ELM Decision Branching

Improving streaming clustering with decision branching

J. Moree

Master Thesis
Embedded Systems

TUDelft

ELM Decision Branching

Improving streaming clustering with decision branching

by

J. Moree

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday June 30, 2021 at 9:00 AM.

Student number:	4387333	
Project duration:	June 11, 2019 – June 30, 2021	
Thesis committee:	Dr. ir. S. E. Verwer,	TU Delft, supervisor
	Prof. dr. ir. R. L. Lagendijk,	TU Delft
	Dr. J. S. de Pinho Gonçalves,	TU Delft

TU Delft

Abstract

Clustering is a commonly used method in data analysis. It is a complex problem that can be very time consuming, especially when clustering large datasets with many features. Most clustering algorithms scale exponentially in time when increasing the dataset size, making it infeasible to use them for large datasets. Streaming algorithms do not have this problem as they will scale linearly. Evolving Local Means (ELM) is a streaming clustering algorithm based on Mean Shift.

The ELM algorithm iterates over all data samples in a single iteration, updating an internal structure of clusters with each sample being added. Similar to Mean Shift, ELM requires only one parameter to be provided by the user: the initial radius of a cluster. In ELM, each sample can either be added to the nearest cluster or turned into a new cluster, depending on the distance to the nearest cluster. Clusters that come too close to each other will be merged to form a single cluster. There is one issue with this approach: the decisions are based on an intermediate state that is constantly evolving. This might cause incorrect clustering results when ELM merges clusters together prematurely.

This research introduces ELM Decision Branching (ELM-DB) which extends ELM with decision branching. ELM-DB aims to reduce the chance of a premature merge by postponing important decisions. Instead of making a decision directly, it will branch and continue all possible options until the decision can be made. ELM-DB is able to improve clustering performance with only a small increase in runtime and produce consistent results for a larger range of radius settings than ELM.

Preface

My graduation period has been a long journey. After several delays and setbacks in my research, my initial thesis topic on SS7 fraud eventually became infeasible. With some effort, I managed to find a new topic and started over in the direction of streaming clustering. On a personal level it has also been a difficult period. I have lost my beloved stepmother after a long fight against cancer. I miss her and know that she would be proud of me for how far I have come. Despite all, I have pushed through and completed my research from which the thesis now lies in front of you.

I would like to thank everyone who has been involved in my research or has supported me in any way. First, I would like to thank my supervisor Sicco Verwer for his support with valuable insights into machine learning and useful feedback during the final phase of my thesis project. I would also like to thank my previous supervisor Christian Doerr, who has supported me during the start of my thesis project, for all the opportunities, enthusiastic meetings and helping me find a new thesis project when my initial research on SS7 fraud became infeasible to complete. Furthermore, I would like to thank my thesis committee members Inald Lagendijk and Joana Gonçalves for being part of my committee and attending my presentation.

Although my research has gone in a different direction than anticipated, I would still like to thank everyone from KPN who has supported me during my thesis project. I would like to thank my intern supervisor Daan Planqué for his support and guidance at KPN. I would also like to thank Philippe Allemandou for all his support and invaluable insights into the SS7 data structure. Furthermore, I would like to thank Jaya Baloo for the opportunity to join the CISO team as intern and experience cyber security in practice.

Finally, I would like to thank my family and friends for there support during this period. I would like to thank all the people that were there for me when I needed them the most. I especially would like to thank my girlfriend Natasja van Heerden, who has always been there for me during this difficult period. I could not have done this without her.

J. Moree
Delft, June 2021

Contents

1 Introduction

Clustering is a well known method for data analysis that is used for grouping similar data points in a dataset. There are many applications where clustering is being used. For example pattern recognition or detection of fraudulent behaviour in network data. Datasets can be noisy and can have many different features that need to be taken into consideration during the clustering process. This makes clustering a complex problem that can be very time consuming, especially for large datasets.

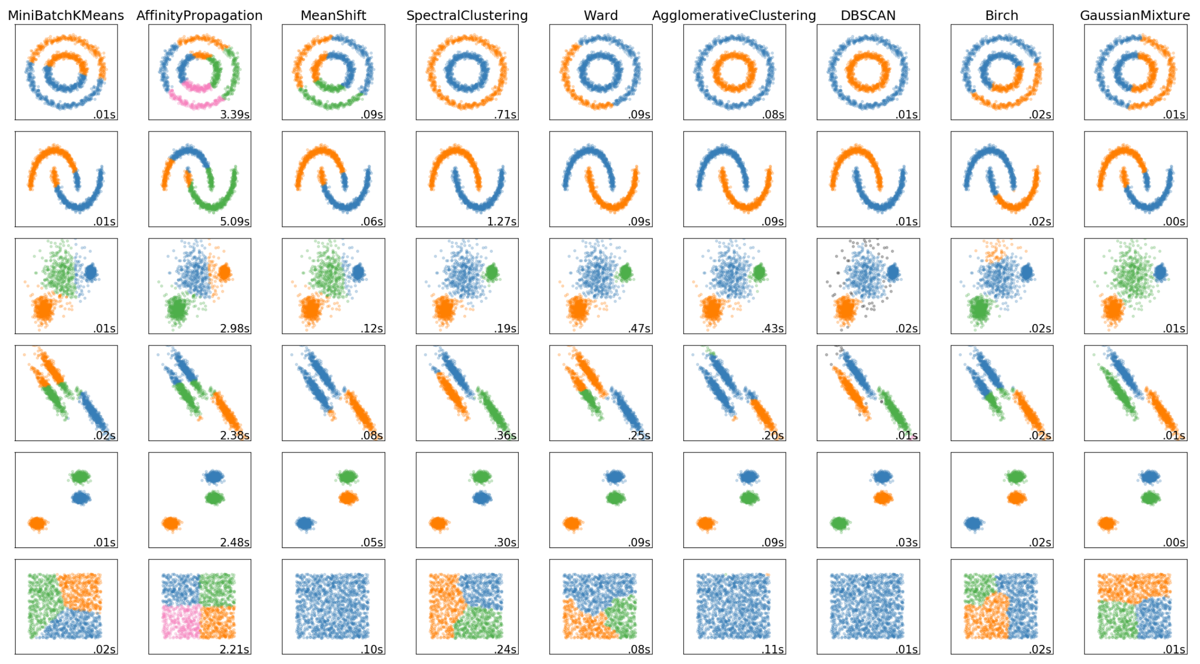


Figure 1.1: Comparison of clustering methods from Scikit Learn [17] using six different datasets.

There are many different clustering methods available, each with their own strengths and weaknesses. Figure 1.1 shows a comparison from Scikit Learn [17] of different clustering methods. They have applied these clustering methods on six generated datasets to visualize some of the differences between these methods.

The choice for a clustering method depends on the use case. Do you want to find many small clusters or only a few large ones? Should the clusters be equal in size? Should the clustering method be able to detect clusters with different shapes? All clustering methods also require some parameters to perform the clustering. K-means and spectral clustering for example, requires the number of clusters to be known.

A commonly used cluster method is Mean Shift. It requires only one parameter to be known: the bandwidth. This parameter gives the algorithm an indication to the expected size of a cluster. There are also methods that can give an estimation of the bandwidth for a data set. This makes it easy to use Mean Shift when analysing a new data set.

A limitation of Mean Shift is its scalability. Increasing the dataset size will increase the required runtime for the algorithm exponentially. This limits the usability of these clustering algorithms to small datasets, as large datasets would take way too long to process. However, the amount of data we are collecting increases every day, and so are the size of the datasets we would like to analyse with clustering.

A solution could be in the concept of streaming algorithms. In streaming algorithms, the input is defined as a stream of items where each item is usually processed at most a few times. This means that the items do not have to be kept in memory, thus limiting the required memory. They can also have a limitation in maximum processing time for each item, resulting in a linearly scaling runtime. Linear algorithms may be more complex and therefore slower on small datasets than comparable exponential algorithms. For large enough datasets, the linear algorithm will become the most efficient option as it maintains a constant time per sample. While an exponential algorithm will require more time for each additional sample. Not all algorithms can be easily converted into streaming algorithms. Usually, some sacrifices have to be made in order to create a streaming approach of an algorithm. An streaming algorithm can for example only give an approximate result or might have some constraints on the input data.

Baruah and Angelov [2] have proposed a streaming solution for Mean Shift, called Evolving Local Means (ELM). This algorithm enables clustering of large datasets with a streaming approach that is based on Mean Shift. ELM will maintain a structure of clusters and updates this structure each time a sample is added. Each cluster in ELM is defined by a centre and an average norm of the samples as a radius. These parameters are updated with each added sample. When a new sample is near a cluster, it will be added to the nearest cluster. Otherwise, a new cluster is created. Clusters that become too close too each other will be merged into a single cluster.

This method requires the samples to be in clearly separated subsets. ELM is evolving its structure of clusters with each additional sample. This means that the first samples in a cluster will have a large influence on the position and radius of the cluster.

The first samples of a cluster could be fairly apart, but just close enough to be added to the cluster. This would result in cluster with a large radius and possibly even get close enough to another cluster to merge together into one large cluster. Even when the next samples show a clear distinction between the clusters, they can only be added to the large cluster in this region. This premature merge, is due to the fact that a merge decision had to be made with insufficient data available.

ELM requires a clear spacing between clusters in order to function correctly and prevent these premature merges. However, datasets used in practice often contain noise and can have clusters that are close to each other. This limits the usefulness of ELM to datasets with low noise and clearly spaced clusters.

1.1. Research question

My focus will be to modify ELM so that it postpones these difficult decisions until there is more data available, with the use of decision branching. This means that with difficult decisions, all cases will be calculated in parallel until a good decision can be made. This method will result in a significant increase in runtime, as multiple branches need to be calculated. These branches might benefit from parallel calculation on a commodity graphics card to counteract the increase in runtime. In this thesis I am going to answer the following research question:

> ***Can streaming cluster analysis with Evolving Local Means be improved using decision branching?***

I have divided this question into the three sub-questions below. Each sub-question will be discussed in a separate chapter.

- ***Question 1: What is decision branching and can it be used to improve the cluster quality of Evolving Local Means?*** In chapter 3, I will discuss the concept of decision branching. What is decision branching and how does it work? Why can this be a useful tool in ELM and how should it be integrated into ELM?
- ***Question 2: What is GPU acceleration and can it be used to improve the performance of decision branching on Evolving Local Means?*** What is GPU calculation and can it be useful in this application? This will be discussed in chapter 4.
- ***Question 3: What is the practical performance of Evolving Local Means with decision branching compared to the original Evolving Local Means algorithm and Mean Shift?*** Chapter 4 will also discuss the performance of my method in practice. I will compare the performance of my method with both ELM and Mean Shift. This comparison will be performed with the use of different real-world data sets.

With these questions, I am going to research the effect of decision branching on the cluster quality and runtime performance of Evolving Local Means.

1.2. Outline thesis

This thesis will start in chapter 2 with a review of related work that has been performed in the field. In this chapter, I will also explain the clustering methods Mean Shift and Evolving Local Means into more detail as background information for the next chapters.

The next chapters will be focussed on answering the sub-questions as described above. This starts with improvement of the clustering accuracy of ELM with the use of decision branching in chapter 3. In this chapter I will discuss how I am going to integrate decision branching into ELM and what effect it will have on the clustering accuracy. In chapter 4, I will focus on the runtime performance of my algorithm. I will compare the accuracy and performance of my algorithm with ELM with the use of both synthetic and real life datasets. I will also discuss the performance impact of decision branching and discuss whether parallel calculation using a commodity graphics card can be used to improve performance. The thesis will be concluded with a conclusion and discussion in chapter 5.

2 Background and related work

In this chapter I will discuss scientific work that is related to this research. I will start with work that has been done on streaming clustering. Next, I will discuss some cluster methods that use commodity graphics cards to improve performance. Furthermore, I will discuss some clustering methods that apply both the streaming concept and the GPU acceleration. After this overview of the related work, I will explain two clustering methods into more detail as background for my research. The first clustering method is Mean Shift, as this forms a basis for the method I am going to extend upon. The second method is Evolving Local Means. This is a streaming clustering algorithm that is based on Mean Shift. In this thesis I will be extending upon the work on Evolving Local Means.

