 does also not require predefined parameters and the condensing set it produces depends significantly on the order of the training

data. Since it is a one pass algorithm, it is much faster and it has lower preprocessing and computational cost than the CNN rule. However, unlike CNN, IB2 can not guarantee that all the discarded training items can be correctly classified in the content of the final condensing set.

Furthermore, an important advantage of IB2 is that it can build its condensing set incrementally, meaning that it can handle new training data even after the creation of the "final" output set. In particular, unlike CNN-rule, new training data can be taken into account when updating an existing condensing set, regardless the discarded training items that were removed while building the condensing set in its current form, in the first place. Consequently, there is no need for the training set to be constantly available even after it has been used, which allows the algorithm to be applied in systems whose memory is not necessarily capable of storing large amount of training (and possibly increasing) data. Moreover, IB2 can handle new class labels and because of its ability to update existing condensing sets incrementally, it is appropriate for dynamic and streaming environments.

2.2 Prototype Abstraction

Prototype abstraction algorithms also share the same motivation with condensing approaches. Their major difference with the condensing algorithms lie in the way they build the condensing set, since instead of selecting some pre-existing training items as prototypes they generate new prototypes by summarizing on similar items. As a result, the final data set in which the classifier is executed is actually an artificial training set. This subsection includes two well-known prototype abstraction algorithms, one of which will be used later on for comparison purposes.

2.2.1 The algorithm of Chen and Jozwik

As mentioned before, prototype abstraction algorithms share the same motivation with condensing algorithms, despite having a different approach. As a result it is important to mention some well-known examples belonging to this category, one of which is the Chen and Jozwik algorithm, known as Chen's Algorithm [20]. In contrast to prototype selection, prototype abstraction involves generating new prototypes by examining and summarizing similar data areas of the training set, leading to an artificial final set. Chen's Algorithm in particular accepts the available training set to be abstracted and the n number of prototypes that are to be generated for the condensing set. The algorithm works by gradually dividing the initial training set to subsets, therefore it uses a data structure that stores each of these subsets. In the beginning of the execution, the entire training set is considered a subset.

Chen's Algorithm initially retrieves the most distant items x and y of the subset (currently entire data set) and calculates the distance between them, a number referred to as the

diameter of the subset. After that, Chen's Algorithm divides the subset based on the following rule: items that are closer to item x belong to subset X while items closer to item y belong to subset Y. When the two subsets are created, they are added in the data structure that is specifically used to store them, while the original subset from which the two new ones originated is removed from this structure. After the first complete division, Chen's Algorithm has to determine which subset will be divided next. In fact, it proceeds by dividing the subset that contains items of more than one classes, also known as non-homogenous sets, and has the largest diameter. This happens because it is assumed that subsets of larger diameter contain more items, thus through dividing them first we can achieve a higher reduction rate. In case all subsets are homogenous, the algorithm proceeds following the same higher diameter criterion for the division determination. The procedure of constructing subsets through division continues like this until the desired, user-predefined n number of subsets is reached.

Algorithm 3 Chen's Algorithm

Input: $TrainingSet$, $userSpecifiedValue$
Output: $CondensingSet$

$S \leftarrow \varnothing$
add(S, $TrainingSet$)
for $i = 2$ to $userSpecifiedValue$ **do**
  $currentSubset \leftarrow$ select the non-homogeneous subset $\in S$ with the largest diameter
  if $currentSubset == \varnothing$ {All subsets are homogeneous} **then**
    $currentSubset \leftarrow$ select the homogeneous subset $\in S$ with the largest diameter
  end if
  $(S_x, S_y) \leftarrow$ divide C into two subsets
  add(S, S_x)
  add(S, S_y)
  remove(S, C)
end for
$CondensingSet \leftarrow \varnothing$
for each subset $T \in S$ **do**
  $r \leftarrow$ compute the mean item by averaging the items in T
  $r.label \leftarrow$ find the most common class label in T
  $CondensingSet \leftarrow CondensingSet \cup \{r\}$
end for
return $CondensingSet$

When that happens, the last step of the algorithm is to create a mean item for each subset. A mean item is an item that summarizes the average values of the items of the subset and belongs to the same class of the majority of them. In fact, The mean item m of each subset S, is computing by averaging the t attribute values of items x_i, $i = 1, 2, \ldots, |S|$

that belong to S. Therefore, the t average attributes $m.d_j$ of m are computed as follows:

$$m.d_j = \frac{1}{|S|} \sum_{x_i \epsilon S} x_i.d_j, j = 1, 2, \ldots, t$$

The mean items created for all subsets are the final content of the condensing set.

Unlike IB2 and CNN, Chen's Algorithm builds the same condensing set regardless of the ordering of the data. However, it has two drawbacks that should be mentioned:

i It is a parametric algorithm, as the user specifies the number of subsets and consequently prototypes to be generated. Although this allows the user to control the size of the condensing set, it is also prohibitive for the automatic determination of its size in accordance with the nature and size of the available training data. Like in many cases of parametric algorithms, the determination of the most appropriate value of n includes a trial-and-error procedure which oftentimes be time-consuming and implies high computational cost.

ii The items that do not belong in the most common class of the subset are practically ignored, therefore they are not properly represented in the condensing set.

2.2.2 Reduction by Space Partitioning algorithms

The Chen and Jozwik algorithm has been a motivation and base for the Reduction By Space Partitioning algorithms, a set of prototype abstraction algorithms. Each of them aims to eliminate one or both of the two weak points of Chen's Algorithm. Like Chen's Algorithm and unlike prototype selection algorithms, all RSP algorithms construct condensing sets that do not rely on the ordering of the training data.

RSP1 deals with the second drawback, by computing as many mean items as the number of different classes in each subset. In other words, it attempts to improve classification accuracy by creating a more representative condensing set, which is only possible trough taking into account all training items, including the ones that do not belong in the major class of a subset. Obviously this results to constructing a larger condensing set than the one Chen's Algorithm would.

Although RSP1 uses the subset diameter as the splitting criterion, just like Chen's Algorithm, RSP2 differs on the way it selects the next subset to be divided by using the criterion of the highest overlapping degree [21]. According to this, items that belong to a specific common class are placed closer to each other, while items of a different class are placed as far as possible from the first ones. The overlapping degree of a subset is the ratio of the average distance between items of different classes and the average distance between items of the same class.

RSP3 on the other hand follows the concept of homogeneity by continuously splitting the non-homogenous subsets until all of them become homogenous, meaning that they result in subsets which consist of only items belonging in the same class. Although RSP3 can use either the largest diameter or the highest overlapping degree criterion, this choice is of secondary importance since the main goal of the algorithm is to divide all non-homogenous subsets. In this way, RSP3 is the only non-parametric algorithm and the size of the condensing set it produces is determined automatically. Consequently, RSP3 eliminates both weaknesses of Chen's Algorithm.

Algorithm 4 RSP3

Input: $TrainingSet$
Output: $CondensingSet$

```
1:  S ← ∅
2:  add(S, TrainingSet)
3:  CondensingSet ← ∅
4:  repeat
5:     C ← select the subset ∈ S with the highest splitting criterion value
6:     if C is homogeneous then
7:        r ← calculate the mean item by averaging the items in C
8:        r.label ← class of items in C
9:        CondensingSet ← CondensingSet ∪ {r}
10:    else
11:       (D1, D2) ← divide C into two subsets
12:       add(S, D1)
13:       add(S, D2)
14:       remove(S, C)
15:    end if
16: until IsEmpty(S)
17: return CondensingSet
```

According to the pseudo-code displayed above, RSP3 uses a data structure to store unprocessed subsets and like in Chen's Algorithm, the entire data set is initially considered such a subset, therefore it is added in the structure. Like the other algorithms, RSP3 selects the subset with the highest splitting criterion value and checks it for homogeneity. If all the items of the subset are indeed of the same class, they are used for the computation of the mean item of the subset which is later added in the condensing set. If however the subset is non-homogenous, it is divided in two others in the same way it would with Chen's Algorithm. Similarly, the two new subsets replace the one they originated from in the structure that stores unprocessed subsets so that they can be examined separately. This loop continues until the forementioned structure has no unprocessed subsets left. This indicates that all subsets that occurred throughout the loop have been divided to the point where they produced strictly homogenous subsets that have been later used for generating mean items as prototypes.

Considering RSP3, we observe that most of the prototypes it generates represent data areas that are closer to class boundaries. RSP3 is also prone to data noise, as higher levels of noise lead to smaller subsets that are greater in number. Consequently, more prototypes are generated and the reduction rate is lower. Another issue that should be mentioned is the computational cost required for the algorithm to calculate all distances between the training items in order to identify the most distant ones. The preprocessing cost of such procedures, especially for large data sets, may render the execution of the algorithm prohibitive.

2.3 Data Clustering

Even though all the previous algorithms of Section 2 are data reduction algorithms, the main purpose of this thesis is to determine whether DBSCAN, a clustering algorithm, can be used as an alternative data reduction technique.

Clustering is a term referring to the grouping of the objects of a database into meaningful subclasses called clusters. In general, clustering can be defined as a multi-objective optimization problem, since it can be achieved by various algorithms -more than 100 clustering algorithms have been proposed during the last few decades- that of course have remarkable differences in their understanding of what constitutes a cluster. These differences are inevitably affecting the way each algorithm works in order to efficiently detect clusters for a given data set. In fact, the appropriate clustering algorithm as well as the parameter settings it will most likely require may vary regarding the application domain, type of data and intended use of the results. As an iterative procedure of knowledge discovery, it usually requires trial and error tasks, especially when parametric algorithms are involved. In machine learning terminology, clustering is a method of unsupervised learning, commonly used as a main task of exploratory data mining or for statistical data analysis. Other fields in which it is frequently used include machine learning, pattern recognition, image analysis, information retrieval, bio-informatics, data compression, and computer graphics.

As mentioned, since the notion of a cluster is not -and neither can be- defined explicitly, the only generalization on which all clustering algorithms can rely on is that a cluster is a group of data objects. There are of course different clustering models depending on researchers' different approaches that aim to specify the grouping of the objects. Comprehending those models is crucial for understanding the different types of clustering algorithms and possibly determine which ones are more appropriate for certain domains and problems -though as noted, in most cases, this decision requires experimenting. Some of the most prominent examples of clustering algorithms' categories are listed below.

i **Connectivity-based.** The main speculation of connectivity-based clustering is that neighboring points are more related to each other than to points lying farther

away in the data space. Algorithms of this category form clusters by connecting objects while taking into account the in between distance of them. In this case, a cluster can be described largely by the maximum distance required to connect its parts and different distances will result to different clusters. These clusters can be later displayed in a dendrogram where the y-axis represents the different distance values and the x-axis is where the objects are placed. Through this sort of representation of a data set, these algorithms do not simply form discreet clusters but also provide a detailed visual hierarchy of them, since we can see which of those clusters can merge and at what distance value that can that happen. This is why connectivity-based clustering is also known as hierarchical clustering in literature.

Connectivity-based clustering algorithms can be classified depending on the way they compute distances, the linkage criterion by which the objects are connected -minimum, maximum or average of object distances- or the way they start forming the cluster hierarchy, either by adding items one by one or by dividing the data set into partitions. It must be noted that they can be easily misleaded by outliers, since those items might either appear to a hierarchical algorithm as additional clusters or cause the merging of other clusters, which is known as the chaining phenomenon.

ii **Centroid-based.** In this case, we assume that for a predefined number k of clusters, each point p of the data set is assigned to the cluster whose mean or central point –or just centroid- lies closer to p, in a manner that the squared distances from the cluster are minimized. In this way, each cluster is represented by its mean point, that serves as a prototype without necessarily being an actual point of the data set, so we have all clusters in the form of a central vector.

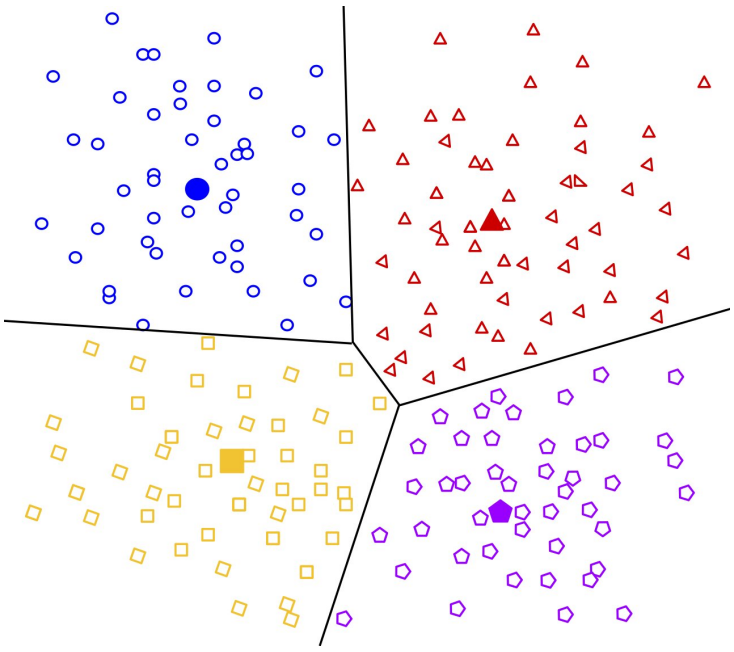

Figure 2.2: Example of centroid-based clustering.[4]

Computationally, this optimization problem is considered difficult and is therefore preferred to search only for approximate solutions. In order to find a sufficient local optimum, experimental runs with random initializations are required. However, some variations of k-means are able to choose the best of multiple runs. Other alternatives restrict cluster means by allowing them to only be picked from the existing data set points, while others focus on selecting the centers of the first run less randomly. In most cases of those algorithms, the number of clusters must be defined in advance, which is of course a disadvantage. Also, since they focus and work around cluster centers, they most likely detect similarly sized clusters, often ignoring borders.

This method is generally popular in machine learning due to the conceptual similarities it shares with the nearest neighbor classification. Another interesting use of k-means, aside from clustering, is the way it can be used successfully as a data reduction technique that tolerates noise, as proved experimentally in recent research[22].

iii **Distribution-based clustering.** This model of clustering relies on distribution models and is mainly related to statistics. Objects of the data set that are likely to belong to the same distribution, constitute a cluster. This approach resembles the way artificial data sets are generated, which is by sampling random objects from a distribution. The weakness of such methods is that they require a usually hard to detect certain model complexity or set of constraints. In case of low or zero model complexity, it is easy for these methods to fail due to overfitting.

