MSC 641 Final Project Presentation: Wine Quality Prediction

BY: AJAY, MANJU, ADITYA, JAMES

Introduction

This project aims to develop an analytical classification model to predict the quality of variants of the Portuguese “Vinho Verde” style of red wine. The data comes from the UCI Machine Learning Repository. The classification model will provide insights into the quality of the wine, either 1(good) or 0(bad).

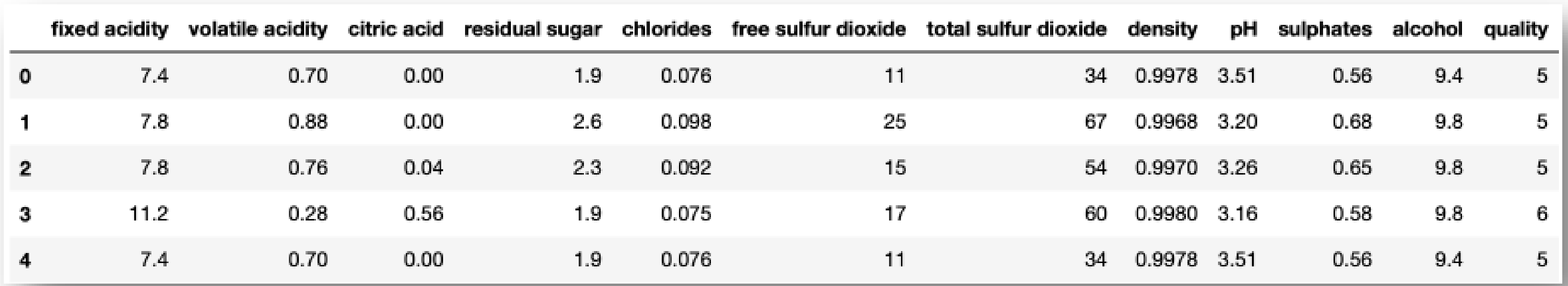
Data Summary

- Source: UCI Machine Learning Repository
- Number of Attributes: 12
- Dependent Variable: Quality
- Analysis Type: Classification
- Number of Missing Values: 0

Bib.
P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis.
Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

Data Summary cont.

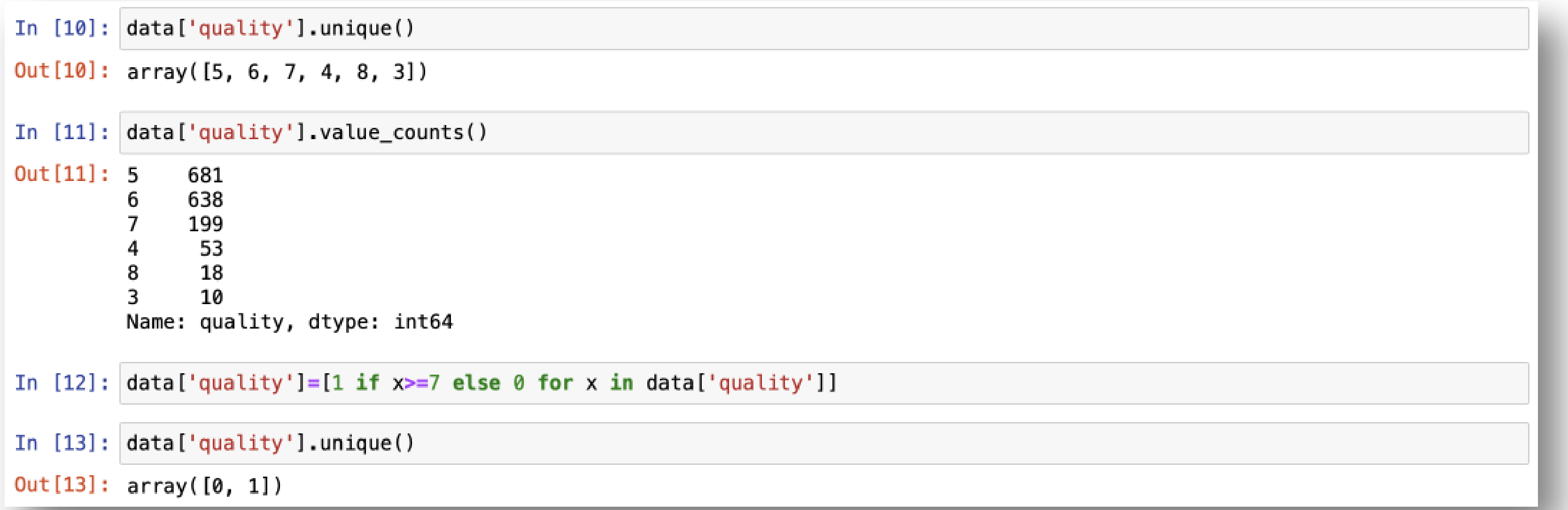
1. fixed acidity
2. volatile acidity
3. citric acid
4. residual sugar
5. chlorides
6. free sulfur dioxide
7. total sulfur dioxide
8. density
9. pH
10. sulphates
11. Alcohol
12. Output variable (based on sensory data): quality (score between 0 and 10)



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25	67	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15	54	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17	60	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5

Converting Target Variable to binary

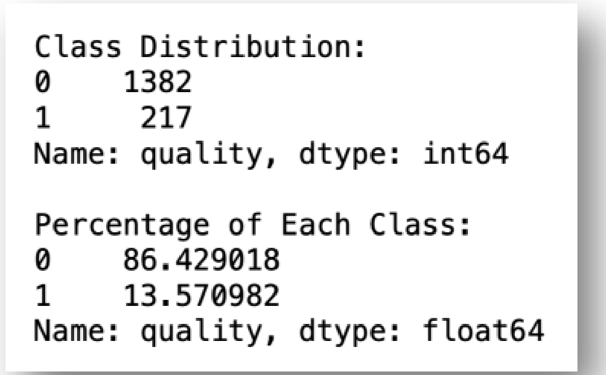
- Binning quality rankings to either 'good' (7, 8) as 1 or 'bad' (3, 4, 5, 6) as 0.
- This new binning is what the models will predict, either 1 or 0.



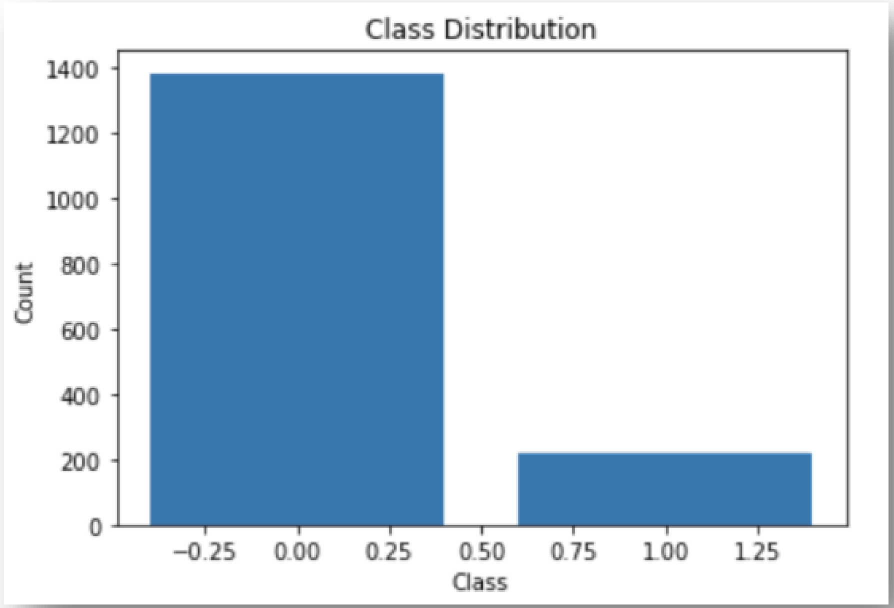
```
In [10]: data['quality'].unique()

Out[10]: array([5, 6, 7, 4, 8, 3])

In [11]: data['quality'].value_counts()

Out[11]: 5    681
         6    638
         7    199
         4     53
         8     18
         3     10
         Name: quality, dtype: int64

In [12]: data['quality']=[1 if x>=7 else 0 for x in data['quality']]

In [13]: data['quality'].unique()

Out[13]: array([0, 1])
```

Dealing with imbalanced data

- **Class Distribution:** Majority of the wine are in the 0 classification. The lower quality wines are underrepresented; This will be addressed with oversampling using **S**ynthetic **M**inority **O**versampling **Te**chnique (SMOTE).

