HOMEWORK #2

Software Project (0368-2161)

Due Date: 09/06/2022

1 Introduction

The K-means++ algorithm is used to choose initial centroids for the K-means algorithm. In this assignment you will implement this algorithm in Python and integrate it with the K-means algorithm of HW1 that will be ported to a C extension using the C API.
The goals of the assignment are:

- Port your existing C code into a C extension using the C API.
- Experience the Numpy, Pandas and other external packages.

1.1 K-means++

Given a set of N datapoints $x_1, x_2, \ldots, x_N \in R^d$, the goal is to find k initial centroids $\mu_1, \mu_2, \ldots, \mu_k \in R^d$ where $1 < K < N$.
This algorithm will replace step 1 of the K-means algorithm presented in HW1.

Algorithm 1 k-means++

```
$i = 1$
Select $\mu_1$ randomly from $x_1, x_2, \ldots, x_N$
repeat
    for $x_l, 1 \le l \le N$:
        $D_l = \min\ (x_l - \mu_j)^2,\ \forall j\ 1 \le j \le i$
    for $x_l, 1 \le l \le N$:
        $P(x_l) = \frac{D_l}{\sum_{m=1}^{N} D_m}$
    $i++$
    randomly select $\mu_i = x_l$, where $P(\mu_i = x_l) = P(x_l)$
until $i = k$
```

The algorithm starts by choosing a random observation as the first centroid. We keep growing our centroids by randomly choosing observations with a weighted probability, favoring a distant observation from the centroids we already chose. **The way we calculate the probabilities ensures**

that observations we previously chose as centroids will have zero probability of getting reselected. Note that all the centroids we chose are "as-is" observations. We have implemented a sampling method. Do not confuse these initial centroids with the calculated ones that represent the means of the clusters in the K-means algorithm.

Algorithm 2 k-means clustering algorithm

```
1: Initialize centroids $\mu_1, \mu_2, \ldots, \mu_K$ as in k-means++
2: repeat
3:     for $x_i, 1 \leq i \leq N$:
4:         Assign $x_i$ to the closest cluster $S_j$: $\underset{S_j}{\operatorname{argmin}} (x_i - \mu_j)^2, \ \forall j \ 1 \leq j \leq K$
5:     for $\mu_k, 1 \leq k \leq K$:
6:         Update the centroids: $\mu_k = \frac{\sum\limits_{x_l \in S_k} x_l}{|S_k|}$
7:
8: until convergence: [1]$(\|\Delta\mu\|_2 < \epsilon) \ OR \ (iteration\_number = max\_iter)$
```

[2] $\|\Delta\mu\|_2$: Euclidean norm for each one of the centroids doesn't change by more than ϵ.
More Info: Euclidean Norm

2 Assignment Description

Implement the following files:

1. `kmeans_pp.py`: The main interface of your assignment. It will contain; all of the command-line argument interface, reading the data, the Numpy, K-means++ implementation, the interface with your C extension and outputting the results.
2. `kmeans.c`: A C extension containing your K-means implementation from HW1 with step 1 of the algorithm (finding the initial centroids) replaced by values you'll pass from the K-means++ algorithm implemented in `kmeans_pp.py`.
3. `setup.py`: The setup file.

2.1 Main Program (`kmeans_pp.py`)

This is the interface of the program, with the following requirements:

1. Reading user CMD arguments:
    - `k`: Number of required clusters.
    - `max_iter`: (Optional) argument determines the number of K-means iterations, if not provided the default value is 300.
    - `eps`: the ϵ value for convergence [2].
    - `file_name_1`: The path to the file 1 that will contain N observations, the file extension is .txt or .csv.

[2]

- file_name_2: The path to the file 2 that will contain N observations, the file extension is .txt or .csv.

2. Combine both input files by **inner join** using the first column in each file as a key.

3. Implementation of the k-means++ algorithm as detailed in 1:

   (a) Use Numpy module for implementation.