2.1. Related work

Different streaming clustering algorithms have been proposed to allow the clustering of huge datasets, containing millions or even billions of samples. WSTREAM [24] is a streaming extension on kernel density clustering. Online Divisive-Agglomerative Clustering (ODAC) [18] and the work from Tu et al. [25] show streaming methods for agglomerative clustering. Streaming approximations for K-means have been proposed in the work from Ailon et al. [1] and Braverman et al. [3]. Evolving Local Means (ELM) [2] is a streaming algorithm that has been based on Mean Shift. CODAS [13], extended by CEDAS [14], continues this work with clustering of arbitrary shaped clusters. These methods show different concepts for streaming clustering. In this thesis I will extend the work on Evolving Local Means. Section 2.3 will discuss this method into more detail as background for this thesis.

The intrinsic parallelism in most cluster algorithms allows the use of commodity graphics cards to accelerate the computational intensive clustering tasks. The work from Farivar et al. [9], Shalom et al. [22] and Hong-tao et al. [12] propose the use of GPUs to accelerate K-Means clustering. The use of GPUs to accelerate Hierarchical Agglomerative Clustering has been proposed in the work from Shalom and Dash [20, 21]. GPU acceleration has also been used to improve Mean Shift, as can be seen in the work from Men et al. [16] and Sirotkovic et al. [23]. These methods use GPUs to accelerate parallel aspects of cluster algorithms. I am going to use a commodity graphics card to process different branches in parallel.

Some methods have been proposed where streaming clustering is combined with the acceleration from commodity graphics cards. The work from Cao et al. [5] propose a GPU-based streaming cluster algorithm based on K-Means. GBIRCH [7] extends the streaming clustering algorithm BIRCH [27] using parallel calculation on GPU. The streaming method DenStream [4] has been extended by G-DenStream [11] using GPU acceleration. These methods show that the performance of some streaming clustering algorithms can be improved with GPU acceleration. In this thesis I am going to discuss whether GPU acceleration can be a meaningful addition to improve runtime performance in my algorithm.

2.2. Mean Shift

Mean Shift [10, 26] is a well known clustering algorithm that is often used in data analysis. It works by calculating the mean of all samples within a radius of a point. The point is then shifted to this mean value, and the process is repeated. This process continues until the point converges at a local maxima. Mean Shift is based on the concept that most samples in a cluster will be close to the centre. In other words, a cluster will have the highest density in the centre. Mean Shift will shift a points towards denser areas to find a local maxima, the centre of a cluster. Samples that shift to the same local maxima will be grouped as a cluster.

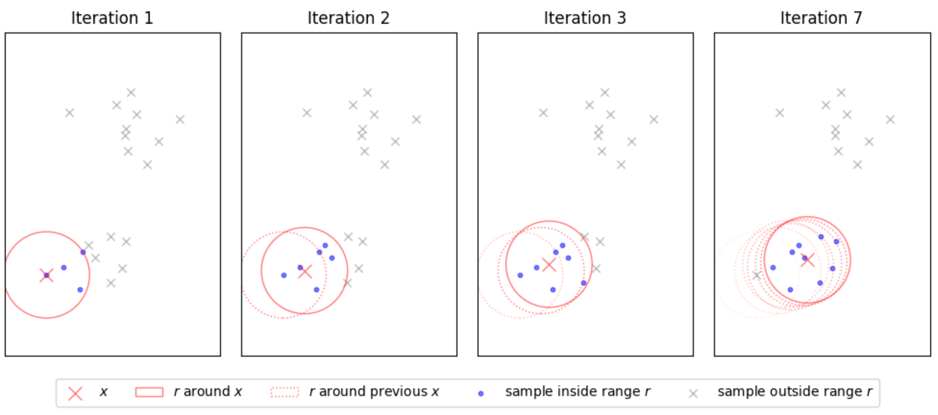


Figure 2.1: Example of Mean Shift, showing the iterative approach on dataset DS1 with the current mean x and radius r around it. Samples inside range r are used to calculate the new mean value $x \leftarrow m(x)$).

I have visualized this concept with an example in fig. 2.1. This example shows the clustering process on a synthetic dataset DS1. This dataset is generated with 20 samples following a Gaussian distribution around two centres. The example starts with a sample in the lower left corner, marked with an $\times$. This point is used as the starting point fro the mean value x of the Mean Shift algorithm. It will then look for samples within a radius r, in this case $r = 1.7$. Mean Shift will then calculate the mean of all samples within radius r. The point x is then shifted to this mean value ($x \leftarrow m(x)$) as can be seen in the second iteration. This process is repeated until the point converges in the 7th iteration. This process is performed for all samples in the data set.

$$m(x) = \frac{\sum\limits_{s \in S} K(s-x)s}{\sum\limits_{s \in S} K(s-x)} \tag{2.1}$$

The mean $m(x)$ of samples in the dataset S that are near a point x, is calculated according to eq. (2.1). Point x is then moved to the value of $m(x)$. This movement is done for all samples simultaneously and repeated until point x converges. The difference between $m(x)$ and x is called *Mean Shift* and the repeated movement is called the *Mean Shift algorithm* in Fukunaga and Hostetler [10].

$$K(x) = \begin{cases} 1, & \text{if } ||x|| \leq \lambda \\ 0, & \text{if } ||x|| > \lambda \end{cases} \tag{2.2}$$

Mean Shift uses a kernel, like the flat kernel in eq. (2.2), to calculate the average value from the samples nearby. The flat kernel is used to give an equal weight to all samples within a λ radius. All other samples are given a weight of 0, and are thus effectively ignored. It is possible to use other kernels in Mean Shift that use a weighted average like the Gaussian kernel. Although, in this thesis I will only refer to Mean Shift with a flat kernel as it has the closest resemblance to Evolving Local Means.

Only one parameter is required to run Mean Shift, the radius. This parameter is also called the bandwidth. The radius parameter is used to select the samples that are going to be used in the calculation of the mean of nearby samples. The influence of the radius parameter on the cluster result can be seen in fig. 2.2. It is clearly visible that a too low radius setting results in a large amount of clusters. This is because the radius is so small that there are often no other samples within the radius of a point.

This limits the shifting of a point, resulting in many separate clusters. A too large value for the radius is also not good. When the radius is large, most of the samples will be within the λ radius of a point, and thus will be included in the mean calculation. This would shift the point towards the mean of the whole dataset. As all points will shift to the same place, a single cluster is created, containing all the samples.

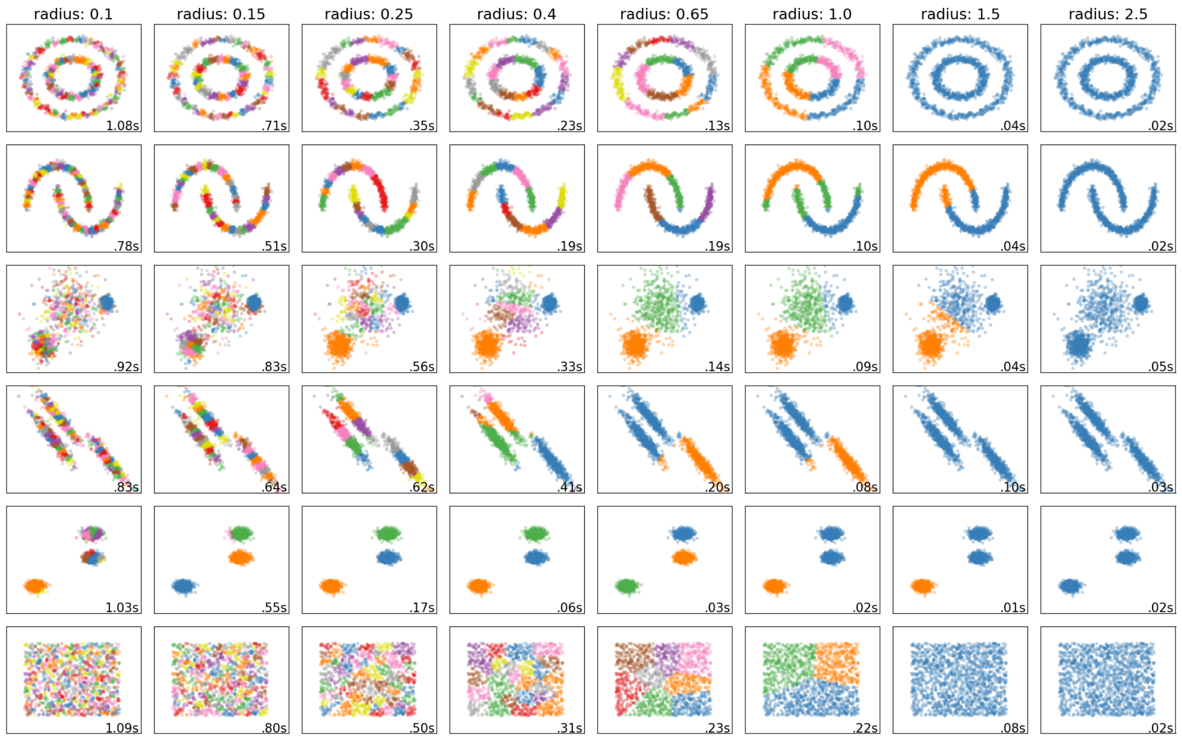


Figure 2.2: Mean Shift comparison of different radius settings using the six datasets from Scikit Learn [17], showing an increase in number of clusters found for a smaller radius.

As described before, Mean Shift uses an iterative approach to calculate the mean values and shift the points repeatedly. This results in a runtime complexity of $\mathcal{O}(idn^2)$ [6], for a dataset of size n with d dimensions and using i iterations. The memory complexity will be $\mathcal{O}(dn)$, as all samples are repeatedly used in the algorithm. The runtime complexity shows that Mean Shift scales exponentially on the size of the dataset. Each additional sample in the dataset will result in a larger penalty on the execution time, making the algorithm infeasible for use on large datasets. For large datasets it would be desirable to use an algorithm with a runtime that scales linearly with the size of the dataset. Besides the runtime complexity of generic clustering algorithms, it is also important to have a limit on the required memory. This can be achieved with the concept of streaming algorithms. Streaming algorithms processes all data samples one by one and does this in one, or at most a few passes. This makes streaming algorithms linearly scalable in terms of runtime complexity. Transformation of an algorithm to a streaming version can be a complex problem. Streaming algorithms are often only able to give an approximation or might only be usable for a limited subset of cases.

2.3. Evolving Local Means (ELM)

Evolving Local Means (ELM) is a streaming clustering algorithm that has been proposed by Baruah and Angelov [2]. It is based on the same concepts as Mean Shift and requires only one pass over the data set to determine the position and size of the clusters. A second pass can be used to label each sample based on the nearest cluster. Unlike Mean Shift, where each point is moved repeatedly, ELM is using a non-iterative approach. Each sample is directly added to the nearest cluster. If a sample is not near a cluster, it will be turned into a new cluster.

Like Mean Shift, the ELM algorithm requires only one user defined parameter, the radius r. This radius defines the minimum size of a cluster. The actual size of a cluster, determined by a distance parameter σ, is based on the location of the samples in the cluster. ELM requires the samples in a dataset to be divided into convex subsets that are a distance greater than r apart.

