Comparative Analysis of Predictive Models in Wine Quality Assessment

Dataset Description:

The Wine Quality dataset is publicly available on Kaggle. It is provided in CSV (Comma-Separated Values) format, ensuring compatibility with a wide range of data analysis tools and programming languages such as Python. Each row in the dataset represents a single observation of wine, with columns corresponding to various physicochemical properties such as acidity levels, sugar content, sulfur dioxide concentration, and alcohol percentage. Additionally, the dataset includes a quality rating assigned by human tasters, providing a subjective measure of the wine's overall quality.

The Wine Quality dataset serves as a valuable resource for researchers, data scientists, and enthusiasts interested in exploring the relationship between physicochemical properties and the perceived quality of wine. Originating from Portugal, this dataset contains information on both red and white variants of wine, providing a diverse and representative sample of the wine industry. With a total of 6497 instances, each representing a unique sample of wine, the dataset offers ample opportunities for analysis and modeling.

The Wine Quality dataset contains the following features:

Fixed Acidity: The amount of non-volatile acids in the wine.

Volatile Acidity: The amount of acetic acid in the wine, which can lead to an unpleasant vinegar taste.

Citric Acid: The amount of citric acid in the wine, which adds freshness and flavor.

Residual Sugar: The amount of sugar remaining after fermentation (g/dm^3).

Chlorides: The amount of salt in the wine (g/dm^3).

Free Sulfur Dioxide: The amount of free sulfur dioxide in the wine, which prevents microbial growth and oxidation (mg/dm^3).

Total Sulfur Dioxide: The total amount of sulfur dioxide in the wine, including both free and bound forms (mg/dm^3).

pH: The pH level of the wine, indicating its acidity or alkalinity.

Sulfates: The amount of sulfur dioxide added to the wine (g/dm^3).

Alcohol: The alcohol content of the wine (% vol).

Quality: A subjective rating of wine quality, scored between 0 and 10 by human tasters.

ID: An identifier column assigned to each instance in the dataset, serving as a unique reference.

Data Preprocessing

For logistic regression, SVM, and neural network modeling of the Wine Quality dataset, the following preprocessing steps were undertaken to ensure data readiness and optimal model performance:

- Column Removal: The 'Id' column, assumed to be an identifier with no predictive value, was removed from the dataset using the drop() function in pandas.
- Feature Extraction: The dataset was split into features (independent variables) and the target variable (dependent variable). Features included physicochemical properties like acidity, sugar content, and alcohol percentage, while the target variable was the quality rating of the wine.
- Label Adjustment: For logistic regression and SVM, adjustments were made to ensure that labels were zero-indexed. This step ensures compatibility with the algorithm's assumptions and facilitates easier interpretation of results.
- Handling Class Imbalance: Addressing potential class imbalance, Synthetic Minority Oversampling Technique (SMOTE) was applied. SMOTE generates synthetic samples for the minority class, thus balancing the class distribution and mitigating biases towards the majority class.
- Data Splitting: The dataset was partitioned into training and test sets using the train_test_split function from the sklearn.model_selection module. This step enables model training on a subset of the data and evaluation on unseen data, assessing generalization performance.
- Feature Scaling: To ensure that all features have a similar scale, StandardScaler from the sklearn.preprocessing module was utilized to standardize the data. Standardization transforms the features to have a mean of zero and a standard deviation of one, which aids in model convergence and performance, particularly for algorithms sensitive to feature scales, such as logistic regression and SVM.

These preprocessing steps collectively prepare the dataset for subsequent model training and evaluation, ensuring data readiness, reducing biases, and enhancing model performance. Each step is crucial in ensuring the robustness and reliability of the machine learning models applied to the Wine Quality dataset.

Algorithms Used:

- Logistic Regression
- Support Vector Machine
- Neural Network

Logistic Regression

Description

Logistic Regression is a statistical method used for binary classification tasks, predicting the probability of a binary outcome based on one or more predictor variables. In our context of wine quality prediction, Logistic Regression estimates the probability that a given wine sample belongs to a specific quality category based on its physicochemical properties. It models the relationship between the independent variables and the probability of the target variable using the logistic function. Logistic Regression is particularly well-suited for binary classification tasks and is widely used due to its simplicity, interpretability, and efficiency.