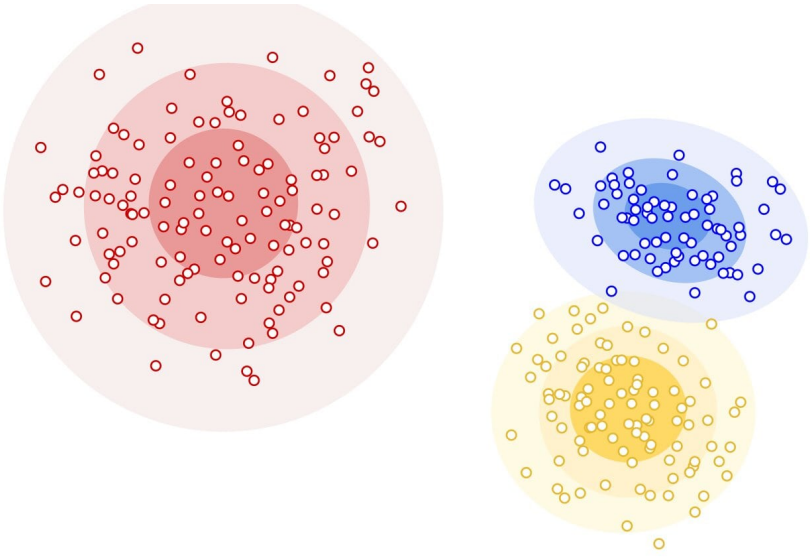

Figure 2.3: Example of distribution-based clustering.[4]

iv **Density-based clustering.** Although the notions of this category will be explained in detail through DBSCAN, which renders the base of the motivation of this dis-

sertation, a short description of a density based cluster would be a data set area of high density -or higher than each surrounding areas. Thinner -in terms of density- data set areas might include either noise or outliers, but both are treated the same since they are ignored.

To conclude, given a set of data items, clustering algorithms are used to assign each of those items into a specific cluster depending -each on a different way- on their properties. In theory, items that have similar properties and features naturally lie in the same or in neighbor areas in the data space and therefore possibly belong in the same cluster, while items of different properties belong to other discreet areas. The distance between the different groups rely on the difference between the features of the items.

DBSCAN seemed to be the most fitting for the purposes of this thesis, since it includes the definitions or criteria that divide items of the same cluster according to their position in it. This is quite important since, as mentioned before, items lying close to the class decision boundaries -or cluster borders for the matter, assuming that each cluster reflects a class- are more essential during classification while items that are located in data areas inside those boundaries are of no use to the classifier, thus can be removed during the construction of a condensing set.

2.3.1 Density Based Spatial Clustering of Applications with Noise

In order to proceed in explaining the DBSCAN and the way it divides items according to their position during the forming and expanding of clusters, it is crucial to mention some important points of the density based notion of clusters.

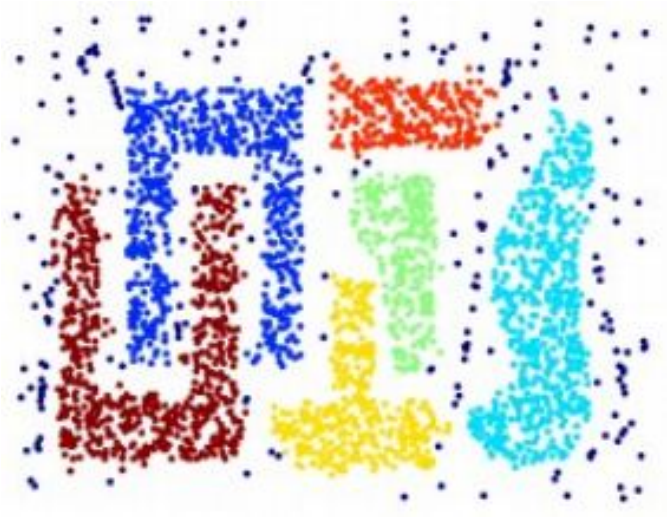

Figure 2.4: Example of data set with irregular or intertwined clusters -formed based on their density.[5]

In the case of a density based algorithm, the data items are no longer perceived as partic-

ular objects but as a substance that fills the data space. Areas of a large amount of items are characterized as high-density areas, whereas the opposite case refers to low-density areas. High density areas, or generally speaking, areas of density that exceeds the density of their surrounding space, can be easily recognized as clusters. Items of areas outside of the boundaries of any of the found clusters, which of course have much lower density, can be either noise or outliers. In order to set a threshold to the difference of density that separates a cluster from its surroundings, we assume that the neighborhood of each point of a cluster for a given radius, contains at least a minimum number of points. The shape of the neighborhood highly depends on the metric used for the distance function between two points p and q, denoted by dist(p,q), though this particular algorithm does not impose the usage of a specific metric. For instance, the usage of Manhattan distance results to rectangular neighborhoods, while the Euclidean distance, which is frequently used in related studies for simplicity purposes, results to spherical neighborhoods. In general, the most appropriate metric varies among different application domains and of course the nature of the data.

Below we recall some formal definitions of a density based cluster and related notions.

Definition 1 (Eps-neighborhood of a point) *The Eps-neighborhood of a point p, denoted by $N_{Eps}(p)$, is defined by $N_{Eps}(p) = \{q \in D | dist(p,q) \leq Eps\}$*

Note that not all points of a cluster have the same amount of neighbors. In fact, the items placed in the inner areas of the cluster, known as core points, have reasonably a much higher number of neighbors in comparison with the items found close to the boundaries of the cluster, referred to as border items. In order to identify where the borders of a cluster lie we must define the minimum number of points that the $N_{Eps}(p)$ must have so that p can be labeled as a core point. Additional points belonging to the neighborhood of a core point without being core points themselves are still part of the cluster, but labeled as border items. This is better described in the following definitions.

Definition 2 (Directly Density Reachable) *A point p is directly density reachable from a point q wrt. Eps, MinPts if*

i $p \in N_{Eps}(q)$

ii $|N_{Eps}(q)| \geq MinPts$ *(core point condition)*

Directly density-reachable is only symmetric for pairs of core points as it does not apply for one core and one border point as shown in the figure below.

Definition 3 (Density-reachable) *A point p is density reachable from a point q wrt. Eps, MinPts if there is a chain of points $p_1, ..., p_n, p_1 = q, p_n = p$ such that $p_i + 1$ is directly density reachable from p_i.*

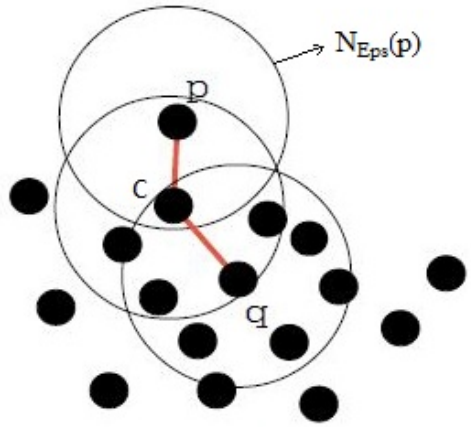

Figure 2.5: For $MinPts$ = 6, point p is directly density reachable from c and density reachable from q.[6]

Simple density-reachable is an extension directly density-reachable as a transitive but not symmetric.

Another case that needs to be mentioned is that of two border items of the same cluster. While they might not be density reachable from each other, since the core point condition does not hold for both of them, there must be a core point that holds both of those items as density reachable. To cover this relation, it is necessary to define the notion of density-connectivity.

Definition 4 (Density-Connected) *A point p is density connected to a point q wrt. Eps, MinPts if there is a point o such that both, p and q are density reachable from o wrt. Eps, MinPts .*

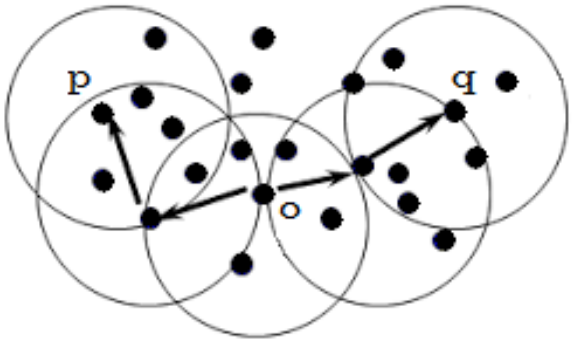

Figure 2.6: Point p is density connected to point q through o

Using the above definitions makes it easier to define a cluster and noise as well.

Definition 5 (Cluster) *Let D be a data set of points. A cluster C wrt. Eps and MinPts is a non-empty subset of D satisfying the following conditions:*

i $\forall p, q$: *if* $p \in C$ *and q is density reachable from p wrt. Eps and MinPts, then* $q \in C$. *(Maximality)*

ii $\forall p, q \in C$: *p is density connected to q wrt. Eps, MinPts. (Connectivity)*

Definition 6 (Noise) *Let* $C_1, \ldots, C_k$ *be the clusters of the database D wrt. Parameters* Eps_i *and* $MinPts_i$, *i=1, ..., k. Then we define the noise as the set of points in the database D not belonging to any cluster* C_i, *i.e. noise* = $\{p \in D | \forall i : p \notin C_i\}$

Algorithm 5 DBSCAN

Input: set of points $D, Eps, MinPts$

```
1: ClusterId = label of the first cluster;
2: for each point p in set D do
3:   if p.ClusterId = UNCLASSIFIED then
4:     if expandCluster(D, p, ClusterId, Eps, MinPts) then
5:       ClusterId = NextId(ClusterId);
6:     end if
7:   end if
8: end for
```

Given these definitions, it is easier to focus on the DBSCAN algorithm itself, which is designed to discover clusters and noise (according to 5 and 6) in a spatial data set. The input parameters of the algorithm are the data set D, as a set of points, the maximum value of radius that encloses a neighborhood, denoted as Eps and known as epsilon value, and the minimum number of points within an Eps-neighborhood, noted as MinPts. The values of Eps and MinPts, which are global, i.e. same for all clusters during the execution of the algorithm can play a significant role to the resulting clusters. Therefore, an effective way to estimate the most appropriate values for those parameters according to each data set is mentioned in the end of this sub-section, as well as a variation of DBSCAN.

Considering that each item of the data set can either belong to a cluster or be noise, there is an associated ClusterId field for each item, representing the cluster the item belongs to, if it does to any. To begin with, the ClusterId of all data items is initialized with the corresponding value of unclassified. Also, the algorithm generates a label or id value for the first cluster that is to be discovered. The following steps are executed through a loop for each item of the data set: if the given item is found to be a core item, its ClusterId, as well as the ClusterId of all items that belong to the newly formed cluster takes the value of the current cluster label. This very value is later updated for future clusters to be found. If however the item has not a sufficient number of neighbors to be a core

item it is temporarily labeled as noise. According the sequence of steps so far, it is clear that some items that are density-reachable from a previously analyzed core item may acquire a ClusterId value before their turn to be analyzed. Those pre-classified points are omitted from further analysis. Furthermore, not all items that are initially labeled as "unclassified" remain so, as they are most likely to be distributed to neighbor clusters as border items.

Algorithm 6 expandCluster

Input: D, point p, cluster label $ClId$, Eps, $MinPts$

```
1: seeds = Neighborhood(D, p, Eps);
2: if |seeds|<MinPts then
3:    p.ClusterId = NOISE;
4:    return false;
5: else
6:    for each point q in seeds do
7:       q.ClusterId = ClId;
8:    end for
9:    delete p from seeds
10:   while |seeds|>0 do
11:      currentPoint = first point in seeds;
12:      currentSeeds = Neighborhood(D, currentPoint, Eps);
13:      if |currentSeeds| ≥ MinPts then
14:         for each point q in currentSeeds do
15:            if q.ClusterId = UNCLASSIFIED then
16:               q.ClusterId = ClId;
17:               append q to seeds
18:            else if q.ClusterId = NOISE then
19:               q.ClusterId = ClId;
20:            end if
21:         end for
22:      end if
23:      delete currentPoint from seeds
24:   end while
25:   return true;
26: end if
```

The expandCluster function, as displayed in Algorithm 6, is basically the core of the algorithm. In detail, the $N_{Eps}(p)$, where p is the item under analysis, is calculated. If the total number of neighbors is not greater that the MinPts value, p is temporarily labeled as noise and the function returns false. That is the end of the analysis of p item and the

loop proceeds to its next run. However, if p has a sufficient amount of neighbors (thus is a core item) the function proceeds in labeling it and all of its neighbors with the current cluster label. The process does not stop there, as the expanding includes examining the Eps-neighborhood of each found neighbor and treating them accordingly.

Given that we are using global values for Eps and MinPts, two clusters of different density in close proximity may be merged into one by the algorithm. The distance between two sets of points S_1 and S_2, denoted as dist(S_1, S_2), is actually the minimum distance between their respective points p and q (min dist(p,q) $|p \in S_1, q \in S_2$). Two such sets of points having at least the density of the thinnest cluster will be separated from each other only if the distance between the two sets is larger than Eps. Consequently, a recursive call of DBSCAN may be necessary for the detected clusters with a higher value for MinPts [23]. However, the recursive application of DBSCAN prompts a clear and practical, basic algorithm, so the recursive clustering of the points of a cluster is not considered a disadvantage. Furthermore, it is only necessary under conditions that can be easily detected.

In case two or more clusters lie too close to each other, it is not rare that a border item might belong to more than one of them. In this circumstance, the item is assigned to the cluster discovered first. Other than that however, the results of DBSCAN do not depend on the order in which the items of the data set are examined.

2.3.2 Determination of Eps and MinPts

As for determining the parameters Eps and MinPts, there is a simple but effective heurestic based on the following observation. Assuming we have a distance d between a point p and its k-th nearest neighbor, the d-neighborhood of p contains exactly k+1 points for nearly all points p. The only case in which the d-neighborhood is populated with more than k+1 points is that in which several points of the d-neighborhood have exactly the same distance d from p, which is extremely rare. Also, changing k for a point does not equal with remarkable changes of d. That would only happen if the k-th nearest neighbors of p for k = 1, 2, 3... are located approximately on a straight line, which is in general very unlikely for point in a cluster.

