INFAIA01
Practical assignment: K-means clustering

GOAL

Given the dataset (containing information about clients of a wine company and their purchasing behaviour), write a program which executes **k-means clustering** to divide the clients in groups based on their behaviour (that is, clients with similar buying behaviour should belong to the same group).

INPUTS

The user-specified parameters of the program should be:

- amount of clusters (k);
- amount of iterations (how many times do you repeat the *whole* k-means algorithm to find the best solution).

The dataset to use comes from Chapter 2 of the book:

- the complete dataset can be found at http://eu.wiley.com/WileyCDA/WileyTitle/productCd-111866146X.html (in the *Downloads* section)
- the pre-processed dataset (a csv file containing only the information about the observations that have to be clustered) is available on N@tschool ("Winedata.csv").

Important note: the clients are the observations/points that have to be clustered. Each of them is a vector composed by 32 numbers, one for each of the 32 wine offers. These numbers can be only 0 or 1:

- 0 if the client didn't take that offer;
- 1 if the client did take that offer.

The points to cluster are thus in 32 dimensions. The number of clients is 100, which means that there are 100 points in total to be clustered.

ALGORITHM

As explained in the slides, one execution of the k-means algorithm consists in:

- *Initialization centroids* (choose a method between those presented in class; you can also try more than one method to see if the results are influenced by this choice);
- *Main loop k-means*
  - Assign each point to the closest[1] centroid;
  - Recompute centroids;
  - … until a termination condition is met (for example, centroids stop changing).

To obtain good results, this algorithm must be executed a certain number of times (how many is specified in the input parameters): at the end of every execution, you must compute the SSE (Sum of Squared Errors) of the solution obtained, and keep only the best solution found until that moment (that is, the one with lowest SSE).

[1] Using the Euclidean distance.

OUTPUT

At the end of the execution, you have to print the **best solution found** (that is, the clustering associated to the smallest SSE). Together with the result of clustering, print also the value of the **associated SSE**. Compare your solution with that of the book to see if they match[2]. You can try first to execute your program with 4 clusters ($k = 4$) and then with 5 ($k = 5$): the book shows the results in both cases.

Important: to improve the understandability of the results, instead of just printing which points belong to each cluster, we need a post-processing step. The output of the algorithm should be displayed in a similar way as the book does (see pictures 2.22, 2.23, 2.24. 2.25): for each cluster, you have to consider which clients belong to it and count how many times each of the 32 offers was bought by that group of clients. Then, you have to sort the offers in order to show only the "most bought" ones in each cluster. Display only the offers which were bought more than 3 times.

Small example: suppose we have a dataset made by only 5 offers and, after executing the algorithm, one of the clusters is composed by clients #1,#2,#3,#4,#5. Suppose also that the information we have about these clients is the following:

Client #1 buys offers 1,2

Client #2 buys offer 3

Client #3 buys offers 2,3

Client #4 buys offer 3

Client #5 buys offers 2,3

Then, we want to print this cluster in the following way:

OFFER 3 → bought 4 times (by clients of this cluster)

OFFER 2 → bought 3 times

OFFER 1 → bought 1 time

OFFER 4 → bought 0 times

OFFER 5 → bought 0 times

Clearly, the clients of this cluster prefer offers 2 and 3.

[2] The results will, of course, not be a 100% match (given the role of randomness in this algorithm) but they should be at least quite similar.