**CSG2341.3 Assignment 2.1**

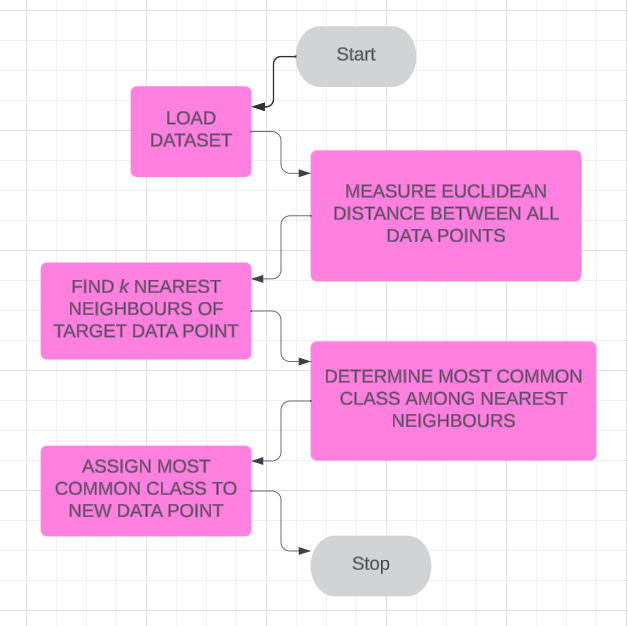
**APPROACH**

The k-Nearest Neighbours algorithm is a simple supervised machine learning algorithm that can be used to make classification or regression predictions, based on the assumption that similar data points are likely to be grouped together in a dataset. The algorithm works by finding the *k* closest data points to a given value, i.e. its *nearest neighbours*, and simply assigning it whichever class represents the majority among these neighbours (IBM, n.d.). For example, if the majority of a data point X’s nearest neighbours are of class A, then a k-NN algorithm will predict that point X is also of class A. In cases where more than two classes are present but none has an outright majority, the algorithm will predict whichever class is represented more than the others, known as the plurality.

To determine the ‘closeness’ of potentially abstract data points, a suitable distance measure for the nature of the data is chosen. Numerical data is typically plotted as Cartesian coordinates, as with the Manhattan Distance metric, which measures the distance between the x and y coordinates of two points along axes at right angles (National Institute of Standards and Technology, 2019). The Hamming Distance metric is used for calculating distance between data points of plain text, comparing the number of similar and different letters between two words of the same length to derive a measure of distance (Analytics Vidhya, 2023). This paper will use Euclidean Distance to determine the nearest neighbours of numerical datapoints. Euclidean Distance can be visualised by plotting points of numerical data as Cartesian coordinates and measuring the length of a line segment joining two given points.

In this paper I will implement the k-NN algorithm to perform classification predictions, as part of a Python program that will accurately identify handwritten numbers provided in a dataset of small grayscale digital images (Alpaydin & Kaynak, 1998). The algorithm will be trained on a subset of already-identified images before being tested on the remaining unidentified images in the dataset. The algorithm results will displayed in a series of figures, outlining the accuracy of its classification predictions and how the algorithm can be optimised with different values of *k*.

