Delft University of Technology

Master Thesis

Accelerated Mean Shift for static and streaming environments

Author:
Daniel
van der Ende

Supervisors:
Prof. Dr. Elmar
Eisemann

Dr. Jean-Marc
Thiery

March 28, 2015

Accelerated Mean Shift for static and streaming environments

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Computer Science

by

Daniel van der Ende
born in Bury St. Edmunds, England

Mobile Professionals
Egelantiersstraat 143-1
Amsterdam
http://www.mobpro.com

Computer Graphics and Visualization Group
Department of Intelligent Systems
Faculty of EEMCS
Delft University of Technology
Delft
http://graphics.tudelft.nl

Accelerated Mean Shift for static and streaming environments

Abstract

Mean Shift is a well-known clustering algorithm that has attractive properties such as the ability to find non convex and local clusters even in high dimensional spaces, while remaining relatively insensitive to outliers. However, due to its poor computational performance, real-world applications are limited. In this thesis, we propose a novel acceleration strategy for the traditional Mean Shift algorithm, along with a two-layers strategy, resulting in a considerable performance increase, while maintaining high cluster quality. We also show how to to find clusters in a streaming environment with bounded memory, in which queries can be answered at interactive rates, and for which no Mean Shift-based algorithm currently exists. Our online structure can be updated at very minimal cost and as infrequently as possible, and we show how to detect the time at which this update needs to be performed. Our technique is validated extensively in both static and streaming environments.

Thesis Committee:

Prof. Dr. E. Eisemann	Computer Graphics & Visualization, TU Delft
Dr. ir. R. Bidarra	Computer Graphics & Visualization, TU Delft
Dr. ir. A. Bozzon	Web Information Systems, TU Delft

Preface

Over the past year, I have been working on the Master Thesis that lies before you today. This thesis would not have been possible without significant contributions from a number of people, who I would like to thank for their efforts. In particular, a number of people deserve thanks.

I would like to thank Mobile Professionals for offering me the opportunity to work on this thesis in co-operation with them.

I would like to thank my supervisors from Delft University of Technology. Prof Dr. Elmar Eisemann for his invaluable advice, academic knowledge, and all his efforts to help me. I also want to thank Dr. Jean-Marc Thiery for all his efforts to get this work done, his patience with me, and generally for helping get (and keep!) it all together. Without his support, this Master Thesis would not have been possible.

My family, for their continued support during my studies at Delft University of Technology as a whole, especially during the completion of this Master Thesis (the replacement of a broken laptop midway through the project springs to mind!).

Finally, last but certainly not least, my girlfriend, for putting up with my mini lectures on the topic of data stream mining, which kept me sane during the work on this thesis. Without our brainstorming sessions, this work would not have been possible.

D.J. van der Ende
Delft, The Netherlands
March 28, 2015

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Gaining insight into large, high-dimensional datasets is a very complex task that has received more public attention recently with the rise of Big Data in popular media and culture. The improved capabilities and technological advancements in terms of data storage have resulted in enormous datasets being stored, without any real idea of what may lie within these datasets. The suspicion is, and has always been, that valuable insights may be gleaned with smart analysis, which could help improve a system's performance or add value in some other way to a real-world system.

Because of these new developments and new demands for intelligent data analysis, the data mining field has grown considerably over the past few years. Initiatives like Delft Data Science [1] underline the importance of data mining within academic environments as well. Solutions to data mining vary from adaptations of existing clustering approaches (such as the one proposed in this thesis), to radical innovations in terms of data management and computations using systems like Hadoop. In particular, this thesis focuses on exploratory data mining, where the user wishes to explore the dataset to (hopefully) extract useful information. Because of the size and complexity of many datasets, such data exploration is impossible without clever ways of automated analysis. The work presented in this thesis is one form of such automated analysis.

Apart from the challenges posed by the size of 'modern' datasets, the speed at which the data is often produced also pushes computational power to the limit. Data streams place a number of additional requirements on

data management and analysis systems, resulting in the development of novel methods of dealing with such data environments.

Apart from the management and computational and statistical analysis of large datasets, another important aspect of any data mining system is the interaction mechanism with the end user. In order for a user to be able to quickly and easily interpret, understand, and respond to patterns and structures found in large datasets, the presentation method is vital. For this purpose, many forms of data visualization have been used.

This thesis deals with all three of these areas in some form: data mining for large datasets; data mining for data streams; and visualizing the data mining results. We present a modification of the Mean Shift algorithm for use with very large, high-dimensional datasets in a traditional static context. Furthermore, a smart trigger mechanism is developed that allows us to only recluster when necessary in a data stream environment. The system developed will be presented in this thesis report. A prototype implementation in Python was also developed during this thesis.

1.1 Mobile Professionals

The work in this thesis was partly funded by Mobile Professionals (MobPro). MobPro operates within the online advertising domain, using Real Time Bidding (RTB) to programmatically place advertisements in mobile applications and websites. Here, a brief description is given of RTB, and how this thesis relates to this domain.

1.1.1 Real Time Bidding

Over the past few years, programmatic advertising has increased steadily in popularity. Real Time Bidding is a form of programmatic advertising that connects advertisers with users through a system based on an auction. An overview of the concept is given in Figure 1.1. Generally speaking, one can describe the process of RTB as follows: when a user opens a website or app, information about the page/app and available information about the user is passed to an ad exchange. This ad exchange auctions the opportunity to place an advertisement on the page/app to the highest bidder. The winning

bidder is allowed to display his/her advertisement to the user. This process takes place in a few hundred milliseconds, as it needs to be executed quickly in order to get the user's attention and not to miss this opportunity. There are four main stakeholders that can be identified in this figure:

- **DSP (Demand-Side Platform)**. DSP's represent a number of clients, on whose behalf they bid on particular advertisement opportunities. This may mean, for instance, that a particular brand wants to appear more often to a particular target group. MobPro's RTB platform fits into this category.
- **Exchange/SSP (Supply-Side Platform)**. This stakeholder can be seen as the marketplace where the exchange of goods takes place. The Publisher makes an advertisement opportunity available on an Exchange, on which DSP's can bid to get their client's advertisements shown.
- **Website/Publisher**. Publishers are the parties who, in the end, are selling their screen space to advertisers when users go to their websites or apps.
- **Client (User)**. The end user is any person who opens an app or website and is shown one or more advertisement that is served up through the RTB chain described here.

Each time such an auction takes place, a data point is generated and stored. This data includes features such as bid price, user location, user gender, etc. This information is very valuable, and is often used to specifically target certain users. With better analysis of patterns in this data, it may be possible to further optimize this targeting.

1.1.2 Thesis-Domain relationship

The work in this thesis was done with MobPro's RTB system in mind. As MobPro's RTB platform is a real-time, real-world system, it was not possible to run our algorithm directly on real-time data without considerable implementation effort in order to integrate the code correctly. Instead, this thesis presents a method that can be applied to MobPro's RTB system, and could give valuable insights into interesting patterns in their data. Due to the scale and speed of the dataset, it is impossible to see such patterns with the

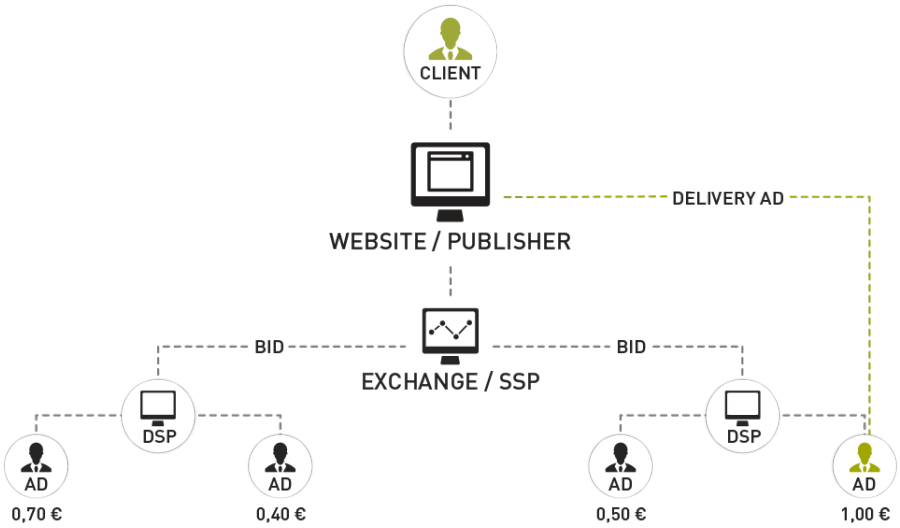


Figure 1.1: Schematic overview of RTB

naked eye, which presents a suitable scenario for an automated data mining approach.

1.2 Research Questions

Combining the concepts of the Mean Shift algorithm, exploratory data mining, and data stream clustering, we formulated two main research questions:

1. How can Mean Shift clustering be effectively applied to large, high-dimensional datasets to be used for exploratory data mining?

2. How can Mean Shift clustering be modified to suit the needs of data stream clustering?

1.3 Contributions

In this thesis, we present an acceleration for Mean Shift, that can be applied in both static and streaming environments. In the static environment, execution time is reduced, while high cluster quality is maintained. Our main

contribution, however, is its application in streaming environments. We derive an efficient triggering mechanism, that determines when an update of our online structure is necessary. Both of these contributions are discussed in Chapter 2.2, and extensively validated in Chapter 2.3.

Our goal regarding a contribution to Mobile Professionals is not to deliver a working system. Rather, our aim is to develop a method in which Mean shift can be applied in high-dimensional streaming environments. By delivering such an academically motivated algorithm, we believe that Mobile Professionals will be able to apply this algorithm and gain the insights they seek.

1.4 Thesis Structure

This thesis is structured as follows. Chapter 2 presents our contributions. This chapter is the core of an article submitted to IARIA Data Analytics 2015. Chapter 3 discusses some other research leads that were investigated during the course of this thesis. In chapter 4 we discuss some limitations of our method, and briefly describe possible avenues of future research. Finally, chapter 5 concludes this work.

Chapter 2

Contributions

The content discussed in this chapter is the core part of our article, which was submitted to IARIA Data Analytics 2015. It appears here in adapted form.

2.1 Background

2.1.1 Preliminaries

Before discussing the details of our method, a number of preliminary concepts should be described. A quick reference is available in the glossary in Appendix A.

A *kernel density estimate* (KDE) of a dataset is an estimate of the density of a dataset using a probabilistic kernel. There are many forms of kernels, with different shapes and characteristics. A KDE is obtained by placing a single kernel of each single data point. Following this, the sum of all kernels for all points is summarized to obtain an estimate of the density for the entire dataset. This construction process is illustrated in Figure 2.1.

Data stream clustering refers to the research field in which clusters are found in a streaming environment. Stream clustering can take different forms, depending on which type of *window model* is used. The aim is to find clusters in the data, as it is streamed in at high speed.

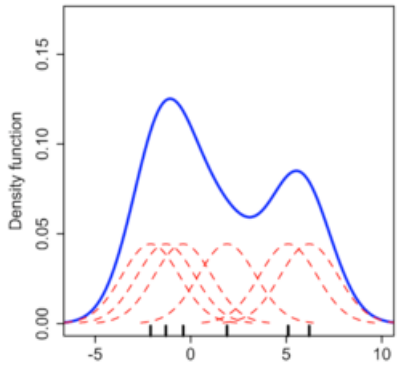


Figure 2.1: Schematic overview of KDE construction. Each black mark along the horizontal axis has a kernel overlaid (dashed red lines). These kernels are them summed to form the kernel density estimate (blue line).

The term *window model* refers to a concept unique to data stream clustering. A window model determines which data to consider when mining the data stream. As data streams tend to be infinite, it is not always feasible to execute a given algorithm over all data ever seen. Not only would this require a massive amount of memory, the execution time would also increase substantially. Various window models exist that are applied in data streaming environments. In this thesis, we have used both landmark and sliding windows. A landmark window includes all data points from a particular time t_0. This time can be defined to be at any point in time during the stream, however, in this thesis a landmark window is always initialized with $t_0 = 0$, i. e., from the beginning of the stream. A sliding window limits the amount of data to a fixed number. This number is predetermined and does not vary during execution. The window 'slides' over the data, streaming in new points and, in order to make space for the new points, streaming out 'old' points.

