Part 2 – Investigating kNN variants and hyper-parameters

1)Investigate the performance of k-NN variants and parameters and to compose a report documenting your findings.
2)which takes as input a training and test dataset and will predict the appropriate class (Severity - benign or malignant) with a certain degree of accuracy.
3) k-Nearest Neighbour algorithm use standard Euclidean distance, Manhattan, Minkowski and Hamming 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 kNN include a function called **euclidianDistance, manhattandistance, minkowskidistance, hammingdistance**. The primary objective of this function is to calculate the distances between a query point and a collection of other data points in space (training instances).

euclidianDistance = sqrt(sum((x - y)**2))

manhattandistance = sum(|x - y|)

minkowskidistance = sum(|x - y|**p)**(1/p)

hammingdistance = N_unequal(x, y) / N_tot

What is k-Nearest Neighbours

The model for kNN is the entire training dataset(test dataset). When a prediction is required for an unseen data instance, the kNN algorithm will search through the training dataset for the k-most similar instances. The prediction attribute of the most similar instances is summarised and returned as the prediction for the unseen instance.

The similarity measure is dependent on the type of data. For real-valued data, the Euclidean distance can be used. Other other types of data such as categorical or binary data, Hamming distance can be used.

In the case of regression problems, the average of the predicted attribute may be returned. In the case of classification, the most prevalent class may be returned.

Implementation Include the Following Steps:

- **Handle** Data: Open the dataset from CSV , test/train datasets.
- **Similarity**: Calculate the distance between two data instances.
- **Neighbours**: Locate k most similar data instances.
- **Response**: Generate a response from a set of data instances.
- **Accuracy**: Summarise the accuracy of predictions.
- **Main**: Tie it all together.

Python file name - KNNeuclid.py

Algorithm Explanation:

Importing the modules(operator , numpy)

1) HANDLE:

The first thing we need to do is load our data file. The data is in CSV format without a header line or any quotes.
We have declare two data sets and loading data set into the declared data set(cancer_training,cancer_test) using np.genfromtxt inside **main()** function.

2) Similarity:

In order to make predictions we need to calculate the similarity between any two given data instances. This is needed so that we can locate the k most similar data instances in the training dataset for a given member of the test dataset and in turn make a prediction.
One approach is to limit the euclidean distance to a fixed length, ignoring the final dimension.

Defining Euclidian distance:

>initialising the variable distance
>assigning the euclidian formula to distance and return the value (using the np.sqrt Method to calculate square root)

Defining manhattan distance:

> returning the value by applying the manhattan formula that is the sum of all absolute values(absolute value of the subtraction b/w instance1 and instance2

Defining minkowski distance:

> minkowski works based on the value of p if we assign p=1 it will calculate manhattan and if p=2 euclidian distance

Defining hamming distance:
>assert acts like a flag or if condition which take the true value into account if false raises error
>we compare the length of the instances and find the distance between that two strings

3) <u>Neighbours:</u>
One approach is to limit the distance to a fixed length, ignoring the final dimension.

Defining get neighbours:
>initialising distances as list
>calculating the distance between the neighbours using the Distance(any distance method)
>and appending the values of the data(neighbours and distance(dist) into distances[]
>sorting the distances list using sort method from operator module
>appending the distances based on the sort to neighbours

4)<u>Responses:</u>

Once we have located the most similar neighbours for a test instance, the next task is to devise a predicted response based on those neighbours.

We can do this by allowing each neighbour to vote for their class attribute, and take the majority vote as the prediction.

Defining get responses:
>taking the votes count of the neighbours which is inherited from getneighbour function and classifying

4)<u>Accuracy:</u>
To evaluate the accuracy of the model is to calculate a ratio of the total correct predictions out of all predictions made
getAccuracy function sums the total correct predictions and returns the accuracy as a percentage of correct classifications.

5)<u>Main:</u>
We have all the elements of the algorithm and we can tie them together with a main function.

Defining main:
>declaration of two lists to hold test and train data
>loading data set into the declared data set(cancer_training,cancer_test) using np.genfromtxt as a list

>printing the count of the dataset values
>assigning the value of k(to pick neighbours)
>passing on the values to neighbours from getneighbour funtion and their value to result that will be considered as prediction
comparision of predicted and actual values
generating accuracy based in actual and predicted values

Output:

Datasets : CancerDatasets: trainingData2.csv
testData2.csv

Algorithm : KNN

Euclidian distance:

K values	Accuracy
3	81.25%
4	82.03125%
5	79.6875%
6	82.03125%
8	79.6875%
10	76.5625%

manhattan distance:

K values	Accuracy
3	85.15625%
4	87.5%
5	85.9375%
6	89.84375%
8	85.15625%
10	85.15625%

minkowski distance:
p=1 (returns Manhattan)

K values	Accuracy
3	85.15625%
4	87.5%
5	85.9375%
6	89.84375%

p=2 (returns Euclidian)

K values	Accuracy
3	81.25%
4	82.03125%
5	79.6875%
6	82.03125%

hamming distance:

K values	Accuracy
3	98.4375%
4	100.0%
5	100.0%
6	100.0%
8	99.21875%