

Practical Machine Learning



Guidelines and Submission Instructions:

1. This project is worth 50% 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 **23:00 on Sunday Nov 15th**. The name of the each .py file should clearly show if it is related to Part 1 or 2 below.

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 as specified below.

3. Go to the “Assignment 1” unit on Canvas to upload your file.
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. 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).
8. Late submissions will be penalized.

If you submit the assignment after the deadline but within 7 days, 10% will be deducted from your final grade.

If you submit the assignment more than 7 days after the deadline but within 14 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. Once the file is unzipped you will see it contains two folders.

The first is called regression and the second is called clustering. The regression dataset contains a training set and a test set. The data in the clustering folder just contains a feature dataset. The regression data will be used for Part 1 and the clustering data will be used for Part 2 below.

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 honesty 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.

The marks for this assignment will be distributed as follows:

- Part 1: k-NN Algorithm for Regression **(50%)**
- Part 2: KMeans Clustering **(50%)**

Part 1 – Distance Weighted Nearest Neighbour Algorithm for Regression (50 Marks)

You should use the training and test file from the **regression folder** in data.zip for Part 1.

In the regression folder you will find a training and test csv file. There are 12 features in the dataset. There are 6400 instances in the training set and 1600 instances in the test dataset. The target regression value is the final column in each dataset (13th column).

Part 1(A) – Development of Distance Weighted k Nearest Neighbour Algorithm for Regression (25 Marks)

Your objective is to build (from the ground up) a distance weighted k-NN regression algorithm. The k-NN algorithm you develop should:

- Use Euclidean distance for calculating the distance between query instances and the instances in the training data.
- Use the inverse distance as the weight.
- Assess the performance of your kNN regression algorithm using R^2 as your evaluation metric.

$$R^2 = 1 - \frac{\text{sum of squared residuals}}{\text{total sum of squares}}$$

Where

$$\text{sum of squared residuals} = \sum_{i=0}^m (f(x^i) - y^i)^2$$
$$\text{total sum of squares} = \sum_{i=0}^m (\bar{y} - y^i)^2$$

As already mentioned you **should not be using imported functionality** from Scikit Learn (or similar packages) for implementing the kNN (or the R^2 metric) but should instead implement it in core Python or NumPy.

As part of the code for your k-NN implementation you should include the following:

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

$$\begin{aligned} d(\mathbf{p}, \mathbf{q}) &= d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} \\ &= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}. \end{aligned}$$

The function *calculate_distances* should accept as arguments a single NumPy 2D array containing all the feature training data as well as a 1D NumPy array, containing a single query instance (please note that no 'for loops' should be used in this function). Also, you should not implement the Euclidean distance 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 distance from the query point to each of the individual training data instances.

2. A function called *predict* that will accept as arguments a NumPy 2D array containing all the feature training data as well as a single 1D NumPy array, containing a query instance. The function should return the predicted regression value for the query instance (This function will need to call the *calculate_distances* function. You may also may find [`np.argsort`](#) useful when coding the *predict* function).
3. A function called *calculate_r2*, which will take in two NumPy arrays. The first array will contain the true target regression value for all test instances and the second array will contain the predicted regression values (predicted by your model) for each test instance. The function should calculate the R^2 performance of your model.

Include a copy of the above 3 functions in your report document. Once you have completed your implementation include the R^2 value achieved by the model for $k=3$ into your report document.

Checklist for Part 1(A)

Your report should contain the following:

- The final R^2 performance of kNN for $K=3$
- Your implementation of the function *calculate_distances*
- Your implementation of the function *predict*
- Your implementation of the function *calculate_r2*

Your submitted zip file should contain the full code entitled Part1A.py (If you are using Jupyter notebooks that you name your notebook PartA).

Please note a failure to include the above material in your written report (or a failure to include these functions as named and described above in your code) will incur a penalty.

Part 1(B) – Parameters/Techniques Impact Model Performance (25 Marks)

“By default, a k-NN algorithm will weigh the contribution of each feature equally when using standard Euclidean distance”. In your report document clearly explain your understanding of this statement (why this is the case and how it could negatively impact the performance of your k-NN model).