A *Kd-tree* is a data structure that creates partitions in order to organize points in a k-dimensional data space. In essence, it keeps partitioning the data space until each data points has its own node within the tree. It can be seen as a form of binary space partitioning tree. We use Kd-trees to quickly find the nearest neighbors for a given data point, as shall become apparent in this chapter.

In this work, we often refer to a *triggering mechanism* for streaming environments. By this, we mean the following. Often in stream clustering systems, an online structure is maintained that summarizes the data as it is streamed in. Then, when the user wishes to see a clustering result, a clustering algorithm is executed over these summaries. A triggering mechanism monitors the data stream, and determines whether a clustering is necessary when a user requests a clustering. If the data has not changed substantially since the previous clustering, there is no need to execute a new clustering. If the data has changed, a clustering is 'triggered'.

2.1.2 Mean Shift

2.1.2.1 Overview

Mean Shift is a mode-seeking, density-based clustering technique, with as main parameter a *kernel bandwidth* h describing the scale at which clusters are expected. In this regard, Mean Shift can be seen as a natural multi-scale clustering strategy.

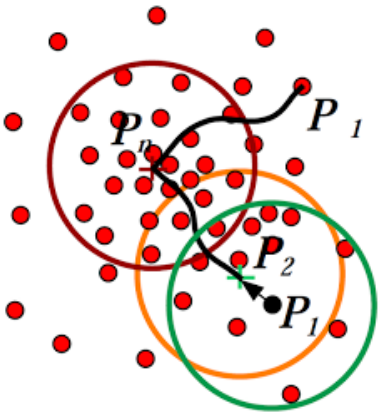
Considering an input data set $\mathcal{P} = \{\mathbf{p_i}\}$ in dimension d and a kernel density estimate K, a Mean Shift clustering of $\mathcal{P}$ is obtained as follows: For every point $\mathbf{p_i}$, initialize $p_i^0 = \mathbf{p_i}$ and iteratively compute p_i^{k+1} from p_i^k by performing a gradient ascent of the kernel density estimate. Upon convergence of this process $p_i^\infty = \bar{p}_i$, where $\bar{p}_i$ is a local maximum of the kernel density estimate. Points of $\mathcal{P}$, which converge towards the same local maximum are then clustered together. Fig. 2.2a gives a schematic overview of this process.

It should be noted, that the underlying geometric structure of the clusters is of course dependent on the kernel that is used. In particular, changing the bandwidth of the kernel results in more or fewer local maxima of the resulting kernel density estimate. Fig. 2.2 illustrates this fact.

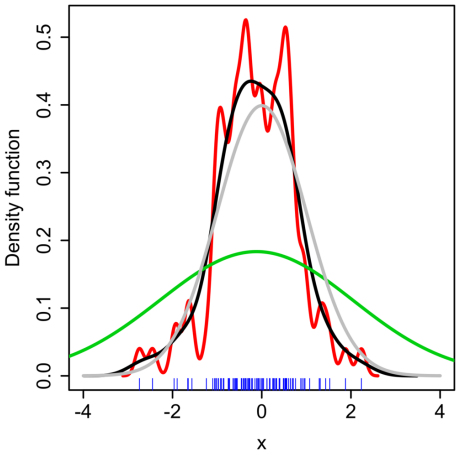
A variety of kernels has been used in the literature, the most common of which is the traditional Gaussian kernel (with *bandwidth* $h \in \mathbb{R}$):

$$K(x, p) = \pi^{-d/2} \exp(-||x-p||^2/h^2) \tag{2.1}$$

Note, that this isotropic kernel is sometimes replaced by an anisotropic Gaussian kernel described by a symmetric positive matrix H (i. e., $\exp(-||x-p||^2/h^2)$ is replaced by $\exp(-(x-p)^{\mathrm{T}} \cdot H \cdot (x-p))$). However, if the anistropy of the kernel is uniform (i. e., $H(x) = H$ regardless of the location x), then



(a) Each point performs an adaptive, iterative gradient ascent of the density towards a local maximum. The local density is estimated by a fixed-size kernel density estimator.



(b) Various bandwidth values result in different local maxima in the estimated density (red, grey, black). Blue points are distributed according to the density in green.

Figure 2.2: Mean Shift overview process

these two approaches are completely equivalent, up to a global rigid transformation of the data.

Indeed, because H is symmetric definite positive, it can be decomposed as $H = U^{\mathrm{T}} \cdot \Sigma \cdot U$, U being a rotation matrix and Σ being a diagonal matrix with positive entries, which describe the different anisotropic scales of the kernel. Then, by noting $\sqrt{\Sigma}$ the diagonal matrix with squared scales ($\sqrt{\Sigma}^{\mathrm{T}} \cdot \sqrt{\Sigma} = \Sigma$), it is easy to verify that $(x-p)^{\mathrm{T}} \cdot H \cdot (x-p) = ||(x'-p')||^2$, with $y' := \sqrt{\Sigma} \cdot U \cdot y$ for every point y.

It is therefore entirely equivalent to use a uniform anisotropic kernel on the input data and use a uniform isotropic kernel on data that has been globally transformed through the rigid transformation $y' := \sqrt{\Sigma} \cdot U \cdot y$. Note, that traditionally, principal component analysis (PCA, [21]) is a common strategy to first transform the input data before applying Mean Shift clustering.

In our work, we will thus only focus on the case of isotropic Gaussian kernels.

2.1.2.2 Bandwidth estimation

The user may not always have an idea of what bandwidth to use, in the context of data which is difficult to explore, visualize and understand, such as high-dimensional data.

There are a great number of bandwidth estimation techniques [9,10,18,23] providing results commonly accepted by the scientific community as *intrinsic* to the data. In this respect, Mean Shift can then be seen as a non-parametric clustering method.

In our work, for datasets with non-provided bandwidth, we will use Silverman's rule of thumb [18]. This rule of thumb is commonly used in real-world applications. It is based on the asymptotic mean integrated squared error (AMISE) of a kernel density estimate. Using this, we can obtain the following expression for an estimate of the bandwidth:

$$h_{AMISE} = \left(\frac{R(K)}{nR(f'')(\int x^2 K)^2} \right)^{\frac{1}{5}} \tag{2.2}$$

Silverman's rule of thumb is obtained by assuming the data is normally distributed, thus enabling us to replace the unknown part in the above equation, giving us:

$$h = (\frac{4\hat{\sigma}^5}{3n})^{1/5} \approx 1.06\hat{\sigma}n^{-1/5} \tag{2.3}$$

A full derivation of Silverman's rule of thumb is available in Jones et al. 's survey [18]. Due to the assumption of normally distributed data, Silverman's rule of thumb's accuracy varies depending on the true distribution of the data. If the data is normally distributed, the bandwidth estimate will be (near) optimal. However, if the data distribution is not normal, the accuracy of the bandwidth estimate depends on how different the actual distribution is compared to the normal

2.1.2.3 Performance

Although Mean Shift has many attractive properties, such as its ability to find non-convex clusters and its multiscale nature, it also has some limitations and issues. The most important of these is its performance. As Fashing et al. [13] have shown, Mean Shift is a quadratic bound maximization algorithm whose performance can be characterized as being $O(kN^2)$, where N is the number of points, and k is the number of iterations per point.

Many modifications to Mean Shift have been proposed [7, 12, 14–16, 24].

Carreira-Perpiñán [7] identify two ways in which Mean Shift can be modified to improve performance: 1) Reduce the number of iterations, k, used for each point, 2) Reduce the cost per iteration. As Carreira-Perpiñán demonstrates, both of these techniques have their own merits and issues. Another class of Mean Shift modifications is that of data summarization, followed by traditional Mean Shift on a summary of the original data. Our algorithm falls into this category. This is an approach that other acceleration strategies have also applied [14, 16]. Further details on this will be given in Sec. 2.2.

2.1.3 Data Stream Clustering

A number of authors have assessed the complexity of mining data streams [4, 5]. Barbara [5] focused on data stream clustering, listing a number of requirements: 1) Compactness of representation, 2) Fast, incremental processing of new data points, 3) Clear and fast identification of outliers. Due to the nature of streams, time is very limited. Because of this, data stream clustering algorithms need to be able to respond extremely quickly to the changes that occur over time in the dataset, often called concept drift. Moreover, because of the often huge datasets, memory is also constrained. Our approach has both attractive time and memory use characteristics, as will be discussed in Sec. 2.2.

Many stream clustering algorithms use a two-phased approach. The approach centers on an online phase, which summarizes the data as it is streamed in, and an offline phase, which executes a given clustering algorithm on the summaries produced. The summaries are generally referred to as micro-clusters, and due to a number of attractive properties can be updated as time progresses and new data is streamed in (and old data is streamed out). CluStream [2] maintains q micro-clusters online, followed by a modified k-means algorithm that is executed when a clustering query arrives. DenStream [6] is similar, exchanging the k-means algorithm for DBSCAN, and distinguishing various quality levels of micro-clusters. Finally, D-Stream [8] uses a sparse grid approach. These three algorithms all have complex parameters. Moreover, when a clustering query arrives, a full clustering algorithm execution always takes place.

2.2 Method

2.2.1 Static Clustering

As discussed in Sec. 2.1.2.3, there are several ways in which previous work has improved Mean Shift. Our algorithm aims to reduce the number of input points for the Mean Shift. This is achieved by first discretizing the data space using a sparse d-dimensional regular grid, with a cell size of the order of the bandwidth (coarser discretizations lead to artifacts, from our experience). For each grid cell C_i, the number of points n_i assigned to it is maintained, along with the sum of these points. This enables the computation of an average position of the points within the cell, denoted $\overline{C_i}$. We then simply cluster the cells $\{C_i\}$ by applying the Mean Shift algorithm over them, using $K_{\overline{C}}(p, C_i) = n_i K(p, \overline{C_i})$ as the underlying kernel. This is exactly equivalent to computing the Mean Shift over all input points, after having set each point to the center of its cell. Although extremely simple, this strategy proved robust and efficient during our experiments. Furthermore, it allows us to run Mean Shift over infinitely growing datasets with bounded memory, as long as the range of the data remains bounded, which is a property which is required in streaming environments.

2.2.2 Stream Clustering

For stream clustering, the most common methodology is a two-phased approach, as discussed in Sec. 2.1.3. A major disadvantage of this approach is that a when a user query arrives, the offline clustering algorithm is executed over the data summaries, regardless of whether the dataset has changed significantly since the previous execution of the offline phase. This leads to unnecessary execution of expensive clustering algorithms.

Our algorithm aims to avoid such needless clustering algorithm execution by accurately detecting when the data has changed sufficiently to warrant a new clustering. This is achieved by fast, effective analysis of the data currently being considered. It should be noted that this approach can be used, regardless of the type of window used. In this thesis, we have applied both landmark and sliding windows of various sizes.

First, when the stream clustering is initialized, a static clustering, as described in Sec. 2.2.1 is performed. This clustering will serve as our initial reference clustering. On each stream iteration, we evaluate whether the data

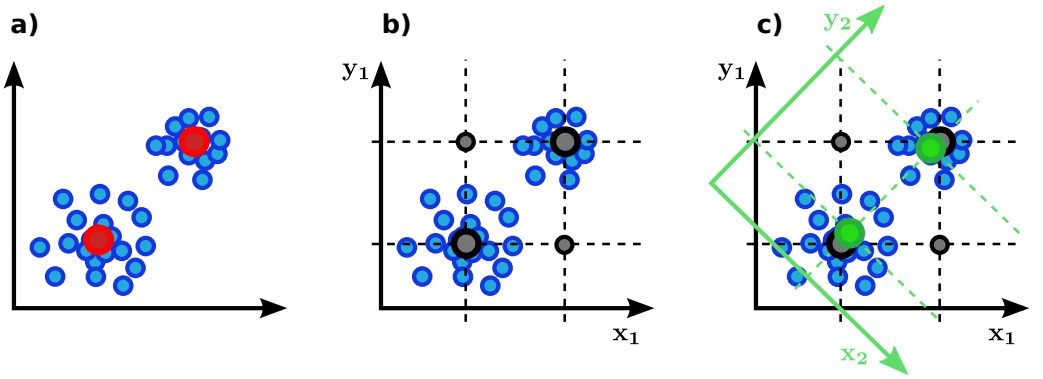


Figure 2.3: **a):** Two clusters in $2D$, with their centers in red. **b):** Clusters in each dimension (dot lines), whose products badly approximate the $2D$ clusters. **c):** Consensus over additional axes helps to identify the $2D$ clusters from the product of the $1D$ clusters of the projected data.

distribution has changed sufficiently compared to this reference clustering to require a reclustering. If so, a clustering is executed and the result is considered the reference clustering for future iterations. By only executing the clustering algorithm, Mean Shift in our case, when necessary, a great amount of execution time is saved. Querying the cluster for a point is then done by finding the closest cell average and retrieving its cluster index (this requires an acceleration structure such as a Kd-Tree, which was already computed at the Mean Shift step).

We base our trigger mechanism on the monotonicity lemma defined by Agrawal et al. [3] as:

Lemma 2.2.1 (Agrawal)**.** *If a collection of points S is a cluster in a k-dimensional space, then S is also part of a cluster in any $(k-1)$-dimensional projections of this space.*

Following this lemma, if any of the k-dimensional clusters change, the $(k-1)$-dimensional subclusters should also change.

We make use of this point and set up a collection of low-dimensional *data observers* (in our case, $1-$dimensional), which we can update efficiently when adding or removing points from the structure, and which will trigger a reclustering of the structure when the underlying data distribution has changed in such a way that the clustering may also have changed. Our algorithm is mainly parametrized by the chosen distribution of observers as

well as by their *sensitivity*.

Each of the observers i is defined as a histogram $\mathbf{H_i}$ of the data projected onto an axis $\mathbf{a_i}$. Because high-dimensional data is likely to overlap in separate dimensions (see Fig. 2.3), we consider not only the original canonical axes $\{\mathbf{e_k}\}$, but also randomly distributed axes in $\mathbb{R}^d$, and we will overcome the induced approximation by defining the final decision for the reclustering as a consensus over the observers. Since we cannot make any assumptions over the data that will be streamed-in (e. g., we cannot compute a PCA to define axes), we simply create a random set of pairs of indices $(k_1, k_2) \in [1, d]^2$ and define the additional axes as $(\mathbf{e_{k_1}} + \mathbf{e_{k_2}})/\sqrt{2}$ and $(\mathbf{e_{k_1}} - \mathbf{e_{k_2}})/\sqrt{2}$ (we thus *intricate* the canonical dimensions (k_1, k_2), see Fig. 2.3**(c)**).

When a clustering is performed, each histogram is saved as $\bar{\mathbf{H}}_\mathbf{i}$. On each subsequent stream iteration, data points are added to the grid (or removed from it if a time-dependent window is used), all histograms are updated, and we determine if the stream iteration has significantly altered the data distribution, in which case we need to update the clustering.

We define the measure between histograms $\bar{\mathbf{H}}_\mathbf{i}$ and $\mathbf{H_i}$ as their Jensen-Shannon divergence:

$$D_{JS}(\bar{\mathbf{H}}_\mathbf{i} \parallel \mathbf{H_i}) = \frac{1}{2} D_{KL}(\bar{\mathbf{H}}_\mathbf{i} \parallel M) + \frac{1}{2} D_{KL}(\mathbf{H_i} \parallel M) \tag{2.4}$$

where $M = \frac{1}{2}(\bar{\mathbf{H}}_\mathbf{i} + \mathbf{H_i})$, and $D_{KL}(P \parallel Q)$ is the Kullback-Leibler divergence between histograms P and Q:

$$D_{KL}(P \parallel Q) = \sum_k P(k) \ln \frac{P(k)}{Q(k)} \tag{2.5}$$

This measure is a distance, which is symmetric and always defined (the direct use of the Kullback-Leibler divergence between $\bar{\mathbf{H}}_\mathbf{i}$ and $\mathbf{H_i}$ would result in $+\infty$ in cases where points would be removed from a cell, i. e., $Q(k) = 0$ in Eq. 2.5).

A histogram i *votes* for a reclustering if $D_{JS}(\bar{\mathbf{H}}_\mathbf{i} \parallel \mathbf{H_i}) > \epsilon$ (ϵ defines the sensitivity of the observers, and is our main input parameter).

A reclustering is then performed if the proportion of histograms voting for a reclustering is larger than a random variable which we take between 0 and 1. This procedure is a standard Montecarlo voting scheme, which will never (resp. always) trigger a reclustering if no (resp. all) histograms vote for it, and which will trigger a reclustering with probability defined by the consensus among the observers.

The procedure described above is easily maintainable in a streaming environment, as its maintenance only requires removal and addition of points to histograms, which can take place very quickly. Moreover, the discretization of the data space bounds the memory use in such a way that very large datasets and data streams can succinctly, but accurately be stored and used.

2.3 Empirical Results

2.3.1 Metrics

We have computed a number of cluster validation metrics, some of which are discussed in the work of Meila et al. [19]. In our work, the following metrics have been used: Jaccard Index, Rand Index, Fowlkes-Mallows Index, Precision, Recall, F-Measure. These metrics are based on pair-wise comparison of points of a reference clustering A and a comparison clustering A'. All metrics assess whether A' correctly classified the relation between the points in each pair. We use these 6 metrics to quantify our results instead of simply picking one, because there is no real consensus on what is the correct metric between clusterings. A complete description of each of these metrics (including equations) is given in Appendix B. Furthermore, the metrics we chose are common in the data clustering scientific community and will hopefully provide a real insight into the behavior of our algorithm to the reader. A value of 0 indicates completely different clusterings whereas a value of 1 indicates identical ones.

For the static clustering experiments, we compared the traditional Mean Shift and our modified algorithms on the input data points (with the same input bandwidth).

For the stream clustering experiments, the metrics show the deviation between the clustering of the cell averages $\{\overline{C_i}\}$, when using the triggering mechanism or instead updating the clustering every time.

The reason for this choice (comparing clusterings of the averages instead of the original points) is simply practical: we could not run the computation of these metrics on huge datasets for every stream step in a reasonable amount of time. Fortunately, the depicted errors are over-conservative: the true errors are actually lower than the ones we show. Indeed, consider the case of false classification of a new point in a clustering of $100k$ points: it will have a minor impact on the metrics as it is an outlier in the data, however it will create a new grid average in our coarse summarization grid and will therefore result in computed errors (based on the averages) which are much higher.

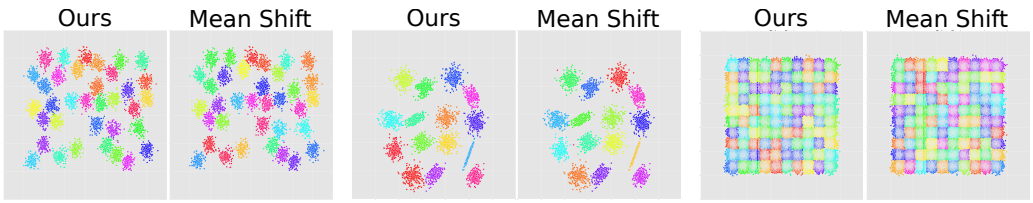


Figure 2.4: Comparison of clusterings between Mean Shift and our approach in 2D

2.3.2 Static Clustering

In order to evaluate our algorithm's performance, a large number of datasets were used. For each dataset, we compare our method with the traditional Mean Shift. Fig. 2.4 shows a comparison between our approach and traditional Mean Shift. Most metric values have a value of the order of 0.99. While there are some minor differences, these are cause by points which are at the boundaries between visible clusters or outliers which are at equal distance to several clusters. In general, higher errors occur for datasets presenting a high variability of the range over its various dimensions (see the remark on the equivalence between isotropic and anisotropic Mean Shift in Sec. 2.1.2.1).

We have conducted experiments in higher dimensions. Although visually comparable, it is difficult to even assess the correctness of the Mean Shift clustering by projecting the data on a 2D space, due to overlap in the visualization. Table 2.1 summarizes our results on various datasets commonly found in the scientific literature. These datasets challenge our approach in various ways, with varying cluster numbers, cluster overlap, dimensionalities, dataset sizes, and other characteristics.

While we experienced a reasonable gain in performance for small to reasonably big datasets, this is of minor importance. Rather, we want to emphasize that our approach produces results which are consistent with the traditional Mean Shift algorithm once the bandwidth has been set by the user. This is the most important part of the validation, as it indicates that our approach can be used for Mean Shift clustering in a streaming environment, with potentially infinitely growing data. Note that, by construction, the error which is introduced by our approximation decreases with the size of the datasets on which it is used, while its efficiency obviously increases drastically.

	d	N	M1	M2	M3	M4	M5	M6
A1	2	3000	0.95	0.97	0.99	0.97	0.97	0.97
A2	2	5250	0.96	0.98	0.99	0.98	0.98	0.98
A3	2	7500	0.96	0.98	0.99	0.98	0.98	0.98
S1	2	5000	0.99	0.99	0.99	0.99	0.99	0.99
S2	2	5000	0.95	0.98	0.99	0.98	0.97	0.98
S3	2	5000	0.87	0.93	0.99	0.95	0.91	0.93
S4	2	5000	0.85	0.92	0.99	0.94	0.90	0.92
Birch 1	2	100000	0.91	0.95	0.99	0.95	0.95	0.95
Birch 2	2	100000	0.64	0.78	0.95	0.76	0.99	0.78
Birch 3	2	100000	0.95	0.97	0.99	0.97	0.97	0.97
Dim 3	3	2026	0.99	0.99	0.99	0.99	0.99	0.99
Dim 4	4	2701	0.99	0.99	0.99	0.99	0.99	0.99
Dim 5	5	3376	0.99	0.99	0.99	0.99	0.99	0.99
D5	5	100000	0.99	0.99	0.99	0.99	0.99	0.99
Abalone	8	4177	0.99	0.99	0.99	0.99	0.99	0.99
D10	10	30000	0.99	0.99	0.99	0.99	0.99	0.99
D15	15	30000	0.99	0.99	0.99	0.99	0.99	0.99

Table 2.1: Summary of static clustering results. d: dimension. N: total number of points. M1: Jaccard Index. M2: Fowlkes-Mallows Index. M3: Rand Index. M4: Precision. M5: Recall. M6: F-Measure

	N (tot.)	d	ϵ	n (init.)	δn	Window	**TC**
a) Cov	$581k$	7	0.001	$25k$	$1k$	$25k$	NO
b) Cov	$581k$	7	0.003	$25k$	$1k$	$25k$	NO
c) Synth.1	$1M$	2	0.003	$50k$	$1k$	$50k$	NO
d) Synth.2	$1M$	2	0.003	$50k$	$1k$	$50k$	YES
e) Synth.3	$1M$	2	0.003	$50k$	$1k$	NO	YES

Table 2.2: Statistics of the experiments conducted in streaming environments. N: total number of points. d: dimension. n: number of points for the initial clustering. δn: number of points added at each stream step. Window: number of points of the sliding window (if used). **TC**: time-coherency of the data stream.

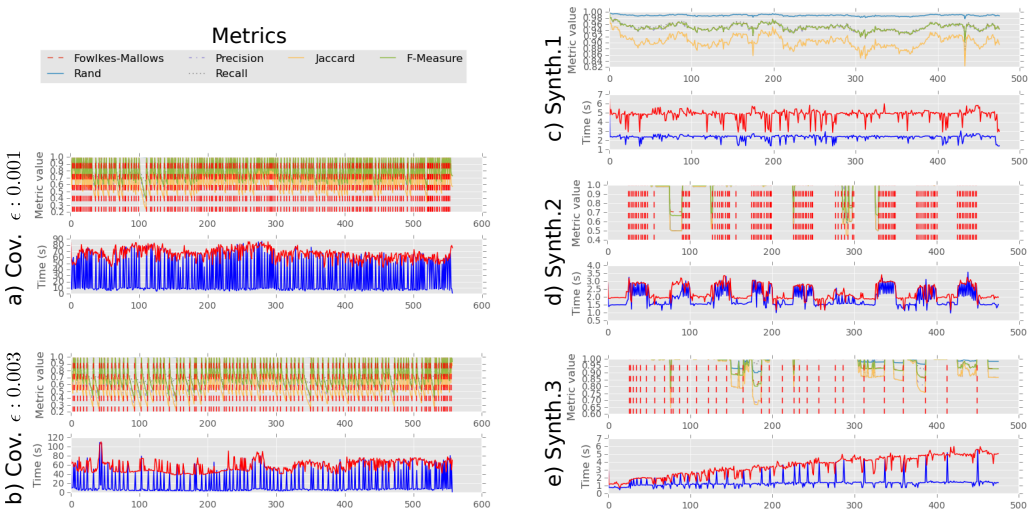


Figure 2.5: Comparison between our triggered clustering and constantly updated clustering on various datasets. For the timings, the red curve indicates the computation time per stream iteration when no triggering is used, and the blue curve indicates the one when our triggering mechanism is used.

2.3.3 Stream Clustering

For the stream clustering validation, we compare the results obtained when running our approach with the reclustering trigger enabled and disabled (i. e., reclustering on every stream iteration, regardless of lack of changes in the data distribution).

We show results on datasets of dimension 2 and 7, with a varying value of ϵ, with a fixed time window (with removal of old points) or not (adding points only), and with time-coherent or time-incoherent streaming of the data (i. e., whether the data is streamed in a structured way). Please note that "time-coherency" refers to the fact that points which are streamed in successively are roughly expected to belong to the same cluster. Table 2.2 summarizes the statistics of the conducted experiments, and the metrics plots for these test runs are presented in Fig. 2.5. The 2-dimensional dataset is a synthetically generated set, with Gaussian blob clusters. The 7-dimensional dataset is the CoverType dataset [11], which is used throughout literature for empirical evaluation of stream clustering algorithms.

One interesting aspect with regards to our reclustering triggering is the value of ϵ which is used to threshold the Jensen-Shannon divergence value.

For the CoverType dataset (dimension 7, $581k$ points), two test runs with identical configurations were performed, with $\epsilon = 0.001$ (Fig. 2.5 (a)), and with $\epsilon = 0.003$ (Fig. 2.5 (b)). The initial clustering was both times performed with $25k$ points, which is also the length of the sliding window that was used, and data points were streamed by sets of $1k$ points.

A lower value for ϵ should result in more frequent reclustering triggers, in an attempt to maintain a higher level of clustering quality. Although hard to see due to the high number of reclusterings, it is clear from these images that the lower ϵ affects the triggering mechanism. Reclusterings are more frequent and generally cluster quality is kept at a higher level. It should be noted that this dataset is also an example of a failure case of our algorithm, but also of the Mean Shift algorithm and any bandwidth-dependent algorithm. From the clustering results it is clear that the bandwidth estimate is completely incorrect. The bandwidth for this dataset was computed to be approximately 105.39. However, the CoverType dataset is not normally distributed, and the range of the data over the various dimensions varies from 1 to 109. Note that, on this dataset, a standard grid approach similar to ours [22] provided results with metric values of 0.2.

For other experiments, the value of ϵ was set to $\epsilon = 0.003$.

Experiment Synth.1 (Fig. 2.5 (c)) was done with a dataset of $1M$ points in dimension 2, with a sliding window of $25k$ points, with $1k$ points added at each stream and for time-incoherent streamed data. We observe that no reclustering is ever performed for this experiment. However, the metrics we obtain over time consistently remain over 0.7, which indicates that the initial clustering we had was good enough for the whole streaming session. Note that 0.7 is roughly the metrics values for which a reclustering was decided in previous experiments under similar conditions ($\epsilon = 0.003$), which indicates that the a-posteriori errors resulting from a given value of ϵ are consistent over the experiments.

On the other hand, experiment Synth.2 (Fig. 2.5 (d)), which was conducted under similar conditions than experiment Synth.1 with the sole difference of streaming time-coherent data, presents highly structured reclustering events. The reclustering events correspond to the appearance and disappearance of complete clusters, and it is obvious that our triggering mechanism adapts in a non-trivial way to the structure of the underlying data.

Note that, for real-life datasets, the reality corresponds probably to a mix of these two behaviors (i. e., there are several levels of consistency in data, e. g., for visited websites during the day or in various places over the

world, etc.). The strength of our approach is that we make no assumption on the structure of the data which is going to be streamed in, and that it adapts automatically to its underlying structure.

Finally, experiment Synth.3 was performed on a dataset of $1M$ points, without window (i. e., no points are removed from the structure over time). It is visible that the time for updating the structure grows almost linearly over time, while the frequence of the triggering events is actually inversely linear over time, which is the behavior which is to be expected in order to provide timely-bounded analysis of growing data. Of course, there is a limit to this, and it is impossible to guarantee this behavior for arbitrarily distributed data (over space and/or time).

Chapter 3

Additionally followed research leads

Before settling on the approach that was discussed in chapter 2, a number of other research leads were investigated. These attempts were ultimately unsuccessful but are still of interest to discuss here. For each approach, we discuss the initial intuition, the algorithm, and finally the results, and why the approach did not produce the results we hoped for.

Our ambition on starting this project was to create a stream clustering algorithm that applied similar concepts to Mean Shift, without requiring any of the expensive iterative steps of the Mean Shift. Mean Shift has a complexity of $O(n^2)$. Some acceleration methods have reduced this to $O(n \log n)$. The goal for our approach for static situations was linear performance in the size of our structure. Moreover, the goal for streaming situations was to have constant performance at every stream step, i. e., $O(1)$, and allow a user to get feedback to a query as quickly as possible (preferably almost real-time). The reason for these goals was that Mobile Professionals' Real Time Bidding (RTB) systems produce such high-speed data streams, and the goal was to enable near real-time analysis of these streams.

3.1 Full Dimensional Grid

Intuition

Our initial intuition was that, in order to be able to handle very large datasets, some form of discretization would be necessary. We quickly opted for a grid-style discretization. This would allow for much faster treatment and handling of all data points, because all points in a cell would be aggregated in some manner. If we used a Kernel Density Estimate (KDE) to estimate the density, we could determine per cell which direction led to the fastest gradient ascent. This would almost simulate the gradient ascent concept upon which Mean Shift is based. Also, having such a direction per cell would allow immediate determination of the final mode for each cell, and therefore each point, by chaining the cells into a sequence of directions. Such a chaining would not completely simulate a gradient ascent, but would produce a similar path as points converge towards their mode. Furthermore, we thought that such an approach would also be suited to streaming environments. When new points arrived, we could add these points to the grid, and follow the chain of directions to find its mode. This process would be very quick and thus usable for data streams.

Algorithm

Initially, the data space was discretized into a full-dimensional, regular grid. The number of cells was fixed to arbitrary numbers, but various values were attempted to see how this affected results. Each cell was assigned a direction, based on the direction of the steepest gradient of the KDE in each cell. This resulted in a grid of directions, automatically pointing towards the modes and allowing immediate determination of cluster centers, without any iterative phase, such as required by Mean Shift. Moreover, adding new points to the structure was very simple, and because the grid immediately indicated a direction (and consequently the final mode) for each cell, clustering in a streaming environment would indeed be $O(1)$. An example of this idea is shown in Figure 3.1.

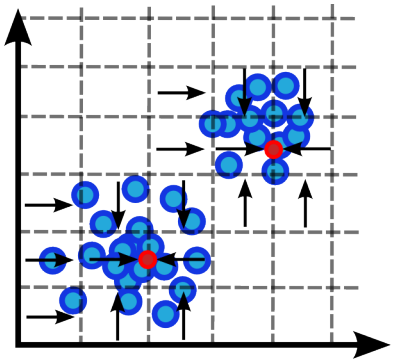


Figure 3.1: Full Dimensional Grid approach. Every populated cell has a direction assigned to it, based on the KDE's gradient. Places where two cells 'point towards each other' are marked as cluster centers (red circles). It can clearly be seen that a lot of cells are empty, but still are created and stored.

Results

This idea was implemented and functioned well in low dimensional (e.g., 2 and 3 dimensional), static situations. However, due to the fact that the grid was not sparse and regular, the memory requirements grew exponentially as the dimensionality increased. An out-of-core solution to this problem, where data was stored on disk, was attempted, in an effort to resolve these issues, but this was not feasible. The design of the algorithm at this time was also such that a different type of grid (i.e., a sparse grid, or non-regular grid) would not have worked, as the following of the directions to the modes would not have been possible. Because the static environment presented such problems already, no work was done for the streaming environment for this method.

3.2 Accelerated Mean Shift

Once it was clear that a full-dimensional, regular grid would cause memory consumption problems, the focus shifted from completely simulating the Mean Shift without any iterative phase to an accelerated Mean Shift with fewer iterations. This would make the Mean Shift suitable for a streaming environment. In terms of performance, our ambition was still to achieve $O(1)$ response time at each stream step in a streaming environment. However, for

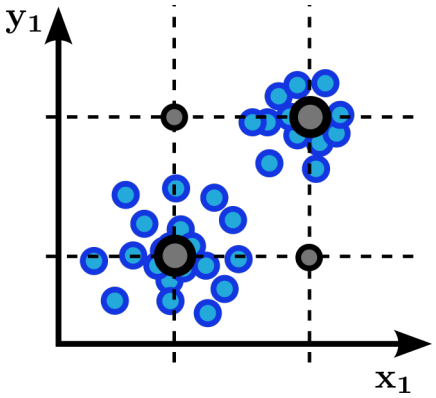


Figure 3.2: Schematic illustration of the incorrect clusters when 1-dimensional clusters are used to construct d- dimensional clusters.

static environments, the goal of achieving $O(n)$ performance was no longer feasible.

Intuition

Instead of using a full-dimensional, fully formed regular grid, our intuition was that combining clusters found in each separate dimension could avoid the previously encountered memory problems. This was based on the monotonicity lemma defined by Agrawal et al. [3]. This lemma is still used in our final approach, and is mentioned in section 2.2, but is re-iterated here for clarity:

Lemma 3.2.1. *If a collection of points S is a cluster in a k-dimensional space, then S is also part of a cluster in any $(k-1)$-dimensional projections of this space.*

This lemma made it possible to 'construct' d-dimensional clusters from 1-dimensional clusters found in each dimension. These 1-dimensional clusters were defined as the local maxima in the KDE. The algorithm developed with this lemma is discussed in the following.

Algorithm

Agrawal et al. [3] used the monotonicity lemma to create the CLIQUE clustering algorithm. The main issue was its creation of spurious clusters,

which is inherent to this method of d-dimensional cluster finding. This can be clearly seen in Figure 3.2, where four clusters are found if the 1-dimensional clusters are used to construct the 2-dimensional clusters. Two of these four clusters are in fact incorrect. These clusters can therefore only be treated as candidate clusters. Agrawal et al. resolve this issue by thresholding each candidate cluster using a pre-defined density threshold value. This threshold value is very difficult to determine, as it is very dependent on the dataset and on the grid cell size in use.

In our algorithm, cluster finding in each dimension was based on finding the modes in the KDE for each dimension, where the KDE was evaluated for each grid cell (i. e., for each grid cell compute the KDE value, use these values to find the local maxima). This, therefore, involved estimating a bandwidth value for each dimension (i. e., a bandwidth value is necessary to compute a KDE). In this way, we believed, we could mitigate some of the problems caused by bandwidth estimation in datasets where the range of data in the dimensions varied greatly. Once the modes in each dimension's KDE were found, we initially wanted to compute the full-dimensional clusters from these 1-dimensional clusters, but quickly realised that this number would exceed memory capacity. Moreover, the problem with the spurious clusters would still be present.

In order to select the correct clusters from the initial set of candidates, a filtering step was implemented. This step involved finding the closest cluster in each dimension for each data point. For each data point, we can then construct a d dimensional cluster because we know the closest 1-dimensional cluster for each dimension. A pseudocode representation of this filtering algorithm is shown in Algorithm 1. The reason for considering each dimension separate was to avoid the computation of all full dimensional cluster candidates, as this number may be extremely large. This is due to the fact that the full dimensional clusters are formed by computing the cartesian product of all 1-dimensional cluster candidates. This filtering step was reasonably effective, as it removed a large portion of the unnecessary cluster candidates.

Once this filtering step had completed, the remaining cluster candidates were used as seeds for the Mean Shift. Using seeds for the Mean Shift means we don't iteratively shift each data point based on the KDE. Rather, we

Algorithm 1 Pseudocode for candidate filtering

```
1: procedure CandidateFiltering
2:     X ← set of data points
3:     D ← set of dimensions
4:     for x_i ∈ X do
5:         for d_j ∈ D do
6:             Find nearest cluster c_k in d_j
7:         Build full dimensional-cluster C from c_k values
```

define a number of points (the seeds) that are to be shifted towards the modes based on the overall KDE. In a sense, we provide Mean Shift with an 'educated guess' of where clusters may be located, resulting in fewer iterations. Moreover, because the number of remaining cluster candidates was considerably lower than the original number of datapoints, this approach also reduced the number of points to be shifted. However, not all spurious candidates were removed. Moreover, this filtering was computationally expensive, as it required $n * d$ nearest neighbor computations (even though these were only in 1 dimension). The most problematic aspect of this filtering was its requirement of processing all data points again, resulting in multiple passes over the data, which generally is not feasible in stream clustering algorithms. Any cluster candidate for which at least 1 point was the closest candidate was kept as possible cluster. This resulted in a large number of cluster candidates. Moreover, these candidates were rigidly placed on grid, often causing issues depending on the grid cell size used. Also, in some cases, single clusters had multiple candidates, due to local maxima in the 1-dimensional density estimates.

Following this realization we attempted to reduce the number of cluster candidates, and perhaps even remove the need for the iterative phase of the Mean Shift, by performing a merge of the cluster candidates. This merging of initial cluster candidates took three forms. An overview of these strategies is shown in Figure 3.3.

1. **Within dimension merging** As a Mean Shift clustering relies on the bandwidth h to cluster data, a merge of any 1-dimensional candidates within a distance of h was performed. Our intuition here was that by performing such a merge before applying Mean Shift, we would remove

many unnecessary candidates, for which the iterative Mean Shift then would no longer take place. (Figure 3.3a)

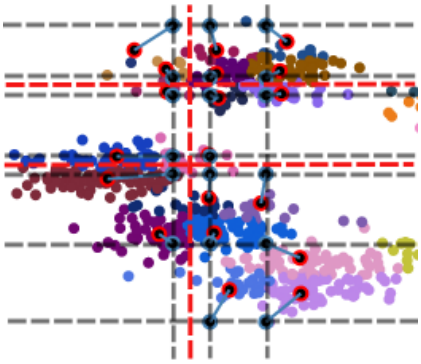
2. **Average Merging** This attempt involved the merging of d-dimensional candidates within a distance h of each other. This would, in a sense, simulate the first step of the Mean Shift algorithm. By merging remaining candidates, the number of candidates that would be used as seeds for Mean Shift would be reduced. For each candidate, the nearest neighbors within a radius h were found. For all points within this radius, an average position was computed, where each candidate was weighted by the number of points attracted by that candidate. (Figure 3.3b)

3. **Discard Merging** Closely related to average merging. Instead of computing the average position of all candidates within the radius and weighting these candidates by their point counts, we took the position of the candidate with the highest point count and discarded all other candidates. Our intuition was that the average merging may compute an average position that resides in a local minimum of the overall KDE, thus making it a poor estimate of a cluster. (Figure 3.3c)

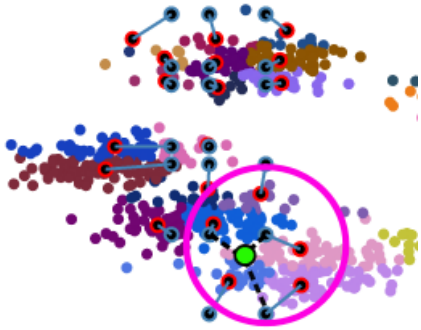
Some thresholding mechanisms were also attempted, in order to identify at least the major clusters. These thresholds focused on removing cluster candidates that attracted very few points (in addition to removing ones that attracted no points at all). This approach showed some promise, but proved to be very difficult to implement in a generic fashion so that acceptable performance was achieved regardless of dataset characteristics or properties. Attempts included incorporating dynamic thresholds, based on the datasets quartiles, but this too was too rigid. The main reason for this, of course, is that datasets differ in their cluster size, i. e., for one dataset a group of 10 points may be a very large cluster, whereas for another these points would be considered outliers.

Results

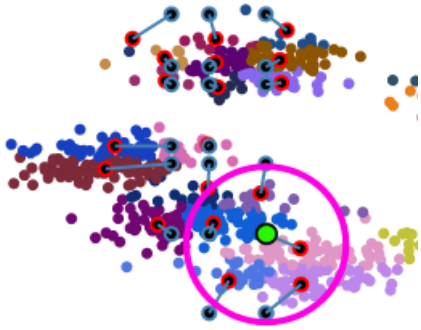
As indicated, the initial set of candidates was huge (too big to fit into memory), so we applied a filtering step. Apart from the fact that this filtering was expensive, it also did not reduce the number of candidates to the number of



(a) Within dimension merging. The red lines indicate the merged candidate lines, based on the neighboring grey lines. This causes a great number of candidates to be merged, leading to problems with higher-dimensional clusters. Only the applicable (grey) grid lines have been drawn, i. e., where candidates were found.



(b) Average merging. The three blue-edged circles within the pink circle (radius) are merged into the green circle.



(c) Discard merging. We see that the final position of the green circle is very similar to that of the average merging.

Figure 3.3: Comparison of the candidate merging techniques attempted.

clusters, as many clusters had multiple candidates. In an attempt to resolve this, we tried three merging strategies. For each of these, we briefly analyze why they did not work.

1. **Within dimension merging** This did not work, due to the fact that a merging of two 1-dimensional candidates influenced a great number of D-dimensional candidates because of the use of the Monotonicity Lemma (e. g., removing a candidate at $x = 1$ influenced all cluster candidates with this x value).

2. **Average Merging** Unfortunately, this merging did not reduce the number of candidates sufficiently to have only clusters remaining. Moreover, the candidates that did remain were not accurate approximations of the cluster centers that would result from running Mean Shift on the dataset. The main reason for the failure of this approach was the fact that in some cases, peripheral cluster candidates from two different clusters would be merged. This would thus result in joining these clusters together, when they clearly should not be. Moreover, this method assumed that there was relatively little overlap between the clusters. If clusters do overlap, candidates could easily be within a bandwidth distance of each other, thus triggering a merge step. These problems are illustrated in Figure 3.3b. Also, because candidates were incorrectly merged across clusters, Mean Shift was unable to correct these mistakes. If inaccurate candidates were presented, the iterative shifting would eventually resolve the issues. However, when candidates were incorrectly merged, this was no longer possible.

3. **Discard Merging** This method had similar problems to those experienced for the average merging approach. The results were actually almost identical to the average merging approach.

Another issue with this approach was its sensitivity to the bandwidth value. Often, particularly in cases where clusters lay 'behind' each other when viewed from the perspective of a dimension, the bandwidth estimate was not sensitive enough to discover these minor modes. In an attempt to resolve these issues, multiple scalings and bandwidth estimations were attempted, but we were unable to make this approach generically applicable. Moreover, this dependence on the bandwidth made it difficult to allow for non-spherical clusters, as it resulted in merging any candidates within a radius, thus often

splitting elongated clusters. We also tried using histograms, as these are essentially a very sensitive KDE (depending on their cell size). These gave better results, but we still had issues with number of candidates and the reduction of this number. The improvement in results did, however, lead us to use histograms in all following approaches.

3.3 Connected Candidates Graph

Intuition

Upon analysis of the positions of the candidates initially found, we noted that for many clusters, multiple candidates were defined. Moreover, because the position of these candidates was often such that a merging operation would not combine these candidates for a given cluster, we started searching in different directions. Rather than trying to directly merge, and thus reduce the number of candidates, our intuition was that perhaps we would be able to somehow connect related candidates with each other. Connecting these nodes in a clever way might lead us to create a skeleton-like structure for each cluster. This intuition is worked out further in this section.

Algorithm

Given the positioning of the candidates in the previous approach, the intuition for the next approach was that we might be able to connect these candidates, based, in order to form a skeleton-like graph for each cluster. Therefore, our approach was to build a number of graphs, with the candidates as nodes. By connecting all candidates that should be in the same cluster, we believed we could accurately assess where clusters were in the data space. The main challenge would of course be how to determine whether or not to create an edge.

Our first step was to shift all candidates to the average position of the points attracted by each candidate. This was similar to a first Mean Shift iteration. This immediately made the candidate positions less rigid and grid-like and offered a more natural representation of the data. These averages are visible in Figure 3.3 as red edged circles. Moreover, it gave us valuable information regarding the position of each cluster candidate relative to the

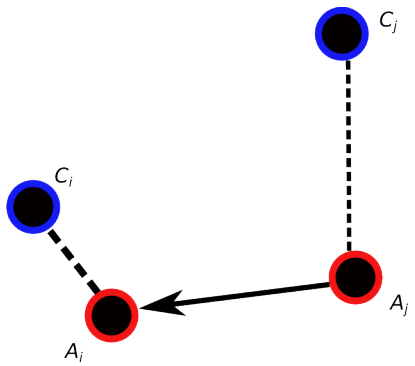


Figure 3.4: Two original candidates, C_i and C_j, with their respective averages, A_i and A_j. We see that the distance between A_j and C_j is much larger than that between C_i and A_i. Therefore, i is the more stable node. Thus, the first approach creates the edge from A_j to A_i

points it attracts. In many cases, we noted that candidates located in very dense areas of the data space hardly shifted at all, meaning its initial position was a good approximation for the average position of its points. For more peripheral candidates, we observed that this shift was far more pronounced. Following this, we attempted to create a directed graph, with the candidates as nodes. For each candidate, we considered its k nearest neighbors, and evaluated whether or not we should create an edge between a candidate and each neighbor. By varying the value of k, we could determine how large the graphs should be, i. e., larger values for k would lead to more edges, thus more nodes per graph, and therefore fewer graphs. Various methods for determining when to create an edge were attempted.

- It was noted that for many original candidates, c_i, the distance between c_i and its average a_i differed, depending on the location of the original candidates. Often, candidates that were on the edges of clusters shifted a considerable distance, while c_i values near a cluster center did not shift very far to their average a_i. We defined this as candidate stability, where the instability of a point i is defined by the $I(i) = d(c_i, a_i)$, or in vector form: $I(i) = \|\mathbf{c_i} - \mathbf{a_i}\|$ and initially used this to connect the nodes. We applied the heuristic to only create a directed edge between two candidates' averages, if the origin node was less stable than the target node. Figure 3.4 shows an example of a stability decided edge.

- An extension to this approach was attempted, in which each node was assigned a budget, B_i for creating edges. Each edge was therefore also assigned a cost, $s_{i,j}$. For the budget of each candidate, we defined a formula that punishes unstable candidates, by allowing them fewer edges. This makes sense, as we expect stable candidates to be in densely populated areas in the data space, probably near the center of the cluster. Such stable candidates, therefore, should be allowed more edges, as they function as a 'crossroads'.

$$B_i = \exp(\frac{-|c_i - a_i|}{h})T \tag{3.1}$$

This formula weights a given threshold T based on a candidate's stability. If a candidate is more stable, its B value will be higher and vice versa. Now to define the cost per edge. Our goal was to connect two averages a_i and a_j if we believe these candidates are in the same cluster. One indication of this would be if the direction of the initial shift from c_i to a_i was continued by the edge between a_i and a_j. Figure 3.5 illustrates this. The value of T is set to depend on the bandwidth, h, and the number of neighbors considered, k: $T = k * h$. This makes sense, as a graph in which more neighbors are considered will automatically also require more edges, and thus a higher budget B per node. In order to capture this intuition, we make use of the dot product of two vectors. By definition, the dot product of two opposite vectors is negative. We define the following two vectors:

$$\mathbf{A} = \mathbf{a_j} - \mathbf{a_i} \text{ and } \mathbf{B} = \mathbf{a_i} - \mathbf{c_i} \tag{3.2}$$

It is known that the dot product of two Euclidean vectors can be defined as:

$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\|\|\mathbf{B}\| \cos\theta \tag{3.3}$$

Using this equation, we can define the following to get an expression for the angle θ_i between the two vector $\mathbf{A}$ and $\mathbf{B}$:

$$\cos(\theta_i) = \frac{\mathbf{AB}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{(a_j - a_i)(a_i - c_i)}{\|a_j - a_i\|\|a_i - c_i\|} \tag{3.4}$$

Due to the inherent characteristics of the cosine, we now have an equation that is 1 when the two vectors are completely aligned, 0 when they

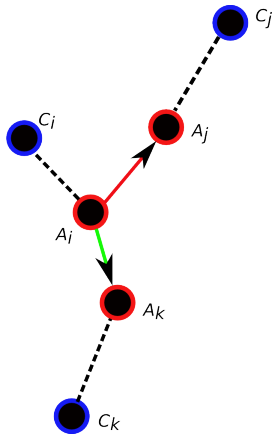


Figure 3.5: Schematic overview of the use of dot products in edge weighting. If we examine candidate i, we see that the edge between a_i and a_k (green edge) lies in a similar direction as the original shift from c_i to a_i, i. e., it is continuation of this shift almost. The edge between a_i and a_j (red edge) travels partially in the opposite direction.

are orthogonal, and -1 when they are completely opposite. Beside the direction, we also take the distance between two averages a_i and a_j into account when assessing whether or not to create an edge. If an edge would be relatively short, it is more likely to be a 'good' edge. This is because it more likely for candidates that are very close to each other to be part of the same cluster. Thus, we define the following cost per edge for each edge between candidates i and j:

$$s_{i,j} = (2 - \cos(\theta_i))d(a_i, a_j) \tag{3.5}$$

Using the definition of the cosine we now see the following:

- If the two vectors are fully opposite (i. e., the cosine terms evaluates to -1), the first term will become 3
- If the two vectors are orthogonal (i. e., the cosine terms evaluates to 0), the first term will become 2
- If the two vectors are fully aligned (i. e., the cosine terms evaluates to 1), the first term will become 1

Moreover, we wish to weight each edge by its distance, where a smaller distance should be considered 'better'. This equation thus evaluates

to small values for attractive' edges, and high values for 'unattractive' edges. These equations were then used as budget and cost functions in order to determine for each node which edges to create, if any, depending on its available budget.

Results

This method was promising initially, as it seemed to create graphs in such a way that clusters were clearly identified. In general, for all approaches attempted, one of the biggest problems was the unpredictability and instability of the positioning of the original candidates c_i. In Figure 3.6, an example is shown of a situation that was optimal. Each cluster has one or two very stable candidates in the center, and a number of peripheral, more unstable, candidates that will be connected to the two central, stable candidates. Figure 3.7 shows an example of a more problematic dataset, in which the candidate distribution was different. As is clear from the figure, this dataset is considerably more dense, with a lot more overlap of clusters. The connectivity of the graphs could be varied, by specifying how many neighboring nodes to consider when determining which edges to create, but we were unable to stabilize this in such a way that it would function adequately for all (types of) datasets. Further modifications of this approach were attempted, taking candidate density, candidate point count, and other properties of the candidates into account, but these suffered from similar issues.

Moreover, the configuration of the value of k was difficult. Our intuition led us to believe, based on the visualizations of the candidates, that 3 or 4 may be suitable values for k. However, for various datasets, these values gave very different results, due to the very different distributions and characteristics of these datasets. Figure 3.8 illustrates this. This image shows identical clusterings, where only the k value was varied. This leads to very large changes in connectivity.

Modifications to the node budget B and cost per edge $s(i, j)$ were also attempted, but these provided inconclusive results. Our initial equations were most intuitive and provided the best results. The effect of the value for k was even more pronounced here, as it also affected the budget of each node.

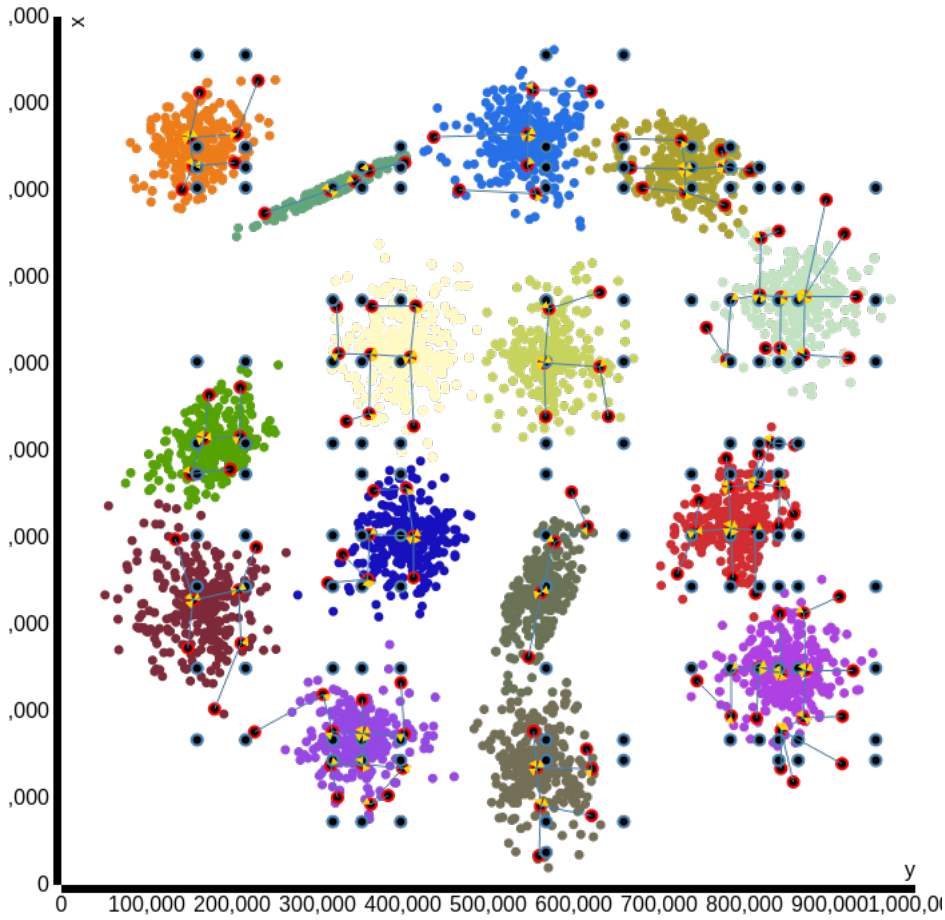


Figure 3.6: Example of a candidates graph result. The red-edged circles represent the shifted candidates (i.e. after step 1). The blue-edged circles are the original candidates.

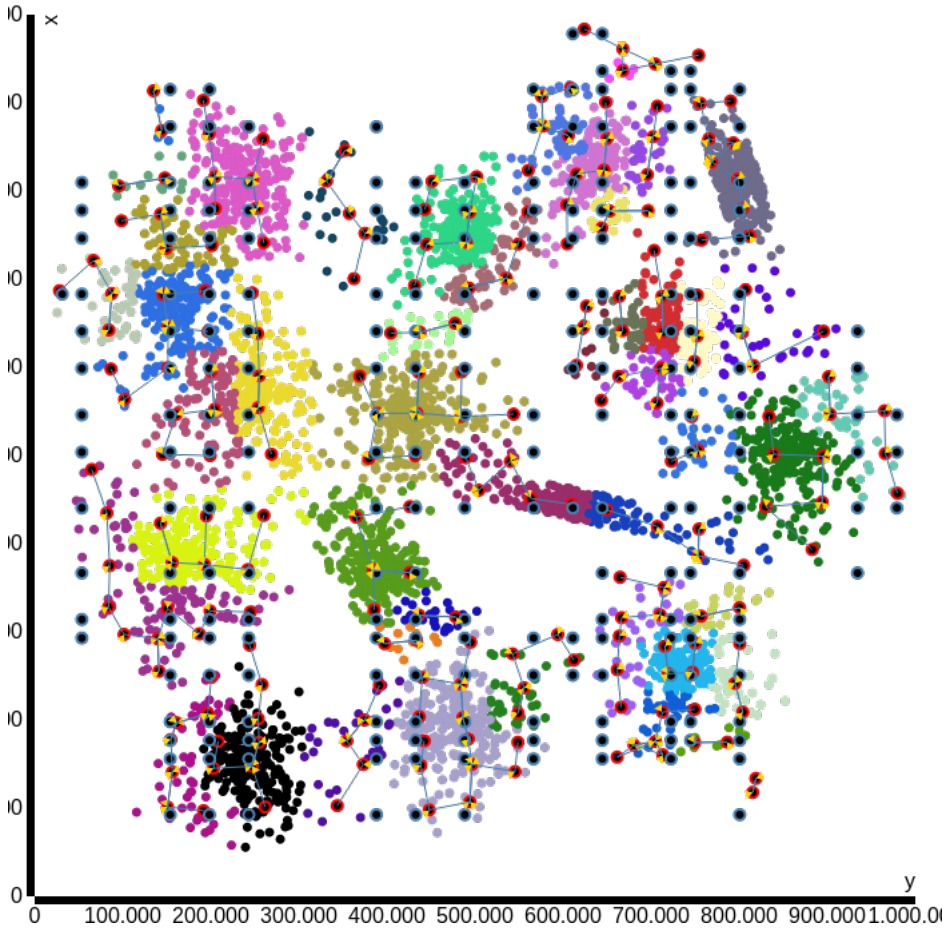
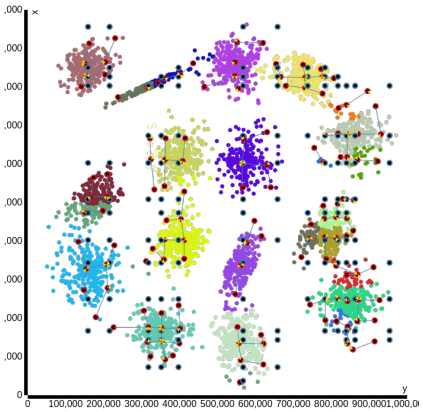
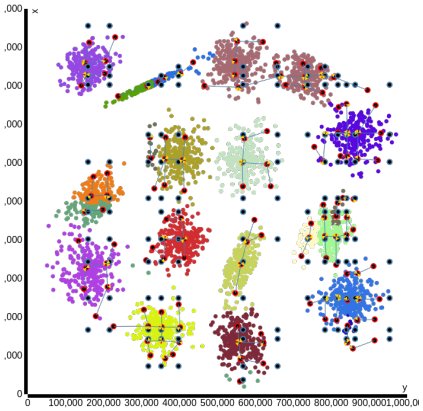


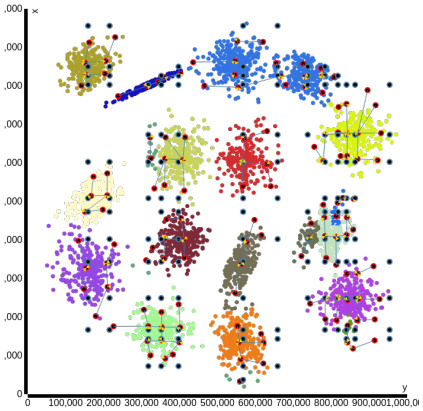
Figure 3.7: Example of a poor candidates graph result. The red-edged circles represent the shifted candidates (i.e. after step 1). The blue-edged circles are the original candidates.



