Review of clustering algorithms subjected to sparse and multidimensional data sets

David Andersson

December 7, 2016

Contents

1 Introduction **3**

2 K-means **4**
2.1 Theory . . . 4
2.2 k-means++ . . . 5
2.3 Determining k - Bayesian Information Criteria . . . 5
2.4 Determining k - Silhouette method . . . 6
2.5 Pseudo code . . . 7

3 Hierarchical Clustering **7**
3.1 Theory . . . 8
3.2 Combinatorial SAHN clustering . . . 8
3.3 Determining k - Dendogram cutoff . . . 9
3.4 Pseudo code . . . 9

4 Evaluating the algorithms **9**
4.1 Data set 1 . . . 11
4.2 Data set 2 . . . 11
4.3 Data set 3 . . . 12

5 Discussion **12**
5.1 Detrending BIC . . . 12
5.2 Using multidimensional data . . . 12
5.3 Adding peripheral cluster . . . 13
5.4 Conditioning on previous frames . . . 13
5.5 Adding constraints to distance function . . . 13

6 Conclusion **13**

Abstract

This paper will explore the method of clustering. Two main categories of algorithms will be used, namely k-means and hierarchical clustering. I will look at algorithms within theses categories and what types of problems they solve, as well as what methods could be used to determine the number of clusters. Finally I will test the algorithms out using sparse multidimensional data acquired from the usage of a touchscreen, showing that a simple implementation can achieve non trivial results. The result will be presented in the form of an evaluation of there potential for online clustering. I will also discuss some task specific improvements and which approach is most suitable.

1 Introduction

Given a set of data with different properties, say that they have different spatial location, amplitude, color e.t.c, it might be the case that there is information about the relationship between data point not obvious on inspection. The method of clustering is a way to divide data into groups which have a high similarity to other members of the cluster, and low similarity to members in other clusters. This similarity can be expressed in the form of a distance function. However, this division is often ambiguous, and relies on the fact that the dissimilarities are distinguishable. [11, p. 201] To give an example of this relationship, lets assume that the data in question is produced by random variables $X_1, .., X_n$. In figure 1 we have an example where the variables have a uniform distribution, and if an algorithm find structure in the data it would only be a false positives. In figure 2 on the other hand, we see example of random distributions over different intervals. Here we can by inspection find two clusters, even though internally there are no significant correlation.

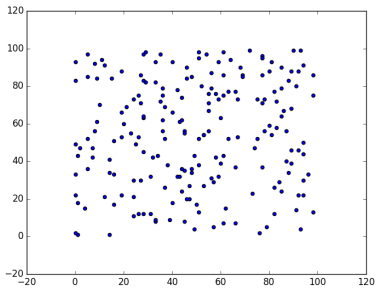

Figure 1: Dataset with random distribution.

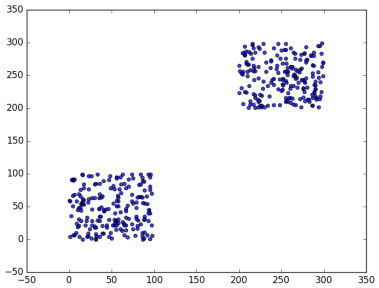

Figure 2: Dataset with random distribution over different intervals.

Finally, in figure 3 we also have an example where the data does have a significant correlation due to the fact that they are Gaussian with different values of μ.

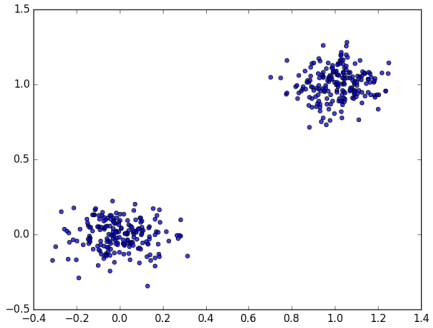

Figure 3: Dataset with Gaussian distribution using different μ.

Depending on what kind of data that is used for clustering, as well as the amount of data available, there will be a difference in performance given different methods. There is also a question of time complexity, since the k-center clustering problem is NP-hard [3] we also have to consider running time as a factor. To prove the the NP-hardness one can reduce it to the 3-SAT problem. This is omitted her, but in [14] we can see an example of a proof.

In this report two categories of such methods will be explored, k-means and hierarchical clustering. Specifically I will focus on how they can be applied to solving an online task. The task in question is determining how many hands are acting on a touch screen surface. Each data point corresponds to a finger, or digit, being registered on the surface. This generates a x and y coordinates, x and y velocity and pressure. The data also contains pre-processed tracking of individual points which will not be used as a clustering parameter.

2 K-means

When referring to k-means, many find it synonymous to Stuart P. Lloyd's least square algorithm from 1982 [13]. Since k-means is a NP hard problem, we need some way of reducing the time complexity, which Lloyd's algorithm does. The issue is however that it does not give any guaranties to how close to ϕOPT we can come. In fact, this algorithm can produce arbitrarily bad clusters [3].

