Group: Javiera Castillo, Diego de Souza
Date: February 6th, 2018.

Datastream Processing - Lab 1

A. Initial steps

We have chosen Python as our programming language. The regular K-Means algorithm is implemented on the section *KMeans algorithm* on the Jupyter Notebook.

B. Online K-Means for datastreams

We have created a python class called `OnlineKMeans` which allows us to simulate a datastream and to apply the online KMeans algorithm. We have duplicated the code of the original K-Means into the function `run`, which will be modified to implement the online k-means algorithm presented on the lecture.

B.1. Adapting K-means code

1. As demanded, we added the variables γ, as `window_size`, and `t` to the algorithm.
2. We have created the function `update_centroids` which updates the corresponding centroid according to the formula presented on the lecture

$$\mu_c = \mu_c + \gamma_t(x_t - \mu_c)$$

3. The K-Means algorithm has been implemented as the function `run` as a 1 pass algorithm. The function `_plot_progress` allows us to visualize the progress of the algorithm, displaying the arriving data and the progression of the centroids.
4. Thanks to the function `data_stream` our code reads the dataset line by line.

B.2. Application on the SHealth dataset

1. From the readme file, we observe that we have 9 different features: Day of the week (Su-M-Tu-W-Th-F-Sa), Date (jj/mm/yyyy), Weight of the smartphone user in kg, Daily step number, Walk time during the day, Sport time for swimming activities, Sport time during the day, Estimated spent kilo calories and GPS country code.
    - To address this problem, we decided to use all the variables, except for the date, which did not provide any more information but the order of arrival of data. In the case of the day of the week, we decided to take it into account because users activity can be influenced by the day of the week (one user may have more time to walk during the weekends than during the week, for instance) and since it is a categorical variable, we took two different approaches: (i) do a one-hot-encoding

of data, (ii) transform it into a binary variable indicating if it was a week day or a weekend day. For the rest of the variables, we used its numerical values.

- We have implemented the parser function called `parse_dataset` which reads the SHealth dataset and applies the following transformations:

  (a) Transform the `Date` column into an index, which will represent the order of arrival of data.

  (b) Transform the `Day-of-the-week` column into one of the two representations mentioned before. If the parameter `enc = 'ohe'` then the column is encoded as one-hot-encoding, if `enc = 'weekend'` it is transformed into a binary variable representing week days and weekends.

  Concerning normalization of data, we decided to normalize it "on-the-fly", this implies that we keep two variables in memory `mean` and `sigma`, the mean and the standard deviation of data, which are updated "on-the-fly" ever since a new data point arrives. Using this approach, every time a new data point arrives, the centroids are denormalized (according to the old mean and sigma) and normalized according to the new mean and sigma, and the new data point is normalized according to the new mean and sigma, this allows us to address the problem with sensitive distances.

- We have implemented the function `distance` which computes the euclidean distance between two points. Since our data is normalized (on-the-fly) this distance should perform well.

2. We tested our online KMeans algorithm for different values of K and γ. The following figures show some of the results. On the jupyter notebook the option `plot_progress= 'True'` allows to visualize the evolution of the distances between the data and the centers as the stream is processed.

   Remark: Just for better visualization purposes we only considered the first 100 data points of the dataset to create the next figures.

