**K – Nearest Neighbors**

In this experiment, a K-Nearest Neighbor Regression model was tested. This model is based off the idea that points close to each other in all but the to be predicted feature will be similar to each other in the to be predicted feature. This density-based model works by incorporating all the training points into a hyper-plane model, commonly called a feature space. Then for each new testing point, that points Euclidean distance is found to all other training points. From there the tuning value K, determines the number of nearest points (based on the Euclidean distances calculated) wanted. This number of nearest points then have their regression feature averaged and the result of said average assigned to the testing point’s desired class. This approach has several advantages, the first of which is no training time, besides memory allocation and feature scaling (which is already done in the preprocessing for most regression models cases). All of the regression calculations are done at run time and is done by a simple distance algorithm. This unfortunately has the side effect of producing a paradox where the best results from the model will come from a high number of training points, but that will also result in a large calculation time.

The primary tuning variable in a K-Nearest Neighbor Regression model is the value K. For this experiment the number of K values tested were 3,5,7, and 9. The reason only odd number for K values were selected was to prevent a tie in the averaging algorithm to assign a class to tested data point. For the purpose of testing 5 models were generated using different samplings of the dataset and tested. Of those models 4 of them had a best K value of 5 and a single model had a best K value of 7. This was determined by accuracy and Kappa value of tested model. This clustering of best tested K values can then be inferred that the best K value for this given problem is 5. These tested models were further tested through 5-fold cross validation. The best results of these tests can be found in Figure 7.

**Random Forest**

In this experiment, a Random Forest Regression model was tested. This model is based off the Classification And Regression Tree (CART), a type of Decision Tree Machine Learning algorithm. The model determines the which class a testing point goes into by following a decision tree. This decision tree is generated by first finding the feature(s) with the correlation and variance to the desired output. This feature(s) is then split into a number of part where each one points to a given class. This generates a single node and its downward branches to its children nodes. The number of parts or number of downward branches generated, and number features tested at each node is tuning variable of this type of model. These methods were tested and found to be best to leave the tree as a binary decision tree with only two children nodes per parent node, and only one feature tested per node. The method of generating nodes is then repeated at each layer in the decision tree until either the tree hits a confidence threshold, or each feature is used. The method used in this model was the later as it had a small number of features and would not overwhelm the model’s computation time with keeping them all. This way of generating a CART decision tree is useful but take a large amount of time in massive datasets and run the risk of overfitting to the data. The fix to this is the random forest regression model. This model first uses bagging, or sampling with replacement, to generate a subset of the dataset. These sub-datasets are then used to generate a single tree each. The optimal number of trees differ based on the model generated and are selected by the method using out-of-bag error of each model. The number of features can also be better improved by improving the feature relations to the forest. This is done by generating a new feature space, using feature bagging, which is meant to prevent every tree from selecting the features at each level. The forest method stated here is called random forest, and the training of the model is then finished. The testing of the model is done by each tree in the forest taking the testing point and running it through its decision tree to find what class it thinks that point should belong to. The random forest model can then either assign a weight to each tree accuracy to influence the or simply take the most common class and assign that as the testing point’s class. The method used here is the second since this in effect treats every tree generated with equal weights. This method was then used to train five different models each one with 5-fold cross validation, the best result of which can be found result.