Name: Yeung Wai Kit

Student ID: 20608456

Course: COMP4331

Title: Report for Decision Tree Mini Project

Preprocessing

Before constructing the decision tree model, data should be preprocessed.

To prevent unexpected behavior in the decision tree model, I first fill all the null values in the dataset with corresponding attribute mean. Then all columns with identical values are dropped because those columns are meaningless for decision tree splits. The process is then followed by standardization to range (-1,1). This transforms data in each column to data with mean=0 and standard deviation =1 and helps unifying data units. Finally, to ensure model training speed we do PCA to reduce the number of features. I choose to preserve 90% portion of explained variance in this model.

Model creation

My final goal is to build a random forest model. I will first do hyperparameter training to select best parameters for my forest. Then I will prune the DecisionTreeClassifier object with best parameter using cost complexity pruning to avoid overfitting. After all these I should be able to get the best parameters for a non-overfitting tree for my model. Finally, I will rebuild the random forest model using the list of best parameters and pruning parameter.

Random Forest Model

The reason I use random forest model is because it is a more reliable compared to single decision tree model. Random forest model creates different decision trees using different randomly split datasets and use a majority vote method for final prediction. Using the hyperparameter tuning, every tree inside the forest has the best parameters and the model should be performing the best. To create a random forest model I will use RandomForestClassifier from sklearn.ensemble. This creates a random forest directly with parameters similar as a Decision Tree Model.

Hyperparameter Tuning

For this step I will use GridSearchCV class. This class basically provide functions that allows me to loop over different parameters of my input model and select the best performing sets automatically. The looping parameter set I will use is:

{'criterion':['gini','entropy'],'max_depth':[250,270,300],'min_samples_split':[3,4,5],'max_leaf_nodes':[150,170,200],'n_estimators' : [500,1000]}

***The best parameters are

{'criterion': 'entropy', 'max_depth': 270, 'max_leaf_nodes': 170, 'min_samples_split': 3, 'n_estimators': 1000}

for the sake of computational speed I will just use the best parameters to do GrindSearchCV in my python notebook***

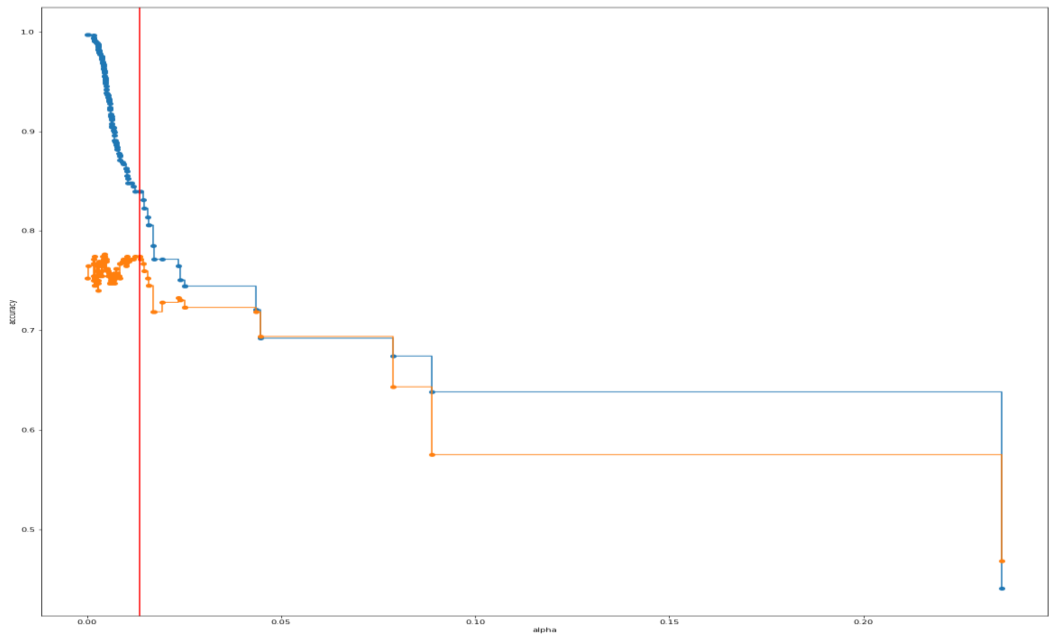
For the parameter cv in GrideSearchCV, I will set cv = 10 to do a 10 fold stratified cross validation on my model.

