MeanShift++: Extremely Fast Mode-Seeking With Applications to Segmentation and Object Tracking

Jennifer Jang
Waymo
jangj@waymo.com

Heinrich Jiang
Google Research
heinrichj@google.com

Abstract

MeanShift is a popular mode-seeking clustering algorithm used in a wide range of applications in machine learning. However, it is known to be prohibitively slow, with quadratic runtime per iteration. We propose MeanShift++, an extremely fast mode-seeking algorithm based on MeanShift that uses a grid-based approach to speed up the mean shift step, replacing the computationally expensive neighbors search with a density-weighted mean of adjacent grid cells. In addition, we show that this grid-based technique for density estimation comes with theoretical guarantees. The runtime is linear in the number of points and exponential in dimension, which makes MeanShift++ ideal on low-dimensional applications such as image segmentation and object tracking. We provide extensive experimental analysis showing that MeanShift++ can be more than 10,000x faster than MeanShift with competitive clustering results on benchmark datasets and nearly identical image segmentations as MeanShift. Finally, we show promising results for object tracking.

1. Introduction

MeanShift [20, 22, 31] is a classical mode-seeking clustering algorithm that has a wide range of applications across machine learning and computer vision. Recent applications within computer vision include object tracking [54, 78, 47], unsupervised image segmentation [67, 13, 97], video segmentation [56, 24, 55], image restoration [4, 9], edge-preserving smoothing [55, 8, 12], point clouds [46, 80, 96], and remote sensing [51, 61, 18, 50]. More broadly in machine learning, MeanShift has been used for semi-supervised clustering [2, 73], manifold denoising [86, 82], matrix completion [83, 21], anomaly detection [6, 94, 70], as well as numerous problems in medical imaging [7, 69, 98, 49, 68, 53, 99, 101], wireless sensor networks [95, 100, 89, 64, 85, 62], and robotics [44, 45, 36, 43, 92, 16].

Given a set of examples, MeanShift proceeds in iterations, where in each iteration, each point is moved to the average of the points within a neighborhood ball centered at that point. The radius of the ball is a hyperparameter, often referred to as the bandwidth or window size. All initial examples that converge to the same point are clustered together and the points of convergence are estimates of the modes or local maximas of the probability density function. It has been shown that MeanShift implicitly performs a gradient ascent on the kernel density estimate of the examples [3]. MeanShift thus serves two purposes: mode-seeking and clustering.

MeanShift is often an attractive choice because it is non-parametric: unlike popular objective-based clustering algorithms such as $k$-means [5, 42] and spectral clustering [52, 79], it does not need to make many assumptions on the data, and the number of clusters is found automatically by the algorithm rather than a hyperparameter that needs to be set. In other words, MeanShift can adapt to general probability distributions. However, one of the main drawbacks of this procedure is its computational complexity: each iteration requires $O(n^2)$ computations. This is because for each example, calculating the window around the example is linear time in the worst case.

In this paper, we propose MeanShift++, a simple but effective procedure which first partitions the input space into a grid. Then, at each iteration, each point is assigned to its appropriate grid cell. We then approximate any point’s window with the average point in its and its neighboring grid cells. Each iteration in this procedure runs in linear time to the number of data points, with the cost of being exponential to the dimension of the feature space (since the size of the grid is exponential in dimension). Such a trade-off is ideal in settings with a large number of data points but low dimensionality, which is often the case in computer vision applications. With the growing size of modern datasets and increasing resolution of data collected by sensors and cameras, it is becoming ever more urgent to have fast versions of classical techniques.

Our contributions are as follows:
• We propose MeanShift++, a new mode-seeking procedure based on MeanShift that runs in $O(n \cdot 3^d)$ per iteration vs $O(n^2 \cdot d)$ for MeanShift. MeanShift++ has no additional hyperparameters over MeanShift.

• We show that MeanShift++’s grid-based approximation attains near minimax optimal statistical consistency guarantees at approximating the true density.

• An extensive empirical analysis shows that MeanShift++ performs at least as well as MeanShift for clustering while being significantly faster.

• Image segmentation results show that MeanShift++ delivers almost identical segmentations as MeanShift while being as much as 10,000x faster.

• Image segmentation experiments on the Berkeley Segmentation Dataset Benchmark (BSDS500) found that MeanShift++ performed on par or better than baselines despite being faster than most (and faster than MeanShift by 1,000x).

• We present a new object tracking algorithm based on MeanShift++ that can adapt to gradual color distributions and scene changes—something most MeanShift-based approaches cannot do due to the computational cost.

2. Related Works

Since MeanShift is a very popular procedure, there have been a number of approaches to speed up the algorithm and other mode-seeking based clustering algorithms in general.

