Supervised Learning Final Project

by Rajdeep Mallick

Github Repo Link: https://github.com/RajdeepMallick21/Supervised_Learning_Final_Project_RajdeepMallick.git

Project Topic

- The quality/ranking of wine will be predicted based on physiochemical components.
- The wine quality will be ranked from 0 to 9
- Appropriate ML models with a combination of hyperparameters will be used to predict on this dataset
- If the accuracy, f1 score, auc score or other metrics provide good enough results, then this prediction model could be useful for wine sellers and customers to pick wines based on specific properties/characteristics.
- The project will use KNN and Decision Tree models to perform Multi-class Classification tasks and predict the ranking of a wine based on its properties

Info about the Data

- This dataset was found on Kaggle.
- It has 13 columns.
- The 12 columns used for features are type, fixed acidity, volatile acidity, citric acidity, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol.
- The 1 column for target variable is called quality and it is the ranking of the wine.
- First 5 rows of the dataset is shown below

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

Info about the Data

- This dataset was found on Kaggle.
- It has 13 columns.
- The 12 columns used for features are type, fixed acidity, volatile acidity, citric acidity, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol.
- The 1 column for target variable is called quality and it is the ranking of the wine.
- First 5 rows of the dataset is shown below

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

Info about the Data

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   type                  6497 non-null   object 
 1   fixed acidity         6487 non-null   float64
 2   volatile acidity      6489 non-null   float64
 3   citric acid           6494 non-null   float64
 4   residual sugar        6495 non-null   float64
 5   chlorides             6495 non-null   float64
 6   free sulfur dioxide   6497 non-null   float64
 7   total sulfur dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6488 non-null   float64
 10  sulphates             6493 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64  
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

- The general shape of the dataset is 6497 rows and 13 columns.
- There are some missing values as can be seen under Non-Null Count column
- These will be cleaned up in the Data Cleaning section
- The type column in the dataset has string object data type.
- This will have to be converted to numerical value in the Data Cleaning section.

Data Cleaning

```
# Calculating the number of missing values
num_of_missing_values = df.isnull().sum()
print(num_of_missing_values)
```

```
type                    0
fixed acidity          10
volatile acidity        8
citric acid             3
residual sugar          2
chlorides               2
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      9
sulphates               4
alcohol                 0
quality                 0
dtype: int64
```

```
# Dropping dataframe rows that have missing values
df = df.dropna()

# Printing shape of the new dataframe
print(df.shape)
```

```
(6463, 13)
```

- The sum of the missing values from every column was calculated here.
- The rows with these missing values will have to be dropped from the dataset.
- After dropping these rows, the final shape of the dataframe is 6463 rows and 13 columns

Data Cleaning

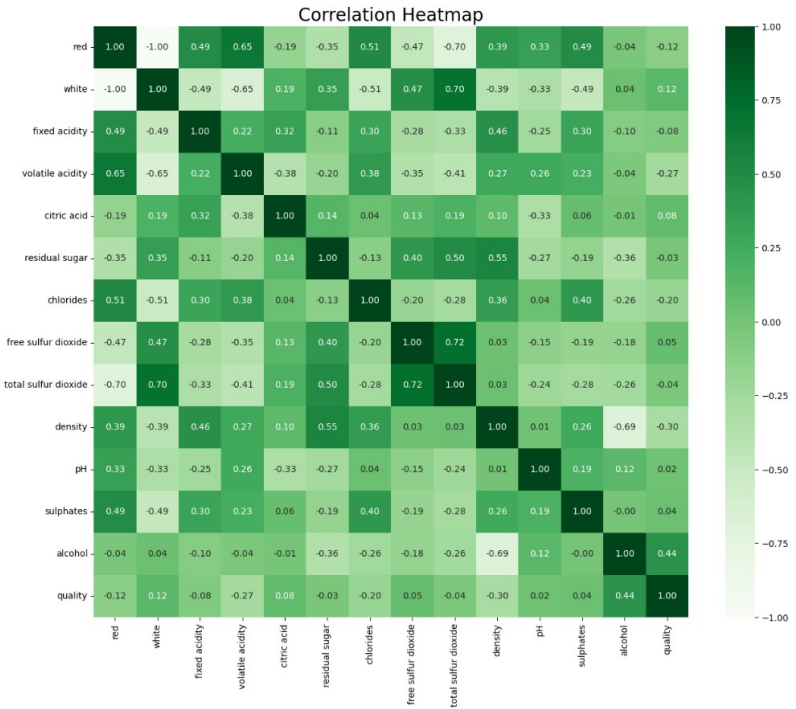
```
# Splitting type column into two columns for white and red
# Converting types of the two new columns from Boolean to int64
df_type = pd.get_dummies(df["type"], dtype='int64')
df = pd.concat((df_type, df), axis=1)
df = df.drop(columns='type')

df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6463 entries, 0 to 6496
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   red                   6463 non-null   int64
 1   white                 6463 non-null   int64
 2   fixed acidity         6463 non-null   float64
 3   volatile acidity      6463 non-null   float64
 4   citric acid           6463 non-null   float64
 5   residual sugar        6463 non-null   float64
 6   chlorides             6463 non-null   float64
 7   free sulfur dioxide   6463 non-null   float64
 8   total sulfur dioxide  6463 non-null   float64
 9   density               6463 non-null   float64
 10  pH                    6463 non-null   float64
 11  sulphates             6463 non-null   float64
 12  alcohol               6463 non-null   float64
 13  quality               6463 non-null   int64
dtypes: float64(11), int64(3)
memory usage: 757.4 KB
```