I will do the search in the function selection(tree,x,y) which inputs the decision tree classifier tree, data x and label data y. This function should return a dictionary of best performing parameters.

Pruning

After selecting the best parameters, I will use it to construct a decision tree model. The problem of overfitting may occur despite the best parameters. To reduce overfitting, I will use cost complexity pruning method to post prune my decision tree model.

Cost complexity pruning method involves a cost complexity pruning parameter, ccp_alpha. By adjusting this parameter, we can get different scores on test sets and training sets. The best value for this parameter is the point where the testing data get the best score while training data gets the lowest score given the highest test score. Details of finding alpha is shown below



From the graph, best cc_alpha value is 0.0135.(This file is also attached as prun_parameter.png)

Now I will explain my procedure in python notebook.

First, I will split the dataset with 80:20 training to test ratio. Then I will use cost_complexity_pruning_path function to find the alphas (cost complexity pruning parameter) and corresponding impurities. Then, similar to hyperparameter tuning, I will use a loop to construct and score different decision tree models with different cost complexity parameters. Then I will plot a graph to see what value of cost complexity parameters gives the best performance in testing set with lowest performance in training set. This cost complexity parameter will be used to construct the random forest model.

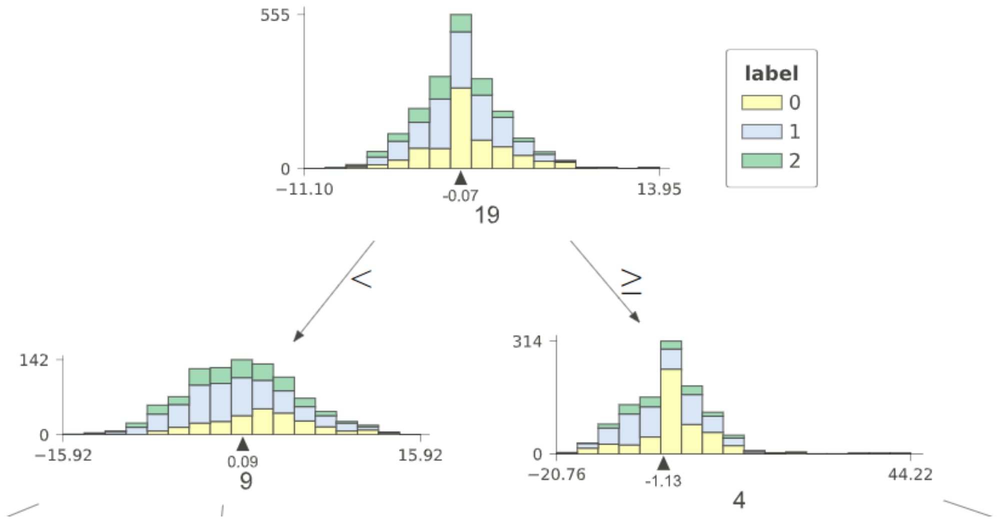
K-Fold Cross Validation

I will use cross_val_score function to do a 10-fold stratified cross validation on my model. The scores should represent the accuracy of my model. As there should be 10 different scores, I will take the average to represent the overall score of my model.

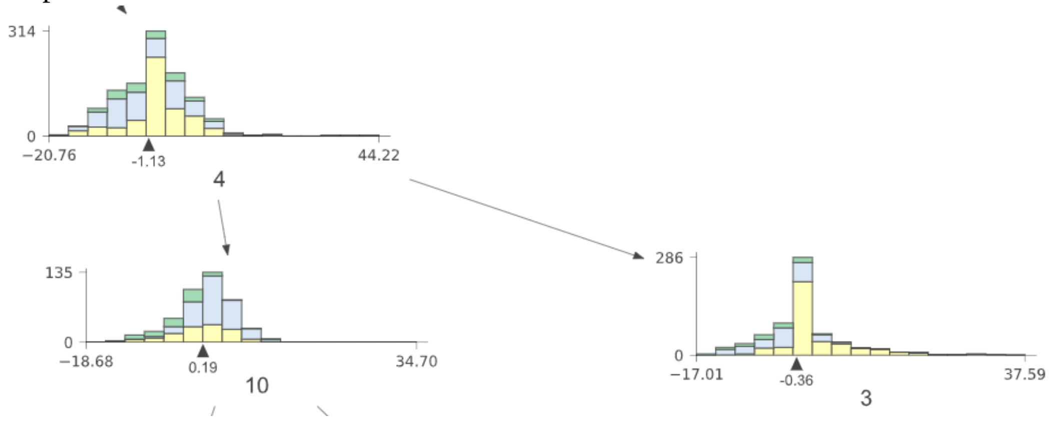
<u>Three splits from my decision tree</u>

As random forest technique is used, I will choose three splits from the first tree in my forest to display.