The algorithm is as follows: Start by creating k empty clusters and for each cluster, choose at random a data point making the coordinates of this point the center of the cluster. Proceed by iterating over each data point and calculate which cluster it is closes to and adding the point to it's closest cluster. When this is done, recalculate the center of each cluster by taking the mean of the assigned points. Continue this process until the centers of the clusters no longer changes.

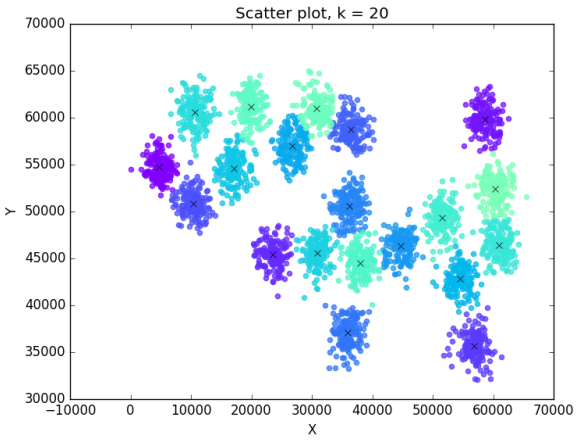

Figure 4: Sucessfull convergence using k-means. Data set from [9]

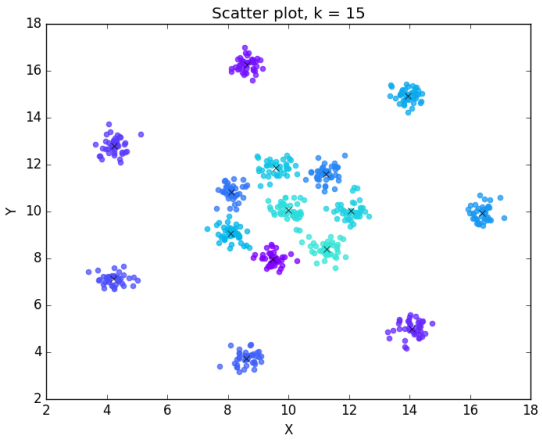

Figure 5: Sucessfull convergence using k-means. Data set from [18]

2.1 Theory

We formulate the problem as follows: Define a set P of points $p_1, p_2, ..., p_n$, $p \in (\mathbb{R}^m, L_2)$, m being the number of features or dimensionality. For each pair p_i, p_j we have a distance given the function $d_{i,j} = dist(p_i, p_j)$. The assumption is all points are in metric space such that $d_{i,j} = d_{j,i}$, $d_{i,i} = 0$, $d_{i,j} \geq 0$ and $d_{i,k} \leq d_{i,j} + d_{j,k}$.

So, denoting features $x_k, y_k,\ k = 1, .., m$ from p_i, p_j respectively, we get the following expression:

$$dist(p_i, p_j) = ||p_i - p_j|| = \sqrt{\sum_{k=1}^{m}(x_k - y_k)^2} \tag{1}$$

The objective is to include each p_i in one of k clusters $C = c_1, c_2, ..., c_k$ determined by the distance function. The center $c_{\theta,i}$ can be calculated as follows:

$$c_{\theta,i} = \frac{1}{|c_i|} \sum_{p \in c_i} p \tag{2}$$

The idea is to minimize ϕ the, total squared distance between each point and its closest center $c_{\theta,i}$ given n data points:

$$\phi = \sum_{p \in P} \min_{c_\theta \in C} \|p - c_\theta\|^2 \tag{3}$$

This minimization we can view as a cost function which is monotonically decreasing. This way, we know that the algorithm always converges. To show this, let $c_{\theta,1}^{(t)}, ..., c_{\theta,k}^{(t)}$, $C_1^{(t)}, ..., C_k^{(t)}$ denote the centers and cluster at iteration t. We start of by assigning each data point to the closest cluster, and by that:

$$cost(C_1^{(t+1)}, ..., C_k^{(t+1)}; c_{\theta,1}^{(t)}, ..., c_{\theta,k}^{(t)}) \leq cost(C_1^{(t)}, ..., C_k^{(t)}; c_{\theta,1}^{(t)}, ..., c_{\theta,k}^{(t)}) \tag{4}$$

We then proceed by recalculating the cluster center.

$$cost(C_1^{(t+1)}, ..., C_k^{(t+1)}; c_{\theta,1}^{(t+1)}, ..., c_{\theta,k}^{(t+1)}) \leq cost(C_1^{(t+1)}, ..., C_k^{(t+1)}; c_{\theta,1}^{(t)}, ..., c_{\theta,k}^{(t)}) \tag{5}$$

2.2 k-means++

As I've discussed earlier, the k-means algorithm is a ways to tackle the NP-hard nature of the problem of clustering, but it generates a new problem, namely accuracy. In [3] the k-means++ algorithm is presented as a way to get some guarantees on the accuracy of the k-means algorithm, through a randomized seeding technique. The bound that k-means++ proposes is as follows:

Theorem 1.1 *For any set of data points,* $E[\phi] \leq 8(lnk + 2)\phi OPT$

What the k-means++ adds to the original k-means is the fact that we can choose the initial clusters using a more strategic manner than simply choosing at random. The probabilities can be calculated as follows:

$$Pr(p_i) = \frac{D(p_i)^2}{\sum_{p \in P} D(p)^2} \tag{6}$$

Where D is the squared distance to the closest cluster center already in C
The paper also proves that k-mean++ is $\emptyset(log(k)$-Competitive, and that the analysis is in fact tight.

