Machine Learning

Guidelines and Submission Instructions:

1. This project is worth 30% of your overall module grade. You should produce a **.py file** containing all your code and a project **document** detailing your findings.

2. Upload your report document and your solution python files (**.py files**) as a single .zip file before **23:59 on Sunday Oct 22nd**.

   Please note that Jupyter notebooks are also accepted instead of .py files. However, if you are using Jupyter notebooks then you must still submit the report document separately as specified in the instructions below.

3. Go to the "Assignment 1" unit on Canvas to upload your file(s).

4. It is your responsibility to make sure you upload the correct files.

5. Please make sure you **fully comment your code**. You should clearly explain the operation of important lines of code.

6. Please note that marks are awarded for code that is efficient with minimum duplication.

7. You should use **NumPy** where possible throughout the assignment. Again, marks are awarded for the use of NumPy where possible. Aside from NumPy you should be using core Python to solve the problems listed below. The use of any high-level toolkit such as Scikit-Learn or high level functions are **not permitted** (except where otherwise stated). Marks are also awarded for efficient code.

8. Late submissions will be penalized.

   If you submit the assignment after the deadline but within 7 days, a full 10% will be deducted from your final grade. For example, if you submit 5 days after the deadline and obtain a grade of 74%, a full 10% points will be deducted, and the final grade is 64%.

   If you submit the assignment more than 7 days after the deadline but within14 days, a 20% penalty will be deducted.

   A grade of 0% will be given to any assignment submitted more than 14 days after the assignment deadline.

9. The data for this assignment can be found in Data.zip in the “Assignment 1” unit on Canvas. Please download and unzip this file.

10. There is a zero-tolerance policy with regards plagiarism. Software is used to detect any plagiarism that may be present in either your submitted document or in your code. CIT policy covering academic honestly and plagiarism can be found here.

11. A discussion forum will be maintained where you can ask assignment related questions. It is very important that you **do not share any code** or refer in any way to the **methodology you are using for solving the problems** outlined below. If you are not fully clear on what is being asked in any part of the questions below you can look for clarification by submitting your question to the discussion forum.

Part1. KMeans Clustering Algorithm (30 Marks)

The objective is to explore the development of a KMeans algorithm. In the dataset folder you will find a csv file containing just feature data (no associated target). There are 6 features (columns) in the dataset and there are 800,000 instances(rows). Your submission for this assignment will be a written report (submitted as a pdf) and a Python .py file. Please make sure that you follow exactly the instructions below and take particular note of what is to be included in your report. A failure to include the required content in your report will result in a loss of marks. Ssubmissions that contain just Jupyter notebooks without an accompanying report containing the required content (as specified below) will lose marks.

The objective of this task is to develop a random restart KMeans clustering algorithm using Python and NumPy. As already mentioned, you should only use NumPy and Python in your code (no high level API such as Scikit Learn).

Your solution should contain the following functions:

1. A function called *create_centroids* that will take in as arguments the feature data and the number of centroids (an integer which we will call k). This will select k feature instances (k rows of the dataset) and return these as the initial centroids (the k centroids should be returned as a single array).

   When selecting your centroids your code should promote wide dispersion of the centroids in feature space. This does not have to be a KMeans++ initialization strategy, you can implement your own strategy. Please include your implementation of *create_centroids* in your report and a paragraph in your report explaining the code in your own words and how you have attempted to promote dispersion.

2. A function called *calculate_distance*. The primary objective of this function is to calculate the distances between a single query point in feature space and a collection of other data points in the same feature space (your training instances). The distance formula you should use is the Manhattan distance metric.

The function *calculate_distance* should accept as arguments: (i) A single 2D NumPy array containing all the feature data and (ii) A single 1D NumPy array, containing a single query instance. Please note that the use of 'for loops' in this function will significantly slow down the algorithm. Using NumPy it is possible to implement this function without for loops.

Also, you should not implement the distance function using any high-level functions. You should only be using basic NumPy operations such as summation, subtraction etc (you should not be using higher level functions such as numpy.linalg.norm).

The function should return a NumPy array containing the calculated distances from the query point to each of the individual feature instances.

Please include the finished function in your report accompanied by a short explanation of the code in your own words.

