Intelligent Systems

ASSIGNMENT 2 REPORT PART 1

Himath Ratnayake
S5209861 | GRIFFITH UNIVERSITY 2021

Software Design

Functions:

split_data	
Description	The split data function takes a percentage value as a parameter which signifies the size of the test set. Then the data set is split into a two smaller sets based on the size specified; the training set and the testing set.
Data Types/Structures	As the percentage parameter is given as a decimal, it must then be multiplied by the total amount of data that is present – for this dataset this is 150 points. Amount = len(data) * (percentage value) Then, the random.sample function is used to select "amount" data points from the data and they are assigned to the testing set. The original complete data set is then copied to the "training" list; then, the "testing" list is iterated through, and any items in the testing list are removed from the training list which initially contained all 150 lines of data. By doing so, it ensures that any potential duplicate data points in the dataset are not completely removed – all 150 data points always accounted for through this method. The function then returns training and testing arrays.

euclidean_distance	
Description	One of the distance metrics used for kNN, it takes in two individual data points – 1 from the training array and 1 from the testing array, the returns the Euclidean distance between those two data points.
Data Types/Structures	First, all values in the training and testing arrays are converted to floats except for the last string value defining the iris class as it is not numerical. Then, the total distance between each individual parameter of the training and testing array data is calculated by subtracting the training value from testing value. Note that the last feature (the type of flower which is a string) is not included in this calculation. This value is then squared. Finally, the distance value's square root is found to give the final Euclidean distance and returned.

find_accuracy	
Description	This function that returns the accuracy of the predictions found, as a percentage of the flowers in the test that were correctly identified
Data Types/Structures	The predictions array containing all the predictions generated, as well as the testing array, which is the data we are trying to predict for, is passed into the find_accuracy function. Then, the testing array is iterated through, and checked to see if the last value in an item in the testing array (which is the iris type) is the same as the value found in the predictions array. If it is, a counter for the number of correct predictions is incremented. Finally, once finished iterating, the counter is divided by the length of the testing array and multiplied by 100 to return an accuracy value in percentage format.

kNN	
Description	This function is responsible for finding the nearest neighbors, given the full training data array, a k value and a single test point.
Data Types/Structures	The Euclidean distances are compared between the testpoint passed in and each index of data in the training array. The calculated distances are stored in a list, and the distance value found for each data test point is then sorted using the operator command in ascending order. This means that the shortest distance data points will appear first in the distance array now. Then, another list known as neighbors is initialized and the "k" closest distance values to the testpoint are appended, which will be the first "k" values of the now sorted distances array. The neighbors list is then returned.

classification	
Description	This function is for classifying the type of flower the test point is based on the majority of k nearest neighbors found. It takes the neighbors array returned in the kNN function as a parameter.
Data Types/Structures	A "classes" dictionary is initialized, which will hold the values of different flower types. The neighbors array is iterated through, and for each response (plant type) found, it is checked whether this plant type is already in the classes dictionary. If it is, the value for the plant type in the dictionary is incremented. If it's not seen before, the new plant type's value is initialized as 1. The dictionary is then sorted by the value of the items, and the plant types with the highest value will be equal to the most prevalent type of flower found within the K Nearest Neighbors. This flower type is then returned.

Run_KNN	
Description	This function is the main driver function for the kNN algorithm. It takes in training, testing, k value, and then returns the accuracy of the algorithm. It is also slightly modified such that it accepts the mean and standard deviation points for the data points too, which will be needed for the alternate distance function.
Data Types/Structures	Then, the data set passed in as the training array is iterated through and the k-Nearest Neighbor algorithm process occurs: • For each test point in the testing array (testing[i]), their nearest "k" amount of neighbors are found with the kNN function • Then, the neighbors are sent into the classification function to find the most common flower type amongst the neighbors. • This result is added into a predictions list. • For the current sample class, the prediction is then printed. • If the prediction is the same iris type as that of the current test point, then a correct prediction was found. Otherwise, it is false. Finally, the testing accuracy is found using the find_accuracy function and is returned to the main function.

