Assignment 1:

Implementation of K-Nearest Neighbors (K-NN)

Submission deadline: 23:59, Friday, Sept.06, 2019

Assignment description:

In this assignment, you will implement the k-Nearest Neighbors (K-NN) algorithm from scratch, evaluate it using a standard dataset and calculate the efficiency.
The assignment consists of two main parts:
Part 1: Implementation and evaluation of K-NN either in Numpy syntax or basic Python syntax (lists and functions).
Part 2: Evaluate the efficiency of the implemented algorithm based on the execution time.
To do Part 1, you are provided with the descriptions of K-NN algorithm, the dataset (Iris flower dataset) and the implementation guide.

K-Nearest Neighbors algorithm

K-NN is a powerful supervised learning algorithm that can be used in both classification and regression problems. K-NN stores all available instances (training data) and classifies new instances (test data) based on a similarity measure (e.g. A distance function). A new instance is classified by a majority of votes for its neighbor classes, which means the new object is assigned to the most common class among its k nearest neighbors. For regression problems, the prediction value is the average of the values of the new object's k nearest neighbors.

For real-valued input variables, the most popular distance measure is Euclidean distance. For other types of data such as categorical or binary data, Hamming distance can be used. Euclidean is a good distance measure to use if the input variables are similar in type (e.g. all measured widths and heights). Manhattan distance is a good measure to use if the input variables are not similar in type (such as age, gender, height, etc.) More information about K-NN can be found in this link: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Iris dataset

The Iris dataset is comprised of 150 instances of Iris flowers from three different species (classes), where each class refers to a type of Iris plant. There are 4 attributes of given flowers: sepal length, sepal width, petal length and petal width,

all in the same unit of centimeters. The predicted attribute is the species, which is one of Setosa, Versicolor and Virginica.

For convenience, in the given dataset which is formatted as a CSV file, the names of four attributes have been removed from the first row and the values of species (target attribute) have been converted to numeric values: Setosa:0, Versicolor:1 and virginica:2. The first four columns represent the values of four features and the fifth column represent the class label ; target attribute.

You will be using this dataset to evaluate your algorithm implementation on classifying the new flowers (test dataset). You should shuffle and split the data into training and test datasets and use the results to compute the accuracy of your algorithm. Good classification accuracy on this problem is above 90% correct, typically 96% or better. (The dataset is available in the same folder containing this pdf.)

Part 1: The Implementation Guide

1. Load data: Open the dataset with CSV format.

You can either import CSV package and use *with open()* syntax or import *pandas* package and use *read_csv()* syntax to open the file.

Shuffle and split the dataset into train/test datasets. (Recommended percentages to split: 66% train, 34 % test). Suppose the dataset is divided into training and test set. Each can be sliced into two parts: X_train: contains all features from the training set, y_train: contains the class label (flower species) of the training set and similarly X_test: contains all features of the testset and y_test: contains the class label of the testset.

2. Calculate the distance

Calculate the distance between a new instance or test object (one row from X_test) and all other instances in the training dataset (all rows in X_train). (Hint: as all flower measurements in Iris dataset are numeric and have the same unit, we can directly use the Euclidean distance measure as a similarity measure.)

Euclidean distance: Suppose we have two instances $x = [x_1, x_2, \dots, x_n]$ and $y = [y_1, y_2, \dots, y_n]$, then the Euclidean distance between two instances is calculated as below:

Euclidean Distance $(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$

3. Find neighbours

Locate *k* most similar data instances (*k*-neighbors) to the test object, based on the computed similarity vector in step 2. (Sort the distance vector in ascending order and choose the first *k* ones).

4. Predict the class of the new instance

Assign the most common label of the *k* nearest neighbors to the class label of the test example.

5. Repeat steps 2,3 and 4 to produce predictions for all instances in the test dataset.

6. Evaluate the accuracy of predictions

One easy way is to calculate the classification accuracy. Classification accuracy is a ratio of the total correct predictions out of all predictions made. (Hint: Write a function which sums up the total correct predictions, divide it by total predictions and returns the accuracy as a percentage of correct classifications.)

Important notes:

- Selection of *k* is important. Small *k* is sensitive to noise points. Very large *k* may include majority votes from other classes. For this assignment k=5 is recommended, but you are free to test other values to see how the results change.

- It is recommended to write separate functions for each step and call them in a Main function with an appropriate order to produce the final results.

- Profile your code to provide an answer for Part 2: Use %*timeit* magic command to check the execution time of any statement. Use *%prun* magic command to profile the entire block.

Part 2: Evaluation of the efficiency

- Use the results of profiling to calculate the performance of the K-NN algorithm implemented in Part 1 based on total and partial execution times; total for all functions, partial for each statement and function.
- Provide the comparison results with numbers resulted from profiling and write two or three sentences explaining them.

Grading

- The result of grading will be either 'Passed' or 'Failed'. Your grade will be assessed on both functionality and style.
- Functionality will be determined entirely by the correctness of your program in Part 1 and the evaluation results in Part 2. (Whether the algorithm is implemented correctly or not and the way you interpret the results in Part 2)

Assignment Submission:

- Deadline: 23:59, Friday, Sept. 06, 2019 (submit your assignment through Canvas.)

- Source code submitted for the assignment should be your own code. If you use the codes from the internet or if you use someone's code without referencing, that will be treated as plagiarism and you will fail the assignment.
- You should **NOT** use Scikit-Learn or any other available machine learning libraries/ packages for implementing the core functionality of the assignment. We encourage you to think critically about writing clean Python code.
- Source code should be written in a Jupyter notebook file. Filename could be the student's first name, last name and name of the assignment. (For example: John.Williams.K-NN)
- Your codes should be documented in the same Jupyter file with Markdown language (LaTeX framework is optional). This documentation is required to clarify the purpose and functionality of your codes.
- No separated report is needed for the assignment, the clear and proper documentation mentioned above is considered as a report.
- The assignment is individual and can NOT be solved in groups.
- Start early. The deadline for submission is soon and there will be no extension.
- If you have any questions, please send me an email to this address: aida.mehdipourpirbazari@uis.no