Utrecht University
Department of Information and Computing Sciences

Master Thesis

Distributed and Incremental Clustering using Shared Nearest Neighbours

Author: Cas Plattèl
c.j.plattel@students.uu.nl
ICA-3344908

July 2, 2014

Sentient Information Systems
Supervisor:
Bas Weitjens Msc.

Utrecht University
Supervisors:
Dr. Ad Feelders
Prof. Dr. Arno Siebes

Abstract

Cluster analysis is a large field inside data mining. It tries to capture the structure of a data set by grouping similar data points into clusters. Increasingly large and high dimensional data sets present a problem for modern clustering algorithms. Especially data sets containing nominal and sparse vector features. In this thesis I explore how cluster analysis can be applied best to these kind of data sets. Data from the Xenon web page crawler is used to test algorithms, which includes numeric, nominal and bag of words data. We study different algorithms and propose a new algorithm called DBSNN, which is based on Shared Nearest Neighbours and local densities. This algorithm is implemented and experimented with. In order to deal with the size of the data set the algorithm is made to work on a distributed system. The algorithm is robust against outliers and can detect noise, finds clusters independent of difference in shape, sizes or densities. In addition, the algorithm is extended to work on incremental data, meaning it can detect new clusters who represent upcoming trends in the data set.

Keywords: Clustering, Algorithms, Incremental, Distributed, Shared Nearest Neighbour

Acknowledgements

I would like to thank a number of people, without whom this master's thesis would not have been here.

Firstly I would like to thank Marten den Uyl and Pieter Bison for providing the possibility to do a research internship at Sentient Information Systems which resulted in this thesis. I would also like to thank all the colleagues at Sentient Machine Research group for their interest in my research and for the pleasant working atmosphere. I would like to thank Bas Weitjens especially for his help and feedback on my research, providing guidance and directions. The useful discussions with Vincent Hoekstra and Michel de Ruiter also contributed to my work, for which I am grateful.

I also want to thank my supervisor from Utrecht University, Dr. Ad Feelders, for his helpful advice during my internship and on this thesis. I also like to thank Prof. Dr. Arno Siebes for reviewing this thesis.

Lastly, I would like to thank my parents, my family and friends, and especially Nienke, for their continuing support and encouragement.

Thank you all so much,
Cas Plattèl

Contents

Chapter 1

Introduction

In the data mining field clustering is one of the larger subjects. It tries to divide data sets into meaningful groups (called clusters). These clusters should contain data which is more similar to each other than to data in the other clusters. This way the clustering partitions the data ,providing a way to explore the underlying structure of the data. Clustering as a technique is often used in data analysis and is applied in many fields, such as machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics [1].

Clustering is usually applied for two purposes; either for understanding or for utility. Understanding data can be seen as unsupervised classification, similar data points get grouped. For instance, finding objects in images, such as buildings, people, animals, etc. This derives useful information from the original data set. Clustering can also be done for utility, since clusters can be used as summarization of the data. This is useful, for example, for compression or algorithmic optimization, like finding Nearest Neighbours.

Often clustering is used to detect unknown patterns or coherence inside a data set. This clustering, however, depends on how clusters are perceived, since there are often multiple ways to look at data and its structure. In his paper [2], Estivill-Castro already mentions that the notion of a cluster cannot be defined precisely and thus, depending on which definition is used, different clusterings can be found. In this thesis I will elaborate more on the different definitions of clusters, clusterings and measures of similarity.

Nowadays data sets become increasingly larger (often referred to as Big Data) and are often high dimensional. These characteristics provide all kinds of problems, such as the feasibility of data analysis techniques and the problems of the *curse of dimensionality* [3]. Can these data sets still be processed and if so, can they somehow be made more insightful without having to check all data points. Clustering is one technique to apply on these data sets to partition the data, and in this thesis we will see how these problems are dealt with.

Chapter 2

Problem Statement

Currently there exists a large collection of clustering algorithms, each often specialised towards specific goals. Sentient Information Systems[1] develops data mining software, such as DataDetective[2], that can be used widely in society. Creating a clustering algorithm that can be used inside this application means the algorithm should be applicable to as many types of data sets as possible. An example data set that should be clustered by the algorithm, is Xenon *Drugs and Medicine* data. This data set is provided by ParaBotS[3]. It is obtained for the purposes of the TAFEIC (Tool Against Financial and Economic Internet Crime) project, which is an European project aimed at fighting various forms of internet fraud and crime. The data set contains websites that may be fraudulent. These sites might sell fake products in the fields of sport enhancing drugs, alternative medication and illegal online pharmacy, in short, drugs and medicine. By clustering this data, new kind of drugs or changes in known drugs can be detected, especially when this data builds up incrementally. This is useful for several government institutions, like the tax office or general health-care. Samples of the data set are shown in Table 6.1 in the experimental setup, chapter 6.

The data set is large, contains many dimensions, has diverse data types (categorical, bag of words) and increases over time (a continuous stream of data). Thus an algorithm has to be found that can handle this kind of data set. In other words, a clustering algorithm is needed that can cluster large, high dimensional data sets, containing any kind of data type and do this in an incremental fashion. An ever increasing data set will require a solution which can store a large amount of data in memory, and can compute over it. Because such an expanding data set requires this, the algorithm has to work in a distributed environment, where we can spread the calculation and storage over multiple machines.

[1] Sentient Information Systems, www.sentient.nl
[2] DataDetective, http://www.sentient.nl/?dden
[3] ParaBotS, http://www.parabots.nl/

2.1 Qualifications

The algorithm to be developed needs to follow certain requirements to fulfil the stated problem.

- High quality result
  This is hard to define, but we want to have a result where data points belong in the cluster they are assigned to, more than they would to any other cluster.
- Large data sets
  The algorithm has to be able to handle large data sets with reasonable performance, thus the algorithm might have to run on a distributed system.
- Incremental
  Data should be able to increase over time and get clustered.
- High dimensions
  The number of dimensions of the data can be high, and thus the algorithm should perform well on this, in addition to keeping meaningful results.
- Interpretable
  The result of the algorithm should be visualised so it can be interpreted.

Clustering large data sets will often lead to a long processing time because of the amount of calculations required. In order to resolve this issue the algorithm can be rewritten to one which can run in parallel. This way the limitation of processing power gets shifted from one machine to the number of machines working on the clustering.

To make the data interpretable, the data will have to be projected into the visual dimensions (two or three dimensions). Thus we will have to apply some form of dimension reduction.

2.2 Research Questions

Following the previous subsections the following research questions follow from the stated problem.

- How can we design a clustering algorithm that can cluster a large set of high-dimensional data incrementally?
- Can we implement this in a distributed way?

Additional sub questions can be raised, such as how do we define a good clustering, and can we prevent runtime complexity becoming too high. These questions also apply to how the algorithm can be implemented.

2.3 Related Work

Previous work at Sentient Information Systems in this field has been done by Alexander Mol, who in his thesis researched optimal parametrization for clustering algorithms [4] and by Marijn Lems, who used clustering as a technique to identify gangs of spammers by verifying if they share resources [5].

Concerning clustering algorithms in general, there are a lot of applications of cluster analysis in the real world, causing different clustering algorithms to emerge, each often tailored towards a specific problem (or a specific type of data set). Common algorithms that are based on prototypes and prefer globular data structures are K-Means [6] and Agglomerative Hierarchical [7] algorithms. Others are based on density, such as DBSCAN [8] or on distribution, such as mixture models (EM [9] for example). And there are the force-driven algorithms based upon graph drawing algorithms, like Fruchterman-Reingold [10], which instead of using all dimensions apply physics rules on a projection to cluster the data. In this thesis, K-Means will be explained in section 3.3.1, and a more advanced version K-Prototypes in section 4.1, DBSCAN will be described in detail in section 3.3.3.

For incremental data there have been extensions to the known algorithms, like incremental DBSCAN [11], but there are also entirely different algorithms, like k-windows [12] (which however, might give meaningless results [13]) or BIRCH [14], which is also a prototype-based algorithm but has a superior running time and works very well on incremental sets. However, it does not work on categorical data. An extension to BIRCH is the LIMBO algorithm [15], which does work on categorical data but has a worse running time if the data set contains a lot of this type of data. More information about LIMBO is given in section 4.2.

Because data sets start to get larger and larger, creating clustering algorithms that work distributed and in parallel become attractive since they provide large speed increases. These algorithms are sometimes based on well-known frameworks like MapReduce [16], or sometimes custom made adaptations of popular algorithms like Distributed DBSCAN [17].

There are a lot of optimizations for clustering, a large part of which unfortunately only work on ordinal (or spatial) data. Examples of such optimizations are kd-trees [18] or metric graphs [19], which speed up the computation for finding nearby points or even algorithms which only work on ordinal data sets, such as Best Bin First [20], which approximates nearest neighbours based on their distance to each other.

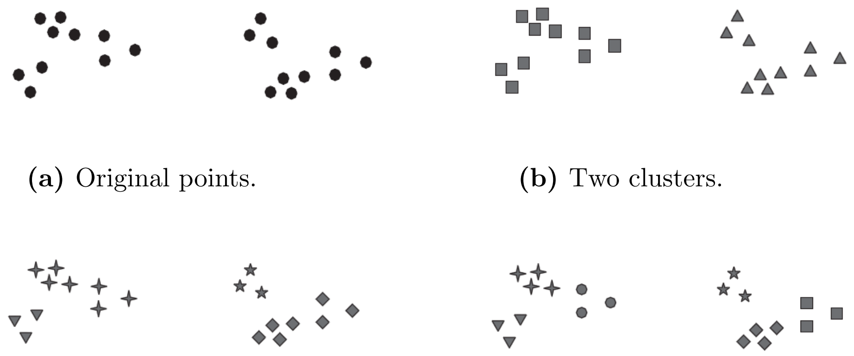
2.4 Outline

The following chapter, chapter 3, provides useful background about clustering in general and on the specific characteristics of this study, such as similarity measures, incremental clustering and distributed clustering. Chapter 4 explores algorithms taken in consideration as solution to this study's research questions. Chapter 5 explains the picked Density-based Shared Nearest Neighbours algorithm, how it works in detail on an initial data set, on incremental data, and how its distributed component works. The experimental setup is explained in chapter 6, and in chapter 7 we show results of the algorithm on the Xenon websites data set and expand upon the visualisation of the clustering. Summarization and conclusions are done in chapter 8 and discussion can be read in chapter 9. In the last chapter, chapter 10, we elaborate on future work that can be done on my research.

Chapter 3

Clustering

Clustering is the task of creating groups inside a data set where data points are more similar to the points inside the group (cluster) they are in, than to any other group. The definition of what the best clusters of a certain data set are, is not always clear. This is clarified in the example below (Figure 3.1). There are multiple clusterings possible, where each partitions the data properly; each clustering makes *sense*. This implies that when looking at a data set there are multiple ways to cluster it, which is best depends on what type of clustering is preferred to be found in the data. This problem is commonly handled in algorithms by requiring parameters, representing characteristics of the algorithm. For instance, in K-Means clustering the number of clusters K is required and for DBSCAN the minimum density MinPts has to be supplied. This problem can never be elegantly solved since this is intrinsic to clustering [1].



(a) Original points.

(b) Two clusters.

(c) Four clusters.

(d) Six clusters.

Figure 3.1: Different ways of clustering the same set of points [1].

In the following sections general information is provided concerning clustering, definitions of clusters and clusterings, common algorithms and their techniques, measurements used for clustering, incremental clustering specifics and information about distributed clustering and merging.

3.1 Notation and Definitions

In this section notation and definitions are given for the basic concepts used throughout this thesis.

Let $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ be a set of data vectors. Each $\mathbf{x}_i$ vector exists in a d dimensional vector space. We thus have a $n \times d$ matrix consisting of n records with d attributes. This will be the input data set for the clustering algorithm.

Let the cluster result be K groups, each consisting of a subset of X.

In the distributed system, let $M = \{machine_1, machine_2, \ldots, machine_m\}$ be the set of machines performing the clustering task, where each machine has at most φ data points of the input set X.

For canopy algorithm, c represents the amount of canopies found, and f is the sample size. In the K-Means and K-Prototypes algorithm, K denotes the number of clusters, b is the batch size, and I is the maximum number of iterations.

3.2 Clustering

As pointed out, there is no real definition for what clusters are. Instead there are multiple ones, dependent on the data set and on the perspective of the interpreter. Below is a short summary of types of cluster definitions.

Well-separated: A cluster is a set of objects in which each object is closer to every other object in its cluster than any other object outside of the cluster. This is a very strict definition, which makes finding proper clusters difficult.

Center-based: Every object inside a cluster is closer to the cluster centroid than to any other centroid of any other cluster. This assumes that the data set has an average, or center point, and it is *meaningful*, which is often the case for numeric data, but often not for nominal. For nominal data the mode of the set can be picked as center value.

Contiguity-based: Every point in a cluster is closer to at least one point in the cluster than to any point outside of its cluster. This is a less strict definition, which makes it possible for clusters to have non-globular shapes.

Density-based: Separation of clusters is dependent on the density of the clusters. High density areas are commonly clusters while low density areas are often just noise.

Distribution-based: If we can make assumptions about the distribution before clustering we can improve on clusterings. We can for instance assume a certain Gaussian distribution. This is however less applicative for this project, because a probability distribution requires a lot of domain knowledge, which might be lacking.

Subspace models: We divide the work over partitions of the attributes, e.g. we just cluster on the pairwise combinations of each two dimensions. These clusters then get merged to a final result.

In order to have an algorithm that works on most types of data sets, we pick the definition which has the least limiting assumptions. This is preferred, since the tool in which this algorithm is to be used can be applied on any kind of application. Thus we choose contiguity-based clusters together with a density definition. This means we want to find clusters independent of shape and size. We can also deal with non-globular structures, which are clusters not spread out as a circle around a center, but in any arbitrary form, because of clustering on density.

Besides choosing the cluster definition, a related choice has to be made on clustering types. There are multiple ways on how we can assign data points to the clusters.

Hierarchical (nested or connectivity-based): We start out with a clustering in which each data point is its own cluster. We merge the clusters which are most related first, and each step merge towards one cluster which has all the data points. The clustering is now built in a hierarchical way, where for any number of clusters needed there is a solution. An alternative is to do this top-down instead of bottom-up.

Exclusive clustering: Any data point in the clustering can only be assigned to one cluster.

Overlapping clustering: A data point may appear in more than one cluster, meaning it has a high similarity (or short distance) to multiple clusters.

Fuzzy (probabilistic) clustering: The clustering is fuzzy, meaning for every data point we have a value between zero and one which represents how well the point fits the cluster. A point can also have multiple fuzzy values, one for each cluster in the clustering.

Complete clustering: The total data set is clustered; every point belongs to a cluster.

Partial clustering: Not every data point has to be in a cluster, thus we can have noise points.

Although fuzzy clustering has its advantages, it is decided to pick partial exclusive clustering as a result of the algorithm, since this gives a strict division of the data points into clusters and noise. Distinct clusters provide clear results and noise detection is beneficial if clustering happens in a distributed manner, since we can then detect noise and send it to a machine with better matching clusterings.

3.2.1 High Dimensions

A common problem when clustering (and in data analysis in general) is the so called *curse of dimensionality* [3], which is commonly described as the problems which occur when data has a high number of dimensions. Such problems are for instance that distances become indistinguishable and lose their usefulness, and that nearest and farthest neighbours become more and more similar. Bernecker *et al.* [21] point out that, although these problems hold for some data sets, the problem is often caused by the increase of irrelevant dimensions, which make the analysis problem harder. We can see this as a change in the ratio of useful features to noise features for the later, resulting in a loss of contrast in the data [22].

Thus we can state that for high dimensional data, adding a lot of irrelevant dimensions, the problem becomes increasingly difficult. To solve this problem for clustering, a number of solutions have been proposed, such as subspace projection [23] or shared nearest neighbour similarity [22]. By applying clustering on subspaces, clusters can still be found, however, it is computationally heavy, since all pairwise combinations of dimensions are clustered, which is a search space of the size 2^d. These type of high dimension clustering algorithms focus on just finding clusters in axis-parallel subspaces. An alternative is using Shared Nearest Neighbours, where clusters are found by finding neighbourhood of similar data points. Shared Nearest Neighbours has been shown to work well on high dimensions by Houle *et al.* [22], performing superior to common measures such as Euclidean distance. Tan *et al.* [1] also show that a variation of SNN using densities performs very well on high dimensional time series data, such as a data set of atmospheric pressure on Earth.

The choice between these two methods comes down to characteristics in the data set, if there are only a few distinguishing dimensions, finding these with subspace projection algorithms is more effective. However, if the data contains a large number of relevant dimensions SNN has been shown to be better [22].

3.2.2 Dimension Reduction for Visualisation

The data will have to be projected into the visual dimensions so the clustering result is interpretable. This means some form of dimension reduction has to be applied. This will lead to a lack of triangle inequality between the represented data points, meaning if point A and point B are both near point C (since they are in the same cluster and are similar to C) they might differ from each other more than is implied by their distance. This is a problem which is unsolvable, because we will always need a dimension reduction to convert to an observable distance relation between data points.

Clustering high dimensional data is a complex task. Instead of clustering algorithms specifically designed for high-dimensions, an alternative is reducing the number of dimensions using a dimension reduction technique. After the reduction any kind of clustering algorithm can be applied to find the clusters in the data. However, often a lot of characteristics of the data are lost by applying a dimension reduction.