For a given k we can define a $k - dist$ function which will be used for mapping each point to the distance from its k-th nearest neighbor. Sorting the points of the data set in descending order of their k-dist values, the graph of this function, known as the $k-distgraph$ can reveal some hints regarding the density distribution in the data set.If we choose an arbitry point p, set the parameter Eps to k-dist(p) and the parameter MinPts to k, all points with an equal or smaller k-dist value will be core points [23]. The desired parameter values are those of the $thresholdpoint$, the point with the maximal k-dist value in the thinnest cluster of the data set. The threshold point is actually the first point of

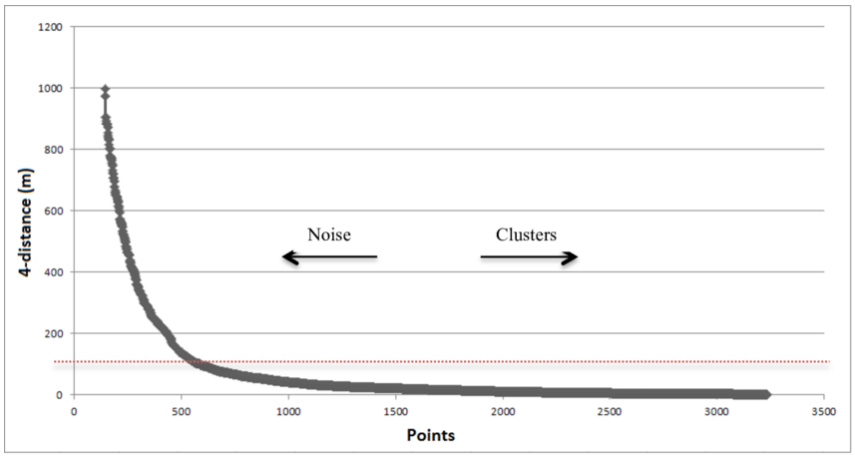

Figure 2.7: Sorted 4-dist graph for sample database.[7]

the first valley in the sorted k-dist graph. Higher k-dist values are considered noise, while all the other points, placed in the right of the threshold, are already assigned to some cluster.

2.3.3 OPTICS: Ordering Points To Identify the Clustering Structure

The determination of parameters for DBSCAN is not the only weak point of the algorithm. Even after detecting the most appropriate values for a data set -using a $k-dist$ graph or by experimenting- the found parameters will be used globally by the algorithm, meaning that they establish a certain level of density according to which the clusters will be detected, and this level applies for the entire data set. However, real-data sets can rarely obey to global density parameters, since they usually consist of clusters of different densities. As indicated in 2.3.1, DBSCAN might merge two clusters of different density in close proximity, a problem which can be resolved with a recursive call of the algorithm, using a higher MinPts value. Yet, using global parameters inevitably restricts the outcome and the local densities that may be needed to reveal meaningful clusters in different data space areas are again ignored. For example, in the data set depicted in 2.8, detecting clusters A, B, C1, C2, and C3 simultaneously, while using a common global density parameter, is not possible. Global density parameters could detect either clusters A, B, and C, or clusters C1, C2, and C3. It is important to note that in the second case, objects of A and B would be considered noise, as they would not form sufficient clusters.

An alternative for discovering such clustering structures would be to use a hierarchical clustering algorithm instead. However, dendrograms of large data sets are usually harder to understand or extract knowledge from. Another alternative would be, as already mentioned, to use a density-based partitioning algorithm with different parameteric settings, but the number of possibly suitable values is infinite and even if we cover an adequate range of those values by experimenting with them -which is time consuming and requires extra memory resources for the storage of the different cluster memberships for each point-

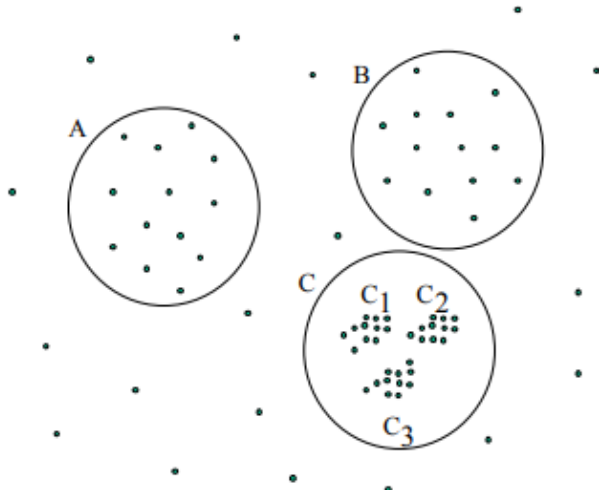

Figure 2.8: Clusters wrt. different density parameters.[8]

we may still miss useful clustering levels and the results will still be hard to analyze.

This is the problem that OPTICS, or Ordering Points To Identify the Clustering Structure, aims to overcome. This algorithm, which is a variation of DBSCAN, produces a special *order* of the database, which relies on its density-based clustering structure and contains useful information about every clustering level of the data set -up to a "generating distance" Epsilon- providing an output that is easy to analyze. To be precise, instead of explicitly clustering a data set, OPTICS creates an augmented cluster-ordering of it, that "contains information which is equivalent to the density-based clusterings corresponding to a broad range of parameter settings"[8]. As a result, it can be used to efficiently provide typical clustering information, such as cluster prototype points or discovery of arbitrary shaped clusters, as well as a way to gain additional insights into the intrinsic clustering structure, which would otherwise be falsely ignored.

The notion of density-based cluster-ordering relies on the following observation: for a constant MinPts-value, density-based clusters of higher density -achieved by a lower value of Epsilon, a tighter neighborhood radius- like clusters C1, C2 and C3 in the depicted example of 2.8, are nested in clusters or density-connected sets of lower density, -achieved by higher value of Epsilon, allowing for wider neighborhoods- like cluster C. In order to extend DBSCAN so that those several distance parameters ε_i -which are smaller than a generating distance Epsilon (i.e. $0 \leq \varepsilon_i \leq Epsilon$)- are processed at the same time -leading to the simultaneous construction of clusters with respect to their different densities- OPTICS has to follow a certain order in which the objects will be processed, so that the nested clusters are not ignored. In detail, objects that are density-reachable with respect to the lowest ε value must be processed first, so that clusters of higher density are detected first. An important difference is that OPTICS does not assign cluster memberships through this procedure. The algorithm focuses on the order in which the objects are processed and this information can later be used by an extended DBSCAN algorithm

to assign cluster memberships. For each object, this information can be described by two additional values, $core-distance$ and $reachability-distance$.

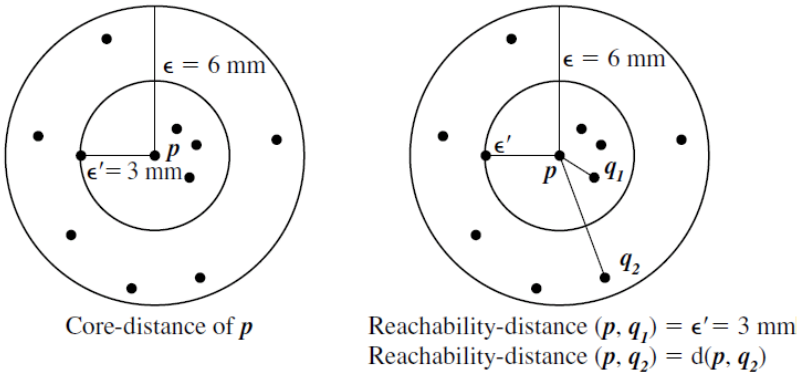

Figure 2.9: Illustration of core-distance and reachability-distances, MinPts=5. [9]

Definition 7 (Core distance of an object p) *Let p be an object from a database D, let ε be a distance value, let $N_\varepsilon(p)$ be the ε-neighborhood of p, let $MinPts$ be a natural number and let $MinPts-distance(p)$ be the distance from p to its MinPts' neighbor. Then, the $core-distance$ of p is defined as*

$$core-distance_{\varepsilon,MinPts}(p) = \begin{cases} UNDEFINED, & if\ |N_\varepsilon(p)| < MinPts \\ MinPts\text{-}distance(p), & otherwise \end{cases}$$

In other words, the core distance of a point p is the minimum ε' value for which p is a core point, given a constant value of MinPts.

Definition 8 (Reachability distance object p w.r.t. object o) *Let p and o be objects from a database D, let $N_Eps(o)$ be the e-neighborhood of o, let $MinPts$ be a natural number. Then, the $reachability-distance$ of p with respect to o is defined as*

$$reachability\text{–}distance_{e,MinPts}(p,o) = \begin{cases} UNDEFINED, & if\ |N_{Eps}(o)| < MinPts \\ max(core\text{-}distance(o),\ distance(o,p)), & otherwise \end{cases}$$

Intuitevely, the reachability-distance of p with respect to another point o is the minimum distance from o, if that is core point and this minimum distance cannot be smaller than the core distance of o.

Pseudocode of Algorithm 7 illustrates the main iteration of OPTICS. The aforementioned values of core distance and a suitable reachability distance for each point of the data set are calculated and stored in an $OrderedFile$. The detailed procedure of determining the order information of each point is placed in the expandClusterOder() method, which is

called for each unprocessed point of the data set in the main OPTICS loop.

Algorithm 7 OPTICS

Input: set of objects $D, Eps, MinPts$, $OrderedFile$

```
1: OrderedFile.open();
2: for each point p in set D do
3:    if !p.processed then
4:       expandClusterOrder(D, p, Eps, MinPts, OrderedFile);
5:    end if
6: end for
7: OrderedFile.close();
```

Algorithm 8 expandClusterOrder

Input: D, point p, Eps, $MinPts, OrderedFile$

```
1: seeds = Neighborhood(p, Eps);
2: p.processed = true;
3: p.reachabilityDistance = UNDEFINED;
4: p.setCoreDistance(seeds, Eps, MinPts);
5: OrderedFile.write(p);
6: if p.coreDistance != UNDEFINED then
7:    OrderSeeds.update(seeds, p);
8:    while !OrderSeeds.empty() do
9:       currentObject = OrderSeeds.next();
10:      currentSeeds = Neighborhood(currentObject, Eps);
11:      currentObject.processed = true;
12:      currentObject.setCoreDistance(currentSeeds, Eps, MinPts);
13:      OrderedFile.write(currentObject);
14:      if currentObject.coreDistance != UNDEFINED then
15:         OrderSeeds.update(currentSeeds, currentObject);
16:      end if
17:   end while
18: end if
```

The expandClusterOrder(), as illustrated in algorithm 8, initially retrieves the neighbors of the under process point p within an Eps radius. The reachability-distance of p is initialized to 'undefined' while its core-distance is computed. Point p is then written in the $OrderedFile$. If it is not a core point for the generating distance Eps -in which case, its core distance is undefined- the flow of execution returns back to the main loop and OPTICS proceeds to the next unprocessed point. However, if p has a core distance for a radius that is lower than Eps, its directly density-reachable points with respect to

Eps and *MinPts* are iteratively collected into an *OrderSeeds* list for further expansion. *OrderSeeds* is sorted by ascending reachability-distance of each point to its closest core point. While the list is still populated with points, each point, denoted as *currentObject*, is processed as described: the neighborhood of *currentObject* with respect to *Eps* is retrieved and the core-distance of *currentObject* is computed with respect to *Eps* and *MinPts*. *currentObject* is later written in the *OrderedFile* and if it is a core point itself, the initial *OrderSeeds* list is updated with its retrieved neighborhood, since its neighboors must be processed in the same way as they render further candidates for expansion.

Algorithm 9 describes the OrderSeeds.update() method. The reachability-distance of each neighbor is determined with respect to the *currentObject*, denoted as *CenterObject* in algorithm 9. Neighbors with undefined reachability distance are points that have not been in the *OrderSeeds* list yet, therefore they acquire a reachability-distance and are added in the list. Neighbors with pre-existing value of reachability distance, that have already been in the *OrderSeeds* list, have to be checked. If a neighbor's acquired reachability-distance is higher than the reachability-distance with respect to the *CenterObject*, the field is updated with the new, lower value and the neighbor is moved further to the top of the *OrderSeeds* list, which works like a priority queue.

Algorithm 9 *OrderSeeds*.update

Input: *Neighbors*, center object *p*

```
1:  cDistance = p.coreDistance;
2:  for each point q in Neighbors do
3:    if !q.Processed then
4:      newRDistance = max(cDistance, p.dist(q));
5:      if q.reachabilityDistance == UNDEFINED then
6:        q.reachabilityDistance = newRDistance;
7:        insert(q, newRDistance);
8:      else
9:        if newRDistance < q.reachabilityDistance then
10:         q.reachabilityDistance = newRDistance;
11:         decrease(q, newRDistance);
12:       end if
13:     end if
14:   end if
15: end for
```

Finally, extractDBSCAN-Clustering(), described in 10, allows us to extract from the newly generated clustering-ordering any density-based clustering wrt. *MinPts* and a clustering-distance $Eps' \leq Eps$ -where Eps was used by OPTICS- and to assign cluster-memberships depending on the reachability-distance and the core-distance of the points. Each point's *p*

reachability distance is checked on whether it is greater than the given Eps'. If it is, that means that p is not density reachable wrt. Eps' and $MinPts$ from any of the preceding points in the cluster-ordering, for if it was, it would be given a reachability distance of at most Eps'. Therefore, the core-distance of p is checked and p is the first point of a new cluster if it is a core point wrt. Eps' and $MinPts$, otherwise, it is assigned to NOISE. If on the other hand, p is density reachable wrt. Eps' and $MinPts$ in the first place, it is assigned to the current cluster.

Algorithm 10 extractDBSCAN-Clustering

Input: $ClusterOrderedObjects$, Eps', $MinPts$

```
1:  ClusterId = NOISE;
2:  for each point q in ClusterOrderedObjects do
3:    if q.reachabilityDistance > Eps' then
4:      if q.coreDistance ≤ Eps' then
5:        ClusterId = nextId(ClusterId);
6:        q.clusterId = ClusterId;
7:      else
8:        q.clusterId = NOISE;
9:      end if
10:   else
11:     q.clusterId = ClusterId;
12:   end if
13: end for
```

It has been experimentally proved that due to the similarities between OPTICS and DBSCAN, their run-times are nearly the same, with OPTICS taking almost 1.6 times the run-time of DBSCAN. Although the resulting clusters of extractDBSCAN-Clustering() are not much different from those of the traditional DBSCAN, it must be noted that OPTICS is mostly intended to provide a clustering-ordering with respect to multiple values of neighborhood radius for a constant MinPts. Information associated with the data set structure can however be extremely useful in case of determining the most fitting pair or pairs of parameters as well as discovering otherwise ignored clusters -whose borders might be useful for our cause. This information can be represented graphically by a reachability plot, especially for medium sized data sets. For larger data sets, there is a separate visualization technique.