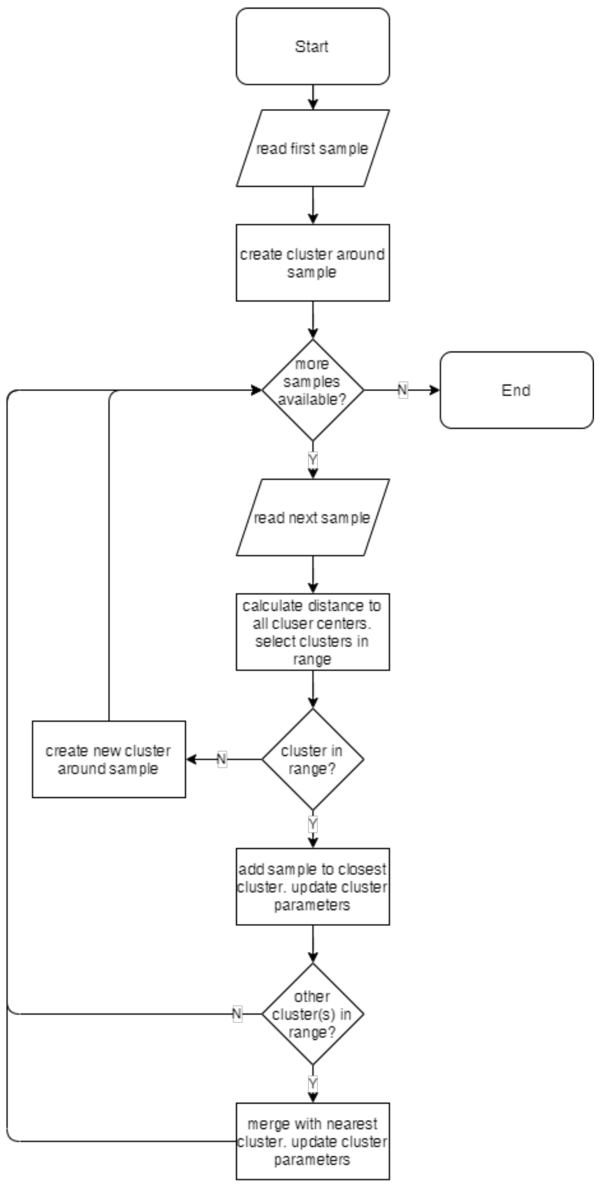

Start
read first sample
create cluster around sample
more samples available?
N
End
Y
read next sample
calculate distance to all cluser centers. select clusters in range
cluster in range?
N
create new cluster around sample
Y
add sample to closest cluster. update cluster parameters
other cluster(s) in range?
N
Y
merge with nearest cluster. update cluster parameters

Figure 2.3: A flowchart of the Evolving Local Means algorithm, iteratively processing all samples.

A flowchart of the ELM algorithm is provided in fig. 2.3. The first sample processed by ELM will be used to create the first cluster. The new cluster will then be defined with the location of the sample as its centre. For each following sample x_i, ELM will determine the distance to the closest cluster centre. If the sample is close enough to the nearest cluster p, according to eq. (2.3), it will add the sample to this cluster. If a sample is not near a cluster, the sample will be turned into a new cluster.

$$d_{ip} < (\max(\sigma_i, r) + r) \tag{2.3}$$

When a cluster p is updated by the addition of a new sample, ELM will search for the nearest neighbouring cluster j. If these clusters are getting too close together, they will be merged into a single cluster. This decision is determined with the condition shown in eq. (2.4).

$$d_{pj} < (\max(\sigma_p, r) + (\max(\sigma_j, r)) \tag{2.4}$$

The steps above are repeated for each sample until all samples have been processed. This process will let the clusters evolve as more samples are being added. ELM has a runtime complexity of $\mathcal{O}(cdn)$, with n as the number of samples in a dataset with d dimensions and c as the number of clusters. The memory complexity is $\mathcal{O}(cd)$, making it independent to the size of the dataset. The number of clusters c depends on the range of the input data and the radius r, as all clusters need to be separated by at least $2r$. Clusters cannot be closer together without being merged together. The runtime complexity shows that the runtime of ELM scales linearly with the size of the dataset.

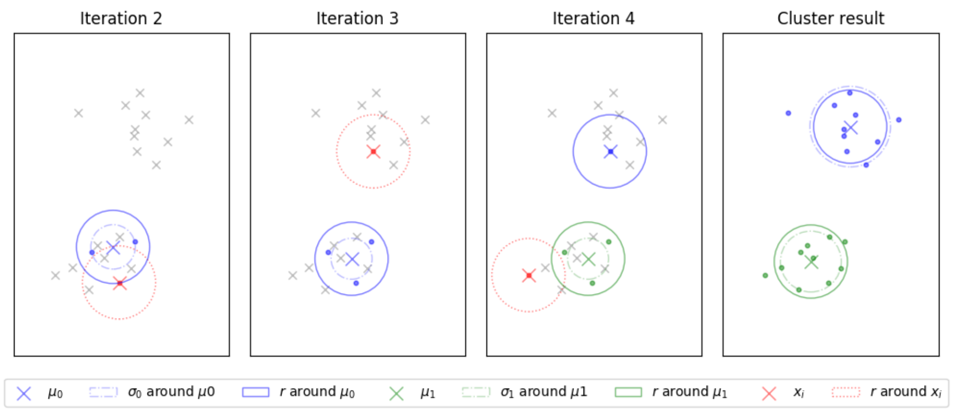


Figure 2.4: Example of ELM, showing different iterations on dataset DS1, with the radius r and distance parameter σ_i around each cluster centre μ_i. The next sample to be processed (x_i) is shown in red, together with the radius r around it.

Figure 2.4 shows the clustering process using dataset DS1 with a radius $r = 1.7$. Iteration 2 shows the first two samples have been added to the blue cluster. The next sample, shown in red, is also close enough to be added to the blue cluster. The parameters of this cluster, such as the centre, are updated as can be seen in iteration 3. The fourth sample is further away and will therefore be turned into a new cluster. After processing all samples, ELM returns a cluster result with two clearly separated clusters.

The ELM algorithm can start from scratch, with no prior knowledge of the dataset, as described above. It is also possible to start ELM with an existing structure of clusters that has been created by a different cluster method.

As ELM is a streaming algorithm, it will not store the individual samples in the clusters. Instead, it will store a sum of the samples in a cluster and a sum of squared samples in a cluster. These are labelled as α_i and β_i respectively, for a cluster i. These parameters, together with the number of samples in a cluster, allows the algorithm to determine the cluster centre μ_i and a distance parameter σ_i at each moment. These parameters are updated for each cluster as the cluster evolves. This approach limits the required memory for each cluster, making it independent of the number of samples in a cluster.

3 Improving ELM with decision branching

My research will be extending the Evolving Local Means (ELM) algorithm. For this research, I will implement the ELM algorithm as described in the appendix of the publication [2]. This implementation of ELM will be used as a basis for my algorithm, called ELM Decision Branching (ELM-DB). It will also be used as a baseline to compare the cluster quality and runtime performance. For this implementation I will be using Python as it is a high level language that is easy to write and has many useful libraries like `numpy` and `sklearn`. This allows me to easily implement and modify the algorithm for experiments. In this chapter will focus on improving cluster quality by extending the ELM algorithm with decision branching.

3.1. Decision branching

Decision branching is a method that is useful when an algorithm does not yet have enough information available to make a good decision. Instead of trying to make a potentially bad decision, it will delay the decision by branching to the different options and continue the calculation for each option in parallel until a better decision can be made. Decision branching can result in many parallel branches, especially when used repeatedly. This can have a large impact on the runtime performance of the algorithm, as each branch needs to be calculated. Unnecessary calculation of branches should therefore be avoided whenever possible, to reduce the amount of calculations.

How can decision branching be used to improve clustering in ELM? ELM will merge clusters that are close to each other. In some occasions, this decision is made with insufficient information, resulting in a premature merge. This merge decision is based on the location and size of the clusters. However, the location and size of a cluster will change each time a new sample is added to the cluster. This is especially relevant for new clusters that only contain a few samples, as the addition of a sample will have a relatively large influence on the location and size of the cluster. An unfavourable sample order might initially move two clusters towards each other or temporarily increase the distance parameter σ of a cluster. When the clusters become too close to each other ELM will merge the clusters together in to a single cluster, an action that cannot be undone in ELM. However, it might be better to keep the clusters separate if they eventually would move away from each other.

I will explain this premature merge problem with an example. For this example I have reordered the samples in dataset DS1, which is clustered correctly by ELM, to create a new dataset DS2. This dataset still meets the requirements set by ELM to have a minimum distance of radius r between all clusters. The samples in DS2 have been set to a specific order that will be causing the premature merge in ELM.

Figure 3.1 will show the clustering process of this reordered dataset DS2 in ELM using a radius $r = 1.7$. Iteration 1 shows the first sample that is used to create a first cluster, as shown in blue. Next to it, we see the next sample to be processed in red. This sample is not close enough to be added to the existing cluster, and thus will be turned into a new cluster in the second iteration. The third sample is just close enough to be added to the green cluster, as can be seen by the overlapping radius. This will result in a distance parameter σ larger than the radius r, as shown in iteration 3. The fourth sample will also be added to the green cluster, increasing σ further. We can see now in iteration 4 that the

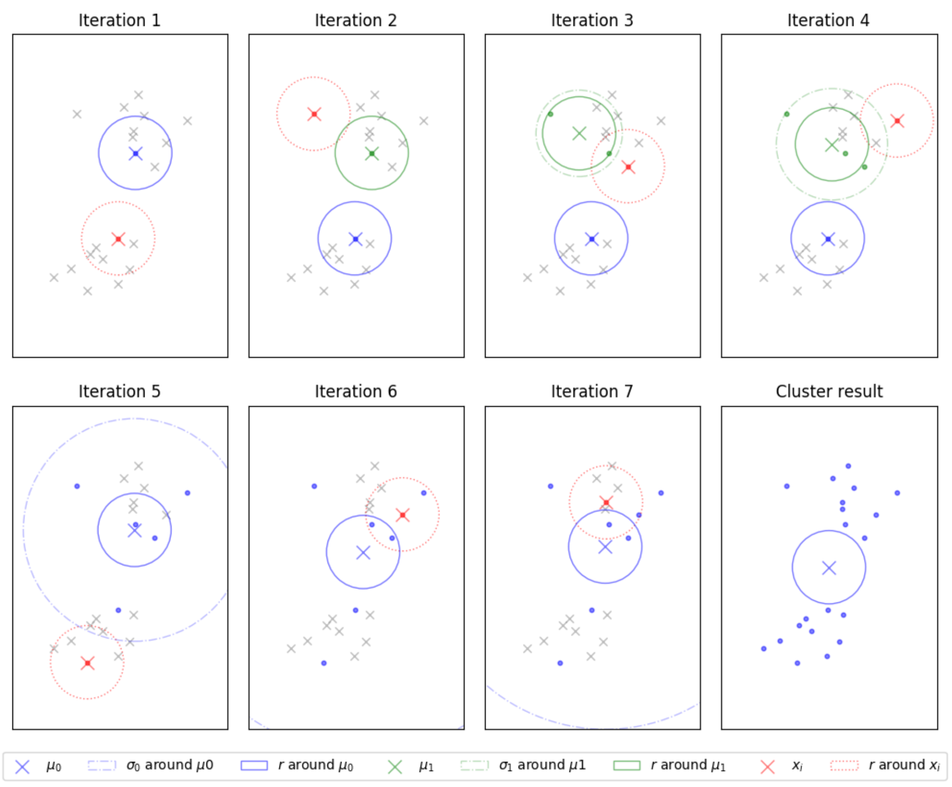


Figure 3.1: Example of premature merge in ELM, showing different iterations on the reordered dataset DS2, with the radius r and distance parameter σ_i around each cluster centre μ_i. The next sample to be processed (x_i) is shown in red, together with the radius r around it. Reordering of the samples causes ELM to return only one cluster.

blue and green clusters are almost overlapping. The fifth sample is also added to the green cluster, increasing σ even more. At this point, the two clusters are overlapping and thus merged into a single cluster. Iteration 5 shows this merged cluster, which has resulted in a large value of σ. The addition of the sixth sample, as shown in iteration 6, increases σ even further. All following samples will now added to this cluster, resulting in a result with only one cluster. So where does the algorithm go wrong? If we look at the clustering of dataset DS2 in fig. 3.1, we can see that the decision to merge the two clusters is made in iteration 5, before both clusters have stabilized enough for the algorithm to make a good decision. A similar situation could occur during the addition of a new sample, when a decision must be made to either add the sample to then nearest cluster or to create a new cluster. This means that two decisions would profit from decision branching, as shown below:

1. When a new sample x_i is processed, it will search for the nearest cluster p. If the distance d_{ip} between sample x_i and cluster p is small enough according to eq. (3.1), it will add the sample to this cluster. Otherwise, a new cluster is created for the sample. This can be visually seen by either the radius r or the distance parameter σ_p around cluster p overlapping with the radius r around the sample x_i.

$$d_{ip} < \left(\max(\sigma_p, r) + r\right) \tag{3.1}$$

2. If a sample has been added to an existing cluster p, the algorithm will search for the nearest neighbouring cluster $j \neq p$. If the distance d_{pj} is small, as determined by eq. (3.2), they will be merged into a single cluster.

$$d_{pj} < \left(\max(\sigma_p, r) + \max(\sigma_j, r)\right) \tag{3.2}$$

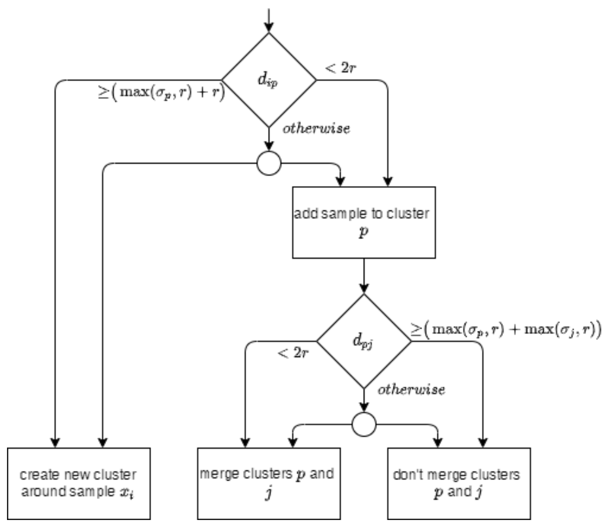

d_{ip}
$< 2r$
$\geq\left(\max(\sigma_p, r) + r\right)$
$otherwise$
add sample to cluster p
d_{pj}
$< 2r$
$\geq\left(\max(\sigma_p, r) + \max(\sigma_j, r)\right)$
$otherwise$
create new cluster around sample x_i
merge clusters p and j
don't merge clusters p and j

Figure 3.2: Flowchart showing the part of the ELM-DB algorithm where decision branching is being used. We can see here that there are three options for both decisions. For the first decision this is to either create a new cluster, add sample to existing cluster or try both using decision branching. Similarly for the second decision: merge clusters, don't merge clusters, or try both using decision branching.

Branching is only needed when it is not possible to make a good decision with the current information available. When looking at the distance d_{ip} between a new sample x_i and the nearest cluster p, there is no need to use decision branching when the sample is very close to the cluster. The sample should in this case always be added to the nearby cluster. So the algorithm will not branch when $d_{ip} < 2r$ and just add the sample to cluster p. There is also no need to branch when the sample is far away from the nearest cluster. Thus, the algorithm will always create a new cluster when $d_{ip} \geq \left(\max(\sigma_p, r) + r\right)$. Otherwise, the algorithm should branch to explore both options: creating new cluster from x_i and adding x_i to cluster p.

Similar steps will be used for the second decision point. If sample x_i has been added to cluster p, the algorithm will check if this cluster p is too close to the nearest neighbouring cluster and should be merged with this cluster. If the distance d_{pj} between cluster p and the nearest cluster j is small e.g. $d_{pj} < 2r$, then clusters p and j will always be merged. When d_{pj} is large e.g. $d_{pj} \geq \left(\max(\sigma_p, r) + \max(\sigma_j, r)\right)$, the clusters will not be merged. Otherwise, a good decision cannot be made and decision branching will be used. Both options will then be calculated: merging clusters p and j, and not merging clusters p and j. The two decisions described above are also illustrated in fig. 3.2.

The different branches will be implemented as a list of states that can be processed in parallel. The algorithm starts with a single state, that will branch into multiple states, each time decision branching is required. The algorithm will iterate over all samples one by one. Each iteration, it will update all states in the list with the new sample.

This approach can result in many states to be processed in parallel, which will significantly increase the runtime of the algorithm. Table 3.1 shows the number of states created after each iteration when processing the reordered dataset DS2. We can see here that the number of states increases to 788 states for only 20 processed samples. Fortunately, this list of states contains a lot of duplicates that can easily be removed. By removing these duplicate states, the number of states is reduced to only 15 unique states.

Table 3.1: Number of parallel states and number of unique states after removing duplicate states for each iteration.

Iteration	States	Unique states
1	1	1
2	1	1
3	1	1
4	1	1
5	2	2
6	3	3
7	3	3
8	3	3
9	3	3
10	4	4
11	5	5
12	7	6
13	10	6
14	12	5
15	22	6
16	51	9
17	72	8
18	150	10
19	357	13
20	788	15

3.2. Scoring metric

The algorithm is now able to create multiple states, each containing a possible clustering result. A scoring metric is now required to know which state has the best clustering performance. This metric will be used to select the state that will be returned as result. It is also used during the clustering process to detect which states have a low clustering performance.

There are several methods for evaluation of the clustering performance. All clustering evaluation methods can be divided into two groups: internal and external validation. Internal validation methods will evaluate the cluster performance without any external information, while external validation requires some external information. This external information can for example be learning samples containing a ground truth. As the scoring metric is used during the clustering process, it cannot use any external information and must thus be based on an internal validation method. It is also important to consider which aspect is being optimized by the clustering algorithm. The Mean Shift and ELM algorithms are optimizing the density of the found clusters.

3.2.1. Silhouette coefficient

The silhouette coefficient [19] is a commonly used internal validation method for clustering performance. It is based on how close together samples are within a cluster and how far they are from samples in the nearest neighbouring cluster. This means that it will produce a higher score for denser clusters that are further apart from each other. The silhouette coefficient gives a score to each sample in a cluster in the range [-1, 1], where 1 stands for perfect fit for the sample in the current cluster. A score of -1 shows a perfect fit with a different cluster which is an indication of bad clustering. The average score of all samples is used to give an overall score of the clustering accuracy, called the silhouette score.

The silhouette coefficient $s(i)$ of a sample x_i is determined by two factors. The average dissimilarity $a(i)$ of sample x_i to all other samples in te same cluster. And the average dissimilarity $b(i)$ to all samples in the nearest cluster that x_i is not a member of.

The average dissimilarity $a(i)$ is defined as the mean distance between sample x_i and all other samples x_j in the same cluster C_i. The definition for $a(i)$ is shown below, with $d(x_i, x_j)$ as the distance between x_i and x_j.

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(x_i, x_j)$$

For $b(i)$, the mean distance between x_i and all samples x_j in another cluster C_k are calculated for each other cluster C_k. $b(i)$ is determined by the cluster with the lowest mean distance to x_i and can be seen in the definition below.

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(x_i, x_j)$$

The silhouette coefficient $s(i)$ is then calculated from the factors $a(i)$ and $b(i)$ with the definition below. Note that the score is set to 0 for any cluster C_i with a size of 1.

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ if } |C_i| > 1 \quad \text{and} \quad s(i) = 0, \text{ if } |C_i| = 1$$

Different distance metrics, for example Euclidean distance or Manhattan distance, can be used for the distance $d(x_i, x_j)$. For this application I will be using the Euclidean distance as this is the distance metric being used in ELM. An overall silhouette score for the clustering can be achieved by calculating the mean of $s(i)$ for all samples x_i, as described by Kaufman and Rousseeuw [15].

3.2.2. Streaming approximation

The silhouette score gives a good indication of the cluster quality. However, it requires all individual samples to calculate the score. These individual samples are not available at arbitrary times, as this would contradict the limitations set by the streaming approach. The individual samples can only be accessed at limited times, in this case when the sample is being added to a state. This means that it is not possible to use the distance d_{ij} between a sample x_i and each other sample x_j. In this case, the exact value of the approximate score compared to the silhouette coefficient is not important. However, a positive difference between two silhouette coefficients should reflect into a positive difference on the approximate score as good as possible.

The silhouette score of each sample is based on two factors: the average dissimilarity to other points in the same cluster ($a(i)$) and the average dissimilarity to all points in the nearest neighbouring cluster ($b(i)$). For the average dissimilarity $a(i)$, I will use the distance parameter σ_{C_i} of the corresponding cluster C_i, that sample x_i is a member of. This will give an indication on the dissimilarity of samples in cluster C_i.

$$a(i) = \sigma_{C_i}$$

The average dissimilarity $b(i)$ will be determined by the distance between the centre μ_i of cluster C_i and the centre μ_k of its nearest neighbouring cluster C_k.

$$b(i) = \min_{C_k \neq C_i} \{d(\mu_i, \mu_k)\}$$

These values only need to be calculated once for each cluster, reducing the number of calculations. In the case where there is only one cluster, the silhouette score is set to -1. This will encourage the algorithm to find results with multiple clusters.

Table 3.2: Parameters used to generate the datasets for testing the Python implementations.

Standard deviation	1.0
Number of features	2
Number of clusters	3
Number of samples	250

To verify the accuracy of this approximation, I have compared the outcome of my approximation with the actual silhouette score of over 10000 states. For this experiment, I have generated different datasets with the `make_blobs()` method from SciKit Learn [17], using the parameters in table 3.2. The datasets have been normalized and clustered with ELM-DB, using a radius $r = 2.0$ and unlimited number of states. All resulting states from the clustering process are used to calculate both the silhouette score and silhouette score approximation. These steps have been repeated for different datasets until 10000 states have been processed. To allow calculation of the silhouette score of a state, I have

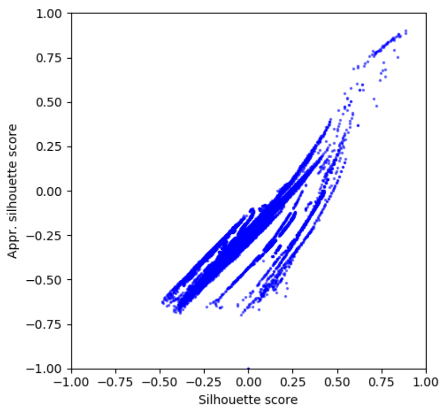


Figure 3.3: Correlation between the silhouette score approximation and the actual silhouette score for 10000 different cluster results shown as points in this plot.

temporarily modified the ELM-DB implementation in this experiment to store the individual samples in each cluster.

I have analysed the correlation between my approximation and the actual silhouette scores. This correlation can be seen in fig. 3.3. We can see here that there is a clear correlation between my approximation and the actual silhouette score. The approximation is most accurate for cluster results with a high score. The difference between the approximation and the actual score becomes larger for low scoring cluster results. As the highest scoring states are most important to be ordered correctly, this approximation should be sufficient for ordering of the states in ELM-DB.

3.3. Multiple states

The algorithm produces many different states, as seen in table 3.1. The number of states need to be limited in order to maintain performance. With the scoring metric we are now able to determine which state contains the best result.