2.3 Determining k - Bayesian Information Criteria

Given a number of clusters k, applying methods like the ones I've described above will return a result. Given that the nature of the algorithm will make local improvement we decrease ϕ until we cannot distinguish a difference from the previous result. However, if we have chosen the wrong k, the convergence will be to a set of clusters not intended as an output. A way to determine the difference between the result of different outputs is using the Bayesian information criteria (BIC). As described in [17] BIC employs the posterior probability $Pr[M_i|P]$ where M_j is a model generated by convergence using a specific k, And P is the data set. This result in the following formula originally from [10]:

$$BIC(M_j) = \hat{l}_j(P) - \frac{\hat{p}_j}{2} log(R) \tag{7}$$

Where $\hat{l}_j(P)$ is the log-likelihood of the data corresponding to the model M_j and $\hat{p}_j$ being the number of parameters in the model. The version of BIC I have been using, which corresponds well to k-means++, assumes a identical spherical Gaussian distribution. This leads to the maximum likelihood estimation as follows:

$$\hat{\sigma}^2 = \frac{1}{R-K}\sum_i (p_i - \mu_{(i)})^2 \tag{8}$$

Fixing $1 \leq n \leq k$ and setting P_n to be the points belonging to centeroid n gives us a maximum likelihood estimate as follows:

$$\hat{l}_j(P_n) = -\frac{R_n}{2}log(2\pi) - \frac{R_n * M_j}{2}log(\hat{\sigma}^2) - \frac{R_n - k}{2} + R_n log(R_n) - R_n log(R) \tag{9}$$

Where R is $|P|$ and R_n and is $|p \in C_n|$.

In practice BIC preformed surprisingly well, to illustrate this lets look at how k was chosen for figure 4 and figure 5. The corresponding BIC result using different values of k can be seen in figure 6 and 7. One can clearly see a peak at value 20 in figure 6 and 15 in figure 7. This is of course an example of a successful convergence, but as we will see, the results are fairly consistent.

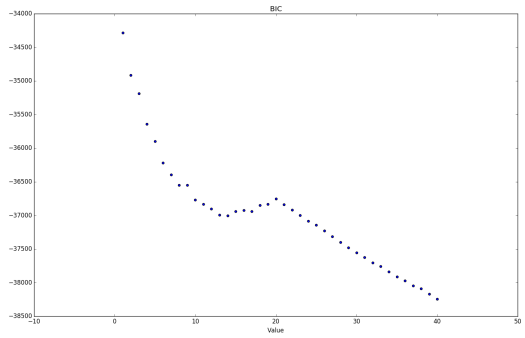

Figure 6: BIC corresponding to figure 5.

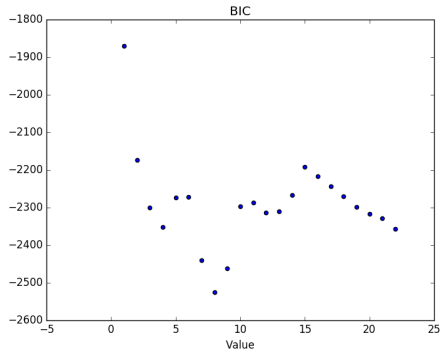

Figure 7: BIC corresponding to figure 4.

2.4 Determining k - Silhouette method

Another method for determining is the Silhouette method where each cluster is represented by a so-called silhouette. The idea is that the silhouette shows which object lie well within their cluster and which ones are skewed from their true position. To make this comprehensible, we combine the silhouettes to serve as a measurement for all clusters. This result in a plot such as in figure 8, which shows different result of cases where 4 is the true k. We can identify a point where the derivative approaches 0 in the plot, which we can interpret as the optimal result. In [15] we find the mathematical notation for this method.

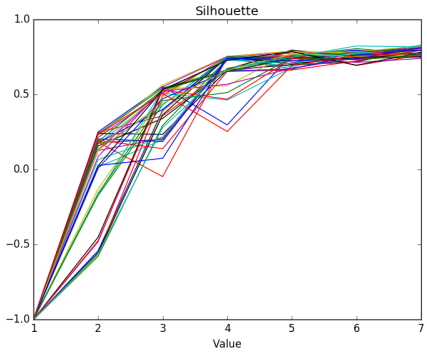

Figure 8: Iterations from data set 1.

When calculating the silhouette, we need the following:

- $a(i)$ = The average dissimilarity of p_i to all other objects of C_{p_i}.
- $d(i, C)$ =The average dissimilarity of p_i to all other objects of C.
- $b(i) = \underset{C \neq C_{p_i}}{minimum}\ d(i, C)$

The silhouettes score $s(i)$ can be calculated as follows:

$$s(i) = \begin{cases} 1 - a(i)/b(i))) & if a(i) < b(i))) \\ 0 & if a(i) = b(i) \\ b(i)/a(i) - 1)) & if a(i) > b(i) \end{cases} \tag{10}$$

As suggested in [15] I have implemented this in the following way:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{11}$$

The problem that I faced when using this method was to determine the "best" value, as one can see in plot 8 it's quite ambiguous. It ended up being a good ocular aid when trying out a new data set, comparing it to BIC.