Below there is an example of a reachability plot, side to side with the two dimensional representation of the sample data set it represents. The y-axis of the plot represents the different reachability distances found while the x-axis refers to the particular order in which the points were processed. Each valley refers to a different cluster.

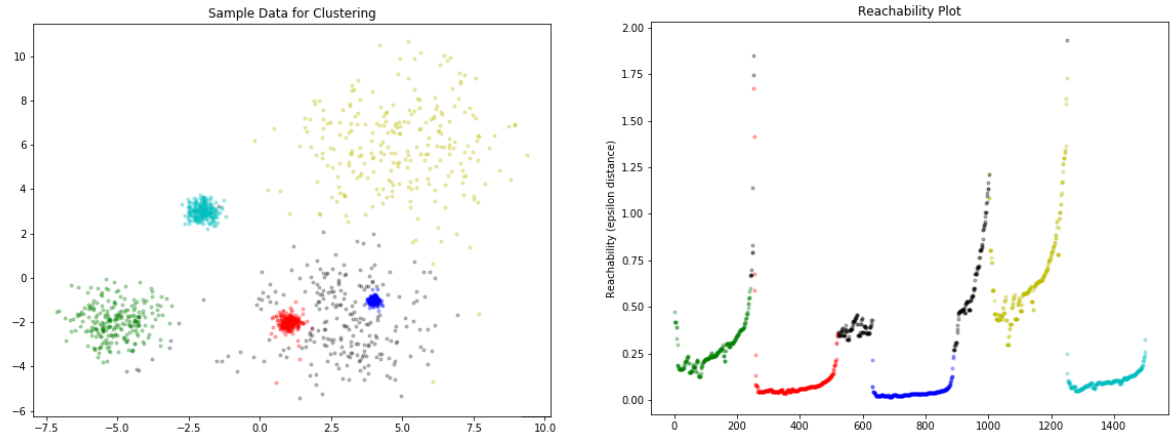

Figure 2.10: Example of a data set with clusters of different densities and its reachability plot.[9]

3 Prototype Selection via DBSCAN Clustering

Given the fact that DBSCAN "labels" through conditions each item of a data set while forming and expanding clusters, we can take advantage of its functionality to determine which of those items are actually meaningful for a classifier, depending on their position. As indicated in previous sections, training items that lie closer to class decision boundaries are far more helpful during classification than training items within those boundaries and therefore should be retained or represented in case the data set undergo any sort of data reduction technique. We see that prototype abstraction algorithms provide generated and not pre-existing prototypes to construct a condensing set, while editing algorithms focus more on eliminating noise.

In this study, we attempt to use DBSCAN as a condensing algorithm instead. The idea is that the cluster border items -or/and even outliers- of a data set are possibly as good as class border items for a classifier, while core items are less pivotal. As displayed in Figure 3.1, core items, which are characterized by the amount of their neighbors, constitute the areas of a DBSCAN cluster of higher density. As the density declines, we reach to the border items of the cluster, that -if shown separately - are enough to define the shape of the cluster. Considering that clusters are meaningful groups of items that have similar features and attribute values, it is not uncommon that they belong in the same class. Assuming that we defined the appropriate global parameters Eps and $MinPts$ and that each cluster reflects a class, retaining those border items or any outlier that lies in between them is equivalent to retaining the items of class decision boundaries areas, the seemingly most useful ones during classification. Consequently, by identifying and "extracting" those items into a data set, we end up having an adequate condensing set.

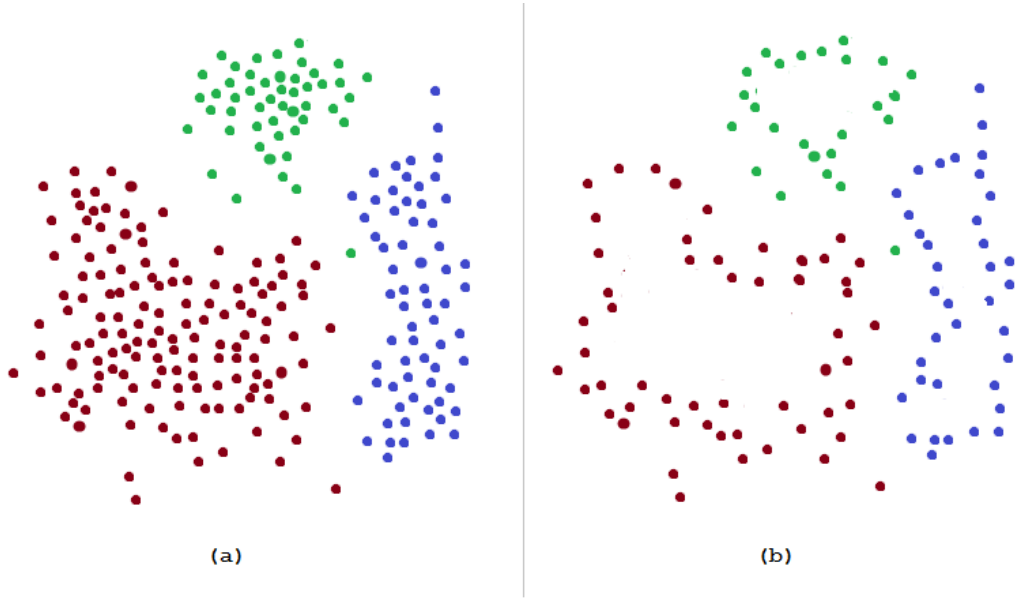

Figure 3.1: (a) the initial data set in clusters and (b) the condensing set that can be extracted.

Since the original algorithm is normally used to form clusters, in addition to the properties of each item, we also have a ClusterId field that represents the cluster in which the item is assigned to in the end, if it does to any. Although DBSCAN assigns the value of "NOISE" to all items that do not belong to any cluster, some of them might actually be just outliers and not noise. Assuming that the given data set is either noise free or has undergo editing prior to the execution of DBSCAN, we can safely claim that items of ClusterId = "NOISE" are just outliers. However, the original algorithm does not include some sort of discrimination field between border and core items. Their difference, which is the sufficiency of number of neighbors, is calculated and tested during the execution of the expandCluster() method, so that the algorithm can spread the value of the current cluster label in as many items as the neighborhood condition allows. For our Java implementation, we add two more boolean properties in each Item object, *corePoint* and *borderPoint*, so that after determining the type of the item, we can store this information and use it while constructing the condensing set.

Algorithm 11 DBSCANcores

Input: set of points $D, Eps, MinPts$

```
1: for each point p in set D do
2:    tempNeighborhood = Neighborhood(D, q, Eps);
3:    if |tempNeighborhood| ≥ MinPts then
4:       q.setCorePoint(true);
5:    end if
6: end for
```

Core points are easily detectable considering that they only have to fulfill one condition and that is that the total number of neighbors of the point in question is greater or equal with the predefined global parameter MinPts. For this reason, as shown in the Algorithm 11, we begin by scanning the entire data set and checking the neighborhood of every point p. If the neighborhood is found sufficient, p is labelled as a core point, otherwise it is ignored.

At this point, we could say that all the necessary items for the condensing set are found, since they are the ones that do not meet the core point criterion. Given that we store all of our items in a data structure -an ArrayList<Item>for this implementation- if the ones that are found to be cores are excluded, what is left is border items and outliers. However, for research purposes, these two cases are worth to be studied separately. Although the significance of the border items is logical and visually comprehensible, outliers might also make a positive difference during classification, especially in data areas where the boundaries are not clear. At the same time, depending on the data set and the combination of the parametric values, outliers might be great in number and despite their beneficial role in improving the accuracy of a classifier, they lower the reduction rate. In such case, retaining only the border items might be more fitting, but requires an extra method that

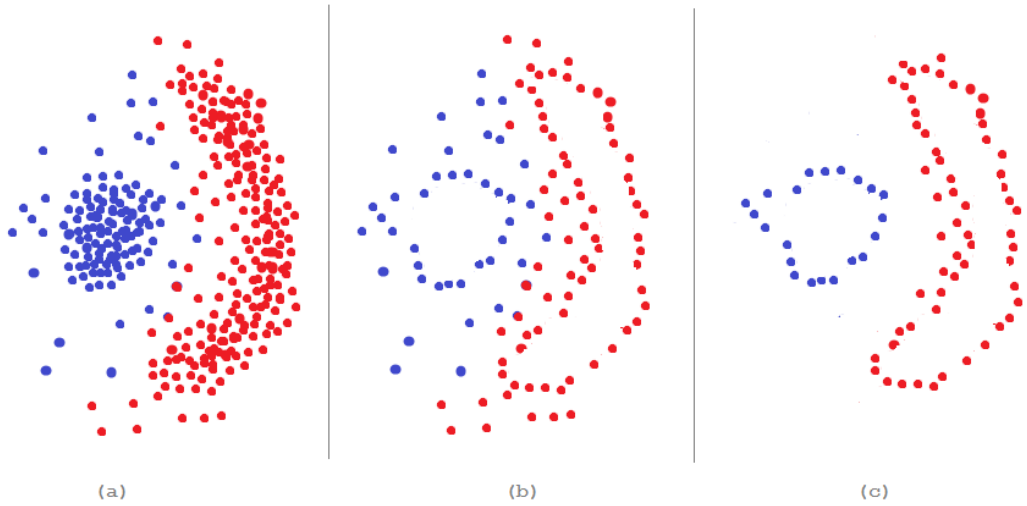

Figure 3.2: (a) the initial data set in clusters, (b) the condensing set of border items and outliers and (c) the condensing set of border items only.

can identify them and store them in a separate data structure. For this task, method DBSCANborders, as described in Algorithm 12 is proposed.

Algorithm 12 DBSCANborders

Input: set of points D, Eps
Output: set of border items B

```
1:  B ← ∅
2:  for each point p in set D do
3:    if !p.getCorePoint() then
4:      tempNeighborhood = Neighborhood(D, q, Eps);
5:      for each point q in set tempNeighborhood do
6:        if q.getCorePoint() then
7:          p.setBorderPoint(true);
8:          add(B, p);
9:          break;
10:       end if
11:     end for
12:   end if
13: end for
14: return B
```

DBSCANborders method can give valid results only if it is executed after DBSCANcores, since it is essential for the core items to be identified first. After that, each point p of the data set is checked. If p is a core item, it is ignored, while if it is not, its neighborhood is detected and examined. If at least one neighbor is a core item, then p must be part of a cluster and since it is not a core item itself, it is inevitably labelled as a border item. On the other hand, if no core item is found in the neighborhood, p must be beyond the borders of any cluster and therefore is ignored as an outlier.

Despite any of the two cases, those being (1) border items-only and (2) plus-outliers, detecting and forming the clusters in which the items might belong is not only unnecessary for our cause, but also useless for the type of data we have. As previously stated, clustering is a method of unsupervised learning, meaning that the set of input data consists of items that do not belong in any particular class or set of classes. Clustering algorithms are meant to obtain information solely depending on the features and attributes of the items, their similarities, possible relations, etc. without previously knowing what the output will be. In our case, items of the training set should be efficiently reduced so that the produced condensing set can be used by a classifier, but as known, classification requires training data that is divided in predefined classes. Since the data items our proposed algorithm will be used for are assigned to classes, there is no need or use to form clusters of them. It is clear that we are only taking advantage of the notions and the "types" of the items depending on their position in the data space, as it is the main criterion for them to be retained.

For this very reason, it might look logical and more effective to try this process of identifying and retaining important items for each class individually instead of running the aforementioned methods for the entire data set. Since we already have the classes of the training items, we could use them to divide the initial training set into class subsets and then identify cores, borders and outliers for each class. After all, class borders might differ from cluster borders, especially in cases where two or more different classes of the same density are quite similar, in a way that their corresponding items lie close to each other and form a cluster with broader and less accurate borders. Indeed, this scenario was also taken into account. Part of our implementation in Java, that can be found in Appendix A, includes a method that reads the entire data set and collects the items of each of the available classes -regardless how many they are- in separate subset structures of type ArrayList¡Item¿. The proposed algorithms were later executed for each of those subsets, extracting borders only or borders *and* outliers for each class.

The resulting condensing sets were studied separately with less success. In some cases, especially with higher number of classes -and consequently borders, with or without outliers- the reduction rate would be quite low to be considered and in attempts to increase it by changing the parameters Eps and MinPts, the accuracy would be negatively affected. For this reason and also due to the complexity of the method, the experimental results of this scenario were not included in this dissertation. A possible reason why class borders have been less efficient than cluster borders might be the fact that classes which lie in close proximity may consist of border items that overlap and are more misleading than helpful. In any case, this scenario was dismissed and we will focus on the cluster borders, as found by the algorithm when executed for the entire data set.

4 Experimental Study

4.1 Experimental Setup

For the evaluation of the proposed algorithms, ten data sets distributed by the KEEL data set repository that are also available at the UCI machine learning repository, were used. They are summarized in the table of subsection 4.1.1. For comparison purposes, we used two condensing algorithms, namely, CNN-rule, and IB2, and a prototype abstraction approach, namely, RSP3. We selected these methods because: (i) CNN-rule and RSP3 are used quite often in many papers for comparison, which makes them popular reference algorithms in the research community and bibliography, (ii) IB2 is also noted as a fast data reduction technique that dynamically builds its condensing set and (iii) both condensing and prototype abstraction algorithms share the same motivation with the proposed algorithms, therefore it is only natural for the later to be compared with well-known algorithms of both categories. In addition to using condensing sets, we also measured the performance of the conventional 1-NN classifier.

For each data set and algorithm, we report three average measurements obtained via five-fold cross-validation: (i) Accuracy, (ii) Reduction Rate, and, (iii) Preprocessing Cost. We report the classification accuracy of kNN for k = 1 (1-NN). For all data sets, we used the five already constructed pairs of training/testing sets hosted by the KEEL repository. Algorithms implementations were written in Java programming language and the Euclidean distance was adopted as the distance metric.[1]All the experiments were done on a multi-processor computer with Debian operating system.

4.1.1 Datasets

Dataset	Size	Attributes	Classes
Letter Recognition (LR)	20000	16	26
Pen-Digits (PD)	10992	16	10
Landsat Satellite (LS)	6435	36	6
Texture (TXR)	5500	40	11
Phoneme (PH)	5404	5	2
Balance (BL)	625	4	3
Ecoli (ECL)	336	7	8
Yeast (YS)	1484	8	10
Twonorm (TN)	7400	20	2
MONK 2 (MN2)	432	6	2

Table 4.1: Dataset description.