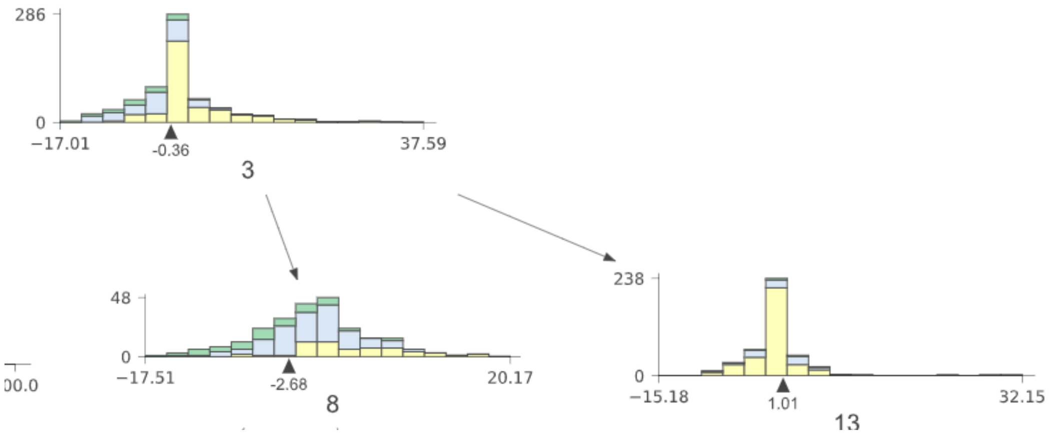
1st split



2nd split

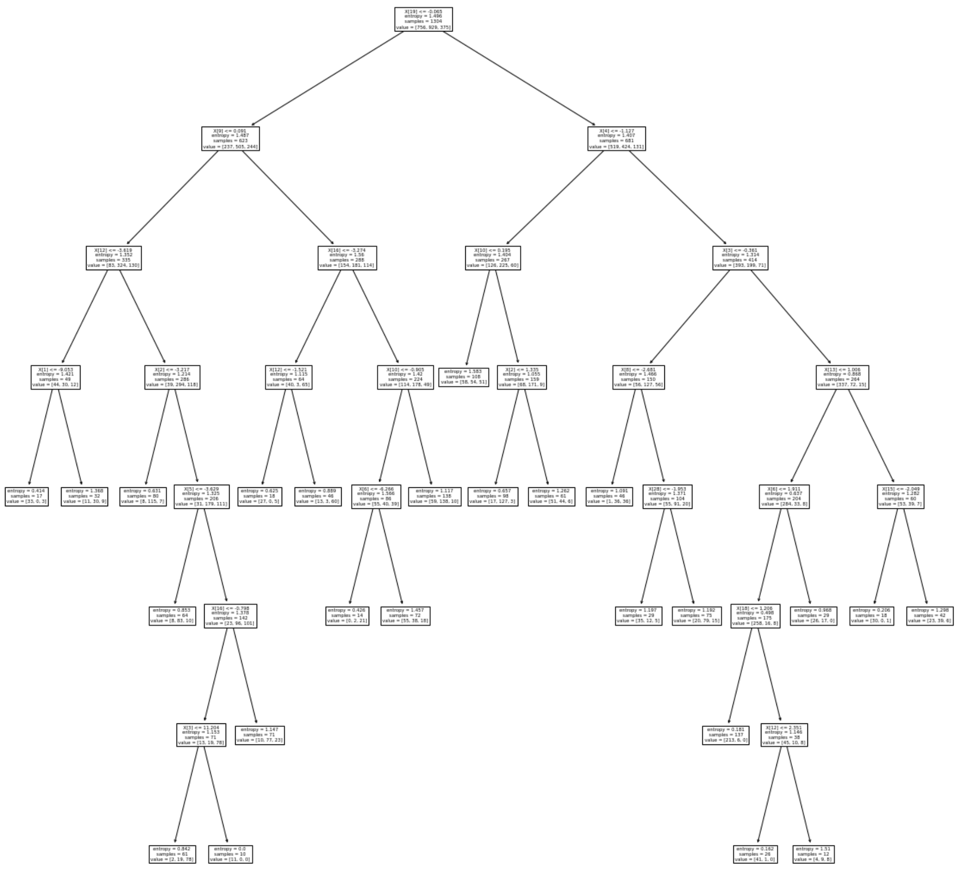


3rd split



Final Model

The final output is also attached as decision_tree.png


X[19] <= -0.065 entropy = 1.496 samples = 1304 value = [756, 929, 375]
X[9] <= 0.091 entropy = 1.487 samples = 623 value = [237, 505, 244]
X[4] <= -1.127 entropy = 1.407 samples = 681 value = [519, 424, 131]
X[12] <= -3.619 entropy = 1.352 samples = 335 value = [83, 324, 130]
X[16] <= -3.274 entropy = 1.56 samples = 288 value = [154, 181, 114]
X[10] <= 0.195 entropy = 1.404 samples = 267 value = [126, 225, 60]
X[3] <= -0.361 entropy = 1.314 samples = 414 value = [393, 199, 71]
X[1] <= -9.053 entropy = 1.421 samples = 49 value = [44, 30, 12]
X[2] <= -3.217 entropy = 1.214 samples = 286 value = [39, 294, 118]
X[12] <= -1.521 entropy = 1.115 samples = 64 value = [40, 3, 65]
X[10] <= -0.905 entropy = 1.42 samples = 224 value = [114, 178, 49]
entropy = 1.583 samples = 108 value = [58, 54, 51]
X[2] <= 1.335 entropy = 1.055 samples = 159 value = [68, 171, 9]
X[8] <= -2.681 entropy = 1.466 samples = 150 value = [56, 127, 56]
X[13] <= 1.006 entropy = 0.868 samples = 264 value = [337, 72, 15]
entropy = 0.414 samples = 17 value = [33, 0, 3]
entropy = 1.368 samples = 32 value = [11, 30, 9]
entropy = 0.631 samples = 80 value = [8, 115, 7]
X[5] <= -3.629 entropy = 1.325 samples = 206 value = [31, 179, 111]
entropy = 0.625 samples = 18 value = [27, 0, 5]