2.5 Pseudo code

The code employed comes straight from the k-means++ theory. The idea of using BIC as a way of determining k comes from [17].

1. for k = k_{min},...,k_{max}
2. Select one data point at random to be the initial cluster.
3. Use the selection method from k-means++
4. Run until convergence:
5. For all points, check which cluster is closest
6. Update cluster center in accordance to the new data points
7. Check distance to clusters previous location, break if below a threshold
8. Calculate BIC_k
9. Return the result corresponding to the best BIC_k

3 Hierarchical Clustering

Hierarchical clustering is an approach to solving the clustering problem which might not be the fastest, but is very effective in certain cases. The version I have researched is the agglomerative hierarchical clustering (AHC) which can be described as a bottoms up approach to clustering. At initiation each data point are it's own cluster. The algorithm then iterates over the clusters determining which pair is, for example, closes (single-link) or furthest away (complete-link) from each other [5]. These clusters will then be merged and form a new cluster. If no stopping criteria is used, the algorithm will eventually have one single cluster containing all data points. For the evaluation I have used the complete-link approach since it suited the data best. To support why this is relevant to this project, n [1] we find that the complete-link AHC is an $\emptyset(log(k))$-approximation to the diameter k-clustering problem.

In figure 10 and 9 we find an example of a data sets from [7] and [8] respectively, where hierarchical clustering might preform better than others algorithm. I.e. when the data is not spherical, or there is a clear link between data points that can not be captured that easily by statistical properties. However, the

method is highly susceptible to outliers. As one can see in the figures the convergence is not perfect, and omitted are the next few iterations where the clusters become quite deformed.

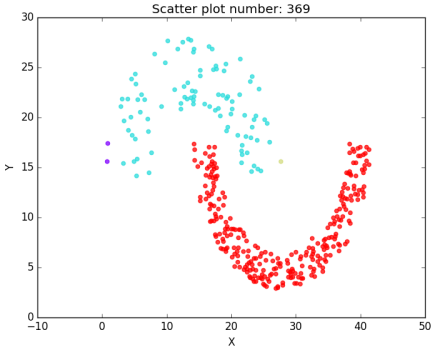

Figure 9: Convergence using single-link AHC. Data set from [8]

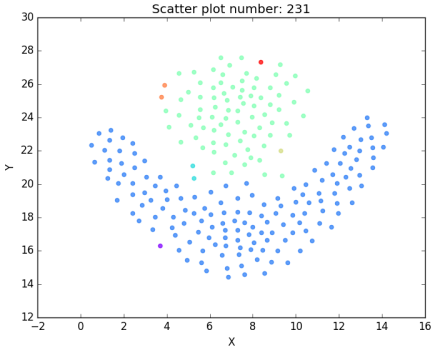

Figure 10: Convergence using single-link AHC. Data set from [7]

3.1 Theory

Lets define a set P of points $p_1, p_2, ..., p_n$, $p \in \mathbb{R}^m$, m being the dimensionality. For each pair p_i, p_j we can calculate a distance given the function $d_{i,j} = dist(p_i, p_j)$.The assumption is all points are in metric space such that $d_{i,j} = d_{j,i}$, $d_{i,i} = 0$, $d_{i,j} \geq 0$ and $d_{i,k} \leq d_{i,j} + d_{j,k}$. What we have to work with is a hollow lower triangular matrix M, which is to be search through. The objective is to include each p_i in one of k clusters $C = c_1, c_2, ..., c_k$ according to some criteria such as the ones discussed above.

When finding clusters in figure 10 and 9 I have employed the single-link clustering. To show the validity of this method we can draw a parallel to the problem of finding a minimum spanning tree. In [11, 202] we find an example of this using Kruskal's algorithm, where the analogous version of the algorithm would work as follows: Start with a strongly connected graph G where points $p_1, p_2, ..., p_n$ represent a vertice and for each point p_i, p_j we have an edge with the weight equal to $d_{i,j}$. Now, instead of merging, lets remove the heavies edge in the graph. Do this k-1 times and the result will be a minimum spanning tree. [11]

3.2 Combinatorial SAHN clustering

When doing online clustering, speed is a factor. To speed up the method I have used a combinatorial SAHN (sequential, agglomerative, hierarchical, non-overlapping) clustering method. As suggested by [5][12] we can describe the distance as follows:

$$d(h,k) = \alpha_i d(i,k) + \alpha_j d(j,k) + \beta d(i,j) + \gamma |d(i,k) - d(j,k)| \tag{12}$$

Where h is the new AHC cluster and k being a cluster already defined in our set C. The parameters α, β, γ can be set in the following way:

- Single-link: $\alpha = 0.5$, $\beta = 0$ $\gamma = -0.5$
- Complete linkage $\alpha = 0.5$, $\beta = 0$ $\gamma = 0.5$

When analyzing how a AHC clustering algorithm would preform using the description used in the introduction, one can conclude that it will be $0(n_3)$. Since n clusters will potentially be reduced to 1, we need n iterations. For each iteration we need to compare each pair i,j which in takes $\approx \frac{n(n-1)}{2}$ iterations. The combinatorial SAHN algorithm does preform a bit better $\emptyset(n^2 log(n))$ [5]. It more practical since we do not have to recalculate distances in the same sense and uses a dynamic programming technique and save the previous distances.