[1]Since we only need the distance for comparinson purposes between the items, to minimize the amount of calculations, we only use the mathematical expression under the square root described in 1.1

The ten data sets were used without data normalization. The LS, TXR and ECL data sets are distributed sorted on the class label and this affects the methods that depend on the order of data. Consequently, we randomized the order of the data items for these data sets. No other data transformation was performed and all experiments were conducted without previous knowledge about the data sets such as data distribution, level of noise, etc.

i **Letter Recognition (LR)** - The purpose of this data set is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters of the English alphabet. The available character images were based on 20 different fonts. Each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique instances. Finally each instance was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were later scaled to fit into a range of integer values from 0 through 15. Such attributes include the horizontal and vertical position of the displayed letter, width and height, total number of pixels, mean number of pixels per axis, count of edges from left to right, etc.

ii **Pen-Digits (PD)** - This data set was created by collecting 250 samples of hand-written digits from 44 different writers. From those, the samples written by thirty writers are used for training, cross-validation and writer dependent testing, while the samples written by the other fourteen are used for writer independent testing. In order to collect the data, a WACOM PL-100V pressure sensitive tablet with an integrated LCD display and a cordless stylus were used. The input and display areas are located in the same place. The writers were asked to write 250 digits in random order inside boxes of 500 by 500 tablet pixel resolution and they were monitored only during the first entry screens. Each screen contains five boxes with the digits to be written displayed above. The writers were instructed to write only inside the fore mentioned boxes. In case they make a mistake or they are unhappy with their writing, they are instructed to clear the content of a box by using an on-screen button. The first ten digits are ignored because most writers are not familiar with this type of input devices, but subjects are not aware of this. The device is attached to the serial port of an Intel 486 based PC, where the handwriting samples are gathered. The sent information includes the tablet coordinates and pressure level values of the pen at fixed time intervals (sampling rate) of 100 miliseconds, but only the cordinate information is used. Normalization is used for the representation to be invariant to translations and scale distortions, so even though the raw data that were captured from the tablet consist of integer values between 0 and 500 (tablet input box resolution), the new coordinates are such that the coordinate which has

the maximum range varies between 0 and 100.

iii **Landsat Satellite (LS)** - This data set originates from a database that was generated from Landsat Multi-Spectral Scanner image data. The collected data by these or other forms of remotely sensed imagery is usually in binary form and is distributed on magnetic tape(s). One frame of Landsat MSS imagery consists of four digital images of the same scene in different spectral bands. Two of these are in the visible region (corresponding approximately to green and red regions of the visible spectrum) and two are in the (near) infra-red. Each pixel is a 8-bit binary word, with 0 corresponding to black and 255 to white. The spatial resolution of a pixel is about 80m x 80m and each image contains 2340 x 3380 such pixels. So the original database consists of the multi-spectral values of pixels in 3x3 neighbourhoods in a satellite image and a classification label associated with the central pixel in each neighbourhood. In the sample data set which was later generated by the initial database, the class of the central pixel is coded as a number. The sample data set is taking only a small section from the original data (82 rows and 100 columns, basically 82 x 100 pixels of a scene) and the binary values are converted to their present ASCII form. Consequently, each line of data corresponds to a 3x3 square neighbourhood and includes 36 pixel values (9 pixels of the 3x3 square neighbourhood * their corresponding values in all four spectral bands (converted to ASCII)), completely contained within the 82x100 sub-area. Additionally, each line contains a number indicating the classification label of the central pixel, according to the following table, though it should be noted that there are no examples of class 6 in this data set.

Class Label	Class
1	red soil
2	cotton crop
3	grey soil
4	damp grey soil
5	soil with vegetation stubble
6	mixture class (all types present)
7	very damp grey soil

Table 4.2: Landsat Satellite Classes.

Certain lines of data have been removed so that the original image from this data set cannot be reconstructed. In each line of data the four spectral values for the top-left pixel are given first followed by the four spectral values for the top-middle pixel and then those for the top-right pixel, and so on with the pixels read out in sequence left-to-right and top-to-bottom.

iv **Texture (TXR)** - This data set has been created in order to study the textures

discrimination with high order statistics. It consists of instances that belong to 11 different textures, as diplayed in the class table of the data set below, and each instance is characterized by 40 attributes. The attributes are built by the estimation of fourth order modified moments (MM4) in four orientations: 0, 45, 90 and 135 degrees. MM4 is a statistical method based on the extraction of fourth order moments for the characterization of natural micro-textures and measures the deviation from first-order Gauss-Markov process, for each texture. The features were estimated in the four aforementioned directions to take into account the possible orientations of the textures. Only correlation between the current pixel, the first neighbourhood and the second neighbourhood are taken into account.This small neighbourhood is adapted to the fine grain property of the textures.

Class Label	Class
2	Grass lawn
3	Pressed calf leather
4	Handmade paper
6	Raffia looped to a high pile
7	Cotton canvas
8	Pigskin
9	Beach sand (1)
10	Beach sand (2)
12	Oriental straw cloth (1)
13	Oriental straw cloth (2)
14	Oriental grass fiber cloth

Table 4.3: Texture Classes.

v **Phoneme (PH)** - The majority of the existing speech recognition systems belong to global systems (typically Hidden Markov Models and Time Delay Neural Networks)that recognise signals without utilizing the speech specificities. On the contrary, analytical systems take into account the articulatory process leading to the different phonemes of a given language, in order to deduce the presence of each of the phonetic features from the acoustic observation. The main difficulty of such systems is to obtain the reliability of the acoustical parameters, as acoustical measurements must contain all the information relative to the concerned phonetic feature, be speaker and context independent and be more or less robust to noise. The aim of the present database is to distinguish between nasal (class 0) and oral vowels (class 1). The phonemes are transcribed as follows: *sh* as in she, *dcl* as in dark, *iy* as the vowel in she, *aa* as the vowel in dark, and *ao* as the first vowel in water. This database contains vowels coming from 1809 isolated syllables (for example: pa, ta, pan,...). Five different attributes were chosen to characterize each

vowel: they are the amplitudes of the five first harmonics AHi, normalised by the total energy Ene (integrated on all the frequencies): AHi/Ene. Each harmonic is signed: positive when it corresponds to a local maximum of the spectrum and negative otherwise. Three observation moments have been kept for each vowel to obtain 5427 different instances.From these, 23 instances for which the amplitude of the 5 first harmonics was zero were removed, leading to the 5404 instances of the present database. The patterns are presented in a random order.

vi **Balance (BL)** - This data set was generated to model psychological experimental results. The class of each instance in this case is the indication of having the balance scale tip to the right, tip to the left, or be balanced. The four attributes of each instance are the left weight, the left distance, the right weight, and the right distance. The correct way of identifying the class for each instance is by comparing the values of $left-distance * left-weight$ and $right-distance * right-weight$. Depending on which one is greater, the instance can be left or right respectively. In case they are equal, the instance is balanced.

vii **Ecoli (ECL)** - The objective of this data set is to predict the localization site of proteins by employing some measures about the cell (cytoplasm, inner membrane, perisplasm, outer membrane, outer membrane lipoprotein, inner membrane lipoprotein inner membrane, cleavable signal sequence). The original data set from which ECL derived included as the first attribute of each instance, the sequence name which indicates the Accession number for the SWISS-PROT database. To asses the data to classification process, this attribute has been removed from both KEEL's and UCI's versions of the data set, thus each instance now consists of seven attributes and can belong to any of the eight classes below. Though diplayed with their string labels, each one of these classes is represented as an integer value between zero and seven in the data set.

Class Label	Class
cp	cytoplasm
im	inner membrane without signal sequence
pp	perisplasm
imU	inner membrane, uncleavable signal sequence
om	outer membrane
omL	outer membrane lipoprotein
imL	inner membrane lipoprotein
imS	inner membrane, cleavable signal sequence

Table 4.4: Ecoli Classes.

viii **Yeast (YS)** - This database contains information about a set of Yeast cells. The

task is to determine the localization site of each cell among 10 possible alternatives each of whom represent a class, determined by a set of 8 attributes. Like ECL, Yeast's original data set also included the sequence name as the first attribute of each instance and this very attribute was excluded from both KEEL's and UCI's versions of the data set, to asses the data to classification process. And again, the diplayed string labels for the data set classes are represented as integer values between zero and nine.

Class Label	Class
CYT	cytosolic or cytoskeletal
NUC	nuclear
MIT	mitochondrial
ME3	membrane protein, no N-terminal signal
ME2	membrane protein, uncleaved signal
ME1	membrane protein, cleaved signal
EXC	extracellular
VAC	vacuolar
POX	peroxisomal
ERL	endoplasmic reticulum lumen

Table 4.5: Yeaststst Classes.

ix **Twonorm (TN)** - This is an implementation of Leo Breiman's twonorm example . It is a 20 dimensional, 2 class classification example. Each class is drawn from a multivariate normal distribution with unit variance. Class 1 has mean (a,a,..a) while Class 2 has mean (-a,-a,..-a). Where a = 2/sqrt(20). Breiman reports the theoretical expected misclassification rate as 2.3%. He used 300 training examples with CART and found an error of 22.1%.

x **MONK 2 (MN2)** - The MONK's problems were the basis of a first international comparison of learning algorithms. One significant characteristic of this comparison is that it was performed by a collection of researchers, each of whom was an advocate of the technique they tested (often they were the creators of the various methods). In this sense, the results are less biased than in comparisons performed by a single person advocating a specific learning method, and more accurately reflect the generalization behavior of the learning techniques as applied by knowledgeable users. The MONK's problems are actually a collection of three binary artificial classification problems (MONK-1, MONK-2 and MONK-3) over a six-attribute discrete domain. Each problem involves learning a binary function defined over this domain, from a

[1]Breiman L. *Bias, variance and arcing classifiers.* Tec. Report 460, Statistics department. University of California. April 1996.

sample of training examples that belong to class 0 or class 1. The data set in use is the second problem of the collection, with no random noise added.

4.1.2 Parameters

As mentioned in 2.3.2, k-dist is a simple but effective way of determining the most appropriate Eps and MinPts values for a data set.Even though we did not implement the k-dist method for the parameters determination in our experiments, we did attempt to use two programms implemented in C++ -the code can be found in the Appendix B - one for each parameter. The Epsilon estimator outputs the total amount of neighboors of a segment of a data set. It also provides the difference in the amount of neighboors between a segment and the one that precedes it as well as the maximum distance between an arbitary point and its nearest neighboor. This very value is the suggested Epsilon. The number of segments in which the data set will be split and examined is typed by the user and certain values might make the suggestion of appropriate Epsilon more prominent. The MinPoints Estimator requires a predefined Epsilon value and a user defined number of segments in order to output the number of data set items that, given the Epsilon, have a certain amount of minPoints per segment. Both estimators are parametric themselves and can give a vague idea of the structure of the data set, without necessarily providing reliable suggested values. In fact, they were mostly used for us to determine the scale of the values we should use for each case, since relying on the output suggestion did not always give ideal results.

Datasets	PSDBSAN1 (borders-only)		PSDBSCAN2 (plus-outliers)	
	Epsilon	MinPoints	Epsilon	MinPoints
LR	15	32	20	32
PD	1000	5	1000	5
LS	1500	72	1500	72
TXR	0,1	10	0,5	200
PH	2	500	2	500
BL	1	5	1	5
ECL	2500	70	2500	75
YS	0,1	500	0,1	500
TN	10	5	20	5
MN2	4	71	4	71

Table 4.6: Parametric values.

Another observation that we choose to try out is the idea that a minimum minPts can be decided in association with the number of dimensions in the data set, meaning the total

number of attributes for each item. It must be noted that the value of minPts = 1 is pointless, since that would mean that each point can stand as a cluster on its own, while with $minPts \leq 2$, the result will be the same as of hierarchical clustering with the single link metric, with the dendrogram cut at height Epsilon. Consequently, MinPoints must be chosen at least 3. For relatively short data sets, $minPts = 2 * dim$ could be a useful value but in general, larger values are usually better for large data sets with noise or for data that contain many duplicates, since they will yield more significant clusters. Indeed, some of the cases where we used a minPts value that equals the double of the dimension gave better results than most, but not for all data sets.

That being said, the parametric values used in the experiments, which are displayed in the table 4.1.2, are only indicative, as they were determined through trial and error for each data set and for each of the two alternatives of PSDBSCAN. For our random experimenting values, we took into account the aforementioned observations, but we did not execute a full grid search or tuning procedure. Our initial goal was to find values that can lead to a reduction rate that is approximately equal or higher to that of RSP3, since it is the one out of all comparing algorithms that has a remarkably low reduction rate. There are only a few cases where our reduction rate falls far below, but even then we tried to keep the difference lower than a 10% and this exception was applied only for cases where the accuracy had a noticeably negative difference otherwise.

4.2 Experimental Results

The results of our experiments for each one of the alternatives of the proposed idea are presented in the tables below. For each combination of data set and algorithm, the parameters that gave the best results in aspect of classification accuracy were used (as shown in the previous section) and in each table, the best metrics are shown in bold. For reference, we mention the precision values of the conventional kNN classifier in the table of Accuracy. In addition, we present the measurements of the ENN rule. The latter reveal the noise level in data sets.

The tests confirm that the proposed algorithms are unfortunately not as effective as the already existing data reduction techniques. Taking into account that we keep the reduction rate above a certain, relatively low, percentage, both condensing sets of PSDBSCAN1 and 2 result to a lower classification accuracy for the majority of the data sets. There are only few exceptions, namely the data sets BL, ECL, YS and TN. From those, with the exception of ECL, it is important to notice that not only the accuracy is higher, but it was also achieved with a reduction rate that is higher than the minimum limit we decided to keep. Other than that, classification accuracy percentages of our algorithms for the rest of the data sets would unfortunately be either close but still lower than the lowest of the comparing values or just totally low, with a difference that exceeds 5-10%. Between

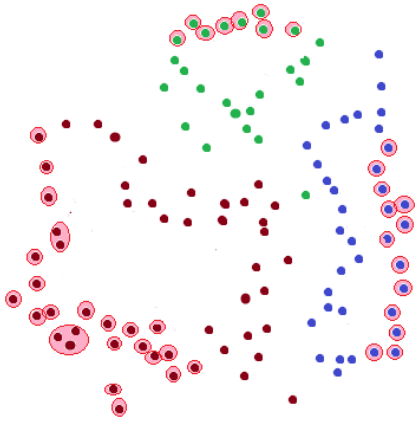

Figure 4.1: Example of condensing set from figure 3.1. Circled items do not play a significant role during classification, even though they are border items.

the two alternatives, PSDBSCAN2 has slightly better results, since it retains outliers and it includes more items that probably lie between clusters. Even if those are not border items, they still belong to the class-boundaries areas -considering that each cluster or cluster side belongs to one class. This does not negate the fact that some outliers might lie in areas of no interest for the classiffier, but either way has a slight impact on the reduction rate. It is also important to note again that all of the data sets we used are not edited. Therefore, PSDBSCAN2 might include as "outliers" items that are actually classification noise - normally ignored by DBSCAN and its notions during clustering- which can be quite misleading during the classification process. Last but not least, there are border items of clusters that are also border items of the entire data set or borders of areas of a cluster that do not abut other clusters and consequently do not lie in class determination boundaries. Those are also included in our condensing sets, decreasing our reduction rate without playing a significant role during the classification by increasing the accuracy.