Examples of dimensionality reduction are Principle Component Analysis [24, 25], Singular Value Decomposition [1] and FastMap [26]. Most of these techniques have a running time of $\mathcal{O}(n^2)$, which, for large data sets, is quite long just for reducing dimensions as subroutine. In addition, PCA and SVD have the assumption that there has to be a linear relation between old and new sets of attributes [1]. FastMap is an example of a linear running time dimensionality reduction technique. It is based of MDS (multidimensional scaling), but it requires that for the data triangle inequality holds, which is not the case when looking at similarity between data points containing nominal attributes[1].

An alternative way to deal with the high dimensionality problem is by using a clustering algorithm which can deal with high dimensionality on its own, such as shared nearest neighbours [22]. We can just apply the clustering algorithm on all dimensions, thus not losing any kind of information, and then visualise by applying dimension reduction only on the results. Dimensionality reduction is in this approach only needed for visualisation, which has inherent problems anyway as explained in the previous subsection.

3.3 Common Algorithms

In order to go more in-depth about clustering algorithms the following subsections each explore an algorithm commonly used in clustering. Some of these will be used as subroutines in the final algorithm.

3.3.1 K-Means Clustering

K-Means is one of the first and most basic clustering algorithm described in the literature. The algorithm iteratively finds local optimal centroid values. It starts with K randomly picked locations (called centroids) in the data range, then it starts iterating. For each iteration it assigns all data points to the closest centroid value, then it recalculates the centroid value to the center value of the assigned data points of

[1] For more information about triangle inequality on nominal data, see section 3.4.

that centroid. This causes the centroid values to shift every iteration towards the local optimal center. After a number of iterations the algorithm has converged or is stopped. The points all get assigned to the closest centroid and represent the K found clusters. A few steps of the algorithm are displayed in Figure 3.2.

The running time of the algorithm is $\mathcal{O}(n \cdot K)$ per iteration.

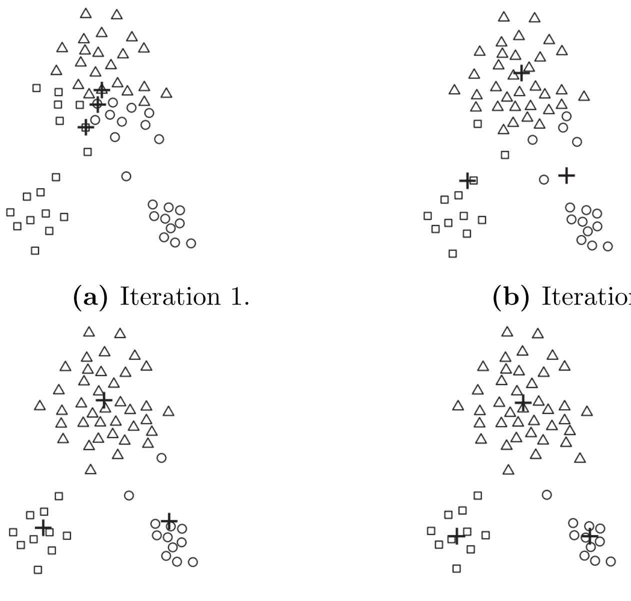

(a) Iteration 1. **(b)** Iteration 2. **(c)** Iteration 3. **(d)** Iteration 4.

Figure 3.2: Using the K-means algorithm to find three clusters in sample data [1].

3.3.2 Canopy Clustering

Canopy clustering is often used as a pre-clustering step instead of actual clustering. The idea is to make a rough partitioning of the data and then further cluster inside these so called canopies, reducing computation time [27].

The algorithm starts by picking a random data point of the data set. We pick two thresholds for which $threshold_2 \leq threshold_1$ has to hold. All data points inside a $threshold_1$ range of this data point get assigned to its canopy. All data points which are even closer (or more similar) than the $threshold_2$ range also get assigned to the canopy but are, together with the selected data point, removed from the data set. We then continue by again at random picking a new data point, selecting canopy members and removing very similar points. We end up with canopies containing roughly similar data points. Data points can also be in multiple canopies, which does not have to be a problem, depending on the follow-up algorithm. The canopies serve as input for this algorithm, which often only considers points inside the same canopy as potential same-cluster points. An example of how canopies are selected can be seen in Figure 3.3.

The running time of the algorithm largely depends on the threshold values but is often negligible compared to any clustering algorithm following it.

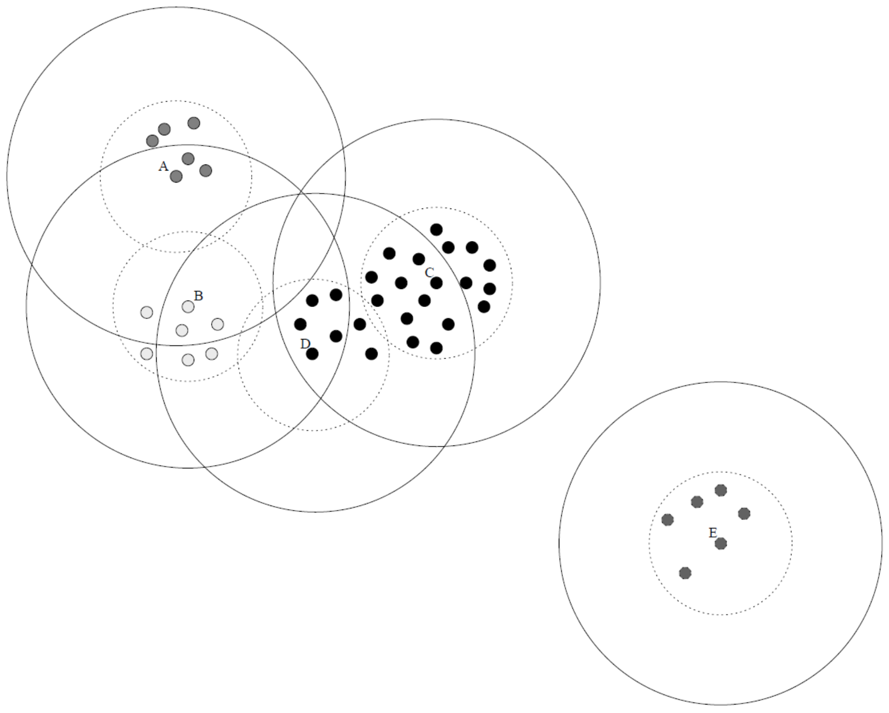


Figure 3.3: Four data clusters and the canopies that cover them [27].

3.3.3 DBSCAN

DBSCAN is a density-based clustering algorithm which separates clusters based on their local densities [8]. For each point we determine how many points are inside a given radius *Eps*, if this number exceeds given parameter *MinPts* we call the point a *core* point. For all points which are non-*core* points we determine for each if it has a *core* point inside its *Eps* radius, if so it is called a *border* point, else it is discarded as *noise* point. We group *core* and *border* points which are in each others *Eps* inside the same cluster. This way clusters are formed depending on density. In Figure 3.4 assignment of *core*, *border* and *noise* points is shown.

The running time of this algorithm is $\mathcal{O}(n^2)$ because we need to calculate all pair-wise similarities for all the data points.

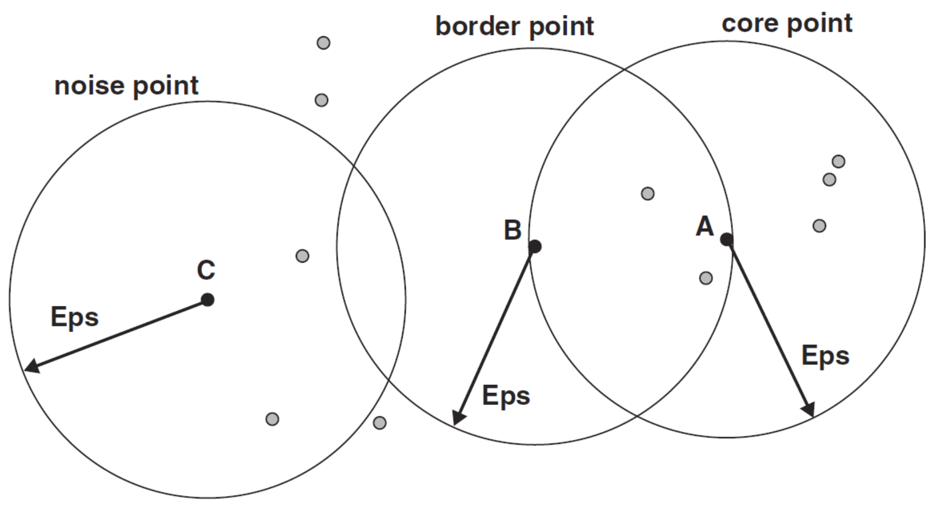


Figure 3.4: Core, border and noise points [1].

3.3.4 Shared Nearest Neighbours

Shared Nearest Neighbours is a technique first described in detail by Jarvis and Patrick [28]. It serves as a second order similarity measurement. Before we apply it, we calculate for each data point its k-nearest neighbours. These are the k data points which have the highest similarity or shortest distance over all dimensions to the data point. The similarity calculation, or distance metric used is the first order measurement. The result, a matrix of $n \times k$, is then used as input for the SNN calculation. For each data point we compare it with each of its k best neighbours and now set their similarity by the number of neighbours they share divided by k, see figure 3.5. In equations this will look like;

$$SNN_k(\mathbf{x}, \mathbf{y}) = \begin{cases} |NN_k(\mathbf{x}) \cap NN_k(\mathbf{y})|, & \text{if } \mathbf{x} \in NN_k(\mathbf{y}) \text{ and } \mathbf{y} \in NN_k(\mathbf{x}) \\ 0, & \text{otherwise} \end{cases}$$

$$similarity_k(\mathbf{x}, \mathbf{y}) = \frac{SNN_k(\mathbf{x}, \mathbf{y})}{k}$$

Converting a k-nearest neighbour graph to one based upon shared nearest neighbours is sometimes called sparsification. The edges between clusters get removed because they share no neighbours, see figure 3.6.

Two points only get a positive shared nearest neighbour similarity if both of them have each other in their nearest neighbour list. This causes clusters to be found dependent on local densities. Points on the border of a high density area might have points of that area in their nearest neighbours list, but the points in the actual high density area are always nearer to their other high density points, thus they do not have the border points in their nearest neighbour list. These points will not get connected in the SNN graph, causing separation of clusters depending on density shifts.

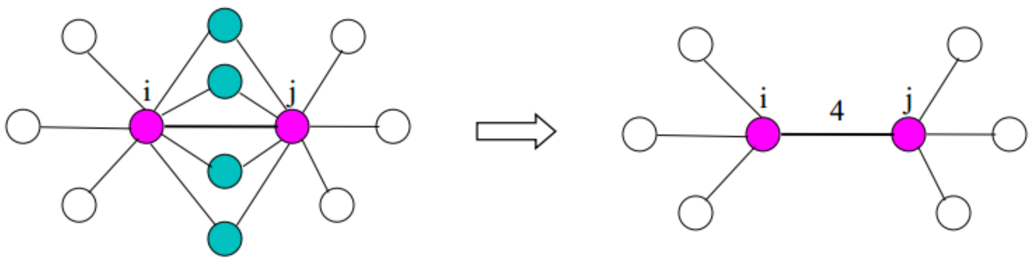
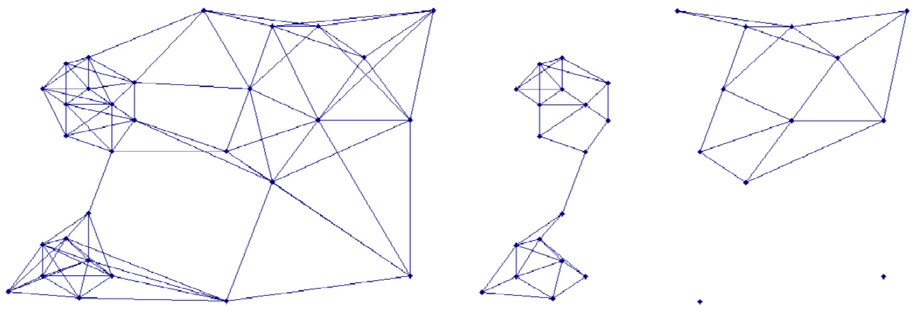


Figure 3.5: Computation of SNN similarity between two points [1].

In order to find the k-nearest neighbours we have to calculate all pair-wise similarities, which costs $\mathcal{O}(n^2)$ time. Additional time is spend on calculating the intersection between the nearest neighbour lists per data point which takes $\mathcal{O}(k^2)$ when unsorted.



(a) K-Nearest Neighbours graph. **(b)** Shared Nearest Neighbours graph.

Figure 3.6: Sparsifying k-nearest neighbours by applying shared nearest neighbours [29].

3.4 Similarity Measures

For clustering we need to have some sort of measure that can determine which data points should be in the same cluster and which data points should not be. We can look at this in two ways; either we take distance as measure, and define a cluster as a group of points which are all *close* to each other, or we can look at how similar the data is by using a similarity measure, and put similar data points in a cluster.

For distances, or metrics, the following well-known properties have to hold:

$$\begin{aligned}\textbf{Positivity} &: d(\mathbf{x},\mathbf{y}) \geq 0 \text{ for all } \mathbf{x} \text{ and } \mathbf{y}.\\ &\quad d(\mathbf{x},\mathbf{y}) = 0 \text{ only if } \mathbf{x} = \mathbf{y}.\\ \textbf{Symmetry} &: d(\mathbf{x},\mathbf{y}) = d(\mathbf{y},\mathbf{x}) \text{ for all } \mathbf{x} \text{ and } \mathbf{y}.\\ \textbf{Triangle Inequality} &: d(\mathbf{x},\mathbf{z}) \leq d(\mathbf{x},\mathbf{y}) + d(\mathbf{y},\mathbf{z}) \text{ for all points } \mathbf{x},\ \mathbf{y}, \text{ and } \mathbf{z}.\end{aligned}$$

Distance measures are based on the distance between data points over all dimensions of the points. This is quite intuitive; points which have a low distance between them need to have values in their dimensions that are very close to each other, and thus the points are very similar. There are a couple of distances measures (or better worded; metrics) to choose from;

- Manhattan distance
  The Manhattan distance is also known as L_1 distance, or taxicab geometry. The distance between two points is the sum over the absolute difference per coordinate (per dimension).
  $$d(\mathbf{x}_a,\mathbf{x}_b) = \sum_{i=1}^{d} |\mathrm{x}_a^i - \mathrm{x}_b^i|$$
- Euclidean distance
  The Euclidean distance is also known as the L_2 distance, it is the common way to measure distance between points. It follows the Pythagorean formula. Generalized for n dimensions, the formula becomes;
  $$d(\mathbf{x}_a,\mathbf{x}_b) = \sqrt{(\mathrm{x}_a^1 - \mathrm{x}_b^1)^2 + (\mathrm{x}_a^2 - \mathrm{x}_b^2)^2 + ... + (\mathrm{x}_a^d - \mathrm{x}_b^d)^2}$$
- Minkowski distance
  Minkowski distance is a generalization of both Manhattan and Euclidean distance, $p = 1$ for Manhattan, and $p = 2$ for Euclidean.
  $$d(\mathbf{x}_a,\mathbf{x}_b) = \left(\sum_{i=1}^{d} |\mathrm{x}_a^i - \mathrm{x}_b^i|^p\right)^{\frac{1}{p}}$$

A different approach is calculating some form of similarity between the data points. This makes more sense when we have data which has non-metric attributes. For instance, persons who have hobbies; calculating a distance between two persons based on these values seems nonsensical, since there is no ordering in the possible categories (or even combinations) of the hobbies. Is football closer of further from hockey than tennis is? How do we use the distance measure between these values?

A better measure here is a similarity measure, which can be seen as the opposite of a distance metrics. Highly similar data has a low distance and very dissimilar data has a very large distance. The difference between distance and similarity measures is that for similarity measures the triangle inequality typically does not hold. In addition, sometimes for nominal attributes symmetry also does not hold. An example would again be hobbies, someone who plays football, hockey and tennis matches someone who only plays tennis by just one-third of his hobbies, but the person who plays only tennis matches for all its hobbies to the other person.

There are quite some similarity measures to choose from;

For the first two measures some additional definitions are needed. If we have two vectors which have a nominal attribute (can be complex, like bag of words), we can represent this as a binary vector. With this we can define;

A_{11} = total number of binary values where both vectors have the value 1.
A_{01} = total number of binary values where first vector has value 1, other has value 0.
A_{10} = total number of binary values where first vector has value 0, other has value 1.
A_{00} = total number of binary values where both vectors have the value 0.

- Jaccard similarity
  Jaccard similarity measures the similarity between two nominal attributes by taking the intersection of both and divide it by their union. In terms of the above definitions this gives;
  $$J = \frac{A_{11}}{A_{01} + A_{10} + A_{11}}$$
- Jaccard distance
  The Jaccard distance is not a typical distance measure. It measures the *dis*similarity between two sets, and thus, it is the complement of the Jaccard similarity. It can be calculated by subtracting the Jaccard similarity from one, or by subtracting the intersection from the union before division.
  $$d_J = \frac{A_{01} + A_{10}}{A_{01} + A_{10} + A_{11}}$$
- Cosine similarity:
  Cosine similarity measures the similarity between two vectors by taking the cosine of the angle the two vectors make in their dot product space. If the angle is zero, their similarity is one, the larger the angle is, the smaller their similarity. The measure is independent of vector length (the two vectors can even be of different length), which makes it a commonly used measure for high-dimensional spaces.

$$sim(\mathbf{x}_a, \mathbf{x}_b) = cos(\theta) = \frac{\mathbf{x}_a \cdot \mathbf{x}_b}{\|\mathbf{x}_a\| \|\mathbf{x}_b\|} = \frac{\sum_{i=1}^{d} \mathrm{x}_a^i \times \mathrm{x}_b^i}{\sqrt{\sum_{i=1}^{d} (\mathrm{x}_a^i)^2} \times \sqrt{\sum_{i=1}^{d} (\mathrm{x}_b^i)^2}}$$

To elaborate a bit more on the formula; we calculate the inner dot product $\mathbf{x}_a \cdot \mathbf{x}_b$ by multiplying for each dimension the value of $\mathbf{x}_a$ with the value of $\mathbf{x}_b$, this results in summing the number of dimensions where $\mathbf{x}_a$ and $\mathbf{x}_b$ match (are equal). We divide this by the size of the vector space $\|\mathbf{x}_a\|\|\mathbf{x}_b\|$.