3.3 Determining k - Dendogram cutoff

When determining the most likely number of cluster, a common way is to look at a dendogram [4]. An example can be seen in figure 11 where one can clearly see how clusters are merged. The dendogram also shows the dissimilarity between the two clusters. A common approach is to use a cutoff value λ, but for this assignment finding the largest distance was a fairly accurate way to guess k. In some sense we are allowing minor adjustments to the clustering, but when a big adjustment happens, it is interpret as a "derailing" from the true clustering. It turns out as we will see in proceeding sections, that this is a fairly good assumption for convergence. In mathematical terms we can describe the selection process as follows:

$$\begin{aligned} \Delta_{d,i} = d_i - d_{i-1} \ i = 2, ..., k_{max} \\ k = i \in max(\Delta_{d,i}) \end{aligned} \tag{13}$$

Where d_i is the distance between the two clusters who are to be merged at iteration i.

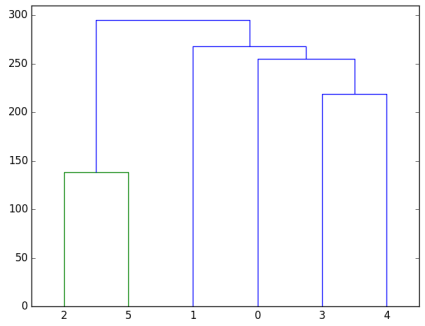

Figure 11: Example of a simple dendogram.

3.4 Pseudo code

When implementing the algorithm I have followed the SAHN method described in [5] originally from Anderberg 1973 [2] and goes as follows:

1. Create dissimilarity matrix $D_{i,j}$
2. Create a priority queue P
3. for iteration $1, ..., N-2$
4. Search for the smallest value in D using P.
5. Replace C_i, C_j with a new cluster C_h
6. Update D and P to account for changes.
7. Check dendogram for any big changes, chose k corresponding to the clustering that was present before the largest change.

4 Evaluating the algorithms

The data sets I have used for the evaluation is provided by FlatFrog Laboratories AB. There are three recordings from one of their touchscreens. The data points are the registered fingers active on the touchscreen, and the clustering task will be to assign each of these data points to a cluster representing one hand. Each finger is generating a x and y coordinate, a x and y velocity as well as pressure per frame. The Data also contains pre-processed tracking of individual points. The screen has been sampled at rate of 150 hz.

At this stage the question is focused on if there is a good way to make a correct clustering, but when formulating the problem one should note that there is a need for fast processing time due to the fact that any practical application will be online. The idea behind the three different recordings is that they should differ in how hard it is to distinguish between the cluster.

To start of, both algorithms I have used for this evaluation are in some sense very simple. The result produced in figures 10,9 and 4,5 are good because of the data used is generated by the authors and used in the right context. In the discussion I will propose extensions to the algorithms as to better suit this specific problem.

When running the algorithm I assign a weight λ to the parameters where λ_1 being spatial coordinates, λ_2 being velocity and λ_3 being pressure.

In table 1 I've presented the result of a simple test. I chose the frames containing four hands and looked at how many times the algorithm guessed right. In the table we can see a average percentage (10 iterations), what method was used, what data set was used as well as the values of λ.

Table 1: Result from cluster number guess.

% correct	Method	Data set	λ
0.84	hierarchical	1	[1, 0.1, 0]
0.541167664671	hierarchical	2	[1, 0.1, 0]
0.789473684211	hierarchical	3	[1, 0.1, 0]
0.891764705882	k-means	1	[1, 0.1, 0]
0.524695776664	k-means	2	[1, 0.1, 0]
0.645374449339	k-means	3	[1, 0.1, 0]

To analyze this data there is a need for tracking. As I have mentioned previously, there is already a pre-processed tracking of individual data points across frames. What we want is a way to check how the current frame F_i relates to the previous frame F_{i-1}. Lets define the function Cf to be a function that given a set of data points from a frame P_i returns a number of clusters. To analyze how well the algorithms tracks between frames I have used the following test: For every iteration, we converge to a number of clusters k_i. We then proceed to go through the previous and current clusters and match the best clusters to each other. I then calculate the cut between the points in the current and previous cluster and count them up, dividing by the total amount of data points. If $k_i \neq k_{i-1}$ we find the best fitting matches and ignore the rest of the clusters. The method I use for comparing clusters between frames is the mean value of the points in the clusters. I deem the distance to large to be the same cluster at some cut off threshold which is the same for both algorithms. I would say that it's quite noticeable when the clusters are not sequential since the distance is very small when two consecutive frames are converged correctly, and a factor of 10 larger when it does not. Since the sampling time is constant, there is a upper bound on how far a point can change position between frames.