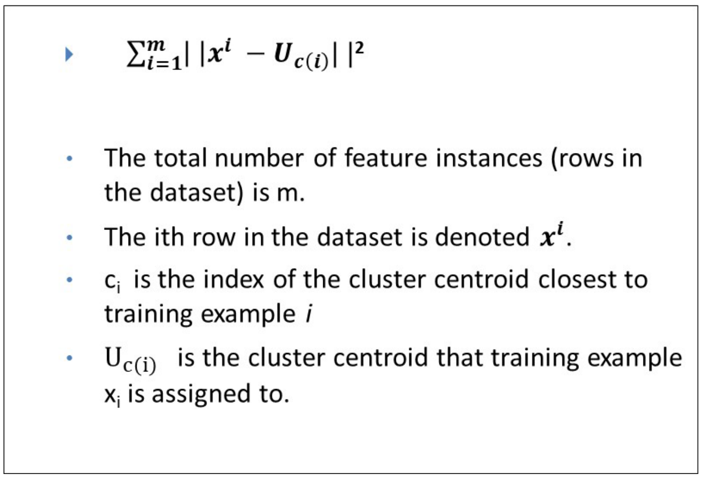
3. A function called *assign_centroids*, which will take in as arguments the feature data and the current array of centroids. This function should calculate the distance between each centroid and all feature data instances (each of the rows in the dataset). Clearly this should take advantage of the code that you wrote in part 2. The function should return the centroid index that is closest to each individual feature instance.

   In other words, if we have 3 centroids and 10 feature instances (10 rows in our data set) then this function calculates the distance between each of the 3 centroids and the 10 rows in the dataset. It should return an array containing 10 integer index values. The first integer value in the array indicates the index of the centroid to which the first feature instance (first row in dataset) is closest. The second integer value in the array indicates the index of the centroid to which the second feature instance is closest and so on.

   So in the example above, if *assign_centroids* was to return the following array [**2**, **0**, **1**, 1, 2, 0, 0, 1, 1, 0] this would mean that the first feature instance is closest to the 3rd centroid (index 2), the second instance is closest to the 1st centroid (index 0), the third instance is closest to the 2nd centroid (index 1) and so on. Please include the finished function in your report accompanied by a short explanation of the code in your own words.

4. A function called *move_centroids*, which will take in as arguments the feature data, an array containing the centroid indices assigned to each feature instance (this is the output of *assign_centroids*) and the current set of centroids. This function will compute the new position of the centroids (by calculating the mean of the datapoints assigned to each specific centroid) and will return an array containing the new centroids. Please include the finished function in your report accompanied by a short explanation of explaining the code in your own words.

   A function called *distortion_cost* which will take in as arguments the feature data, an array containing the centroid indices assigned to each feature instance (this is the output of *assign_centroids*) and the current array of centroids. It should calculate and return the current distortion cost function. The distortion cost function can be calculated as shown below. Please include the finished function in your report accompanied by a short explanation of the code in your own words.



$$\sum_{i=1}^{m} ||\boldsymbol{x}^{i} - \boldsymbol{U}_{c(i)}||^2$$

- The total number of feature instances (rows in the dataset) is m.
- The ith row in the dataset is denoted $\boldsymbol{x}^{i}$.
- c_i is the index of the cluster centroid closest to training example *i*
- $U_{c(i)}$ is the cluster centroid that training example x_i is assigned to.

5. A function called *restart_KMeans*, which will take in as arguments the feature data, the number of centroids (an integer), the number of iterations (an integer) and the number of restarts (an integer).

   The number of iterations just specifies the number of iterations of the inner loop of KMeans (consisting of assigning centroids and moving centroids). This can be set to 10 for this problem (for this data your algorithm will likely converge quickly).

   The number of restarts is the number of times you will restart KMeans from scratch (generate new random centroids and iterate again through the assign centroid and move centroid steps). Again, we will set this value to 10 for our problem.

   The *restart_KMeans* function will use the functions described above to implement a random restart KMeans algorithm and will return the best solution found over the 10 restarts. More specifically it will return:

   - The array of centroid IDs that produced the best distortion cost function value (this defines the cluster to which each feature instance belongs).
   - The corresponding best distortion cost function value

   Please include the finished function in your report accompanied by a short explanation of the code in your own words.

Marks will be given for an efficient implementation of the above functions that minimizes duplication and uses NumPy where possible. Once you have fully implemented the above code, describe what you consider to be the most appropriate number of centroids (clusters) for this dataset.