Plot_graph	
Description	This is the main function used to create a graph of k vs accuracy %.
Data Types/Structures	In this graph, the value of k is iterated through from 1 to the length of the training data + 1, and for each k value the kNN algorithm is performed. The accuracy found for each k value is then appended to an accuracies array, and a new array known as kvals is created which will have the corresponding k value for the calculated accuracy. Then, both the kvals array and the accuracies array are converted to numpy arrays with np.array, so that the plt.plot function can be called to successfully plot the array and output a graph.

main	
Description	Main function which executes the kNN algorithm, and calls the other functions to plot the k vs accuracy graphs
Data Types/Structures	K and the split percentage values are initialized, and then the iris.csv file is opened. The first line is skipped, as it does not contain data necessary for the analysis, and then each line after is appended to a “data” list. The training and testing arrays are then initialized by passing the data and split percentage into the split_data function. The run_kNN function is then called. For generating the graphs, the k value was iterated a number of times up to the training set length, and the accuracy found at each k value was recorded. This was then used to plot a graph of k vs % accuracy using the matplotlib python library and plot_graph function.

Find_stats	
Description	This function is a helper function for the normalized Euclidean distance function elaborated upon in the results section.
Data Types/Structures	For each of the training and testing sets, it finds the standard decisions and averages of the data points. It does this by copying both sets of data to new arrays, then calling the np.array function on the data. After this, the numpy library is able to calculate the standard deviation of the data using the np.std() inbuilt function. Both the data average and the standard deviation will be necessary for the normalized Euclidean distance function.

Results

The K-Nearest Neighbours algorithm is a method who's output is based on the majority vote of the "k" number of nearest neighbours to a given test point. When choosing K, a concept known as the bias variance trade-off must be kept in mind.

High bias usually stems from oversimplification of model assumptions. When the model suffers from high bias, the average response of the model is far from the true value – this is known as *underfitting,* meaning the model doesn't predict the training data well. High variance usually stems from overcomplex assumptions. When the model suffers from high variance, this is usually a result of its inability to generalize well beyond the training data – this is known as *overfitting.* The goal is to build a model that achieves a balance between bias and variance so that the combined error of the these two forces is minimal. For this, cross-validation can be used, where the dataset is split into a training and testing set. The model for kNN is trained on the training test and then tested on B.

In the context of kNN, when K is increased, the accuracy is also decreased due to an increase in bias, as can be seen in the figure above – this is because as K is larger, it has to take more neighbors into account and so the model becomes more complex. This trend is an instance of underfitting when the k value increases.

On the other hand, using less neighbors (lower k value) leads to what is known as overfitting. At k = 1, the bias will be smaller, and variance is greater; when the new data is encountered, there is a higher chance at an error stemming from high variance. At low k, the model selects the accuracy on the training data set more than the accuracy on a test data set, so it will have a higher accuracy in the training set but lower for the unseen test set, due to the model not being as generalized.

With all of this is mind, the optimal k value was found to be 8 after testing many different values. Though due to the random sampling the k value could have some variance at k = 8, the accuracy between the two sets (training and testing) was most *consistently* highest; results of this test are illustrated in the graph below.