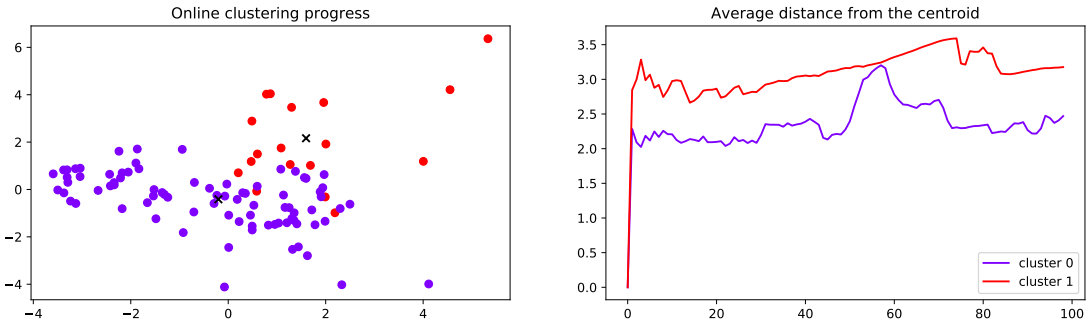

Figure 1: K=2, $\gamma = 7$

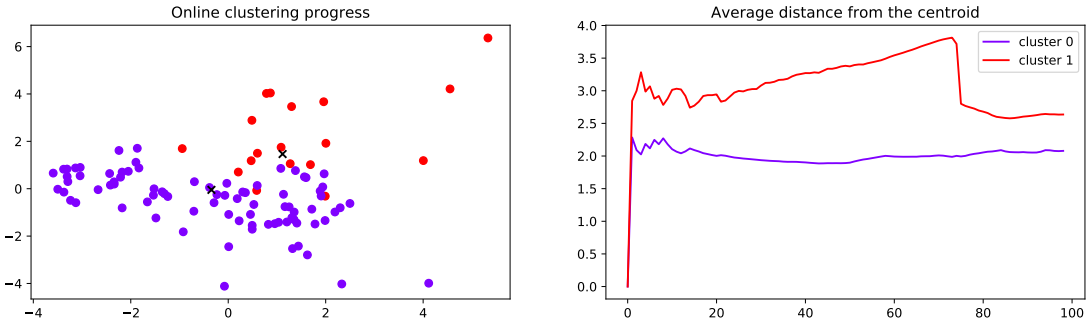

Figure 2: K=2, $\gamma = 30$

On Figures 1 and 2, we see the online KMeans algorithm applied for $K = 2$ and two different values for γ. We observe that the algorithm seems correct, since it clearly displays two well separated clusters.

By the other hand, we observe indeed that on the average distance from the centroid plot (right hand plot) there are some spikes on the distance. We remark that the spikes differ depending on the value of γ.

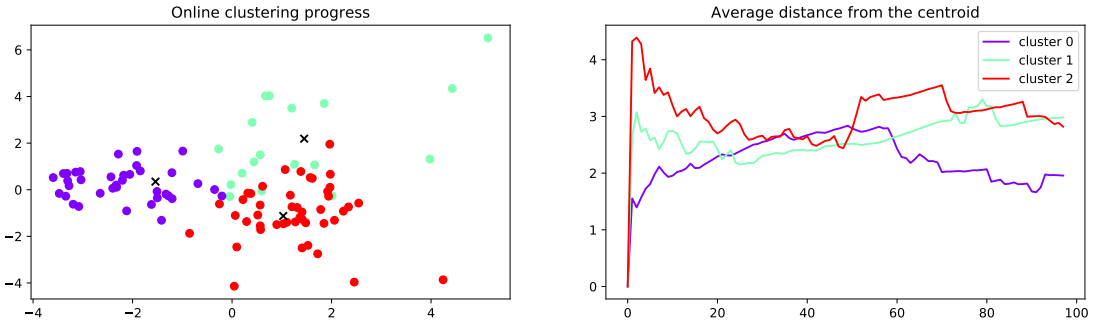

Figure 3: K=3, $\gamma = 7$

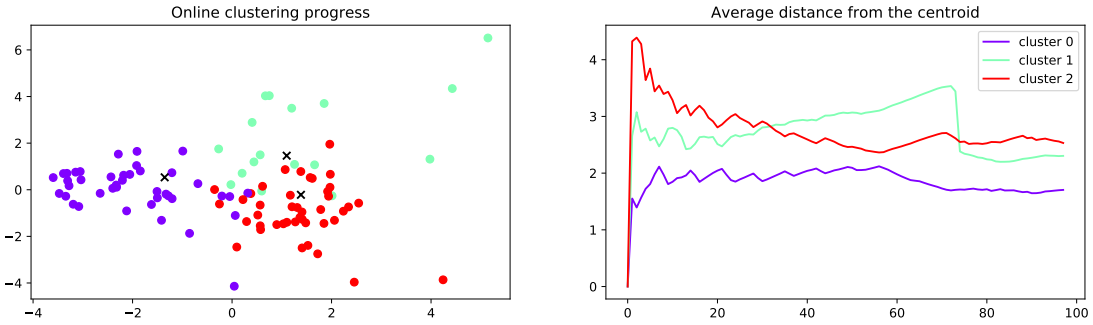

Figure 4: K=3, $\gamma = 30$

On Figures 3 and 4, we display the results of the online KMeans algorithm applied for $K = 3$. As in the previous case, we observe that the algorithm works correctly, since it clearly displays three well separated clusters.

Same as before, we observe that the distance function features some spikes, which vary depending on the window size parameter.

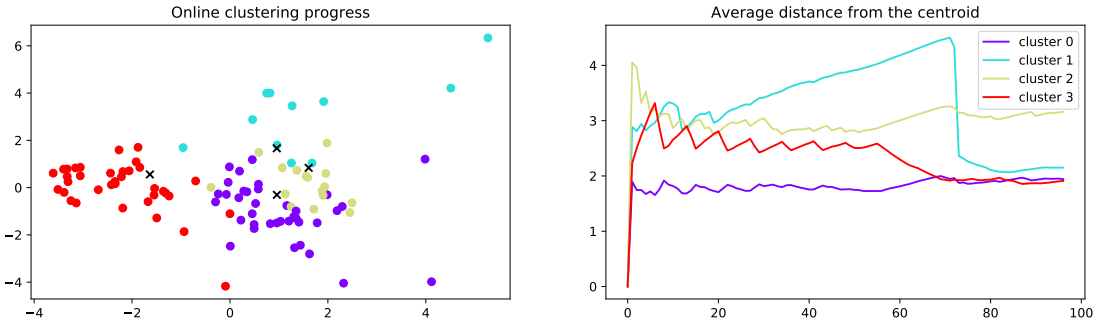

Figure 5: K=4, $\gamma = 30$

Figure 5 shows the performance of the algorithm for $K = 5$ clusters.

3. From the figures on the previous part, we observe that when $\gamma = 30$, the distance plot is more stable, while when $\gamma = 7$ the distance plot is more variable. This might be explained by the fact that as γ is bigger, the algorithm takes into account more points to update the centroids, and when γ is small last arriving points are more relevant when updating the centroids.

   Those curves may help us to choose the best γ parameter.

   It is important to notice the existing trade-off of choosing a big or a small γ. If we consider a too big window size γ, the distance will be more stable, but we might be considering irrelevant points to update the centroids. By the other hand, considering a very small γ implies giving too much importance to last arrived data points and the distance is less stable.

4. We modified our code to create a new cluster when a data point is too far from all existing centers and to remove the clusters that did not get any data assigned for a while.

   To do this, we added some new variables:

   (a) `add_remove`: boolean variable. If `add_remove='False'` we consider a fixed number of clusters, otherwise we will add or remove clusters on the fly.

   (b) `threshold_create`: If the distance between the arriving data point and all the existing centroids is bigger than `threshold_create`, the algorithm creates a new cluster whose centroid is the new data point.

   (c) `threshold_remove`: If one cluster has not been assigned any data point in more than `threshold_remove` passes, the cluster is removed.

   Remark: If one cluster is removed it means that from that moment, there will not be any data point assigned to that cluster, nevertheless all the previous data points that were assigned to this cluster will remain on it. In practice what our implementation does to *remove* the cluster is to set all its centroid coordinates as infinity.

   The following figure shows an example of this version of the algorithm.

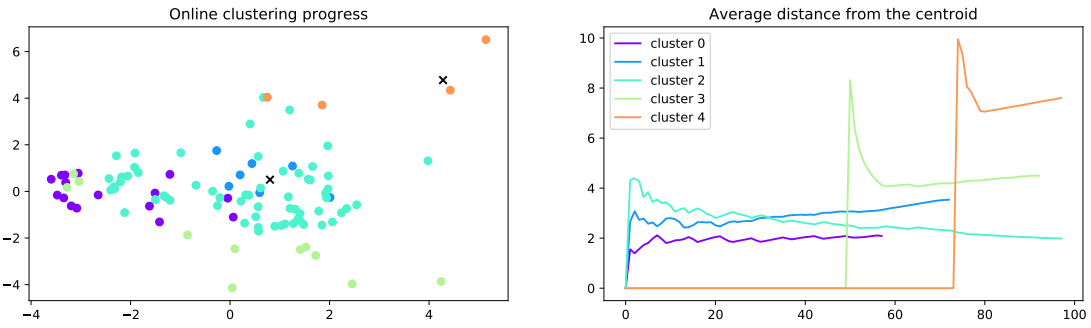

Figure 6: New version of Online KMeans alogorithm

Figure 6 represents an instance of the new versio of the online KMeans algorithm, which was initialized with $K = 3$ clusters.