For the clustering algorithm we have chosen to use a combination of measures. For nominal data the cosine similarity is used since it performs well on sparse vectors, such as bag of words, since it ignores 0-0 matches. Since it is independent of vector length it works quite good on high dimensional data [30]. For metric data the Euclidean distance is used, it is calculated between two points and divided by the range of the values. This creates a little error rate since we do not know for sure if our lower and upper bound of the metric range are the real bounds, but without external knowledge there is no other way to solve this.

3.4.1 Combining Measurements

For each attribute of the data set the corresponding measure is applied, for nominal data the cosine similarity, for metric data the Euclidean distance. We want these measures to have an equal range, so we can combine them. This means that for for Euclidean distance we have to divide the distance by the range of the metric data. This gives a value between zero and one. From the similarities per attribute we calculate the average which represents the match of the two data points. More formal;

$$similarity(\mathbf{x}_a, \mathbf{x}_b) = \frac{1}{d}\sum_{i=1}^{d} \omega_i \delta_i(\mathrm{x}_a^i, \mathrm{x}_b^i)$$

$$\delta_i(\mathrm{x}_a^i, \mathrm{x}_b^i) = \begin{cases} 1 - \dfrac{EuclideanDistance(\mathrm{x}_a^i, \mathrm{x}_b^i)}{range_i}, & \text{if attribute } i \text{ is numeric} \\ CosineSimiliarity(\mathrm{x}_a^i, \mathrm{x}_b^i), & \text{if attribute } i \text{ is nominal} \end{cases}$$

In the algorithm weights (ω_i) can be assigned per attribute which are used in the averaging calculation, a high weight will cause an attribute to contribute more to the total match. This is particularly useful for bag of words data, where one attribute may contain a lot of hidden dimensions (like words in the Xenon data). We want the words of a website to contribute more to the total match than for instance the country of the host of the site. With the weights this problem gets addressed.

3.5 Incremental Clustering

Incremental clustering is a powerful concept which makes it possible to summarize data on the fly. This makes it easier to spot developing trends inside a changing data set that builds up over time. Sometimes this is called *dynamic* clustering, since the clusters dynamically adapt to the incoming data. Incremental clustering invalidates assumptions made on the data set, like number of clusters or densities. It also brings the problem of choosing between clustering noise and outliers or leaving them out of the clustering.

There have been proposed many incremental clustering algorithms in literature, some to cluster a continuous stream of data (as we would like to), and some as a performance improvement. In the first case, the data stream is getting clustered, often using some kind of time window. An example of such an algorithm is k-windows algorithm [12]. In the second case, a subset of the data is clustered and the remainder is done incrementally, speeding up the process. An example of this is IncrementalDBSCAN [11]. Another algorithm often named in literature is BIRCH [14], which is a tree based incremental clustering algorithm.

A common concept in data stream clustering algorithms is to make use of *micro* and *macro* clusterings [31, 32]. The *micro* clustering is the online clustering, in which incremental data gets best fitted into a cluster (which might potentially be a new cluster). Either a *micro* cluster already exists because it has sufficient relevant data points (inside a reasonable time window), or it is a potential cluster, still waiting for enough new data points to join. The *macro* clustering is the offline clustering, this is the output generated when a user requests a clustering from the algorithm.

3.6 Distributed Clustering

Since clustering is often a computationally heavy task, and it is more and more used on very large data sets, distribution and parallelization are often employed techniques to improve running time. Some cluster algorithms are easier to distribute than others. An example of distributing is MapReduce clustering using K-Means [16]. Every machine starts a run of K-Means with different initial centroid points. After each machine has clustered the data set, the best clustering is picked depending on some quality measure, like mean squared error or silhouette coefficient. A different example is the k-windows algorithm [12]. This algorithm uses a sliding window technique, it applies this technique per machine to a subset of the total data set. The resulting windows (clusters) are considered for merging if they overlap in any way.

When using distributed techniques the limiting factor of the algorithm gets moved from CPU time or memory size of one machine to the number of machines that are connected to the task. This makes a distributed algorithm more scalable and thus more suitable for large data sets (or even incremental data sets). There are, however, some disadvantages to distributed computing that need to be considered. In his short paper, Rotem-Gal-Oz [33] mentions common fallacies that are assumed concerning distributed computing.

- Synchronisation:
  The state of the calculation has to be equal over all machines to avoid double

computations or getting false solutions because of not yet updated information. Things to think about here are consensus over machines, deadlocks and synchronised methods.

- Failure:
  How can we best handle errors in corrupt data, failed machines, lost messages, etc?
- Stabilisation:
  There has to be termination, there are situations where messages might keep getting sent or where no decision is made and no termination will ever occur.
- Fallacies:
  Common fallacies of distributed programming by making wrong assumptions [33]:
  - Network is reliable
  - Latency is zero
  - Bandwidth is infinite
  - Network is secure
  - Topology does not change
  - There is one administrator
  - Transport cost is zero
  - Network is homogeneous

These are things to consider before and while using a distributed solution.

3.6.1 Merging

When using a distributed solution, clustering results often need to be merged into a final result. This is the case when machines work on a subset of the entire data set. The found clusters might overlap or even be almost identical. To solve this issue, the clusters need to be merged. This can be done in multiple ways. One is by applying a distance or similarity measure between the centroids of the clusters, and, if they are similar, merge them. Another would be to apply the measure on data points of the clusters and see if the data points are more or less similar, and if so, to merge the clusters.

Chapter 4

Candidate Algorithms

For this research a comparison is made between clustering algorithms to determine which one provides the best solution to the problem. Most common algorithms have been looked at and an eventual selection was made. In this section a number of candidate algorithms are described. We start with Mini-batch K-Prototypes, which is a fast algorithm based upon K-Means, then LIMBO, which is an algorithm which uses any kind of clustering algorithm as subroutine and provides a tree structure which speeds up the entire process, and we present a density-based Shared Nearest Neighbour algorithm, which calculates groups of neighbours sharing nearby neighbours.

4.1 Mini-batch K-Prototypes

The first candidate algorithm is the Mini-batch K-Prototypes algorithm. This is an adaptation of the original and well-known K-Means algorithm by using prototypes instead of means, as shown by Huang [34]. In addition, instead of calculating new centroids based upon all data points, we only use a batch each iteration and gradient step so computation and convergence is quicker, this is shown in a paper by D. Sculley [35]. Instead of a running time of $\mathcal{O}(n \cdot K \cdot I)$ for the classic K-Means algorithm, this algorithm has a running time of $\mathcal{O}(b \cdot K \cdot I + n \cdot K)$ because each iteration only b data points are used. The algorithm is described below and in more (technical) detail in Algorithm 1.

Input: n data points, d dimensions, K number of clusters, b batch size
Output: clustered data

1. Load all data inside memory.
2. Pick center points (c) by picking K points at random from the n data points.
3. Iterate until the algorithm stabilizes or we exceed the max number of iterations.
    (a) Pick b examples at random from the total data set.
    (b) For each of these samples assign it to the nearest center point c.
    (c) Recompute the center points by taking the average of these samples.

   (d) If the new center is little different from the old one for all center points c, stop iterations.

4. Return the clusters.

As said, a difference between this algorithm and K-Means is the representation of center points. For nominal data there is no ordinal mean, thus an alternative has to be used. A simple solution could be using the mode, like in the K-Modes algorithm, but instead in K-Prototypes a histogram is used. This histogram represents how often each category is counted in the samples. With this histogram we can match how well a certain data point matches the centroid (more than an other centroid).

The algorithm can be made even faster by instead of taking the new average as center point, using gradient stepping (explained in a paper by D. Sculley [35]). This will only shift the new mean slightly each time, making it more robust against outliers and poor sampling. Even without this optimization, the algorithm generally performs so fast that no distribution of the algorithm is needed.

Algorithm 1 Mini-batch K-Prototypes algorithm

```
procedure MiniBatchKPrototypes(data X, size n, batchsize b, number of clusters K, max iterations I)
    Initialize each c ∈ C with an x picked randomly from X
    change = true
    while change and iterations < I do
        change = false
        M ← b random samples from X
        for x ∈ M do
            for y ∈ C do
                d[x] ← max(d[x], similarity(M[x], c[y]))
            end for
        end for
        for y ∈ C do
            c[y] ← RecomputeCentroid(c[y], d)
        end for
        if similarity(c[y], old_c[y]) < threshold then
            change = true
        end if
    end while
    return C
end procedure
```

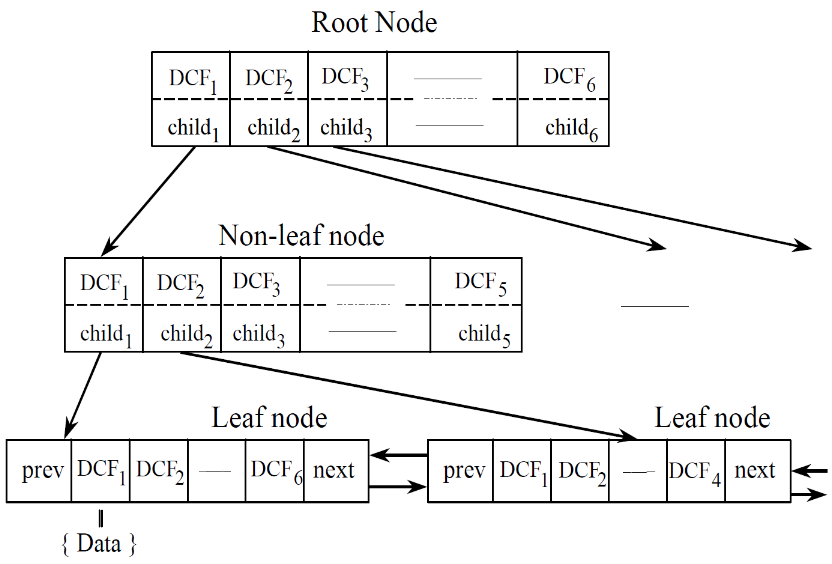


Figure 4.1: A DCF Tree with branching factor 6 [36].

4.2 LIMBO

LIMBO [15] is an algorithm that is actually a preprocessing step, much like the canopies technique from section 3.3.2. The concept is to build a tree in one pass over the data and group similar points into the same leaf. The resulting leaves then get clustered in a secondary algorithm, K-Means for example. The LIMBO algorithm is based upon the BIRCH algorithm [14], which is a popular algorithm for large data sets. The difference is that BIRCH cannot cluster nominal data since it uses centroid calculations which only work on metric data. LIMBO is an extension to BIRCH and works on nominal data by keeping track of the distribution of the nominal data inside the centroids. How this exactly works is described below and more in depth in Algorithm 2, and of course in the original paper of Andritsos *et al.* [15].

Input: n data points, d dimensions, B branching factor
Output: clustered data, can be merged, but, since it is incremental, we might have yet to decide this.

1. Load all data inside the memory by building a DCF-tree, this tree represents a summary of the data (see Figure 4.1).
   (a) The DCF-tree is build by adding each point one by one into the tree.
   (b) For each point go through the tree towards the leaf-node which has the best similarity match.
   (c) If the similarity is high enough it gets absorbed by one of the leaves in the leaf node, else it becomes a new leaf (equal to the CF-tree [14]).

   (d) Store averages of leaves and propagate to parent nodes. Also save nominal attributes inside the leaves by making a distribution of their occurrences in the leaf. Propagate this upwards.
   (e) For nominal attributes which can have multiple values (such as word lists), save either all items as attributes or as a random sample (this is something to consider for performance).

2. Reduce the tree (optional) in the same way as is done in BIRCH.
3. Apply a clustering algorithm upon all leaf nodes. This can be any type of clustering, since the number of leaves is generally low, calculating a similarity matrix is done fast.
4. The clusters found at the previous step are used as centroids, each point can now be assigned to their nearest centroid.
5. Incremental data can be easily dealt with by using the DCF-tree. It is easy to determine quickly if a point fits a cluster which already exists, or might form a new cluster.

This algorithm can handle incremental data extremely well, since the build up is also done incrementally. Time and quality performance are, however, heavily dependent on the follow-up clustering algorithm.

Algorithm 2 LIMBO algorithm

```
procedure LIMBO(data X, size n, branchsize B, clusters K, leafsize L)
    tree ← new DCF-Tree
    for x ∈ X do
        Insert(tree, x, b, L)
        if size(tree) > memoryThreshold then
            tree ← RebuildTree(tree)
        end if
    end for
    (Optional) GroupLeaves(tree)                 // If leaves overlap, we can merge
    (Optional) RemoveOutliers(tree)
    K-Prototypes(Leaves(tree))                   // Or any other clustering algorithm
    (Optional) Assign outliers to found clusters
end procedure

procedure INSERT(DCF-Tree tree, datapoint x, branchsize b, leafsize L)
    if tree.isLeaf then
        for l ∈ tree.node.children do
            // If leaf matches and has free space, put the data inside
            if similarity(l, x) > threshold and l.size < L then
                l ← l + x
            else if No leaf matches then
                Remove(x)                                        // Outlier point
            else
                Split(l, x)                      // We split the leaf with highest match
            end if
        end for
    else
        for c ∈ tree.node.children do
            bestNode ← similarity(c, x)                  // Save highest similarity match
        end for
        Insert(bestNode, x, b, L)
    end if
end procedure
```

4.3 Density-based Shared Nearest Neighbour

The concept of shared nearest neighbours as clustering technique was first used by Jarvis and Patrick [28] as said earlier in section 3.3.4. Density-based Shared Nearest Neighbour is an extension on this, described by Ertöz *et al.* [29, 37]. More recently, there have even been incremental versions of the algorithm, described by Singh and Awekar [38] and more in depth by Mendes *et al.* [39].

The basic algorithm starts by computing the k-nearest neighbours for each point and converting this into a shared nearest neighbours graph, much like the original Jarvis-Patrick algorithm. This is then used as input into DBSCAN (which is explained in section 3.3.3). The SNN part detects strongly connected groups inside the data set, even on high dimensions [1]. By applying DBSCAN on its output the algorithm detects the core points in the data instead of putting a flat threshold on the SNN value like Jarvis and Patrick did. This way the algorithm is less brittle to density shifts, outliers and noise. The algorithm automatically detects the number of clusters, and can discard the noise points.

The DBSNN (density-based Shared Nearest Neighbour) algorithm is extended in this research by adding an incremental component and distributing the clustering, in addition to minor optimizations and tweaks (see the following chapter). Below the algorithm is described, and in Algorithm 3 explained in detail.

Input: n data points, d dimensions, k nearest neighbour value, m machines, Eps radius, $MinPts$ threshold
Output: clustered data

1. An elected master machine M builds a distribution structure from the input data. For a small number of canopies or machines an index will suffice, else a decision tree is more appropriate.
   (a) A sample is taken to calculate the radius thresholds and then used to create canopies, as shown in section 3.3.2.
2. The leaves of the decision tree (or centroids of the index) are linked to the m machines.
   (a) We determine how many leaves are linked per machine. This is done as best as possible so the data load is balanced about equally over the machines.
   (b) For every machine the number of points does not exceed the threshold φ.
3. The data points are distributed over the machines using the decision tree.
4. For each machine their data points are clustered as described in the SNN algorithm.
   (a) This works as shown in section 3.3.4. In short:
      i. Calculate k-nearest neighbours of all data points on the machine.
      ii. Construct the shared nearest-neighbour graph.
      iii. Run DBSCAN over the SNN graph.
   (b) Noise points get passed on to the master machine M. It decides where to send them based on the decision structure, it picks the highest matching machine excluding the machines the point already has seen.

   (c) Every certain time period we can refine the decision tree of M by using the clusters found on all machines. This can be done together with the merge step, which is applied when visualising the clustering.

5. Incremental data points get distributed using the decision tree, refining the tree continuously solves the issue of inaccuracies. In addition, new clusters get detected at the tree level, so bursts of incoming data get assigned to one machine quickly if they are all very similar.

   (a) If we hit the threshold φ for maximum number of data on one machine, new data points get redirected to the machine which is next in line in similarity. The inaccuracy created will be solved with the final merging.
   (b) (Optional) Incremental input can cause too large clusters to form on machines or can cause clusters on separate machines to converge towards each other. The algorithm can be extended to that data gets split or merged after refining the decision tree.

6. When the user wants to see a final result, master machine M requests all clusters from the machines.
7. The received cluster centroids are used by M to determine the final clusters. We might want to merge overlapping clusters, we do this by calculating the centroids of all the clusters in the clusterings done on each machine. We compare the centroids with each other and if centroids exceed a certain threshold of similarity between the clusters we can merge them.
8. The result is shown on screen using the similarities between the cluster centroids as distance measures.

