Accelerated Mean Shift For Static And Streaming Environments

Daniel van der Ende*, Jean Marc Thiery†, and Elmar Eisemann‡
Delft University of Technology
Delft, The Netherlands
Email: *`daniel.vanderende@gmail.com`, †`j.thiery@tudelft.nl`, ‡`e.eisemann@tudelft.nl`

***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 article, we propose a novel acceleration strategy for the traditional Mean Shift algorithm, along with a two-layer 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 need to be answered at interactive rates, and for which no mean shift-based algorithm currently exists. Our online structure is updated at very minimal cost and as infrequently as possible, and we show how to detect the time at which an update needs to be triggered. Our technique is validated extensively in both static and streaming environments.**

Keywords*–*Data stream clustering; Mean Shift

I. Introduction

Although streams of data have been generated for a considerable amount of time, the analysis of these streams is a relatively young research field. Data streams present additional challenges for the field of data mining. Traditional data mining algorithms cannot be directly applied to data streams, due to the additional data stream constraints, which has led to considerable research into new methods of analyzing such high-speed data streams [1], [2].

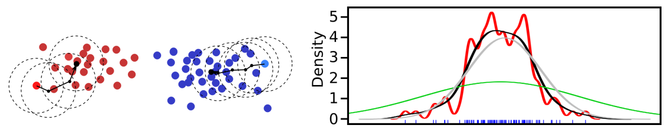
Mean Shift, initially proposed by Fukunaga et al. [3] and later generalized by Cheng et al. [4] and Comaniciu et al. [5], is a well-known clustering algorithm that has a number of attractive properties, such as its ability to find non-convex clusters. However, its performance has always been a concern, and it is because of this that, we believe, Mean Shift has never been applied in a streaming environment.

In this article, we present a modification of Mean Shift that can be used in both static and stream clustering environment. In the static environment, execution time of Mean Shift is reduced while a high level of cluster performance is maintained (Section III-A). Our main contribution concentrates on streaming environments, and we derive an efficient triggering mechanism, used to determine when a reclustering of the structure is necessary (Section III-B). We provide extensive experimental validation of our two contributions.

II. Background

A. Mean Shift

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.



(a) Each point performs an iterative gradient ascent of the estimated density towards a local maximum. (b) Estimated density (red, grey, black) using various bandwidths. Blue points are distributed according to the green density.

Figure 1. Mean Shift overview process

Considering an input data set $\mathcal{P} = \{\mathbf{p_i}\}$ in dimension d and a density kernel 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 density kernel. Upon convergence $p_i^\infty = \bar{p}_i$, where $\bar{p}_i$ is a local maximum of the density kernel. Points of $\mathcal{P}$, which converge towards the same local maximum are then clustered together. Figure 1(a) 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 density kernel. Figure 1(b) 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* $h \in \mathbb{R}$):

$$K(x,p) = \pi^{-d/2} \exp(-||x-p||^2/h^2) \qquad (1)$$

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 these two approaches are completely equivalent.

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) [6] is a common strategy to first transform the input data before applying Mean Shift.

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

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 [7]–[10] 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 [9].

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 limitation is its performance. As Fashing et al. [11] 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 [12]–[17]. Carreira-Perpiñán [12] 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 Section III.

B. Data Stream Clustering

A number of authors have assessed the complexity of mining data streams [18], [19]. Barbara [19] 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 Section III.

Many stream clustering algorithms use a two-phase 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 [20] maintains q micro-clusters online, followed by a modified k-means algorithm that is executed when a clustering query arrives. DenStream [21] is similar, exchanging the k-means algorithm for DBSCAN, and distinguishing various quality levels of micro-clusters. Finally, D-Stream [22] uses a sparse grid approach. All these three algorithms have complex parameters. Moreover, when a clustering query arrives, a clustering is always executed.