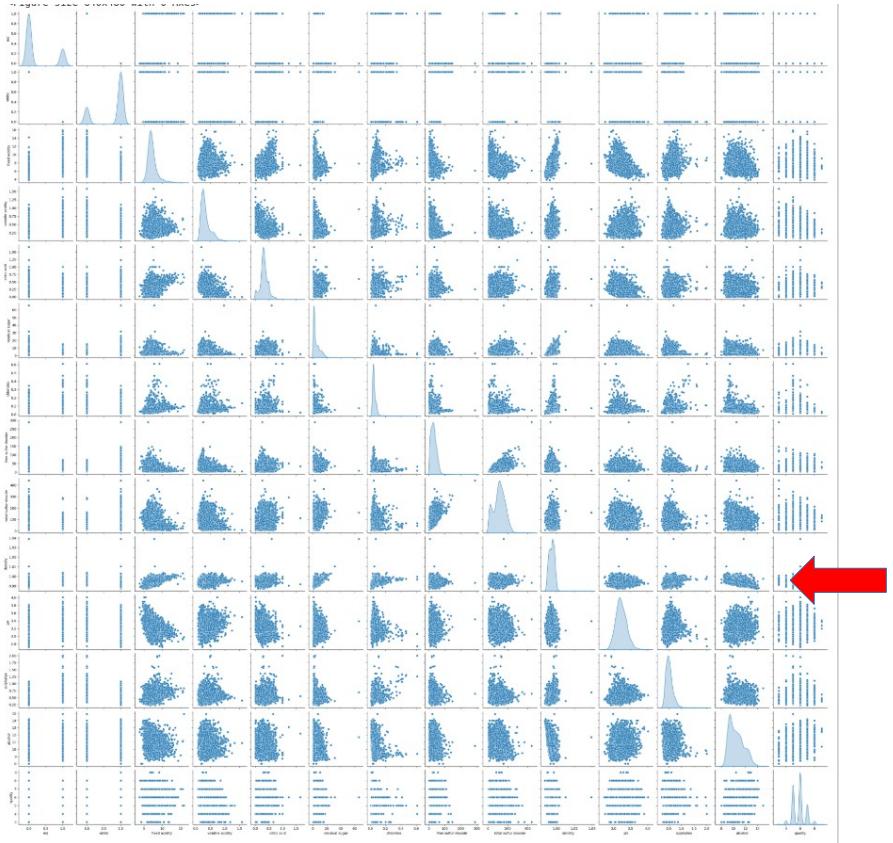
- The type column had a data type of string object which will have to be changed.
- It is a binary non-ordinal categorical variable.
- The values have to be converted to 0 and 1.
- The type column holds string values of ‘red’ and ‘white’.
- Therefore, two indicator variable columns ‘red’ and ‘white’ with binary numerical values 1 and 0 will be created.
- The original type column will be dropped since its no longer necessary.

