# TI2736-C Datamining
# Assignment 5: Clustering

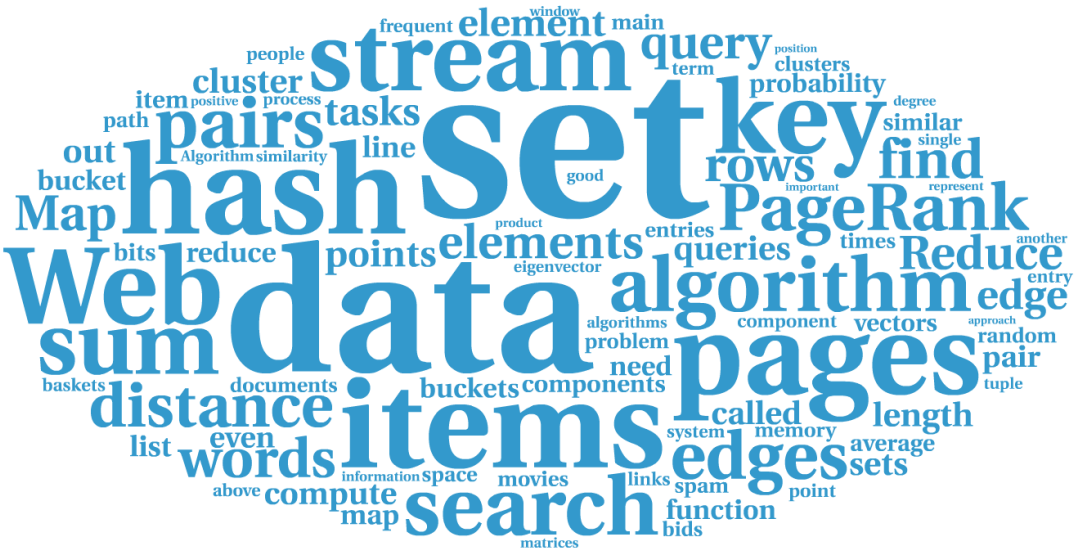
Delft University of Technology, February–April 2015

Thomas Abeel, Marcel Reinders
Zekeriya Erkin, Julian Kooij
Daan Rennings, Tom Viering

*Pattern Recognition and Bioinformatics Group*

based on work from

Laurens van der Maaten, Hans Gaiser




TU Delft

# 5 Clustering

In the following exercises you will implement algorithms designed to detect clusters in unlabeled data. The exercises will begin with the implementation of a hierarchical clustering method, followed by the k-means clustering method.



**Exercise 5.1. Hierarchical clustering**

Before we will work on the actual clustering of our data, we need some `Cluster` class with which we can work. In `Cluster.java` you can already find a template for this class. For the hierarchical clustering method we need to do three operations with clusters:

1. We need to merge two clusters.
2. We need to calculate the centroid of a cluster.
3. We need to calculate the distance between two clusters. This comes down to calculating the Euclidean distance between its centroids.

**Step 1.** First we will create the method that calculates the centroid of the cluster, `Cluster.centroid`, as the mean vector of all feature vectors. This method will get called frequently when performing step 3, while the cluster (and thus the centroid) often remains unchanged. To increase efficiency, once the centroid is calculated it is stored and reused, until the cluster changes, at which point a new centroid should be calculated.

Hint: use the `changed` flag to see if the centroid needs to be recomputed, or if the current value is still valid.

**Step 2.** Next we will complete the `meanDistanceTo` method, which calculates the distance of the current cluster centroid to another cluster centroid for *average linkage clustering*.

**Step 3.** We will now create the `update()` method in `HierarchicalClustering.java`. This method gets called every time the user presses the space bar when viewing the data in the scatter plot, so that progress can be viewed in real time. The method should first check the distance between each pair of clusters. The two clusters with the minimum distance between them will be merged into one bigger cluster.

Hint: Instead of adding a new cluster which is the union of clusters $a$ and $b$, it is advised to merge cluster $a$ into cluster $b$ and remove cluster $a$ from the clusters for clearer visualization.

Note: The order of a pair of clusters does not matter; the distance between $c_1$ and $c_2$ is the same as the distance between $c_2$ and $c_1$. It is therefore (for efficiency) not advised to use a `for-each` loop as this will check both combinations.

**Step 4.** In `main.java`, complete the `hierarchical` method by constructing a `HierarchicalClusteringPlotter` object to visualize the clustering process. This object should read in data from `"data/cluster.txt"`, which contains three Gaussian distributed clusters, with $k$ set to 3. Execute the hierarchical clustering code and verify that the clusters are roughly correctly identified.