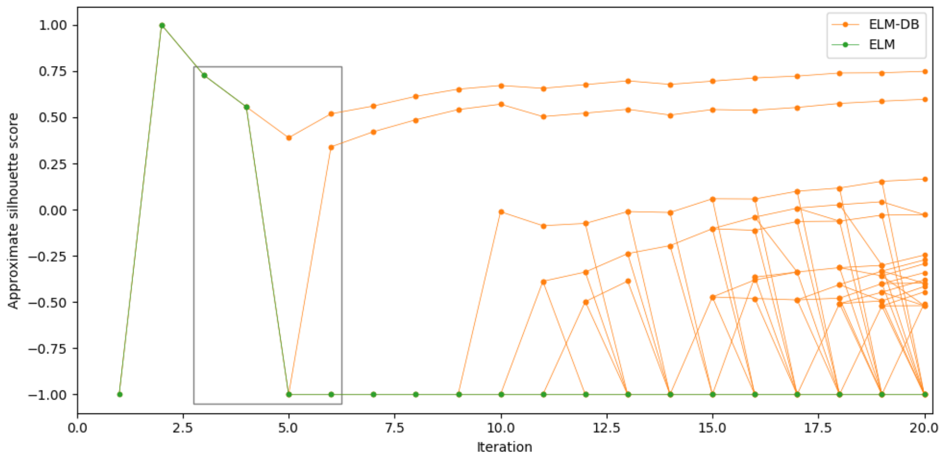
Figure 3.4a shows the approximate silhouette scores of all states after each iteration of the reordered dataset DS2. This dataset has been clustered with ELM-DB and ELM using a radius $r = 1.7$ and an unlimited number of states for ELM-DB. Each orange dot shows the intermediate states in ELM-DB for each iteration and the approximate silhouette score given to this state by ELM-DB. The branches from each iteration in ELM-DB is visualized with the orange line. The green line shows the decision path ELM would take for this dataset. We can see here that the algorithm branches for the first time in the fifth iteration. The first state has an approximate score of 0.39. The second one has an approximate score of -1.00, which indicates that it contains only one cluster. When we look at the whole graph, we see that this first decision is important to get to a good clustering result of 0.75. A second chance is given in the sixth iteration to get a clustering result of 0.60. If the algorithm had created a single cluster at iteration 6 (score of -1.00), it would not be able to recover from this decision and could at most get a score of 0.17 at the end.

I will explain the decision branching in more detail using the data in table 3.3. These iterations are also visualized in fig. 3.4b. We can see that iteration 3 resulted in two clusters: C_0 with one sample and C_1 with two samples. Iteration 4 will process the next sample $x_3 = (1.64, 1.37)$ (shown in red in iteration 3 of fig. 3.4b). ELM-DB will now check the distance to each cluster centre to find the nearest cluster. The distance to cluster C_0 is $d(x_3, \mu_0) = 3.8$, and the distance to C_1 is $d(x_3, \mu_1) = 2.7$.

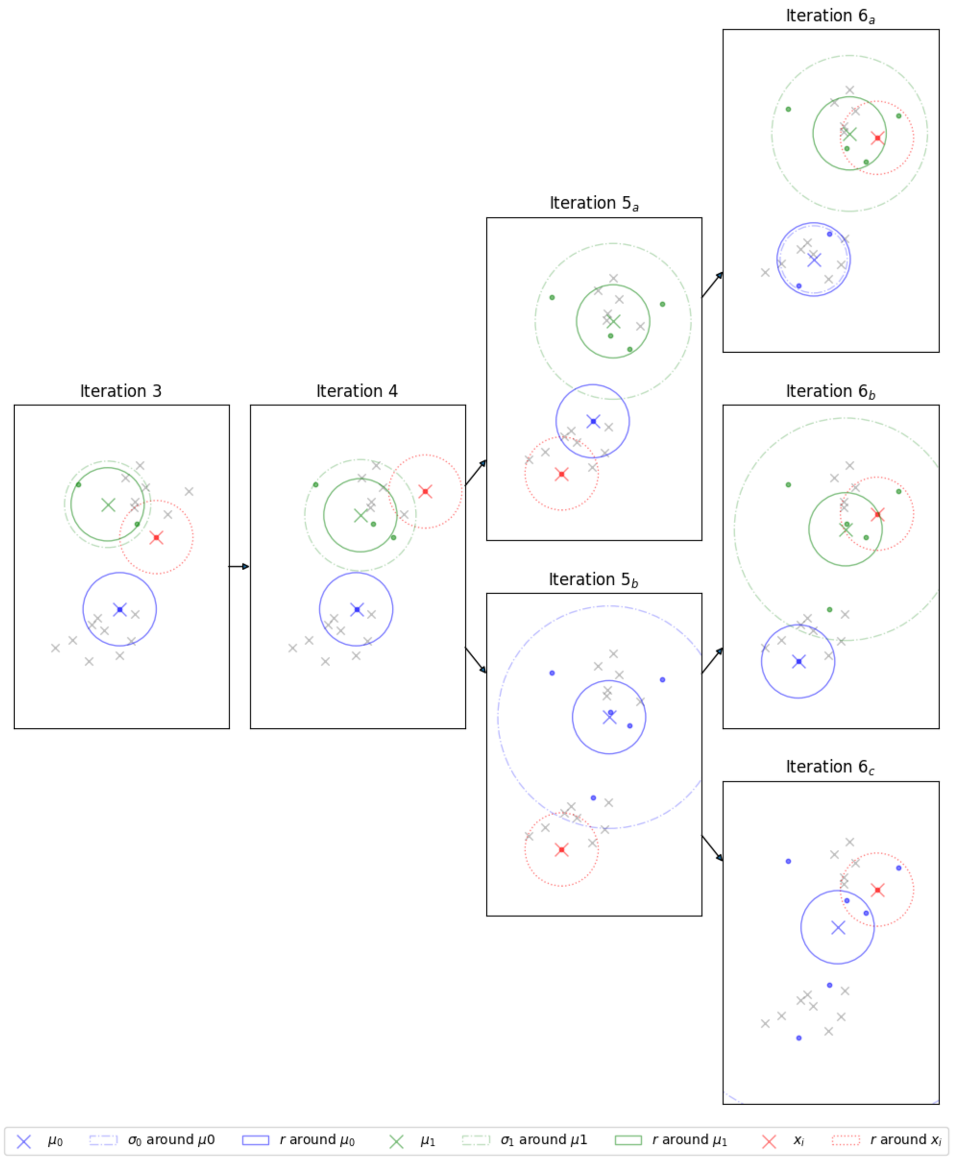
$$d(x_3, \mu_1) < 2r \quad \Longrightarrow \quad 2.7 < 3.4$$

This makes C_1 the nearest cluster with a distance smaller than $2r$. Sample x_3 will thus be added to cluster C_1. As the sample has been added to a cluster, the algorithm will now determine the distance to the nearest cluster to check if it should be merged. There is only one other cluster, C_0 with a distance $d(\mu_1, \mu_0) = 4.4$. The equation below shows that C_0 is far enough away and does not have to be merged.

$$d(\mu_1, \mu_0) \geq \left(\max(\sigma_1, r) + \max(\sigma_0, r)\right) \quad \Longrightarrow \quad 4.4 \geq 4.3$$



(a) Branching in ELM-DB visualized with the approximate score of all states in each iteration, shown as orange points. Connecting lines show how states evolve to the next iteration. Multiple lines from a state indicates branching, while multiple lines towards a point shows that there are duplicate states. The green line shows the decision path that would have been taken by ELM.



(b) Similar to fig. 3.5, showing iterations 3-6 with the decision branching in state 4 to $5_a/5_b$ and in state 5_b to $6_b/6_c$.

Figure 3.4: Decision branching in ELM-DB visualized using dataset DS2 with unlimited states. Iterations 3-6 has been shown into more detail.

Table 3.3: Internal variables for all states in iteration 3-6 for ELM-DB, when clustering dataset DS2.

Iter.	Sample	Cluster C_0			Cluster C_1			Comments
	x_i	μ_0	σ_0	$\lvert C_0\rvert$	μ_1	σ_1	$\lvert C_1\rvert$	
2→3		-0.06, -1.98	0.0	1	-0.62, 2.89	2.0	2	
3→4	1.64, 1.37	-0.06, -1.98	0.0	1	0.14, 2.38	2.6	3	Add x_3 to C_1
4→5_a	3.15, 3.48	-0.06, -1.98	0.0	1	0.89, 2.66	3.6	4	Add x_4 to C_1
4→5_b		0.70, 1.73	5.2	5	-	-	-	Add x_4 to C_1, merge C_1, C_0
5_a→6_a	-1.50, -4.41	-0.78, -3.19	1.6	2	0.89, 2.66	3.6	4	Add x_5 to C_0
5_b→6_b		-1.50, -4.41	0.0	1	0.70, 1.73	5.2	5	Create new cluster C_0
5_b→6_c		0.33, 0.71	9.3	6	-	-	-	Add x_5 to C_0

In the next iteration, sample $x_4 = (3.15, 3.48)$ will be processed. Again, we start with finding the nearest cluster. C_1 is the nearest cluster with a distance $d(x_4, \mu_1) = 3.2$. Sample x_4 will also be added to cluster C_1 as $3.2 < 2r = 3.4$. The algorithm will then check the distance to the nearest cluster being $d(\mu_1, \mu_0) = 4.7$. At this moment, the algorithm will make the decision to branch into two different states, following the equation below.

$$2r \leq d(\mu_1, \mu_0) < \big(\max(\sigma_1, r) + \max(\sigma_0, r)\big) \quad \Longrightarrow \quad 3.4 \leq 4.7 < 5.3$$

The first state (5_a) will keep the clusters separate, while the second state (5_b) will merge clusters C_1 and C_0 into a single cluster. Iteration 6 starts with two different states. We will start with the first state, derived from 5_a. Sample $x_5 = (-1.50, -4.41)$ is closest to cluster C_0, with a distance $d(x_5, \mu_0) = 2.8$. As the sample is close to the cluster it will be added to cluster C_0, resulting in state 6_a. The second cluster is not close enough, so the clusters will not be merged. The second state in this iteration is derived from 5_b. Again, the distance $d(x_5, \mu_0) = 6.5$ is calculated to the single cluster C_0.

$$2r \leq d(x_5, \mu_0) < \big(\max(\sigma_0, r) + r\big) \quad \Longrightarrow \quad 3.4 \leq 6.5 < 6.8$$

This means that the algorithm will branch on this decision. The first state (6_b) will turn sample x_5 into a new cluster. While the second state (6_c) adds x_5 to cluster C_0. As there is only one cluster in this state, there will be no cluster nearby to merge with. As the algorithm continues with the remaining samples, the states will evolve. After processing the states in each iteration, the algorithm will remove any duplicate states. This results in 15 unique states after processing all 20 samples from dataset DS2.

The number of parallel states need to be limited to maintain performance. Figure 3.4a shows that the states with the best intermediate scores will have the highest potential to a good clustering result. I will thus limit the states by keeping the best s states after each iteration. The number of states used in the algorithm has a large effect on the runtime of the algorithm. In section 4.2, I will further analyse the effect of this parameter on the cluster result and runtime of the algorithm.

With the introduction of multiple states, ELM-DB needs to update all states in each iteration. This already increases the runtime complexity to process all samples to $\mathcal{O}(scdn)$. Similar to ELM, this is based on n samples in a dataset with d dimensions and having c clusters. However, for ELM-DB we also need to consider the number of states s. In order to manage these states, we also need time to update the score of each state ($\mathcal{O}(sc^2d)$) and sort the states to find the best states ($\mathcal{O}(s\log(s))$). This gives a total runtime complexity of $\mathcal{O}(scdn + sc^2d + s\log(s))$, maintaining linear scalability on the number of samples n. The memory complexity of ELM-DB is $\mathcal{O}(scd)$, as multiple states s need to be maintained. Note that the memory requirements are still independent on the dataset size n.

The clustering algorithm processes the data samples as a stream. This means that the algorithm is not able to keep track of which samples have contributed to each cluster. Instead, it returns a structure of clusters, containing the position, number of samples and the average norm σ of each cluster.

In order to label a dataset, I have extended the algorithm to iterate a second time over all samples x_i. In this iteration each sample is labelled based on the nearest cluster. As some clusters have a larger average norm σ than others, I have subtracted the average norm σ_k from the distance to each cluster centre μ_k.

$$\min\{d(x_i, \mu_k) - \sigma_k\}$$

The clusters created by ELM and ELM-DB evolve as more samples are being processed. This means that for a sample the nearest cluster during clustering might not be the nearest cluster any more when labelling this sample in the second iteration over the dataset. that samples processed by ELM-DB will thus not always be labelled with the same cluster it has contributed to during the clustering process.