The Massive Online Analysis (MOA) [23] framework offers a sandbox environment for easy comparison of several stream clustering algorithms. Among those, D-Stream [22] is the most related to our approach. Even though, D-Stream is not a Mean Shift algorithm. It is a density-based approach, and it partitions the space by computing the connected components of the set for which the local density is higher than a given threshold. Other parts of the space are considered outliers (Mean Shift offers a soft characterization of outliers, through the number of points in clusters and the value of the density at the clusters' center). It integrates a particular kind of density decaying mechanism, whereas our approach allows for various windowing strategies. Further, our main contribution is a new triggering mechanism, which detects events for when the clustering needs to be updated. Although not investigated, our triggering mechanism could be used for D-Stream as well. Note that the purpose of this paper is not to claim the superiority of Mean Shift over other existing clustering algorithms. Each algorithm has its advantages and drawbacks.

III. Method

A. Static Clustering

As discussed in Section II-A3, 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). For each grid cell C_i, the number of points n_i assigned to it is maintained, along with the sum of these points S_i. This enables the computation of an average position of the points within the cell, denoted $\overline{C_i} = S_i/n_i$. 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 (see Algorithm 1). This is 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. It also 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 required property in streaming environments.

Algorithm 1 Update clustering of cells $\{C_i\}$.

```
for all cell $C_i$ do
    $\overline{C_i} = S_i/n_i$
end for
kdt = computeKdTree( $\{\overline{C_i}\}$ )
for all cell $C_i$ do
    $\hat{c}_i = \overline{C_i}$
    for $it < ItMax$ do
        $NN$=kdt.nearestNeighbors($\hat{c}_i$)
        $\hat{c}_i = \sum_{k \in NN} n_k K(\hat{c}_i, \overline{C_k})\overline{C_k} / \sum_{k \in NN} n_k K(\hat{c}_i, \overline{C_k})$
    end for
end for
cluster $\{C_i\}$ based on proximity of $\{\hat{c}_i\}$.
```

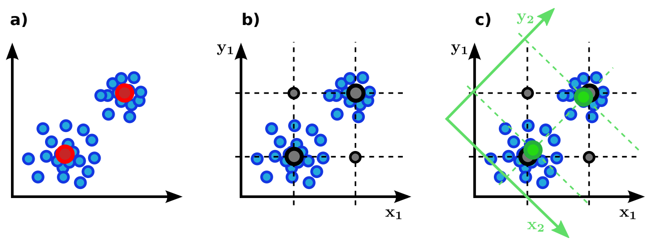


Figure 2. **a):** Two clusters in $2D$, with their centers in red. **b):** Clusters in each dimension (dot lines), whose product badly approximate the $2D$ clusters. **c):** Consensus over additional axes helps to identify the $2D$ clusters from the product of the $1D$ clusters of the projected data.

B. Stream Clustering

For stream clustering, the most common methodology is a two-phase approach, as discussed in Section II-B. A major disadvantage of this approach is that 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 and expensive computations.

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 article, we have applied both landmark and sliding windows of various sizes.

First, when the stream clustering is initialized, a static clustering, as described in Section III-A is performed. This clustering will serve as our initial reference clustering. On each stream iteration, we evaluate whether the 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. [24] as:

Lemma 3.1 (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, a $(k-1)$-dimensional subcluster should also change. We make use of this point and set up a collection of low-dimensional *data observers* (in our case, $1D$), which we can update efficiently when adding or removing points from the structure, and which will trigger a reclustering of the structure when necessary. Our algorithm is parametrized by the chosen distribution of observers as well as by their *sensitivity*.

Each observer i is defined as an histogram $\mathbf{H_i}$ of the data projected onto an axis $\mathbf{a_i}$. Because high-dimensional data usually overlaps in separate dimensions (see Figure 2), we consider not only the canonical axes $\{\mathbf{e_k}\}$, but also randomly distributed axes in $\mathbb{R}^d$, and we will define the final decision for the reclustering as a consensus over the observers. Since we cannot make any assumption on the upcoming data (e. g., align data using PCA), we 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 Figure 2(**c**)). All histograms are treated equally throughout.