(a) Candidate graph clustering with $k = 4$



(b) Candidate graph clustering with $k = 5$



(c) Candidate graph clustering with $k = 6$

Figure 3.8: Overview of various k settings. We see that the k value has a significant effect on results

Chapter 4

Discussion

The research questions formulated in chapter 1 provided a clear description for the goals of this project. In this chapter, we discuss the results of the thesis as a whole, some known limitations, and possible directions for future work.

In this thesis, we have presented a novel way of applying Mean Shift in a very challenging data stream environment. The challenges posed by such environments, the most significant of which are the limited time and limited memory, led to a modification of the Mean Shift algorithm and the development of a triggering mechanism that allows us to avoid clustering too often. Both of these contributions have been extensively empirically evaluated in section 2.3. These results show good performance in both static and streaming environments, thus enabling the use of the computationally intensive Mean Shift on very large, streaming datasets.

For clarity, we re-iterate our main research questions here, and discuss how these have been answered.

1. How can Mean Shift clustering be effectively applied to large, high-dimensional datasets to be used for exploratory data mining?

2. How can Mean Shift clustering be modified to suit the needs of data stream clustering?

We have shown a modification of Mean Shift, that uses a discretization step to allow execution over very large, high-dimensional datasets. By construction, our method scales very well in both dimensionality and dataset size, as the number of populated grid cells will not increase significantly for large datasets. Moreover, the use of a triggering mechanism in a two-phased stream clustering approach allows the use of our modified Mean Shift in a streaming environment. Our online structure can be maintained and updated very quickly, and when a cluster query arrives, we assess whether a re-clustering is needed compared to the reference clustering. Because of this, in many cases, our approach can give a user an instant response when a clustering is requested.

Because of the nature of our triggering mechanism, we believe it may be possible to apply similar logic for other stream clustering algorithms. Algorithms such as CluStream [2], DenStream [6], and D-Stream [8] all are two-phased approaches, where a clustering is always executed when a user cluster query arrives, regardless of whether this clustering is necessary to maintain cluster quality. Although our approach uses Mean Shift as clustering algorithm and no other algorithms have been examined, our triggering mechanism can be easily applied using other clustering algorithms. Some minor modifications may be needed in order to efficiently apply our reasoning, but we believe this is an interesting avenue for future research.

Although no real-world prototype of a stream clustering-based, exploratory data mining system was developed for the RTB system of MobPro, we believe that our method is suitable for such an environment. Due to time and scope constraints of this thesis, such a prototype was not developed. However, we have developed our own GUI system, that can be used for other datasets than that of MobPro. We therefore believe that an adaptation of our algorithm is a distinct possiblity and may give MobPro valuable insight into the patterns in their dataset. Such an implementation would be best done by the development team at MobPro, given their expertise and experience with development in the RTB domain and their RTB system.

In chapter 3, a number of research leads were discussed, which were ultimately unsuccessful during the course of this thesis. However, based on results obtained, which were presented in chapter 3, these research leads still

hold some promise. Particularly the construction of the connected candidate graph approach is interesting, despite the fact that we were unable to tune the edge weighting functions to function adequately on various types of datasets. One of our recommendations is therefore that further work be done in this direction, as such directed graphs would provide a nice, intuitive representation of clusters that could be updated in a streaming environment.

Despite the high cluster quality and the low execution times reported in section 2.3, a number of limitations of our approach should be discussed. One of these limitations is its sensitivity to the bandwidth. Such issues are not unique to our method, however. All density estimate based clustering approaches suffer from similar issues. Further investigation into better bandwidth estimation techniques may result in a mitigation of this issue. Another possible avenue of research would be the maintenance of multiple grids, using different bandwidth values. These multiple grids could then be used to assess the data distribution using very different perspectives.

Another limitation of our approach, which applies in particular to static environments, is its inability to deal with data sparseness. In very sparse datasets, the acceleration is nullified, because the distance between two points i and j, $d(x_i, x_j)$ is too large, thus resulting in each point being assigned to a separate grid cell. As our grid is dependent on the bandwidth value, this issue is closely related to the problem of incorrect bandwidth values, discussed inthe above. If each data point is assigned to its own grid cell, our method is equivalent to running the Mean Shift on the original set of data points. We do note, however, that the phenomenon of data sparseness mainly appears in relatively small datasets. For larger datasets, such sparseness becomes increasingly improbable.

Hinneburg et al. [17] proposed a clustering algorithm based on cutting hyperplanes, which split the data space into disjunct spaces. This splitting of spaces allowed the authors to create an irregular grid, which was then used for clustering. Such cutting hyperplanes may be useful in separating the clusters in our approach too. By splitting the data space into disjunct subspaces, it may be possible to apply our triggering mechanism on each separate subspace. In such a way, the entire data space would not be reclustered if a single subspace changed. Moreover, it would allow for greater

parallelism, as subspaces could be clustered simultaneously. Currently, our approach is single-threaded, which means we do not take advantage of a lot of the capabilities and features of modern, multi-core processors.

Overall, we have seen that in general a discretization step is necessary when dealing with large, high-dimensional datasets. Without such discretizations steps, the amount of data values possible becomes immensely large, particularly as the dimensionality increases (also known as the curse of dimensionality). The grid approach that was applied throughout all attempts in this thesis (i. e., regular), however, is somewhat limited. Alternative types of discretization have been performed in the past successfully by a number of authors [17, 20]. However, these approaches often create a dynamic grid, that is determined by the data distribution. In streaming environments, this data distribution inherently changes considerably over time, meaning the discretization structure would also have to be adapted. To our knowledge, no dynamic discretization structure for the acceleration of the Mean Shift algorithm has been updated in such a manner in a streaming environment. Moreover, a regular, fixed grid is conceptually simpler to deal with.

Based on our experiences with the construction of d-dimensional clusters from 1-dimensional ones, we can conclude that although some indication of cluster locations can be gleaned from these results, there is a considerable amount of noise. Methods developed in the past, that use this concept, [3,20] always applied some filtering step after this construction. Particularly in higher-dimensional datasets, the number of candidates generated becomes problematic. From our experience, simple filtering steps, such as thresholding or merging of these candidates, have proven inadequate in dealing with the variety of datasets, both in terms of size and dimensionality as in terms of distribution. Even after filtering the initial set of candidates, many spurious candidates remained in the data space, causing problems when trying to assign points to clusters.

Our approach, discussed in chapter 2, uses a sparse, regular grid. Such a structure is relatively memory efficient. Moreover, it allows easy modification of the desired accuracy via the cell size. Other sparse grid-based approaches, such as the one implemented in the Scikit-Learn library [22], also show good performance. Because of the way our structure is defined, however, our grid

is suitable for streaming environments, which require fast, efficient updating of the online structure. Given the empirical results, presented in section 2.3, we see that this approach works, on various types, shapes, and sizes of datasets.

Chapter 5

Conclusion

In this thesis, we present an acceleration for the Mean Shift, and a triggering mechanism that can detect when to re-cluster in a streaming environment. The experiments performed have shown that our algorithm produces accurate clusterings, at reduced cost, and only when absolutely necessary to maintain cluster quality. Moreover, our triggering mechanism allows Mean Shift to be applied in a streaming environment, which, to our knowledge, has not been achieved before.

We achieve the acceleration in static environments by discretizing the data space using a sparse grid, and applying Mean Shift only on the average positions in each grid cell. Details of this are presented in section 2.2.1. The empirical results (section 2.3) obtained confirmed the acceleration achieved, and also confirmed that cluster quality was maintained at a very high level.

For streaming environments, we developed a two-phased approach, similar to other work in the field of data stream clustering. Our online structure is easily maintained, flexible, and offers high cluster quality. Moreover, our approach includes a method to detect when a re-clustering is necessary, based on changed in the data distribution. This too is extensively validated in section 2.3.

Section 2.3 thoroughly evaluates our method, using well-known cluster quality metrics to assess performance. Various datasets, with different characteristics, are used to evaluate the static clustering performance. We also

show that the triggering mechanism correctly determines when a re-clustering is necessary, and that it can be configured to suit a user's needs in terms of cluster quality.

In chapter 4, we propose a number of improvements for our method, based on the limitations encountered. Two of these are related to aspects that were outside of the scope of this thesis, namely bandwidth estimation and data sparseness. Our other recommendation for future work is to use cutting hyperplanes in order to allow for parallel computations within the data space.

In terms of real-world application to the domain of RTB for Mobile Professionals, a working prototype was not developed. This was also not one of the main goals of this thesis. Rather, the research presented in this study can serve as a basis for integration with their systems. The reason for this lack of a prototype is mainly due to the complexity of the systems in question, as it would have required a considerable amount of time to integrate our algorithm correctly and efficiently. We leave this to the development team at Mobile Professionals, whose expertise lies in such integration. We believe that an application of our algorithm would allow fast, almost real-time, insight into emerging patterns, and may alert MobPro to problems or issues as they arise, instead of after the fact. Moreover, analysis of their data may aid MobPro find interesting patterns in their data, that may allow them to optimize their targeting algorithms, or optimize the amounts paid for advertisement placements.

Appendix A

Glossary

Kernel Density Estimate (KDE) An estimate of the density of a dataset using a probabilistic kernel. A KDE is obtained by placing a single kernel of each single data point. Following this, the sum of all kernels for all points is summarized to obtain an estimate of the density for the entire dataset.

Data stream clustering refers to the research field in which clusters are found in a streaming environment. The aim is to find clusters in the data, as it is streamed in at high speed.

Landmark Window Includes all data points from a particular time t_0. This time can be defined to be at any point in time during the stream, however, in this thesis a landmark window is always initialized with $t_0 = 0$, i. e., from the beginning of the stream.

Sliding Window Limits the amount of data to a fixed number. This number is predetermined and does not vary during execution. The window 'slides' over the data, streaming in new points and, in order to make space for the new points, streaming out 'old' points.

Kd-tree Data structure that creates partitions in order to organize points in a k-dimensional data space. It can be seen as a form of binary space partitioning tree.

Triggering mechanism Often in stream clustering systems, an online structure is maintained that summarizes the data as it is streamed in. Then, when the user wishes to see a clustering result, a clustering algorithm is executed over these summaries. A triggering mechanism monitors the data stream, and determines whether a clustering is necessary when a user requests a clustering. If the data has not changed substantially since the previous clustering, there is no need to execute a new clustering. If the data has changed, a clustering is 'triggered'.

Silverman's Rule of Thumb Bandwidth estimation technique used in this thesis.

$$h = (\frac{4\hat{\sigma}^5}{3n})^{1/5} \approx 1.06\hat{\sigma}n^{-1/5} \tag{A.1}$$

Appendix B

Clustering Evaluation Metrics

All of the clustering evaluation metrics used are based on the pair-wise comparison of points between a reference clustering A and a comparison clustering A'. For each pair of points (x_i, x_j), we assess whether the two clusterings agreed on whether the points should be in the same cluster or not. The possible outcomes of this are defined as follows:

1. True Positive (TP). Both clusterings agree that the points x_i and x_j should be in the same cluster.

2. False Positive (TP). Comparison clustering A' indicates points x_i and x_j are in the same cluster, whereas reference clustering indicates this is not the case, i. e., x_i and x_j are in different clusters.