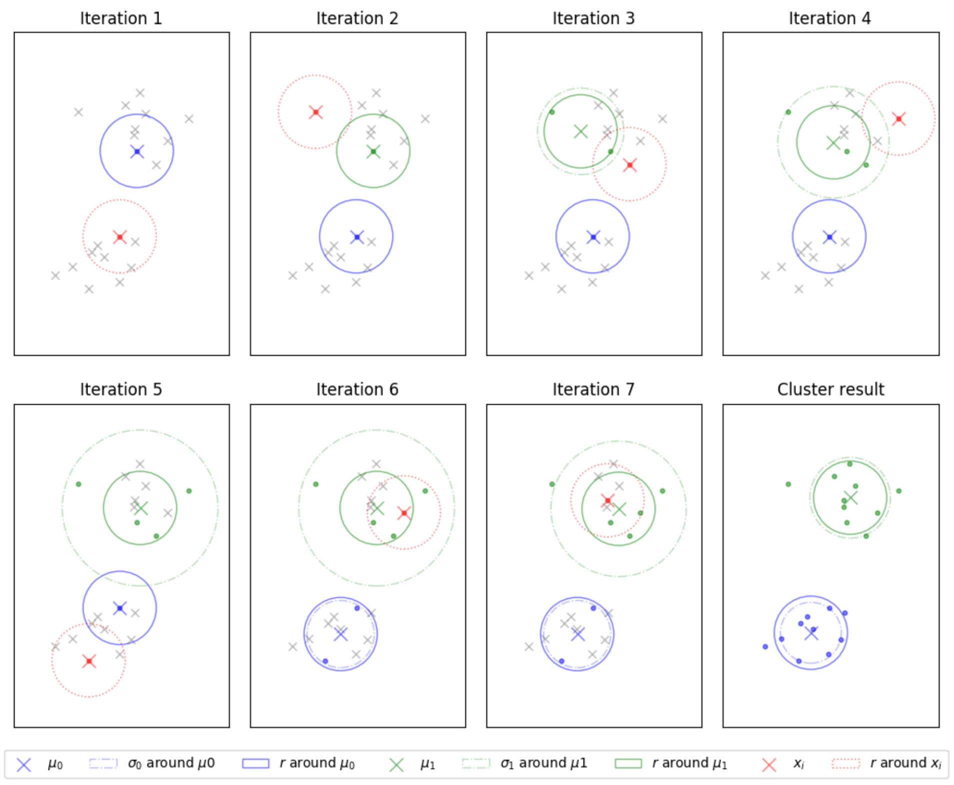


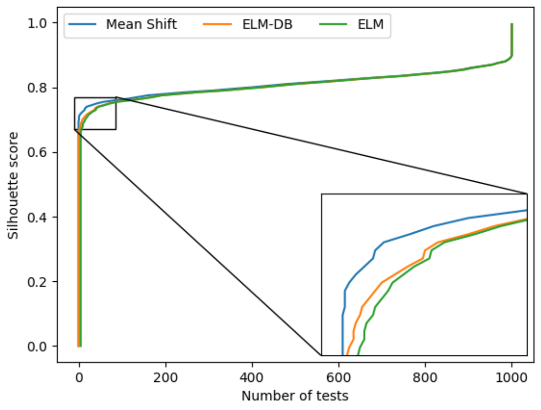
Figure 3.5: Similar to fig. 3.1, showing the clustering process of the reordered dataset DS2 in ELM-DB. WE can see here that ELM-DB is able to prevent the premature merge and returns clearly separated clusters. For each iteration, the state with the highest approximate score is shown.

3.4. Results

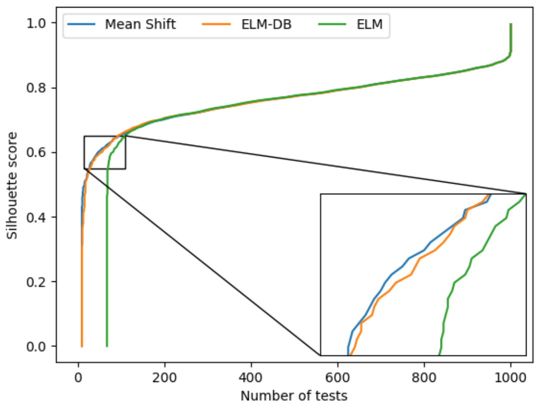
With the use of decision branching my algorithm, ELM Decision Branching (ELM-DB), should be able to prevent a premature merge as described with the reordered dataset DS2. Figure 3.5 shows the state with the highest score at each iteration. We can see here that ELM-DB is able to prevent the premature merge and create a cluster result with two clusters.

I have compared the cluster performance of ELM-DB with Mean Shift and ELM, using 1000 synthetic datasets with a Gaussian distribution. All synthetic datasets are for this section are generated with the `make_blobs()` method from SciKit Learn [17], using the parameters in table 3.2.

The datasets have been normalized before clustering and have been selected to have a minimum distance r between clusters as required by ELM. For ELM and ELM-DB I have used a radius $r = 2.0$ and for Mean Shift a radius $r = 2.6$, as these values produce the highest average silhouette score. ELM-DB is set to use a maximum of 10 states.



(a) Using datasets generated with a minimum distance of r between clusters, as required by ELM.



(b) Using datasets generated without the minimum distance requirement.

Figure 3.6: Comparison of cumulative silhouette scores of 1000 datasets clustered by Mean Shift, ELM and ELM-DB.

A cumulative distribution of these cluster results can be seen in fig. 3.6a. We can see here that Mean Shift produces the best results, closely followed by ELM and ELM-DB. We can also see that ELM-DB has a slightly better score than ELM, although the differences between the algorithms are very small. Both Mean Shift and ELM-DB are able to produce good cluster results for all datasets. However, ELM was not able to cluster all datasets correctly. 5 datasets (0.5%) were given a score of 0.0, as the results from ELM contained only one cluster.

The requirement in ELM to have a minimum distance between clusters, makes it easy to cluster data as the clusters are clearly separated. Without this requirement ELM would be more likely to suffer from premature merge. I have therefore repeated the last comparison without a minimum distance requirement. The cumulative distribution of these results can be seen in fig. 3.6b. These results show a larger difference between ELM and Mean Shift as ELM is not able to cluster all of these datasets correctly. ELM-DB does not have this problem and produces results close to Mean Shift. Both Mean Shift and ELM-DB were unable to cluster 10 datasets (1.0%) correctly, compared to ELM unable to cluster 68 datasets (6.8%). In these cases, the algorithms returned a single cluster, resulting in a silhouette score of 0.0.

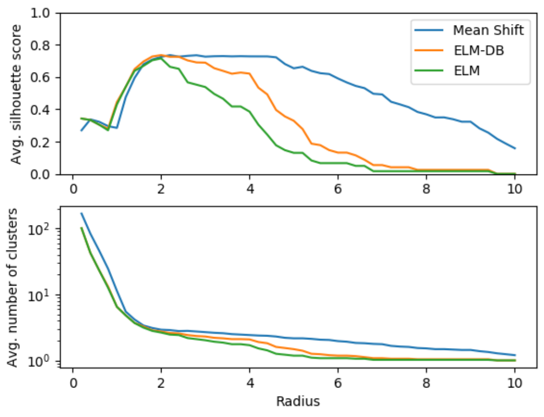


Figure 3.7: Average silhouette score of 50 datasets using Mean Shift, ELM and ELM-DB for different radius settings. We can see here that Mean Shift is able to produce a high silhouette score for a large range of radius settings, followed by ELM-DB. While the radius needs to be chosen carefully to receive the highest silhouette score.

Mean Shift, ELM and ELM-DB all require the same parameter, the radius r. To see the effect of this parameter, I have clustered 50 datasets with different radius values. Figure 3.7 shows the average silhouette score for each radius value. The top plot shows that Mean Shift is able to give consistent clustering results for every radius in the range of 2 to 4.5. ELM, on the other hand, requires a precise radius of 2 to get the best clustering result. The silhouette score quickly drops as the radius is increased or decreased. My algorithm is able to keep a better silhouette score than ELM when the radius is increased above 2. The score is slowly deceasing until a radius of 4, after which the silhouette score quickly drops. The slight increase for the silhouette scores with a radius below 1, can be explained by the large number of clusters. When the radius is too low, the algorithms are not able to group samples together. This results in a large number of clusters, as can be seen in the bottom plot.

In this chapter, I have explained the concept of decision branching and how I have implemented it to create ELM Decision Branching to improve cluster quality. With these results we are now able to answer the first sub-question:

Question 1: What is decision branching and can it be used to improve the cluster quality of Evolving Local Means?

Decision branching is a method used to postpone decisions by processing all possible cases in parallel until the decision can be made. With the use of decision branching, ELM-DB is able to reduce the chance of premature merge as can be seen in fig. 3.6, resulting in a higher average silhouette score. Figure 3.7 shows that it is also able to produce a high silhouette score for a larger range of radius settings, making it more flexible to use in practice.

4 Performance of the algorithm

In the previous chapter we discussed the cluster quality of ELM Decision Branching (ELM-DB). This chapter will focus on the runtime performance. I have therefore created more efficient implementations of ELM and ELM-DB in C. These implementations will be faster than Python and will thus be useful to cluster large datasets. I have verified that the C implementations produces the same results as the Python implementations by comparing results from different datasets. The runtime performance will be measured on a Windows 10 system with an Intel Core i7-8750H CPU at 2.2 GHz and 16 GB RAM, running Windows Subsystem for Linux (WSL).

4.1. Analysing performance using C implementations

With the C implementations I will now analyse the runtime performance of ELM-DB. I have compared the runtime of Mean Shift, ELM and ELM-DB for different dataset sizes. The number of samples for Mean Shift have been limited to 1000, as larger datasets would take too long for a non streaming algorithm.

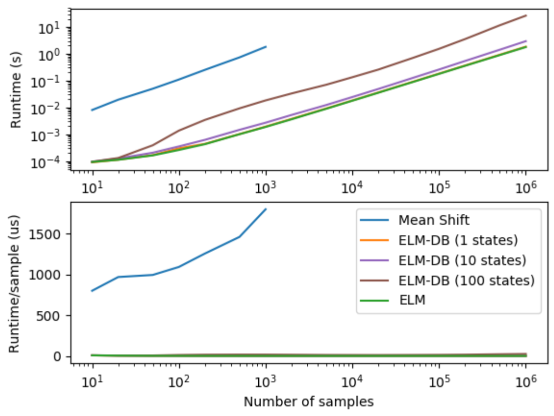


Figure 4.1: Comparing total runtime and runtime per sample for Mean Shift, ELM and ELM-DB using different number of states.

Figure 4.1 shows the average runtime on 50 datasets for each size. We can see that Mean Shift is clearly slower than the other algorithms. This can easily be explained by the large number of calculations it needs to perform when iteratively moving each point. It must also be noted that Mean Shift is implemented in Python, giving it a disadvantage to the other algorithms that were written in C.

More importantly, as Mean Shift is not a streaming algorithm, the processing time per sample increases as the dataset become larger. Even if Mean Shift would be faster than ELM and ELM-DB for small datasets, it would eventually become slower when the datasets becomes large enough due to the exponential nature of the algorithm. Streaming algorithms like ELM and ELM-DB on the other hand, scale linearly with the size of the dataset. When we compare ELM-DB with ELM, we see that it has a comparable performance when using one state. Increasing the number of states in ELM-DB clearly increases the runtime of the algorithm.

4.2. Number of parallel states

As discussed in section 3.3, the number of parallel states needs to be limited to the best n states in order to maintain performance. In this section I will analyse the effect of this setting on the cluster results. For this experiment I have compared the silhouette score and runtime when clustering with different number of states, on 50 datasets with 1000 samples. As the number of states will have a large influence on the runtime of the algorithm, this value should be kept as low as possible without affecting the cluster result too much.

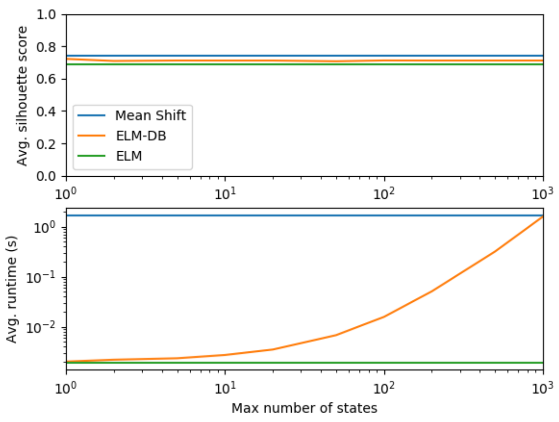


Figure 4.2: Comparing the average silhouette score and runtime from clustering 50 datasets with ELM-DB using different number of states. The silhouette score and runtime from Mean Shift and ELM have been added for comparison.

