BIRZEIT UNIVERSITY

Faculty of Engineering and Technology

Electrical and Computer Engineering Department

MACHINE LEARNING AND DATA SCIENCE

ENCS5341

Assignment No.3

Students's Name:

Nour Rabee'
Yousef Shamasneh

ID Numbers:

1191035
1190300

Instructor's Name: Dr. Yazan Abu Farha

Section: 2

January 26th, 2024

Table of Contents

Table of Figures 3
Introduction 4
White Wine Quality Dataset 5
Experiments & Results 9
  Baseline Model KNN 9
  The proposed ML models 10
Analysis 11
Conclusion 13

Table of Figures

Figure 1: Analyzing Wine Data 5
Figure 2: Calculating the number of outliers for each feature 5
Figure 3: Visualization the distribution and observe the outliers in Volatile acidity 5
Figure 4: Calculating the number of outliers after log transformation 6
Figure 5: Visualization the distribution and observe the outliers in Volatile acidity after log transformation 6
Figure 6: Checking missing values 7
Figure 7: Checking redundancy 7
Figure 8: Bar chart of sample statistics after oversampling 7
Figure 9: Quality statistics bar chart 7
Figure 10:Data Correlation after applying Log transformation 8
Figure 11: Initial Data Correlation 8
Figure 12: Data Correlation after removing duplicates 8
Figure 13: Results when k = 3 9
Figure 14: Results when K =1 9
Figure 15: Confusion Matrix when k = 3 9
Figure 16: Confusion Matrix when k =1 9
Figure 17: SVC Classifier Results 10
Figure 18: Random Forest Classifier Results 10
Figure 19: Comparing model's accuracy 11
Figure 20: ROC curve for all proposed models 12
Figure 21: Some instances from the test set where SVC model exhibits errors 12

Introduction

Wine has a rich history dating back 7500 years, with white wine gaining attention in ancient Greece for medicinal use. Today, the production of white wine involves a detailed process, from planting grapes to bottling, and any small change in any step will lead to a great difference in taste and quality after the final white wine is produced.

Evaluating wine quality traditionally involves complex tasting steps. To simplify and enhance this process, we analyze data to understand how different ingredients impact white wine quality. Our goal is to predict if a white wine is "good" or "bad". we have employed a variety of machine learning models like k-Nearest Neighbors as a baseline model, Random Forests, and Support Vector Machines. KNN leverages the majority class of nearby neighbors, Random Forests constructs multiple decision trees, and SVM seeks an optimal hyperplane for class separation.

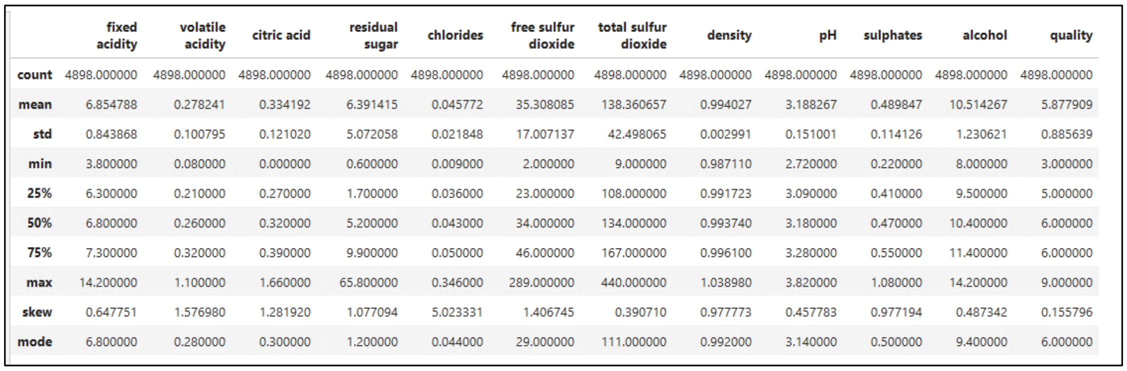
To check how well they're doing, we use metrics like accuracy, precision, recall, and F1 score. Accuracy tells us how often the model is correct overall. Precision shows how good the model is at avoiding wrong positive predictions. Recall checks if the model captures all the actual positive cases. F1 Score combines precision and recall for a balanced view.

We also use the ROC “Receiver Operating Characteristic” curve to compare between models. And finally we use the confusion matrix to evaluate the performance of a classification model. It provides a summary of the model's predictions, comparing them to the actual class labels in the dataset. The matrix is particularly useful for understanding the types and frequency of errors made by the model.

White Wine Quality Dataset

The dataset which consists of 4898 examples was employed in this project is designed for predicting the quality of white wine, comprising 11 features, all of which are continuous values of type float64 related to the chemical composition and properties of the wines. The target variable (quality), which consists of discrete values of type int64, is the quality rating of the wine, typically assigned by human tasters.