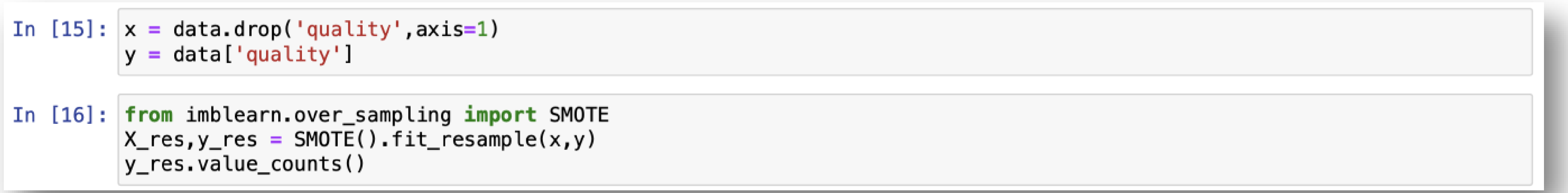


```
Class Distribution:
0    1382
1     217
Name: quality, dtype: int64

Percentage of Each Class:
0    86.429018
1    13.570982
Name: quality, dtype: float64
```



Dealing with imbalanced data cont.

- Using **oversampling** with **SMOTE**, we can resample and balance out the representation of the wine quality categories by randomly duplicating examples in the minority classes.



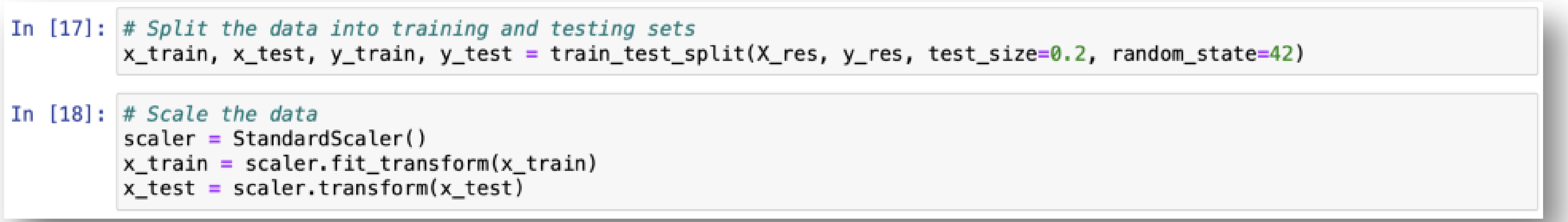
```
In [15]: x = data.drop('quality',axis=1)
         y = data['quality']

In [16]: from imblearn.over_sampling import SMOTE
         X_res,y_res = SMOTE().fit_resample(x,y)
         y_res.value_counts()
```

The results demonstrate that there are now 2218 samples total, with 1109 samples for the negative (quality = 0) and 1109 samples for the positive (quality = 1).

NOTE: The categories are now balanced, this prevents the model from becoming biased towards one class.

Splitting and Scaling the data

- To prepare for model building me must both split and scale the data
  - **Splitting**: For both the dependent and independent variables, we will split them into testing and training sets. The ratio used in this analysis is the 80/20 split, which is fairly common. **Resulting cuts of data:** *x_train, x_test, y_train, y_test*
  - **Scaling**: For any large discrepancies in variables' values, we will use scaling to make sure the weights of the variables are seen similarly by the machine learning models.