entropy = 0.889 samples = 46 value = [13, 3, 60]
X[6] <= -6.266 entropy = 1.566 samples = 86 value = [55, 40, 39]
entropy = 1.117 samples = 138 value = [59, 138, 10]
entropy = 0.657 samples = 98 value = [17, 127, 3]
entropy = 1.262 samples = 61 value = [51, 44, 6]
entropy = 1.091 samples = 46 value = [1, 36, 36]
X[28] <= -1.953 entropy = 1.371 samples = 104 value = [55, 91, 20]
X[6] <= 1.911 entropy = 0.637 samples = 204 value = [284, 33, 8]
X[15] <= -2.049 entropy = 1.282 samples = 60 value = [53, 39, 7]
entropy = 0.853 samples = 64 value = [8, 83, 10]
X[16] <= -0.798 entropy = 1.378 samples = 142 value = [23, 96, 101]
entropy = 0.426 samples = 14 value = [0, 2, 21]
entropy = 1.457 samples = 72 value = [55, 38, 18]
entropy = 1.197 samples = 29 value = [35, 12, 5]
entropy = 1.192 samples = 75 value = [20, 79, 15]
X[18] <= 1.206 entropy = 0.498 samples = 175 value = [258, 16, 8]
entropy = 0.968 samples = 29 value = [26, 17, 0]
entropy = 0.206 samples = 18 value = [30, 0, 1]
entropy = 1.298 samples = 42 value = [23, 39, 6]
X[3] <= 11.204 entropy = 1.153 samples = 71 value = [13, 19, 78]
entropy = 1.147 samples = 71 value = [10, 77, 23]
entropy = 0.181 samples = 137 value = [213, 6, 0]
X[12] <= 2.351 entropy = 1.146 samples = 38 value = [45, 10, 8]
entropy = 0.842 samples = 61 value = [2, 19, 78]
entropy = 0.0 samples = 10 value = [11, 0, 0]
entropy = 0.162 samples = 26 value = [41, 1, 0]
entropy = 1.51 samples = 12 value = [4, 9, 8]