There is no communication between machines, only with the master. Because of this, we can easily scale up the number of machines, for instance when using a cloud computing approach. This does create a potential bottleneck for this algorithm, since the master machine might have to deal with an overload of incremental data (or a lot of noise). It might be too much to be processed in time, to solve this we can also distribute the master machine into multiple machines.

Four parameters have to be determined for this algorithm. For the radius thresholds of the canopies we sample the data. The *Eps* radius and the *MinPts* threshold for cluster creation are both correlated with the k value in the k-nearest neighbours calculation. So only k has to be determined manually, which might be determined by some sampling techniques or heuristics.

Algorithm 3 Density-based SNN algorithm

```
1: procedure DensitySNN(data X, size n, nearest-neighbours k, machines M, radius
   Eps, threshold MinPts)
2:     Y ← random sample of X
3:     T1 ← averageSimilarity(Y)                     // Using a similarity matrix on Y
4:     T2 ← averageSimilarity(Y) + $\sigma$(Y)
5:     c ← Canopies(X, T1, T2)
6:     tree ← BuildDecisionTree(c)
7:     tree.distribution(X, m)
8:     for m ∈ M do
9:         NN-List ← KNearestNeighbours($\varphi$, k)              // Costs $\varphi^2 \cdot d$
10:        for x ∈ NN-List do                                // From 1 till $\varphi$
11:            for y ∈ NN-List do                            // From 1 till K
12:                similarities ← SNNSimilarity(x, y, NN-List)   // Costs $\varphi \cdot k^2$
13:            end for
14:        end for
15:         // Neighbours and similarities are already calculated, so fast
16:        clustering[m] ← DBSCAN($\varphi$, similarities, Eps, MinPts)
17:        noise[m] ← X \ clustering[m]
18:        tree.distribution(noise[m], m)
19:    end for
20:    for m ∈ M do
21:        totalClustering ← Merge(clustering[m])
22:    end for
23:    tree.Rebuild(totalClustering)     // We rebuild the tree using the new clustering
24:    return totalClustering
25: end procedure
```

4.4 Comparison

To make a decision on which algorithm is best suited a comparison between the algorithms is made. In addition to the qualifications named in section 2.1, criteria such as network usage, robustness and others are added to give a global picture on the three algorithms. See Table 4.1 for these.

Comparing the three algorithm on these criteria gives a rough estimate of how well each algorithm performs. See Table 4.2 for this comparison. The main consideration is time performance versus clustering quality. Almost all algorithms that run faster than $\mathcal{O}(n^2)$ cannot properly cluster data independent of shape, size or density shifts. Additionally a lot of algorithms perform poorly on non-globular structures.

- Picking **Mini-batch K-Prototypes** as algorithm would give a fast algorithm where no distribution is necessary. However, it is centroid based, and when initialized badly gives poor results. This problem can be overcome by running the algorithm a few times. In addition, there is a problem with representing a mean of nominal data. A possible solution could be to use the mode of the data, or by making a probability distribution (perhaps only for the top x elements, to be determined by sample).

- Picking **LIMBO** gives a fast, already incremental algorithm, and it might not be needed to make it distributed. However, it is centroid based and order dependent. It needs additional computation to cluster bag of words data properly. It has the same problem with categorical data averages as the previous algorithm.
- Picking **Density-based SNN** as algorithm gives an algorithm that gives good clusters independent of data size and shapes. In addition, it works well on noisy data in comparison to the other algorithms. A disadvantage is, that it is a computationally heavy algorithm, meaning it will have a longer runtime.

Criteria	**Explanation**
Distributions	Whether the algorithm works on all distributions of data points
Complexity	Running time in big-$\mathcal{O}$ notation
Large data sets	How well it scales on large data sets
Data structure	The data structure underneath the algorithm
Categorical Data	Can it handle categorical data
Incremental	Can it deal with incremental (online) data
Robustness	Is it insensitive to outliers and noise
Memory usage	How much memory is needed
Database usage	How often is the database contacted
High Dimensions	Does it work on high dimensions
Parameters	How many parameters are used
Network usage	When distributed, does it cause a lot of network traffic
Order-dependent	Is it order dependent
Distributed	Is distribution possible
Parallel	Can we run it in parallel on one machine

Table 4.1: Comparison criteria between the candidate algorithms.

4.5 Choice

Following the comparison made in section 4.4, the density-based Shared Nearest Neighbour algorithm is picked as preferred choice. Although the algorithm is computational heavy compared to the others, we can run the algorithm on a distributed system, lowering the runtime complexity somewhat. This makes it possible to choose a high quality clustering algorithm.

Criteria	**Mini-batch K-Prototypes**	**LIMBO**	**Density-based SNN**
Distributions	Centroid-based	Centroid-based	All types
Complexity	$\mathcal{O}(b \cdot K \cdot I + n \cdot K)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(\varphi^2 + n \log c)$
Large data sets	Scales linearly	Scales linearly	Scales linearly, but quadratic error
Data structure	Centroids	Tree structure	Similarity matrix
Categorical Data	Yes	Yes (poorly)	Yes
Incremental	Yes	Yes, very well	Yes
Robustness	No	No	Yes
Memory usage	$\mathcal{O}(n \cdot K)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \cdot k)$
Database usage	Low	Low	High
High Dimensions	Yes	Less than the other algorithms	Yes
Parameters	Two	Three	Three
Network usage	Very low	Low	Medium
Order-dependent	No	Yes	No
Distributed	Not needed	Yes, but difficult	Yes
Parallel	Yes	Yes	Yes

Table 4.2: Comparison between the candidate algorithms.

The picked algorithm has the advantage that it works well on various types of data, since it finds clusters independent of size, shapes, densities and noise. This is a useful solution for the problem of this thesis, since we prefer an algorithm that can deal with as many types of data sets as possible. Especially the density-independence is something that a lot of alternative algorithms lack, such as Mini-batch K-Prototypes and LIMBO. So a fast runtime is sacrificed for better distinguishable clusters.

In addition, the algorithm works well on high dimensional and nominal data. This can address the problem with clustering bag of words data, such as web pages. This type of data often has on average, a low similarity, but by converting this direct similarity into a shared nearest neighbours similarity we have a suitable match between data points.

The principle that two points are similar to many of the same points implies that they are similar to each other, which overcomes problems of the *curse of dimensionality* [22]. A low direct similarity can get fixed into a high match.

Chapter 5

Algorithmic Approach

As stated in the previous section, DBSNN is picked, the density-based Shared Nearest Neighbour algorithm. A large part of this algorithm has already been explained in section 4.3. In the following sections we go more in depth, explaining characteristics of the algorithm. At first some specific concepts of the algorithm are explained, then how the algorithm clusters incrementally is explained and finally we elaborate on how the algorithm runs on a distributed system. A general scheme of the algorithm is shown in Figure 5.1.

5.1 Density-based Shared Nearest Neighbour Clustering

In this section the asymptotic complexity of the algorithm is derived so we have a better indication of its running time. We are excluding the incremental part of the algorithm for now. After this, the detection of a locally optimal k value for the nearest neighbours calculation is explained.

5.1.1 Complexity Analysis

In order to analyse the complexity of the initial algorithm, we have to look at all separate steps. Starting with the first; sampling and the canopy algorithm.

Canopies: The sample is read from the database and for each of its points we calculate its similarities with the other points. This has a running time depending on sample size f, of $\mathcal{O}(f^2)$, since we compute pairwise similarities for each combination of the sample records. The canopy algorithm has a running time of $\mathcal{O}(n^2)$, but this is in the extreme case where each data point becomes a canopy. If we set the thresholds properly - based on the sample - we can assure that the running time will be a lot less. In tests, the algorithm has shown to be finding a very low percentage of n as canopies by setting the thresholds to average similarity. This will cause its complexity to be more towards linear time, $\mathcal{O}(n)$.

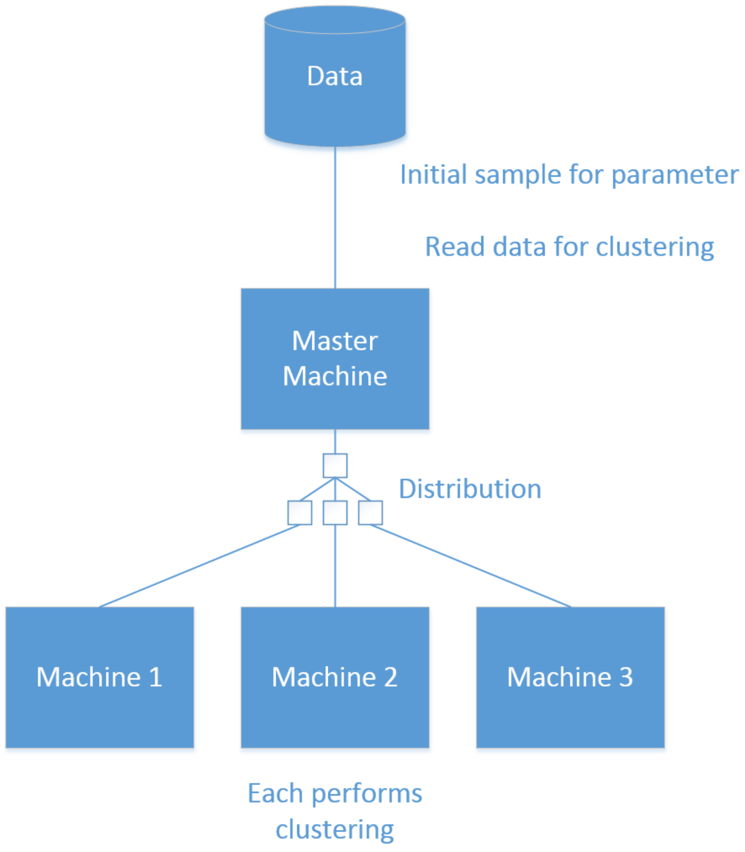


Figure 5.1: Algorithm Scheme

Distribution: The next step in the algorithm is creating the distribution structure and distributing the data points. If we would use a decision tree for this, building it based on canopies would take $\mathcal{O}(c \log c)$, where c is the number of canopies. This tree is $\log c$ levels deeps if perfectly balanced. Distributing the data set would take $\mathcal{O}(n \log c)$, since we have to traverse the tree for each data point. An alternative is using an index with all the canopies. This will cost for n data points $\mathcal{O}(n \cdot c)$, since we have to match each point to the canopies. In practice the index is easier to implement and maintain, since a decision tree would require some form of hierarchical structure of the canopies and balancing might not always be possible.

k-Nearest Neighbours: The dominating factor of the complexity is likely to be the distributed computation. Each machine receives φ data points of the total data set. For each of these points its k-nearest neighbours are calculated, for which all pairwise similarities have to be computed. This takes $\mathcal{O}(\varphi^2 \cdot d)$, where d is the number of dimensions of the data.

Shared Nearest Neighbours: After this we compute the shared nearest neighbour similarity per k nearest neighbours. For each data point we compare its k neighbours with each of those neighbours' neighbours. This takes $\mathcal{O}(\varphi \cdot k^2)$. The computation is independent of the number of dimensions, which means it takes less time than the neighbours calculation for $k < \sqrt{\varphi \cdot d}$. In practice this can easily be even further improved by using sorting or hashing techniques to improve the comparison of neighbour lists.

DBSCAN: The next step is computing the clusters using DBSCAN per machine. Since similarities are already computed and stored it only has to iterate over the pairwise similarities, taking only $\mathcal{O}(\varphi^2)$.

Merging: The last step is merging the found clusters on the master machine. This calculation depends on the K number of clusters found, but this is often not too large. It takes $\mathcal{O}(K^2 \cdot d)$.

Combining all these steps, the running time complexity of the algorithm becomes $\mathcal{O}(\varphi^2 \cdot d)$ for $k < \sqrt{\varphi \cdot d}$ and $K < \varphi$. Memory usage is $\mathcal{O}(\varphi^2)$ per machine for storing the pairwise similarities.

5.1.2 Optimal *k* Detection

The algorithm uses local optimal k detection [40] for determining the k-nearest neighbours. The method finds an optimal number of neighbours per data point still dependent on the input (preferred) k. Using this method to calculate a local optimal k, the algorithm prevents errors by putting a hard k bound on the nearest neighbours list. Instead of this, neighbours are picked outside the preferred k if they have a similarity which is equal or almost equal to that of the k^{th} neighbour, which has the lowest similarity compared to the record.

We first define:

$k^* =$ The preferred value for k.
$n =$ The total number of potential neighbours (data set size).

$$X \sim Bin\left(\frac{k^*}{n}, n\right)$$

The optimal value for k is now found by using the following formula, for $k^* \leq k \leq n$:

$$k_{opt} = \underset{k}{\operatorname{argmax}}\ (\log(d_{k+1} - d_k) + \log(P(X = k))$$

Where $d_{k+1} = 1$ when k = n.

5.2 Incremental

Incremental data is dealt with in a distributed fashion, the master machine matches which machine best fits the incremental data point and passes it to that machine. This machine then tries to cluster it, and if classified as noise point, sends it back. This is as described in section 4.3. Now we will go more in depth on how new data gets fit into the already existing clustering.

In Figure 5.2 the four cases are pictured which can happen when a new data point enters an already existing clustering. The noise and absorption cases are quite easily dealt with, noise is send back and absorption points are just added to the clustering. Absorption points might cause surrounding noise to become border points, but the effect is only in its own k-nearest neighbours.

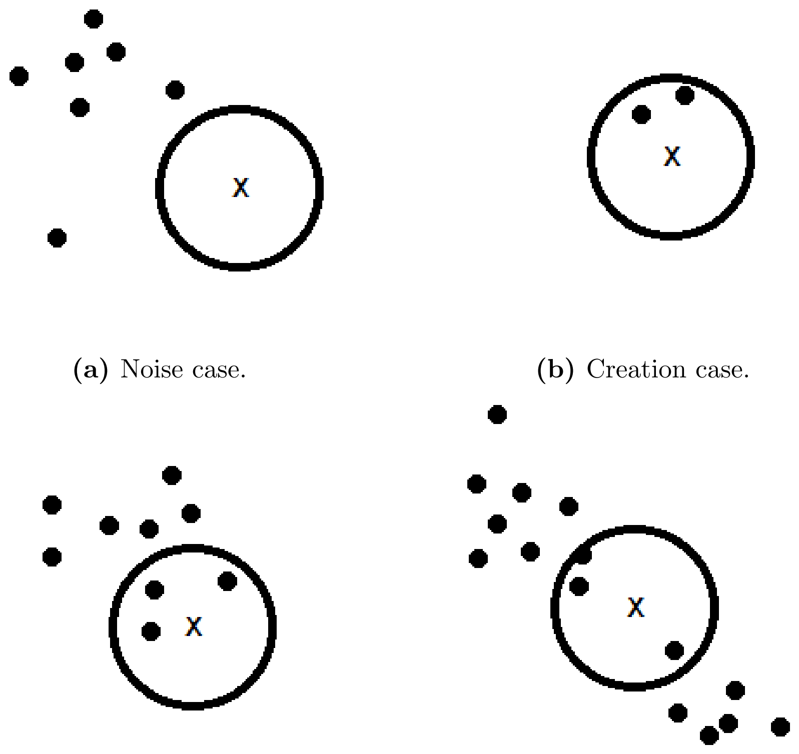
When a data point enters that causes the local density to surpass the *MinPts* threshold a new cluster is formed. This is the creation case. All neighbouring points need to be updated and switched status from noise to either core or border point, and if they become core points, their neighbours need to be checked as well for a possible new status. It is hard to predict how many points will be affected by such an event.

In case a new point has core points in its radius of different clusters, these clusters will need to be merged into one. This means all points of these clusters have to be altered, or at least all minus one.
To keep the computation efficient, instead of adding points one by one, we add batches of data points. This smooths out computation spikes from data points which cause creations or merges of clusters, since these take significantly more time to process. For each batch of new data points we compute the k-nearest neighbours. All already clustered data points compute if the new points are possibly better nearest neighbours and update their list accordingly.

Updating the k-nearest neighbours list can cause gaps to be formed inside clusters or even clusters becoming dissolved (although this is an unlikely case). After updating the k-nn lists, the shared nearest neighbours similarity is computed. Whenever the master machine requests the clusters only DBSCAN needs to be applied to the precomputed similarities, causing a quick response.

The master machine calculates the similarity of different clusters and possibly merges them. More importantly, whenever the master machine pulls for clusters it updates its distribution mechanism with the new found clusters. This way new incremental data gets assigned to their best matching machine throughout time.

Looking at the complexity for an incremental batch of data points of size B; we have directed this batch towards one machine by going through the distribution tree,



(a) Noise case. **(b)** Creation case.

(c) Absorption case. **(d)** Merge case.

Figure 5.2: Different cases for incremental data, based on IncrementalDBSCAN [11].

taking $\mathcal{O}(B \log K)$. Calculating and updating the k-nearest neighbours for all points, including the B new ones takes $\mathcal{O}(\varphi \cdot B \cdot d)$. Shared nearest neighbours computation takes the same as on an initial set, $\mathcal{O}(\varphi \cdot k^2)$. Merging costs $\mathcal{O}(K^2 \cdot d)$ for matching all centroids to each other. Recomputing the distribution tree costs $\mathcal{O}(K \log K)$.

5.3 Distributed

Distribution of data points over the available machines has already been described in section 4.3 in general. In section 3.6 disadvantages and potential problems of distributed computing are pointed out. In this section we will see how the algorithm deals with these problems.