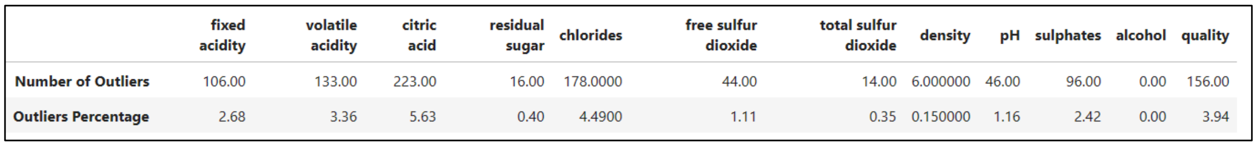
So, before applying any machine learning model, it's essential to conduct data cleaning to ensure that the dataset is in a suitable and reliable state. This includes checking for missing values, handling redundancies or duplicates, addressing any imbalances in the data distribution, and normalizing the dataset. All of these cannot be happen without analyzing the data.



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657	0.994027	3.188267	0.489847	10.514267	5.877909
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991	0.151001	0.114126	1.230621	0.885639
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723	3.090000	0.410000	9.500000	5.000000
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740	3.180000	0.470000	10.400000	6.000000
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100	3.280000	0.550000	11.400000	6.000000
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980	3.820000	1.080000	14.200000	9.000000
skew	0.647751	1.576980	1.281920	1.077094	5.023331	1.406745	0.390710	0.977773	0.457783	0.977194	0.487342	0.155796
mode	6.800000	0.280000	0.300000	1.200000	0.044000	29.000000	111.000000	0.992000	3.140000	0.500000	9.400000	6.000000

Figure 1: Analyzing Wine Data

The above table presents statistical information about the dataset. We observe positive skewness in all features, including the target variable, with skew values greater than 0. This positive skewness indicates a tendency for higher values in the dataset, suggesting the presence of outliers or extreme values on the higher side. Upon calculating, we find a total of 1018.0 outliers in the dataset.



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
Number of Outliers	106.00	133.00	223.00	16.00	178.0000	44.00	14.00	6.000000	46.00	96.00	0.00	156.00
Outliers Percentage	2.68	3.36	5.63	0.40	4.4900	1.11	0.35	0.150000	1.16	2.42	0.00	3.94

Figure 2: Calculating the number of outliers for each feature

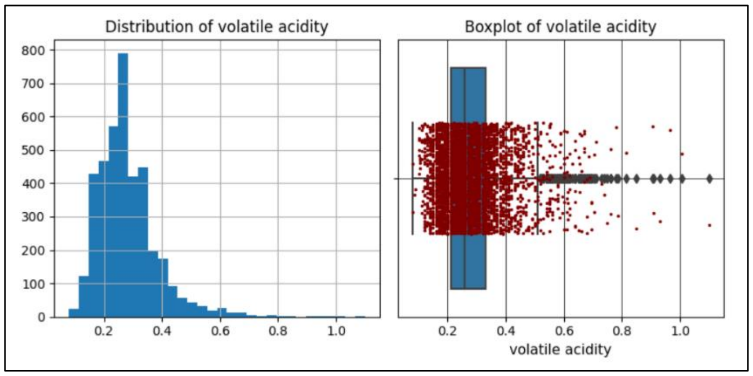
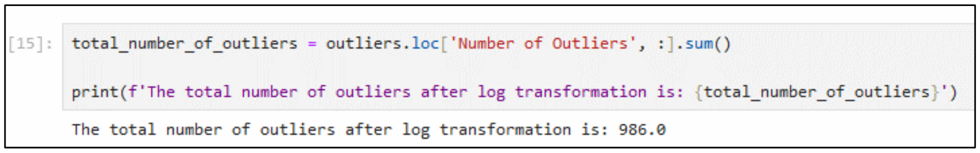


Figure 3: Visualization the distribution and observe the outliers in Volatile acidity

The left side of Figure 3 illustrates the distribution of Volatile Acidity, and the right side displays the corresponding box plot. Examining the box plot reveals outliers, defined as data points located outside the whiskers. It's evident that the distribution is positively skewed, with numerous outliers extending beyond the box plot whiskers.

We applied log transformation to all attributes to minimize the number of outliers and their impact. However, log transformation is not applicable for citric acid, due to its minimum value of zero. Therefore, we excluded it from the log transformation process and also we exclude the target variable.



```
[15]: total_number_of_outliers = outliers.loc['Number of Outliers', :].sum()

      print(f'The total number of outliers after log transformation is: {total_number_of_outliers}')

      The total number of outliers after log transformation is: 986.0
```

Figure 4: Calculating the number of outliers after log transformation

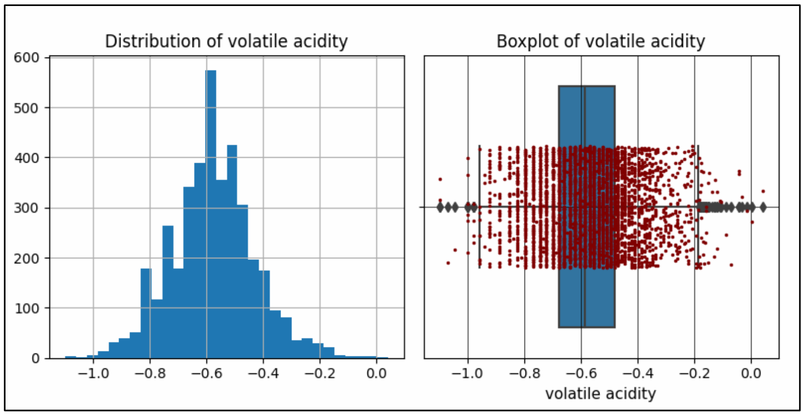
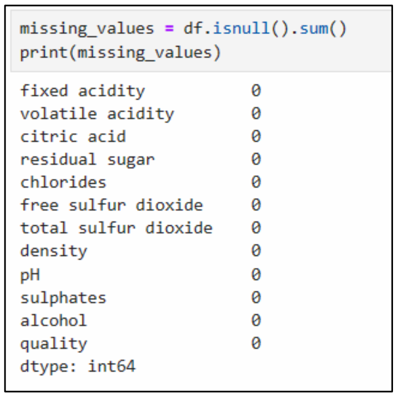


