BVA 501E Assignment 2

Due 25 December 2023

Submission Guidelines:

- Please submit a single python code file (your submission should have .py extension). You can use the template given to you with this assignment.
- You can work on your assignment and submit it through Ninova until the midnight of the due date.
- Any questions regarding the assignment and late submissions can be directed to mergun@itu.edu.tr.
- In case you cannot submit until midnight, you can submit you homework through email between 00:00 - 00:15 on December 26, 2023. Any email submissions before or after this period will be ignored.
- Please put "[BVA 501E] Assignment 2 Submission" (without quotes) to the subject of your email.
- **Use of any library, including numpy and pandas, is prohibited for this assignment.**
- This assignment is meant to be completed **individually**. Please **do not** share your codes with anyone for any reason.
- Enter your full name and student number into the relevant functions comes with the code template as directed.
- **All codes you submit must be in the designated function**. Writing code outside of the function scope will create errors when your file is imported.

Your task:

For this assignment you will be coding function that implements a basic K-Means clustering algorithm. In a nutshell, a clustering algorithm aims to identify underlying groups within a data-set. In other words, we use clustering methods when we want to divide our data into groups that have similar data instances. K-Means algorithm selects k random centers for clusters and updates these center points at each iteration. A pseudocode for the K-Means algorithm is given below:

1. **Pre-step 1:** An appropriate k value is chosen. This value k specifies the number of groups to be identified in the data-set. This value is **specified by the user** and will be given as a **parameter**.
2. **Pre-step 2:** Randomly select k instances from the data-set to be the initial cluster center points.
3. **Repeat** until maximum number of iterations is reached:
   - a- **Assignment Step:** Assign each instance to the closest cluster center.
   - b- **Update Step:** Calculate new cluster centers by calculating the means of the instances assigned to that cluster.
4. **Return final clusters:** After the iterations are finished we, hopefully, will have the reasonable partition over our data.

In order to implement this logic, you need to have instances at least as many as the number of clusters (k). Before starting the k-means implementation, check if this rule holds. If not **throw an exception**.

When choosing random initial centers, make sure not to choose same instance more than once (Use sampling without replacement). For this task, use functions from NumPy's random package. If a random seed is given, set NumPy's random seed with it.

Additional Files:

1. **assignment2_template.py**: Write your code using this template. **DO NOT** change the function names or add new parameters to the functions. You may add extra functions if needed.

Inputs to your function:

1. **df_data**: A pandas dataframe that contains whole data. Columns correspond to individual features and rows correspond to data instances. **Required.**
2. **n_clusters**: An integer parameter that specifies the number of clusters. Corresponds to the k value in K-Means. **Required.**
3. **n_iter**: An integer parameter that specifies the number of iterations. Assignment and update steps need to be run as many as this parameter. Default value is 100. **Optional.**
4. **random_seed**: An integer value that needs to be used as the random seed if specified. **Optional.**

Outputs of your function:

1. If the number of instances is not enough (i.e. less than n_clusters) then throw an exception
2. If the number of instances is enough, continue with the clustering and return an updated pandas dataframe with an additional column named ”Cluster”. The column cluster should have the final cluster number (start from 0 and move upwards for enumerating the clusters)

For testing your function: Use **scikit-learn** library's **make_blobs** function to create artificial clustering data. This function will give you corresponding cluster labels as well. Compare the results of your function with the given cluster labels.

Note: For certain reasons, your function may return a sub-optimal solution and may not match actual labels. If that is the case, rerun with setting a different random seed.