In terms of preprocessing cost, the results are once again discouraging, but this can be easily explained by the fact that both PSDBSCAN1 and 2 require the finding of the Epsilon Neighborhood for each item in the data set, in order to determine whether it is a core item or not. In both cases, core items are the first to be excluded from the condensing set and they are required in PSDBSCAN1 for the determination of the borders. The method getEpsilonNeighbours() that can be found in the Appendix and is responsible for the retrieval of the neighborhood involves the computation of distance between the item whose neighborhood we are searching for and every single item of the data set -to see if that belongs to the neighborhood. As a result, not only we have a big amount of needed distance calculations, but also all of the distances between two different points will be calculated twice -once for every point while searching for its neighbors.

Dataset	ENN	CNN	IB2	RSP3	PSDBSCAN1 (borders-only)	PSDBSCAN2 (plus-outliers)
LR	4.33	83.54	**85.66**	61.98	70.96	68.40
PD	0.67	95.36	**96.23**	89.22	89.16	74.15
LS	9.07	80.22	**84.62**	73.19	79.27	63.67
TXR	1.24	91.90	**93.33**	83.31	85.24	78.18
PH	11.25	76.04	**80.85**	69.94	68.47	67.13
BL	-	65.72	69.36	64.64	**71.20**	69.56
ECL	-	59.55	**68.77**	52.27	52.23	43.01
YS	-	32.68	44.82	27.36	**62.99**	58.41
TN	3.61	82.09	88.25	84.56	86.64	**94.27**
MN2	2.08	87.23	**91.68**	61.33	54.75	54.75
Avg	4.03	75.43	**80.35**	66.78	72.09	67.15

Table 4.7: Experimental results: Reduction Rate (%)

Dataset	1-NN	ENN	CNN	IB2	RSP3	PSDBSCAN1 (borders-only)	PSDBSCAN2 (plus-outliers)
LR	95.83	94.98	92.84	91	**95.43**	73.98	93.03
PD	99.35	99.30	98.68	98.04	**99.05**	93.06	95.95
LS	90.60	90.29	88.21	86.87	**90.57**	83.37	72.26
TXR	99.02	98.64	97.16	96.35	**98.29**	83.64	80.27
PH	90.10	88.14	**87.82**	85.57	86.94	79.74	79.93
BL	78.4	-	70.88	70.72	73.28	**84.80**	84.0
ECL	79.78	-	72.05	66.97	**73.53**	64.71	**73.53**
YS	52.02	-	49.06	46.02	50.47	52.53	**53.54**
TN	94.88	95.69	92.00	89.15	92.68	**97.43**	94.80
MN2	90.51	89.58	**95.84**	93.75	91.22	85.06	85.06
Avg	87.49	93.80	76.77	82.44	**85.14**	79.83	81.24

Table 4.8: Experimental results: Accuracy (%)

Dataset	**ENN**	**CNN**	**IB2**	**RSP3**	**PSDBSCAN1 (borders-only)**	**PSDBSCAN2 (plus-outliers)**
LR	127.99	163.03	23.37	326.52	**394.72**	336.86
PD	38.65	11.75	1.78	86.66	**97.30**	**97.30**
LS	13.25	17.99	2.22	**37.70**	1.37	36.12
TXR	9.68	5.65	0.84	**27.63**	26.54	23.57
PH	9.35	13.45	1.96	20.31	**24.82**	**24.82**
BL	-	0.21	0.04	0.3	**0.32**	**0.32**
ECL	-	0.06	0.003	0.08	**0.11**	**0.11**
YS	-	1.41	0.19	**2.12**	1.99	1.99
TN	17.52	22.13	2.07	37.13	**70.08**	37.04
MN2	0.06	0.04	0.006	0.13	**0.17**	**0.17**
Avg	30.92	23.57	3.24	53.85	**65.22**	62.02

Table 4.9: Experimental results: Preprocessing Cost (millions of distance computations)

5 Conclusions and Future Work

Searching for alternative data reduction techniques is an important research field considering the impact of such techniques in improving the classification process. For this purpose, finding a way of utilizing already known algorithms that are widely implemented in data mining and machine learning tools -like DBSCAN algorithm- is something that can accelerate and ease the research process directed in this issue. In this thesis we proposed two alternatives based on the notions of density-based clustering and the type of items of a data set depending on their position. The main idea in which our approaches are based on is the fact that some of the training items, namely the ones that lie closer to class boundary areas, are more pivotal during classification than others, and thus should be retained in a condensing set. DBSCAN is an algorithm that can isolate such items.

The proposed algorithms belong to the category of prototype selection, since they select and retain items directly from the original training sets. While DBSCAN is used for clustering the data by identifing items as cores, borders and outliers, Prototype Selection DBSCAN1 constructs a condensing set using only the items that are considered borders and Prototype Selection DBSCAN2 keeps both borders and outliers. Both of the alternatives have been executed for data sets in their entirety, assuming that the clusters -and consequently, the items' position types derived from them- might not reflect separate classes. The same algorithms were initially meant to be tested for each individual class in every data set, but the results were discouraging from an early stage, therefore that approach was abandoned. However, their performance for whole data sets was unfortunately less efficient than expected as well, with some few but important to mention exceptions.

It turns out that even though our initial speculation can be applied successfully in some simplified cases of data set structures, our algorithms face some basic weaknesses. The first one is the fact that the algorithm in which they are based of is parametric, so both approaches inevitably rely on parametric values that vary significantly from data set to data set and separate procedures are required for their determination in each case. It is important to note that the values we used for our experiments are only indicative and occurred through trial and error experiments, but while selecting them we did took into account several observations associated with best practices for their determination. Last but not least, by selecting borders and/or outliers we also include items that lie on the edges of a data set, or in areas of no neighbouring clusters. Those items do not really fit in the close to class-boundaries criterion we had in mind and are probably not particularly useful to a classifier. By keeping them, we lower the reduction rate, without helping the increase of the accuracy.

Despite the generally negative results, there were, as mentioned before, some exceptions. Our alternatives did work as expected for a couple of data sets, but we could not detect

a similarity between them that differentiates them from the rest of the data sets or can confirm the reasons why we believe our approach failed for the majority of them. A visualization of those data sets would be enough to reinforce or cancel the assumption that the structure of the data set -regardless the number or shapes of the clusters- is one of the main factors that determines whether the proposed algorithms are appropriate for usage. In other words, it is recommended that future research towards this direction should include whether there are certain forms, sizes or values of dimensionality that allow one or both of PSDBSCANs to be used successfully.

However, the subject of future research on which it would be better to focus is eliminating the aforementioned weaknesses of the proposed alternatives, as an attempt to improve their performance and possibly establish one or both of them as a reliable data reduction technique. As far as the determination of the parameters is concerned, multiple experiments through grid search would give us a better idea of the appropriate values for each combination of data set and proposed alternative, and so would a full tuning procedure. It is also worth to mention that there are other known algorithms, such as OPTICS (Ordering points to identify the clustering structure), whose basic idea is similar to DBSCAN while at the same time, it is not affected by the minPts parameter and providing a maximum value of Epsilon is only optional. Another advantage of this DBSCAN variation is that it detects clusters of varying density in the same data set, which means that if its logic is used for our purpose, it can provide border items and outliers of regions that could otherwise be ignored or misrepresented -because of the global usage of the predefined Epsilon and minPts and the "fixed" global density they define for the entire data set. That would of course lead to two different alternatives than the ones we used, since the base algorithm is -only slightly- different, but the motivation and later contribution in case of successful results remain the same, especially since OPTICS is not only similarly structured with DBSCAN but also already implemented in some useful data mining and machine learnign tools, like Scikit-Learn.

Considering the possibility of classification noise -which is normally ignored by DBSCAN since it is a clustering algorithm and no classified items are involved during clustering- it seems that no alternative is entirely safe without using an editing algorithm prior to prototype selection, especially before PSDBSCAN2, which is the most promising approach, yet includes a wider range of items. Therefore experiments on noise free data sets might be useful in order to see whether there is going to be a significant difference. Noise removal algorithms can guarantee that we end up with regions belonging to a particular class, therefore clustering after noise removal would result to clusters that correspond to one class each and they would probably have discreet borders. That would be an ideal data set structure as it renders the base of speculation, but real data sets rarely have this form without undergoing editing.

As for the border items that lie close to the edges of the data set or in areas where there are no other border items of a different cluster within a specified radius, excluding them from the condensing set with an assisting algorithm might not be ideal in terms of preprocessing cost and it even contradicts our motivation -which is to utilize a pre-existing algorithm for research purposes. It can still be tried and tested accordingly though, however it must be noted that it is more risky and it relies on a parametric value as well. A safer option that resembles the logic of CNN is the following: for every border item or outlier that is about to be extracted into the condensing set, we could check the classes of its neighbors. Then the border item or outlier would be indeed included in the condensing set only if its class differs from the major class of the neighborhood. In that way, we could reduce the amount of border items that do not neighbor other cluster borders.

To conclude, even though the experimental results were not expected, there are clearly many ways in which this research can be continued and possibly improved. The present dissertation is only a preliminary attempt towards our speculation and there are still many more assisting methods and improvements that worth to be tried.

References

[1] S. Ougiaroglou, *Algorithms and Techniqes for Efficient and Effective Nearest Neighbours Classification*. PhD thesis, Department of Applied Informatics School of Information Sciences, University of Macedonia, 6 2014.

[2] S. Garcia, J. Derrac, J. Cano, and F. Herrera, "Prototype selection for nearest neighbor classification: Taxonomy and empirical study," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, pp. 417–435, Mar. 2012.

[3] I. Triguero, J. Derrac, S. Garcia, and F. Herrera, "A taxonomy and experimental study on prototype generation for nearest neighbor classification," *Trans. Sys. Man Cyber Part C*, vol. 42, pp. 86–100, Jan. 2012.

[4] Google, "Machine Learning Crash Course - clustering algorithms." `https://developers.google.com/machine-learning/clustering/clustering-algorithms`. Accessed: 2020-06-01.

[5] G. Academic, "NCKU Data Mining: Clustering." `https://sejkai.gitbook.io/academic/ncku-data-mining/clustering`. Accessed: 2020-06-01.

[6] L. S. Marzena Kryszkiewicz, "Faster Clustering with DBSCAN," in *Intelligent Information Processing and Web Mining*, (Institute of Computer Science, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland), pp. 605 – 614, 2005.

[7] D. Peixoto, "Mining trajectory data." `https://www.researchgate.net/publication/275381558_Mining_Trajectory_Data`, November 2013. Accessed: 2020-06-01.

[8] H.-P. K. J. S. Mihael Ankerst, Markus M. Breunig, "OPTICS: Ordering Points To Identify the Clustering Structure," (Institute for Computer Science, University of Munich Oettingenstr. 67, D-80538 München, Germany), 1999.

[9] C. Sinclair, "Clustering Using OPTICS - a seemingly parameter-less algorithm." `https://towardsdatascience.com/clustering-using-optics-cac1d10ed7a7`, January 2019. Accessed: 2020-06-01.

[10] M. James, *Classification algorithms*. New York, NY, USA: Wiley-Interscience, 1985.

[11] L. Rokach, *Data Mining with Decision Trees: Theory and Applications*. Series in machine perception and artificial intelligence, World Scientific Publishing Company, Incorporated, 2007.

[12] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2nd ed., 1998.

[13] G. P. Zhang, "Neural networks for classification: A survey," *Trans. Sys. Man Cyber Part C*, vol. 30, pp. 451–462, Nov. 2000.

[14] B. V. Dasarathy, *Nearest neighbor (NN) norms : NN pattern classification techniques*. IEEE Computer Society Press, 1991.

[15] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theor.*, vol. 13, pp. 21–27, Sept. 2006.

[16] E. Deza and M. M. Deza, *Encyclopedia of Distances*. Berlin, Heidelberg: Springer, 2009.

[17] P. E. Hart, "The condensed nearest neighbor rule," *IEEE Transactions on Information Theory*, vol. 14, no. 3, pp. 515–516, 1968.

[18] D. W. Aha, "Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms," *Int. J. Man-Mach. Stud.*, vol. 36, pp. 267–287, Feb. 1992.

[19] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," *Data Min. Knowl. Discov.*, vol. 6, pp. 153–172, Apr. 2002.

[20] C. H. Chen and A. Jóźwik, "A sample set condensation algorithm for the class sensitive artificial neural network," *Pattern Recogn. Lett.*, vol. 17, pp. 819–823, July 1996.

[21] J. S. Sánchez, "High training set size reduction by space partitioning and prototype abstraction," *Pattern Recognition*, vol. 37, no. 7, pp. 1561–1564, 2004.

[22] S. Ougiaroglou and G. Evangelidis, "A simple noise-tolerant abstraction algorithm for fast k-nn classification," in *Hybrid Artificial Intelligent Systems* (E. Corchado, V. Snášel, A. Abraham, M. Woźniak, M. Graña, and S.-B. Cho, eds.), vol. 7209 of *Lecture Notes in Computer Science*, pp. 210–221, Springer Berlin Heidelberg, 2012.

[23] J. S. X. X. Martin Ester, Hans-Peter Kriegel, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *KDD-96*, (Institute for Computer Science, University of Munich Oettingenstr. 67, D-80538 München, Germany), pp. 226– 231, 1996.

[24] S. learn developers, *Scikit-learn user guide*. Scikit-learn.org, March 2019.

[25] N. community, *NumPy reference*. SciPy.org, March 2019.

[26] M. K. Patel, *Pandas guide*, October 2018.

[27] F. Thabtah, "A review of associative classification mining," *Knowl. Eng. Rev.*, vol. 22, pp. 37–65, Mar. 2007.

