# MMD-PnP5

Tobias Winter

May 2025

## 1 MMD-PnP5

## Task 1: Sequential k-means (8 points)

In this question you will consider an algorithm for sequential k-means clustering (see Algorithm 1). We would like to perform an assignment of examples $x_1, x_2, \ldots$ that are streamed (i.e., not all known a priori) into $k$ clusters. One way to derive an algorithm is to perform online gradient descent on the k-means quantization error.

We define the loss function $L(x, \mu)$ where $\mu$ is the vector of means $\mu_1, \ldots, \mu_k$ for each label, and

$$L(x, \mu) = \min_i \|x - \mu_i\|_2^2.$$

Once each example arrives, the algorithm calculates the gradient $\partial L/\partial \mu$ and takes a step in the negative gradient direction:

$$\frac{\partial L(x)}{\partial \mu_i} = \begin{cases} 2(\mu_i - x) & \text{if } i = \arg\min_j \|x - \mu_j\|_2 \\ 0 & \text{otherwise} \end{cases}$$

Thus, for each new example $x_t$, we first identify the currently closest cluster $i$ with mean $\mu_i$, and update it such that $\mu_i$ takes a step towards $x_t$.

(a) (3 points) Prove that with a fixed step size $a \in (0, 1)$, the updated mean after $n$ points $x'_1, \ldots, x'_n$ assigned to cluster $i$ is

$$\mu_i^{(n)} = (1-a)^n \mu_i^{(0)} + a \sum_{k=1}^{n} (1-a)^{n-k} x'_k$$

Answer:
Proof by induction.
From algorithm 1 we know the update rule

$$\begin{aligned} \mu_i^{(n+1)} &= \mu_i^{(n)} + a(x'_{n+1} - \mu_i^{(n)}) \\ &= \mu_i^{(n)} + a x'_{n+1} - a\mu_i^{(n)} \\ &= (1-a)\mu_i^{(n)} + a x'_{n+1} \end{aligned}$$

Base case: n=1

$$\begin{aligned} \mu_i^{(1)} &= (1-a)\mu_i^{(1-1)} + a x_1 \\ &= (1-a)\mu_i^{(0)} + a x_1 \end{aligned}$$

This has already the form from the formula above since if we also plug in n=1 in there we get

$$\begin{aligned}
\mu_i^{(1)} &= (1-a)\mu_i^{(0)} + a\sum_{k=1}^{1}(1-a)^{n-k}x'_k \\
&= (1-a)\mu_i^{(0)} + a(1-a)^{1-1}x'_1 \\
&= (1-a)\mu_i^{(0)} + a(1-a)^{1-1}x'_1 \\
&= (1-a)\mu_i^{(0)} + a\cdot 1\cdot x'_1 \\
&= (1-a)\mu_i^{(0)} + ax'_1
\end{aligned}$$

Induction hypothesis: assuming true for n

$$\mu_i^{(n)} = (1-a)^n\mu_i^{(0)} + a\sum_{k=1}^{n}(1-a)^{n-k}x'_k$$

Induction step: proving for n + 1

Starting from formula from above which was

$$\mu_i^{(n)} = (1-a)^n\mu_i^{(n-1)} + ax_n$$

plugin in n+1

$$\mu_i^{(n+1)} = (1-a)^{n+1}\mu_i^{(n)} + ax'_{n+1}$$

Plugin in IH

$$\begin{aligned}
\mu_i^{(n+1)} &= (1-a)\left[(1-a)^n\mu_i^{(0)} + a\sum_{k=1}^{n}(1-a)^{n-k}x'_k\right] + ax'_{n+1} \\
&= (1-a)^{n+1}\mu_i^{(0)} + a(1-a)\sum_{k=1}^{n}(1-a)^{n-k}x'_k + ax'_{n+1}
\end{aligned}$$

bring in the $(1-a)$ into the sum

$$\mu_i^{(n+1)} = (1-a)^{n+1}\mu_i^{(0)} + a\sum_{k=1}^{n}(1-a)^{n+1-k}x'_k + ax'_{n+1}$$

bringing in the last summand into the sum since $ax'_{n+1}$ can be brought into the sum

$$\mu_i^{(n+1)} = (1-a)^{n+1}\mu_i^{(0)} + a\sum_{k=1}^{n+1}(1-a)^{n+1-k}x'_k$$

That is our IH. Therefore the formula holds for all n $\geq$ 1

(b) (3 points) Propose an alternative update rule such that $\mu_i^{(n)} = \frac{1}{n}\sum_{k=1}^{n}x'_k$. Hint: You might need to change the initialization.