   (b) Set np.random.seed(0)

   (c) Use np.random.choice() for random selection.

4. Interfacing with your C extension:

   (a) Import C module mykmeanssp

   (b) Call the fit() method with passing the initial centroids, the datapoints and other arguments if needed.

   (c) Get the final centroids that returned by fit().

5. Outputting the following:

   - The first line will be the indices of the observations chosen by the K-means++ algorithm as the initial centroids. Observation's index is given by the first column in each input file.
   - The second line onwards will be the calculated final centroids from the K-means algorithm, separated by a comma, such that each centroid is in a line of its own as in HW1.

An example output (assuming K = 3, max_iter = 100, eps=0.01 and that the initial centroids resulting from the K-means++ algorithm are the 1st, 5th and 23rd observations):

```
$python3 kmeans_pp.py 3 100 0.01 input_1.txt input_2.txt
0,4,22 Rows Index
-4.2435,9.1568,5.4105,9.6870,-5.7564,-7.2314      Centroids: 1,2,3
3.3226,-1.3896,-9.1927,-6.0907,-0.9954,-8.7412
8.2239,-8.5714,-8.4985,0.8969,-8.2158,-2.3753
```

2.2 K-means Algorithm (kmeans.c)

In this file you will define your C extension which will be, mainly, your implementation of the K-means algorithm from HW1, excluding step 1 which will be replaced by the K-means++ algorithm result. Requirements of this extension:

1. The C extension module should be named mykmeanssp.

2. The module API provides a function called fit():

   (a) The Function receives the initial centroids, datapoints and other necessary arguments.

   (b) Skip step 1 from HW1 and run the algorithm from HW1.

   (c) Return the centroids.

You can use or implement any auxiliary functions within the module, but only expose to the API the one described above. It's up to you to decide which and what type of arguments you want to pass when calling the API, same thing about the return.

2.3 Setup (setup.py)

This is the build used to create the *.so file that will allow kmeans_pp.py to import mykmeanssp.

2.4 Build and Running

1. The extension must built cleanly (no errors, no warnings) when running the following command:

```
$python3 setup.py build_ext --inplace
```

2. After succefull build, the program must run as detailed in example 2.1.

2.5 Assumptions and requirements:

Note that the following list applies to all pf the files in this assignment:

1. You may assume that the input files are in the correct format and that it is supplied.
2. Validate that the command line arguments are in correct format.
3. Outputs must be formatted to 4 decimal places (use: '%.4f') in both languages, for example:
   - 8.88885 ⇒ 8.8888
   - 5.92237098749999997906 ⇒ 5.9224
   - 2.231 ⇒ 2.2310
4. You may import external includes (in C) or modules (in Python) that are not mentioned in this document, but has to be supported at NOVA.
5. 3 input files and their corresponding output files examples are provided within the assignment in Moodle. (**YES**, the files have an extra empty row and this is the expected behaviour).
6. Comment your code. If you use code from the internet, please cite the source.
7. Handle errors as following:
   (a) In case of invalid input, print "Invalid Input!" and terminate the program.
   (b) Else, print "An Error Has Occurred" and terminate.
8. Do not forget to free any memory you allocated.
9. You can assume that all given data points are different.
10. Use double in C and float in Python for all vector's elements.

2.6 Submission

1. Please submit a file named id1_id2_assignment2.zip via Moodle, where id1 and id2 are the ids of the partners.
   (a) In case of individual submission, id2 must be 111111111
   (b) Create the zip file using **only** linux cmd:

```
$zip -r id1_id2_assignment2.zip your_directory_name
```

2. The zipped file must contain **only** the following files (at the root, no other folders):
   (a) `kmeans_pp.py`
   (b) `kmeans.c`
   (c) `setup.py`
   (d) `bonus.py` (optional)

2.7 Remarks

For any question regarding the assignment, please post at the HW_2 discussion forum.

