HW 2 Instructions

Irina Gaynanova

K-means

K-means is one of the most popular clustering algorithms. Given n data points x_i in p dimensions, $x_i \in \mathbb{R}^p$, the algorithm iteratively divides the points into K clusters/groups such that the points within each group are most similar. Specifically, it aims to minimize

$$\sum_{k=1}^{K} \sum_{x_i \in \text{cluster k}} \|x_i - \mu_k\|_2^2,$$

where

$$\mu_k = \frac{1}{n_k} \sum_{x_i \in \text{cluster k}} x_i$$

is the center or **centroid** of the kth cluster with n_k being the number of points in the cluster k. Here $\|x_i - \mu_k\|_2^2$ is the squared Euclidean distance between x_i and μ_k, although other distance metrics are possible.

In this HW, you will implement the K-means algorithm and apply it to ZIPCODE data.

Algorithm's implementation

To minimize the above objective, K-means performs iterative adjustment of cluster centers. The algorithm is not guaranteed to find the absolute minimizer, but (hopefully) comes pretty close.

To start, the algorithm chooses random K points out of n as the cluster centers μ_k.

Given the current cluster centers, at each iteration the algorithm

- computes the Euclidean distance from each point $i = 1, \ldots, n$ to each center μ_k
- assigns each point to the cluster corresponding to the nearest center (among the K)
- after all n points are assigned, the algorithm recomputes the centroid values μ_k based on new assignments

These iterations are repeated until some convergence criterion is met (i.e. the centroid values don't change from one iteration to the next). For simplicity, here we will instead **fix the number of iterations**, and stop after than number if reached.

You are provided the starter code in **FunctionsKmeans.R** that contains **MyKmeans(X, K, M, n_iter)**: This function takes $n \times p$ data matrix X, provided number of clusters K, (optional) initial $K \times p$ matrix of cluster centers M and (optional) pre-specified number of iterations for the algorithm. It returns the vector Y of length n of cluster assignments (numbers from 1 to K). More details are in the function comments. **Please strictly follow the name, input and output guidelines as everything will be checked using automatic tests**

You are welcome to create any additional functions you would like, and place them within **FunctionsKmeans.R**. We will only test **MyKmeans(X, K, M, n_iter)** function, but presumably its correct performance will rely on correctness of your other functions. You are **not allowed to use any external R libraries** for this assignment, and you **are not allowed to use dist function**.

Other things to keep in mind when implementing:

- You can and should indirectly check your code on simple examples before proceeding to the data (i.e. what happens if I use two normal populations? what happens with different initial M? How do cluster centroids change from one iteration to next, do they appear to stabilize? how does it compare

with built in **kmeans** function in R?). I will use automatic tests to check that your code is correct on more than just the data example with different combinations of parameters.

- Make sure to check compatibility of M. I will purposefully supply wrong M to the algorithm and see if it complains correctly.
- You will need to vectorize the distance calculations to pass the speed requirements.

Application to ZIPCODE data

The file **ZIPCODE.txt** provides 7,291 points of vectorized 16 by 16 pixels of image that should represent one of the 10 digits (from 0 to 9). The first column contains the correct cluster assignment (from 1 to 10), and the rest (256) are pixel values. The starter code in **ZIPCODE_example.R** loads the data, and guides you through the analysis. You will need to fill the remaining steps.

Because the output of k-means algorithm is dependent on initial centroid values, you are asked to try the algorithm nRep = 50 times with a different random initialization of M. Choose true number of clusters for K ($K = 10$). At each replication, call your algorithm as implemented in **MyKmeans** and evaluate the performance of the algorithm using the **RandIndex** implemented in the R package **fossil**. RandIndex takes values between 0 and 1, with 1 being the perfect match. The example code is provided but you will need to install the package **fossil** first if you don't have it. At the end, report the mean RandIndex for your implementation across 50 replications, and your mean run time.

Grading for this assignment

Your assignment will be judged as follows (based on 100 points)

- correctness *(50% of the grade)*

We will apply automatic tests to MyKmeans function with different examples.

- speed of implementation (you need to vectorize your code to pass the speed requirement) *(30% of the grade)*

Your speed will be dependent on your machine, but as a guidelene, my code is around 10 times slower than built-in **kmeans** on ZIPCODE data. As a rule of thumb, I expect your code on ZIPCODE data to take less than 1 minute to run on 1 replication. **(bonus)** you get 5 bonus points if your (correctly implemented) code is faster than my code

- comments *(10% of the grade)*

You need to comment different parts of the code so it's clear what they do, have good identation, readable code with names that make sense, etc.

- version control workflow *(10% of the grade)*

I expect you to start early on this assignment, and work gradually. You want to follow good commit practices: commit often, have logically organized commits with short description that makes sense.