Homework 2 - K-means algorithm

Acknowledgment: This assignment is inspired by the similar assignment by Ping Li when I took Statistical Computing Class at Cornell as a graduate student.

Attention: Because math rendering of .Rmd is not ideal, please see the enclosed pdf for correct rendering of all equations (an exact copy of this file)

K-means

K-means is one of the most popular clustering algorithms. Given n data points x_i, $i = 1, \ldots, n$, 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. Note that the number of clusters K must be specified by the user. In HW2, you will implement the K-means algorithm and see its application on 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 either a **convergence criterion is met** (i.e. the centroid values don't change from one iteration to the next), or the **maximal number of iterations is reached**, or **one of the clusters has disappeared** (this indicated bad starting point, see below). This specific implementation of K-means is sometimes referred to as **Lloyd's algorithm**.

Starter code

- **FunctionsKmeans.R** contains function `MyKmeans(X, K, M, numIter)`: This function takes $n \times p$ data matrix `X`, number of clusters `K`, (optional) initial $K \times p$ matrix of cluster centers `M` and (optional) maximal number of iterations for the algorithm `numIter`. 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**

Attention: (1) If M is supplied, than those values are used as starting cluster centers. If M is NULL (default), your function should randomly pick K points out of n as starting cluster centers M (see algorithm above). (2) Depending on the value of M, it is possible for the clusters to disappear (e.g. starting from 3 clusters, at some iteration all points are assigned only to cluster 1 or 2, and not to 3). If this happens, you should stop (function `stop`) and return the error message alerting the user to change the value of M.

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, numIter)` 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**.

Things to keep in mind when implementing:

- Make sure that your first iteration is where the first assignment takes place (see Algorithm's description), and your last iteration is where the last assignment takes place. Doing assignment earlier or later will make your iteration numbers misaligned for automatic tests.
- You should 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 when Lloyd's algorithm is used?). You will be expected to save your checks in **Tests.R** file. 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`. We will purposefully supply wrong `M` to the function and see if it complains correctly.
- You will need to **vectorize** the distance calculations to pass the speed requirements. (hint: open the brackets in Euclidean squared norm and think about how to compute the required terms efficiently). You should not be taking a lot of extra memory for vectorization.

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. Compatibility checks are part of the correctness grade. Evidence that you tested your code on more than just our example is part of your correctness grade,

- 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 2 times slower than built-in `kmeans` on ZIPCODE data. You will get full points if your code is **at most 2 times slower** comparable to mine. You will loose 5 points for every fold over. You will get +5 **bonus** points if your **completely correct** code on 1st submission is faster than mine (median your time/median mine time $<$ 0.9).

- code style/documentation *(10% of the grade)*

You need to comment different parts of the code so it's clear what they do, have good indentation, readable code with names that make sense. See guidelines on R style, and posted grading rubric.

- version control/commit practices *(10% of the grade)*

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