Machine learning, prediction methods and applications

Pierre Michel

MASTER in Economics - Track EBDS - 2nd Year (2022)

K-means algorithm from scratch

Motivation and objectives

The aim of this session is to implement your own version of K-means, in order to deal with clustering and pattern recognition tasks. You will develop an algorithm that takes K, the number of clusters as input, and returns individuals' cluster assignments and clusters' coordinates.

During this work, you are strongly invited to search for information by yourself on which functions you will use and how to optimize your code. Feel free to draw inspiration from existing publications or tutorials (by citing them).

You will have to submit a report of your activity including your developments and results, illustrated if possible. The format of the report is free and the document must be submitted on AMeTICE.

Tools you need

Python or R

For Python users, the use of the library **numpy** for high-level computations is recommended, for the ones who are interested in the application of linear algebra in the development of machine learning algorithms from scratch. The library **MatplotLib** is very useful for drawing graphics. For R users, basis functions can be used to solve the exercise.

K-means algorithm (pseudo-code)

Randomy initialize K cluster centroids $\mu_1, \mu_2, ..., \mu_K \in \mathbb{R}^p$

Repeat

for $i = 1$ to n (**cluster assignment** step)

$c^{(i)}$:= index(from 1 to K) of cluster centroid closest to $x^{(i)}$

$$c^{(i)} = \min_k ||x^{(i)} - \mu_k||^2$$

for $k = 1$ to K (**update centroid** step)

μ_k := average (mean) of points assigned to cluster k

$$\mu_k = \frac{1}{\#\{i : c^{(i)} = k\}} \sum_{\{i:c^{(i)}=k\}} x^{(i)}$$

Exercise: K-means from scratch

In this exercise, you will code an algorithm from scratch to perform K-means clustering.

1. Data generation process: First generate a "toy example", a 2-dimensional dataset (only two features x_1 and x_2) in order to test your implementation. For example, by setting $K = 4$, you could define a simulation model specific for each cluster, as follows:

   Cluster 1: $x_1 \sim \mathcal{N}(1, \sigma)$ and $x_2 \sim \mathcal{N}(1, \sigma)$

   Cluster 2: $x_1 \sim \mathcal{N}(1, \sigma)$ and $x_2 \sim \mathcal{N}(-1, \sigma)$

   Cluster 3: $x_1 \sim \mathcal{N}(-1, \sigma)$ and $x_2 \sim \mathcal{N}(-1, \sigma)$

   Cluster 4: $x_1 \sim \mathcal{N}(-1, \sigma)$ and $x_2 \sim \mathcal{N}(1, \sigma)$

2. According to the previous data simulation model, setting $\sigma = 0.1$, generate $K = 4$ clusters, each cluster should contain 25 observations. Draw the corresponding scatterplot, using a different color for each cluster. Draw the same scatterplot for different values of σ ($\sigma = 0.2, 0.3, 0.4, 0.5, ...$). Explain the effect of parameter σ on the resulting clusters.

3. The training dataset should be a matrix $X \in \mathbb{R}^{n \times 2}$ ($n = 25K$ observations and 2 features). Propose a way to initialize K-means, in other words: randomly pick K cluster centroids $\mu_1, \mu_2, ..., \mu_K$.

4. Implement the cluster assignment step: for each observation $x^{(i)}$, assign this observations to its closest cluster centroid, in other words compute for all $i \in \{1, ..., n\}$:

$$c^{(i)} = \min_k ||x^{(i)} - \mu_k||^2$$

5. Implement the update cluster centroid step: for each cluster k, consider the set of obervations assigned to cluster k, and compute the average (mean) of these observations, in other words, compute for all $k \in \{1, ..., K\}$:

$$\mu_k = \frac{1}{\#\{i : c^{(i)} = k\}} \sum_{\{i:c^{(i)}=k\}} x^{(i)}$$

6. Draw again the corresponding scatterplot, using a different color for each point in a cluster, and add K points representing the current cluster centroids (you should use a different symbol to distinguish between points and cluster centroids).

7. Use a loop in order to repeat steps 4 to 6. The number of iterations should be large enough for the algorithm to have time to converge. Note that a parameter ϵ (a small value) can be introduced here to stop the algorithm automatically, we can force the algorithm to stop when $||\mu_k^{(t+1)} - \mu_k^{(t)}|| < \epsilon$, where $\mu_k^{(t)}$ represents the cluster centroid μ_k at iteration t.

8. Implement the whole algorithm in a same Python (or R) funtion, named `Kmeans`, that takes the matrix X as input, and returns as outputs: the clusters assigned to the observations $c^{(1)}, c^{(2)}, ..., c^{(n)}$ and the cluster centroids $\mu_1, \mu_2, ..., \mu_K$. Ideally, the function should plot the final scatterplot, with a different color for each cluster, and the corresponding cluster centroids.

9. Modify your function and implement "repeated" K-means, in order to reduce the risk of finding local optima. The final function must be applied to the simulated data (see question 1), with different values of σ. Propose a way to compute the quality of the obtained partition: is there a way to compute a misclassification error rate in this case ?

10. Generalize your function for a matrix $X \in \mathbb{R}^{n \times p}$, where $p > 2$.

11. Test your function on a real dataset. Many datasets can be easily loaded on Python with the library `scikit-learn`: https://scikit-learn.org/stable/datasets/index.html. For R users, have a look on the

package `datasets`. Here again, propose a way to compute the quality of the obtained partition: is there a way to compute a misclassification error rate in this case ?

Note: if present in the real dataset, the feature corresponding to the target variable Y should be removed before the clustering.

Example: Loading a dataset with Python (`scikit-learn`, iris dataset example)

```python
# import the module datasets of scikit-learn
from sklearn.datasets import load_iris
# load the dataset and store it in `data`
data = load_iris()
# the data (without the target variable) can be accessed
print(data.data)
# for informtion, the target is accessed as follows
print(data.target)
```

Example: Loading a dataset with R (`datasets`, iris dataset example)

```r
# import the library datasets (available by default)
library(datasets)
# load the dataset and store it in `data`
data = iris
# the first four columns are  the predictors
print(data[,1:4])
# "Species" is the target variable
print(data$Species)
```