Note: Use the space bar to trigger a call to the `HierarchicalClustering.update` method.

**Question 4.1.** If we imagine the process of the hierarchical clustering as that of constructing a tree, what would we find at the leaves of this tree? What would we have as root if we set $k$ to 1?

**Question 4.2.** What does the variable $k$ do to the tree?

**Step 5.** Run the algorithm on `"data/cluster_lines.txt"` instead.

**Question 5.1.** Can you explain the results?

Now we will change the clustering method from *average linkage* to *single linkage clustering*.

**Step 6.** Implement `Cluster.distancesToCluster` and `Cluster.minDistanceTo`. The first method calculates a list of distances for each pair of feature vectors between two clusters, while the second method calculates the minimum of such a list.

**Step 7.** Use `Cluster.minDistanceTo` instead of `Cluster.meanDistanceTo` in `HierarchicalClustering.update`. Run the algorithm again on the `"data/cluster_lines.txt"` file.

**Question 7.1.** What is the result now? Is it better?

**Step 8.** Run the algorithm again, using `Cluster.minDistanceTo`, on the `"data/cluster.txt"` file.

**Question 8.1.** Again, what is the result now, and is it better?

**Step 9.** In `main.java`, complete the `hierarchicalDigits` method. This method should construct a `HierarchicalClustering` object and have it cluster the `"data/train_digits.txt"` file with $k$ set to 10. This dataset contains 100 images of hand-written digits (0 to 9, 10 images each). Keep updating the clusters until there are 10 clusters left.

**Step 10.** Visualize the clusters centroids using `DigitFrame` objects. Try to identify the different digits in the cluster centroids.

**Exercise 5.2. $k$-means clustering**

Next we will implement the (original) $k$-means clustering algorithm. This algorithm works slightly different from the previous hierarchical clustering method; instead of building a tree of clusters until only $k$ clusters remain, it will try to improve its clustering each round until convergence. It does so by first selecting $k$ randomly chosen points of the original dataset and classify each item as belonging to its closest cluster. Note that this is also slightly different from the algorithm discussed in the book, however this is how $k$-means is generally implemented. The steps of the $k$-means clustering algorithm approach are as follows:

1. Initialize cluster centers at random locations.
2. Assign each point to a cluster.
3. Update the cluster centers.
4. Go to step 2 unless converged or a number of iterations have been done.

**Step 1.** First we will finish the `KMeans.addInitPoint` method. This method selects a random point in the dataset, constructs a cluster around it and adds the cluster to the list of clusters.

**Question 1.1.** What would be a better method to determine which points are the initialization points? What are its advantages/disadvantages?

**Step 2.** Next we will implement the update step of the KMeans clustering algorithm in `KMeans.update`. First, it will use the `addInitPoint` method if no clusters are present yet to initialize $k$ clusters. After this check, we perform an iteration of the KMeans clustering algorithm. This exists out of multiple steps:

1. Calculate the centroids of each cluster and save them.
2. Then remove all points from all clusters.
3. Finally add each point to the closest cluster centroid (the centroids that you saved earlier).

Hint: You will want to save the cluster centroids, because otherwise the cluster centroids will change during step 3 because they will be recalculated.

**Step 3.** In `main.java`, complete the `kmeans` method; construct a `KMeansPlotter` object and have it analyse `"data/cluster.txt"` with $k$ set to 3. Run the $k$-means clustering algorithm and verify that it is clustering the points as intended.

Note: Use the space bar to trigger a call to the `KMeans.update` method.

**Step 4.** Implement the `Cluster.calculateAverageRSS` method. This method should calculate the average sum of residual squares of the cluster. It should do this by summing over the squared difference with each point to the centroid and dividing by the total number of points.

**Step 5.** In `main.java`, implement the `kmeansTuneK` method. This method should test out several values for $k$. Let $k$ range from 1 to 10 and for each $k$, run the $k$-means algorithm on `"data/cluster.txt"` with 10 `update` iterations. After the algorithm is done, calculate the average RSS of all clusters and print them to the screen.

**Question 5.1.** What seems to be a good choice for $k$, judging by the results?

**Question 5.2.** What would be the average RSS if we set $k$ equal to the number of points in our dataset?

**Step 6.** In `main.java`, complete the `kmeansDigits` method; construct a `KMeans` object and have it analyse `"data/train_digits.txt"` with $k$ set to 10. Visualize the cluster centroids using `DigitFrame` objects. Try to identify the digits in the cluster centroids.