Answer:
The new update should be the mean of the new points. We can also use the previous mean to get a recursive from. That is

$$new\ mean = old\ mean + \frac{1}{n}(x_n - old\ mean)$$

$$\mu_i^{(n)} = \mu_i^{(n-1)} + \frac{1}{n}\left(x'_n - \mu_i^{(n-1)}\right)$$

Proof that this produces the mean of the first n points:

Base case $n = 1$

$$\mu^{(1)} = x_1 = \frac{1}{1}\sum_{k=1}^{1} x_k = x_1$$

Induction Hypotheses - true for some $n - 1 \geq 1$:

$$\mu^{(n-1)} = \frac{1}{n-1}\sum_{k=1}^{n-1} x_k$$

Induction Step:

$$\begin{aligned}\mu^{(n)} &= \mu^{(n-1)} + \frac{1}{n}\left(x_n - \mu^{(n-1)}\right) \\ &= \left(1 - \frac{1}{n}\right)\mu^{(n-1)} + \frac{1}{n}x_n \\ &= \frac{1}{n-1}\mu^{(n-1)} + \frac{1}{n}x_n\end{aligned}$$

Substitute IH:

$$\begin{aligned}\mu^{(n)} &= \frac{1}{n-1}\left(\frac{1}{n-1}\sum_{k=1}^{n-1} x_k\right) + \frac{1}{n} + x_n \\ &= \frac{1}{n}\sum_{k=1}^{n-1} x_k + \frac{1}{n}x_n \\ &= \frac{1}{n}\sum_{k=1}^{n} x_k\end{aligned}$$

Therefor the update rule

$$new\ mean = old\ mean + \frac{1}{n}(x_n - old\ mean)$$

is the same as recomputing the whole mean for up to k points.

(c) (2 points) Which of the two update rules would you prefer in which situation?

Answer:

The update rule from a) is better if arriving points change over time and the weights of older points should be decreased.
The update rule from b) is better if all arriving points have and will have the same weights over time - so the influance does not change.

**Algorithm 1** Sequential k-means

1: Initialize $\mu$ with initial guess $\mu_i^{(0)}$ for each $\mu_i$
2: **for** $t = 1, 2, \ldots$ **do**
3: $\quad i \leftarrow \arg\min_j \|x_t - \mu_j\|_2^2$
4: $\quad \mu_i \leftarrow \mu_i + a(x_t - \mu_i)$
5: **end for**
6: **return** $\mu$

## Task 2: Actively Learning a Union of Intervals (8 points)

You are given a pool $X = \{x_1, \ldots, x_n\}$ of $n$ unlabeled examples where $x_i \in [0, 1]$. There exist unknown constants $0 \leq a < b < c < d \leq 1$ such that all $x_i \in [a, b] \cup [c, d]$ have label 1 and others label -1. The goal is to develop a pool-based active learning scheme to infer the labels.

(a) (2 points) Show that in general $n$ labels are needed to infer all labels.

Answer:

Intuition: Lets assume that the points $x_1, ..., x_n$ are sorted such that $x_1 < x_2 < ... < x_n$. Lets now pick two points $x_i$ and $x_j$ with $i \neq j$

We can choose $[a, b]$ and $[c, d]$ to be arbitrary small or large with

(a) $[a, b] := [x_i - \epsilon, x_i + \epsilon]$

(b) $[c, d] := [x_j - \epsilon, x_j + \epsilon]$

All other $x_k$ not in $[a, b]$ and $[c, d]$ will be labeled with -1 otherwise with 1.

Proof:
Let $x_i$ and $x_j$ two points from X with $i \neq j$. The two intervals $[a, b]$ and $[c, d]$ can now be choose around $x_i$ and $x_j$ with a $\epsilon$ small enough that no other $x_k$ fits in the intervals. Meaning all $x_k \notin [a, b] \cup [c, d]$. If the labels of $x_i$ and $x_j$ are not checked (they are skiped) then only the label -1 exists. Therefor if we do not check all $x_n$ we can never know the correct labeling.