Yang et al. (2003) [91] propose a speedup of MeanShift by using a fast Gauss transform to efficiently compute the kernel density estimator, reducing the computational complexity down to linear per iteration; however, they found the fast Gauss transform to be impractical for any dimension higher than 3. Yang et al. (2005) [90] then applies this technique to a modified similarity function specifically for color histograms and show its effectiveness on frame-tracking in image sequences. Elgamma (2003) [26] also leverage fast Gauss transform for color modeling and tracking.

Then there are other computer vision application specific speedup methods for MeanShift. Yin et al. (2011) [93] leverage frame-differences to speed up MeanShift in the specific application of target tracking. Carreira-Perpinan (2006) [13] shows that a spatial discretization strategy can accelerate Gaussian MeanShift image segmentation by one to two orders of magnitude while attaining almost the same segmentation. Carreira-Perpinan [14] also provides similar results for general Gaussian MeanShift; however, in this paper, we show that our method can achieve a far better improvement compared to MeanShift.

Another set of approaches leverage space-partitioning data structures in order to speed up the density estimation calculations. Wang et al. (2007) [81] propose using a dual-tree to obtain a faster approximation of MeanShift with provable accuracy guarantees. Xiao et al. (2010) [88] propose a heuristic to make the computations more efficient by approximating MeanShift using a greatly reduced feature space via applying an adaptive Gaussian KD-Tree.

Freedman et al. (2009) [29] propose speeding up MeanShift by randomly sampling data points when computing the kernel density estimates. This approach only reduces the runtime by small orders and does not address the underlying quadratic runtime issue unless only a small number of samples are used, but this leads to high error in the density estimates. Our method is both linear runtime and utilizes all of the data points to construct an optimal density estimator.

Vedaldi and Soatto (2008) [76] propose a procedure called QuickShift, which is modification of MeanShift in which the trajectories of the points are restricted to the original examples. The same procedure was proposed later by Rodriguez and Laio (2014) [63]. The procedure comes with theoretical guarantees [39], and GPU-based speedups have been proposed for the algorithm [32]. We will show later in the experimental results that QuickShift is indeed much faster than MeanShift, but MeanShift++ is still orders of magnitude faster than QuickShift.

3. Algorithm

We first introduce MeanShift in Algorithm 1 to cluster data points $X_n := \{x_1, \ldots, x_n\}$. The most popular version uses a unit flat kernel (i.e. $K(x) := 1[\|x\| \leq 1]$ [20]), but other kernels can be used as well including the Gaussian kernel [15]. At each iteration, it moves points to its kernel-weighted mean w.r.t. the last iteration’s points until stable. Rodriguez and Laio (2014) [63]. The procedure comes with theoretical guarantees [39], and GPU-based speedups have been proposed for the algorithm [32]. We will show later in the experimental results that QuickShift is indeed much faster than MeanShift, but MeanShift++ is still orders of magnitude faster than QuickShift.

**Algorithm 1 MeanShift [20, 22, 31]**

Inputs: bandwidth $h$, tolerance $\eta$, kernel $K$, $X_n$.

Initialize $y_{t,i} := x_i$ for $i \in [n]$, $t = 1$.

**do**

For $i \in [n]$:

$y_{t,i} \leftarrow \frac{\sum_{j \in [n]} K \left( \frac{\|y_{t-1,i} - y_{t-1,j}\|}{h} \right) y_{t-1,j}}{\sum_{j \in [n]} K \left( \frac{\|y_{t-1,i} - y_{t-1,j}\|}{h} \right)}$.

$t \leftarrow t + 1$.

**while** $\sum_{i=1}^n \|y_{t,i} - y_{t-1,i}\| \geq \eta$.

**return** $\{y_{t,1}, \ldots, y_{t,n}\}$. 
We now introduce MeanShift++ (Algorithm 2). In each iteration, it first performs a preprocessing step of partitioning the input data points into appropriate grid cells, which are hypercubes of side length $h$. A graphical comparison with MeanShift is shown in Figure 1. In practice, we keep two hash tables $C$ (to store the count in each cell) and $S$ (to store the sum in each cell). $C$ and $S$ are defined as initially empty mappings from lattice points in $D$-dimension (corresponding to grid cells) to non-negative integers and $\mathbb{R}^D$ respectively. We show how they are updated via a single pass through the dataset.

To compute which grid cell an example belongs to, we first divide each entry by $h$ and then take the element-wise floor function, which gives a $d$-dimensional integer vector index of the grids. Then, we can simply use the preprocessed information in the point’s grid cell and neighboring grid cells to compute the shifted point. As a result, each point is moved to the average of all the points within its cell and neighboring cells. This grid-based approach is a much faster approximation of the density at low dimensions. The runtime of MeanShift++ grows linearly with $n$, while MeanShift is quadratic (Figure 2).

