A Report On: -

Impact of Hyperparameter Tuning on Model Performance

Submitted By: -

Rajan Ghimire
C0924991

Submitted To: -

Victoria Shtern

Contents

Abstract

We aim to evaluate and compare the performance of three classification algorithms, DecisionTreeClassifier, RandomForestClassifier and LogisticRegression, on the Wine Quality Dataset. This dataset contains physicochemical properties of various Portuguese "Vinho Verde" wines and their quality ratings. The goal is to find the optimal hyperparameters for each algorithm and determine which model provides the best accuracy in predicting wine quality.

Introduction:

Hyperparameter tuning is essential for optimizing the performance of machine learning models. This process involves selecting the best set of hyperparameters for a model to achieve the highest accuracy and generalization to unseen data. In this assignment, we evaluate and compare the performance of three classification algorithms: Decision Tree, Random Forest, and Logistic Regression. By systematically tuning their hyperparameters, we aim to identify the most accurate model for predicting wine quality based on its physicochemical properties.

Objectives:

- Use Grid Search or Random Search to find the optimal hyperparameters for each algorithm.
- Train the models on the training set and evaluate their performance on the testing set using accuracy and other relevant metrics.
-

About the Dataset:

The dataset used in this study is the Wine Quality Dataset from the UCI Machine Learning Repository. It contains 1,143 samples of red "Vinho Verde" wine, each described by 11 physicochemical properties and a quality rating. The features and their descriptions are : 'fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar', 'chlorides, 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density', 'ph', 'sulphates', 'alcohol', 'quality'.

Data Validation:

Valid Data Types:

```
Incorrect df types:
None
Data is correct with following:
fixed_acidity           float64
volatile_acidity        float64
citric_acid             float64
residual_sugar          float64
chlorides               float64
free_sulfur_dioxide     float64
total_sulfur_dioxide    float64
density                 float64
ph                      float64
sulphates               float64
alcohol                 float64
quality                   int64
id                        int64
dtype: object
```

There were no Incorrect data, and all data was correct and was in correct format.

