CO 544 Machine Learning and Data Mining
Lab 03

E/18/224

Mihiranga G.D.R

TODO 1: Discuss how Figure 2 can be used to gain insights into the features that most influence the wine quality and to identify highly correlated features.

The correlation coefficients between related variables in a dataset are displayed in a table called a correlation matrix. It is a helpful statistical method for comprehending the connections between variables and acquiring understanding of the characteristics that have the greatest impact on a target variable.
The degree and direction of the linear link between two variables are measured by the correlation coefficient. A value of 1 denotes a perfect positive correlation, a value of -1 denotes a perfect negative correlation, and a value of 0 denotes no connection.
You can identify the characteristics that have the biggest impact on the target variable by analyzing the correlation matrix. You specifically search for factors that are highly correlated with the target variable, either positively or negatively. A strong positive correlation indicates that the target variable tends to rise together with the value of the characteristic. On the other hand, a strong negative correlation means that the target variable tends to fall when the value of that characteristic rises.

```python
# Get the correlation coefficients of the features with wine quality
wine_quality_correlation = corr_matrix['quality'].drop('quality')
wine_quality_correlation = wine_quality_correlation.abs().sort_values(ascending=False)

# Print the highly correlated features
print(wine_quality_correlation)
```

[16] ✓ 0.0s

```
alcohol                 0.476166
volatile acidity        0.390558
sulphates               0.251397
citric acid             0.226373
total sulfur dioxide    0.185100
density                 0.174919
chlorides               0.128907
fixed acidity           0.124052
pH                      0.057731
free sulfur dioxide     0.050656
residual sugar          0.013732
Name: quality, dtype: float64
```

As given above in the figure we can see alcohol, volatile acidity and sulphates are highly correlated to the wine quality.

TODO 2: What is the most appropriate performance metric to evaluate the model's performance? Briefly explain why.

```python
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, precision_score, f1_score, roc_auc_score, recall_score

# Define the performance metrics to evaluate
metrics = {
    'Accuracy': 'accuracy',
    'Precision': 'precision',
    'F1 Score': 'f1',
    'Area under ROC Curve': 'roc_auc',
    'Recall': 'recall'
}

# Evaluate the model using cross-validation and calculate the performance for each metric
results = {}
for metric_name, scoring in metrics.items():
    scores = cross_val_score(logreg_model, X, y, cv=5, scoring=scoring)
    results[metric_name] = scores.mean()

# Print the performance results for each metric
for metric_name, score in results.items():
    print(f'{metric_name}: {score}')
```

[65] ✓ 1.8s

```
Accuracy: 0.7292104231974921
Precision: 0.7569707978178475
F1 Score: 0.7396385830380772
Area under ROC Curve: 0.8122596457283846
Recall: 0.7403508771929824
```

Since the area under the ROC curve (AUC-ROC) is the best fit statistic, it can be utilized to evaluate how well the model performs on the wine quality dataset. Because it provides a balanced evaluation. AUC-ROC considers both the true positive rate and the false positive rate. In situations when there is an imbalance in the class distribution, it offers a fair assessment of the model's capacity to discriminate between positive and negative occurrences.

TODO 3: Eliminate irrelevant features that may negatively affect the performance of the model and discuss how that may affect the model's performance using your chosen metric.

```python
# Step 1: Evaluate initial model performance using area under roc curve
initial_predictions = logreg_model.predict_proba(X)[:, 1]
initial_auc_roc = roc_auc_score(y, initial_predictions)

# Step 2: Identify features that may negatively affect the performance
feature_importance = pd.DataFrame({'Feature': X.columns, 'Coefficient': logreg_model.coef_[0]})
negative_features = feature_importance.loc[feature_importance['Coefficient'] < 0, 'Feature']

# Step 3: Remove the negatively affecting features
X_new = X.drop(negative_features, axis=1)

# Step 4: Retrain and evaluate updated model performance
X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.2, random_state=42)
updated_model = LogisticRegression()
updated_model.fit(X_train, y_train)

updated_predictions = updated_model.predict_proba(X_test)[:, 1]
updated_auc_roc = roc_auc_score(y_test, updated_predictions)

print("Features with Negative Impact on Performance:")
print(negative_features.tolist())
print("Initial AUC-ROC:", initial_auc_roc)
print("Updated AUC-ROC:", updated_auc_roc)
```