**Algorithm 2 MeanShift++**

Inputs: bandwidth $h$, tolerance $\eta$, $X_{[n]}$.

Initialize $y_{0,i} := x_i$ for $i \in [n]$, $t = 1$.

do

Initialize empty hash tables $C : \mathbb{Z}^d \rightarrow \mathbb{Z}_{\geq 0}$ (stores cell count), $S : \mathbb{Z}^d \rightarrow \mathbb{R}^d$ (stores cell sum).

$C([yt_{i-1,i},i/h]) := C([yt_{i-1,i}/h]) + 1$ for $i \in [n]$.

$S([yt_{i-1,i}/h]) := S([yt_{i-1,i}/h]) + yt_{i-1,i}$ for $i \in [n]$.

Next, for all $i \in [n]$:

$$yt_{i,i} \leftarrow \frac{\sum_{\forall v \in (-1,0),d} S([yt_{i-1,i}/h] + v)}{\sum_{\forall v \in (-1,0),d} C([yt_{i-1,i}/h] + v)}.$$

$t \leftarrow t + 1$.

while $\sum_{i=1}^{n} \|yt_{i,i} - yt_{i-1,i}\| \geq \eta$.

return $\{yt_{1,1}, \ldots, yt_{n,n}\}$.

4. Theory

In this section, we give guarantees on our grid-based approach. Suppose there is some underlying distribution $P$ with corresponding density function $p : \mathbb{R}^d \rightarrow \mathbb{R}_{\geq 0}$ from which our data points $X_{[n]} = \{x_1, \ldots, x_n\}$ are drawn i.i.d. We show guarantees on the density estimator based on the grid cell counts.

We need the following regularity assumptions on the density function. The first ensures that the density function has compact support with smooth boundaries and is lower bounded by some positive quantity, and the other ensures that the density function has smoothness. These are standard assumptions in analyses on density estimation e.g. [35, 40, 19, 65].

**Assumption 1.** $p$ has compact support $X \subseteq \mathbb{R}^d$ and there exists $\lambda_0, r_0, C_0 > 0$ such that $p(x) \geq \lambda_0$ for all $x \in X$ and $\text{Vol}(B(x,r) \cap X) \geq C_0 \cdot \text{Vol}(B(x,r))$ for all $x \in X$ and $0 < r \leq r_0$, where $B(x,r) := \{x' \in \mathbb{R}^d : |x - x'| \leq r\}$.

**Assumption 2.** $p$ is $\alpha$-Hölder continuous for some $0 < \alpha \leq 1$: i.e. there exists $C_\alpha > 0$ such that $|p(x) - p(x')| \leq C_\alpha \cdot |x - x'|^\alpha$ for all $x, x' \in \mathbb{R}^d$.

We now give the result, which says that for $h$ sufficiently small depending on $p$ (if $h$ is too large, then the grid is too coarse to learn a statistically consistent density estimator), and $n$ sufficiently large, there will be a high probability finite-sample uniform bound on the difference between the density estimator and the true density. The proof can be found in the Appendix.
Theorem 1. Suppose Assumption 1 and 2 hold. Then there exists constants $C, C_1 > 0$ depending on $p$ such that the following holds. Let $0 < \delta < 1, 0 < h < \min\{\left(\frac{\lambda_0}{2C_0}\right)^{1/\alpha}, r_0\}$, $nh^d \geq C_1$. Let $\mathcal{G}_h$ be a partitioning of $\mathbb{R}^d$ into grid cells of edge-length $h$ and for $x \in \mathbb{R}^d$. Let $G(x)$ denote the cell in $\mathcal{G}_h$ that $x$ belongs to. Then, define the corresponding density estimator $\hat{p}_h(x)$ as:

$$\hat{p}_h(x) := \frac{|X| \cap G(x)|}{n \cdot h^d}.$$  

Then, with probability at least $1 - \delta$:

$$\sup_{x \in \mathbb{R}^d} |\hat{p}_h(x) - p(x)| \leq C \cdot \left(h^\alpha + \sqrt{\frac{\log(1/(\delta h))}{n \cdot h^d}}\right).$$

Remark 1. In the above result, choosing $h \approx n^{-1/(2\alpha + d)}$ optimizes the convergence rate to $\tilde{O}(n^{-\alpha/(2\alpha + d)})$, which is the minimax optimal convergence up to logarithmic factors for the density estimation problem as established by Tsybakov [72, 71].

In other words, the grid-based approach statistically performs at least as well as any estimator of the density function, including the density estimator used by MeanShift. It is worth noting that while our results only provide results for the density estimation portion of MeanShift++ (i.e. the grid-cell binning technique), we prove the near-minimax optimality of this estimation. This implies that the information contained in the density estimation portion serves as an approximately sufficient statistic for the rest of the procedure, which behaves similarly to MeanShift, which operates on another, also nearly-optimal density estimator. Thus, existing analyses of MeanShift e.g. [3, 17, 87, 48, 34, 33, 66] can be adapted here; however, it is known that MeanShift is very difficult to analyze [23] and a complete analysis is beyond the scope of this paper.

5. Experiments

We compare MeanShift++ against MeanShift on various clustering tasks in Table 1. These comparisons are made using the Scikit-Learn [57] implementation of MeanShift and our own implementation of MeanShift++ in Cython.

To measure the quality of a clustering result, we use the Adjusted Rand Index (ARI) [37] and the Adjusted Mutual Information (AMI) [77] scores, which compare the clustering with the partitioning induced by the labels of the data points, a popular way of comparing clustering performance [38]. The benchmark datasets we use are labeled datasets, and we only cluster the features.

As stated earlier, MeanShift++ is linear with respect to the number of data points and exponential in dimension. We thus show results on low-dimensional datasets. In Figure 3, we ran both algorithms on 19 benchmark datasets with 5 or fewer dimensions, ranging from less than 100 data points to millions of data points.

For the top five largest datasets, MeanShift failed to return a result for any setting of bandwidth despite running for more than 24 hours. MeanShift++ consistently outperformed MeanShift in both clustering quality and runtime for the rest of the datasets, as shown in Table 2. We saw a significant speed reduction of over 100x on both small and large datasets, showing that MeanShift++ does not have significantly more overhead costs than MeanShift either.

We also show in Figure 3 that the effect the bandwidth setting has on clustering performance and runtime for a few of the datasets to provide further insight into the stability of the procedures under the bandwidth hyperparameter.

We note that MeanShift++ outperforms MeanShift on many datasets, possibly due to a regularizing effect: by partitioning the space into grids and assigning every point in the same cell the same value instead of a unique value for each point, the gradient-ascent shifting step is more stable than in MeanShift. This regularization effect, combined with the option to tune the cell-length which essentially controls the amount of regularization, allows MeanShift++ to outperform MeanShift in some cases.

However, these results are unlikely to generalize to higher dimensions. It is known that density-based procedures perform poorly in high dimensions due to the curse of dimensionality. Our theoretical results also show that rates become exponentially worse in higher dimension.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$n$</th>
<th>$d$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a Phone Accelerometer</td>
<td>13,062,475</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>b Phone Gyroscope</td>
<td>13,932,632</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>c Watch Accelerometer</td>
<td>3,540,962</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>d Watch Gyroscope</td>
<td>3,205,431</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>eStill</td>
<td>949,983</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>f Skin</td>
<td>245,057</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>g Iris</td>
<td>150</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>h Lupus</td>
<td>87</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>i Confidence</td>
<td>72</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>j Geyser</td>
<td>22</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>k Balance Scale</td>
<td>625</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>l Vinnie</td>
<td>380</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>m Sleep Data</td>
<td>1,024</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>n Transplant</td>
<td>131</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>o Slope</td>
<td>44</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>p PRNN</td>
<td>250</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>q Wall Robot</td>
<td>5,456</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>r User Knowledge</td>
<td>403</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 1. Summary of datasets used. Includes dataset size ($n$), number of features ($d$), and number of clusters ($c$).
6. Image Segmentation

We compare MeanShift++ to a number of baselines for unsupervised image segmentation in Figure 4. We include Felzenszwalb [27], QuickShift [76], and $k$-means, three popular image segmentation procedures from the Python Scikit-Image library [74], as well as Quickshift++ [41], a recent algorithm shown to be an improvement over Quickshift on image segmentation. We also include MeanShift, which often produces qualitatively better clusters than the other baselines, but runs for so much longer that it is impractical for high-resolution image segmentation.

For image segmentation, we run each algorithm on a preprocessed image with each pixel represented in a 3D RGB color channel space, with the exception of Quickshift++, which takes ($r, g, b, x, y$) color and spatial coordinates. MeanShift was run with both ($r, g, b$) (shown in Figure 4) and ($r, g, b, x, y$) inputs, but we did not see a difference in segmentation quality or runtime. For each algorithm, the returned clusters are taken as the segments.

Our image segmentation experiments in Figure 4 show that MeanShift++ is able to produce segmentations that are nearly identical to that of MeanShift with an up to 10,000x speedup. We capped the sizes of our images at 187,500 pixels to allow MeanShift to finish running, so this speedup would surely be greater on even higher resolution images.

