Kevin Blanco

Kaushik Kolla

Project Report

### PART 1 - CLASSIFICATION

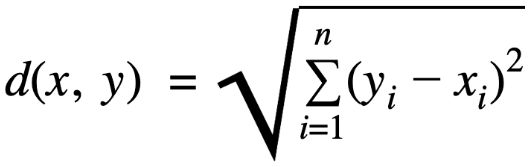
To start off on the classification task, we had to decide between which algorithm to use. We decided to use **K-Nearest Neighbors** to reach this goal as the straightforwardness of the algorithm and the nature of the data (all numeric values and no alphabetical values) meant that the necessary computations would be easier to achieve. Some other reasons that we chose K- Nearest Neighbors were:

* Flexibility to use with non-linear decision boundaries
* No training time, as all work happens during prediction itself
* KNN constantly evolves with new data
* Single hyperparameter
* Choice and flexibility with which distance metric to use

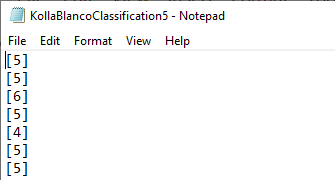
To start our implementation of the K-Nearest Neighbor algorithm, we first had to read in the data files, which had different methods for delimiting (commas, spaces and tabs). Once this was done, the basic idea of the implementation was to - take the test data and go through the training features data line-by-line and calculate the euclidean distance between each feature and aggregate the distance into one measure. Then, the final step was to get the most common label in the K-Nearest data points and return it.

In Depth explanation-

Optional Step- For datasets with missing values, we used the KNN algorithm from Part 2 to fill them in.

1. For each test data point, go through all the Training Features Data row-by-row and calculate the Euclidean Distance:
   1. Euclidean Distance Formula: 
2. Sort the resulting output by distance and return the K-Smallest distances-
   1. dist = np.argsort(point\_distance)[:k]
3. Get the most common label for each input from the K-Nearest Neighbors-
   1. lab = mode(labels)
   2. lab = lab.mode[0]
   3. results.append(lab)

These steps resulted in a prediction file that spits out the predicted labels for each row of test data:



### PART 2 - MISSING VALUE ESTIMATION

To estimate the missing values in the given datasets, we decided to use the KNN imputation method. We first read in the text file that contains the dataset we want to work with. The data from the dataset is inputted into a nested list, where each list within the nested list represents a tuple, and the elements in the list represent the tuple’s attributes. Our method then involves using three different functions to estimate and replace the missing values of the given dataset

The first function “find\_distance”, returns a nested list called “distances”, where each list within the nested list represents a tuple, and each element in the list represents the tuple’s distance to another tuple in the dataset. The function loops through each tuple in the dataset, and compares each one at a time to every other tuple in the dataset to find the distance between each one and itself. When two tuples are being compared, the difference of the two attributes from each tuple at each respective index is found and then that number is squared, if neither of the attributes is a missing value, and all of those results are summed together. That value is then multiplied by the weight, which is equal to the number of total attributes of one tuple, divided by the number of present attributes. The product of those values is then square rooted to find the distance, this value is then inputted into the “distances” list. This repeats until all of the distances for every tuple is calculated.

The second function “find\_missing\_value”, iterates through the dataset and finds which tuples have missing values. It loops through every tuple in the dataset, and at each tuple all of its attributes are then looped through. If the loop comes across an attribute that is equal to "1.00000000000000e+99", then that means that the current tuple contains a missing value. The function “calculate\_missing\_values” is then called, so that the missing values for that tuple can be estimated and then replaced. It keeps going until all tuples have been iterated through, and all tuples that have a missing value are inputted into the “calculate\_missing\_values” function.

The third function “calculate\_missing\_values”, estimates all of the missing values for the inputted tuple, and then inputs them accordingly in the dataset. The list that contains the distances to every other tuple from the inputted tuple is taken, and its k lowest elements are put into another list, which is sorted, called “nearest”. In this implementation k is equal to the square root of the amount of tuples, but the k + 1 lowest distances are put into “nearest” as there will always be a distance of 0, and this represents the distance between the current tuple and itself. The “nearest” list contains the distances between the inputted tuple and its nearest neighbors. This list is then used along with the inputted tuple’s distance list to find the index of each of the nearest neighbors in the “dataset” list. The index for the distance of 0 is of course ignored as stated before, it represents the distance between the tuple and itself. These indexes are placed into the “neighbors” list. The inputted tuple’s attributes are then looped through, and the index of each missing attribute is put into the “missing\_index” list. For each missing index, the attributes of each of the nearest neighbors at the same index are averaged. This value is the estimated value for the inputted tuple at the current missing index, and this value is then placed into the “dataset” list, replacing the missing value that was previously there. If there is a case where the attributes of each of the nearest neighbors at the current missing index are all missing values as well, then a naive approach is taken to find the current missing value. This is done by taking all of the other present attributes of the inputted tuple and finding the average.

The three functions “find\_distance”, “find\_missing\_value”, and “calculate\_missing\_values” all work together to estimate and replace all of the missing values in a dataset. The function “find\_distance” is first called with its input being the “dataset” list that contains all of the data in the given dataset text file. The “distances” list that is returned is then inputted along with the “dataset” list into the “find\_missing\_value” function. The “find\_missing\_value” function calls the “calculate\_missing\_values” function each time a tuple with a missing value is found. Once the functions finish running, the “dataset” list is now updated with the estimated values. The updated “dataset” list is then finally written into a new text file that contains estimated values instead of the missing values that the initial dataset text file that was read in contained.