Synchronisation will not be a problem, since the master machine will keep a synchronised state and all machines work independent of each other on their own tasks. Failure is dealt with by time-outs, whenever a machine fails to respond within a certain time frame, the master machine will initiate a new task which can be carried out by a different machine. Stabilisation will always occur, since we do not have non-determinate tasks. An exception to this is the noise movement (data points wrongly clustered on a machine), these get moved around. To prevent non-termination of this movement, each data point is tracked when redistributed to a different machine, so that after an x number of tries it gets assigned to a machine and stays there.

Distributing data points and dividing tasks over workers (often processing units on one workstation machine) are both cases where scheduling can be applied to improve results. During this research only a simple solution for these problems is used. Creating a sophisticated scheduler can be a thesis on its own and since this research intention is to focus on clustering algorithmic it is deemed out of scope.

5.4 Parameters

Parametrization is one of the hardest part of the DBSNN algorithm, since parameters are mostly dependent on the characteristics of the data set. During this research the values for parameters have mostly been determined by testing. Further analysis should be done towards which values are optimal estimations for average data sets. In the distributed density-based Shared Nearest Neighbour algorithm there are a number of parameters to be set:

- $Threshold_1$
  Values within this similarity threshold are added to the canopy. This value is set to the average similarity between samples (which are taken in the initial steps of the algorithm) minus the standard deviance of these samples. By using a sample we make sure there will be a reasonable number of canopies, preventing one canopy consuming all data points or bad performance due to too many canopies being formed.

- $Threshold_2$
  Values within this similarity threshold are removed once added to the canopy, this lead to reduction in the data points on which canopies are calculated. While testing the value of this parameter turned out to be best performing when set equal to $Threshold_1$.

- k
  The number of nearest neighbours that need to be calculated for the nearest neighbouring step of the algorithm. The value is set to $\sqrt{n}$, which is a good estimate according to Saravanan Thirumuruganathan [41]. This parameter directly influences the degree of fine-grainedness of the clustering, meaning, a low k will cause a large number of clusters, where a large k will cause a low number of clusters to be found. This number of clusters depends on what the user of the algorithm wants out of the data set, this problem is described before in the intro of chapter 3.

- *Eps*
  A parameter for the DBSCAN subroutine of the algorithm. It represents the radius for which data points are counted towards being *near*, meaning they can create core points, or get assigned to a cluster if they are a border point themself. If *Eps* is increased we will have less core points, leading to more data points getting removed as noise. Decreasing *Eps* has the opposite effect. The value is set as the average of the similarities in the k-Nearest Neighbours list, this way we capture a lot of points of the data set and remove only truly unrelated noise points.

- $MinPts$
  The number of points threshold that need to be inside *Eps* in order for a point to become a core point, and thus create a cluster or join one. This is of course dependent on the size of k, thus it is estimated as $\sqrt{k}$.

- Number of Machines
  Not a real parameter, but more of a hardware constraint. The clustering performance and quality are dependent on this. If the number of machines increases, performance is increased, but quality is decreased since data points are spread over the machines and cannot become clusters with each other in the Shared Nearest Neighbours step of the algorithm. They might however get merged when a result is calculated by the master machine.

As stated in short at the k parameter, the number of clusters found is directly dependent on this value. This is because *Eps* and $MinPts$ are derived from k in the current implementation. This makes it easier to tweak the clustering of a data set into returning less or more clusters.

5.5 Implementation

The algorithm is implemented in Visual C# version 4.5 using part of the DataDetective code from Sentient Information Systems. The backend framework of DataDetective is inspired by MapReduce [42], mapping tasks to connected machines and reducing the result by combining. Tweaks were made to the framework in order for the clustering algorithm to work, such as the new possibility for machines to locally save part of the computation in memory. Additional speed improvements were made, such as serialization of task commands through Protocol Buffers [43]. The general implementation scheme is shown in Figure 5.3.

In order to keep running time reasonable, parallel programming is used per machine. For the incremental part of the algorithm this has a notable effect in the clustering quality. Because each separate process running in parallel can not know the other process' record information they cannot become neighbours. This error is minimized by performing clustering on batches of records which are as small as possible. An alternative is running the incremental part in sequence, but this slows down the entire clustering considerably.

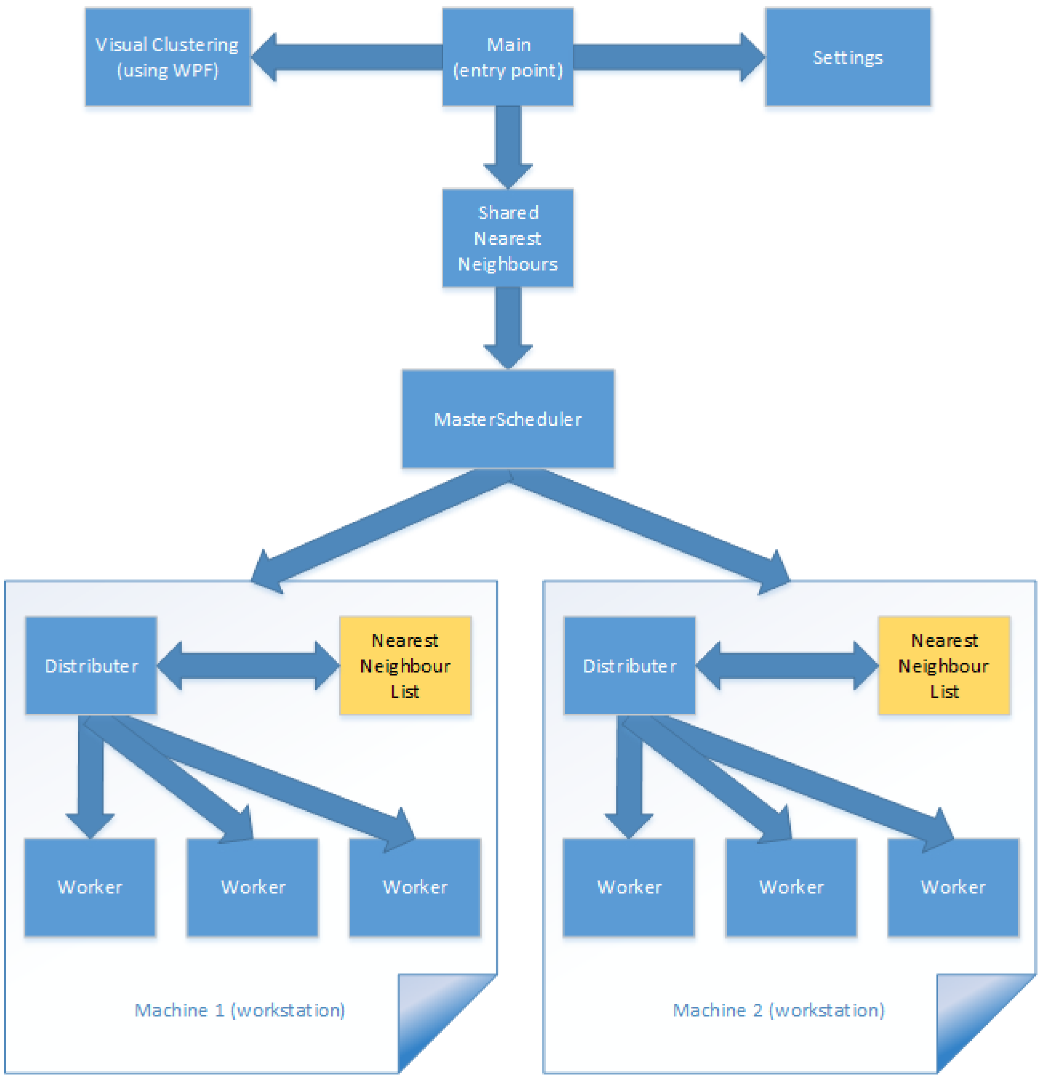


Figure 5.3: Implementation Scheme

Chapter 6

Experimental Setup

This chapter describes the experiments performed to test the performance of the algorithm. The goal is to test the shared nearest neighbours algorithm on cluster quality, performance on large data sets, on incremental data sets and on high dimensionality, see section 2.1.

Experimentation is done with two data sets, the *Drugs and Medicine* web page data set, as described in the problem statement, and an anonymized *NOM* data set, which is a data set containing survey data for marketing purposes where participants are made anonymous.

In order to validate if the clustering algorithm works we have to use quality measures on the found clusters. Which methods are used is also described in this chapter.

6.1 Drugs and Medicine Data Set

The *Drugs and Medicine* data set contains 283.713 records containing information about web pages that are possibly selling drugs and/or medicine. These records are obtained by a web crawler, created by ParaBotS[1]. The web crawler, Xenon, is used by taxing authorities to investigate possible tax evasion by various web sites. It uses time-controlled web spiders that crawl through the web and parse web pages by downloading a page and following its links. Each record in the data set contains the following information about the website:

- Title
  The title of the website. For clustering this value is ignored since we assume almost all titles are unique, and thus they provide little additional grouping discrimination in the data set.

[1] ParaBotS, `http://www.parabots.nl/`

- Country
  The country where the website is hosted, this is encoded as a country code, like US, NL, etc.
- Character Encoding
  Encoding of the website, examples are UTF-8, Windows-1252 or ISO 8859-1.
- Reference Count
  The number of times a reference to this website is found.
- Download Date
  When the site has been downloaded by Xenon.
- URLs
  The URLs the page can be found on.
- Words
  All the unique words the web page contains. Frequency of words is not used.
- Main Language
  Xenon automatically tries to assign a main language per page.
- Sub Languages
  All possible sub languages used on the page.

The words make this data set high dimensional, since each unique value for these attributes can be seen as a separate dimension. In addition, there is no limit to the number of words, and thus, the number of possible dimensions this data set has. This is because the data set incrementally increases by running the web crawler online and letting the cluster algorithm continuously cluster the incoming data. For this thesis, we will experiment with the initial data set, and feed it incrementally into the algorithm. In total there are 1.669.507 unique URLs and 3.562.872 unique words. An example of two records can be seen in table 6.1.

id	**title**	**country-Code**	**charSet**	**URLs**	**words**	**main-Language**	**sub-Languages**
1	Beter in Balans	NL	utf-8	1454	97; 100; 101; 102;	null	null
5	Homeopathie Geldrop	NL	iso-8859-1	1764; 1984	89; 101; 103; 114	Dutch	null

Table 6.1: Samples from the Xenon database. For URLs and words there is a lexicon which maps unique integers to their values.

6.2 National Readership Survey Data Set

The *National Readership Survey*, in short: *NOM*, data set is a data set from the *Nationaal Onderzoek Multimedia* foundation, which is a company that performs research in the field of multimedia for marketing purposes. The data set consists of 21.544 records, each containing information of a person. For each record there are 2004 attributes stored containing information on various subjects, such as sex, age, marital status, income level, residence, etc. and also survey data, such as preference for certain brands, newspaper they read, favorite beverage, etc. All these attributes make this a high dimensional data set, again containing mixed data types.

For this data we already know that an underlying cluster structure should exist, since a lot of attributes are correlated. Only a few main attributes determine the real differences between records, these are sex, age, income level and education. All other attributes are almost all correlated with these attributes. Thus a clustering of the data should be possible and should separate on these main attributes. This makes it ideal for testing the clustering algorithm.

6.3 Validation

It is important to validate any found clustering, so it is possible to determine if the clustering is correct or not, and if the data set even contains some form of clustering tendency. This is important, because we have to distinguish whether or not the data contains any cluster structure, else clustering the set is pointless. Evaluating a clustering can be done in multiple ways, for this experiment we have no external classification of the data, thus we will have to do this unsupervised.

6.3.1 Silhouette Coefficient

Common methods for cluster evaluation are detecting cluster cohesion, determining if points in a cluster are highly similar, and cluster separation, which is determining if clusters are well-separated from each other. The silhouette coefficient [44] combines both cohesion and separation, it determines for an entire clustering, or cluster, or even just a data point a silhouette value between -1 and 1 which represents how well the object fits the cluster it has been assigned to. An advantage of using the silhouette coefficient is its independence of cluster size and it has no overfitting issues like some other measures have. It does, however, generally favor convex clusters over other concepts of clusters, such as density based ones. This is because direct dissimilarity between points is used instead of, for instance, some form of density separation measure. It is defined as follows;

a_i = average dissimilarity of data point i to the other points in its cluster.
b_i = average dissimilarity of data point i to the data points in the nearest other cluster.

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

The smaller b_i is, the better the data point i belongs to the *nearest* other cluster. The smaller a_i is, the better it is assigned to its own cluster. If s_i is negative, point i would better fit its neighbouring cluster than its own cluster and thus the point would be badly clustered. A positive s_i indicates a reasonable partitioning of the data, so a silhouette coefficient closer to 1 is better. Values near 0 indicate overlapping clusters.

For using the silhouette coefficient as a measure we want to give a total clustering one quality number, so we average over all silhouette coefficients of all points and give the result as an overall quality value of the clustering.

$$quality = \frac{1}{n}\sum_{i=1}^{n} s_i$$

For this quality measure negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar. A coefficient very near to -1 indicates either a bad clustering or that there is probably no noticeable clustering found and the data set might not have any kind of underlying cluster structure. A positive quality number would indicate a proper cluster structure.

6.3.2 Simplified Silhouette Coefficient

The silhouette coefficient requires a heavy computation, since for all records pairwise similarities have to be calculated. While our algorithm saves these locally on machines, the matches between points on different machines still need to be computed. This ends up being in the order of magnitude $\mathcal{O}(n^2)$. To reduce this computation an alternative, simplified silhouette coefficient [45] is used. Instead of a_i representing the average dissimilarity of i to the points in its cluster, it now is redefined as the dissimilarity between i and its cluster centroid. Likewise, b_i is redefined as the lowest dissimilarity between i and the nearest cluster centroid. This way we only need to compute all dissimilarities between the data points and the cluster centroids, reducing computation complexity to $\mathcal{O}(n \cdot K)$. The measure seems to perform almost as well as the original silhouette coefficient according to Vendramin *et al.* [45].

Like the silhouette coefficient we again average over all data points, thus the only difference is the calculation of the silhouette coefficient. This gives one quality value over the total clustering.

6.3.3 Davies-Bouldin Index

Another measure of evaluating clusters is the Davies-Bouldin index [46]. This is another internal evaluation, but based on intra and inter cluster distances. It is defined as follows;

σ_i = average dissimilarity of all points in cluster i to its centroid.
$d(x, y)$ = dissimilarity between cluster centroids x and y.

$$DB = \frac{1}{K}\sum_{i=1}^{K}\max_{i\neq j}\left(\frac{\sigma_i+\sigma_j}{d(c_i,c_j)}\right)$$

We average over each clusters' Davies-Bouldin index by summing over the clusters and dividing by the number of clusters K. Per cluster combination i and j, we sum the average distance of all elements inside each cluster as σ. We divide this by the distance between the cluster centroids c. For each cluster we take the maximum of its combinations with other clusters. This results in a measure that is the ratio between intra cluster distances $(\sigma_i + \sigma_j)$ and inter cluster distances $(d(c_i, c_j))$.

The lower the Davies-Bouldin index, the better the clustering has separated clusters with high intra cluster similarities. It can range from 0, where intra cluster distances are very small and inter cluster distances are very large, till positive infinity when inter cluster distances are very small and intra cluster distances are very large. Generally, a value ranging from 0 till 2 indicates a good clustering. This method has the drawback that a good low value returned by this measure does not imply the best clustering [46].

6.4 Tests

In this section the tests are defined which are going to be done with the shared nearest neighbour algorithm. Tests are carried out on an `Intel i7-3770` running at 3.4 GHz with 12 GB of memory. Each test is to be repeated five times and the average picked as result. Exceptions are tests **SNN4** and **Distr4** which are run three times, and the total data set tests **SNN8** and **Distr5** which are run just once, because of their high computation time.

The tests are designed to validate how well the algorithm performs on the stated qualifications in the problem statement.

6.4.1 Data Set Size

First the scaling on data input size is tested, see Table 6.2. To make sure the validation measures are meaningful, we compare the clustering algorithm with an implementation of Mini-batch K-Prototypes, see Table 6.3. If the measures are higher for the DBSNN algorithm we know it performs better than Mini-batch K-Prototypes. The parameter K for the Mini-Batch K-Prototypes algorithm is set to the number of clusters found by the DBSNN algorithm to keep the comparison as fair as possible.

Tests	**Number of data points**	**Parameter** k	**Number of Machines**	**Workers per Machine**	**Sample size**	**Weights**
SNN1	100	10	1	8	10%	All equal
SNN2	1.000	32	1	8	10%	All equal
SNN3	10.000	100	1	8	10%	All equal
SNN4	50.000	224	1	8	10%	All equal

Table 6.2: Test setup for testing with Drugs and Medicine data set using DBSNN. Parameter k is set to $\sqrt{n}$.

Tests	**Number of data points**	**Number of Clusters** K	**Iterations**	**Workers**	**Batch size**	**Weights**
Proto1	100	6	10	8	20	All equal
Proto2	1.000	17	10	8	200	All equal
Proto3	10.000	32	10	8	2.000	All equal
Proto4	50.000	69	10	8	10.000	All equal

Table 6.3: Test setup for testing with Drugs and Medicine data set using K-Prototypes. Parameter K is set to the number of clusters found by the DBSNN algorithm.

6.4.2 Parametrization

Parameter values of the algorithm can influence the number of clusters found, to verify this, a series of tests are done, see Table 6.4. All parameters of the algorithm are derived from the input parameter k, which represents the number of nearest neighbours picked per data point. We expect that the number of clusters increases with the increase of k and that quality measures stay about equal. This last property is desirable, since this makes the quality measures independent of the number of clusters found.