(b) (3 points) Define $E(x, x') = |X \cap [x, x']|$. Assume $E(a, b) \geq m$, $E(b, c) \geq m$, and $E(c, d) \geq m$ for some known $m \geq 1$. Design an active learning scheme using $m$. How many samples are needed?

see Algorithm "active_learning_scheme_using_m"

**Algorithm 2** active_learning_scheme_using_m (corrected)

```
Input: sorted pool X = (x_1, ..., x_n), integer m ≥ 1
step = m
i = 0
prev = label_of(x_1)
while prev = −1 do
  i = min(i + step, n-1)
  current = label_of(x_{i+1})
  if current == +1 then                                   ▷ bounder crossed
    a_index = binary_search(X, i-step, i, "up")
    break
  end if
  prev = current
end while                          ▷ fist boarder found - now find transition from a to b
i = a_index + step
while label_of(x_{i+1} == +1 do                           ▷ we are now in +1 section
  i = min(i + step, n-1)
end while                                                 ▷ crossed to -1 somewhere
b_index = binary_search(X, i-step, i, "down")
i = b_index + step
while label_of(x_{i+1} == -1 do      ▷ We are now in a -1 section and search for the next +1
  i = min(i + step, n-1)
end while
c_index = binary_search(X, i - step, i, "up")
i = c_index + step
while query(x_{i+1} == +1 do
  i = min(i + step, n-1)
end while
d_index = binary_search(X, i-step, i, "down")
for j = 0 to n-1 do
  if a_index ≤ j ≤ b_index or c_index ≤ j ≤ d_index then
    label[j] = +1
  else
    label[j] = -1
  end if
end for
return label
```

This algorithm needs $\mathcal{O}(\log m)$ function calls to *label_of* per interval boarder and not n.

(c) (3 points) Devise a strategy without knowing $m$ that still uses approximately the same number of labels (in expectation). Randomization allowed.

Answer:

We can call label_of() at random until we found two points with label 1. For that we expect $\mathcal{O}(\frac{n}{m})$ calls, where n is the length of X and m is the unknown interval length, until we found a positive label. If we found a positive label, we then perform binary search to find the boundaries of the interval - this then takes $\mathcal{O}(\log m)$ calls. So this takes $\mathcal{O}\left(max\left(\frac{n}{m}, \log m\right)\right)$ calls.

# Task 3: UCB (8 points)

We analyze the regret of the UCB1 algorithm over $T$ rounds. Assume $k$ arms with rewards in $[0, 1]$ and mean values $\mu_1, \ldots, \mu_k$ with optimal mean $\mu^* = \max_i \mu_i$. Let $\Delta_i = \mu^* - \mu_i$ be the suboptimality gap.

**Algorithm 3** UCB1 Policy for $k$-armed bandits with fixed $T$

1: Initialize $\hat{\mu}_i^0 = 0$, $n_i^0 = 0$ for all $i$
2: Play each arm once to initialize $\hat{\mu}_i^t$ and $n_i^t$
3: **for** $t = k + 1$ to $T$ **do**
4: $\quad j \leftarrow \arg\max_i \hat{\mu}_i^t + \sqrt{\frac{\log T}{n_i^t}}$
5: $\quad$ Update: $n_j^{t+1} \leftarrow n_j^t + 1$, $\hat{\mu}_j^{t+1} \leftarrow \hat{\mu}_j^t + \frac{y_t - \hat{\mu}_j^t}{n_j^t + 1}$
6: **end for**

(a) (1 point) Show $E[R_T] = \sum_i E[n_i^T]\Delta_i$.

Answer:
We know the defenition for the regret is

$$R_T = \sum_{t=1}^{T} \mu^* - \mu_i$$

We now can define a indicator function $I_{t,i} = 1\{i_t = i\}$ to indicate that amr $i$ was drawn in round $t$.

$$\begin{aligned} R_T &= \sum_{t=1}^{T} \sum_{i=1}^{k} I_{t,i}(\mu^* - \mu_i) \\ &= \sum_{t=1}^{T} (\mu^* - \mu_i) \sum_{i=1}^{k} I_{t,i} \\ &= \sum_{i=1}^{k} \Delta_i n_T^i \end{aligned}$$

where $n_T^i = \sum_{i=1}^{T} I_{t,i}$. We now can aply the expecation.

$$\mathbb{E}[R_T] = \mathbb{E}\left[\sum_{i=1}^{k} \Delta_i n_T^i\right]$$

Since $\Delta_i$ are constants we can draw in the expectation into the sum to get

$$\mathbb{E}[R_T] = \sum_{i=1}^{k} \Delta_i \mathbb{E}[n_T^i]$$

(b) (2 points) Define confidence set $C_i^t = \{\mu : |\mu - \hat{\mu}_i^t| \leq \sqrt{\frac{\log T}{n_i^t}}\}$. Use Hoeffding's inequality to show:

$$\mathbb{P}(\mu_i \notin C_i^t) \leq \frac{2}{T^2}$$

Hoeffding's inequality

$$P(|\hat{\mu} - \mu| \geq b) \leq 2\exp(-2b^2 m)$$

We now want to show that this:

$$Pr\left(|\hat{\mu}_i^t - \mu_i| > \sqrt{\frac{\log T}{n_i^t}}\right) \leq \frac{2}{T^2}$$

Proof:

We know for $X_1, ..., X_n \in [0,1]$ iid. random variables with a expected value of $\mu = \mathbb{E}[X_j]$ and a empirical mean

$$\hat{\mu}_n := \frac{1}{n}\sum_{j=1}^{n} X_j$$

that the Hoeffdings inequality holds. In our case the empirical mean is $\hat{\mu}_i^t$ and the true mean is $\mu_i$ Lets now set our upper bound in for b in the inequality to get the follwoing:

$$\begin{aligned} Pr\left(|\hat{\mu}_i^t - \mu_i| \geq \sqrt{\frac{\log T}{n_i^t}}\right) &\leq 2\exp\left(-2n_i^t \frac{\log T}{n_i^t}\right) \\ &= 2\exp\left(-2\log T\right) \\ &= 2T^{-2} \\ &= \frac{2}{T^2} \\ Pr\left(|\hat{\mu}_i^t - \mu_i| \geq \sqrt{\frac{\log T}{n_i^t}}\right) &\leq \frac{2}{T^2} \end{aligned}$$

So $\mu_i$ lays in the in $\left[\hat{\mu}_i^t - \sqrt{\frac{\log T}{n_i^t}}, \hat{\mu}_i^t + \sqrt{\frac{\log T}{n_i^t}}\right]$

(c) (2 points) Show that if $\mu_i \in C_i^t$ and $\mu^* \in C_{i^*}^t$ for all $t$, then:

$$n_i^T \leq \frac{4\log T}{\Delta_i^2} + 1$$

Answer:

Suppose in round $t$ we choose a suboptimal arm $i$, then

$$\hat{\mu}_i^t + \sqrt{\frac{\log T}{n_i^t}} \geq \hat{\mu}_{i^*}^t + \sqrt{\frac{\log T}{n_{i^*}^t}}$$

if $\mu_i \in C_i^t \Rightarrow |\hat{\mu}_i^t - \mu_i| \leq \sqrt{\frac{\log T}{n_i^t}} \Rightarrow \hat{\mu}_i^t \leq \mu_i + \sqrt{\frac{\log T}{n_i^t}}$

For $\mu^*$ the same. If $\mu^* \in C_{i*}^t \Rightarrow \hat{\mu}_{i*}^t \geq \mu^* - \sqrt{\frac{\log T}{n_{i*}^t}}$

If we plug this two inequality into the first one we get:

$$\begin{aligned}
\mu_i + \sqrt{\frac{\log T}{n_i^t}} + \sqrt{\frac{\log T}{n_i^t}} \geq \mu^* - \sqrt{\frac{\log T}{n_{i^*}^t}} + \sqrt{\frac{\log T}{n_{i^*}^t}} & = \\
\mu_i + 2\sqrt{\frac{\log T}{n_i^t}} \geq \mu^* & = \\
\mu^* - \mu_i \leq 2\sqrt{\frac{\log T}{n_i^t}} & = \\
\Delta_i \leq 2\sqrt{\frac{\log T}{n_i^t}} & = \\
\Delta_i^2 \leq 4\frac{\log T}{n_i^t} & = \\
n_i^t \leq \frac{4\log T}{\Delta_i^2}
\end{aligned}$$

We just shown that if we draw arm i and our confidence intervals are correct then we only ever draw $n_i^t$ arms with

$$n_i^t \leq \frac{4\log T}{\Delta_i^2}$$

This means that $n_i^t$ is our last valid draw therefore

$$n_i^T \leq \frac{4\log T}{\Delta_i^2} + 1$$

(d) (2 points) Use the probabilistic bounds above to bound the expected number of times $\mathbb{E}[n_i^T]$ a suboptimal arm i played, and put everything together to obtain the desired regret bound.