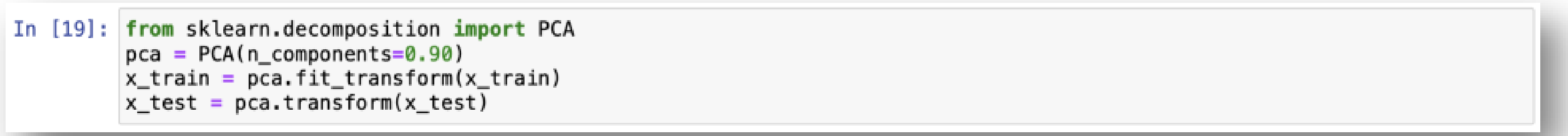
```
In [17]: # Split the data into training and testing sets
         x_train, x_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42)

In [18]: # Scale the data
         scaler = StandardScaler()
         x_train = scaler.fit_transform(x_train)
         x_test = scaler.transform(x_test)
```

Principal Component Analysis (PCA)

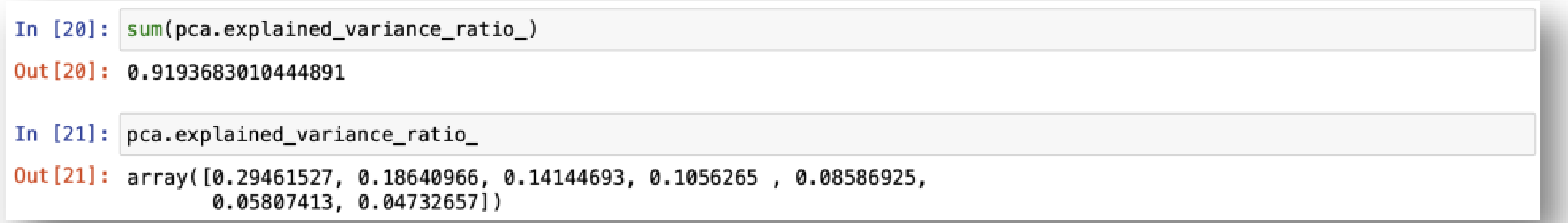
- ***What is PCA?*** PCA is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.
- ***What is the Explained Variance Ration (EVR)?*** the percentage of variance that is attributed by each of the selected components.

Principal Component Analysis cont.

- What we expect for the ***EVR***: 0.9
- ***Resulting EVR:*** *0.9193*



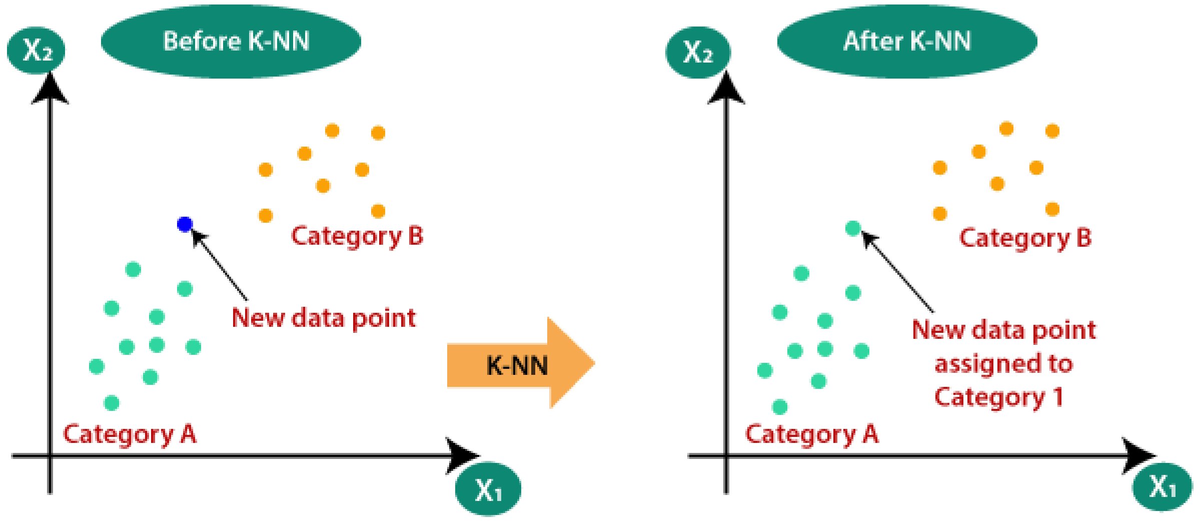
```
In [19]: from sklearn.decomposition import PCA
         pca = PCA(n_components=0.90)
         x_train = pca.fit_transform(x_train)
         x_test = pca.transform(x_test)
```



```
In [20]: sum(pca.explained_variance_ratio_)

Out[20]: 0.9193683010444891

In [21]: pca.explained_variance_ratio_

Out[21]: array([0.29461527, 0.18640966, 0.14144693, 0.1056265 , 0.08586925,
                0.05807413, 0.04732657])
```

