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 **20:00 on Wednesday Oct 30th**. The name of the each .py file should clearly show if it is related to Part 1, 2 or 3 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.

8. 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 is not permitted for this assignment.

9. 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 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.

10. 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 in unzipped you will see it contains two folders.

    The first is called classification and the second is called regression. Both folders contain a training set and a test set. The data in the classification folder will be used for Part 1 and Part 2 below The data in the regression folder will be used for part 3.

The marks for this assignment will be distributed as follows:

- Part 1: k-NN Algorithm **(40%)**
- Part 2: Investigating k-NN variants and hyper-parameters **(35%)**
- Part 3: Implementation of k-NN for Regression **(25%)**

Part 1 – Development of Basic Nearest Neighbour Algorithm (40 Marks)

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

The training set contains 4000 training instances, while the test set contains 1000 test instances. Each instance is defined by a feature vector containing 10 feature values. The first 10 columns in both the training and test dataset correspond to the 10 features. All features are continuous valued features (there are no categorical features)

The 11th column, in the test and training dataset, contains the class. You will notice that this is a multi-class classification problem. There are three classes (1, 2 or 3).

The objective of part 1 of this assignment is to build a k-Nearest Neighbour (k-NN) algorithm (you should initially set k=1). Your algorithm should take as input a training and test dataset and will predict the appropriate class (1, 2 or 3).

The k-NN algorithm you develop for part 1 should use standard Euclidean distance for calculating the distance between query instances and the instances in the training data. It should report the overall accuracy of the algorithm as the percentage of test (query) instances classified correctly.

As part of the code for your k-NN you should include a function called *calculateDistances*. The primary objective of this function is to calculate the Euclidean distances between a single query point and a collection of other data points in feature space (your training instances).

The distance formula you should use is the standard Euclidean distance

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

The function calculateDistances 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.

Using the standard Euclidean distance formula as outlined above, the function should calculate the distance between the query point and all data points in the NumPy array.

The function should return a NumPy array containing the distance from the query point to each of the individual training data instances. It should also return a second NumPy array containing the indices that would sort the distances array. (Note: You may find the method argsort useful).

Part 2 – Investigating k-NN variants and hyper-parameters (35 Marks)

You should use the training and test file from the **classification folder** in data.zip for Part 2.

The objective of this section is to investigate the performance of a k-NN and a distance weighted variant and to write a report documenting your findings.

(a) You should implement a distance-weighted variant of the k-NN algorithm you developed in part 1. In your report include the performance achieved by the distance-weighted k-NN for k=10.

[10 Marks]

(b) There are a range of different techniques you can investigate that could potentially improve the performance of either the distance-weighted k-NN and the basic k-NN from part 1. For example, you could look at the performance of these algorithms for different hyper-parameter settings (such as the value of k).

You should describe and investigate a comprehensive range of techniques that could potentially improve the accuracy of your basic k-NN and distance-weighted k-NN. Your report should document fully the different techniques, provide a justification for selecting and investigating these techniques and present the resulting accuracy.
Please note that when you incorporate additional techniques you should implement these techniques in your own code using core Python or NumPy (rather than relying on imported functionality or high level functions).

The recommended length of the report section is 4 pages (this is just a recommendation and you will not be penalized if you exceed the 4 pages).

[25 Marks]

Part 3 – Developing k-NN for Regression Problems (25 Marks)

You should use the training and test file from the **regression folder** in data.zip for Part 3.

(i) The final section of this assignment requires you to modify the distance-weighted k-NN algorithm, developed in Part 2, so that it can be used for a regression problem.

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). You should assess the accuracy of your regression-based k-NN using an R^2 metric. As already mentioned you should not be using imported functionality for R2 but should implement it in core Python or NumPy.

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

Where

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

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

(10 Marks)

(ii) 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 why this 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. 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.
- 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 or to implement the technique in your own code without the aid of an external library or toolkit.

(15 Marks)