Assignment 2 - Kmeans++

Software Project (0368-2161) Due: 30.01.2025

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++

Algorithm 1 k-means++ centroids initialization

1: Choose one center uniformly at random among the data points.
2: For each data point x not chosen yet, compute D(x), the distance between x and the nearest center that has already been chosen.
3: Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to $P(x_l) = \frac{D_l}{\sum_{m=1}^{N} D_m}$.
4: Repeat Steps 2 and 3 until k centers have been chosen
5: Now that the initial centers have been chosen, proceed using standard k-means clustering.

Algorithm 2 k-means clustering algorithm

1: Initialize centroids as in Algorithm.1
2: **repeat**
3: Assign every x_i to the closest cluster k: $\underset{k}{\operatorname{argmin}}\, d(x_i, \mu_k),\ \forall k\ 1 \leq k \leq K$
4: Update the centroids: $\mu_k = \frac{1}{|k|}\sum_{x_i \in k} x_i$
5: **until** convergence: $(\Delta\mu_k < \epsilon)\ OR\ (iteration\_number = iter)$

Where:

- $d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 \cdots (p_d - q_d)^2}$: More Info: Euclidean Distance.
- $\Delta\mu_k$: Eclidean Distance, between the updated centroid to the previous one. (this should be checked for every centroid).
- D(x): minimal Eclidean Distance between point x and centroids.

2 Assignment Description

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. `kmeansmodule.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 `kmeans_pp.py`

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

1. Reading user CMD arguments:
   - K: Number of required clusters.
   - `iter`: (Optional) argument determines the number of K-means iterations, if not provided the default value is 300.
   - `eps`: the ϵ value for convergence 1.1.
   - `file_name_1`: The path to the file 1 that will contain N observations.
   - `file_name_2`: The path to the file 2 that will contain N observations.
   -

Input Variable	Valid Values	Error Message
K	$1 < K < N,\ K \in \mathbb{N}$	"Invalid number of clusters!"
iter	$1 < iter < 1000,\ iter \in \mathbb{N}$	"Invalid maximum iteration!"
file_name_*	txt or .csv file	NA [1]
eps	eps >= 0	"Invalid epsilon!"

[1] You can assume that it's valid, and is provided.

2. Combine both input files by **inner join** using the first column in each file as a key.
3. After join, sort the data points by the 'key' in ascending order.
4. Implementation of the k-means++ algorithm as detailed in 1.1:
   (a) Use `Numpy` module for implementation.
   (b) Set `np.random.seed(1234)`
   (c) Use `np.random.choice()` for random selection.
5. 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()`.

6. 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. Indices should be separated by a comma.
   - 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, iter = 100, eps=0.01` and that the initial centroids resulting from the K-means++ algorithm are the 1, 5 and 23 observations):

```
$ python3 kmeans_pp.py 3 100 0.01 input_1.txt input_2.txt
0,4,22
-4.2435,9.1568,5.4105,9.6870,-5.7564,-7.2314
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 `kmeansmodule.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. Start the file with:

```c
#define PY_SSIZE_T_CLEAN
#include <Python.h>
```

2. The final module (as seen in Python) should be named `mykmeanssp`.
3. The module API provides a function called `fit()`:
   (a) The Function receives the initial centroids, datapoints, and other necessary arguments (based on your design).
   (b) Skip step 1 from HW1 and run the algorithm from HW1.
   (c) Return the centroids.
   (d) Write a docstring for the method explaining what arguments it expects.
4. Auxiliary code can be implemented in other C files.

2.3 `setup.py`

This is the build used to create the *.so file (as shown in lesson) 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.

3. **Don't compile the C module with gcc.**

2.5 Assumptions

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

1. Validate that the command line arguments meets the requirements see Table. 1.

2. Outputs must be formatted to 4 decimal places (use: '%.4f') in both languages, for example:

   - $8.88885 \Rightarrow 8.8888$
   - $5.92237098749999997906 \Rightarrow 5.9224$
   - $2.231 \Rightarrow 2.2310$

3. 3 input files and their corresponding output files examples are provided within the assignment in Moodle. (**YES**, the input files have an extra empty row and this is the expected behaviour)

4. Handle errors as following:

   (a) In case of invalid input, print the relevant message as detailed in Table. 1 and terminate the program.

   (b) Else, print "An Error Has Occurred" and terminate.

5. Do not forget to free any memory you allocated.

6. You can assume that all given data points are different.

7. You may not import external includes (in C) or modules (in Python) that are not mentioned in this document.

8. Use `double` in C and `float` in Python for all vector's elements.

9. Don't use the key (first column) as vector component while clustering.

3 Submission

1. `id1_id2_id3_assignment1.tar.gz` via Moodle, where id1 and id2 are the ids of the partners (replace id1, id2, id3 with your ids).
   (a) In case of individual submission or submission in couples, id3 must be 111111111 (and id2 as well when submitting individually).
2. Put the following files ONLY in a folder called id1_id2_id3_assignment2:
   (a) `kmeans_pp.py`
   (b) `kmeansmodule.c`
   (c) `setup.py`
   (d) `bonus.py` (optional)
   (e) more `*.c, *.h` (optional)
3. Zip the folder using the following Linux cmd:

```
$tar -czvf id1_id2_{id3_}assignment2.tar.gz  \
        id1_id2_{id3_}assignment2
```

4 Optional Bonus (5 points)

Please note max grade including the bonus is 100 !
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.

4.0.1 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: