HOMEWORK #2

SUBMISSION DUE DATE: 29/12/2020 23:00
SUBMISSION IN SINGLES OR PAIRS

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 Numpy 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 API through an algorithm implementation.
- Create a reproducible workflow to allow an easy installation of your code, using the invoke framework.

Throughout this document, the sections are in blue and their subsections are in lighter blue.

This assignment relies heavily on the previous one and should be read with that in mind.

As in HW1, this assignment will be tested and graded on Nova.

THE K-MEANS++ ALGORITHM

Given a set of N observations $\{x_1, x_2, \dots, x_N\}$, where each observation is a d - dimensional real vector, $x_i \in R^d$, we aim at finding K initial centroids $\{\mu_1, \mu_2, \dots, \mu_K\}$ from these observations, where $K < N$.

This algorithm will replace step 1 of the K-means algorithm presented in HW1.

The K-means++ algorithm is as follows:

1. Choose μ_1 at random from $\{x_1, x_2, \dots, x_N\}$
2. For $j = 2$ till K:
    a. For each x_i in $\{x_1, x_2, \dots, x_N\}$ calculate D_i
    b. Choose μ_j at random from $\{x_1, x_2, \dots, x_N\}$ with the probability of selecting x_p given by $P(x_p) = \frac{D_p}{\sum_{k=1}^{N} D_k}$

Where D_i is defined as the minimal squared Euclidean distance between x_i and **all** of the centroids that were found up to that point:

$$\min_{z \in \{1,\dots,j-1\}} \|x_i - \mu_z\|^2$$

The algorithm starts by choosing a random observation as the first centroid. If $K = 1$ the algorithm terminates.
In step 2 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 the probability 0 (zero) 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.

To view a visualization of the algorithm, please visit:
https://www.coursera.org/lecture/ml-clustering-and-retrieval/smart-initialization-via-k-means-T9ZaG

THE IMPLEMENTATION

Your project will contain the following files:

- **kmeans_pp.py** – The "glue" and the main entry point of your project. It will contain all of the command-line argument interface, reading the data using Pandas, the Numpy K-means++ implementation, the interface with your C extension and outputting the results.
- **kmeans.c** – The 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**.
- **tasks.py** – The invoke tasks file where you'll define the API for installing your C extension.
- **setup.py** – The setup file.

Please see Moodle for a range of examples regarding the C API and invoke.

INPUT

In this assignment, the **kmeans_pp.py** will contain command line arguments.

Those arguments are *K, N, d, MAX_ITER, filename* (in this order) which stand for:

- *K* – the number of clusters required.
- *N* – the number of observations in the file
- *d* – the dimension of each observation and initial centroids
- *MAX_ITER* – the maximum number of iterations of the K-means algorithm
- *filename* – the path to the file that will contain the *N* observations of *d*-dimensional that were produced by the **"gen_file.so"** file.
  Note that **unlike** the previous assignment, this file will be read using the Pandas API.

The *filename* format (its content) is the same as in HW1.

OUTPUT

At the end of **kmeans_pp.py** the following output is expected:

- The first line will be the indices of the observations chosen by the K-means++ algorithm as the initial centroids. We refer to the first observation index as 0, the second as 1 and so on, up until *N* - 1.
- 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. **Unlike** the previous assignment, the centroids will be printed with a precision of numpy.float64.

An example output (assuming K = 3, d = 2 and that the initial centroids resulting from the K-means++ algorithm are the first, fifth and 22nd observations):

0,4,21
146.15397935502773,138.9654307430039
148.71602720183992,116.7257839635179
170.3648749558064,131.91648775208841

KMEANS_PP.PY

As stated before, this file will serve as the integration of the entire assignment. Please follow these guidelines:

- **The command-line argument interface** - similar to HW1, with the exception of reading the observations file from a path and not from the standard input, as specified in subsection "INPUT".
  To do so, use the **`argparse`** module.
  All the command-line arguments should be positional ones and have the appropriate typing.
- **Reading *filename*** – the file containing the *N* observations. It should be read using the **`Pandas`** module *read_csv()* API (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html).
- **The Numpy K-means++ implementation** – in order to implement an efficient version of the K-means++ algorithm we will use the **`Numpy`** module.
  To isolate your implementation and to guarantee that the random observations picked are aligned with those of the tester, please define the following function:

```
def k_means_pp(...):
    np.random.seed(0) #Seed randomness
```

  The first line is used to seed randomness with the value 0 and should only be called once.
  The parameters that the function receives, and returns are for you to decide.

  This function will contain your Numpy K-means++ implementation.
  In your implementation you will need to randomly select observations. To this end, use Numpy's *random.choice()* API (https://numpy.org/doc/stable/reference/random/generated/numpy.random.choice.html).
  For example, It allows to select a single value between 0 and 4 using:
  `np.random.choice(5, 1)`
  Or selecting a single value between 0 and 4 with the probability of choosing the value 0 being 0.2, the value 1 with probability of 0.3 and so on, by using:
  `np.random.choice(5, 1, p=[0.2,0.3,0,0.5,0])`

- **Interfacing with your C extension** – you will import your C extension module (that will be called **`mykmeanssp`**) and interact with it, based on an API you define**.** Keep in mind that you will need to transfer to it the initial centroids you found using the K-means++ algorithm and the observations read from *filename*.
- **Outputting the results** – uphold the instructions in subsection "OUTPUT".
  Regarding the final centroids, it is up to you to decide if to print the final centroids from the C extension (using *printf()*) or from the **kmeans_pp.py** file.