Multiple attempts have been made to speed up MeanShift for image segmentation at the cost of quality, but MeanShift++ does not seem to trade off segmentation quality despite running in a sub-fraction of the time.

For a more quantitative comparison, we ran experiments using the Berkeley Segmentation Dataset Benchmark (BSDS500) of 500 images with 6 human-labeled segmentations each. We ran MeanShift++, MeanShift, SLIC, and QuickShift on each image and used the adjusted RAND index [37] (ARI) and Fowlkes-Mallows [28] (FM) scores to compare the clusters. Scores were averaged over the 6 ground truth segmentations. We found that MeanShift++ performed on par or better than baselines despite being faster than MeanShift by 1,000x on average (Figure 6).

7. Object Tracking

The mode-seeking behavior of MeanShift makes it a good candidate for visual tracking. A basic implementation would take in a mask and/or search window of an object and build a histogram of the colors found in that object in RGB or HSV space. Afterwards, the histogram is normalized by the colors found in surrounding, non-target points into a probability distribution. At each step, the tracking algorithm would backproject each point in the window into a probability that it is part of the original object. The center of the window moves in ($x, y$) space to the mode of the distribution within the window until convergence in the same way that MeanShift iteratively moves each point to the mean of its neighbors. Every frame thereafter would be initialized with the final window of the previous frame.

MeanShift++ can be used for object tracking in a similar, albeit more principled way (Algorithm 3). Instead of color histograms, which need to be computed and require extra hyperparameters to determine the size of bins, the grid cells generated by MeanShift++ during clustering are already precomputed and suitable for tracking: we can quickly compute which points fall into any bin belonging to the target cluster(s).
MeanShift is also too slow to generate masks needed for real-time tracking. It often requires the user to provide a precomputed mask or color range. Relying on user input is imperfect and subject to biases. MeanShift++ is fast enough to generate masks through real-time clustering.

Off-the-shelf versions of MeanShift tracking rely on a histogram calculated from the original frame throughout the whole scene. This does not work well if the illumination in the scene changes, since the algorithm cannot make fast updates to the histogram [30, 84, 58, 59]. MeanShift++ can adapt to changing color distributions by finding and adding neighboring grids of points to the histogram in linear time, making it more robust to gradual changes in lighting, color, and other artifacts of the data.

CamShift [11, 1] improves MeanShift tracking by adjusting window sizes as objects move closer or farther away and updating color histograms based on lighting changes, among other things. Future work may involve adapting these ideas to a MeanShift++-based tracking algorithm.