[94] ✓ 0.1s

```
Features with Negative Impact on Performance:
['volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'total sulfur dioxide', 'density', 'pH']
Initial AUC-ROC: 0.8175281393447777
Updated AUC-ROC: 0.7772296842188676
```

TODO 4: Try different test/train split ratios and evaluate the model performance in terms of your chosen metric to find the optimal split ratio.

```python
# Define different split ratios to try
split_ratios = [0.6, 0.7, 0.8, 0.9]

best_split_ratio = None
best_auc_roc = 0

# Iterate over different split ratios
for split_ratio in split_ratios:
    # Split the data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1 - split_ratio, random_state=42)

    # Train the logistic regression model
    model = LogisticRegression()
    model.fit(X_train, y_train)

    # Make predictions on the test set
    predictions = model.predict_proba(X_test)[:, 1]

    # Calculate the AUC-ROC score
    auc_roc = roc_auc_score(y_test, predictions)

    # Check if this split ratio yields a better AUC-ROC score
    if auc_roc > best_auc_roc:
        best_auc_roc = auc_roc
        best_split_ratio = split_ratio

print("Best Split Ratio:", best_split_ratio)
print("Best AUC-ROC Score:", best_auc_roc)
```

[95] ✓ 0.5s

```
Best Split Ratio: 0.9
Best AUC-ROC Score: 0.8323863636363638
```

According to this figure we can see the best split ratio is 0.9 (test Data = 10% and Train data 90%).

TODO 5: Perform hyperparameter tuning for the logistic regression model using grid search to optimize the model's performance with the dataset. Try different hyperparameters, such as C (regularization strength), penalty (regularization type), solver (optimization algorithm), and max iter (maximum number of iterations to converge). Compare the results with our initial model and choose the best set of hyperparameters.