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.
It will contain: prototypes, method definitions, the module definition and the module initialization.

The C extension module should be named **`mykmeanssp`**.

You have full freedom to use the rich C API and to choose the way you interact with your module.
As stated before, you'll have to consider the way in which you pass the initial centroids you found from the K-means++ algorithm and the observations read from *filename*.

In order to create the shared object (*.so) file that can be imported in the **kmeans_pp.py** file, as seen in class, you will need to use your **setup.py** file.
On Nova this command looks like:

```
>>python3.8.5 setup.py build_ext --inplace
```

In addition, you will wrap this command with an invoke task as defined below.

TASKS.PY

This file will be used for the invoke tasks.
You should create the following tasks:

- **build** – this task is a wrapper task for the command: `>>python3.8.5 setup.py build_ext --inplace`
- **delete** – this task will delete any previous .so file of your C extension. It will use the command `>>rm *mykmeanssp*.so` and will have the alias **del.**

To run the invoke task named 'build' on Nova, for example, use the following command:

```
>>python3.8.5 -m invoke build
```

As a side note and not as requirement for your submission – we propose considering using invoke tasks to measure the runtime of your code and to test it against the provided tester. You can submit those tasks if you wish to.

SETUP.PY

This is the file used to create the *.so file that will allow **kmeans_pp.py** to import **`mykmeanssp`** . You may use **`setuptools`** or **`distutils`**. The default compilation flags that result by using this file are acceptable.

SUPPLEMENTARY MATERIAL

TESTER FILE

Attached to this assignment is an executable file **"kmpp.so"**. Copy it to Nova and give it execution permissions (see the command `chmod` above).
Its execution API is:

```
>>python -m kmpp.so K N d MAX_ITER filename file_to_compare
```

Where 'file_to_compare' would be your program's output that was directed into a file.

You can use the tester to check your output using the following procedure (assuming K = 3, N = 5, d = 2, MAX_ITER = 100, filename = "input.txt" and that you have the *.so file of your C extension):

1. Execute your code and output the result into a file (in this case named "y") using this command:
   ```
   >>python3.8.5 kmeans_pp.py 3 5 2 100 input.txt > y
   ```
2. Use the following command to check your output:
   ```
   >>python -m kmpp.so 3 5 2 100 input.txt y
   ```

Important note: to compare the final centroids, use your results against the tester's with converging sets (those that do not reach the MAX_ITER number of iterations).

INPUT FILE GENERATOR

The supplied executable file **"gen_file.so"** from HW1 can be used for generating the observations file (see the explanation in the previous assignment).

OPTIONAL BONUS

This section covers an optional bonus (of 10 points) and has no tester file to check against.
It will be submitted as a Colab notebook named **bonus.ipynb**.
This program will only produce an output and will not require any input.

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.

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

$\textbf{inertia} = \sum_{i=1}^{N} \|x_i - \mu_{x_i}\|^2$ 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.

Annotate (as explained in class) the chosen k on the plot, where the 'elbow' is.

ASSUMPTIONS AND REQUIREMENTS

Note the following instructions for this assignment:

- Keep all the previous, relevant requirements from HW1 regarding the command-line arguments and the K-means algorithm.
- Hold the requirements as presented throughout this document.
- Follow the Moodle forum dedicated for this assignment for clarifications and updates.
- The module `sklearn` can only be used in the **bonus.ipynb** file.
- It is your responsibility to ensure that your code is compiled on **Nova** and produces the shared object (*.so of the C extension).
- Any further questions should be answered by using the supplied tester and observing its behavior.

SUBMISSION

You may submit this assignment alone or in pairs. Please submit a zip file named **id1_id2_hw2.zip** replacing *id1* and *id2* with the actual **9-digit** ID of both partners. Only a **single** partner should submit the assignment! If you submit alone, name it **id_hw2.zip**.

The zipped file must contain the following files (at the root, **no folders**, all lowercase):

- **kmeans.c** – Your source code for the C extension program.
- **tasks.py** – Your invoke tasks file.
- **setup.py** – Your setup file.
- **kmeans_pp.py** – Your source code for the Python program.
- **bonus.ipynb (optional submission) –** Your bonus assignment file, if you choose to submit it.

Submit **only these files!**

MAC users: see the instruction under section “SUBMISSION” of HW1.

REMARKS

- Please post questions you have about this assignment in the assignment forum (it will be opened later on).
- The bonus section gives an additional 10 points, but the maximum score of the assignment remains 100.
- Late submissions are not acceptable unless approved by the TA.
- Borrowing from others' work is unacceptable and bears severe consequences.

GOOD LUCK