Answer:
We know that

$$\mathbb{E}[n_i^T] \leq \frac{4\log T}{\Delta_i^2} + 1$$

and we want to show:

$$\begin{aligned}
\mathbb{E}[R_T] \leq \sum_{i:\mu_i<\mu^*} \mathbb{E}[n_i^T]\Delta_i & = \\
\leq \left(\frac{4\log T}{\Delta_i^2} + 1\right)\Delta_i & = \\
\mathbb{E}[R_T] \leq \left(\frac{4\log T}{\Delta_i} + \Delta_i\right)
\end{aligned}$$

from a) we got

$$R_T = \sum_{t=1}^{T}(\mu^* - \mu_i) = \sum_{i=1}^{k} n_i^T \Delta_i$$

with $\Delta_i = \mu^* - \mu_i$ and $n_i^T = \sum_{t=1}^{T} 1_{\{i_i=i\}}$ Only $\Delta_i > 0$ causes a regret so:

$$R_T = \sum_{i:\Delta_i>0} n_i^T \Delta_i$$

If we now use the expectation we get

$$\mathbb{E}[R_T] = \sum_{i:\Delta_i>0} \mathbb{E}[n_i^T]\Delta_i$$

We now can use the found bound from c)

$$\mathbb{E}[n_i^T] \leq \frac{4\log T}{\Delta_i^2} + 1$$

if we plug this into the equation we get the following inequality

$$\begin{aligned}
\mathbb{E}[R_T] &\leq \sum_{i:\Delta:i>0} \left(\frac{4\log T}{\Delta_i^2} + 1\right)\Delta_i &= \\
\mathbb{E}[R_T] &\leq \sum_{i:\Delta:i>0} \left(\frac{4\log T}{\Delta_i} + \Delta_i\right) &= \\
\mathbb{E}[R_T] &\leq \left(\frac{4\log T}{\Delta_i} + \Delta_i\right)
\end{aligned}$$

(e) (1 point) Show worst-case regret bound: $E[R_T] = \mathcal{O}(\sqrt{kT\log T})$.

We assume that $\mathbb{E}[n_i^T] \in O\left(\frac{\log T}{\Delta_i^2}\right)$

From d) we know the regret bound

$$\mathbb{E}[R_T] = \sum_{i=1}^{k} \Delta_i \mathbb{E}[n_i^T]$$

We plug in for $\mathbb{E}[n_i^T]$ and get again

$$\mathbb{E}[R_T] = \sum_{i=1}^{k} \Delta_i \left(\frac{4\log T}{\Delta_i^2} + 1\right) = \sum_{i=1}^{k} \left(\frac{4\log T}{\Delta_i} + \Delta_i\right)$$

Lets set $\Delta_i$ to the worst possible value for all i $\Delta_i = \Delta$ to get

$$\sum_{i=1}^{k} \left(\frac{4\log T}{\Delta} + \Delta\right) = k \cdot \left(\frac{4\log T}{\Delta} + \Delta\right)$$

As worst case we can set $\Delta = \sqrt{\frac{4\log T}{T}}$. Then for the fraction in the sum we get

$$\frac{4\log T}{\Delta} = 4\log T \cdot \sqrt{\frac{T}{4\log 4}} = \sqrt{4T\log T}$$

If we plug this in

$$\mathbb{E}[R_T] \leq k\left(\sqrt{4T\log T} + \sqrt{\frac{4\log T}{T}}\right) = O\left(\sqrt{kT\log T}\right)$$

$$\mathbb{E}[R_T] = O\left(\sqrt{kT\log T}\right)$$

# Task 4: Clustering Streaming Data (6 points)

(a) (3 points) Read Sections 4.1 and 4.6 of the MMDS book. Explain the streaming model and approach for counting ones in data streams.

Explaination:

Data arrives in one or multiple streams. We can only keep track of a window of size N or a window of some time. If the data in the window is not processes immediately, then the data is lost, since the streaming data is to large to keep in main memory. There are two ways one can queries a database with streaming data.

(a) Standing queries:
In this type of queries we ask questions that only requiere the most recent input. For example, if we want to know the average from the stream we only need to keep track of the most recent entire and the number of the overall entries from the stream. This is the same for the maximum. Here we only need the current entry and the maximum until now, which will be checked against the current new entry.

(b) Ad-hoc queries:
Since we can not ask any type of query we need to know in advance what type of queries will be asked, so we can prepare and keep track of some summaries to answer the ad-hoc queries. For this we can keep a window of the stream in main memory. This window can be fixed in size or fixed in time. Then we can ask queries over the current window. For example the sliding window average. Or the maximum in the current window.