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
id                      0
```

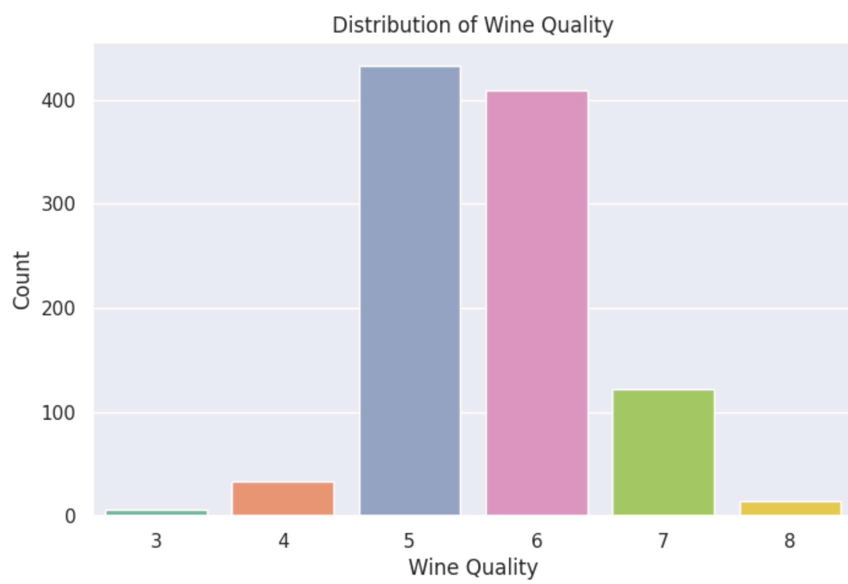
There are no missing values in the dataset.

Duplicate Values:

There were **125** duplicate values in the dataset. And the total percentage of duplicate values was **10%** and has been removed in the final dataset.

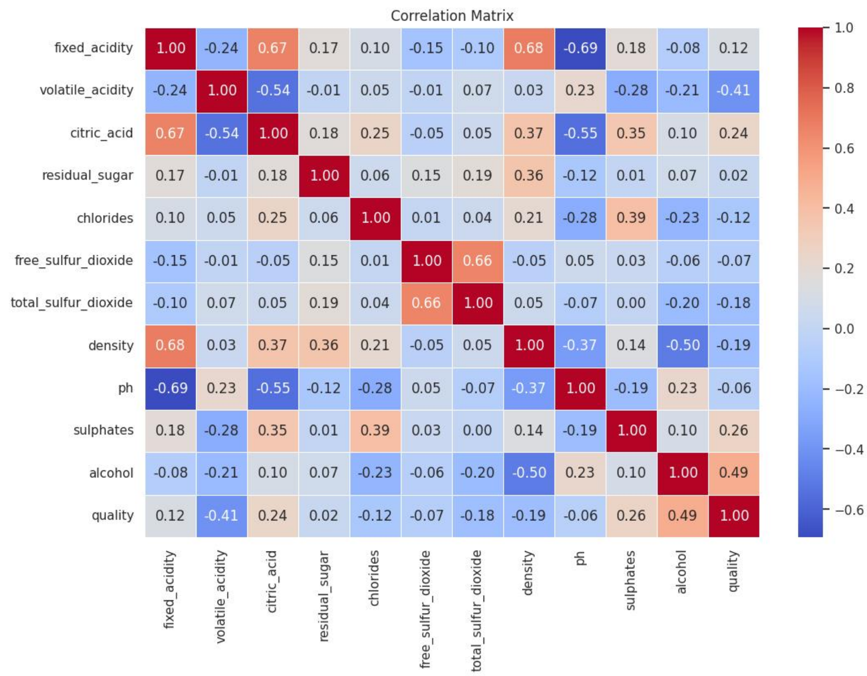
Exploratory Data Analysis:

Target Distribution:



The dataset is not well balanced with majority of class belonging to 5 and 6 labels.

Co-relation:



This correlation matrix shows the relationships between various features in the wines dataset. The most positively correlated features are 'density' and 'fixed_acidity' (0.68), while 'quality' is highly correlated with 'alcohol' (0.49). The most negatively correlated features are 'density' and 'alcohol' (-0.50), and 'fixed_acidity' and 'pH' (-0.69). 'Volatile_acidity' and 'quality' also show a strong negative correlation (-0.41).

Data Engineering from Data Mining:

```
df['Total_sulphur_Dioxide'] = df['free_sulfur_dioxide'] + df['total_sulfur_dioxide']
df = df.drop(columns = ['free_sulfur_dioxide','total_sulfur_dioxide'])
df['Acidity'] = df['fixed_acidity'] + df['volatile_acidity'] + df['citric_acid']

df = df.drop(columns = ['fixed_acidity','volatile_acidity','citric_acid'])

def categorize_sugar(sugar):
  if sugar< 1.5 :
    return "low"
  elif sugar >1.5 and sugar<7:
    return "medium"
  else:
    return "high"

df['residual_sugar'] = df['residual_sugar'].apply(categorize_sugar)

def categorize_pH(pH):
  if pH<3:
    return "acidic"
  elif pH>=3 and pH<=4:
    return "neutral"
  else:
    return "basic"

df['ph'] = df['ph'].apply(categorize_pH)

cate_cols = ['residual_sugar', 'ph']

df = pd.get_dummies(df, columns=cate_cols)

df["residual_sugar_high"]= df["residual_sugar_high"].astype(int)
df["residual_sugar_low"]= df["residual_sugar_low"].astype(int)
df["residual_sugar_medium"]= df["residual_sugar_medium"].astype(int)
df["ph_acidic"]= df["ph_acidic"].astype(int)
df["ph_basic"]= df["ph_basic"].astype(int)
df["ph_neutral"]= df["ph_neutral"].astype(int)


# Train test Split
X=df.drop("quality",axis=1)
y=df['quality']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=42)
```

The feature engineering code creates a new feature `Total_sulphur_Dioxide` by summing `free_sulfur_dioxide` and `total_sulfur_dioxide`, and an `Acidity` feature by summing `fixed_acidity`, `volatile_acidity`, and `citric_acid`, then drops the original columns. It categorizes `residual_sugar` into 'low', 'medium', and 'high', and `ph` into 'acidic', 'neutral', and 'basic', followed by one-hot encoding these categorical variables.

Finally, the dataset is split into features (`X`) and target (`y`), and then into training and testing sets using a **75-25** split with `train_test_split`, ensuring a robust evaluation of the model on unseen data.

Hyperparameter Experiments:

DecisionTreeClassifier:

Before the Hyperparameter Tuning:

```
clf2 = DecisionTreeClassifier()
clf2.fit(X_train, y_train)

# Make predictions on the test set
y_pred2 = clf2.predict(X_test)

# Calculate the test accuracy
accuracy2 = accuracy_score(y_pred2, y_test)

# Calculate the training accuracy
train_accuracy2 = clf2.score(X_train, y_train)

print("Training Accuracy for DecisionTreeClassifier: ", train_accuracy2)
print("Test Accuracy for DecisionTreeClassifier: ", accuracy2)
```

Here we got Training Accuracy for DecisionTreeClassifier as **100%** and Test Accuracy for DecisionTreeClassifier: as **43.52%**.

After Hyperparameter Tuning:

```
parameters2 = {
    'criterion' : ['gini','entropy'],
    'splitter' : ['best','random'],
    'max_depth' : [1,2,3,4,5],
    'max_features' : ['auto','sqrt','log2']
}

clf2 = GridSearchCV(treeclassifier, param_grid = parameters2, cv=5,scoring='accuracy')
clf2.fit(X_train,y_train)
from sklearn.metrics import accuracy_score
# Extract the best parameters from the grid search
best_params2 = clf2.best_params_
# Refit the DecisionTreeClassifier with the best parameters
clf2 = DecisionTreeClassifier(**best_params2)
clf2.fit(X_train, y_train)
# Make predictions on the test set
y_pred2 = clf2.predict(X_test)
# Calculate the test accuracy
accuracy2 = accuracy_score(y_pred2, y_test)
# Calculate the training accuracy
train_accuracy2 = clf2.score(X_train, y_train)
```

After Hyperparameter Tuning the best parameters were: **'criterion': 'gini', 'max_depth': 5, 'max_features': 'log2', 'splitter': 'best'.** And Training Accuracy for DecisionTreeClassifier was **66.97%** Test Accuracy for was **46.27%.**

Testing on Data:

```
# Generate random data with the same shape and features as X_train
X_train_random = np.random.rand(X_train.shape[0], X_train.shape[1])

print(f"Random Data: {[X_train_random[0]]}")
# Make predictions on the random data
y_pred_random = clf2.predict([X_train_random[0]])

# Print predictions for the random data
print("Predictions for random data: ", y_pred_random)
```

✓ 0.0s

```
Random Data: [array([0.46698044, 0.2203128 , 0.97074431, 0.88338327, 0.77901451,
       0.99648123, 0.05409705, 0.11569042, 0.22857033, 0.10779196,
       0.82735638, 0.0428412 , 0.90598367])]
Predictions for random data:  [5]
```

RandomForestClassifier:

Before the Hyperparameter Tuning:

```
clf3__ = RandomForestClassifier()
clf3__.fit(X_train,y_train)

train_accuracy3 = clf3__.score(X_train, y_train)

y_pred3 = clf3__.predict(X_test)
accuracy3 = accuracy_score(y_test,y_pred3)
```

Here we got Training Accuracy for RandomForestClassifier as **100%** and Test Accuracy for RandomForestClassifier: as **50.1%**.

After Hyperparameter Tuning:

```
clf3 =RandomForestClassifier()

parameters3 = {
    'criterion' : ['gini','entropy'],
    'max_depth' : [1,2,3,4,5,6,7,8,9],
    'n_estimators' : [1,10,100,200,300,500,1000]
}
clf3 = RandomizedSearchCV(clf3, param_distributions =parameters3, scoring='accuracy',cv=5,verbose=3)

clf3.fit(X_train,y_train)



best_params3 = clf3.best_params_
clf3__ = RandomForestClassifier(**best_params3)
clf3__.fit(X_train,y_train)

train_accuracy3 = clf3__.score(X_train, y_train)

y_pred3 = clf3__.predict(X_test)
accuracy3 = accuracy_score(y_test,y_pred3)
```

After Hyperparameter Tuning the best parameters were: **'n_estimators': 300, 'max_depth': 6, 'criterion': 'entropy'.** And Training Accuracy for RandomForestClassifier was **74.5%** Test Accuracy for was **50.41%.**

Testing on Random Data:

```
# Generate random data with the same shape and features as X_train
X_train_random = np.random.rand(X_train.shape[0], X_train.shape[1])

print(f"Random Data: {[X_train_random[0]]}")
# Make predictions on the random data
y_pred_random = clf3__.predict([X_train_random[0]])

# Print predictions for the random data
print("Predictions for random data: ", y_pred_random)
```
✓ 0.0s

```
Random Data: [array([0.36107381, 0.76302195, 0.41081088, 0.90317133, 0.65297781,
       0.63784172, 0.52289753, 0.35836302, 0.96985855, 0.78196639,
       0.25221759, 0.63533294, 0.23602128])]
Predictions for random data:  [5]
```

LogisticRegression

Before the Hyperparameter Tuning:

```
clf1 = LogisticRegression()

clf1.fit(X_train, y_train)

# Make predictions on the test set
y_pred2 = clf1.predict(X_test)

# Calculate the test accuracy
accuracy2 = accuracy_score(y_pred2, y_test)

# Calculate the training accuracy
train_accuracy2 = clf1.score(X_train, y_train)

print("Training Accuracy for LogisticRegression: ", train_accuracy2)
print("Test Accuracy for LogisticRegression: ", accuracy2)
```

Here we got Training Accuracy for LogisticRegression as **50.12%** and Test Accuracy for LogisticRegression: as **42.3%**.

After Hyperparameter Tuning:

```
parameters1 = {'penalty' : ['l1','l2','elasticnet','None'],'C':[1,5,10,20,50,75,100]}

clf1 = GridSearchCV(clf1,param_grid=parameters1,cv=5)

clf1.fit(X_train,y_train)

train_accuracy1 = clf1.score(X_train, y_train)

best_params = clf1.best_params_

clf1 =LogisticRegression(C = best_params['C'], penalty = best_params['penalty'])


clf1.fit(X_train,y_train)


y_pred1 = clf1.predict(X_test)
accuracy = accuracy_score(y_pred1,y_test)
print("Training Accuracy for LogisticRegression: ", train_accuracy1)
print("Test Accuracy for LogisticRegression: ", accuracy)
```

After Hyperparameter Tuning the best parameters were: **'C': 50, 'penalty': 'l2.** And Training Accuracy for LogisticRegression was **51.37%** Test Accuracy for was **43.13%.**

Testing on Data:

```
# Generate random data with the same shape and features as X_train
X_train_random = np.random.rand(X_train.shape[0], X_train.shape[1])

print(f"Random Data: {[X_train_random[0]]}")
# Make predictions on the random data
y_pred_random = clf1.predict([X_train_random[0]])

# Print predictions for the random data
print("Predictions for random data: ", y_pred_random)
```
✓ 0.0s

```
Random Data: [array([0.38462709, 0.30344133, 0.74119232, 0.96516443, 0.77662722,
       0.54122864, 0.31798035, 0.58620753, 0.89246497, 0.53142969,
       0.79791426, 0.01602806, 0.27923642])]
Predictions for random data:  [6]
```

Comparison:

Models	Before Hyperparameter			After Hyperparameter			% Test Improvement
Accuracies	Train	Test	Overfit/Underfit	Train	Test	Overfit/Underfit	
DecisionTreeClassifier	100%	43.52%	Overfit	66.97%	46.27%	No	2.75%
RandomForestClassifier	100%	50.10%	Overfit	74.50%	50.41%	No	0.31%
LogisticRegression	51.12%	42.30%	No	51.37%	43.13%	No	0.83%

Conclusion:

The study concludes that hyperparameter tuning significantly reduced overfitting in both the DecisionTreeClassifier and RandomForestClassifier, resulting in slight improvements in test accuracy. The DecisionTreeClassifier's test accuracy increased from **43.52%** to **46.27%**, while the RandomForestClassifier's test accuracy improved marginally from **50.10%** to **50.41%**. LogisticRegression showed a minimal improvement in test accuracy from **42.30%** to **43.13%**, indicating a more stable performance with no overfitting both before and after tuning. Overall, hyperparameter tuning led to better model generalization and slight test performance enhancements.