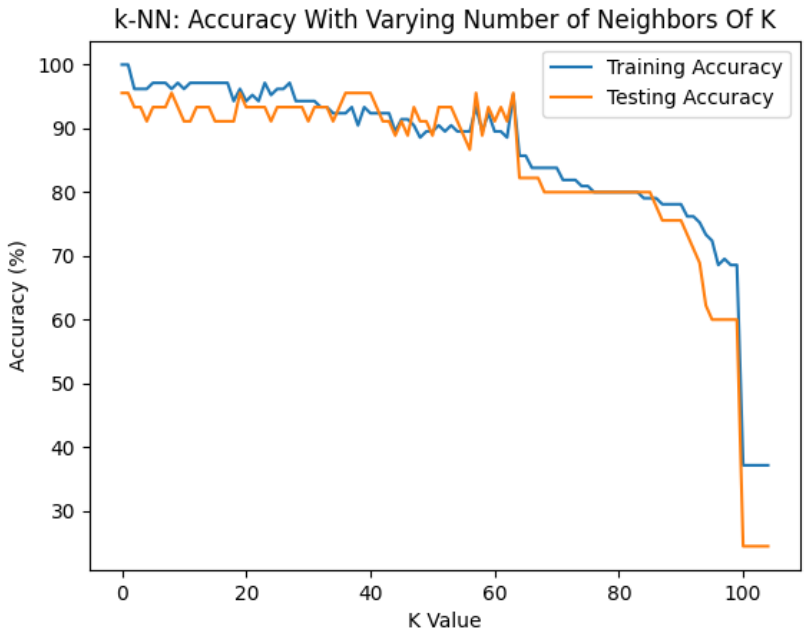

Figure 1: Graph showing the accuracy of the kNN algorithm at varying values of 'k'

It was found that the training set was usually slightly more accurate than the testing set too, which can be owed to the fact that there was a 70/30 split in the tests run, with the training data having 70% of the data points and the training data having 30% of the points. With more training data points, a model better suited to the training data could be created, as opposed to the test data which has less points to compare with.

Distance Function

One problem with the distance algorithm used (Euclidean distance) is that if a specific attribute of the data set has a larger range of values compared to other attributes in the data, then the value with a larger range would numerically dominate and would be "weighted" more. Hence, one modification that can be done to the data is to normalise the values over the same range, such that features and attributes with larger ranges won't have a higher impact on the distance value found. For this, the mean and standard deviation of the training and test sets must be computed first. Then, the normalized Euclidean distance equation is used where the data point is subtracted from the data mean and divided by the data standard deviation.

By using this new distance function, the following results were found for the testing and training data sets. As can be seen from this graph, there is a slightly higher accuracy when using a normalized Euclidean distance as opposed to a standard one over different k values. However, this difference is not huge, as even though each individual iris attribute had a different range of data, the difference in this range was small enough that the impact on overall accuracy was somewhat minimal even before standardization. However, an increase in accuracy was still certainly seen, with the "AlterTraining" accuracy (green line, the alternate normalized distance) having a consistently higher accuracy in both training and testing cases when compared with the original Euclidean distance function.

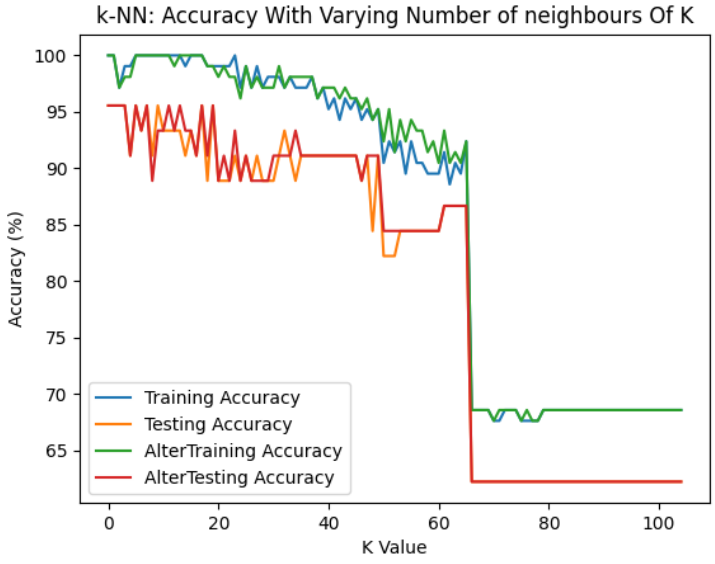

Figure 2: Graph showing the accuracy of the kNN algorithm with both the alternate distance function (green/red lines) and the original function (blue/orange lines)

Conclusion

The k-nearest neighbours function was found to have an optimal k value of 8, which was the k value point where the accuracy between training and testing was able to be most consistently high. There was a general downward trend in accuracy as k increased after a certain point, owing to underfitting which increases model complexity/bias due to more neighbours. Two distance functions were also tested: Euclidean distance and normalised Euclidean distance. There was a slight increase in accuracy after normalising the data and using normalised Euclidean distance, but the actual difference was found to not be much, which may be due to the data set values having minimal variance in ranges. The training set was also found to have a generally more accurate result over multiple tests, which is most likely due to the training set having more values.