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:

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

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:

$$D_{KL}(P \parallel Q) = \sum_k P(k) \ln \frac{P(k)}{Q(k)} \quad (3)$$

This measure is a distance, which is (symmetric and) always defined. Note that the direct use of the Kullback-Leibler divergence between $\bar{\mathbf{H}}_\mathbf{i}$ and $\mathbf{H_i}$ results in $+\infty$ in cases where points are removed from a cell, i. e., $Q(k) = 0$ in (3).

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

A reclustering is then decided if the proportion of histogram voting for a reclustering is larger than a random variable, which we take between 0 and 1. This procedure is a standard Monte Carlo 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 (for which pseudo-code is given in Algorithm 2) is easily maintainable in a streaming environment, as it 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.

Algorithm 2 Add (remove) p during streaming.

```
Require: saved histograms $\{\bar{\mathbf{H}}_\mathbf{i}\}$, sensitivity $\epsilon$
  grid $\leftarrow (\rightarrow)$ $p$                                   ▷ update grid
  $n^{\text{vote}}_{\text{histo}} = 0$
  for all histogram $\mathbf{H_i}$ with axis $\mathbf{a_i}$ do
      $\mathbf{H_i} \leftarrow (\rightarrow)\ \mathbf{a_i}^{\mathrm{T}} \cdot p$                         ▷ update $\mathbf{H_i}$
      $n^{\text{vote}}_{\text{histo}} += D_{JS}(\bar{\mathbf{H}}_\mathbf{i} \parallel \mathbf{H_i}) > \epsilon\ ?\ 1 : 0$   ▷ get vote of $\mathbf{H_i}$
  end for
  if $n^{\text{vote}}_{\text{histo}} > rand() * N^{\text{total}}_{\text{histo}}$ then
      for all histogram $\mathbf{H_i}$ do $\bar{\mathbf{H}}_\mathbf{i} = \mathbf{H_i}$        ▷ save $\mathbf{H_i}$ in $\bar{\mathbf{H}}_\mathbf{i}$
      end for
      require update of Mean Shift
  end if
```

IV. Empirical Results

A. Metrics

We have computed the following cluster validation metrics: Jaccard Index, Rand Index, Fowlkes-Mallows Index, Precision, Recall, F-Measure (see the work of Meila et al. [25]). 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. Furthemore, the metrics we chose are common in the data clustering scientific community and will hopefully provide a real insight into the behaviour of our algorithms to the reader. A value of 0 indicates completely different clusterings whereas 1 indicates identical ones.

We implemented our method in Python, since it is cross-platform and integrates well with real-world systems.

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 (e.g., summarized by 100 cells) and will therefore result in computed errors (based on the averages), which are much higher.

B. 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. Figure 3 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 regard points which are at the boundaries between visible clusters or outliers. 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 Section II-A1).

We have conducted experiments in higher dimension. 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 I summarizes our results on various datasets commonly found in the scientific literature.

While we experienced a reasonable gain in performance for small to reasonably big datasets, this is of small importance. Rather, we emphasize that our approach produces results which are consistent with the traditional Mean Shift. 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.

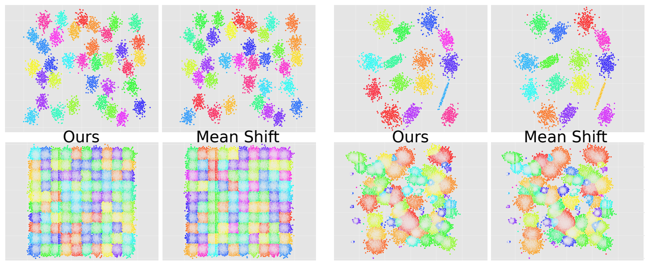


Figure 3. Comparison with Mean Shift on 2D data.