EDA



- A correlation matrix was created to show the correlation between all columns.
- The target variable quality has highest correlation with alcohol, density, chlorides and volatile acidity columns.
- Density and alcohol had a high correlation of 0.69 which could indicate that they are linearly dependent and might have redundant info.
- They will be checked for collinearity in the pairplot next

EDA



- The scatterplot for density vs alcohol can be next to the red arrow.
- The shape is not very thin or linear which could mean that these two features are not collinear.
- Although, they could still be multicollinear which wouldn't be identified by using pairplots and correlation matrix alone.
- The vif value for each feature will be calculated next to check for multicollinearity.

EDA

```
# Creating vif_df dataframe with just the names of the feature columns
vif_df = pd.DataFrame()
vif_df['features'] = df.columns[:len(df.columns)-1]

# Producing vif values for these feature columns
vif_df['VIF_value'] = [vif(df.values, i) for i in range(len(df.columns) - 1)]
print(vif_df)
```

	features	VIF_value
0	red	5.993218e+05
1	white	1.824207e+06
2	fixed acidity	5.082700e+00
3	volatile acidity	2.287008e+00
4	citric acid	1.621795e+00
5	residual sugar	9.790884e+00
6	chlorides	1.660492e+00
7	free sulfur dioxide	2.252867e+00
8	total sulfur dioxide	4.058575e+00
9	density	2.251996e+01
10	pH	2.572168e+00
11	sulphates	1.576768e+00
12	alcohol	5.752019e+00

- A vif dataframe was created here to show the feature names alongside their vif value.
- In the lectures, any feature with vif of ‘inf’ or very high value was removed from the dataset.
- Since all of the vif values are below 10, we can conclude that there is no multicollinearity between the features.
- Therefore, no feature columns will be further removed from the dataset.

Preparing Data for Training Models

```
# Creating separate dataframes for features and target variable
Y = df['quality'].values
X = df.drop('quality', axis=1).values

# Creating training and validation datasets using train_test_split
train_x, test_x, train_y, test_y = train_test_split(X, Y, test_size=0.25, random_state=0)

# Printing the shape of each dataframe to check the size
print('train_x shape: ', train_x.shape)
print('train_y shape: ', train_y.shape)
print('test_x shape: ', test_x.shape)
print('test_y shape: ', test_y.shape)
```

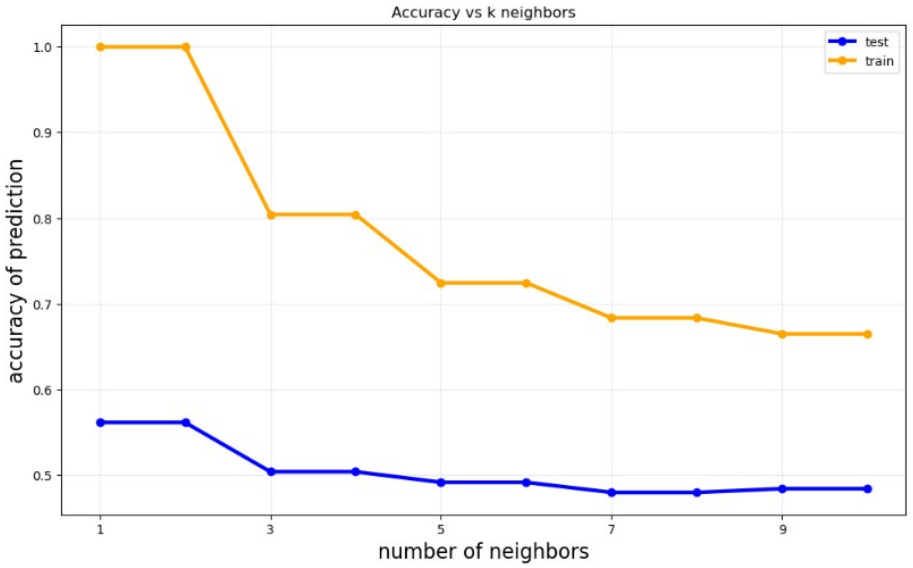
```
train_x shape:  (4847, 13)
train_y shape:  (4847,)
test_x shape:  (1616, 13)
test_y shape:  (1616,)
```

- The columns for features and target variables were split into separate dataframes
- These dataframes were then further split to create training and validation datasets.
- The testing/validation datasets get 25% of the original data.
- The shapes of these dataframes are shown.