In Figure 5, we show the performance of MeanShift++

<table>
<thead>
<tr>
<th>Original</th>
<th>MeanShift++</th>
<th>MeanShift</th>
<th>Felzenszwalb</th>
<th>Quickshift</th>
<th>K – Means</th>
<th>Quickshift++</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Original Image" /></td>
<td><img src="image2" alt="MeanShift++" /></td>
<td><img src="image3" alt="MeanShift" /></td>
<td><img src="image4" alt="Felzenszwalb" /></td>
<td><img src="image5" alt="Quickshift" /></td>
<td><img src="image6" alt="K – Means" /></td>
<td><img src="image7" alt="Quickshift++" /></td>
</tr>
<tr>
<td>1.22s</td>
<td>1h 40m</td>
<td>0.23s</td>
<td>29.00s</td>
<td>0.15s</td>
<td>1m 29s</td>
<td></td>
</tr>
<tr>
<td><img src="image8" alt="Original Image" /></td>
<td><img src="image9" alt="MeanShift++" /></td>
<td><img src="image10" alt="MeanShift" /></td>
<td><img src="image11" alt="Felzenszwalb" /></td>
<td><img src="image12" alt="Quickshift" /></td>
<td><img src="image13" alt="K – Means" /></td>
<td><img src="image14" alt="Quickshift++" /></td>
</tr>
<tr>
<td>2.25s</td>
<td>1h 35m</td>
<td>0.97s</td>
<td>1m 25s</td>
<td>0.51s</td>
<td>51.42s</td>
<td></td>
</tr>
<tr>
<td><img src="image15" alt="Original Image" /></td>
<td><img src="image16" alt="MeanShift++" /></td>
<td><img src="image17" alt="MeanShift" /></td>
<td><img src="image18" alt="Felzenszwalb" /></td>
<td><img src="image19" alt="Quickshift" /></td>
<td><img src="image20" alt="K – Means" /></td>
<td><img src="image21" alt="Quickshift++" /></td>
</tr>
<tr>
<td>1.64s</td>
<td>2h 42m</td>
<td>0.51s</td>
<td>5m 49s</td>
<td>0.25s</td>
<td>4m 40s</td>
<td></td>
</tr>
<tr>
<td><img src="image22" alt="Original Image" /></td>
<td><img src="image23" alt="MeanShift++" /></td>
<td><img src="image24" alt="MeanShift" /></td>
<td><img src="image25" alt="Felzenszwalb" /></td>
<td><img src="image26" alt="Quickshift" /></td>
<td><img src="image27" alt="K – Means" /></td>
<td><img src="image28" alt="Quickshift++" /></td>
</tr>
<tr>
<td>3.70s</td>
<td>2h 16m</td>
<td>0.61s</td>
<td>1m 23s</td>
<td>0.35s</td>
<td>1m 4s</td>
<td></td>
</tr>
<tr>
<td><img src="image29" alt="Original Image" /></td>
<td><img src="image30" alt="MeanShift++" /></td>
<td><img src="image31" alt="MeanShift" /></td>
<td><img src="image32" alt="Felzenszwalb" /></td>
<td><img src="image33" alt="Quickshift" /></td>
<td><img src="image34" alt="K – Means" /></td>
<td><img src="image35" alt="Quickshift++" /></td>
</tr>
<tr>
<td>1.21s</td>
<td>21m 26s</td>
<td>0.21s</td>
<td>30.57s</td>
<td>0.20s</td>
<td>1m 6s</td>
<td></td>
</tr>
<tr>
<td><img src="image36" alt="Original Image" /></td>
<td><img src="image37" alt="MeanShift++" /></td>
<td><img src="image38" alt="MeanShift" /></td>
<td><img src="image39" alt="Felzenszwalb" /></td>
<td><img src="image40" alt="Quickshift" /></td>
<td><img src="image41" alt="K – Means" /></td>
<td><img src="image42" alt="Quickshift++" /></td>
</tr>
<tr>
<td>4.38s</td>
<td>4h 24m</td>
<td>0.79s</td>
<td>1m 3s</td>
<td>0.50s</td>
<td>27.80s</td>
<td></td>
</tr>
<tr>
<td><img src="image43" alt="Original Image" /></td>
<td><img src="image44" alt="MeanShift++" /></td>
<td><img src="image45" alt="MeanShift" /></td>
<td><img src="image46" alt="Felzenszwalb" /></td>
<td><img src="image47" alt="Quickshift" /></td>
<td><img src="image48" alt="K – Means" /></td>
<td><img src="image49" alt="Quickshift++" /></td>
</tr>
<tr>
<td>6.58s</td>
<td>2h 1m</td>
<td>0.56s</td>
<td>1m 6s</td>
<td>0.47s</td>
<td>27.74s</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4. *Comparison of six image segmentation algorithms.* We show the results of MeanShift++, MeanShift, Quickshift++, and three other popular image segmentation algorithms from the Scikit-Image library[74]. MeanShift returns qualitatively good results on image segmentation but takes very long to run. MeanShift++ returns segmentations that are the most similar to MeanShift with an up to 10,000x speedup. We expect the speedup to be far greater for high resolution images—the images shown here are low resolution (under 200k pixels).
Figure 5. Comparison of MeanShift++ and MeanShift on object tracking. Unlike MeanShift++, MeanShift is too slow to generate masks for real-time object tracking. In practice, the user manually provides a color range that they want to track, which is often incomplete, inaccurate, or biased. Here, we initialize both MeanShift++ and MeanShift with a mask from clustering results generated by MeanShift++ to save time. For MeanShift, we use OpenCV’s [10] implementation of color histograms to track the object in question. For MeanShift++, we naturally use the grid cells that are returned from the clustering step. We find that MeanShift is more likely to get distracted by backgrounds, foregrounds, and other objects in the scene. **First scene**: MeanShift returns less accurate object centers and search windows. **Second scene**: MeanShift fails to find the object altogether due to an abundance of similar colors in the frame that cannot be decoupled from the object of interest. **Third scene**: MeanShift starts tracking similar objects nearby when the original objective moves out of frame. In contrast, MeanShift++ stops tracking when it finds the center of mass in the search window disappear. **Fourth scene**: MeanShift loses the skater faster than MeanShift++ and fails to find him again (instead it starts to track another skater altogether).
Table 2. Summary of clustering performances. MeanShift++’s and MeanShift’s best results for 19 real-world datasets after tuning bandwidth. Datasets from the UCI Machine Learning Repository [25] and OpenML [75]. In cases where the original target variable is continuous, binarized versions of the datasets were used. These experiments were run on a local machine with a 1.2 GHz Intel Core M processor and 8 GB memory. MeanShift did not finish (DNF) within 24 hours for the top five largest datasets. ARI, AMI, and runtime are reported for each run, and the highest score obtained for that metric and dataset is bolded.