Figure 5: Visualization the distribution and observe the outliers in Volatile acidity after log transformation

We can observe a reduction in the total number of outliers. The count of outliers before log transformation was 1018, and it decreased to 986 after transformation, which is a positive outcome! The left side of Figure 5 illustrates the distribution of Volatile Acidity after log transformation. It approaches a more normal distribution, and the right side represents the significant reduction in the number of outliers as shown in the box plot.

Now we go through data preprocessing. We started by **cleaning** it by making sure there are no missing values, fortunately, there aren't any! Next, we looked for redundancy, and we found 937 rows that are repeated. Redundancy means having the same data in different places, and it can waste space and make things more complicated to manage. Since we plan to split our data into

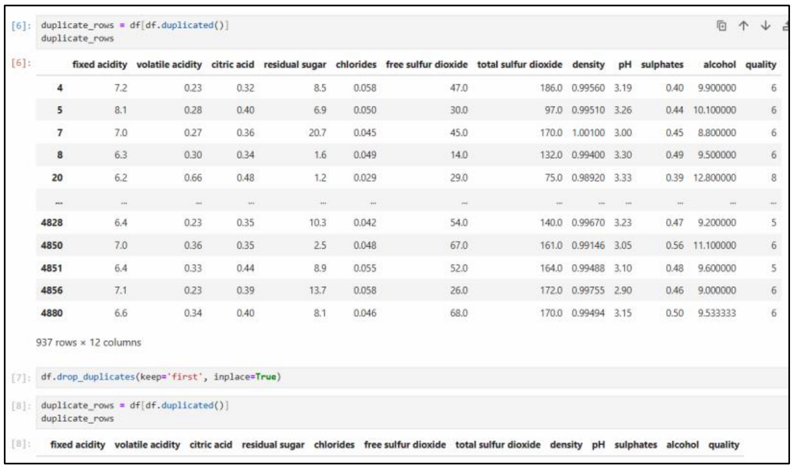
training, validation, and testing sets, having redundant rows might cause a problem. It could lead to a training sample appearing in the testing set or validation set or both, making it harder to check how well our model works on new, unseen data! So we end up dropping them and keeping the first instance of each duplicated value. This is one of the ways of **data reduction**.



```
missing_values = df.isnull().sum()
print(missing_values)
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

Figure 6: Checking missing values



```
[6]: duplicate_rows = df[df.duplicated()]
     duplicate_rows
```

[6]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.900000	6
5	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.100000	6
7	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.800000	6
8	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.500000	6
20	6.2	0.66	0.48	1.2	0.029	29.0	75.0	0.98920	3.33	0.39	12.800000	8
...	...	...	...	...	...	...	...	...	...	...	...	...
4828	6.4	0.23	0.35	10.3	0.042	54.0	140.0	0.99670	3.23	0.47	9.200000	5
4850	7.0	0.36	0.35	2.5	0.048	67.0	161.0	0.99146	3.05	0.56	11.100000	6
4851	6.4	0.33	0.44	8.9	0.055	52.0	164.0	0.99488	3.10	0.48	9.600000	5
4856	7.1	0.23	0.39	13.7	0.058	26.0	172.0	0.99755	2.90	0.46	9.000000	6
4880	6.6	0.34	0.40	8.1	0.046	68.0	170.0	0.99494	3.15	0.50	9.533333	6

937 rows × 12 columns

```
[7]: df.drop_duplicates(keep='first', inplace=True)