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

	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

C. 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 II summarizes the statistics of the conducted experiments, and the metrics plots for these test runs are presented in Figure 4. 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 $\epsilon = 0.001$ (Figure 4 (a)), and with $\epsilon = 0.003$ (Figure 4 (b)). The initial clustering was both times performed with $25k$ points, which is

TABLE II. Statistics of the experiments conducted in streaming environments. N: 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.

	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

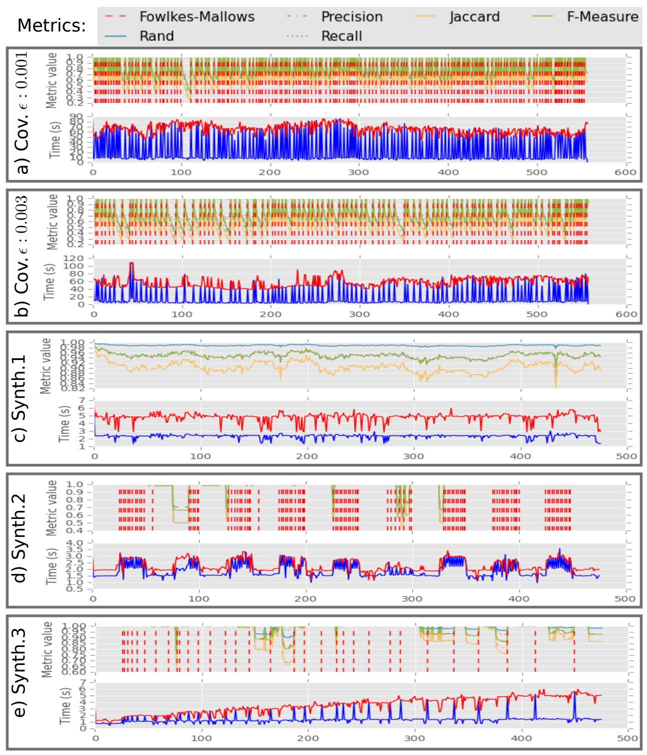


Figure 4. Comparison between our triggered clustering and constantly updated clustering on various datasets. For the timings, the red curve indicates the computation time per stream iteration when no triggering is used, and the blue curve indicates the one when our triggering mechanism is used. The vertical, dashed red lines indicate triggering events.

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 for example that, on this dataset, a state-of-the-art Mean Shift grid approach implemented in Scikit-learn [26] provided results with metric values of 0.2 (with the same grid parameters).

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

Experiment Synth.1 (Figure 4 (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.

Experiment Synth.2 (Figure 4 (d)), which was conducted under similar conditions as 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, Our triggering mechanism visibly 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 behaviours (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). 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 behaviour 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 behaviour for arbitrarily distributed data (over space and/or time).

V. Discussion

The experiments performed have shown that our algorithm produces accurate clusterings, at reduced cost, and only when 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 now discuss possible extensions of our method.

First, it may be possible to apply a divide-and-conquer approach to the overall data space using the information contained in the histograms. In some cases, the clusters in the data space are clearly separated. It could then be useful to split the space, based on this information, into separate areas using cutting hyperplanes, and run our method on these distinct subspaces. This would allow for greater parallelism and avoid unnecessary work being done on clusters that do not change. Hinneburg et al. [27] developed a clustering algorithm based only on such cutting hyperplanes. Their technique for finding the optimal cutting hyperplanes could be applied to the sparse grid used in the approach discussed in this article.

Second, data sparseness can reduce the performance improvement over Mean Shift to some extent. In these extreme cases, if each point is placed in a grid cell of its own, running the Mean Shift on these cell averages $\overline{C_i}$ is effectively the same as running on the input data. This is also dependent on the bandwidth value used. Note however, that this effect appears mostly for "small" datasets. For very big datasets, which we target, it becomes improbable to keep a high degree of sparsity.