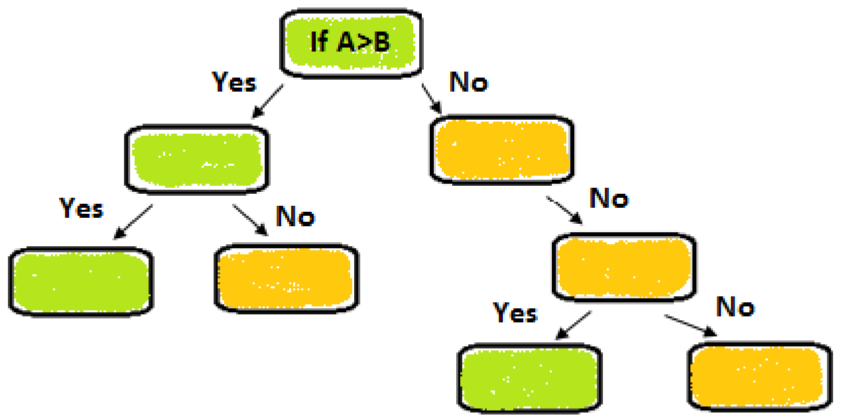
K- Nearest Neighbors Classifier

The K-Nearest Neighbors (KNN) classifier is a type of supervised learning algorithm used for classification tasks. In KNN, the classification of a new data point is based on the class of the K nearest data points in the training dataset.



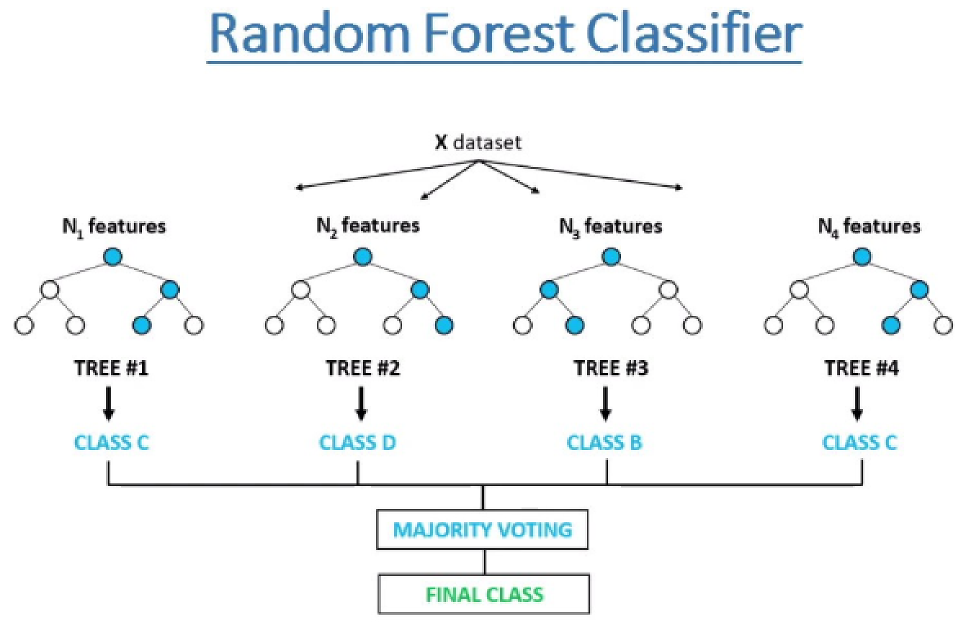
Decision Tree Classifier

The Decision Tree Classifier is a type of supervised learning algorithm used for classification tasks. The algorithm creates a decision tree model based on the training data, which is then used to classify new data points.