KNN Model

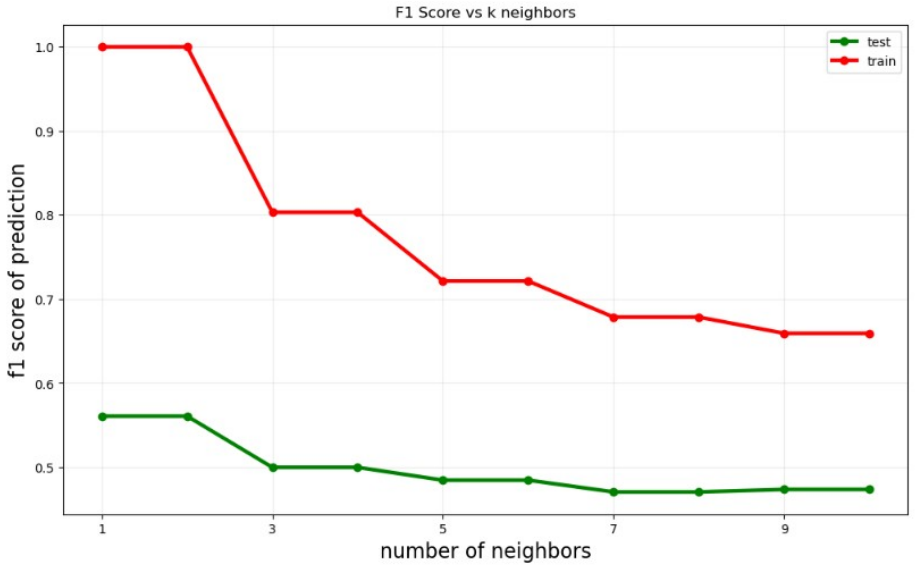
- Since there are only 13 features, the KNN classifier won’t suffer from curse of dimensionality.
- There are an adequate amount of samples for the number of features/dimensions.
- This should keep the accuray of the predictions high for this model.
- BallTree was used to train the knn model with our training datasets and specific k values

KNN Model Results and Analysis



- The accuracy values with k neighbors of 1 to 10 were found for predicting on both the train and validation datasets.
- The highest values for accuracy are at k=1 and k=2 for both.
- Highest accuracy for train dataset is 1.0
- Highest accuracy for test dataset is 0.56
-

KNN Model Results and Analysis



- Since precision and recall usually have a tradeoff, the F1 score was used as it captures both and is more robust.
- F1 score was calculated for every k value from 1 to 10 for both datasets.
- Best values were once again at k=1 and k=2.
- Best F1 score for train dataset was 1.0
- Best F1 score for test dataset was around 0.56

Discussion and Conclusion for KNN Model

- Both the f1 score and the accuracy score curves of our KNN models produced similar results for k values ranging from 1 to 10.
- The highest f1 and accuracy score for our validation dataset was 0.56 at k=2.
- This is not a very high value and shows that the KNN model may need further tweaks or a different model may have to be used for this classification task.
- In the description of the dataset it was mentioned that the labels were very imbalanced. This can be viewed in the confusion matrix for the k=1 KNN model below as the diagonal from top left to bottom right doesnt have too many values in every slot. This means that there are very low values for certain labels especially at the extremities. There are also no values for certain labels which is why the confusion matrix is not the right size. It is a 7x7 matrix when it should be a 10x10 matrix for quality/ranking scores of 0 to 9. The author of the dataset warns about this by saying that there are a lot of normal/average wines than bad or good wines.
- Therefore, the dataset doesn't have enough data to accurately train the KNN model which would explain the poor results/predictions.

```
# Code for creating confusion matrix with k=2 model predicting on validation dataset
confusion_matrix(train_y, KNN(train_x, train_y, K=1).predict(train_x))
```

```
array([[  21,    0,    0,    0,    0,    0,    0],
       [   0,  169,    0,    0,    0,    0,    0],
       [   0,    0, 1592,    0,    0,    0,    0],
       [   0,    0,    0, 2095,    0,    0,    0],
       [   0,    0,    0,    0,  808,    0,    0],
       [   0,    0,    0,    0,    0,  158,    0],
       [   0,    0,    0,    0,    0,    0,    4]])
```