Tests	**Number of data points**	**Parameter** k	**Number of Machines**
Param1	10.000	25	1
Param2	10.000	50	1
Param3	10.000	150	1
Param4	10.000	200	1

Table 6.4: Test setup for testing parameter k with Drugs and Medicine data set using DBSNN.

6.4.3 Words Attribute

The *Words* attribute is one of the most interesting characteristics of the data set, since it is a bag of words representation, and contains hidden dimensions, in the sense that each word can be seen as a separate dimension. Because of this, we might prefer to weight the *Words* attribute more important than the other attributes. In Table 6.5 weights are changed on the attribute to see how this affects the quality of the clustering.

Tests	**Number of data points**	**Parameter** k	**Number of Machines**	**Weights**
Words1	10.000	100	1	Words = 2
Words2	10.000	100	1	Words = 5
Words3	10.000	100	1	Words = 10
Words4	10.000	100	1	Words = 100
Words5	10.000	100	1	Words = 200
Words6	10.000	100	1	Words = 300

Table 6.5: Test setup for testing with Drugs and Medicine data set using DBSNN with different weights on *Words*.

6.4.4 Incremental Testing

In Table 6.6 tests for incremental data input are stated. The initial data set size is varied to compare how the algorithm deals with a high ratio of incremental data versus a low initial data set. This way we can examine the quality of the incremental clustering.

Tests	**Number of data points**	**Parameter** k	**Number of Machines**	**Initial Set**
Inc1	10.000	100	1	75%
Inc2	10.000	100	1	50%
Inc3	10.000	100	1	25%
Inc4	10.000	100	1	10%
Inc5	10.000	100	1	1%

Table 6.6: Test setup for testing with Drugs and Medicine data set using DBSNN and incremental clustering.

6.4.5 Distributed Testing

Testing the distributed component of the algorithm is done in two ways. Firstly, the number of machines used for the clustering task is varied, see Table 6.7. Secondly, the number of data points is varied, see Table 6.8. With the first set of tests we can verify how an increasing number of machines influence the running time, and how it affects clustering quality. The second group of tests, compared with the results of Table 6.2, will give insight in quality loss due to distributing the computation. Although more data is used than in the tests of Table 6.2, the data points per machine and the parameters derived from this are the same, providing a somewhat fair comparison.

Tests	**Number of data points**	**Parameter** k	**Number of Machines**
Distr1	10.000	71	2
Distr2	10.000	58	3
Distr3	10.000	50	4

Table 6.7: Test setup for testing with Drugs and Medicine data set using DBSNN and multiple machines. Parameter k is set to $\sqrt{n/m}$. The last test is the entire data set.

Tests	**Number of data points**	**Parameter** k	**Number of Machines**
Distr4	200	10	2
Distr5	2.000	32	2
Distr6	20.000	100	2
Distr7	100.000	224	2
Distr8	283.713	266	4

Table 6.8: Test setup for testing with Drugs and Medicine data set using DBSNN and multiple machines while increasing the data set. Parameter k is set to $\sqrt{n/m}$. The last test is the entire data set.

6.4.6 NOM Data Set

Using the *NOM* data set we can verify if the algorithm works on other data sets too. If we compare the tests in Table 6.9 with the first batch of tests with the DBSNN algorithm we can see how it runs on a different data set.

Tests	**Number of data points**	**Parameter** k	**Number of Machines**
SNN5	500	31	1
SNN6	1.000	50	1
SNN7	2.000	79	1
SNN8	21.544	387	1

Table 6.9: Test setup for testing with NOM data set using DBSNN. Parameter k is set to $1/2 \cdot n^{\frac{2}{3}}$, found by testing. Last test is entire data set.

6.4.7 DataDetective Clustering Algorithm

In addition to testing new algorithms, the current algorithm used for clustering by Sentient in DataDetective[2] is tested. The setup for this is shown in Table 6.10. This clustering algorithm is force–directed, meaning it will apply a dimension reduction first, and then use forces on each data point to cluster. This works by applying attracting forces between similar points and opposing forces between dissimilar points. Data points that end up near each other form clusters. We want to compare this technique with the DBSNN algorithm.

Tests	**Number of data points**	**Number of Machines**
DD1	100	1
DD2	1.000	1
DD3	10.000	1
DD4	50.000	1

Table 6.10: Test setup for testing with Drugs and Medicine data set using DataDetective.

[2] DataDetective, `http://www.sentient.nl/?dden`

Chapter 7

Results

The results of the tests as defined in section 6.4 are discussed in this chapter. In addition, the visualisation of the results is shown in the last section.

7.1 Experimental Results

An example of the clusters found can be seen in Table 7.1, where for the clusters found in the *Drugs and Medicine* data set, a sample of the words is shown. Each test that has been done uses the internal validation criteria described before, namely the simplified silhouette coefficient and the Davies-Bouldin index. A higher silhouette coefficient and a lower Davies-Bouldin index generally indicate better clusterings. As stated before, each experiment is repeated five times. In this chapter only averages are displayed, detailed results can be found in the appendices.

Cluster	**Words (7 most significant)**
1	praktijk, behandeling, tarieven, klachten, afspraak, behandelingen, lichaam
2	information, life, events, health, news, history, resources
3	arnhem, amersfoort, utrecht, breda, amsterdam, leeuwarden, zwolle
4	agenda, links, therapie, contact, leven, vragen, jezelf
5	geestelijke, gezondheidszorg, zorg, verpleging, ziekenhuizen, eerstelijnszorg, farmacie
6	producten, voorwaarden, algemene, privacy, winkelwagen, eiwitten, creatine
7	vitamins, vetzuren, berry, magnesium, vezels, aminozuren, anti-oxidanten
8	health, care, conditions, treatment, therapy, physical, condition
9	arts, informatie, disclaimer, complementaire, therapeut, geneeswijzen, alternatieve

Table 7.1: Example of clusters found on Drugs and Medicine data set using DBSNN. Here weight of the *Words* attribute is set to 1000, creating clusters mostly separated on their word lists. Statistical significance determined by using Fisher's exact test, which is a statistical test used to determine association between the cluster's data points compared to the total data set, for more information see [47].

7.1.1 Data Set Size

When we look at the results in table 7.2, we can see that the running time of the algorithm increases quadratic with the input size. This increase is as expected, since we compute pairwise similarities. A more severe problem is that the silhouette of the clustering decreases rapidly as the data set size increases, which indicates less well-defined clusters. This can imply multiple things. Since the silhouette does not become negative it looks as if the clustering found is still correct, however, a silhouette near zero indicates either a large portion of the clusters overlapping or that no well-defined structure exists on the data set. When comparing clusters using Fisher's exact test on their attributes we still see that clusters are significantly different from each other. This, and the Davies-Bouldin index indicating a good clustering seem to imply that we do find a well-defined clustering, but it might just contain a lot of overlap. Overlapping clusters seem plausible since we cluster web pages with a lot of words, where each page might have some overlapping words with any other page.

Comparing the results in tables 7.2 and 7.3 we can see how well the Shared Nearest Neighbours algorithm compares against the Mini-batch K-Prototypes algorithm. We can see that, as expected, Mini-batch K-Prototypes runs significantly faster on larger data sets than Shared Nearest Neighbours. However, quality of the clusters is much lower. The quality difference is even higher when considering that the quality measures return better values on convex clusters, since they use centroids in their calculations. Thus, the choice between either DBSNN or K-Prototypes is a choice between either finding high quality clusters or having a fast running time.

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
SNN1	0:22	6	22 (22%)	0,592	1,799
SNN2	0:35	17	53 (5%)	0,581	2,009
SNN3	5:15	32	806 (8%)	0,363	1,884
SNN4	1:37:39	69	5107 (10%)	0,296	2,548

Table 7.2: Test results with Drugs and Medicine data set using DBSNN.

Tests	**Clustering Time**	**Number of Clusters**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Proto1	1:34	6	0,297	2,945
Proto2	1:46	17	0,373	2,840
Proto3	3:08	32	0,302	4,247
Proto4	10:21	69	0,220	12,447

Table 7.3: Test results with Drugs and Medicine data set using K-Prototypes.

7.1.2 Parametrization

Table 7.4 shows the results for the tests using various values for parameter k. Comparing these with each other and with the **SNN3** test, we can see that higher values for k reduce the number of clusters found, which is as expected. Quality measures seem to stay about equal over the different parameter settings. This means that the algorithm has the desirable property that clusters found and their quality is independent of the number of clusters. The parameter k setting thus provides a way to configure the fine-grainedness of the clustering as a user of the algorithm. Comparing the results with the result of the **SNN3** test we can see that the estimation of k as $\sqrt{n}$ seems reasonable, it finds 32 clusters, which is a reasonable number to define the structure of the data set.

We can also note, when looking at the results in Table 7.4, that the DBSNN algorithm removes more noise on lower k values. This is because less data points are kept as nearest neighbours, making it easier for data points to not have a core point in their k nearest neighbours list. Not having a core point in the k nearest neighbours causes a data point to become discarded as noise point.

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Param1	5:18	207	1838 (18%)	0,353	2,789
Param2	5:13	76	1074 (11%)	0,327	2,493
Param3	5:30	16	318 (3%)	0,416	3,050
Param4	5:31	11	196 (2%)	0,373	2,869

Table 7.4: Test results with Drugs and Medicine data set using various k values as parameter.

7.1.3 Words Attribute

In Table 7.5 we can see how the DBSNN algorithm responds to increasing weights on the *Words* attribute. Comparing these tests with the **SNN3** test, we can see that overall quality is considerably worse than when using equal weights for all attributes. Significantly more noise gets removed from the data set when the weight for *Words* is increased. Discriminating on the *Words* attribute thus seems hard for the DBSNN algorithm. Probably this is either because of overlapping clusters or because there is no deeper underlying structure when looking at just the *Words*. However, when we look at the words found inside the clusters, we can clearly see that the clusters make *sense*. In Table 7.1 a few of these clusters for an extremely high weight of 1000 are shown.

It seems that the cluster validation measures do not work that well on these tests. A possible reason might be that adding *Words* to the clustering decreases contrast, for example: if we were to have only a few dimensions such as *Country* and

Character Encoding the clustering algorithm can find pretty strictly separated clusters, providing good results to the measures. However, if we add *Words* to the data set it becomes harder to find strictly defined clusters. This might be a possible reason why increasing weights on *Words* leads to worse quality measure results.

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Words1	4:52	34	824 (8%)	0,272	1,885
Words2	4:55	28	816 (8%)	-0,145	2,168
Words3	5:25	20	815 (8%)	-0,343	2,492
Words4	5:44	23	1058 (11%)	-0,523	11,097
Words5	5:48	21	1063 (11%)	-0,533	12,041
Words6	5:51	18	1056 (11%)	-0,525	12,973

Table 7.5: Test results with Drugs and Medicine data set using DBSNN with a different weight on *Words*.

7.1.4 Incremental Testing

Table 7.6 shows the results for incremental clustering. We can see that all qualifications stay somewhat constant when changing the size of the initial data set. The running time improves a little without quality decreasing that much. This means that an incremental clustering result seems to be independent of the initial data set size.

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Inc1	5:20	43	780 (8%)	0,485	2,689
Inc2	4:59	41	543 (5%)	0,444	3,226
Inc3	4:19	39	578 (6%)	0,417	3,947
Inc4	4:04	39	595 (6%)	0,395	2,861
Inc5	3:46	44	914 (9%)	0,432	4,064

Table 7.6: Test results with Drugs and Medicine data set using DBSNN and incremental clustering.

7.1.5 Distributed Testing

In tables 7.7 and 7.8 the results for different distributed setups are shown. The first table shows the results for scaling up the number of machines on the task. Comparing these results with the **SNN3** test we can see that increasing machines from one to two yields the best running time improvement, cutting time in half. Further addition of machines does not reduce running time in the same way, this is probably due to overhead costs on the master machine.

Besides running time improvement, the number of clusters increases. This is something that is expected, since each machine clusters its own subset, creating a more fine-grained overall clustering. Quality measures indicate that clusters found are even a little better than when only using one machine. This is probably because of the more fine-grained clustering. If we were to change parameters a bit when running on one machine we might get similar results.

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Distr1	2:42	65	939 (9%)	0,470	2,374
Distr2	2:32	89	975 (10%)	0,447	2,415
Distr3	2:21	105	1038 (10%)	0,448	2,601

Table 7.7: Test results with Drugs and Medicine data set using DBSNN and multiple machines.

Comparing the results of the second group of distributed tests, Table 7.8, with Table 7.2, we can see that doubling the number of machines and data points causes little increase in running time, only some additional overhead is computed. Quality measures stay equal or even become better. This is probably because of the pre-clustering, causing the clusters found per machine to be already pre-picked. There are only small differences between the clusters because of this.

The amount of noise does peak in test **Distr6** for unknown reasons, it seems to be normal on the other tests though. Running time on the last test is quite high. This is because this test was ran on four machines and on the entire data set to see if it was computationally possible to cluster all data. On this test, 3 of the 4 machines were done in less than 10 hours, but one machine ran out of memory, becoming a bottleneck and slowing down the entire test considerably. Solving this scheduling issue is not simple, and is something done in possible future work, see chapter 10.

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Distr4	0:32	16	36 (18%)	0,656	1,840
Distr5	0:52	29	79 (4%)	0,514	2,485
Distr6	6:30	89	3902 (20%)	0,403	9,915
Distr7	1:59:42	120	7400 (7%)	0,213	4,749
Distr8	50:11:32	214	21620 (8%)	0,204	8,089

Table 7.8: Test results with Drugs and Medicine data set using DBSNN and multiple machines. Note that the last test is on the entire data set.

7.1.6 NOM Data Set

In Table 7.9 the results for clustering the *NOM* data set are shown. Quality stays good when increasing the data set size. however, computation time increases rapidly (take into account that the last test was on the entire data set). This is mainly caused by the high number of dimensions of the data; for each pair-wise similarity computation has to match over all these dimensions. In addition, a more technical aspect causes high computation: the data is saved as binary bitstrings for which similarity comparisons are not optimized yet. When clustering this type of data optimising this calculation is something that would be one of the first things to do in future work. Computation time can furthermore be reduced by running the clustering on a distributed system, which as shown in the distributed tests reduces running time with little quality loss.

Examples of clusters found in the NOM data set can be viewed in Table 7.10.

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
SNN5	1:10	23	49	0,090	3,033
SNN6	3:27	16	68	0,178	2,964
SNN7	13:20	16	122	0,717	2,917
SNN8	23:43:23	22	813	0,399	3,102

Table 7.9: Test results with NOM data set using DBSNN.

Clusters	Attribute (4 most significant)	Property
Cluster 1	Dagbladcombinaties Provincie Luisterfrequentie Radio Kijkfrequentie Televisie	Noordhollands Dagblad Noord-Holland Radio Noord-Holland Nederland 3
Cluster 2	Welstand Frequentie activiteiten Hoogst voltooide opleiding Kleinkinderen	W5 (Laag) Puzzelen Geen onderwijs, basisonderwijs Van 18+ jr
Cluster 3	Regionale dagbladen Intresses Provincie Luisterfrequentie Radio	Leeuwarder Courant Politiek Friesland Radio Noord
Cluster 4	Activiteiten Geslacht respondent Huurwoning/eigen woning Drogisterijartikelen	Familie, vrienden bezoeken Vrouw Eigen woning ICI PARIS XL
Cluster 5	Advies geven aan anderen Activiteiten Burgelijke staat Leeftijdsklasse	Cosmetica Naar discotheek/club gaan Ongehuwd 15-24 jaar
Cluster 6	Online gamen Burgelijke staat Positie in huishouden Bezit rijbewijs	Vaak Ongehuwd Kind Geen
Cluster 7	Gezinsfase NOM Provincie Kerkelijke gezindte Energiebedrijf elektriciteit	Oude tweepersoonshuishoudens Noord-Brabant Rooms Katholiek Essent
Cluster 8	Sportbladen Merk maaltijdpakket Informatie kunnen geven over Auto accessoire meest gebruikt	Voetbal International Wagner Pizza Auto's Carkit
Cluster 9	Intresses Leeftijd respondent 25-54 jaar Intresses Gebruik haarproducten	Automatisering en computers Ja Gadgets/nieuwe technologien Gel/gelspray
Cluster 10	Beroepsgroep respondent Dagbladabonnementen Huurwoning/eigen woning Inkomensklasse	Werkloos/arbeidsongeschikt/bijstand Geen abonnement Huurwoning Laag

Table 7.10: Example of clusters found on NOM data set using SNN. Statistical significance is again determined by using Fisher's exact test [47].

7.1.7 DataDetective Clustering Algorithm

Table 7.11 shows the results of clustering the *Drugs and Medicine* data set with the current DataDetective clustering algorithm. Comparing these results with Table 7.2 shows a lot of differences. Quality of clusters found are generally higher with DataDetective on smaller data sets, however, on larger data sets the quality seems to decrease rapidly. Considering that DataDetective does not run on a distributed system, we can state that the DBSNN algorithm scales better on larger data sets, since it maintains its clustering quality and its running time can be reduced by adding machines to the computation.

In addition, the DataDetective clustering algorithm discards a lot more data points as noise. The quality measures are only calculated over the clustered (assigned) data, and it seems that the DataDetective algorithm only assigns the best points to a cluster. This creates a quality advantage since the DBSNN algorithm discards far less data as noise.