Figure 4.2 shows the average silhouette score and average runtime for 1 to 1000 states. The results from Mean Shift and ELM have been added to this plot for comparison. These results are shown as horizontal lines, as the number of states does not apply to these algorithms. We can see that ELM-DB is able to get a score of 0.72 using only one state, Which is close to the score from Mean Shift with a score of 0.74. The score is slightly lower (0.71) when using two or more states, but still higher than ELM which has a score of 0.68. At the same time we can see a large increase in runtime from $1.94ms$ with one state to $1.58s$ with 1000 states. The runtime of ELM-DB with one state is only 3.4% slower compared to ELM which has a runtime of $1.88ms$. These results show that using one state gives the best result. However, this does not have to be true for al scenarios. Some datasets might require a higher number of states to get a good clustering result.

It is interesting to see that the score is decreasing when ELM-DB is using more states. I have analysed the individual results for one and two states in more detail to locate a possible cause for this issue. From the 50 datasets clustered, 5 datasets return a lower score when using two states compared to using only one state. The ordering of states on decreasing approximate score is done correctly, the second state always has a lower approximate score than the first state. However, in these 5 datasets, the second state has a higher silhouette score than the first state. This shows an inaccuracy in the silhouette score approximation, used in ELM-DB: the approximation orders the states differently than the actual silhouette score would have done.

It must be noted though that the silhouette score approximation is based on the cluster structure in ELM-DB, while the actual silhouette score in this experiment is based on the labelled dataset. As described in section 3.3, samples might not always be labelled with the same cluster it has contributed to during clustering. This might have influenced the silhouette score results.

4.3. Parallel calculation on a commodity graphics card

In this thesis I have researched the use of decision branching on ELM. This branching will result in the parallel calculation of different states. When there are many parallel states to be processed, it might be useful to use a commodity graphics card.

A commonly used language for data processing on GPU(s) is CUDA. The GPU architecture works with the Single Instruction, Multiple Threads (SIMT) model. It is based on the concept of Single Instruction, Multiple Data (SIMD) where a single instruction can be performed on multiple sets of data simultaneously. In CUDA hundreds to thousands of threads are grouped together in to Warps, with each warp containing 32 threads. All threads in a warp will synchronously perform the same instructions on its own data. In order to effectively utilize the GPU, there needs to be enough parallelism to perform hundreds of calculations in parallel. Without this parallelism, calculation on GPU will not weight up to the performance of a CPU.

The results from fig. 4.2, show that ELM-DB produces the best result with one or at most a few states. This means that there is not enough parallelism in the algorithm to efficiently utilize a commodity graphics card.

4.4. Practical performance

In this section I will analyse the practical performance by using real life datasets from the UCI Machine Learning Repository [8]. I have selected four datasets with different sizes and number of features, as shown in table 4.1. The datasets Iris and Wine are the same datasets as have been used in the Evolving Local Means paper [2].

Table 4.1: Real life datasets from the UCI Machine Learning Repository [8] used to compare practical performance.

Datasets	# Instances	# Features	Name in UCI repository
Iris	150	4	Iris
Wine	178	13	Wine
Motion Capture	78095	36	Motion Capture Hand Postures
Power Consumption	2049281	7	Individual household electric power consumption

The datasets have been normalized with the `StandardScaler()` function from Scikit Learn. I have removed the Class column from the Iris, Wine and Motion Capture datasets, as this column contains the ground truth for these datasets. For the Motion Capture dataset, I have also removed the User column as this is not relevant for clustering. The missing values in this dataset have been replaced with zeros. In the Power Consumption dataset, I have removed the date and time columns. The rows with missing values (1.25% of the dataset) have been removed from this dataset. I will compare the performance of three different algorithms:

- **Mean Shift** Implementation from Scikit Learn in Python.
- **Evolving Local Means** My implementation in C, based on the algorithm in the paper.
- **ELM Decision Branching** The C implementation of my algorithm.

The cluster algorithms will be run with different radius settings to find the cluster result with the highest silhouette score. Mean Shift will only be used for the two smallest datasets Iris and Wine, as the other datasets would take too much time for a non-streaming algorithm. The silhouette scores of the Power Consumption dataset are based on a random subset of 100.000 samples.

The results from this experiment are shown in table 4.2. All clustering algorithms are getting the same silhouette score for the Iris dataset. For the Wine dataset, the streaming algorithms are getting a score of 0.30, while Mean Shift is only able to get a score of 0.22. ELM-DB is able to get a maximum score on a large range of radius settings when only using one state. This range becomes smaller as the number of states is increased. These first two datasets also show a large difference in runtime between Mean Shift and the streaming algorithms. The difference in runtime between ELM and ELM-DB with various number of states is very small. This is because ELM-DB uses at most 2 states when clustering Iris and at most 3 states for Wine.

Table 4.2: Comparison of clustering results from Mean Shift, ELM and ELM-DB on four real life datasets.

Dataset / Algorithm	Radius	Number of clusters	Silhouette score	Runtime	Compared to ELM
Iris					
Mean Shift	1.5 - 2.4	2	0.58	220 - 250 ms	
ELM	1.0 - 1.4	2	0.58	0.463 - 0.482 ms	
ELM-DB (1 state)	1.0 - 1.4	2	0.58	0.479 - 0.484 ms	+3.5 %
ELM-DB (2 states)	1.0 - 1.4	2	0.58	0.474 - 0.493 ms	+2.4 %
ELM-DB (5 states)	1.0 - 1.4	2	0.58	0.474 - 0.484 ms	+2.4 %
ELM-DB (10 states)	1.0 - 1.4	2	0.58	0.460 - 0.474 ms	-0.6 %
ELM-DB (100 states)	1.0 - 1.4	2	0.58	0.464 - 0.481 ms	+0.2 %
Wine					
Mean Shift	4.0 - 4.2	2	0.22	428 - 568 ms	
ELM	2.2 - 2.6	2	0.30	1.38 - 1.44 ms	
ELM-DB (1 state)	2.1 - 3.0	2	0.30	1.46 - 1.58 ms	+5.6 %
ELM-DB (2 states)	2.4 - 3.0	2	0.30	1.52 - 1.68 ms	+9.9 %
ELM-DB (5 states)	2.7 - 3.0	2	0.30	1.47 - 1.52 ms	+6.3 %
ELM-DB (10 states)	2.7 - 3.0	2	0.30	1.43 - 1.53 ms	+3.7 %
ELM-DB (100 states)	2.7 - 3.0	2	0.30	1.46 - 1.56 ms	+5.8 %
Motion Capture					
ELM	4.8 - 5.3	2	0.48	1.21 - 1.26 s	
ELM-DB (1 state)	5.8	2	0.46	1.24 s	+2.5 %
ELM-DB (2 states)	5.8	2	0.46	1.33 s	+9.5 %
ELM-DB (5 states)	5.7 - 5.8	2	0.48	1.35 - 1.46 s	+11 %
ELM-DB (10 states)	5.8	2	0.48	1.37 s	+13 %
ELM-DB (100 states)	5.8	2	0.48	1.35 s	+12 %
Power Consumption					
ELM	3.3 - 3.5	2	0.65	8.70 - 9.11 s	
ELM-DB (1 state)	2.8	7	0.60	12.2 s	+40 %
ELM-DB (2 states)	3.5	2	0.65	10.6 s	+22 %
ELM-DB (5 states)	3.5	2	0.66	15.9 s	+83 %
ELM-DB (10 states)	3.5	2	0.65	20.1 s	+132 %
ELM-DB (100 states)	3.5	2	0.65	483 s	+5456 %

The results from the third dataset, Motion Capture, show that ELM-DB is not able to get the same silhouette score as ELM with only one or two states. The difference in runtime between these algorithms is small, even when increasing the number of states. ELM-DB uses at most 6 states when clustering this dataset.

The Power Consumption dataset shows more differences. ELM-DB is giving a clearly different cluster result with 7 clusters when using only one state. When using two or more states, ELM-DB is producing results similar to ELM. The silhouette score is even higher than ELM when using 5 states, with a silhouette score of 0.66 compared to 0.65 for ELM. This dataset also shows the large time penalty when using more states. ELM-DB is almost 40 times slower when using 100 states, compared to using only one state. This is the only dataset where ELM-DB utilizes all 100 states.

Overall, we can see that ELM-DB is able to produce the same results as ELM while using up to 22% more time. On the largest dataset, ELM-DB is even able to produce a higher Silhouette score, but this comes at a cost of a 83% longer runtime than ELM.

This chapter has focussed on the runtime performance as well as the practical performance of ELM Decision Branching. We have compared the runtime of ELM-DB using different number of states with Mean Shift and ELM. We have also analysed the effect on cluster performance when using different number of states. This allows us to answer the second sub-question:

Question 2: What is GPU acceleration and can it be used to improve the performance of decision branching on Evolving Local Means?

The results show that ELM-DB is able to achieve the highest average silhouette score using only one state on the synthetic datasets as visible in fig. 4.2. For the real life datasets, up to 5 states are required to achieve the highest silhouette score in ELM-DB, as can be seen in table 4.2. GPU acceleration is a useful tool for algorithms that contain a large number of parallelism. As described in section 4.3, ELM-DB performs best with a low number of states. This means that there is not enough parallelism in the ELM-DB algorithm to efficiently utilize a GPU.

The last sub-question focusses on the practical performance of ELM Decision Branching:

> ***Question 3: What is the practical performance of Evolving Local Means with decision branching compared to the original Evolving Local Means algorithm and Mean Shift?***

In section 4.4, I have analysed the practical performance of ELM-DB compared to Mean Shift and ELM using four real life datasets. Table 4.2 shows that ELM-DB is able to achieve the same silhouette score as ELM using up to 5 states and that this takes up to 22% more time, compared to ELM. For the largest dataset Power Consumption, ELM-DB is even able to improve the silhouette score from ELM using 83% more time.

5 Conclusion and discussion

In this research I have shown the effect of decision branching on the streaming clustering algorithm Evolving Local Means (ELM). I have extended the ELM algorithm with decision branching to create a new algorithm called ELM Decision Branching (ELM-DB). In this chapter I am going to answer the research questions I have stated in the introduction. These answers will be used to form a conclusion. After the conclusion there will be a discussion of the research together with some recommendations for future work.

5.1. Answering the research questions

Chapter 1 has introduced the three sub-questions and main research question for this research. In this section I will discuss each of these sub-questions and the answers I have derived in this research. With the answers to these sub-questions I will be able to answer the main research question and form a conclusion for this research.

Question 1: What is decision branching and can it be used to improve the cluster quality of Evolving Local Means? Decision branching allows important decisions to be postponed by processing all possible cases in parallel until the decision can be made. It is used in two places in the ELM-DB algorithm. The first place where decision branching will be used is when deciding whether a new sample should be added to the nearest cluster or should be turned into a new cluster. Samples very close to a cluster will always be added to the cluster, while samples far from the nearest cluster will always create a new cluster. Only when this decision is not clear, will the algorithm branch and continue both options. This approach prevents unnecessary branching to reduce the amount of calculations.

If a sample has been added to a cluster, the algorithm will check how close this cluster is to the nearest neighbouring cluster. This is the second place where decision branching may occur. When the clusters are very close together, the algorithm will always merge the clusters together. Similarly, ELM-DB will never merge clusters that are far away from each other. Only when the decision is unclear, will decision branching be used to continue both options.

The branches are implemented as a list of states. The algorithm starts with one state and will duplicate a state whenever branching is required. Each iteration, the algorithm will update all states with the new sample. After all states have been updated with the new sample, the algorithm will remove any duplicate states caused by the branching. It will also reduce the total number of states by removing states with the lowest score. This score is based on an approximation of the silhouette score that can be calculated within the streaming limitations of the algorithm.

Experiments on 1000 synthetic datasets, with a minimum distance of r between the clusters, show that ELM-DB is able to improve the average silhouette score compared to ELM. Both Mean Shift and ELM-DB were able to cluster all datasets correctly. 5 datasets were not clustered correctly by ELM, returning a single cluster for these datasets. ELM-DB is also able to maintain better cluster results than ELM when there is no minimum distance between the clusters. ELM was unable to produce a correct cluster result for 68 out of 1000 datasets, returning a single cluster as result. While, Mean Shift and ELM-DB returned a single cluster result for only 10 datasets. This means that ELM-DB can handle clusters close to each other better than ELM. The decision branching also allows ELM-DB to produce good clustering results over a larger range of radius settings than ELM, making it more versatile to use.

Increasing the number of states used by ELM-DB has a large impact on the performance of the algorithm. However, experiments on synthetic datasets show the highest average Silhouette score of 0.72 when using only one state. This score drops slightly to 0.71 when using two or more states. This means it is not useful to use a large number of states when clustering, as it would only increase runtime without improving the cluster result. Using only one state, ELM-DB is only 3.4% slower than ELM.

Question 2: What is GPU acceleration and can it be used to improve the performance of decision branching on Evolving Local Means? GPU acceleration using CUDA can be a useful tool for algorithms that contain a lot of parallelism. The main parallel aspect in ELM-DB is the calculation of multiple states. As the algorithm performs best using up to five states, there is not enough parallelism to efficiently utilize a commodity graphics card.

Question 3: What is the practical performance of Evolving Local Means with decision branching compared to the original Evolving Local Means algorithm and Mean Shift? Results from four real life datasets show that ELM-DB is able to get the same Silhouette score as ELM using up to 5 states and up to 22% more time. On the largest dataset, ELM-DB is even able to produce a higher silhouette score than ELM. This requires a 83% longer runtime compared to ELM.

With the answers from each of the sub-questions we are now able to answer the main research question:

Can streaming cluster analysis with Evolving Local Means be improved using decision branching?

ELM Decision Branching is able to prevent the issue of premature merge with the use of decision branching. This allows the algorithm is able to produce a higher average Silhouette score than ELM using only a few states. It is able to produce consistent results for a larger range of radius values than ELM, making it more versatile in use. This does come with a cost in terms of runtime, making ELM-DB up to 83% slower than ELM.

5.2. Discussion and future work

The results from this research are based on a random selection of synthetic datasets. Although multiple datasets have been used to create an average result, this selection of datasets is only a small sample from the infinite number of possible datasets. The synthetic datasets used in this research are generated using a Gaussian distribution. This distribution fits the use case of Mean Shift and ELM, expecting a cluster to become denser towards the centre. However, datasets in practice will not always be shaped like this. Secondly, both ELM and ELM-DB require clusters to be circular shaped as they define clusters using a centre and radius.

The silhouette score approximation used in ELM-DB to order the states, does not always produce the same order of states as the silhouette score would do. Although the accuracy is higher for better silhouette scores, this could still cause ELM-DB to return the wrong state as final result, as we have seen in section 4.2. We saw here that the score decreased when using two states instead of one state. This was caused by ELM-DB selecting a state with a lower silhouette score as best state. Improving the streaming silhouette score approximation would allow ELM-DB to make better decisions on which state contains the best (intermediate) result.

The labelling of the dataset is performed after the clustering, in a second iteration over the dataset. Each sample is labelled based on the nearest cluster using the centre μ and distance parameter σ. This means that a sample can be labelled with a different cluster than it has contributed to in the clustering process.

Experiments using the real life datasets show that ELM and ELM-DB produces the highest silhouette score when using a large radius, resulting in only two clusters on nearly all cases. The streaming algorithms returned more clusters when the radius is decreased, but this resulted in a lower silhouette score. The silhouette score might thus not be the best cluster performance metric for ELM and ELM-DB. It would be interesting to analyse the results using other cluster performance metrics.

ELM-DB is using multiple states to calculate different branches. Section 4.3 has discussed the parallel calculation of these states using GPU acceleration. Although there is not enough parallelisation in the algorithm to efficiently utilize a GPU, It would be interesting to analyse whether performance can be improved by using multithreading on CPU.

The results from clustering real life datasets show that Mean Shift achieves a lower silhouette score of 0.22 for dataset Wine than ELM and ELM-DB with a score of 0.30. I would expect from Mean Shift to produce the highest score, as it is not constrained by streaming limitations. It would be interesting to analyse what caused Mean Shift to return a lower silhouette score than ELM and ELM-DB.

Bibliography

[1] Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni. Streaming K-means Approximation. In *Proceedings of the 22Nd International Conference on Neural Information Processing Systems*, NIPS'09, pages 10–18, USA, 2009. Curran Associates Inc. ISBN 9781615679119. URL http://dl.acm.org/citation.cfm?id=2984093.2984095. event-place: Vancouver, British Columbia, Canada.

[2] R. Dutta Baruah and P. Angelov. Evolving local means method for clustering of streaming data. In *2012 IEEE International Conference on Fuzzy Systems*, pages 1–8, June 2012. doi: 10.1109/FUZZ-IEEE.2012.6251366.

[3] Vladimir Braverman, Adam Meyerson, Rafail Ostrovsky, Alan Roytman, Michael Shindler, and Brian Tagiku. Streaming k-means on Well-Clusterable Data. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 26–40. Society for Industrial and Applied Mathematics, October 2011. doi: 10.1137/1.9781611973082.3.

[4] Feng Cao, Martin Estert, Weining Qian, and Aoying Zhou. Density-Based Clustering over an Evolving Data Stream with Noise. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, Proceedings, pages 328–339. Society for Industrial and Applied Mathematics, April 2006. ISBN 9780898716115. doi: 10.1137/1.9781611972764.29. URL https://epubs.siam.org/doi/10.1137/1.9781611972764.29.

[5] Feng Cao, Anthony K. H. Tung, and Aoying Zhou. Scalable Clustering Using Graphics Processors. In Jeffrey Xu Yu, Masaru Kitsuregawa, and Hong Va Leong, editors, *Advances in Web-Age Information Management*, Lecture Notes in Computer Science, pages 372–384. Springer Berlin Heidelberg, 2006. ISBN 9783540352266.

[6] Miguel Carreira-Perpiñán. A review of mean-shift algorithms for clustering. *arXiv:1503.00687 [cs, stat]*, March 2015. URL http://arxiv.org/abs/1503.00687. arXiv: 1503.00687.

[7] Jianqiang Dong, Fei Wang, and Bo Yuan. Accelerating BIRCH for Clustering Large Scale Streaming Data Using CUDA Dynamic Parallelism. In Hujun Yin, Ke Tang, Yang Gao, Frank Klawonn, Minho Lee, Thomas Weise, Bin Li, and Xin Yao, editors, *Intelligent Data Engineering and Automated Learning – IDEAL 2013*, Lecture Notes in Computer Science, pages 409–416. Springer Berlin Heidelberg, 2013. ISBN 9783642412783. doi: 10.1007/978-3-642-41278-3_50.

[8] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[9] Reza Farivar, Daniel Rebolledo, Ellick Chan, and Roy H. Campbell. A Parallel Implementation of K-Means Clustering on GPUs. In *PDPTA*, pages 340–345, January 2008.

[10] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, January 1975. ISSN 0018-9448. doi: 10.1109/TIT.1975.1055330.

[11] Marwan Hassani, Ayman Tarakji, Lyubomir Georgiev, and Thomas Seidl. Parallel Implementation of a Density-Based Stream Clustering Algorithm Over a GPU Scheduling System. In Wen-Chih Peng, Haixun Wang, James Bailey, Vincent S. Tseng, Tu Bao Ho, Zhi-Hua Zhou, and Arbee L.P. Chen, editors, *Trends and Applications in Knowledge Discovery and Data Mining*, Lecture Notes in Computer Science, pages 441–453. Springer International Publishing, 2014. ISBN 9783319131863.

[12] B. Hong-tao, H. Li-li, O. Dan-tong, L. Zhan-shan, and L. He. K-Means on Commodity GPUs with CUDA. In *2009 WRI World Congress on Computer Science and Information Engineering*, volume 3, pages 651–655, March 2009. doi: 10.1109/CSIE.2009.491.

[13] Richard Hyde and Plamen Angelov. A new online clustering approach for data in arbitrary shaped clusters. In *2015 IEEE 2nd International Conference on Cybernetics (CYBCONF)*, pages 228–233, June 2015. doi: 10.1109/CYBConf.2015.7175937.

[14] Richard Hyde, Plamen Angelov, and A. R. MacKenzie. Fully online clustering of evolving data streams into arbitrarily shaped clusters. *Information Sciences*, 382-383:96–114, March 2017. ISSN 0020-0255. doi: 10.1016/j.ins.2016.12.004. URL http://www.sciencedirect.com/science/article/pii/S0020025516319247.

[15] Leonard Kaufman and Peter J. Rousseeuw, editors. *Finding Groups in Data*. John Wiley & Sons, Inc., mar 1990. ISBN 9780470316801. doi: 10.1002/9780470316801. URL https://onlinelibrary.wiley.com/doi/book/10.1002/9780470316801.

[16] Liang Men, Miaoqing Huang, and John Gauch. Accelerating Mean Shift Segmentation Algorithm on Hybrid CPU/GPU Platforms. In *Modern Accelerator Technologies for Geographic Information Science*, pages 157–166. hgpu.org, October 2012. doi: 10.1007/978-1-4614-8745-6_12. URL https://hgpu.org/?p=8340.

[17] 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.

[18] P. P. Rodrigues, J. Gama, and J. Pedroso. Hierarchical Clustering of Time-Series Data Streams. *IEEE Transactions on Knowledge and Data Engineering*, 20(5):615–627, May 2008. ISSN 1041-4347. doi: 10.1109/TKDE.2007.190727.

[19] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, November 1987. ISSN 0377-0427. doi: 10.1016/0377-0427(87)90125-7. URL http://www.sciencedirect.com/science/article/pii/0377042787901257.

[20] S. A. A. Shalom and M. Dash. Efficient Hierarchical Agglomerative Clustering Algorithms on GPU Using Data Partitioning. In *2011 12th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 134–139, October 2011. doi: 10.1109/PDCAT.2011.38.

[21] S. A. A. Shalom and M. Dash. Efficient Partitioning Based Hierarchical Agglomerative Clustering Using Graphics Accelerators With Cuda. *International Journal of Artificial Intelligence & Applications*, 4(2):13–33, March 2013. ISSN 0976-2191. doi: 10.5121/ijaia.2013.4202.

[22] S. A. Arul Shalom, Manoranjan Dash, and Minh Tue. Efficient K-Means Clustering Using Accelerated Graphics Processors. In Il-Yeol Song, Johann Eder, and Tho Manh Nguyen, editors, *Data Warehousing and Knowledge Discovery*, Lecture Notes in Computer Science, pages 166–175. Springer Berlin Heidelberg, 2008. ISBN 9783540858362.

[23] J. Sirotkovic, H. Dujmic, and V. Papic. Accelerating mean shift image segmentation with IFGT on massively parallel GPU. In *2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 279–285, May 2013.

[24] D. K. Tasoulis, N. M. Adams, and D. J. Hand. Unsupervised Clustering In Streaming Data. In *Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06)*, pages 638–642, December 2006. doi: 10.1109/ICDMW.2006.165.

[25] Q. Tu, J. F. Lu, B. Yuan, J. B. Tang, and J. Y. Yang. Density-based hierarchical clustering for streaming data. *Pattern Recognition Letters*, 33(5):641–645, April 2012. ISSN 0167-8655. doi: 10.1016/j.patrec.2011.11.022. URL http://www.sciencedirect.com/science/article/pii/S0167865511004120.

[26] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, August 1995. ISSN 0162-8828. doi: 10.1109/34.400568.

[27] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. *SIGMOD Rec.*, 25(2):103–114, June 1996. ISSN 0163-5808. doi: 10.1145/235968.233324. URL https://doi.org/10.1145/235968.233324.