Research and present a range of possible methods for tackling this issue in your report. Incorporate one of these methods into your code and assess the impact.

Please take note of the following two points:

- To score well in this section you should expand your research beyond the techniques presented in the lecture notes and support your answer with references where possible. You should clearly explain all techniques presented.
- Incorporate one of these methods in your code and present the results. Please note **it is acceptable to use a high-level toolkit for this section**.

Aside from the above there are a range of other techniques/parameters that you can investigate that could potentially impact the performance of k-NN regression algorithm. Describe the set of techniques/parameters and why they may potentially impact the performance of a kNN.

Select two of these technique/parameters and comprehensively investigate their impact on the performance of your model.

Checklist for Part 1(B)

Your report should contain the following:

- Your understanding of the statement *“By default, a k-NN algorithm will weigh the contribution of each feature equally when using standard Euclidean distance”*
- A detailed account of the research you undertook into a range of possible methods for addressing this problem.
- A detailed description and assessment of the impact of incorporating one of these researched techniques.
- List and describe the techniques/parameters that can potentially impact the performance of a kNN.
- Assessment of two selected techniques/parameters on your model performance.

Your submitted zip file should contain the full code entitled Part1B.py (Please note if you are using Jupyter notebooks that you name your notebook Part1B).

Part 2 – Development of KMeans Clustering Algorithm (50 Marks)

You should use the feature dataset from the clustering folder in data.zip for Part 2.

In the cluster folder you will find a csv file containing just feature data. There are 6 features in the dataset. There are 10000 instances.

Part 2(A) – Development of a KMeans Clustering Algorithm (30 Marks)

The objective of this section 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.

The KMeans algorithm you develop should reuse the *calculateDistances* function from Part 1. It will be used for calculating the Euclidean distance between all training instances and a single centroid in feature space.

Your solution should contain the following functions:

1. A function called *generate_centroids* that will take in as arguments the feature data and the number of centroids (an integer which we will call k). The function will randomly 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).
2. 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). It 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 will then 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.

3. 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.

4. A function called *calculate_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 below.

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

- The total number of feature instances (rows in the dataset) is m .
- The i th row in the dataset is denoted \mathbf{x}^i .
- c_i is the index of the cluster centroid closest to training example i
- $\mathbf{U}_{c(i)}$ is the cluster centroid that training example \mathbf{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 could 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 function described above (1 - 4) 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

As usual marks will be given for an efficient implementation that minimizes duplication and uses NumPy where possible.

Once you have fully implemented the above code, you should use it to produce an elbow plot and describe what you consider to be the most appropriate number of centroids (clusters) for this dataset. Clearly explain your interpretation of the plot and how you selected the appropriate number of clusters (centroids).

Checklist for Part 2(A)

Your report should contain the following:

- Your implementation of the function *generate_centroids*
- Your implementation of the function *assign_centroids*
- Your implementation of the function *move_centroids*
- Your implementation of the function *calculate_cost*
- Your implementation of the function *restart_KMeans*
- A picture of the elbow plot you produced and your written interpretation of the elbow plot and your justification for the selection of the appropriate number of clusters (centroids)

Your submitted zip file should contain the full code entitled Part2A.py (Please note if you are using Jupyter notebooks that you name your notebook PartB).

Please note a failure to include the above material in your written report (or a failure to include these functions as named and described above in your code) will incur a penalty.

Part 2(B) – Researching Alternatives to KMeans (20 Marks)

When dealing with large amount of data, the KMeans algorithm can be quite slow. The number of necessary distance calculations is $(n*c*a)$ where n is the number of feature instances (rows in our dataset), c is the number of centroids and a is the number of iterations (of course this does not consider the issue of random restarts).

A number of alternative strategies have been proposed in literature to improve the performance of the KMeans category of clustering algorithm. A commonly used alternative to KMeans, which is particularly useful when dealing with large amounts of data is called Mini-Batch KMeans.

Research Mini-Batch KMeans and include a detailed description of its operation in your report document. To grade well in this section, you should demonstrate that you clearly understand the operation of this algorithm and demonstrate evidence of research (maximum 2 pages).