Third, as was discussed in Section II-A2, the bandwidth value h to a large extent determines the final clustering result. Thus, the quality of the results are also dependent on the quality of the bandwidth estimate or the value provided by the user. This sensitivity to the bandwidth parameter is an inherent problem for all approaches based on a kernel density estimate. An interesting avenue of research could be to maintain clusterings for various bandwidth values, and to use this information to derive a continuous clustering as the interpolation of the computed ones. Currently, if the bandwidth is reset by the user during streaming, our approach cannot update the clustering efficiently.

VI. Conclusion and Future Work

In this paper, we have presented an effective and efficient modification of the Mean Shift. The modification allows for faster execution when applied in a static context, and makes it possible to use Mean Shift in a streaming environment. The application of Mean Shift in a streaming environment is based on a two-phase approach, where a memory-efficient structure is maintained online, and Mean Shift is executed on this summary structure offline. Our triggering mechanism ensures that the expensive process of executing Mean Shift happens as infrequently as possible and only when necessary to ensure a high clustering quality. Only if the data distribution has changed strongly Mean Shift is executed again. Our approach is extensively validated in both the static and streaming environments, and shows good performance in both. We believe that our triggering mechanism might be usable for other stream algorithms. However, designing optimal mechanisms for specific stream algorithms as well as for specific error measures is an interesting lead for future work.

Acknowledgments

This work was partly funded by Mobile Professionals BV and EU FET Project Harvest4D.

References

[1] C. C. Aggarwal, Ed., Data Streams - Models and Algorithms, ser. Advances in Database Systems. Springer, 2007, vol. 31.

[2] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. de Carvalho, and J. Gama, "Data stream clustering: A survey," ACM Computing Surveys (CSUR), vol. 46, no. 1, 2013, p. 13.

[3] K. Fukunaga and L. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," Information Theory, IEEE Transactions on, vol. 21, no. 1, 1975, pp. 32–40.

[4] Y. Cheng, "Mean shift, mode seeking, and clustering," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 17, no. 8, 1995, pp. 790–799.

[5] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 24, no. 5, 2002, pp. 603–619.

[6] 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, vol. 2, no. 11, 1901, pp. 559–572.

[7] 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, vol. 1. IEEE, 2001, pp. 438–445.

[8] D. Comaniciu, "An algorithm for data-driven bandwidth selection," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 25, no. 2, 2003, pp. 281–288.

[9] 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, vol. 91, no. 433, 1996, pp. 401–407.

[10] 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), 1991, pp. 683–690.

[11] M. Fashing and C. Tomasi, "Mean shift is a bound optimization," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 3, 2005, pp. 471–474.

[12] M. A. Carreira-Perpinan, "Acceleration strategies for gaussian mean-shift image segmentation," in Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on, vol. 1. IEEE, 2006, pp. 1160–1167.

[13] 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.

[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. IEEE, 2009, pp. 1818–1825.

[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. IEEE, 2003, pp. 456–463.

[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, vol. 3. IEEE, 2006, pp. 2385–2389.

[17] X.-T. Yuan, B.-G. Hu, and R. He, "Agglomerative mean-shift clustering," Knowledge and Data Engineering, IEEE Transactions on, vol. 24, no. 2, 2012, pp. 209–219.

[18] 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. ACM, 2002, pp. 1–16.

[19] D. Barbará, "Requirements for clustering data streams," ACM SIGKDD Explorations Newsletter, vol. 3, no. 2, 2002, pp. 23–27.

[20] 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. VLDB Endowment, 2003, pp. 81–92.

[21] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise." in SDM, vol. 6. SIAM, 2006, pp. 326–337.

[22] 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. ACM, 2007, pp. 133–142.

[23] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," The Journal of Machine Learning Research, vol. 11, 2010, pp. 1601–1604.

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

[25] M. Meilă, "Comparing clusteringsan information based distance," Journal of Multivariate Analysis, vol. 98, no. 5, 2007, pp. 873–895.

[26] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, 2011, pp. 2825–2830.

[27] A. Hinneburg and D. A. Keim, "Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering," in Proceedings of the 25th International Conference on Very Large Data Bases, ser. VLDB '99, 1999.