Tests	**Clustering Time**	**Number of Clusters**	**Unassigned Points**	**Silhouette Coefficient**	**Davies-Bouldin Index**
DD1	0:14	3	56 (56%)	0,604	0,956
DD2	0:22	25	272 (27%)	0,738	0,742
DD3	5:15	45	1362 (14%)	0,494	2,316
DD4	1:44:22	152	6812 (14%)	-0,111	1,624

Table 7.11: Test results with Drugs and Medicine data set using DataDetective.

7.2 Visualisation

Visualisation of the clustering is done in WPF[1], and using the Extended WPF Toolkit[2]. Firstly, centroids of the clusters are determined and are placed in relation to each other on the drawing field. Their relative position is calculated by a custom made algorithm based on Fruchterman-Reingold [10]. Then they are spread over the canvas. For each data point we calculate similar they are to the cluster centroids. Depending on these values the data gets spread out over the canvas, closer to the centroid position if they have a high similarity to its own cluster, and more towards other centroids if they have a high similarity with them. Data points of the same cluster get colored an equal color. This is shown in Figures 7.1 and 7.2.

By only computing similarities between centroids and data points the positions of the data might not be accurately describing their relation between them and other data points. This has been a trade-off, since computing pairwise similarities just for drawing clusters is a heavy performance price to pay, even worse considering spreading the data points depending on Fruchterman-Reingold takes several iterations. For large data sets this just does not work out.

Another thing to note is that when we represent data in a metric space (like the canvas) in terms of their similarities we get two problems [48];

1. Triangle inequality implies transitivity.
   The triangle inequality that one assumes when looking at metric spaces would mean that similarities are transitive, which they are not. When point A is similar to point B, and point B is similar to point C, point A and C can have very little similarity. For example; the word *fall* is similar (or related) to words like *autumn* or *season*, if clustered these would appear near each other. But *fall* can have a different meaning, as in falling down. Then it is similar to words like *tumble* or *slip*, so it should be clustered with these as well. However, *autumn* and *tumble* have very little to do with each other but are mapped next to each other on metric space.

2. Number of nearest neighbours is limited.
   In metric space only a limited number of points can have the same point as their nearest (most similar) data point. In high dimensions this is possible because of the high number of dimensions creating sufficient space, but when mapped to low dimensions this becomes a problem. A data point which is the best neighbour for a lot of other data points can never be properly in the middle of these data points, more than the points are to each other.

[1] Windows Presentation Foundation, http://msdn.microsoft.com/en-us/library/aa970268(v=vs.110).aspx

[2] Extended WPF Toolkit, http://wpftoolkit.codeplex.com/

Because of these two fundamental problems there will be some interpretation distortion when viewing the clustered data.

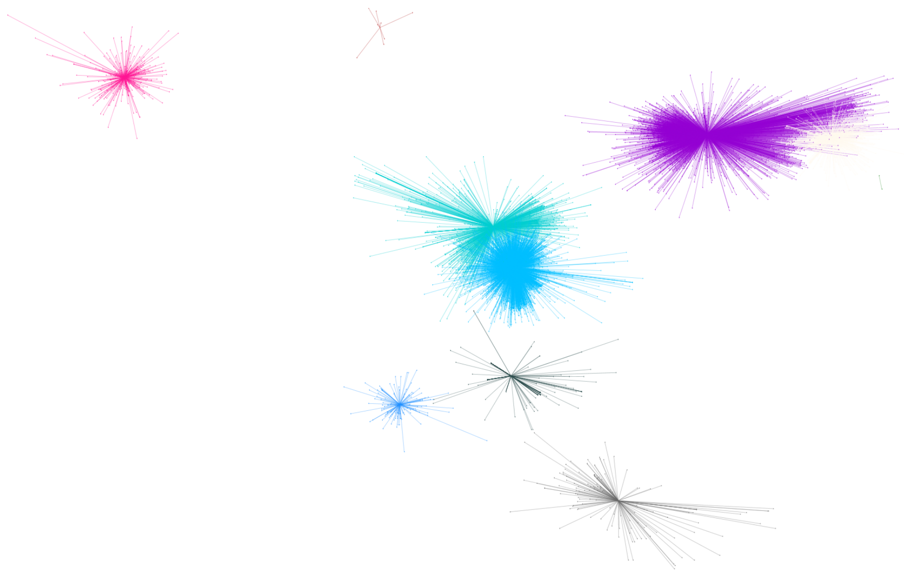


Figure 7.1: Example of visualisation of a clustering, in this case of 10.000 records of the Xenon data set.

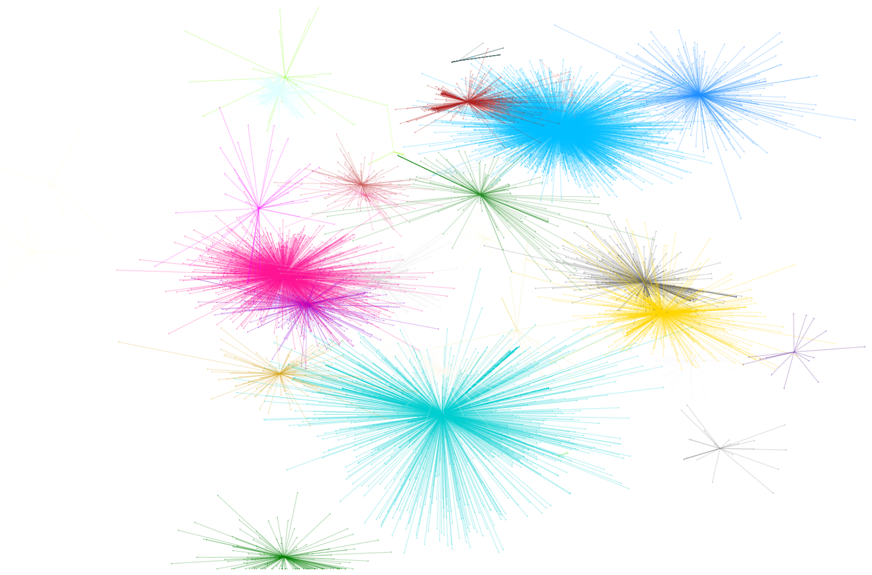


Figure 7.2: Another visualisation of a clustering, 10.000 records of the Xenon data set but with a lower k parameter, creating more clusters.

Chapter 8

Conclusions

In this research clustering algorithms and techniques for clustering high dimensional, incremental data containing nominal and numeric attributes have been explored. An implementation is given for a distributed density-based shared nearest neighbours clustering algorithm that clusters this type of data. In addition, the result gets visualised on screen.

This approach provides a generic way to summarize any kind of data and detect groups of similar points. Following the results from chapter 7 we can state that the algorithm performs reasonably well compared to Mini-batch K-Prototypes and the currently implemented clustering algorithm of DataDetective. It can cluster data incrementally without much quality loss and can be run on a distributed system to reduce computation time.

To answer the research questions posed:

- *How can we design a clustering algorithm that can cluster a large set of high-dimensional data incrementally?*
  As shown in this thesis, there are multiple ways of creating such an algorithm. I have picked the density-based SNN approach which works well on this problem. It obtains high quality clusters independent of shape, size, densities and noise, while working on various types of data. Its only major downside is its runtime complexity, but this is reduced by distribution of the computation.

- *Can we implement this in a distributed way?*
  The implemented algorithm runs on a distributed system, making it scalable to large data sets by adding more machines, reducing running time. When looking at the results of the distributed tests we can also state that it does not lower clustering quality a lot.

Chapter 9

Discussion

Cluster analysis is an approach of unsupervised learning; we want to find patterns in unlabeled data. Since we never know how these patterns might look, it can be hard to evaluate the results properly. This problem is inherent in unsupervised learning, and makes it hard to verify any results.

In this thesis we have seen that density-based shared nearest neighbours clustering works well on the specified problem. The algorithm is a good all-round clustering algorithm, which is flexible on input data types and can detect clusters in different structures, such as density separated ones. There are, however, as mentioned in the introduction, a lot of possible clustering algorithms. For different cases of data, specific algorithms will often work better, this is simply because in unsupervised learning there is no defined objective for finding patterns. Because of this, algorithms tailored specifically for a data set will give a more satisfying result.

The most notable shortcoming of the DBSNN algorithm is its quadratic running time, which can only be reduced by distribution of the computation. The distribution does, however, disable data points separated by being on different machines from forming a cluster together. This increases the error rate of the clustering, lowering overall quality. It does not need to be a bad thing though, since for large data sets we might prefer data points not grouping together into very large clusters, but instead forming small clusters per machine. A lot of small clusters do provide more information about the data set than only a few large clusters, which can be found more easily by just clustering a reasonable sample of the data set.

A different problem of the algorithm is the parametrization. While this is reduced to just defining a k value, it is hard to determine for which value a proper clustering is found for the data set. Unfortunately, this is a fundamental problem of unsupervised learning, since it is unknown at which level of fine-grainedness a clustering is considered *good*. This was also described in the introduction of chapter 3. There is no solution to this, and instead we will have to use trial and error on parameter values.

Chapter 10

Future Work

In this thesis cluster analysis on large, incremental data sets has been investigated. While some interesting work has been done, there are still ways to expand on this research.

A number of algorithmic optimizations could be possible. Most important, a more robust and optimised scheduler of the distribution could be constructed. This way the total work load is better spread over the distributed system. The initial distribution method could be changed to Locality-Sensitive Hashing [49], reducing the running time of the distribution of the data.

To improve interpretation of the clustering result a better visualisation algorithm could be picked. At the moment a custom altered Fruchterman-Reingold algorithm, using only centroids initially, is used. This could be improved by applying some optimized form of this algorithm on the total clustered data set [50]. An additional improvement could be converting the 2D canvas to a 3D representation.

Although data set specific, implementing a better similarity matcher for bag of words data could improve the clustering. In the current implementation all words have the same weight in the similarity calculation, but by using techniques such as *term frequency-inverse document frequency* (TF-IDF [51]), less common words could have a more significant impact on the similarity match. For instance, when two pages both contain a very rare word, they will have a relative higher match. An alternative solution to the bag of words data problem would be applying feature selection and select only a subset of (possibly rare) words.

Lastly, in the future the work done can be integrated into the DataDetective[1] tool created by Sentient Information Systems[2], so that the users of the tool can benefit from the clustering algorithm.

[1] DataDetective, http://www.sentient.nl/?dden

[2] Sentient Information Systems, www.sentient.nl

List of Algorithms

List of Figures

List of Tables

Bibliography

[1] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson - Addison-Wesley, Boston, 2006.

[2] Vladimir Estivill-Castro. Why so many clustering algorithms - A Position Paper. *ACM SIGKDD Explorations Newsletter*, 4(1):65–75, 2002.

[3] Richard Ernest Bellman. *Dynamic Programming*. Rand Corporation, 1957.

[4] Alexander Mol. Sentient Segmentatiealgoritme, September 2013. Data Science Intern.

[5] Marijn Lems. Identifying Spam Gangs Based on Shared Resources. Master's thesis, VU University Amsterdam, March 2013.

[6] Vipin Kumar. An Introduction to Cluster Analysis for Data Mining, 2000. http://www-users.cs.umn.edu/~han/dmclass/cluster_survey_10_02_00.pdf.

[7] Rui Xu and Donald Wunsch II. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, May 2005.

[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD-96 Proceedings*, pages 226–231, 1996.

[9] Arthur P Dempster, Nan M Laird, Donald B Rubin, et al. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal statistical Society*, 39(1):1–38, 1977.

[10] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, and Xiaowei Xu. Incremental clustering for mining in a data warehousing environment. In *VLDB*, volume 98, pages 323–333, 1998.

[12] Dimitris K Tasoulis and Michael N Vrahatis. Unsupervised Distributed Clustering. *Parallel and Distributed Computing and Networks*, pages 347–351, 2004.

[13] Eamonn Keogh and Jessica Lin. Clustering of time-series subsequences is meaningless: implications for previous and future research. *Knowledge and information systems*, 8(2):154–177, 2005.

[14] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An Efficient Data Clustering Method for very Large Databases. *ACM SIGMOD Record*, 25(2):103–144, June 1996. ACM.

[15] Periklis Andritsos, Panayiotis Tsaparas, Renée J Miller, and Kenneth C Sevcik. LIMBO: Scalable Clustering of Categorical Data. *Advances in Database Technology-EDBT 2004*, pages 123–146, 2004.

[16] Weizhong Zhao, Huifang Ma, and Qing He. Parallel K-Means Clustering Based on MapReduce. *Cloud Computing*, pages 674–679, 2009.

[17] Eshref Januzaj, Hans-Peter Kriegel, and Martin Pfeifle. DBDC: Density Based Distributed Clustering. In *Advances in Database Technology - EDBT 2004*, pages 88–105. Springer Berlin Heidelberg, 2004.

[18] Jon Louis Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):509–517, 1975.

[19] Rodrigo Paredes, Edgar Chávez, Karina Figueroa, and Gonzalo Navarro. Practical Construction of k-Nearest Neighbor Graphs in Metric Spaces. In *Experimental Algorithms*, pages 85–97. Springer, 2006.

[20] Jeffrey S Beis and David G Lowe. Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 1000–1006. IEEE, 1997.

[21] Thomas Bernecker, Michael E Houle, Hans-Peter Kriegel, Peer Kröger, Matthias Renz, Erich Schubert, and Arthur Zimek. Quality of similarity rankings in time series. *Advances in Spatial and Temporal Databases*, pages 422–440, 2011.

[22] Michael E Houle, Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Can Shared-Neighbor Distances Defeat the Curse of Dimensionality? In *Scientific and Statistical Database Management*, pages 482–500. Springer Berlin Heidelberg, 2010.

[23] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Clustering High-Dimensional Data: A Survey on Subspace Clustering, Pattern-Based Clustering, and Correlation Clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(1):1, 2009.

[24] Karl Pearson. On lines and Planes of Closest Fit to Systems of Points in Space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[25] Jonathon Shlens. A Tutorial on Principal Component Analysis. *Systems Neurobiology Laboratory, University of California at San Diego*, 82, 2005.

[26] Christos Faloutsos and King-Ip Lin. *FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets*, volume 24. ACM, 1995.

[27] Andrew McCallum, Kamal Nigam, and Lyle H Ungar. Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178. ACM, 2000.

[28] Raymond A Jarvis and Edward A Patrick. Clustering Using a Similarity Measure Based on Shared Near Neighbors. *Computers, IEEE Transactions on*, 100(11):1025–1034, 1973.

[29] Levent Ertöz, Michael Steinbach, and Vipin Kumar. A New Shared Nearest Neighbor Clustering Algorithm and its Applications. In *Workshop on Clustering High Dimensional Data and its Applications at 2nd SIAM International Conference on Data Mining*, pages 105–115, 2002.

[30] Gerard Salton and Michael J McGill. Introduction to Modern Information Retrieval. 1983.

[31] Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. A Framework for Clustering Evolving Data Streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 81–92. VLDB Endowment, 2003.

[32] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-Based Clustering over an Evolving Data Stream with Noise. In *SDM*. SIAM, 2006.

[33] Arnon Rotem-Gal-Oz. Fallacies of Distributed Computing Explained. page 20, 2006. http://www.rgoarchitects.com/Files/fallacies.pdf.

[34] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(3):283–304, 1998.

[35] D Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178. ACM, 2010.

[36] Periklis Andritsos, Panayiotis Tsaparas, Renee J Miller, and Kenneth C Sevcik. LIMBO: A Linear Algorithm to Cluster Categorical Data. Technical report, UofT, Dept of CS, CSRG-467, 2003.

[37] Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data. In *SDM*, 2003.

[38] Sumeet Singh and Amit Awekar. Incremental Shared Nearest Neighbor Density-based Clustering. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1533–1536. ACM, 2013.

[39] Fernando Mendes, Maribel Yasmina Santos, and João Moura-Pires. Dynamics Analytics for Spatial Data with an Incremental Clustering Approach. 2013.

[40] Vincent Hoekstra and Michel de Ruiter. Finding the optimal K for fuzzy matching. *Sentient Information Systems*, 2014.

[41] Saravanan Thirumuruganathan. A Detailed Introduction to K-Nearest Neighbor (KNN) Algorithm, May 2010. http://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/.

[42] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[43] Marc Gravell. Protocol Buffers, September 2013. http://code.google.com/p/protobuf-net/.

[44] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[45] Lucas Vendramin, Ricardo JGB Campello, and Eduardo R Hruschka. Relative Clustering Validity Criteria: A Comparative Overview. *Statistical Analysis and Data Mining*, 3(4):209–235, 2010.

[46] David L Davies and Donald W Bouldin. A Cluster Separation Measure. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2):224–227, 1979.

[47] Rick Routledge. Fisher's exact test. *Encyclopedia of biostatistics*, 2005.

[48] Laurens van der Maaten and Geoffrey Hinton. Visualizing non-metric similarities in multiple maps. *Machine learning*, 87(1):33–55, 2012.

[49] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[50] Josh Barnes and Piet Hut. A hierarchical o (n log n) force-calculation algorithm. 1986.

[51] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets.* Cambridge University Press, 2012.