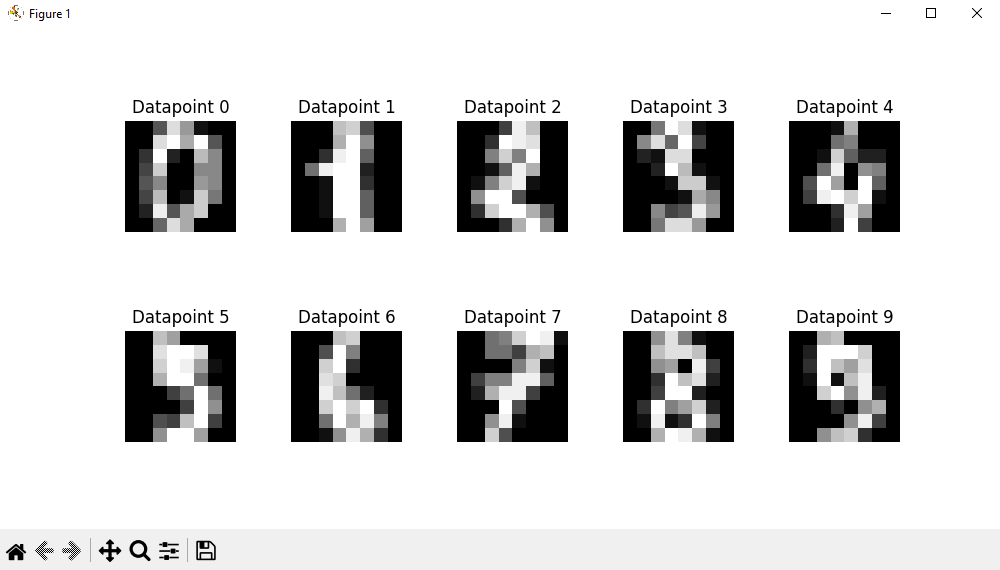
**IMPLEMENTATION**

****

**Figure 1 – A flowchart representing the steps of the k-NN algorithm**

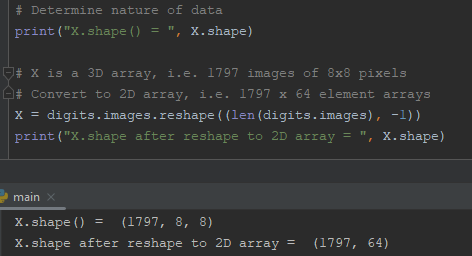
**Load and display dataset**

This implementation of the k-NN machine learning algorithm will be trained to predict images of handwritten digits from 0-9, provided by the MNIST dataset (LeCun, et al. 1998). These digits are depicted as 8x8 pixel, grayscale images. To better understand the nature of the data, we first display the first ten images of the dataset using functions from the matplotlib library (https://matplotlib.org/):



**Feature Extraction**

The dataset comes in the form of a 3D array, visualised as 1797 images that are each 8 rows of 8 pixels. The k-NN algorithm, however, is designed to work with data in two dimensions, so during pre-processing a step is taken to flatten the images into a single row of 64 pixels. Thus each pixel in each data point can be represented as a numerical value representing a grayscale shade between black and white, allowing for an appropriate distance metric (in this case Euclidean Distance) to calculate the distance between individual data points:

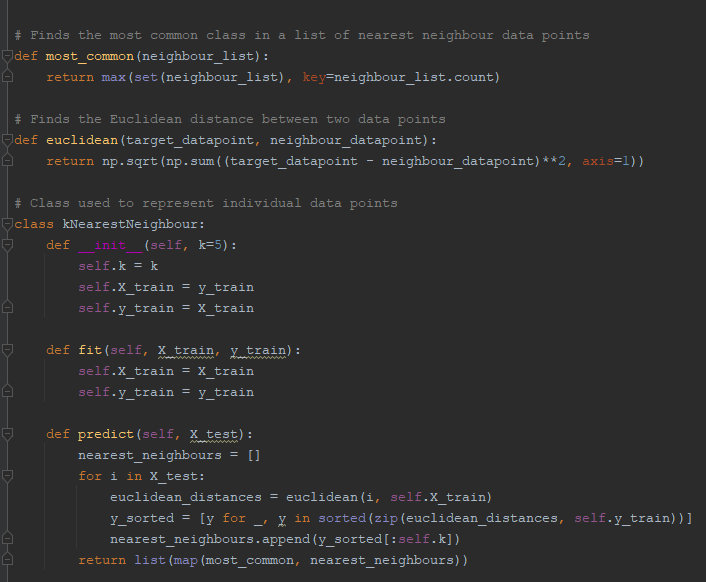


**Pre-processing**

The data is split into training and testing subsets at a ratio of four training images to one testing image using the scikit-learn tool’s built-in test\_train\_split() function (https://scikit-learn.org).:



The algorithm is implemented as a class, with the value of *k* and the distance metric as properties. Member functions of the kNearestNeighbour class are used to train and test the algorithm with the appropriate datasets generated in the previous pre-processing step. Separate utility functions are used to calculate the most common class of data point among the *k* nearest neighbours and to calculate the Euclidean Distance between data points:



**Training and testing**

The k-NN algorithm requires only two parameters: the value of *k* that determines how many of a data points nearest neighbours should be evaluated to find the most represented class, and a distance metric. For this implementation, Euclidean Distance is used as the sole distance metric. The value of *k* is provided at instantiation of the kNearestNeighbours object. The predict() member function represents the majority of the work performed by the algorithm. Iterating through the list of test data points, the algorithm repeats four steps for each:

* It calculates the Euclidean distances between the data point and every other point, saving these values in a list
* This list is sorted in order of closest distance to the data point
* The list is trimmed to include only the first *k* data points
* This trimmed list is passed to the utility function that calculates the most common class of data point, and this class is returned

This final value represents the classification prediction for every data point within the test set.

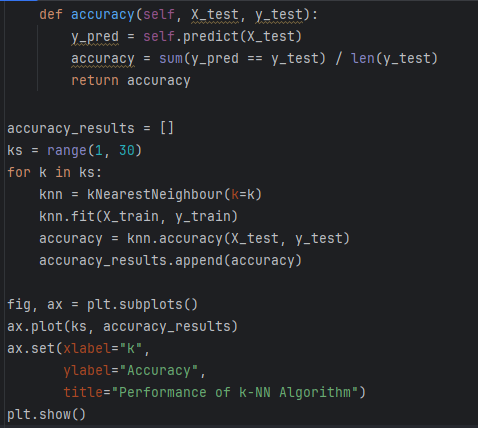
**Performance Evaluation Section (individual):**