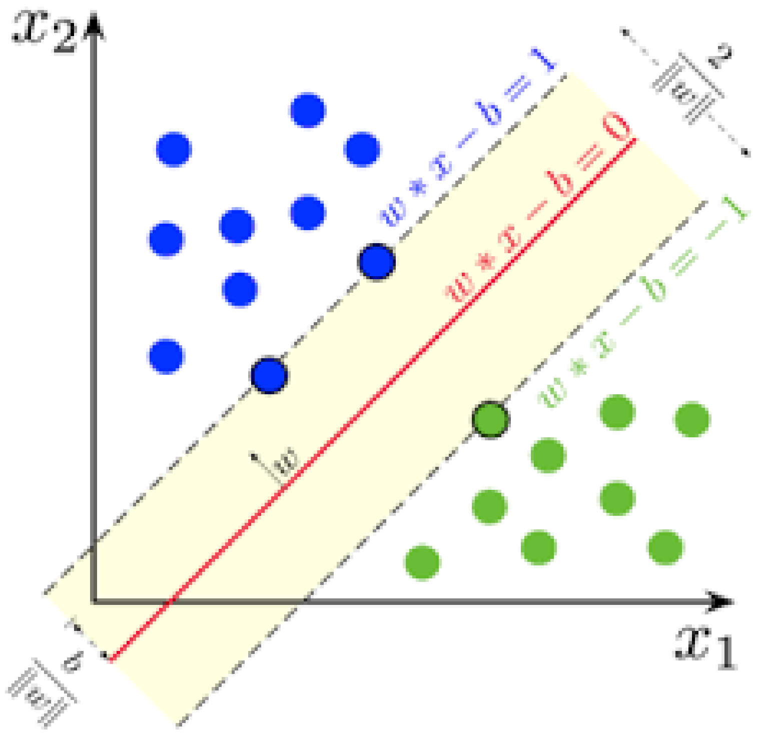
Random Forest Classifier

The Random Forest Classifier is a type of supervised learning algorithm used for classification tasks. The algorithm creates an ensemble of decision trees based on the training data, which are then used to classify new data points.



Support Vector Classifier

The Support Vector Machine (SVM) Classifier is a type of supervised learning algorithm used for classification tasks. The algorithm works by finding the hyperplane that best separates the data into different classes.



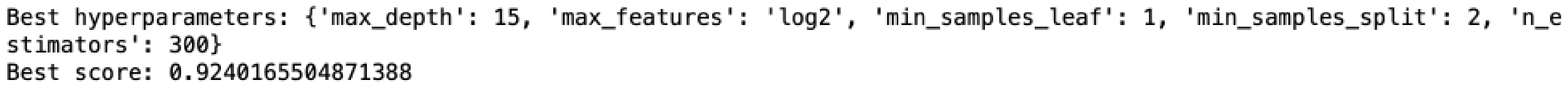
Model Comparison

	KNN	CT	RF	SVR
Accuracy	0.8752	0.8481	0.8987	0.8716
Precision	0.8063	0.8272	0.8678	0.8259
Recall	0.9694	0.8587	0.9274	0.9236
F1	0.8804	0.8426	0.8966	0.8720

Based off the Accuracy and F1 scores, we will move ahead with the Random Forest as the best model.

Hyperparameter Tuning

- Based on the accuracy of the above 4 models, the ***Random Forest Classifier*** had the best accuracy and will be used as the best model
- Tuning using grid_search to pick optimal hyper parameters:
  - 'n_estimators': [100, 200, 300],
  - 'max_depth': [5, 10, 15],
  - 'min_samples_split': [2, 5, 10],
  - 'min_samples_leaf': [1, 2, 4],
  - 'max_features': ['sqrt', 'log2’]
- ***Results:***



```
Best hyperparameters: {'max_depth': 15, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_e
stimators': 300}
Best score: 0.9240165504871388
```