[8]: duplicate_rows = df[df.duplicated()]
     duplicate_rows
```

[8]:

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality

Figure 7: Checking redundancy

Upon analyzing the data, we noticed that the quality ratings for the "Target variable" in the dataset range from 3 to 9. This diverse range suggested a classification problem. Initially, we attempted various models for multiclass classification, but the results were not as satisfying as desired. Therefore, we decided to convert our task into binary classification, and with this adjustment, we achieved higher accuracy using machine learning models. This shift allows us to better distinguish between two classes, improving the model's performance.

We categorized the examples into two classes: good wine quality (1) and bad wine quality (0). High-quality wines are those with a quality value greater than the mean value of the output (quality) data feature, while low-quality wines have quality values less than the mean. After creating the target variable, we noticed that the number of wines classified as high quality (1) was greater than those classified as low quality (0). This imbalance, where one class has more instances than the other, presents a challenge known as an imbalanced classification problem. It's a situation that requires special attention during model training to ensure fair and accurate predictions for both classes.

To fix this problem, we used a method called oversampling. This involves creating more synthetic data for the less common type of wine quality. We did this using a technique called Synthetic Minority Oversampling Technique (SMOTE). This helps ensure our models don't get biased and can make accurate predictions for both types of wine quality.

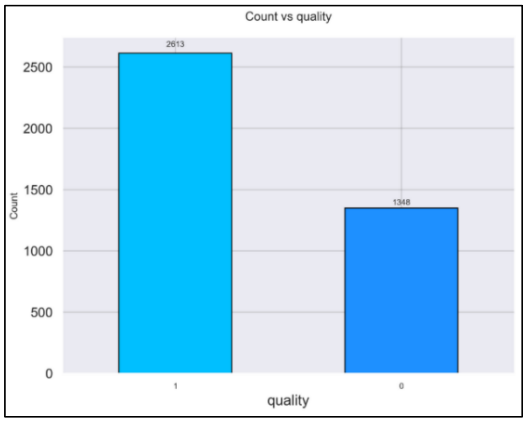


Figure 9: Quality statistics bar chart

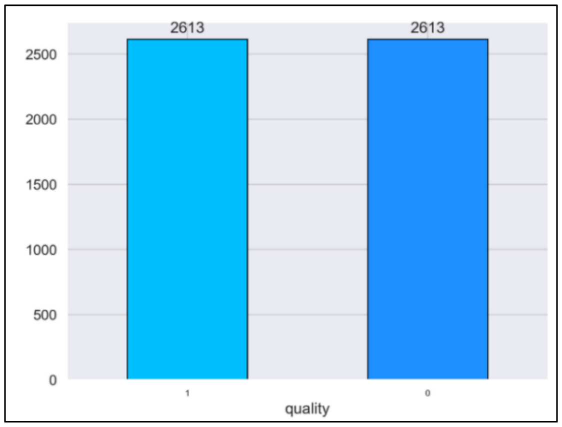
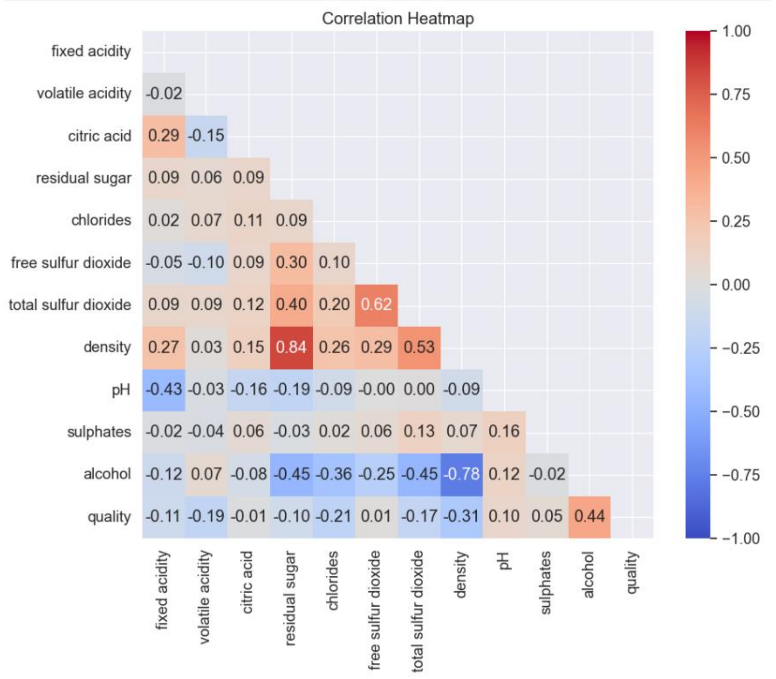


Figure 8: Bar chart of sample statistics after oversampling

We also do **data transformation** by doing min-max normalization, which involves scaling the values to limit the range between 0 and 1. This helps in reducing the influence of outliers and ensures a more consistent and balanced representation of the data.

We can observe from heat map figures how the correlation between features have been reduced after log transformation and after removing the duplicates which is sometimes better because models trained on data with lower feature correlations often generalize better to new, unseen data.

Also, for instance, Random forests can be good at detecting interactions between different features, but highly correlated features can mask these interactions.



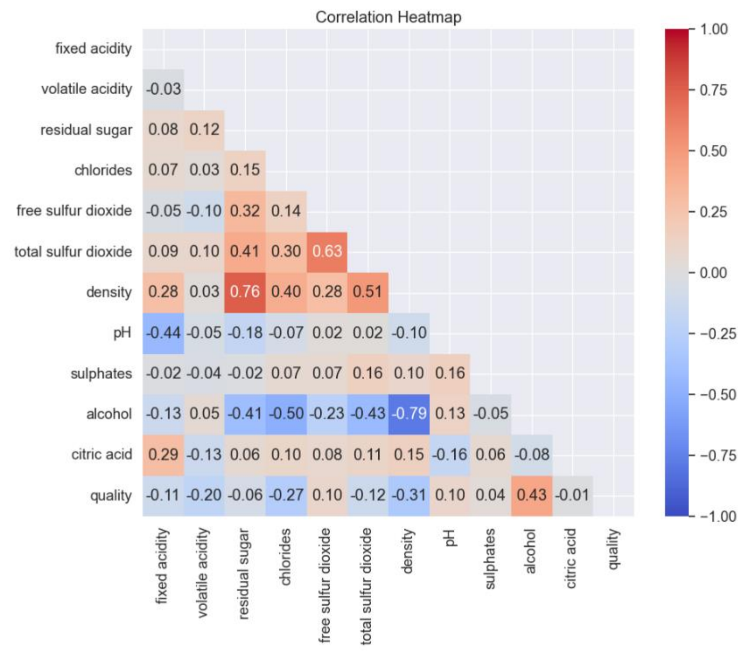


Figure 10:Data Correlation after applying Log transformation

Figure 11: Initial Data Correlation

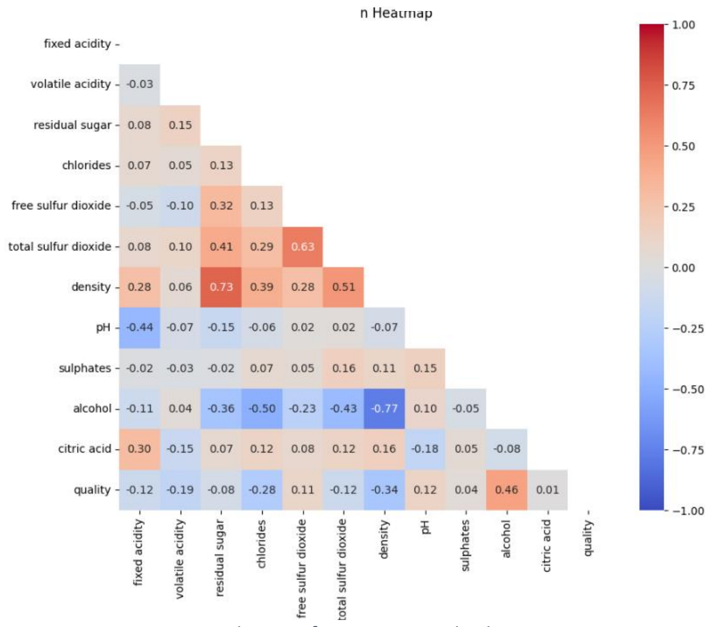


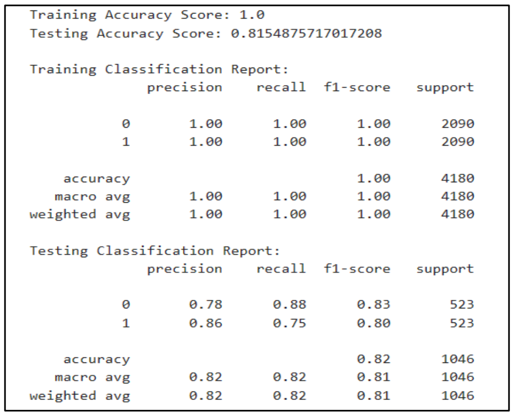
Figure 12: Data Correlation after removing duplicates

Experiments & Results

Before applying any model, we split the data into 80% training set and 20% testing set.

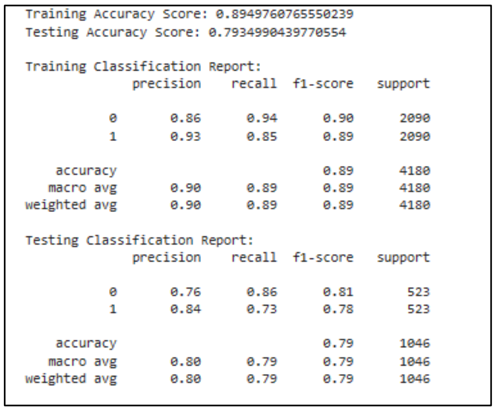
Baseline Model KNN

After applying kNN with k = 1 and k = 3 on the dataset using Euclidean as a distance metric, the results are as follows:



```
Training Accuracy Score: 1.0
Testing Accuracy Score: 0.8154875717017208

Training Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2090
           1       1.00      1.00      1.00      2090

    accuracy                           1.00      4180
   macro avg       1.00      1.00      1.00      4180
weighted avg       1.00      1.00      1.00      4180

Testing Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.88      0.83       523
           1       0.86      0.75      0.80       523

    accuracy                           0.82      1046
   macro avg       0.82      0.82      0.81      1046
weighted avg       0.82      0.82      0.81      1046
```

Figure 14: Results when K =1



```
Training Accuracy Score: 0.8949760765550239
Testing Accuracy Score: 0.7934990439770554

Training Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.94      0.90      2090
           1       0.93      0.85      0.89      2090

    accuracy                           0.89      4180
   macro avg       0.90      0.89      0.89      4180
weighted avg       0.90      0.89      0.89      4180

Testing Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.86      0.81       523
           1       0.84      0.73      0.78       523

    accuracy                           0.79      1046
   macro avg       0.80      0.79      0.79      1046
weighted avg       0.80      0.79      0.79      1046
```

Figure 13: Results when k = 3

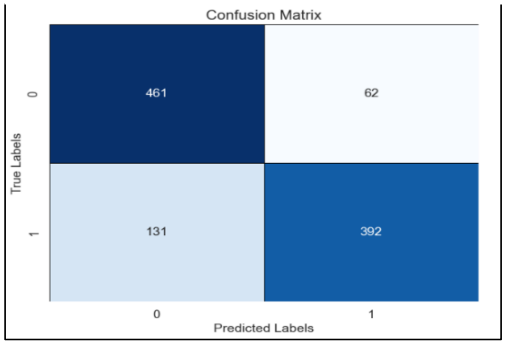


Figure 16: Confusion Matrix when k =1

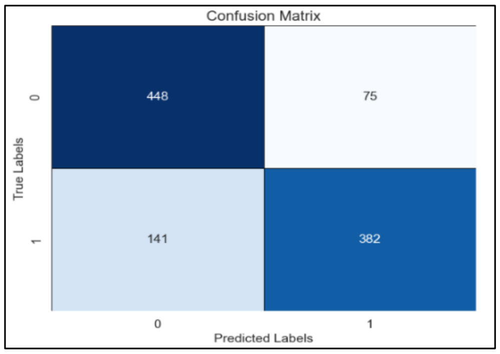


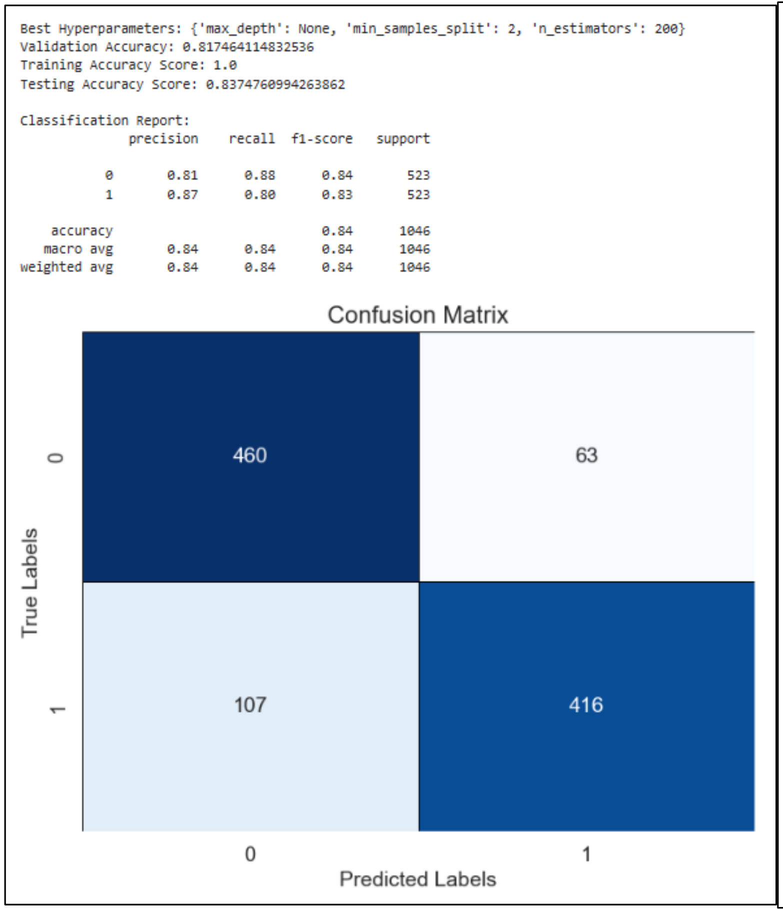
Figure 15: Confusion Matrix when k = 3

It's evident that the training accuracy when k=1 is 1, which is expected because the closest point to any training point is itself. However, the model's performance on the testing data is 0.815, 0.185 less than the training accuracy. This inconsistency indicates overfitting, as the model appears to memorize the training data but struggles to generalize during testing. In contrast, for k=3, the training accuracy is 0.89, and the testing accuracy is 0.79, with only a 0.1 difference. This model generalizes better than when k=1.