[28] T. Y. Xu Dongkuan, "A comprehensive survey of clustering algorithms," *Annals of Data Science*, vol. 2, pp. 165–193, June 2015.

[29] S. Ougiaroglou and G. Evangelidis, "Efficient dataset size reduction by finding homogeneous clusters," in *Proceedings of the Fifth Balkan Conference in Informatics*, BCI '12, (New York, NY, USA), pp. 168–173, ACM, 2012.

[30] S. Ougiaroglou and G. Evangelidis, "RHC: Non-parametric cluster-based data reduction for efficient k-nn classification," *Pattern Analysis and Applications*.

[31] S. Ougiaroglou and G. Evangelidis, "AIB2: An abstraction data reduction technique based on ib2," in *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, (New York, NY, USA), pp. 13–16, ACM, 2013.

[32] S. Ougiaroglou and G. Evangelidis, "Efficient data abstraction using weighted ib2 prototypes," *Computer Science and Information Systems (ComSIS)*.

[33] J. A. Olvera-López, J. F. Martínez-Trinidad, and J. A. Carrasco-Ochoa, "Mixed data object selection based on clustering and border objects," in *Proceedings of the Congress on pattern recognition 12th Iberoamerican conference on Progress in pattern recognition, image analysis and applications*, CIARP'07, (Berlin, Heidelberg), pp. 674–683, Springer-Verlag, 2007.

[34] J. A. Olvera-Lopez, J. A. Carrasco-Ochoa, and J. F. M. Trinidad, "A new fast prototype selection method based on clustering.," *Pattern Anal. Appl.*, vol. 13, no. 2, pp. 131–141, 2010.

[35] J. A. Olvera-López, J. A. Carrasco-Ochoa, and J. F. M. Trinidad, "Object selection based on clustering and border objects," in *Computer Recognition Systems 2* (M. Kurzynski, E. Puchala, M. Wozniak, and A. Zolnierek, eds.), vol. 45 of *Advances in Soft Computing*, pp. 27–34, Springer, 2008.

[36] A. Tsymbal, "The problem of concept drift: definitions and related work," Tech. Rep. TCD-CS-2004-15, The University of Dublin, Trinity College, Department of Computer Science, Dublin, Ireland, 2004.

[37] M. Lozano, *Data Reduction Techniques in Classification processes (Phd Thesis)*. Universitat Jaume I, 2007.

[38] G. Toussaint, "Proximity graphs for nearest neighbor decision rules: Recent progress," in *34th Symposium on the INTERFACE*, pp. 17–20, 2002.

[39] D. R. Wilson and T. R. Martinez, "Reduction techniques for instance-basedlearning algorithms," *Mach. Learn.*, vol. 38, pp. 257–286, Mar. 2000.

[40] N. Jankowski and M. Grochowski, "Comparison of instances seletion algorithms i. algorithms survey," in *Artificial Intelligence and Soft Computing - ICAISC 2004*, vol. 3070 of *Lecture Notes in Computer Science*, pp. 598–603, Springer Berlin / Heidelberg, 2004.

[41] M. Grochowski and N. Jankowski, "Comparison of instance selection algorithms ii. results and comments," in *Artificial Intelligence and Soft Computing - ICAISC 2004*, vol. 3070 of *LNCS*, pp. 580–585, Springer Berlin / Heidelberg, 2004.

[42] J. A. Olvera-López, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, and J. Kittler, "A review of instance selection methods," *Artif. Intell. Rev.*, vol. 34, pp. 133–143, Aug. 2010.

[43] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, pp. 1–37, Jan 2008.

[44] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[45] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE trans. on systems, man, and cybernetics*, vol. 2, pp. 408–421, July 1972.

[46] I. Tomek, "An experiment with the edited nearest-neighbor rule," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 6, pp. 448–452, 1976.

[47] P. A. Devijver and J. Kittler, "On the edited nearest neighbor rule," in *Proceedings of the Fifth International Conference on Pattern Recognition*, The Institute of Electrical and Electronics Engineers, 1980.

[48] M. García-Borroto, Y. Villuendas-Rey, J. A. Carrasco-Ochoa, and J. F. Martínez-Trinidad, "Using maximum similarity graphs to edit nearest neighbor classifiers," in *Proceedings of the 14th Iberoamerican Conference on Pattern Recognition: Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, CIARP '09, (Berlin, Heidelberg), pp. 489–496, Springer-Verlag, 2009.

[49] J. Sánchez, F. Pla, and F. Ferri, "Prototype selection for the nearest neighbour rule

through proximity graphs," *Pattern Recognition Letters*, vol. 18, no. 6, pp. 507 – 513, 1997.

[50] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Mach. Learn.*, vol. 6, pp. 37–66, Jan. 1991.

[51] J. McQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of 5th Berkeley Symp. on Math. Statistics and Probability*, (Berkeley, CA : University of California Press), pp. 281– 298, 1967.

[52] J. Wu, *Advances in K-means Clustering: A Data Mining Thinking*. Springer Publishing Company, Incorporated, 2012.

[53] B. S. Everitt, S. Landau, and M. Leese, *Cluster Analysis*. Wiley Publishing, 4th ed., 2009.

[54] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, vol. 31, pp. 264–323, Sept. 1999.

[55] P. Berkhin, "A survey of clustering data mining techniques," *Grouping Multidimensional Data*, pp. 25–71, 2006.

[56] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[57] J. L. J. D. S. G. L. S. F. H. J. Alcalá-Fdez, A. Fernandez, "Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. journal of multiple-valued logic and soft computing," 2011.

Appendices

A PSDBSCAN Source Code

Listing 1: Item.java

```
package dbscan_wholedataset;

/**
 * @author Athina Drakopoulou
 */
public class Item {
   float attributes[];
   int classAtt;
  /**
   * -2:Noise
   * -1:Unclassified
    * 0++ :Cluster
  */
   int cluster_id = -1;
  boolean corePoint = false;
   boolean borderPoint = false;

   public Item() {}

   public Item(float[] attributes, int classAtt, int cluster_id, boolean
       corePoint, boolean borderPoint) {
       this.attributes = attributes;
       this.classAtt = classAtt;
       this.cluster_id = cluster_id;
       this.corePoint = corePoint;
       this.borderPoint = borderPoint;
   }

   public void setAttributes(float[] attributes) {
       this.attributes = attributes;
   }

   public void setClassAtt(int classAtt) {
       this.classAtt = classAtt;
   }
```

```
public void setCluster_id(int cluster_id) {
    this.cluster_id = cluster_id;
}

public void setCorePoint(boolean corePoint) {
    this.corePoint = corePoint;
}

public void setBorderPoint(boolean borderPoint) {
    this.borderPoint = borderPoint;
}

public float[] getAttributes() {
    return attributes;
}

public int getClassAtt() {
    return classAtt;
}

public int getCluster_id() {
    return cluster_id;
}

public boolean isBorderPoint() {
    return borderPoint;
}

public boolean isCorePoint() {
    return corePoint;
}

@Override
public String toString() {
    String s = "";
    for(int i=0;i<attributes.length;i++){
        s+= String.valueOf(attributes[i]) + "\t";
    }
    s+=classAtt + "\n";
    return s;
}

public String printID(){
```

```
        return "Cluster " + cluster_id + " of class " + classAtt + ". Border
            Point: " + borderPoint + "\n";
    }
}
```

Listing 2: Utils.java

```
package dbscan_wholedataset;

import dbscan_wholedataset.Item;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

/**
 * @author Athina Drakopoulou
*/
public class Utils {

    //List of classes
    static ArrayList<Integer> cl = new ArrayList<Integer>();

    static long computations;

    //Counter of lines and columns, given '\t' as delimiter
    public int[] Counter(String filename) {
    int lines = 0, attributes = 0;
    try {
        FileReader obj2 = new FileReader(filename);
        BufferedReader filtered_obj2 = new BufferedReader(obj2);
        while (filtered_obj2.readLine() != null) {
            lines++;
        }
        filtered_obj2.close();
        FileReader obj3 = new FileReader(filename);
        BufferedReader filtered_obj3 = new BufferedReader(obj3);
        String line1 = filtered_obj3.readLine();
        char line[] = line1.toCharArray();
        for (int i = 0; i < line.length; i++) {
            if (line[i] == '\t') {
```

```
            attributes++;
        }
    }
    filtered_obj3.close();
} catch (IOException ex) {
    System.out.println("No file with this name.");
}
int table[] = {lines, attributes};
return table;
}

/**
 * Method for filling a list with the Items of the given file
 * @param filename
 * @param dim[], table of 2 integer values. The first represents the
    amount of items, the second the amount of attributes
 * @return ArrayList<Item>
*/
public ArrayList<Item> fillTable(String filename, int dim[]) {
    //The size of the final list equals the amount of the items
    ArrayList<Item> dataAtt = new ArrayList<Item>(dim[0]);
    String s, sarray[];
    float tmp = 0;
    boolean flag = true;
    int numberOfClasses = 0;

    try {
        FileReader obj4 = new FileReader(filename);
        BufferedReader filtered_obj4 = new BufferedReader(obj4);

        for (int i = 0; i < dim[0]; i++) {
            dataAtt.add(new Item());

        //The length of the table of attributes equals the amount of
            attributes
            dataAtt.get(i).attributes = new float[dim[1]];
            s = filtered_obj4.readLine();
            sarray = s.split("\t");

            //The class of each item is stored in a separate structure
            dataAtt.get(i).classAtt = Integer.valueOf(sarray[sarray.length
                - 1]);
            if (cl.isEmpty()) {
```

```
                cl.add(dataAtt.get(i).classAtt);
            } else {
                for (int k = 0; k < cl.size(); k++) {
                    if (cl.get(k) == dataAtt.get(i).classAtt) {
                        flag = false;
                    }
                }
                if (flag) {
                    cl.add(dataAtt.get(i).classAtt);
                }
                flag = true;
            }
            for (int j = 0; j < sarray.length - 1; j++) {
                dataAtt.get(i).attributes[j] = Float.valueOf(sarray[j]);
            }
        }
        numberOfClasses = cl.size();
        filtered_obj4.close();
    } catch (IOException ex) {
        System.out.println("No file with this name.");
    }
    return dataAtt;
}

/**
 * Method for returning the filenames for which the algorithm will run
 * @param filename (generic, common part of the name)
 * @return String[] foldFiles
 */
public String[] keepFolds(String filename) {
    String foldFiles[] = new String[5];
    for (int i = 0; i < 5; i++) {
        foldFiles[i] = filename + (i + 1);
    }
    return foldFiles;
}

/**
 * Distance Calculator between items
 * @param Item a
 * @param Item b
 * @return float distance
 */
```

```
public float CalcDistance(Item a, Item b) {
    computations++;
    int att = a.attributes.length;
    float distance = 0, u;
    for (int j = 0; j < att; j++) {
        u = a.attributes[j] - b.attributes[j];
        distance = distance + u * u;
    }
    return distance;
}

/**
 * Neighbours retrieval of Item a
 * @param idx, index of item a
 * @param oneClassList, list of items of the set we're examining
 * @param epsilon parametre
 * @return EpsNeighbours (list of indexes of neighbors)
 */
private ArrayList<Integer> getEpsilon_Neighbours(int idx, ArrayList<Item>
    oneClassList, double epsilon) {
    float distance;
    ArrayList<Integer> EpsNeighbours = new ArrayList<Integer>();
    for (int i = 0; i < oneClassList.size(); i++) {
        if (i == idx) {
            continue;
        }
        distance = CalcDistance(oneClassList.get(i), oneClassList.get(idx));
        if (distance>epsilon) {
            continue;
        } else {
            EpsNeighbours.add(i);
        }
    }
    return EpsNeighbours;
}

/**
 * Separator of Items according to class
 * @param allinList, list of all Items of dataset, regardless of class
 * @return itemsPerClass
 */
@SuppressWarnings("unchecked")
public ArrayList<Item>[] separateClasses(ArrayList<Item> allinList) {
```

```
    ArrayList<Item>[] itemsPerClass = new ArrayList[cl.size()];
    ArrayList<Item> tmpClassList = new ArrayList<>();
    for (int i = 0; i < cl.size(); i++) {
        for (int j = 0; j < allinList.size(); j++) {
            if (allinList.get(j).classAtt == cl.get(i)) {
                tmpClassList.add(allinList.get(j));
            }
        }
        itemsPerClass[i] = new ArrayList<Item>(tmpClassList);
        tmpClassList.clear();
    }
    return itemsPerClass;
}

/**
 * DBSCANcores
 * Finding core items of dataset
 */
public void DBSCANcores(ArrayList<Item> aClassList, double epsilon, int
    minPoints) {
    ArrayList<Integer> tmpNeigh = new ArrayList<>();
    for (int i = 0; i < aClassList.size(); i++) {
        tmpNeigh = getEpsilon_Neighbours(i, aClassList, epsilon);
        if (tmpNeigh.size() >= minPoints) {
            aClassList.get(i).setCorePoint(true);
        }
    }
}

/**
 * notCores
 * @return ArrayList<Item> that are not cores
 */
public ArrayList<Item> notCores(ArrayList<Item> aClassList){
    ArrayList<Item> noCores = new ArrayList<>();
    for (int j = 0; j < aClassList.size(); j++) {
        if (!aClassList.get(j).isCorePoint()) {
            noCores.add(aClassList.get(j));
        }
    }
    return noCores;
}
```

```
    /**
     * DBSCANborders
     * Finding border items of dataset
     */
    public ArrayList<Item> DBSCANborders(ArrayList<Item> aClassList, double
        epsilon) {
        ArrayList<Integer> tmpNeigh = new ArrayList<>();
        ArrayList<Item> borderItems = new ArrayList<>();
        for (int j = 0; j < aClassList.size(); j++) {
            if (!aClassList.get(j).isCorePoint()) {
                tmpNeigh = getEpsilon_Neighbours(j, aClassList, epsilon);
                for (int k = 0; k < tmpNeigh.size(); k++) {
                    if (aClassList.get(tmpNeigh.get(k)).isCorePoint()) {
                        aClassList.get(j).setBorderPoint(true);
                        borderItems.add(aClassList.get(j));
                        break;
                    }
                }
            }
        }
        return borderItems;
    }

    /**
     * Condensing set file constructor
     */
    public void createCSFile(String filename, ArrayList<Item> selectedItems,
        int trNumber) throws IOException {
        String tmpFilename;
        tmpFilename = filename + trNumber;
        FileWriter bf = new FileWriter(tmpFilename);
        BufferedWriter filterBF = new BufferedWriter(bf);
        for (int j = 0; j < selectedItems.size(); j++) {
            filterBF.write(selectedItems.get(j).toString());
        }
        filterBF.close();
    }
}
```