Counting numbers problem:

Lets say we have a window of N and we want to know how many 1s we saw in a bit stream. Doing that requires us to go through the whole window, so we need $O(N)$ space and time to count all 1s, if we do it the classic way. In the following we will go through a algorithm that only that only needs $O(\log^2 N)$ space. By doing that we sacrifice precision and only give back a estimate but gain sub linear space usage.

The DGIM (Datar-Gionis-indyk-Motwani) Algorithm:

In this algorithm we use buckets. Each bucket has the timestamp of its most recent entrie (to most right one) and the number of 1s in it (which will be its size). For this algorithm to work the following preconditions need the be present:

(a) the right end of a bucket is always a postition with a 1

(b) every postition with a 1 is in some bucket

(c) No position is in more than one bucket

(d) There are one or two buckets of any given size, up to some maximum size

(e) all sizes must be a power of 2

(f) Buckets cannot decrease in size as we move to the left

There are $O(\log N)$ bucket.

As we follow the stream we keep track on what the incomming bits are. If we see a 0, then we do nothing but if we see a 1, then we create a new bucket with a size of 1 and the current timestamp. Since no more then two buckets of the same size are allowed to exists, if we see 3 buckts with the same size, we merge the two oldest buckets to one (this causes a ripple effect). So if three bucktes of size $2^r$

exists, we start the merging process and we get a new bucket of size $2^{r+1}$. If a bucket lays complitly outside the window of size N then we delete it.

To now estimate the number of 1s in the window, we add up all the sizes of all buckets + half of the size of the oldest bucket.

$$\sum_{j=1}^{m-1} size(Bucket_j) + \frac{1}{2} \cdot size(Bucket_m)$$

We only count half of the last bucket sinz half of it might be outside the window. This gives us an error of max 50 %.

We can achieve arability small error if we accept a large constant factor. We can do this by allowing $r$ buckets of the same size with $r > 2$. In the worst case we get an error of $\frac{1}{r-1}$. The algorithm stays the same.

(b) (3 points) Read Section 7.6 of the MMDS book. Explain the BDMO algorithm and how parallelism aids clustering.

The BDMO algorithm:

This is a variant of k-means clustering but instead of keeping track of all points we get a stream of data points and we keep track of statistics of the data points.

We keep track of:

(a) N - number of points
(b) SUM - sum of the coordinates where the i'th position is the sum of the coordinates in that dimension
(c) SUMSQ - here is the i'th component the sum of squers of the coordinates in the i'th dimension
(d) Discard set (DS) - Points close enough to a centriod to be summarized
(e) Compression set (CS) - groups of points that are close together but not to a centriod
(f) Retained set (RS) - outlayers that do not belong to DS or CS

(a) We start by reading points from the stream in memory and we run classic k-means on the read in data and initialize the variables from above
(b) After initialization we read in the next chunk of data
(c) For each point in the chunk we calculate the Mahalanobis distance to DS
(d) if the distance is small enough then we assign the point to DS
(e) if not we then calculate if the Mahalanobis distance from the point to CS and assign if it is small enough
(f) if by now the point was not assigned to DS or CS we assign it to RS since its a outlayer
(g) every time a point is added to any set, we update N, SUM, SUMSQ
(h) Now we need to check if we need to compress RS. That is when $|RS| > 2 \cdot d$ where $d$ is the dimention of features.
(i) then we check if we need to merge two close CS clusters. We do that by again calculating the Mahalanobis distance between two CS clusters $(c_1, c_2)$. If they are close enough, we merge them.
(j) now we check if any centriod from CS can be merged into DS. We do that again by calculating the distance to DS and merging the centriod if its close enough
(k) that was the main loop, if no more data is read from the stream we start finalizing

(l) in this step we again check the centriods in CS and merge its stats into DS nearest centriods. Meaning if $c_i$ is a centriod from CS and $ds_j$ is a centriod from DS, we check the nearest neighbor of any $c_i$ to any $ds_j$ and merge the stats from $c_i$ into $ds_j$

(m) we do the same for every point in RS

(n) and DS can now be returned

Now we can start with parallelizing it with the MapReduce framework by running the BDMO algorithm for every chunk and grouping it by DS. Reduce now only needs to merge each DS by merging the nearest points from each Mapper.

NOTE: We need to use the Mahalanobis distance since the euclidian distance treats all dimensions the same, with no regard to the variance in the respective dimension. The Mahalanobis takes the variance in each dimension and the correlation between dimensions into account.