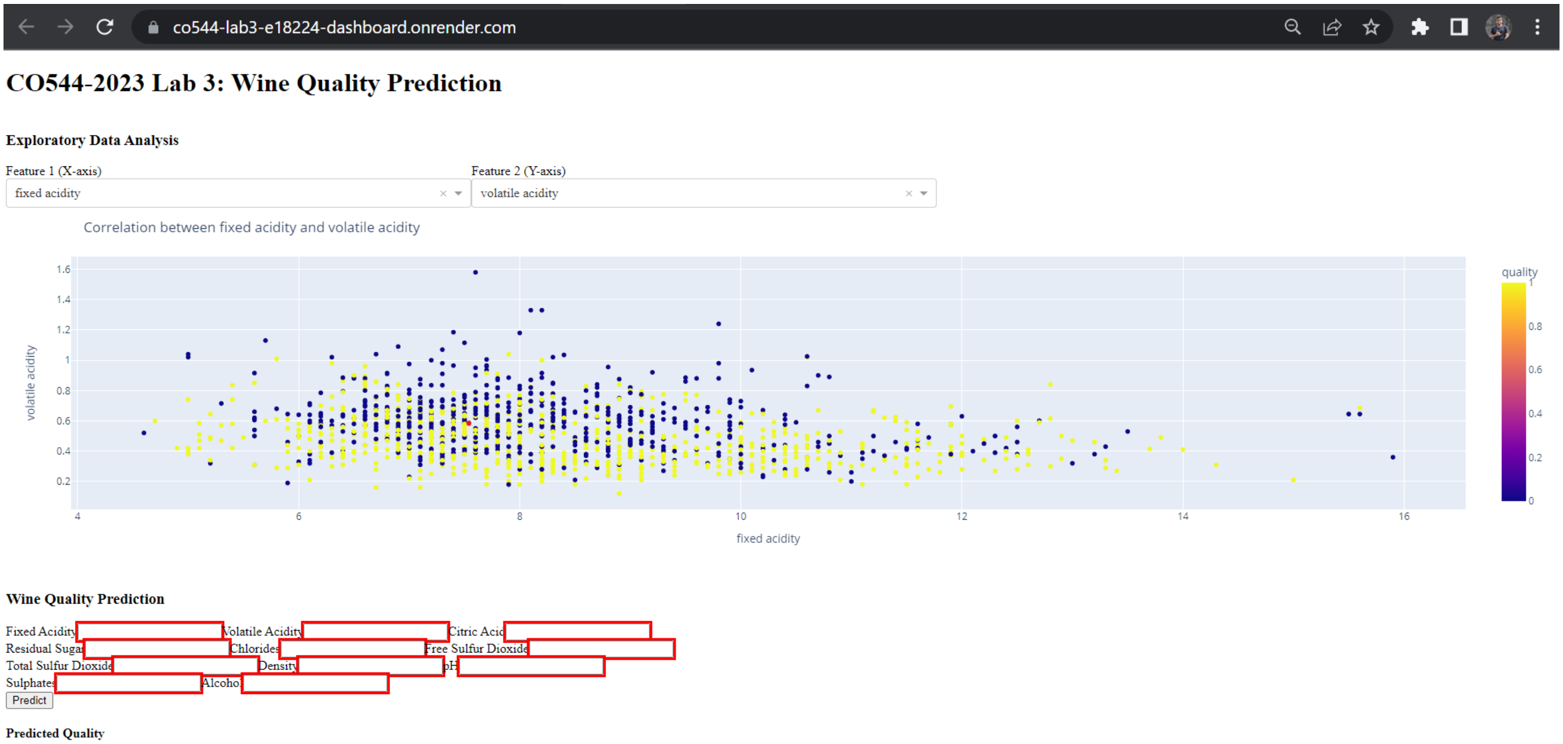
```
# performance metrics of the existing model
# Accuracy: 0.7292104231974921
# Precision: 0.7569707978178475
# F1 Score: 0.7396385830380772
# Area under ROC Curve: 0.8122596457283846
# Recall: 0.7403508771929824

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, f1_score, roc_auc_score, recall_score

newModel = LogisticRegression()

hyper_parameters = {
    'C': [0.1, 0.9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['newton-cg', 'lbfgs', 'liblinear'],
    'max_iter': [100, 500, 1000]
}

clf = GridSearchCV(newModel, param_grid=hyper_parameters, scoring='roc_auc', cv=10)
clf.fit(X_train, y_train)

bestParams = clf.best_params_

best_logreg_model = LogisticRegression(**bestParams)
best_logreg_model.fit(X_train, y_train)

# Predict the labels of the test set
y_pred = best_logreg_model.predict(X_test)
```

```
# Predict the labels of the test set
y_pred = best_logreg_model.predict(X_test)

# Compute the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
# Compute the precision of the model
precision = precision_score(y_test, y_pred)
# Compute the recall of the model
recall = recall_score(y_test, y_pred)
# Compute the F1 score of the model
f1 = f1_score(y_test, y_pred)
y_pred = best_logreg_model.predict_proba(X_test)[:, 1]
# Calculate the AUC-ROC score
auc_roc = roc_auc_score(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
print("Area under ROC curve:", auc_roc)
```

[125] ✓ 2m 47.2s

```
Accuracy: 0.74375
Precision: 0.7701149425287356
Recall: 0.7613636363636364
F1 score: 0.7657142857142857
Area under ROC curve: 0.836489898989899
```

TODO 5: Deploy the dashboard on a cloud platform to make it accessible to a wider audience.



REFERENCES:

Repo Link for the source code:

https://github.com/RavinduMihiranga/co544-2023-lab3-interactive-python-dashboards

Link for website:

https://co544-lab3-e18224-dashboard.onrender.com