Additionally, true positives are 461 when k=1 and 448 when k=3. In the context of wine classification, prioritizing false positives (FP) over false negatives (FN) suggests that incorrectly classifying a lower-quality wine as higher quality (FP) has more significant consequences. This could be due to factors such as higher production costs, potential damage to consumer trust and brand reputation, or adherence to industry standards. This prioritization aligns with the goal of minimizing false positives, making the k=3 model superior to k=1 as it exhibits better generalization and fewer false positives.

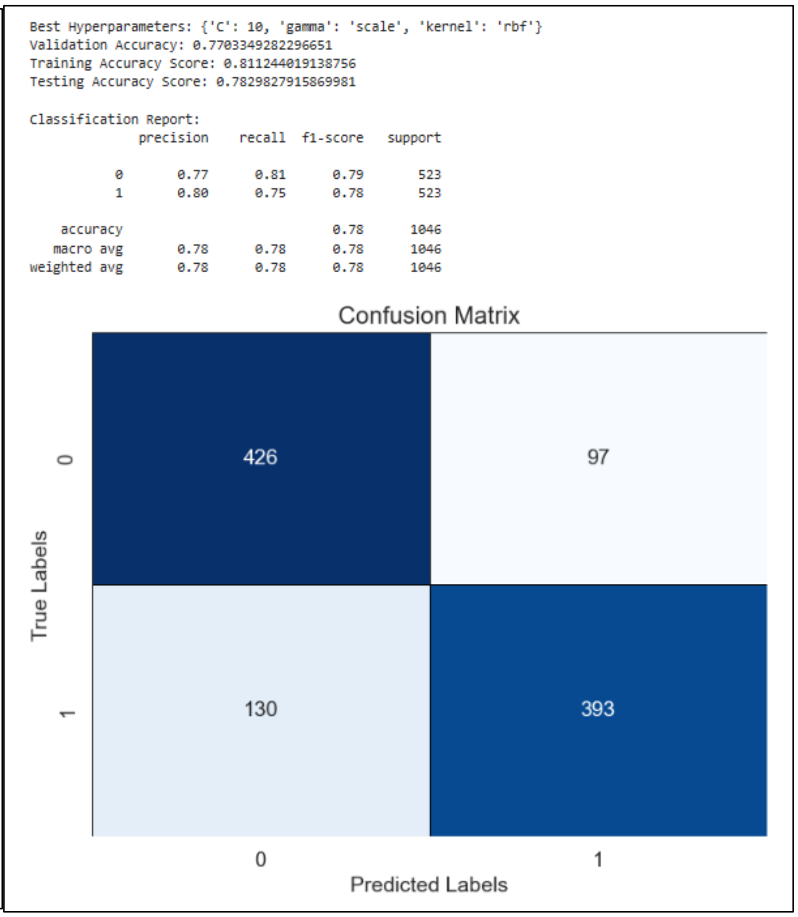
The proposed ML models

We have evaluated two additional models, the Random Forest Classifier and the SVC Classifier, using the grid search cross-validation technique. For each model, we applied a similar approach: tuning the hyperparameters to optimize performance on the validation set, which is selected using cross-validation with the number of folds set to 5. The best-performing model for each classifier on the validation set will be selected based on the achieved results during this tuning process. In the Random Forest, we tuned three different hyperparameters with four different values for each: max_depth, min_samples_split, and n_estimators. The best hyperparameters for Random Forest Classifier are when the maximum depth is set to None, the minimum samples split is 2, and the number of estimators (n_estimators) is 200. Choosing max_depth as None allows the trees in the forest to grow without any maximum depth constraint, potentially capturing more complex relationships in the data. A min_samples_split value of 2 means that a node will only be split if it contains at least 2 samples, supporting finer splits in the tree. Additionally, choosing n_estimators as 200 indicates that the Random Forest consists of 200 trees.