Appendix A

Results Density-based Shared Nearest Neighbours Algorithm

Tests	Clustering Time	Number of Clusters	Noise Removed	Silhouette Coefficient	Davies-Bouldin Index
SNN1.1	0:21	5	20	0,553	1,929
SNN1.2	0:23	6	23	0,596	1,867
SNN1.3	0:23	6	23	0,596	2,029
SNN1.4	0:21	8	23	0,618	1,769
SNN1.5	0:24	6	23	0,596	1,403
SNN1 avg	0:22	6	22	0,592	1,799
SNN1 sdv	0:01	1,095	1,342	0,024	0,241

Table A.1: *Drugs and Medicine*, n = 100, k = 10

Tests	Clustering Time	Number of Clusters	Noise Removed	Silhouette Coefficient	Davies-Bouldin Index
SNN2.1	0:34	15	52	0,566	1,869
SNN2.2	0:35	19	54	0,596	2,023
SNN2.3	0:35	17	53	0,578	2,139
SNN2.4	0:37	18	53	0,584	1,983
SNN2.5	0:37	17	53	0,578	2,034
SNN2 avg	0:35	17	53	0,581	2,009
SNN2 sdv	0:01	1,483	0,707	0,011	0,098

Table A.2: *Drugs and Medicine*, n = 1.000, k = 32

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
SNN3.1	5:13	33	811	0,364	1,859
SNN3.2	5:16	33	816	0,365	1,923
SNN3.3	5:16	32	797	0,362	1,875
SNN3.4	5:17	31	788	0,359	1,958
SNN3.5	5:16	33	816	0,365	1,804
SNN3 avg	5:15	32	806	0,363	1,884
SNN3 sdv	0:01	0,894	12,542	0,002	0,059

Table A.3: *Drugs and Medicine*, n = 10.000, k = 100

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
SNN4.1	1:37:33	69	5067	0,296	2,526
SNN4.2	1:37:24	69	5110	0,297	2,554
SNN4.3	1:38:00	69	2143	0,296	2,565
SNN4 avg	1:37:39	69	5107	0,296	2,548
SNN4 sdv	0:19	0	38,109	0	0,020

Table A.4: *Drugs and Medicine*, n = 50.000, k = 224

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
SNN5.1	1:07	23	47	0,081	3,042
SNN5.2	1:07	24	49	0,106	3,007
SNN5.3	1:13	24	50	0,110	3,040
SNN5.4	1:10	23	51	0,075	3,043
SNN5.5	1:14	23	47	0,078	3,035
SNN5 avg	1:10	23	49	0,090	3,033
SNN5 sdv	0:03	0,548	1,789	0,017	0,015

Table A.5: *NOM*, n = 500, k = 31

Tests	Clustering Time	Number of Clusters	Noise Removed	Silhouette Coefficient	Davies-Bouldin Index
SNN6.1	3:27	16	68	0,177	2,965
SNN6.2	3:30	16	68	0,178	2,964
SNN6.3	3:27	16	68	0,178	2,963
SNN6.4	3:26	16	68	0,177	2,941
SNN6.5	3:25	16	68	0,178	2,989
SNN6 avg	3:27	16	68	0,178	2,964
SNN6 sdv	0:01	0	0	0,001	0,017

Table A.6: *NOM*, n = 1.000, k = 50

Tests	Clustering Time	Number of Clusters	Noise Removed	Silhouette Coefficient	Davies-Bouldin Index
SNN7.1	13:10	16	122	0,717	2,908
SNN7.2	13:22	16	122	0,717	2,923
SNN7.3	13:26	16	122	0,717	2,932
SNN7.4	13:15	16	122	0,717	2,913
SNN7.5	13:27	16	122	0,717	2,907
SNN7 avg	13:20	16	122	0,717	2,917
SNN7 sdv	0:07	0	0	0	0,011

Table A.7: *NOM*, n = 2.000, k = 79

Tests	Clustering Time	Number of Clusters	Noise Removed	Silhouette Coefficient	Davies-Bouldin Index
SNN8.1	23:43:23	22	813	0,399	3,102

Table A.8: *NOM*, n = 21.544, k = 387

Tests	Clustering Time	Number of Clusters	Noise Removed	Silhouette Coefficient	Davies-Bouldin Index
Param1.1	5:26	210	1901	0,362	2,788
Param1.2	5:20	199	1777	0,349	2,443
Param1.3	5:18	204	1769	0,361	2,289
Param1.4	5:09	215	1955	0,348	2,964
Param1.5	5:17	206	1787	0,343	3,461
Param1 avg	5:18	207	1838	0,353	2,789
Param1 sdv	0:06	6,06	84,766	0,008	0,462

Table A.9: *Drugs and Medicine*, n = 10.000, k = 25

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Param2.1	5:12	76	1030	0,311	2,644
Param2.2	5:06	68	1021	0,321	2,671
Param2.3	5:15	84	1061	0,330	2,507
Param2.4	5:16	76	1110	0,332	2,416
Param2.5	5:20	77	1150	0,343	2,228
Param2 avg	5:13	76	1074	0,327	2,493
Param2 sdv	0:05	5,67	54,738	0,012	0,181

Table A.10: *Drugs and Medicine*, n = 10.000, k = 50

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Param3.1	5:35	16	309	0,402	2,559
Param3.2	5:38	18	320	0,422	3,293
Param3.3	5:20	14	329	0,419	3,061
Param3.4	5:37	17	309	0,422	2,863
Param3.5	5:24	17	321	0,417	3,475
Param3 avg	5:30	16	318	0,416	3,050
Param3 sdv	0:08	1,52	8,591	0,008	0,359

Table A.11: *Drugs and Medicine*, n = 10.000, k = 150

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Param4.1	5:41	11	199	0,376	2,729
Param4.2	5:24	13	200	0,358	2,896
Param4.3	5:33	10	173	0,387	3,022
Param4.4	5:26	9	181	0,363	3,027
Param4.5	5:32	14	227	0,381	2,673
Param4 avg	5:31	11	196	0,373	2,869
Param4 sdv	0:06	2,07	20,857	0,012	0,164

Table A.12: *Drugs and Medicine*, n = 10.000, k = 200

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Words1.1	4:52	34	823	0,272	1,813
Words1.2	4:48	35	823	0,272	2,079
Words1.3	4:52	36	836	0,272	1,984
Words1.4	4:52	33	818	0,272	1,800
Words1.5	4:56	33	818	0,272	1,746
Words1 avg	4:52	34	824	0,272	1,885
Words1 sdv	0:02	1,304	7,369	0	0,141

Table A.13: *Drugs and Medicine*, n = 10.000, k = 100, words weight = 2

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Words2.1	4:51	28	817	-0,146	1,719
Words2.2	4:59	28	814	-0,146	2,336
Words2.3	4:56	28	814	-0,146	2,515
Words2.4	4:52	27	829	-0,145	2,288
Words2.5	4:59	30	804	-0,140	1,984
Words2 avg	4:55	28	816	-0,145	2,168
Words2 sdv	0:03	1,095	8,961	0,003	0,316

Table A.14: *Drugs and Medicine*, n = 10.000, k = 100, words weight = 5

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Words3.1	5:24	21	809	-0,343	2,526
Words3.2	5:24	21	809	-0,343	2,951
Words3.3	5:34	20	814	-0,342	2,203
Words3.4	5:10	20	823	-0,343	2,173
Words3.5	5:34	20	818	-0,343	2,609
Words3 avg	5:25	20	815	-0,343	2,492
Words3 sdv	0:09	0,548	6,025	0,001	0,320

Table A.15: *Drugs and Medicine*, n = 10.000, k = 100, words weight = 10

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Words4.1	5:33	21	1061	-0,524	9,306
Words4.2	5:48	22	1044	-0,527	19,814
Words4.3	5:43	23	1062	-0,520	6,661
Words4.4	5:44	23	1082	-0,520	7,219
Words4.5	5:53	24	1040	-0,526	12,487
Words4 avg	5:44	23	1058	-0,523	11,097
Words4 sdv	0:07	1,140	16,739	0,004	5,381

Table A.16: *Drugs and Medicine*, n = 10.000, k = 100, words weight = 100

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Words5.1	5:41	22	1074	-0,534	20,119
Words5.2	5:49	22	1097	-0,536	3,976
Words5.3	6:05	18	1017	-0,529	6,205
Words5.4	5:35	20	1051	-0,531	8,415
Words5.5	5:52	22	1074	-0,534	21,493
Words5 avg	5:48	21	1063	-0,533	12,041
Words5 sdv	0:11	1,789	30,237	0,003	8,168

Table A.17: *Drugs and Medicine*, n = 10.000, k = 100, words weight = 200

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Words6.1	6:03	20	1072	-0,535	12,204
Words6.2	6:00	17	1064	-0,522	12,318
Words6.3	5:41	17	1048	-0,523	27,887
Words6.4	5:45	17	1040	-0,523	8,134
Words6.5	5:46	18	1054	-0,524	4,323
Words6 avg	5:51	18	1056	-0,525	12,973
Words6 sdv	0:09	1,304	12,681	0,005	8,968

Table A.18: *Drugs and Medicine*, n = 10.000, k = 100, words weight = 300

Appendix B

Results Mini-Batch K-Prototypes Algorithm

Tests	Clustering Time	Silhouette Coefficient	Davies-Bouldin Index
Proto1.1	1:31	0,297	2,271
Proto1.2	1:46	0,299	2,638
Proto1.3	1:28	0,257	3,709
Proto1.4	1:37	0,342	2,728
Proto1.5	1:30	0,291	3,378
Proto1 avg	1:34	0,297	2,945
Proto1 sdv	0:07	0,030	0,585

Table B.1: *Drugs and Medicine*, n = 100, K = 10, b = 20

Tests	Clustering Time	Silhouette Coefficient	Davies-Bouldin Index
Proto2.1	1:48	0,400	2,161
Proto2.2	1:43	0,359	2,815
Proto2.3	1:46	0,365	3,057
Proto2.4	1:47	0,372	3,338
Proto2.5	1:47	0,367	2,827
Proto2 avg	1:46	0,373	2,840
Proto2 sdv	0:01	0,016	0,435

Table B.2: *Drugs and Medicine*, n = 1.000, K = 32, b = 200

Tests	**Clustering Time**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Proto3.1	3:12	0,289	4,625
Proto3.2	3:09	0,265	4,639
Proto3.3	3:07	0,333	4,407
Proto3.4	3:05	0,292	3,547
Proto3.5	3:07	0,329	4,016
Proto3 avg	3:08	0,302	4,247
Proto3 sdv	0:02	0,029	0,465

Table B.3: *Drugs and Medicine*, n = 10.000, K = 100, b = 2000

Tests	**Clustering Time**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Proto4.1	10:14	0,206	16,120
Proto4.2	10:20	0,202	13,304
Proto4.3	10:35	0,225	10,560
Proto4.4	10:17	0,233	11,591
Proto4.5	10:23	0,233	10,660
Proto4 avg	10:21	0,220	12,447
Proto4 sdv	0:08	0,015	2,330

Table B.4: *Drugs and Medicine*, n = 50.000, K = 224, b = 10.000

Appendix C

Results Incremental Clustering

Tests	Clustering Time	Number of Clusters	Noise Removed	Silhouette Coefficient	Davies-Bouldin Index
Inc1.1	5:17	43	753	0,484	2,974
Inc1.2	5:11	42	764	0,485	2,508
Inc1.3	5:14	42	815	0,490	2,836
Inc1.4	5:17	44	761	0,472	2,520
Inc1.5	5:41	44	809	0,492	2,608
Inc1 avg	5:20	43	780	0,485	2,689
Inc1 sdv	0:12	1	29,203	0,008	0,206

Table C.1: *Drugs and Medicine*, n = 10.000, k = 100, initial set size = 75%

Tests	Clustering Time	Number of Clusters	Noise Removed	Silhouette Coefficient	Davies-Bouldin Index
Inc2.1	5:14	39	563	0,423	3,631
Inc2.2	5:00	42	541	0,450	3,024
Inc2.3	4:54	44	519	0,431	2,982
Inc2.4	4:56	41	572	0,454	3,429
Inc2.5	4:54	40	522	0,464	3,065
Inc2 avg	4:59	41	543	0,444	3,226
Inc2 sdv	0:08	1,924	23,776	0,017	0,288

Table C.2: *Drugs and Medicine*, n = 10.000, k = 100, initial set size = 50%

Tests	Clustering Time	Number of Clusters	Noise Removed	Silhouette Coefficient	Davies-Bouldin Index
Inc3.1	4:19	39	561	0,417	3,895
Inc3.2	4:20	37	576	0,406	5,416
Inc3.3	4:21	37	549	0,426	3,944
Inc3.4	4:17	39	564	0,414	3,679
Inc3.5	4:18	41	639	0,422	2,801
Inc3 avg	4:19	39	578	0,417	3,947
Inc3 sdv	0:01	1,673	35,534	0,008	0,941

Table C.3: *Drugs and Medicine*, n = 10.000, k = 100, initial set size = 25%

Tests	Clustering Time	Number of Clusters	Noise Removed	Silhouette Coefficient	Davies-Bouldin Index
Inc4.1	3:59	35	563	0,381	2,245
Inc4.2	4:05	42	648	0,422	3,270
Inc4.3	4:08	40	563	0,368	2,439
Inc4.4	4:01	36	550	0,380	2,372
Inc4.5	4:09	40	649	0,426	3,978
Inc4 avg	4:04	39	595	0,395	2,861
Inc4 sdv	0:04	2,966	49,490	0,027	0,744

Table C.4: *Drugs and Medicine*, n = 10.000, k = 100, initial set size = 10%

Tests	Clustering Time	Number of Clusters	Noise Removed	Silhouette Coefficient	Davies-Bouldin Index
Inc5.1	3:42	44	976	0,435	3,603
Inc5.2	3:48	40	911	0,430	4,025
Inc5.3	3:46	46	974	0,435	3,813
Inc5.4	3:47	46	917	0,429	5,166
Inc5.5	3:51	44	795	0,430	3,713
Inc5 avg	3:46	44	915	0,432	4,064
Inc5 sdv	0:03	2,449	73,521	0,003	0,635

Table C.5: *Drugs and Medicine*, n = 10.000, k = 100, initial set size = 1%

Appendix D

Results Distributed Clustering

Tests	Clustering Time	Number of Clusters	Noise Removed	Silhouette Coefficient	Davies-Bouldin Index
Distr1.1	2:43	63	938	0,467	2,130
Distr1.2	2:38	69	968	0,475	2,124
Distr1.3	2:41	62	914	0,471	2,751
Distr1.4	2:48	65	909	0,462	2,654
Distr1.5	2:41	65	964	0,474	2,212
Distr1 avg	2:42	65	939	0,470	2,374
Distr1 std	0:03	2,683	27,346	0,006	0,303

Table D.1: *Drugs and Medicine*, n = 10.000, k = 71, m = 2

Tests	Clustering Time	Number of Clusters	Noise Removed	Silhouette Coefficient	Davies-Bouldin Index
Distr2.1	2:42	96	988	0,449	2,522
Distr2.2	2:32	84	1016	0,444	2,534
Distr2.3	2:39	87	949	0,451	2,429
Distr2.4	2:26	91	983	0,441	2,235
Distr2.5	2:21	85	941	0,453	2,353
Distr2 avg	2:32	89	975	0,447	2,415
Distr2 std	0:08	4,930	30,599	0,005	0,125

Table D.2: *Drugs and Medicine*, n = 10.000, k = 58, m = 3

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Distr3.1	2:10	94	1073	0,443	2,596
Distr3.2	2:24	112	1013	0,468	2,549
Distr3.3	2:33	97	996	0,445	2,864
Distr3.4	2:24	110	1052	0,437	2,365
Distr3.5	2:18	114	1058	0,447	2,633
Distr3 avg	2:21	105	1038	0,448	2,601
Distr3 std	0:08	9,209	32,439	0,012	0,179

Table D.3: *Drugs and Medicine*, n = 10.000, k = 50, m = 4

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Distr4.1	0:22	14	44	0,662	2,389
Distr4.2	0:20	17	26	0,618	1,788
Distr4.3	0:38	16	34	0,692	1,320
Distr4.4	0:39	16	38	0,698	1,262
Distr4.5	0:41	15	38	0,609	2,442
Distr4 avg	0:32	16	36	0,656	1,840
Distr4 std	0:10	1,140	6,633	0,041	0,564

Table D.4: *Drugs and Medicine*, n = 200, k = 10, m = 2

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Distr5.1	0:56	30	78	0,523	2,2580
Distr5.2	0:51	26	75	0,508	2,283
Distr5.3	0:53	33	77	0,512	2,282
Distr5.4	0:52	32	76	0,518	2,370
Distr5.5	0:50	26	90	0,507	2,907
Distr5 avg	0:52	29	79	0,514	2,485
Distr5 std	0:02	3,286	6,140	0,007	0,266

Table D.5: *Drugs and Medicine*, n = 2.000, k = 32, m = 2

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Distr6.1	6:45	89	3871	0,394	11,449
Distr6.2	6:38	90	3867	0,413	9,120
Distr6.3	6:32	89	3915	0,407	9,983
Distr6.4	6:25	89	3868	0,404	9,784
Distr6.5	6:11	87	3989	0,395	9,242
Distr6 avg	6:30	89	3902	0,403	9,915
Distr6 std	0:13	1,095	52,631	0,008	0,930

Table D.6: *Drugs and Medicine*, n = 20.000, k = 100, m = 2

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Distr7.1	2:01:34	121	7690	0,218	4,776
Distr7.2	1:58:29	120	7202	0,209	4,715
Distr7.3	1:59:03	120	7307	0,211	4,755
Distr7 avg	1:59:42	120	7400	0,213	4,749
Distr7 std	1:38	0,577	256,859	0,005	0,031

Table D.7: *Drugs and Medicine*, n = 100.000, k = 224, m = 2

Tests	**Clustering Time**	**Number of Clusters**	**Noise Removed**	**Silhouette Coefficient**	**Davies-Bouldin Index**
Distr8.1	50:11:32	214	21620	0,204	8,089

Table D.8: *Drugs and Medicine*, n = 283.713, k = 266, m = 4