Listing 3: DBSCAN2.java - main for Condensing Set that includes only border items

```
package dbscan_wholedataset;
```

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * @author Athina Drakopoulou
 */
public class DBSCAN2 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        if (args.length != 4) {
            System.out.println("Invalid number of parameters(input file, output
                file, epsilon & minPoints).");
        }

        Utils obj = new Utils();
        int minP;
        double eps;
        double RR = 0, sumRR = 0;
        long fcost, preCost=0;
        ArrayList<Item> fileBorders = new ArrayList<>();

        String filename = args[0];
        String outputFile = args[1];
        eps = Double.parseDouble(args[2]);
        minP = Integer.parseInt(args[3]);

        String folds[] = obj.keepFolds(filename);
        System.out.println("Epsilon: " + eps + " | minPoints: " + minP);

        for (int i = 0; i < folds.length; i++) {
            int[] table = obj.Counter(folds[i]);
            ArrayList<Item> dataSet = obj.fillTable(folds[i], table);
            obj.DBSCANcores(dataSet, eps, minP);
            fileBorders = obj.DBSCANborders(dataSet, eps);
            try {
                obj.createCSFile(outputFile, fileBorders, i + 1);
            } catch (IOException ex) {
                Logger.getLogger(DBSCAN2.class.getName()).log(Level.SEVERE,
```

```
                null, ex);
            }
            RR = ((double) (dataSet.size() - fileBorders.size()) /
                dataSet.size()) * 100;

            preCost = preCost + Utils.computations;
            fcost = Utils.computations;

            System.out.println("| Reduction Rate " + (i + 1) + " : " + RR + "%
                | Borders: " + fileBorders.size() + " | Computations: " +
                fcost);
            Utils.computations = 0;
            sumRR = sumRR + RR;
            fileBorders.clear();
        }
        double avgPreCost = (double)(preCost / folds.length);

        System.out.println("| Average Reduction Rate : " + sumRR /
            folds.length + "%");
        System.out.println("Average Computations Amount: " + preCost /
            folds.length);
    }
}
```

Listing 4: DBSCAN2.java - main for Condensing Set that includes only border items and outliers[3]

```
package dbscan_wholedataset;

import java.io.IOException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Athina Drakopoulou
 */
public class DBSCAN2 {

/**
 * @param args the command line arguments
 */
```

```
public static void main(String[] args) {
    if (args.length != 4) {
        System.out.println("Invalid number of parameters(input file, output
            file, epsilon & minPoints).");
    }

    Utils obj = new Utils();
    int minP;
    double eps;
    double RR = 0, sumRR = 0;
    long preCost = 0, fcost;
    ArrayList<Item> fileBorders = new ArrayList<>();
    ArrayList<Item> fileNotCores = new ArrayList<>();

    String filename = args[0];
    String outputFile = args[1];
    eps = Double.parseDouble(args[2]);
    minP = Integer.parseInt(args[3]);

    String folds[] = obj.keepFolds(filename);

    for (int i = 0; i < folds.length; i++) {
        int[] table = obj.Counter(folds[i]);
        ArrayList<Item> dataSet = obj.fillTable(folds[i], table);
        //ArrayList<Item>[] itemsPerClass = obj.separateClasses(dataSet);
        obj.DBSCANcores(dataSet, eps, minP);
        fileBorders = obj.DBSCANborders(dataSet, eps);
        fileNotCores = obj.notCores(dataSet);
        try {
            obj.createCSFile(outputFile, fileNotCores, i + 1);
        } catch (IOException ex) {
            Logger.getLogger(DBSCAN2.class.getName()).log(Level.SEVERE, null,
                ex);
        }

        preCost = preCost + Utils.computations;
        fcost = Utils.computations;

        RR = ((double) (dataSet.size() - fileNotCores.size()) /
            dataSet.size()) * 100;
        System.out.println("Reduction Rate " + (i + 1) + " : " + RR + "% |
            Borders: " + fileBorders.size() + " | Outliers: " +
            (fileNotCores.size()-fileBorders.size()) + " | Computations: " +
```

```
            fcost);
        sumRR = sumRR + RR;
        fileBorders.clear();
        fileNotCores.clear();
            Utils.computations = 0;
        }
        double avgPreCost = (double)(preCost/folds.length);

        System.out.println("Average Reduction Rate : " + sumRR / folds.length
            + "%");
        System.out.println("Average Computations Amount : " + avgPreCost );
    }
}
```

[3]Precisely, any item that is not a core in a noise-free data set

Appendices

B DBSCAN Parameters' Estimators

Listing 5: eEstimator.cpp

```
#include <stdlib.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <string>
#include <cstring>
#include <cstdlib>

struct TrainItem{
   int classAttr;
   float *attr;
   double distKNN;
};

int convKNN;
int ATTR;
int CLASSES;

using namespace std;

void readData(TrainItem[], char[], int);
double distKNN(TrainItem[], int, int);
int countLinesAttrs(char[], int&, int&);
void quicksort(TrainItem[], int, int);
void quicksort(int [], int low, int high);

int main(int argc, char *argv[]){
   static TrainItem *TR;
   int n;

   if (argc != 4){
      cout<<"ERROR. Number of parameters"<<endl;
      return 1;
   }
   if (!isdigit(*argv[2])){
      cout<<"ERROR. parameter must be numeric"<<endl;
```

```
   return 1;
}
if (!isdigit(*argv[3])){
   cout<<"ERROR. parameter must be numeric"<<endl;
   return 1;
}
convKNN=atoi(argv[2]);
int segs = atoi(argv[3]);

char *fileName = new char[strlen(argv[1])];
strcpy(fileName,argv[1]);
if (countLinesAttrs(fileName, n, ATTR)){
   cout<<"File "<<fileName<<" does not exist"<<endl;
   return 1;
}

TR = new TrainItem[n];
readData(TR, fileName, n);

cout <<"Nearest Neighbours: "<< convKNN<<endl;
cout<<"classes: "<<CLASSES<<endl;
cout<<"Attributes: "<<ATTR<<endl;
cout<<"DataCounter: "<<n<<endl;

double max2=0;
for (int i=0; i<n; i++){
   TR[i].distKNN = distKNN(TR, i, n);
   if (TR[i].distKNN > max2){
      max2=TR[i].distKNN; //h megaliteri kontinoteri apostasi
   }
}

double limit = (double)max2 / (double)segs;
int *counter = new int[segs];

for (int i=0; i<segs; i++){
   counter[i] = 0;
}
for (int i=0; i <n; i++){
   for (int j=1; j<=segs; j++){
      if (TR[i].distKNN <= limit *j){
         counter[j-1]++;
         break;
```

```
            }
        }
    }

    int maxx = abs(counter[0]-counter[1]);
    int th = 0;
    for (int i=0; i<segs; i++){
        cout<<i*limit<<"-"<<(i+1)*limit<<" : "<<counter[i]<<" : ";
        if (i>0){
            cout<<abs(counter[i-1]-counter[i]);
            if (abs(counter[i-1]-counter[i]) > maxx){
                maxx = abs(counter[i-1]-counter[i]);
                th = i*limit;
            }
        }
        cout<<endl;
    }

    cout<<"e suggestion: "<<th<<endl;

    return 0;
}

double distKNN(TrainItem trainData[], int q, int n){
    int h, g, k, j;
    double u,x, *min;
    float dist;

    min = new double[convKNN];

    for (h=0; h<convKNN; h++){
        min[h]=9999999999999999;
    }

    for (k=0; k<n; k++){
        if (k == q){
            continue;
        }
        x=0;

        for (j=0; j<ATTR; j++){
            u = (float)trainData[k].attr[j] - (float)trainData[q].attr[j];
            x=x + u*u;
```

```
      }
      dist=x;
      for (h=0; h<convKNN; h++) {
         if (dist<min[h]){
            for (g=convKNN-1; g>h; g--){
               min[g]=min[g-1];
            }
            min[h]=dist;
            break;
         }
      }
   }
   return min[convKNN-1];
}

void readData(TrainItem trainData[], char fileName[], int n){
   int i,j;

   ifstream dat;
   dat.open(fileName);
   CLASSES=0;
   bool fl;

   for (i=0; i<n; i++){
      trainData[i].attr = new float[ATTR];
      for (j=0; j<ATTR; j++){
         dat>>trainData[i].attr[j];
      }
      trainData[i].distKNN=0;
      dat>>trainData[i].classAttr ;
      fl = false;
      for (j=0; j<i; j++){
         if (trainData[i].classAttr == trainData[j].classAttr){
            fl = true;
            break;
         }
      }
      if (!fl){
         CLASSES++;
      }
   }
   dat.close();
}
```

```
int countLinesAttrs(char fileName[], int &lines, int &attrs){
   int i, c;

   ifstream dat;
   dat.open(fileName);

   if (!dat){
      return 1;
   }

   string lline;

   getline(dat, lline);
   c=0;
   for (i=0; i<lline.length(); i++){
      if (lline[i] == '\t'){
         c++;
      }
   }
   dat.close();
   attrs = c;

   dat.open(fileName);
   i=0;
   while (!dat.eof()){
      getline(dat, lline);
      i++;
   }
   lines = i-1;
   dat.close();
   return 0;
}


void quicksort(TrainItem R[], int low, int high){
      int pivot, i, j;
      TrainItem temp;

      if(low < high) {
        pivot = low;
        i = low;
        j = high+1;
```

```
        while(1) {
          do ++i; while (R[i].distKNN >= R[pivot].distKNN && i<=high);
          do --j; while (R[j].distKNN < R[pivot].distKNN);
          if(i >= j) break;
          temp = R[i];
          R[i] = R[j];
          R[j] = temp;
        }
        temp = R[j];
        R[j] = R[pivot];
        R[pivot] = temp;

        quicksort(R, low, j-1);
        quicksort(R, j+1, high);
      }
}

void quicksort(int R[], int low, int high){
  int pivot, i, j;
  int temp;

  if(low < high) {
    pivot = low;
    i = low;
    j = high+1;
    while(1) {
      do ++i; while (R[i] >= R[pivot] && i<=high);
      do --j; while (R[j] < R[pivot]);
      if(i >= j) break;
      temp = R[i];
      R[i] = R[j];
      R[j] = temp;
    }
    temp = R[j];
    R[j] = R[pivot];
    R[pivot] = temp;

    quicksort(R, low, j-1);
    quicksort(R, j+1, high);
  }
}
```

Listing 6: minptsEstimator

```
#include <stdlib.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <string>
#include <cstring>
#include <cstdlib>

struct TrainItem{
   int classAttr;
   float *attr;
   int pts;
};

int epam;
int ATTR;
int CLASSES;

using namespace std;

void readData(TrainItem[], char[], int);
void distKNN(TrainItem[], int);
int countLinesAttrs(char[], int&, int&);

int main(int argc, char *argv[]){
   static TrainItem *TR;
   int n;

   if (argc != 4){
      cout<<"ERROR. Number of parameters"<<endl;
      return 1;
   }
   if (!isdigit(*argv[2])){
      cout<<"ERROR. parameter must be numeric"<<endl;
      return 1;
   }
   if (!isdigit(*argv[3])){
      cout<<"ERROR. parameter must be numeric"<<endl;
      return 1;
   }
   epam=atoi(argv[2]);
   int segs=atoi(argv[3]);
```

```
char *fileName = new char[strlen(argv[1])];
strcpy(fileName,argv[1]);
if (countLinesAttrs(fileName, n, ATTR)){
   cout<<"File "<<fileName<<" does not exist"<<endl;
   return 1;
}

TR = new TrainItem[n];
readData(TR, fileName, n);

cout <<"Segments: "<< segs<<endl;
cout <<"E parameter: "<< epam<<endl;
cout<<"classes: "<<CLASSES<<endl;
cout<<"Attributes: "<<ATTR<<endl;
cout<<"DataCounter: "<<n<<endl;

distKNN(TR, n);

int max2 = 0;
for (int i=0; i<n; i++){
   if (TR[i].pts > max2){
      max2=TR[i].pts;
   }
}

cout<<"the highest number of nn in a neighbourhood: "<<max2<<endl;

int limit = max2 / segs;
int *counter = new int[segs];

for (int i=0; i<segs; i++){
   counter[i] = 0;
}
for (int i=0; i <n; i++){
   for (int j=1; j<=segs; j++){
      if (TR[i].pts <= limit *j){
         counter[j-1]++;
         break;
      }
   }
}
```

```
   for (int i=0; i<segs; i++){
      cout<<i*limit<<"-"<<(i+1)*limit<<" : "<<counter[i]<<endl;
   }
   return 0;
}

void distKNN(TrainItem trainData[], int n){
   int h, g, k, j;
   double u,x;
   float dist;

   for (int q=0; q<n;q++){
      for (k=0; k<n; k++){
         if (k == q){
            continue;
         }
         x=0;

         for (j=0; j<ATTR; j++){
            u = (float)trainData[k].attr[j] - (float)trainData[q].attr[j];
            x=x + u*u;
         }
         dist=x;

         if (dist <= epam){
            trainData[q].pts++;
         }
      }
   }
}

void readData(TrainItem trainData[], char fileName[], int n){
   int i,j;

   ifstream dat;
   dat.open(fileName);
   CLASSES=0;
   bool fl;

   for (i=0; i<n; i++){
      trainData[i].attr = new float[ATTR];
      for (j=0; j<ATTR; j++){
         dat>>trainData[i].attr[j];
```

```
      }
      trainData[i].pts=0;
      dat>>trainData[i].classAttr ;
      fl = false;
      for (j=0; j<i; j++){
         if (trainData[i].classAttr == trainData[j].classAttr){
            fl = true;
            break;
         }
      }
      if (!fl){
         CLASSES++;
      }
   }

   dat.close();
}

int countLinesAttrs(char fileName[], int &lines, int &attrs){
   int i, c;

   ifstream dat;
   dat.open(fileName);

   if (!dat){
      return 1;
   }

   string lline;

   getline(dat, lline);
   c=0;
   for (i=0; i<lline.length(); i++){
      if (lline[i] == '\t'){
         c++;
      }
   }
   dat.close();
   attrs = c;

   dat.open(fileName);
   i=0;
   while (!dat.eof()){
```

```
        getline(dat, lline);
        i++;
    }

    lines = i-1;
    dat.close();
    return 0;
}
```