3 Optional Bonus (5 points)

This section covers an optional bonus that involves working with matplotlib. The goal of this bonus is to demonstrate the use of the elbow method to determine the optimal number of clusters for the k-means clustering.

3.0.1 Elbow Method

We will define the quality of the k-means clustering algorithm as the sum of squared distances of samples to their closest cluster center:

$$inertia := \sum_{i=1}^{N} (x_i - \mu_{x_i})^2$$

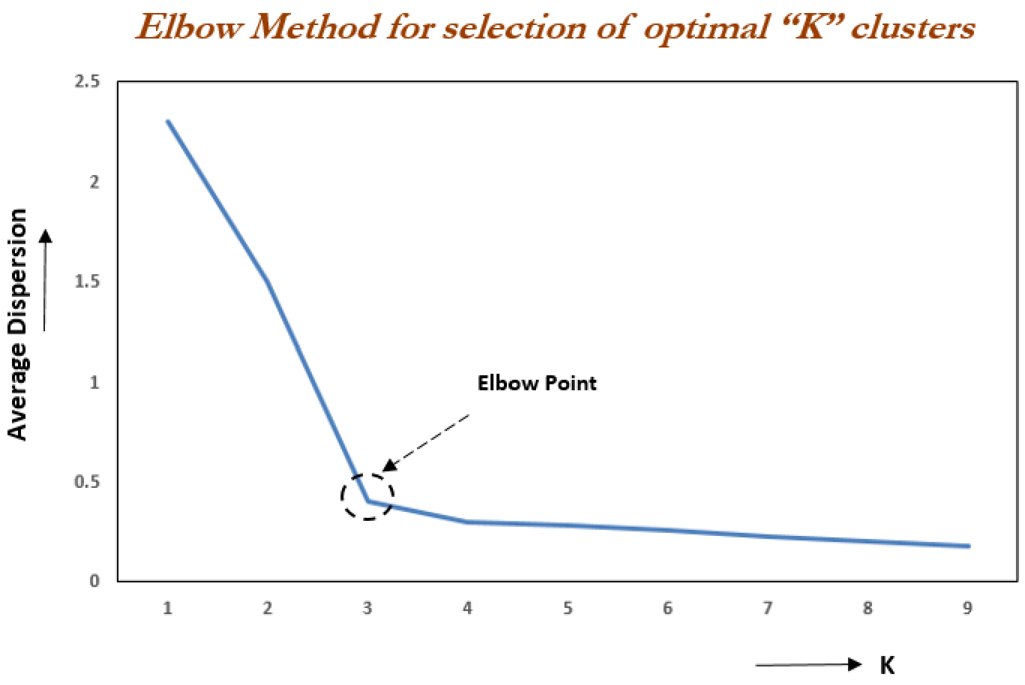
where μ_{x_i} is the centroid of the cluster x_i is in.

The idea of the elbow method is to run k-means clustering with a range of values of k and for each value of k calculate the inertia. Then, plot a line chart of the inertia for each corresponding value of k. If the line chart looks like an arm, then the "elbow" on the arm is the value of k that is the best. That elbow could be defined as a range of k's if it is unclear exactly where the elbow is at. We would like you to plot that line chart on the iris dataset, using the module sklearn . The iris dataset contains a 4-dimentional 150 observations. Use the load_iris() API to access the iris dataset. Run the sklearn k-means algorithm (using the k-means++ initialization and with a random_state=0) for the values of k ranging from k=1 till k=10 and plot the inertia for each value of k using the matplotlib module.

It will be submitted with the following requirements:

1. As a Python file named `bonus.py`
2. This program will only produce an output `elbow.png` in the program folder.
3. Doesn't require any input.
4. Has no tester file to check against.

5. Annotate the chosen k on the plot, where the ‘elbow’ is. See example below:


Elbow Method for selection of optimal “K” clusters
Average Dispersion
Elbow Point
K