On the average distance plot we observe that:

- Around the 50-th pass, cluster 3 was created.
- Around the 60-th pass, cluster 0 was removed.
- Around the 75-th pass, cluster 1 was removed and a little bit later cluster 4 was created.
- Around the 95-th pass, cluster 2 was removed.

At the same time, we observe that data that was already assigned to a cluster is removed after (such cluster 0, cluster 1 or cluster 2), remain assigned to the corresponding cluster, but no new data will be on this class.

5. The "cold start problem" refers to the fact that, when the centroids are initialized at random, during the first iterations the algorithm has not enough information to draw any meaningful information about data.

   Our algorithm already addresses this problem, since by default it does not initialize the centroids at ramdom, but taking the first K data points to arrive as centroids. Anyway, we have implemented the cold start version of the algorithm, which can be used setting the variable `cold_start = 'True'`.

   Figures 7 and 8 show the same setting, with a cold start and without cold start.

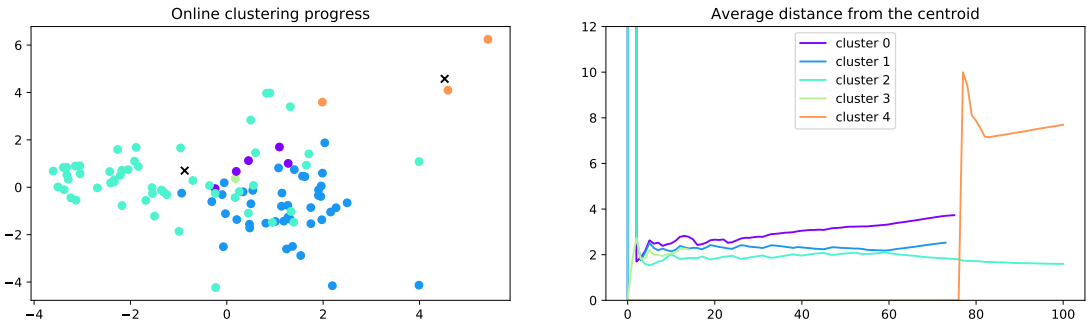

Figure 7: Cold Start

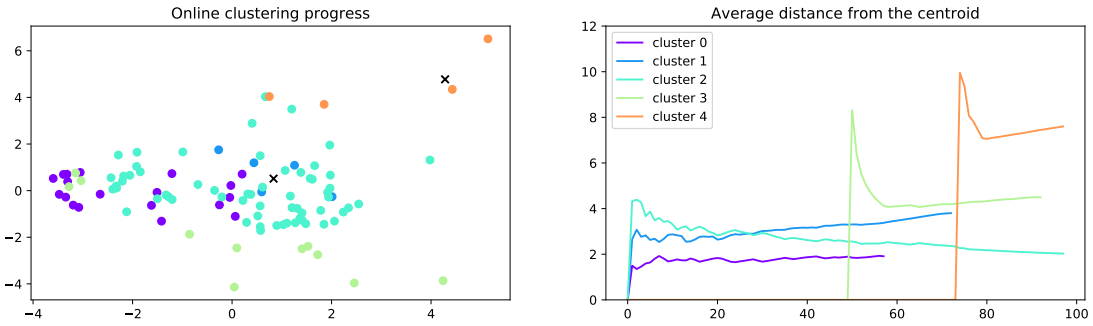

Figure 8: Not cold start

From the previous figures we observe that results are not that different from those two approaches, since they both start with $K = 3$ clusters, create 2 more clusters during the streaming and the end it finishes with $K = 2$ clusters.

Nevertheless the distances in the second one are more stable.