**DATASET**

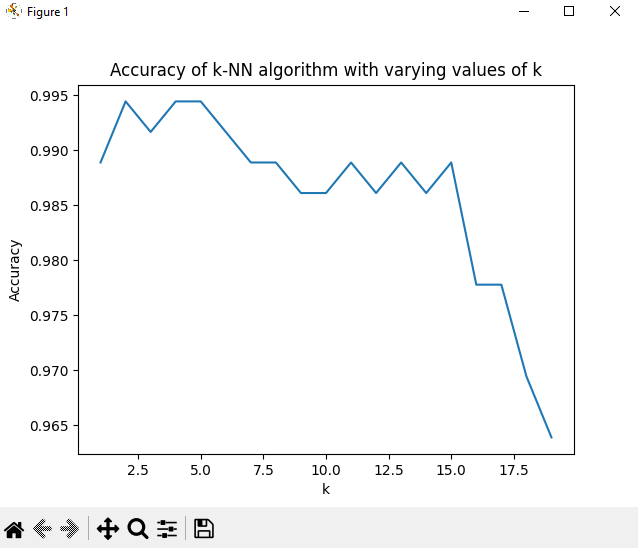
The MNIST, or Modified National Institute of Standards and Technology, dataset, is made up of 70,000 images of handwritten digits (LeCun, et al. 1998). Each is a 28x28 pixel grayscale image of a single number from 0-9. 60,000 images are labelled and used for training the algorithm, with the remaining unlabeled images used for testing.

**FINDINGS**

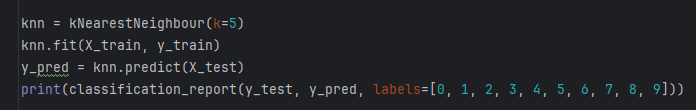
**Optimising parameters**



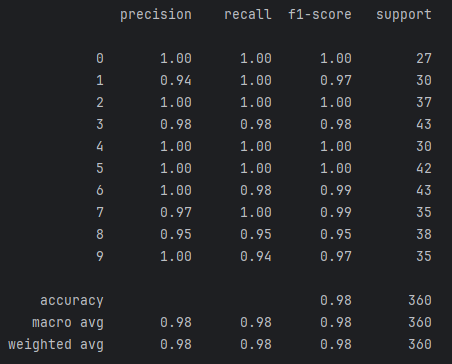
An extra member function is implemented to return a value representing the prediction accuracy of the algorithm. This function is run in a loop, iterating through a range of values of *k* to find the optimum value. The results of this optimisation step are plotted in the figure below, where we can see that the algorithm is most accurate with low values of *k*, with best accuracy achieved when *k* = 5:

****

**Classification Accuracy Report**

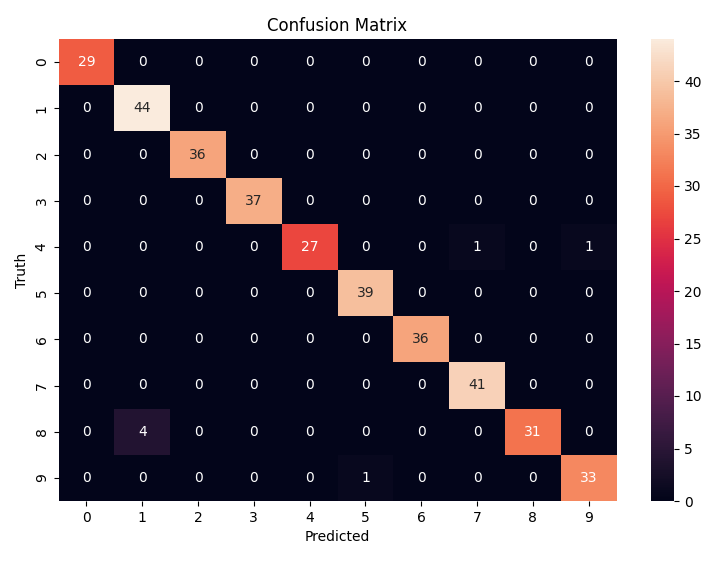
****

Using this optimised *k* value, the algorithm is trained again and re-tested on the MNIST dataset, and the classification\_report() function of the scikit-learn tool is used to generate a report of the algorithm’s prediction accuracy. The k-Nearest Neighbours algorithm implementation achieves an impressive rate of 98% accuracy overall, with near-perfect precision across the digits from 0-9. The only digit shown to cause a significant decrease in prediction accuracy is the number ‘1’, perhaps due to a tendency to confuse its shape with that of another digit.

****

**Confusion Matrix**

Using the seaborn statistical data visualisation tool (<https://seaborn.pydata.org/>) we can plot the results as a confusion matrix, allowing us to identify where the algorithm is making inaccurate predictions. We can see that the most common misclassified digit is ‘1’ as expected, and the confusion matrix shows that in a small percentage of cases the algorithm will identify a number ‘8’ as a ‘1’.



**Performance Comparison Section (joint)**

- Test algorithm with Iris dataset

- Demonstrate findings with tables/figures

**REFERENCES:**

**https://www.ibm.com/topics/knn**

**https://xlinux.nist.gov/dads/HTML/manhattanDistance.html**

[**https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/**](https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/)

**https://ieeexplore.ieee.org/document/726791**