A more clear explanation of the method is given as:

$$\frac{1}{|F| * |P|} \sum_{i=2}^{|F|} |Cf(P_i) \cap Cf(P_{i-1})| \tag{14}$$

The result of the test is shown in table 2, were we have an averaged percentage (10 iterations) of correct transfers between frames, the method used, what data set was used as well as the values of λ.

Table 2: Result from the data points tracking

% correct	Method	Data set	λ
0.834865227719	hierarchical	1	[1, 0.1, 0]
0.754034713764	hierarchical	2	[1, 0.1, 0]
0.768226840163	hierarchical	3	[1, 0.1, 0]
0.911985983716	k-means	1	[1, 0.1, 0]
0.712328247172	k-means	2	[1, 0.1, 0]
0.841447317338	k-means	3	[1, 0.1, 0]

A further application of the test used in table 2 is the method of coloring. In the following sections we can see the result of these coloring. The coloring algorithm works as follows: For each cluster in F_i we check if there is a matching cluster in the previous frame F_{i-1}. If there is, we use the color from the previous frame, otherwise we pick a new color. This means that every time we find $k_i \neq k_{i-1}$ or $|Cf(P_i)| \neq |Cf(P_{i-1})|$ we change color. There are however a limited amount of colors, which means that they may reoccur. The images are supposed to work as a complement to table 1 and 2. The sequence is also long enough to produce images to complex to make sense of, so I have mainly used the first frames needed to show the problem areas.

4.1 Data set 1

For data set 1, both algorithms got a fairly good score. This is not very odd since the entire data set is supposed to function as a reference of an "easy" clustering task. Looking at figure 13 and 12 we see that the majority of the time we have a uniform color from frame to frame. What we can identify from a purely ocular inspection is that the changes seem to happen when in a "curve".

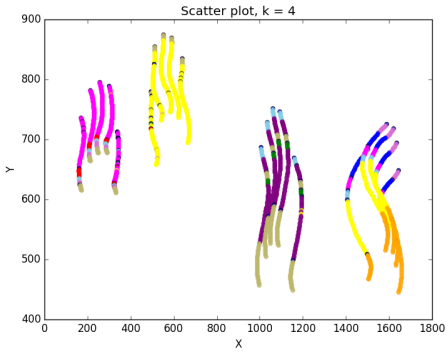

Figure 12: Example run from data set 1 using hierarchical clustering.

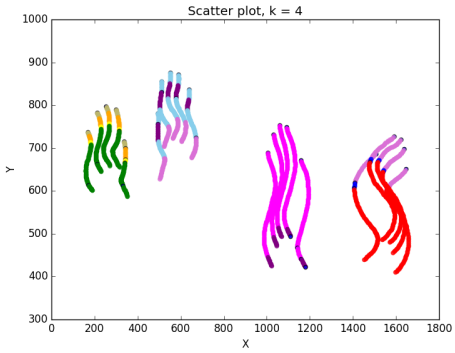

Figure 13: Example run from data set 1 using k-means clustering.

4.2 Data set 2

Data set 2 seem to be the hardest of the data sets to clusters. Both k-means and hierarchical clustering returns a score of around 50 % in table 1. Also in 2 the result is worse than for the other data sets. The conclusion that I can draw, also looking at figure 14 and 15, is that there is a large alternation between 3 and 4 clusters. A large portion of the time two hands have no clear distance, but in stead they are aligned in parallel.

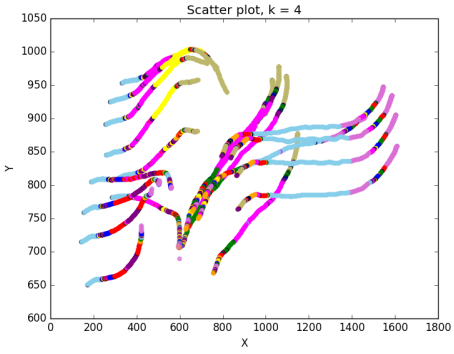

Figure 14: Example run from data set 2 using hierarchical clustering.

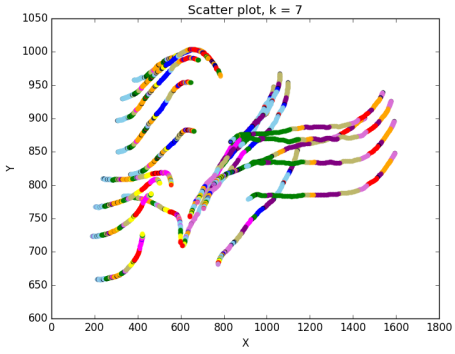

Figure 15: Example run from data set 2 using k-mean clustering.

4.3 Data set 3

In data set 3 we find a middle ground of the previous two. There are no obvious causes for not converging, except the one I discussed in "Data set 1". What differ between data set 1 and 3 are the amount of complex movement. My best bet is that it is the same phenomenon but worsened by the added complexity.