```
Best Hyperparameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 200}
Validation Accuracy: 0.817464114832536
Training Accuracy Score: 1.0
Testing Accuracy Score: 0.8374760994263862

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.88      0.84       523
           1       0.87      0.80      0.83       523

    accuracy                           0.84      1046
   macro avg       0.84      0.84      0.84      1046
weighted avg       0.84      0.84      0.84      1046
```

Figure 18: Random Forest Classifier Results



```
Best Hyperparameters: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
Validation Accuracy: 0.7703349282296651
Training Accuracy Score: 0.811244019138756
Testing Accuracy Score: 0.7829827915869981

Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.81      0.79       523
           1       0.80      0.75      0.78       523

    accuracy                           0.78      1046
   macro avg       0.78      0.78      0.78      1046
weighted avg       0.78      0.78      0.78      1046
```

Figure 17: SVC Classifier Results

In the SVC, we tuned three different hyperparameters with four different values for each: C, gamma, and kernel. The best hyperparameters for the SVC Classifier are when C is set to 10, gamma is set to 'scale,' and the kernel is set to 'rbf.' Setting C to 10 implies a regularization parameter that balances the trade-off between achieving a smooth decision boundary and accurately classifying the training points. The gamma parameter defines how far the influence of a single training example reaches, where low values mean 'far' and high values mean 'close.' Choosing gamma as 'scale' indicates that the algorithm will automatically adapt the value of gamma based on the scale of the input data. Additionally, selecting the kernel as 'rbf' suggests the use of the Radial Basis Function kernel for non-linear classification.

Figure 17 and Figure 18 show us the results for both random forest and SVC classifiers. Random Forest Classifier achieved higher testing accuracy than SVC, while shows signs of potential overfitting by perfect training accuracy. Random Forest has 460 true positives while SVC has only 426.

Analysis

It’s obvious that KNN with both k equals 1 and 3, and random forest overfit, while SVC with hyper-parameters: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'} is more generalizable because the difference between training and testing is very small compared to other proposed models.

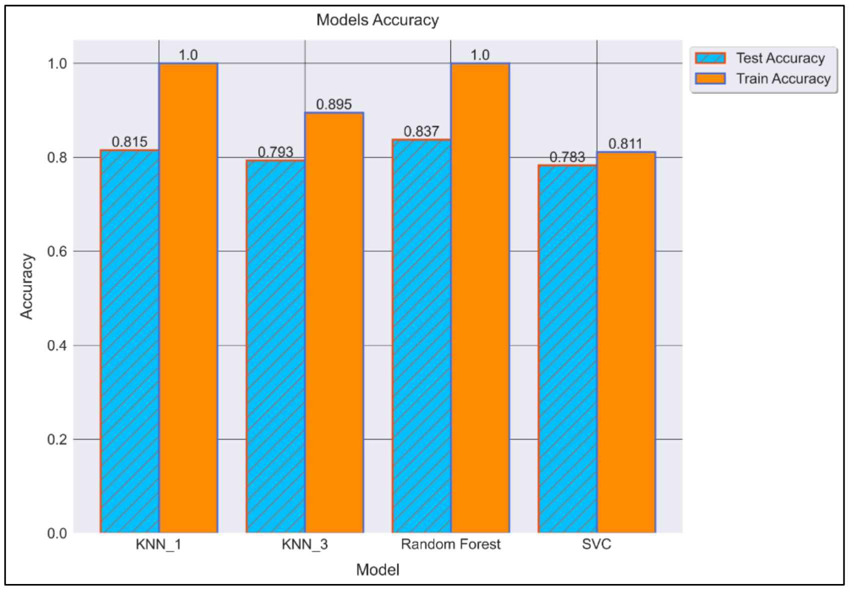


Figure 19: Comparing model's accuracy

Testing accuracy is an important metric, but it's not the only one to consider. A model with slightly lower testing accuracy might still be preferable if it generalizes better to new data. Our primary goal is to deploy a model in a real-world scenario where it will encounter new, unseen data, so we are going to prioritize a model with better generalization, even if it has slightly lower testing accuracy. If our model is used in a critical application where correctly identifying positive instances is crucial, and false positives are costlier than false negatives, we might lean toward the model with a higher AUC which is the Random Forest Classifier.

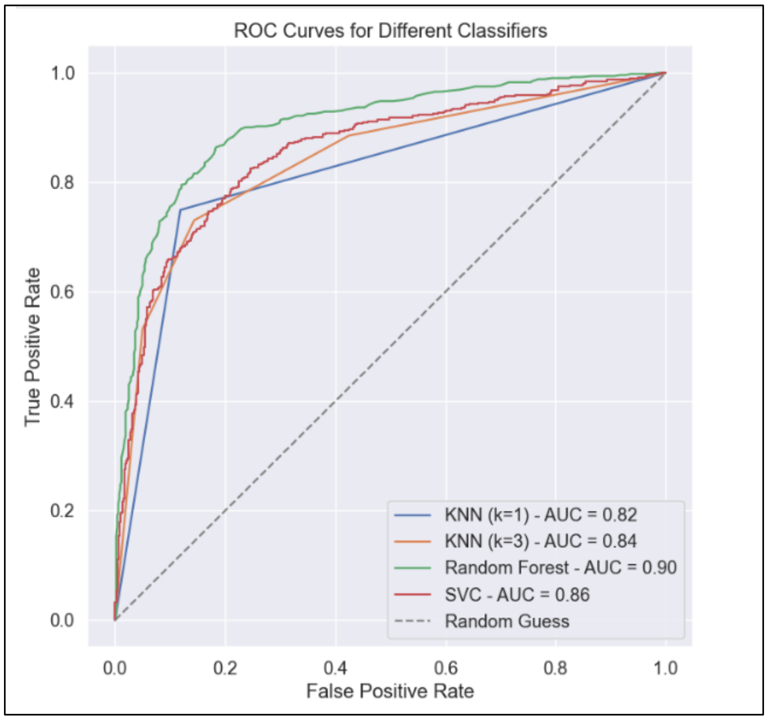
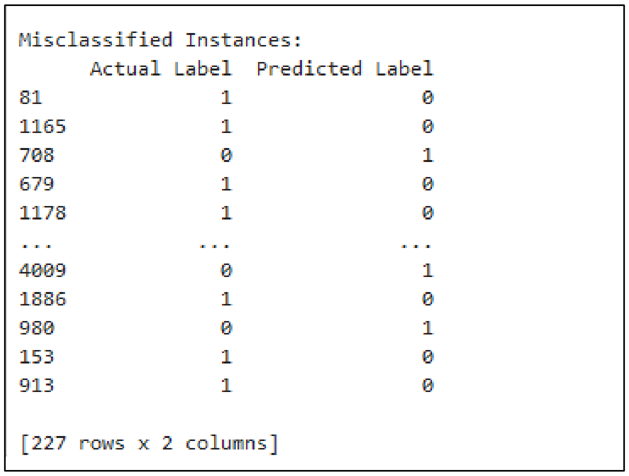


Figure 20: ROC curve for all proposed models

We can also apply regularization techniques to mitigate the impact of overfitting in Random Forest. This can be achieved through early stopping criteria or by exploring a broader range of values for certain hyper-parameters, such as n-estimators and maximum depth.



```
Misclassified Instances:
      Actual Label  Predicted Label
81               1                0
1165             1                0
708              0                1
679              1                0
1178             1                0
...            ...              ...
4009             0                1
1886             1                0
980              0                1
153              1                0
913              1                0

[227 rows x 2 columns]
```

Figure 21: Some instances from the test set where SVC model exhibits errors

Conclusion

To sum up, our study aimed to find the best model for classifying wines. After testing various machine learning models, we found that the Support Vector Classifier (SVC) with specific settings worked well. It balanced accuracy on both training and testing data, indicating it can handle new, unseen wine samples.

However, it's important to note some limitations. The dataset itself might have biases, and the metrics we used might not cover all real-world scenarios. Despite our efforts to fine-tune the models, issues like overfitting were still present, especially in models like K-Nearest Neighbors and Random Forest. Also the size of the dataset is not very large and diverse to develop more robust models.

For future work, exploring more advanced techniques like deep neural networks and considering domain knowledge could improve our models.