3. True Negative (TP). Both clusterings agree that the points x_i and x_j should be in different clusters.

4. False Negative (TP). Comparison clustering A' indicates points x_i and x_j are in different clusters, whereas reference clustering indicates this is not the case, i. e., x_i and x_j are in the same cluster.

These relations are used to compute various metrics. Each of these metrics' values range from 0 to 1, where 0 indicates completely different clusterings, and 1 indicates identical clusterings. The metrics used were chosen based on the survey presented by Meila et al. [19]. We believe these analyze the clustering results from multiple perspectives, and thus provide a good overview of clustering performance. The equations for each metric are presented in the following.

Jaccard Index

$$\text{Jaccard} = \frac{TP}{TP + FP + FN} \tag{B.1}$$

Rand Index

$$\text{Rand} = \frac{TP + TN}{TP + FP + FN + TN} \tag{B.2}$$

Fowlkes-Mallows Index

$$\text{Fowlkes-Mallows} = \sqrt{\frac{TP}{FP + TP} * \frac{TP}{TP + FN}} \tag{B.3}$$

Precision

$$\text{Precision} = \frac{TP}{TP + FP} \tag{B.4}$$

Recall

$$\text{Recall} = \frac{TP}{TP + FN} \tag{B.5}$$

F-Measure

$$\text{F-Measure} = \frac{2 * Precision * Recall}{Precision + Recall} \tag{B.6}$$

Appendix C

Implementation

This appendix discusses some details on the implementation of the algorithm discussed in this thesis. The source code for this project is available on GitHub [1]. The algorithm was implemented from scratch in Python, without any existing infrastructure. A number of libraries were used during the development, which are also dependencies for the source code:

- Python 2.7.5
- NumPy 1.9.1
- SciPy 0.14.0
- SciKit Learn 0.15.2
- PyMongo 2.7.2

The choice for Python was mostly due to its excellent data mining and data exploration reputation and the number of mature libraries that facilitate such activities. Moreover, MobPro's RTB platform is built in Python. Therefore, by developing our algorithm in Python, we allow for easier integration into this system. We used the implementation of Mean Shift provided by Scikit-Learn as reference implementation. However, the implementation of Mean Shift was not completely correct, resulting in quite some work having to be done to correct this library code to execute the real Mean Shift. These errors in the library implementation were mainly related to the way points

[1] https://github.com/danielvdende/Medusa

were assigned cluster labels. The Mean Shift algorithm iteratively shifts a point towards a peak in the density, and marks all points whose iterative ascent ends close to each other with the same label. In this way, a cluster is intuitively defined based on the data density. However, the Scikit-Learn implementation labels each data point based on the Euclidean distance to the final cluster centers. This means that the actual iterative shifting of the point along the steepest gradient only is used to find the cluster centers, and not to label the data points.

Beside the core algorithm, a GUI was developed to display the clustering results, both in terms of scatterplots and metrics, and in order to allow easy, on-the-fly configuration and setup of the algorithm. This GUI was developed with Python's default TkInter library, which allows it to function cross-platform without any modifications. A screenshot of this GUI running under Ubuntu 14.04 can be seen in Figure C.1.

Although all three Mean Shift related algorithms that can be executed through the GUI are Mean Shift related and based, each has its own distinct implementation, which is optimized and modified to suit the unique aspects of these algorithms.

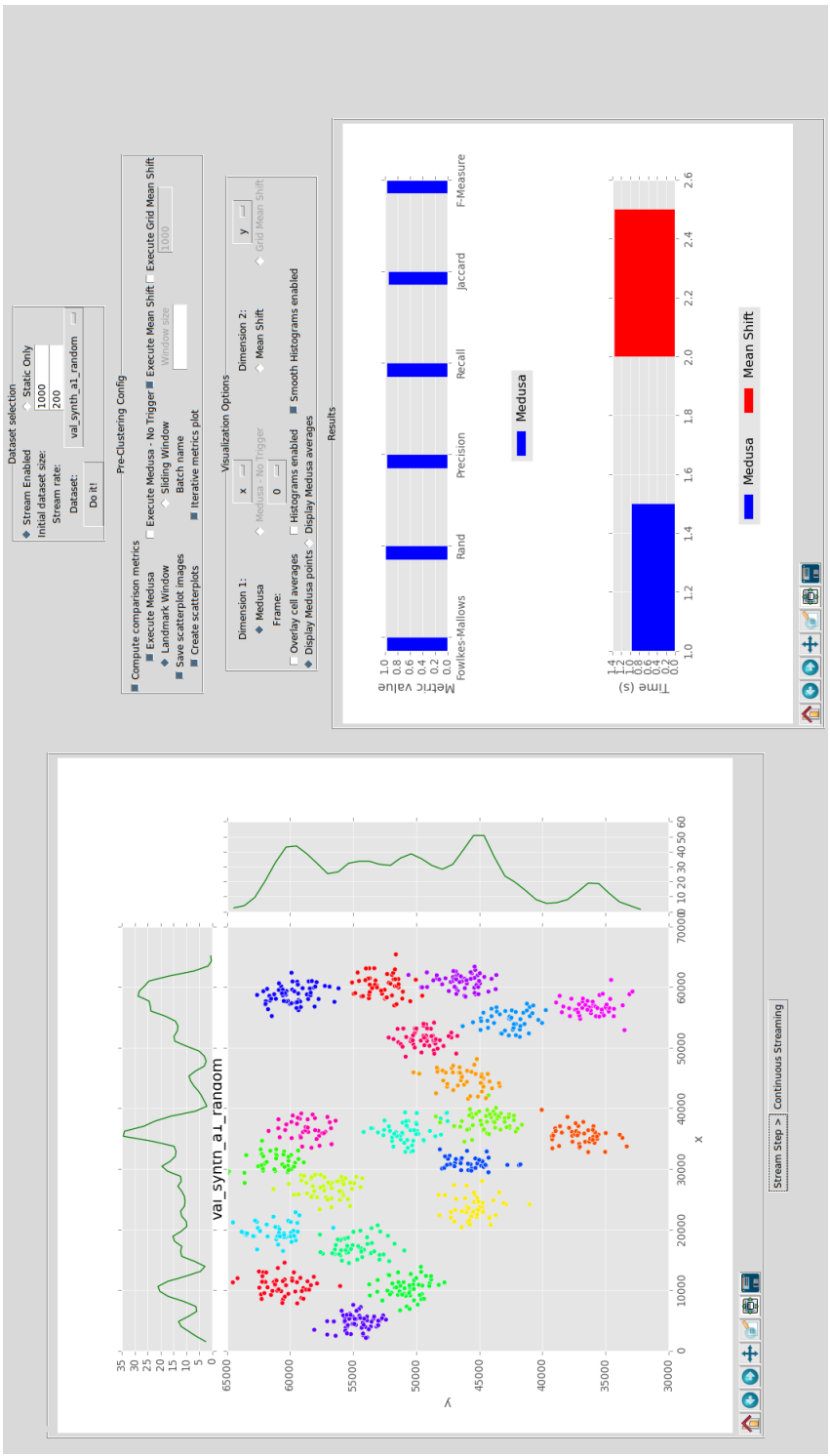


Figure C.1: Screenshot of our GUI for a static clustering

Appendix D

Personal Reflection

Having discussed all technical and academic details of my master thesis, in this section I will give a brief personal reflection on the project.

During any project, decisions are made that seem to be the right decision at the time, but should have been different when examined in hindsight. I think it is useful and important to evaluate these decisions, and examine how to improve on these in the future. During this thesis, I kept a journal detailing all my thoughts and decisions along the way. Looking back at these records, I noticed a number of things:

- During the initial months I was very focused on the execution time of the algorithm. Within Mobile Professionals, there is a huge drive to improve performance and within the development team there is strong competition to find the best speed up. Unfortunately, despite my project being separate from the work being done by the rest of the development team, my thinking was influenced by this, and I spent a lot of time optimizing code for which the academic basis was not sound yet.

- Another aspect of this thesis where I believe I made a mistake was my initial focus on the Mobile Professionals business case. Reading through my old journal entries, I found a large emphasis on their datasets and practical applications, which should have been of later concern.

- It only came to my attention relatively late in the project that the reference implementation I had been using, implemented in the Scikit-Learn library, was in fact incorrect and did not represent a proper Mean

Shift implementation. I should have examined their source code sooner to find this out, and I placed too much trust in the quality of the library implementation.

- During the project I quickly realised the need for some graphical representation of the clustering results. I decided to stick with what I was familiar with, which involved a basic web application. This architecture decision, however, made the visualization extremely complex, as it meant the Python application had to write data to the database, which then had to be read by the web application to visualize. It meant I spent a lot of time ensuring data validity, which was very hard to guarantee.

The field of big data, and big data visualization in particular, still fascinates me. I have learned a lot in terms of state-of-the-art academic research in this field, and I believe there is still a lot of work to be done. I hope this master thesis somehow contributes to this. I have also learned a lot from my supervisors, who made excellent points at numerous stages in the process.

While I regret the mistakes I have made during this project, I have definitely learned from them. As a whole, this master thesis project presented me with very different and new challenges compared to other projects during my time at TU Delft. Particularly the academic research aspect and the fact that I had no project group meant a lot of responsibility was placed with me.

Overall, I believe that this master thesis project has taught me a lot, and I hope that the resulting work has contributed (if in a very minor way) towards furthering big data research.

Bibliography

[1] Delft data science. `http://www.delftdatascience.tudelft.nl/`.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. 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.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. *Automatic subspace clustering of high dimensional data for data mining applications*, volume 27. ACM, 1998.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM, 2002.

[5] D. Barbará. Requirements for clustering data streams. *ACM SIGKDD Explorations Newsletter*, 3(2):23–27, 2002.

[6] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SDM*, volume 6, pages 326–337. SIAM, 2006.

[7] M. A. Carreira-Perpinan. Acceleration strategies for gaussian mean-shift image segmentation. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 1160–1167. IEEE, 2006.

[8] Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2007.

[9] D. Comaniciu. An algorithm for data-driven bandwidth selection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(2):281–288, 2003.

[10] D. Comaniciu, V. Ramesh, and P. Meer. The variable bandwidth mean shift and data-driven scale selection. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 438–445. IEEE, 2001.

[11] CoverType. Covertype dataset. https://archive.ics.uci.edu/ml/datasets/Covertype. Accessed: 2014-09-30.

[12] D. DeMenthon and R. Megret. *Spatio-temporal segmentation of video by hierarchical mean shift analysis*. Computer Vision Laboratory, Center for Automation Research, University of Maryland, 2002.

[13] M. Fashing and C. Tomasi. Mean shift is a bound optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):471–474, 2005.

[14] D. Freedman and P. Kisilev. Fast mean shift by compact density representation. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1818–1825. IEEE, 2009.

[15] B. Georgescu, I. Shimshoni, and P. Meer. Mean shift based clustering in high dimensions: A texture classification example. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 456–463. IEEE, 2003.

[16] H. Guo, P. Guo, and H. Lu. A fast mean shift procedure with new iteration strategy and re-sampling. In *Systems, Man and Cybernetics, 2006. SMC'06. IEEE International Conference on*, volume 3, pages 2385–2389. IEEE, 2006.

[17] A. Hinneburg and D. A. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. 1999.

[18] M. C. Jones, J. S. Marron, and S. J. Sheather. A brief survey of bandwidth selection for density estimation. *Journal of the American Statistical Association*, 91(433):401–407, 1996.

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

[20] H. S. Nagesh, S. Goil, and A. N. Choudhary. Adaptive grids for clustering massive data sets. In *SDM*, pages 1–17. SIAM, 2001.

[21] K. Pearson. Liii. 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.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[23] S. J. Sheather and M. C. Jones. A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 683–690, 1991.

[24] X.-T. Yuan, B.-G. Hu, and R. He. Agglomerative mean-shift clustering. *Knowledge and Data Engineering, IEEE Transactions on*, 24(2):209–219, 2012.