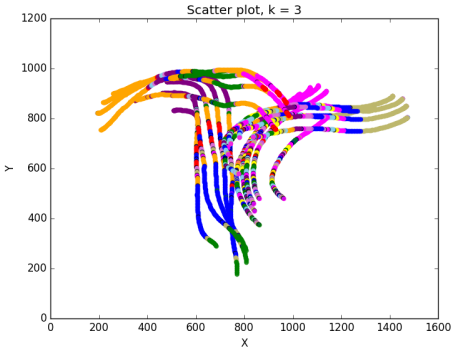

Figure 16: Example run from data set 3 using hierarchical clustering.

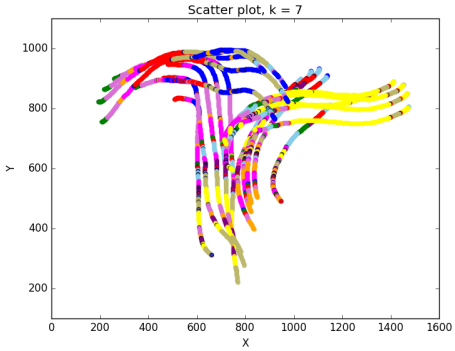

Figure 17: Example run from data set 3 using k-mean clustering.

5 Discussion

5.1 Detrending BIC

One of the main issues with the BIC plots in section "Determining k - Bayesian Information Criteria" is that it could be challenging to find the maximum, given that the function is not monotonously decreasing before we get close to our local maximum. When looking at the BIC for the touchscreen data, it's actually an even more subtle change, as can be seen in figure 18. To help this search along, making it possible to look for a global maximum of the data set, I have used linear detrending. This can be seen in figure 19. This example comes from data set 2, but a similar improvement can be seen in all data sets. Both plots are created by taking the average of all plots produced when running the algorithm.

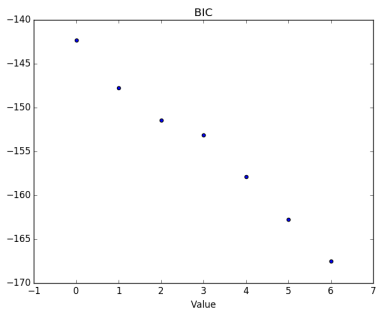

Figure 18: Average BIC result for data set 2.

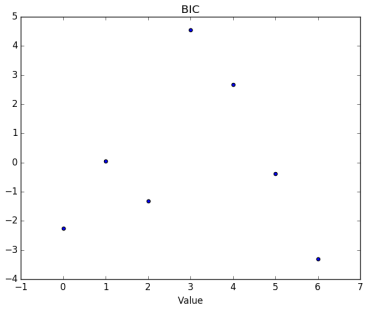

Figure 19: Average BIC result for data set 2 when removing linear trend.

5.2 Using multidimensional data

The idea behind adding dimensions is that it could help when there are ambiguities. Say that spatial coordinates would return one cluster, but in reality they have velocity with different sign, or different pressure. Adding these features could potentially help to separate the clusters.

In reality, a reasonable use of these values was: $\lambda = \{1, 0.1, 0\}$ This is of course a rough estimation, but serves as a benchmark. This was the values used to produce the table 1 and 2. As one can see, I have not

used λ_3, pressure, since it gave a worse score regardless of the values I tried. Running the algorithm using only this parameters gave no indication of the true clusters. All cases benefited from having λ_2, velocity, but in reality the score could be improved by adjusting the the value. A further exploration could add additional parameters, making it even more unambiguous.

One thing to address about the pictures in "Data set 1", "Data set 2" and "Data set 3" is that there seems to be a problem when changing direction. My initial theory was that it had to do with the velocity changing sign, but there were no noticeable effect when excluding velocity as a parameter. Further analysis could focus on this since it seem to create a big decrease in performance.

5.3 Adding peripheral cluster

One addition to k-means that I have been experimenting with is to add a dummy cluster. The method is quite a non-scientific, but could be an interesting addition to get a better score in practice. The idea is to add a cluster, for this problem I used a cluster positioned at $\{0\}$ as a reference. Since it does not fit anywhere in the clusters that are active it is easily recognized as a cluster containing only this point. What it potentially does is that it makes the total variation within the rest of the clusters relatively smaller. Imagine having only one hand, a reasonable clustering is just having all fingers as there own clusters, or maybe one for the thumb and one for the rest of the digit. Having two hands means we have a distance between hands that is larger than the distance between fingers, making it easier to identify as separate clusters. The idea would be to artificially create this distance.

5.4 Conditioning on previous frames

Continuing on the thought from "Evaluating the algorithms", one change that could be made to the distance function is a measurement of similarity to the clusters in previous frames. Seeing as though the data contains the tracking of individual fingers we could add a punishment when converging to a cluster not in the previous frame. For example, lets say that we have continuously found a some data points in one cluster, then we get closer to another cluster. At some point we could face a situation where the distance between data points are very similar through both clusters such that $var(C_i) \approx var(C_j) \approx var(C_i + C_j)$. At this point we might also have the same velocity and it would be reasonable that the convergence would be to find one cluster. However, we have this information from previous frames saying that data points belong to different clusters, making it even more probable that they should be separated.