Decision Tree Model

- For the decision tree models, hyperparameter tuning will be performed to produce the best results for accuracy scores.
- The max_depth and min_samples_leaf hyperparameters will be determined using the GridSearchCV function from sklearn. This will also allow for cross validation. The input dataset will get split into chunks and the Decision Tree Classifier model will be fit to each of these chunks separately. The average of the accuracy scores from these fits will be determined. The best_estimator_ attribute will tell us which combination of these hyperparameters yielded the best average accuracy scores.
- Decision Tree models tend to overfit to the data. To prevent this, pruning was performed by testing out different ccp_alphas values over a certain range. This will allow us to decide a value for this hyperparameter.

Results and Analysis for Decision Tree Model

```
# Creating Decision Tree Classifier object
rf = DecisionTreeClassifier()
# Creating Dictionary for hyperparameters max_depth and min_samples_leaf
parameters = {'max_depth':[1,3,6,10],
              'min_samples_leaf':[1,3,6,10]}
# Creating GridSearchCV object to pass in DecisionTreeClassifier and hyperparameters
# This will return the optimal values for both hyperparameters from the range of values given [1, 3, 6, 10]
clf_1 = GridSearchCV(rf, parameters)

# Fits the DecisionTreeClassifier to the training dataset
clf_1.fit(train_x, train_y)
```

```
/home/rajdeep/anaconda3/lib/python3.12/site-packages/sklearn/model_selection/_split.py:776: UserWarning: The
only 4 members, which is less than n_splits=5.
  warnings.warn(
```

GridSearchCV

best_estimator_: DecisionTreeClassifier

DecisionTreeClassifier

DecisionTreeClassifier(max_depth=6, min_samples_leaf=3)

```
# Outputs the best combination for max_depth and min_samples_leaf
clf_1.best_estimator_
```

DecisionTreeClassifier

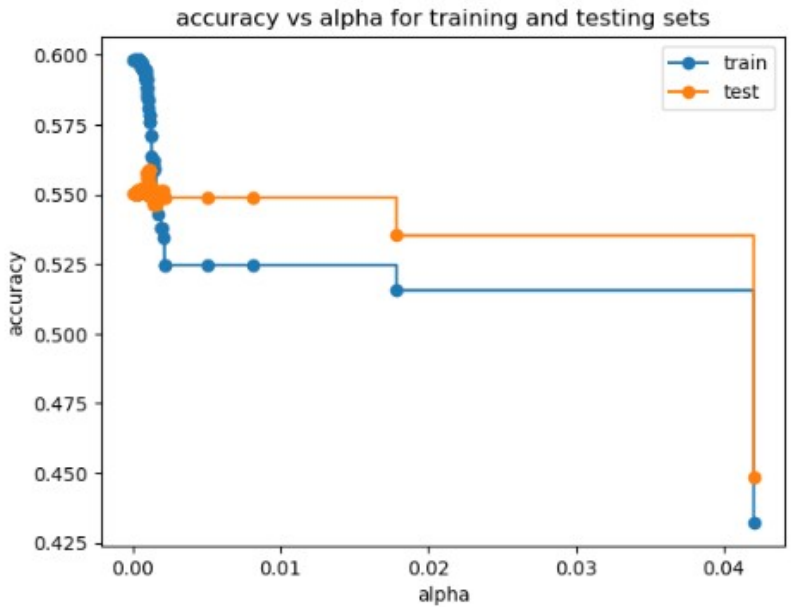
DecisionTreeClassifier(max_depth=6, min_samples_leaf=3)

```
# Outputs the accuracy score created with these hyperparameter values
clf_1.best_score_
```

```
0.539296330577809
```

