K-Nearest Neighbors Classification

In the machine learning world, k-nearest neighbors is a type of non-parametric supervised machine learning algorithm that is used for both classification and regression tasks. For classification, the principle behind k-nearest neighbors is to find k training samples that are closest in distance to a new sample in the test dataset, and then make a prediction based on those samples.

These k closest neighbors are used to try and predict the correct discrete class for a given test sample. This prediction is typically done by a simple majority vote of the k nearest neighbors of each test sample; in other words, the test sample is assigned the data class which has the most representatives within the k nearest neighbors of the sample. An alternative method for prediction is to weigh the neighbors such that the nearer neighbors contribute more to the fit than do the neighbors that are further away. For this, a common choice is to assign weights proportional to the inverse of the distance from the test sample to the neighbor. The distance can, in general, be any metric measure, but the standard Euclidean distance and Manhattan distance metrics are the most common choices.

k-nearest neighbors is also known as a non-generalizing machine learning method since it simply "remembers" all of its training data as opposed to other methods that update specific coefficients that fit a model to the training data.

What to Do. Your goal in this part is to implement a k-nearest neighbors classifier from scratch. Your GitHub repository contains the skeleton code for two files that will be used to implement the algorithm: `utils.py` and `k_nearest_neighbors.py`.

The `utils.py` file contains helpful utility functions that will be used by the machine learning algorithms. For this part, the only functions you need to concern yourself with are the functions `euclidean_distance` and `manhattan_distance`.

The `k_nearest_neighbors.py` file defines the `KNearestNeighbors` class that we will use to implement the algorithm from scratch. As you can see, the `__init__` function has already been properly implemented for you. This function is run whenever a new `KNearestNeighbors` object is created, and it checks the arguments passed in for the parameters in addition to setting up the class attributes based on those arguments. The attributes for the class itself are described in detail in the skeleton code. When creating the `KNearestNeighbors` object, the following parameters must be specified (or their default values will be used):

- `n_neighbors`: the number of neighbors a sample is compared with when predicting target class values (analogous to the value k in k-nearest neighbors).
- `weights`: represents the weight function used when predicting target class values (can be either `'uniform'` or `'distance'`). Setting the parameter to `'distance'` assigns weights proportional to the inverse of the distance from the test sample to each neighbor.
- `metric`: represents which distance metric is used to calculate distances between samples. There are two options: `'l1'` or `'l2'`, which refer to the Manhattan distance and Euclidean distance respectively.

Between these two files, there are **four functions** that you are required to implement. The four functions currently raise a `NotImplementedError` in order to clearly indicate which functions from the skeleton code you must implement yourself. Comment out or remove the `raise NotImplementedError(...)` lines and implement each function so that they work as described in the documentation. You may assume that the input data features are all numerical features and that the target class values are categorical features. As a reminder from the guidelines, feel free to create other functions as you deem necessary, but the functions defined by the skeleton code itself must work as intended for your final solution, and therefore you cannot change the parameters for these functions. This is required because we may call any of these functions directly when testing your code. The four functions you must implement and their descriptions are as follows:

- `euclidean_distance(x1, x2)` in `utils.py`: computes and returns the Euclidean distance between two vectors.
- `manhattan_distance(x1, x2)` in `utils.py`: computes and returns the Manhattan distance between two vectors.
- `fit(X, y)` in `k_nearest_neighbors.py`: fits the model to the provided data matrix `X` and targets `y`.
- `predict(X)` in `k_nearest_neighbors.py`: predicts class target values for the given test data matrix `X` using the fitted classifier model.

Testing your Implementation. We have provided you with a driver program called `main.py` that allows you to run and test your implementation of the `KNearestNeighbors` class and its associated functions. This driver program will run your implementation multiple times on two different datasets with different arguments set for the class' parameters. Alongside your implementation, the corresponding `scikit-learn` implementation will be run on the same datasets and with the same arguments, allowing you to directly compare the accuracy scores achieved by your program with those achieved by the standard `scikit-learn` implementation.

In order to run the program, you must have the following packages installed on your system: `numpy`, `pandas`, and `scikit-learn`. You should already have `numpy` installed from the previous instructions. To install `pandas`, please see the instructions at https://pandas.pydata.org/docs/getting_started/install.html. To install `scikit-learn`, please see the instructions at https://scikit-learn.org/stable/install.html.

To test your k-nearest neighbors implementation, enter the command shown below on your terminal.

```
python3 main.py knn
```

If you have not implemented one of the required functions (or have forgotten to remove the line that raises a `NotImplementedError`), then the driver program will terminate unsuccessfully. A successful program call, upon finishing, will result in the output shown below.

```
$ python3 main.py knn
Loading the Iris and Digits Datasets...

Splitting the Datasets into Train and Test Sets...

Standardizing the Train and Test Datasets...

Testing K-Nearest Neighbors Classification...
- Iris Dataset Progress:    [==========================================================] 100%
- Digits Dataset Progress:  [==========================================================] 100%

Exporting KNN Results to HTML Files...

Done Testing K-Nearest Neighbors Classification!

Program Finished!  Exiting the Program...
```

You should now see two new HTML files in your project directory. These HTML files contain tables depicting the accuracy scores of both your implementation and the `scikit-learn` implementation for all of the parameters tested. These files, called `knn_iris_results.html` and `knn_digits_results.html`, are accordingly named based on the dataset that was used for testing.

Open both files using an internet browser and you will see that a table was generated from the results of the driver program. In the table, there is a blank row in between each pair of implementations for readability. If you have implemented the four functions correctly, the accuracy scores computed for your implementation and the `scikit-learn` implementation should be very close to each other for all of the cases. Note that the accuracy scores may not be *exactly* the same due to slight differences in the implementations and the stochastic nature of machine learning algorithms.