What I do now is to cluster using a cluster function Cf with P_i as input, but in reality we should input $P_i, Cf(P_{i-1}), .., Cf(P_1)$ since there is a high correlation between different frames. This is to much data to process, but one could extend the method I have been using to coloring and limit the input to the previous frame. In some sense we could employ a Markov assumption as follows:

$$Cf(P_i|P_{i-1}, ..., P_1) \approx Cf(P_i|P_{i-1}) \tag{15}$$

To support this lets compare table 1 and 2. We can conclude from table 2 that the clustering is fairly consistent. However, if we are constantly converging to the wrong clusters, it's not really a good measure. Looking at table 1 however, speaks to the fact that we actually do find the right clustering number quite often.

5.5 Adding constraints to distance function

One somewhat non algorithmic addition one could make to the distance function is to add the physical constraint of the assignment. For example, a cluster in this setting is a hand, which we can assume has no more than six digits. Also, spacing between individual digits has a limit. Imagine the distance between your first and fifth digit, one could add a max that is around some 99 percentile of distances found in humans.

6 Conclusion

The first thing we can conclude is that none of the algorithms produced a result that was sufficient for the task. That is, there would be little use of a clustering with as low as 50% success rate in any real life scenario.

What we can see is however that these rather primitive algorithms do produce a non trivial result. Even this low score does indicate that there is potential, and maybe given a non ambiguous initiation something like conditioning on previous frames, adding peripheral clusters, or extending the dimensionality might improve the score as to make if relevant.

The real time or online use of the algorithm is also something that needs to be explored more. k-means is faster than hierarchical clustering. But there is a chance that since the maximum number of clusters are so low, both might be relevant. Since tracking for individual digit is already in place, the clustering might not need to be done at every frame, but at a given interval or when changes happen.

To comment on the scores in table 1 and 2, I was expecting an even better performance from the hierarchical clustering. As to my knowledge hierarchical clustering should preform better on these types of data sets. I might however have underestimated the spherical nature of data. One explanation for the lack of performance is I did not find a way of determining k that could rival BIC. That is, the clustering might be better if k was given.

If one would to venture outside of algorithms who are easy to implement, more modern algorithm are available. One example would be BDSCAN, which is a density based algorithm who can process clusters of arbitrary shaped, does not need a supplied k and is deterministic [6]. Another is the hierarchical method CLARANS which is based in randomized search [16]. Both of these are focused on large data sets, but are also more sophisticated. Another noticeable feature is that both are also faster than k-means++.

Another algorithm which I have considered is the x-means algorithm [17] which works by starting with k_{min} clusters and alternating between converging using k-means and then splitting clusters into two or more. Each iteration they use BIC to evaluate k. This way there is no need to redo the whole algorithm when trying out different k. What made this unsuitable for this assignment was that I wanted an ρ-approximation, which I could not find.

References

[1] Marcel R Ackermann, Johannes Blömer, Daniel Kuntze, and Christian Sohler. Analysis of agglomerative clustering. *Algorithmica*, 69(1):184–215, 2014.

[2] Michael R Anderberg. Cluster analysis for applications. monographs and textbooks on probability and mathematical statistics, 1973.

[3] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[4] Ian Davidson and SS Ravi. Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 59–70. Springer, 2005.

[5] William HE Day and Herbert Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of classification*, 1(1):7–24, 1984.

[6] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[7] Limin Fu and Enzo Medico. Flame, a novel fuzzy clustering method for the analysis of dna microarray data. *BMC bioinformatics*, 8(1):1, 2007.

[8] Anil K Jain and HC Martin. Law, data clustering: a user's dilemma. In *Proceedings of the First international conference on Pattern Recognition and Machine Intelligence*, 2005.

[9] Ismo Kärkkäinen and Pasi Fränti. *Dynamic local search algorithm for the clustering problem.* University of Joensuu, 2002.

[10] Robert E Kass and Larry Wasserman. A reference bayesian test for nested hypotheses and its relationship to the schwarz criterion. *Journal of the american statistical association*, 90(431):928–934, 1995.

[11] Jon Kleinberg and Eva Tardos. *Algorithm design.* Pearson Education India, 2006.

[12] Godfrey N Lance and William T Williams. Computer programs for hierarchical polythetic classification ("similarity analyses"). *The Computer Journal*, 9(1):60–64, 1966.

[13] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[14] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. In *International Workshop on Algorithms and Computation*, pages 274–285. Springer, 2009.

[15] Raymond T Ng and Jiawei Han. E cient and e ective clustering methods for spatial data mining. In *Proc. of*, pages 144–155, 1994.

[16] Raymond T. Ng and Jiawei Han. Clarans: A method for clustering objects for spatial data mining. *IEEE transactions on knowledge and data engineering*, 14(5):1003–1016, 2002.

[17] Dan Pelleg, Andrew W Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML*, volume 1, 2000.

[18] Cor J. Veenman, Marcel J. T. Reinders, and Eric Backer. A maximum variance cluster algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1273–1280, 2002.