<table>
<thead>
<tr>
<th>a) Phone Accelerometer</th>
<th>b) Phone Gyroscope</th>
<th>c) Watch Accelerometer</th>
<th>d) Watch Gyroscope</th>
<th>e) Still</th>
<th>f) Skin</th>
<th>g) Iris</th>
<th>h) Lupus</th>
<th>i) Confidence</th>
<th>j) Geyser</th>
<th>k) Balance Scale</th>
<th>l) Vinnie</th>
<th>m) Sleep Data</th>
<th>n) Transplant</th>
<th>o) Slope</th>
<th>p) PRNN</th>
<th>q) Wall Robot</th>
<th>r) User Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS++</td>
<td>MS</td>
<td>MS++</td>
<td>MS</td>
<td>MS++</td>
<td>MS++</td>
<td>MS++</td>
<td>MS++</td>
<td>MS++</td>
<td>MS++</td>
<td>MS++</td>
<td>MS++</td>
<td>MS++</td>
<td>MS++</td>
<td>MS++</td>
<td>MS++</td>
<td>MS++</td>
<td>MS++</td>
</tr>
<tr>
<td>0.0897</td>
<td>DNF</td>
<td>0.1959</td>
<td>DNF</td>
<td>0.2354</td>
<td>DNF</td>
<td>0.1835</td>
<td>DNF</td>
<td>0.0913</td>
<td>DNF</td>
<td>0.2309</td>
<td>DNF</td>
<td>0.1595</td>
<td>DNF</td>
<td>0.1336</td>
<td>DNF</td>
<td>0.7900</td>
<td>DNF</td>
</tr>
<tr>
<td>29m 59s</td>
<td>&gt;24h</td>
<td>49m 19s</td>
<td>&gt;24h</td>
<td>1h 35m</td>
<td>&gt;24h</td>
<td>32m 32s</td>
<td>&gt;24h</td>
<td>17m 52s</td>
<td>&gt;24h</td>
<td>43m 32s</td>
<td>&gt;24h</td>
<td>24m 45s</td>
<td>&gt;24h</td>
<td>9m 3s</td>
<td>&gt;24h</td>
<td>13.12s</td>
<td>&gt;24h</td>
</tr>
<tr>
<td>0.5681</td>
<td>&lt;0.01s</td>
<td>0.6832</td>
<td>&lt;0.01s</td>
<td>0.1827</td>
<td>&lt;0.01s</td>
<td>0.1399</td>
<td>&lt;0.01s</td>
<td>0.2080</td>
<td>&lt;0.01s</td>
<td>0.2059</td>
<td>&lt;0.01s</td>
<td>0.2455</td>
<td>0.2215</td>
<td>0.2409</td>
<td>0.2198</td>
<td>0.0883</td>
<td>0.0836</td>
</tr>
<tr>
<td>&lt;0.01s</td>
<td>6.35s</td>
<td>&lt;0.01s</td>
<td>3.82s</td>
<td>&lt;0.01s</td>
<td>7.90s</td>
<td>&lt;0.01s</td>
<td>0.98s</td>
<td>&lt;0.01s</td>
<td>0.98s</td>
<td>&lt;0.01s</td>
<td>6.02s</td>
<td>&lt;0.01s</td>
<td>6.02s</td>
<td>&lt;0.01s</td>
<td>0.99s</td>
<td>&lt;0.01s</td>
<td>0.99s</td>
</tr>
<tr>
<td>0.1181</td>
<td>0.1242</td>
<td>0.1028</td>
<td>0.0998</td>
<td>0.7687</td>
<td>0.6328</td>
<td>0.7175</td>
<td>0.7018</td>
<td>0.2777</td>
<td>0.2715</td>
<td>0.3877</td>
<td>0.3630</td>
<td>0.2912</td>
<td>0.3690</td>
<td>0.2912</td>
<td>0.2079</td>
<td>0.1788</td>
<td>0.1706</td>
</tr>
<tr>
<td>0.02s</td>
<td>45.25s</td>
<td>0.02s</td>
<td>45.25s</td>
<td>&lt;0.01s</td>
<td>0.70s</td>
<td>&lt;0.01s</td>
<td>0.70s</td>
<td>&lt;0.01s</td>
<td>0.43s</td>
<td>&lt;0.01s</td>
<td>0.21s</td>
<td>&lt;0.01s</td>
<td>0.43s</td>
<td>&lt;0.01s</td>
<td>0.21s</td>
<td>&lt;0.01s</td>
<td>0.43s</td>
</tr>
</tbody>
</table>

and MeanShift (from the Python OpenCV library [10]) on object tracking in various scenes. In practice, we found that MeanShift tends to be easily misled by the surroundings, particularly when there are areas of similar color. Finally, MeanShift++ usually finds better windows than MeanShift.

