‘Data Analysis’
Computer Lab Session
Principal Component Analysis

Master 1 MLDM / CPS^2 / COSI / 3DMT
Saint-Étienne, France

Ievgen Redko

1 Outline

1. Introduction to Python (1/2)
2. Introduction to Python (2/2)
3. Probability, Random Variables and Probability Distributions
4. Linear Algebra (1/2)
5. Linear Algebra (2/2)
6. Linear Regression (1/2)
7. Linear Regression (2/2)
8. Principal Component Analysis
9. **Clustering (1/2)**
10. **Clustering (2/2)**

Outcome

The objective of this lab is to become familiar with a simple clustering algorithm (k-means) and its translation in a Python program.

2 Background

k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

This algorithm works as follows:

- Given a set of individuals $(x_1, x_2, \ldots, x_n)$, where each individual is a vector of dimension d, k-means algorithm aims to partition the n individuals into k sets $S = S_1, S_2, \ldots, S_k$ $(k \leq n)$ while minimizing the distance between each point into each partition:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

  where $\boldsymbol{\mu}_i$ is the mean of the points in S_i.

- Choose k points representing the mean positions of the initial partitions $m_1^{(1)}, \ldots, m_k^{(1)}$ (e.g., randomly)

- Repeat until convergence:

  - **Assignment step:** Assign each observation to the cluster whose mean yields the least within-cluster sum of squares. Since the sum of squares is the squared Euclidean distance, this is intuitively the 'nearest' mean (e.g., according to the Voronoi diagram generated by the means):

$$S_i^{(t)} = \left\{ \mathbf{x}_j : \left\| \mathbf{x}_j - \mathbf{m}_i^{(t)} \right\| \leq \left\| \mathbf{x}_j - \mathbf{m}_{i^*}^{(t)} \right\| \forall\, i^* = 1, \ldots, k \right\}$$

  - **Update step:** Calculate the new means to be the centroids of the observations in the new clusters:

$$\mathbf{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$$

- Convergence is reached when there is no more change.

Exercise 1: Implementing k-means

We will code the k-means algorithm in order to see how the partitioning of the data changes during the iterations. For this we consider a classical dataset: iris plant. This dataset is a set of measurements made on 150 flowers of iris plants with three classes (*iris setosa*, *iris versicolor*, and *iris virginica*). In the dataset, there are 50 representatives of each species. The 4 variables measured (in centimeters) are the widths and lengths of the sepals and petals.

Iris dataset is present in the packages launched by default when loading Python, so we can use the dataset simply by loading it into memory with the sklearn.datasets.load_iris() function. We will only copy the variables measures (the first 4) in a data matrix called X, the fifth variable will be copied in a vector called y and can be used for further processing (e.g., to measure the quality of final partition obtained).

1. Load Iris data set as explained above. Store the size and the number of features of X in two variables n and d.

2. Implement the z-score normalization (for each column, we subtrack its mean and divide it by its standard deviation). Use np.apply_along_axis() function and check that your implementation gives the same result as the built-in function sklearn.preprocessing.StandardScaler.

3. Implement the k-means algorithm function by following the pseudo-code given below:

```
def find_clusters(X, n_clusters):
    centers ← randomly choose k points of X
        while not converged:
        labels ← pairwise_distances_argmin(X, centers)
        new_centers ← average of the points with new cluster assignments
        if centers == new_centers:
                break
        centers = new_centers
    return centers, labels
```

4. Run the program with X containing only the 3rd and 4th variable of Iris dataset. What are the differences?

5. Run the program with ruspini.txt dataset available on *Claroline* platform. According to you, just by seeing a simple graphical representation of Ruspini dataset, how many clusters can be found in the dataset? Try different values for k to see the impact on the results.

Exercise 2: Analyzing the clustering results

Issues that arise when doing clustering are:

- What is "good clustering"?
- How many clusters?
- How to find the clusters?
- A good clustering method will produce high quality clusters with:
  - high intra-class similarity
  - low inter-class similarity

To analyze a given clustering qualitatively, do the following.

1. Experiment with the Iris data set using the built-in sklearn's sklearn.cluster.KMeans function.
2. Implement the Elbow methods that visually allows to define the optimal number of clusters. To do this,
   - Compute the distortion withing each cluster given by the sum of inter-cluster distances divided by the number of points in the cluster.
   - Plot the distortion as the number of cluster K and find the elbow point in the plot.