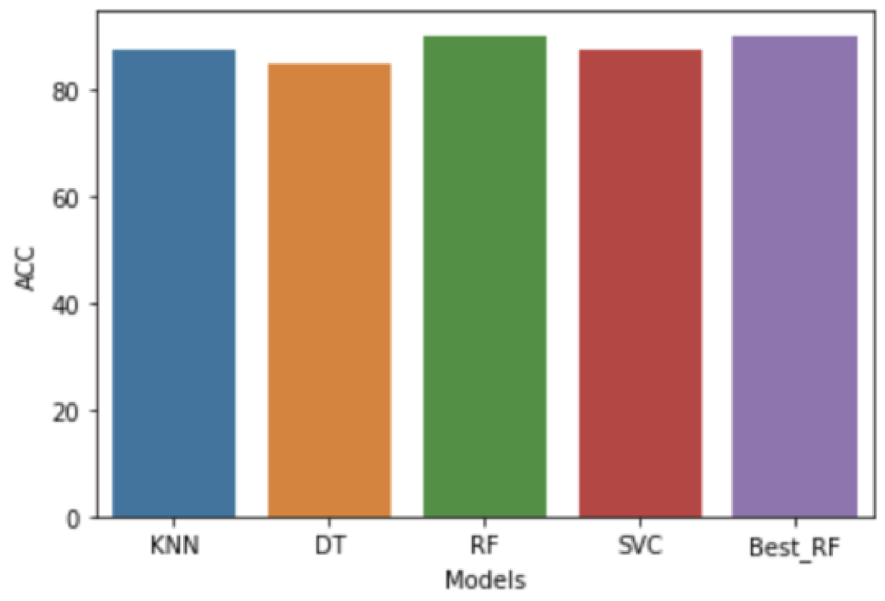
Running Best Model

A random forest classification model employing the best hyper parameters was found by a grid search on the training data. The model is then applied to forecast the test data, and its performance is assessed using accuracy, precision, recall, and F1 scores.

The model's acquired results reveal that it has:
Accuracy = 0.903
Precision score = 0.786
Recall score = 0.468
F1 score = 0.587

Models Compared

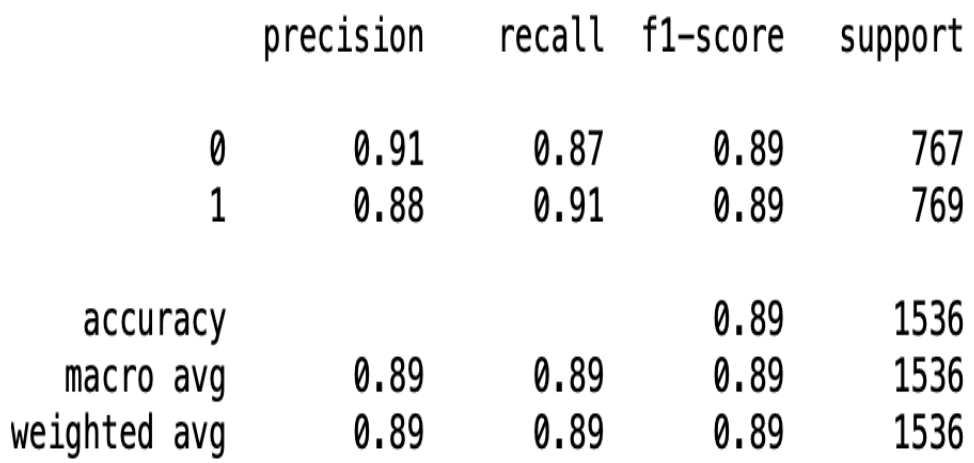
	Models	ACC
0	KNN	87.522604
1	DT	84.810127
2	RF	89.873418
3	SVC	87.160940
4	Best_RF	90.054250



Predicting Quality on new wine data

Results:

The most effective random forest classifier model that was trained on a dataset to predict the quality of red wine is used here. An overall assessment of the classifier model's performance on the test data is provided in this report.



```
              precision    recall  f1-score   support

           0       0.91      0.87      0.89       767
           1       0.88      0.91      0.89       769

    accuracy                           0.89      1536
   macro avg       0.89      0.89      0.89      1536
weighted avg       0.89      0.89      0.89      1536
```

Thank you!