8. Conclusion

We provided MeanShift++, a simple and principled approach to speed up the MeanShift algorithm in low-dimensional settings. We applied it to clustering, image segmentation, and object tracking, and show that MeanShift++ is competitive with MeanShift in low dimensions while being as much as 10,000x faster. This dramatic speedup makes MeanShift++ practical for modern computer vision applications.
References


[25] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. 8


A. Experiments

Figure 7. Additional experiments. MeanShift++ and MeanShift performances on 10 more real-world datasets, for a total of 14 small datasets. Again, we illustrate how both algorithms were tuned over an appropriate range of bandwidth, and ARI, AMI, and runtime are reported for each run. MeanShift++ consistently performs as well or better than MeanShift in a fraction of the time, besting MeanShift in 12 out of these 14 select small datasets in ARI and AMI.
B. Proofs

Proof of Theorem 1. We first give an upper and lower bound on \( \mathcal{P}(G) \) for \( G \in \mathcal{G}_h \), where \( \mathcal{P}(G) \) denotes the total probability mass of \( G \) w.r.t. \( \mathcal{P} \). Since \( \mathcal{X} \) is bounded in \( \mathbb{R}^d \), there exists a constant \( C_X \) such that for all \( h > 0 \), we have \( |\mathcal{G}_h| \leq C_X \cdot h^{-d} \). Next, we have by the assumptions that for any cell \( G \in \mathcal{G}_h \), for some \( C_1 \) depending on \( p \):

\[
\mathcal{P}(G) \geq \min_{x \in \mathcal{X}} p(x) \cdot \text{Vol}(G \cap \mathcal{X}) \\
\geq C'(\lambda_0 - C_\alpha \cdot h^\alpha) \cdot h^d \geq \frac{1}{2} C' \lambda_0 \cdot h^d.
\]

Next, since \( \mathcal{X} \) is compact, there exists \( p_{\max} \) such that \( \sup_{x \in \mathcal{X}} p(x) = p_{\max} < \infty \). Thus, \( \mathcal{P}(G) \leq p_{\max} \cdot h^d \).

We now bound \( \sup_{x \in \mathcal{X}} \left| \frac{\mathcal{P}(x)}{h^d} - \hat{p}_h(x) \right| \). We have that the event a sample drawn according to \( x \) lies in \( G \) is a Bernoulli random variable of probability \( \mathcal{P}(G) \). From the above, we have that this variance is upper and lower bounded by \( O(h^d) \), and let us denote the number of samples in \( \mathcal{X}_{[n]} \) that lie in \( G \) as \( \mathcal{P}_n(G) \). Therefore, by Hoeffding-Chernoff inequality (i.e. Theorem 1.3 of [60]), we have for some \( C'' \) depending on \( p \) that

\[
\mathbb{P} \left( \left| \frac{\mathcal{P}(G(x))}{h^d} - \mathcal{P}_n(G(x)) \right| \geq \frac{t}{n} \right) \leq \exp \left( - \frac{t^2}{4 C'' \cdot n \cdot h^d} \right),
\]

for \( n \cdot h^d \) sufficiently large depending on \( p \). Now choosing \( t = 2\sqrt{n \cdot h^d \cdot \sqrt{C'' \cdot \log(C_X \cdot h^{-d}/\delta)}} \), we have

\[
\mathbb{P} \left( \left| \frac{\mathcal{P}(G(x))}{h^d} - \hat{p}_h(x) \right| \geq \frac{2\sqrt{C'' \cdot \log(C_X \cdot h^{-d}/\delta)}}{\sqrt{n} \cdot h^d} \right) \leq \frac{\delta}{|\mathcal{G}_h|}.
\]

Thus, by union bound, we have the following holds:

\[
\mathbb{P} \left( \sup_{x \in \mathcal{X}} \left| \frac{\mathcal{P}(G(x))}{h^d} - \hat{p}_h(x) \right| \geq \frac{2\sqrt{C'' \cdot \log(C_X \cdot h^{-d}/\delta)}}{\sqrt{n} \cdot h^d} \right) \leq \delta.
\]

Next, we have by the smoothness assumption that

\[
\sup_{x \in \mathcal{X}} \left| p(x) - \frac{\mathcal{P}(G(x))}{h^d} \right| \leq C_\alpha h^\alpha.
\]

The result follows by triangle inequality. \( \square \)