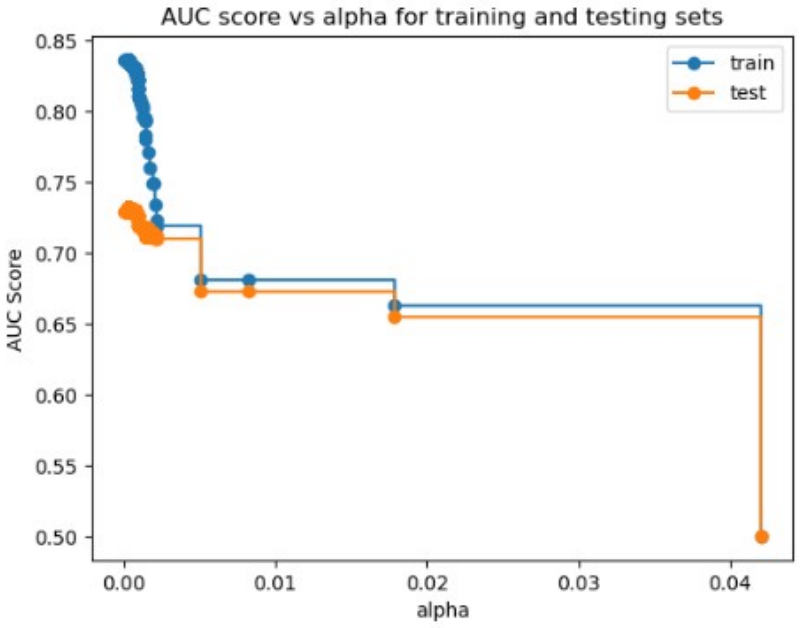
- Hyperparameter optimization using GridSearchCV() resulted in max_depth value of 6 and min_samples_leaf value of 3 as the best values chosen from the range of [1,3,6,10].
- The accuracy for this best estimator was 0.539 on the training dataset.
- This best accuracy score can be further improved when these two hyperparameters are combined with the ccp_alpha parameter which will be optimized next.

Results and Analysis for Decision Tree Model



- The lowest ccp_alphas values produced the highest accuracy.
- The highest accuracy value was 0.5983 for the train datset.
- The ccp_alpha values that produced this highest accuracy value were 0.00015473 and 0.00016505.
- Using these same ccp_alpha values for the test dataset produced accuracy value of 0.550124.
- The best accuracy for the train dataset was therefore improved from 0.539 to 0.598 by adding ccp_alphas value alongside the max_depth and min_samples_leaf value found earlier.

Results and Analysis for Decision Tree Model



- Lowest ccp_alphas values produced the highest AUC score.
- The highest AUC value was 0.8359 for the train datset.
- The ccp_alpha values that produced this highest AUC value were 0.00015473 and 0.00016505 again.
- Using these same ccp_alpha values for the test dataset produced AUC value of 0.729012.

Discussion and Conclusion for Decision Tree Model

- The resulting best accuracy values from this model for the test datasets was 0.55 which is very close to the best accuracy value of 0.56 on the same dataset using k=2 KNN model.
- The best train accuracy score dropped from 1 in the k=2 KNN model to 0.598 in our best Decision Tree model.
- The introduction of the ccp_alpha hyperparameter improved the accuracy performance of the Decision Tree Classifier model compared to just using the max_depth and min_samples_leaf hyperparameters alone.
- The accuracy improved from 0.539 to 0.598 for the train dataset.
- The best ccp_alpha, max_depth and min_samples_leaf values for this model using this dataset are 0.0015473, 6 and 3 respectively.
- The best values for train and test datasets were 0.8359 and 0.729 respectively which shows that there is a high True Positive to False Positive ratio produced by the best Decision Tree Models.

Conclusions Continued

- The failure of two different models to produce accuracy values above 0.56 for the test dataset can also be due to there not being any features in the dataset that can predict the quality/ranking labels well.
- This was suspected when the correlation matrix showed that none of the features had correlation values with the target variable above 0.7.
- It could also be be due to there not being enough samples in the dataset.
- This was indicated by the author of the dataset who mentioned that the labels were very imbalanced as there were not enough samples for very good or very bad wines but a lot of samples for normal/average ranked wines.
- Ensembling methods such as Random forests, AdaBoost and Gradient Boost could improve the accuracy of the Decision Tree model by preventing overfitting.
- Adjustment of more hyperparameters might yield better accuracy results in the future.