Performance Metric and Data Splitting:

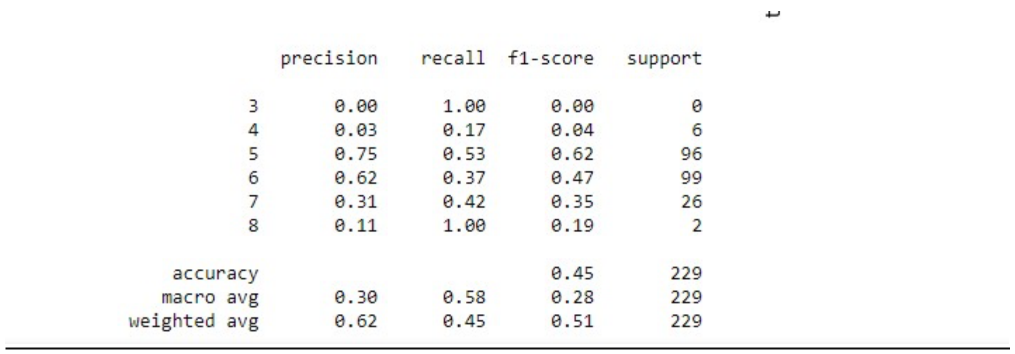
The primary performance metric used for evaluating the Logistic Regression model is the classification report. This report offers detailed insights into the model's performance by calculating precision, recall, F1-score, and support for each class in the dataset, along with a weighted average across all classes. These metrics provide a comprehensive understanding of the model's ability to correctly classify instances of each quality category based on their physicochemical properties. The classification report is generated using the `classification_report` function from sklearn.metrics, which takes the true labels of the test set (`y_test`) and the predicted labels generated by the Logistic Regression model (`y_pred`) as inputs. Additionally, the `zero_division=1` parameter handles scenarios where there may be a division by zero in the computation of precision, recall, or F1-score, setting the corresponding metric to 1 when necessary.

The confusion matrix, visualized using a heatmap, further enhances the interpretation of the model's performance by providing a graphical representation of its predictions in terms of true positives, false positives, true negatives, and false negatives. This visualization aids stakeholders in understanding the model's classification results and identifying areas for improvement.

The dataset is split into features (X) and target variable (y). The data is then split into training and testing sets using train_test_split with 80% for training and 20% for testing. The features are standardized using StandardScaler to ensure consistent scale across features. Imbalanced classes are handled by oversampling the minority class using RandomOverSampler. Finally, the Logistic Regression model is trained on the oversampled training data, and its performance is evaluated on the test set using classification_report and confusion_matrix. The confusion matrix is visualized using a heatmap to provide a graphical representation of the model's predictions.

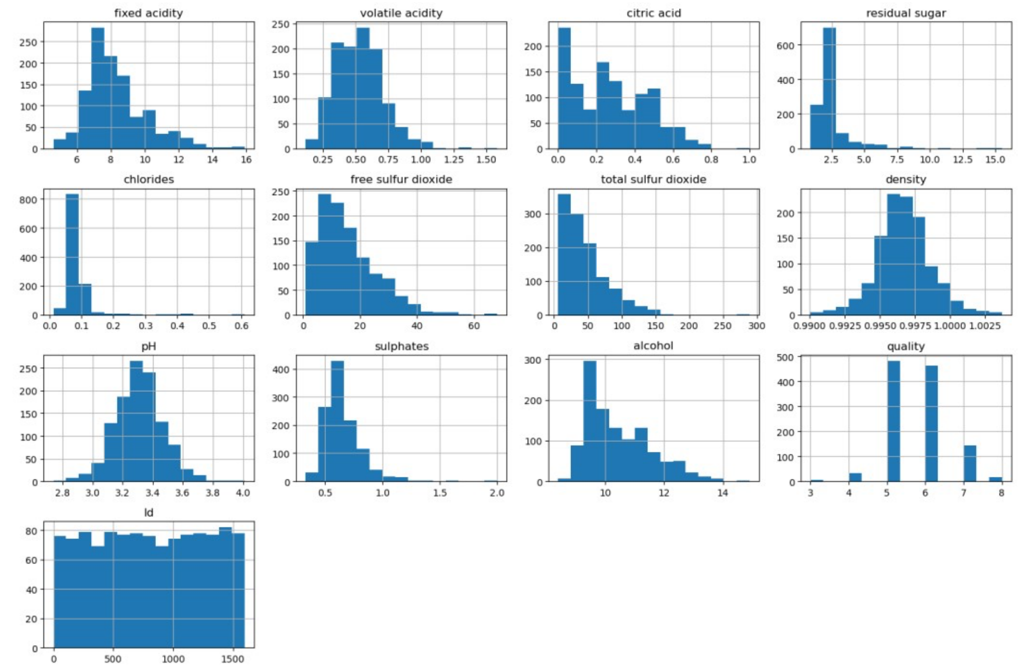
Results and Plots:

This image shows a classification report for a logistic regression model with metrics such as precision, recall, f1-score, and support for classes labeled 3 to 8. The accuracy of the model is shown to be 0.45. Macro average and weighted average metrics are also provided for precision, recall, and f1-score.

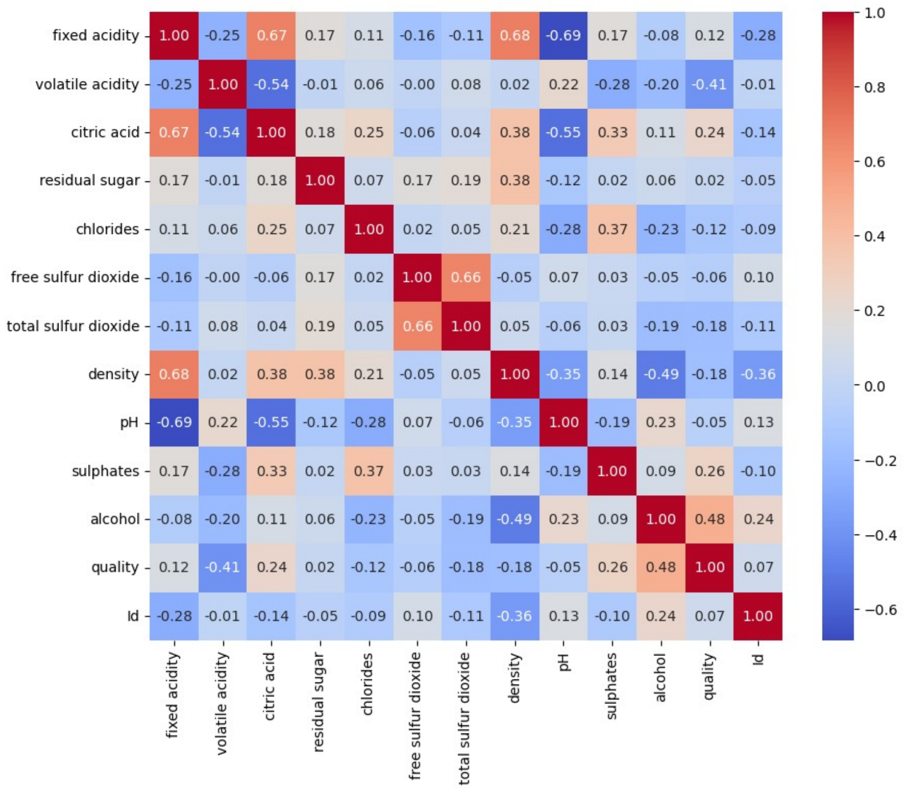


```
              precision    recall  f1-score   support

           3       0.00      1.00      0.00         0
           4       0.03      0.17      0.04         6
           5       0.75      0.53      0.62        96
           6       0.62      0.37      0.47        99
           7       0.31      0.42      0.35        26
           8       0.11      1.00      0.19         2

    accuracy                           0.45       229
   macro avg       0.30      0.58      0.28       229
weighted avg       0.62      0.45      0.51       229
```

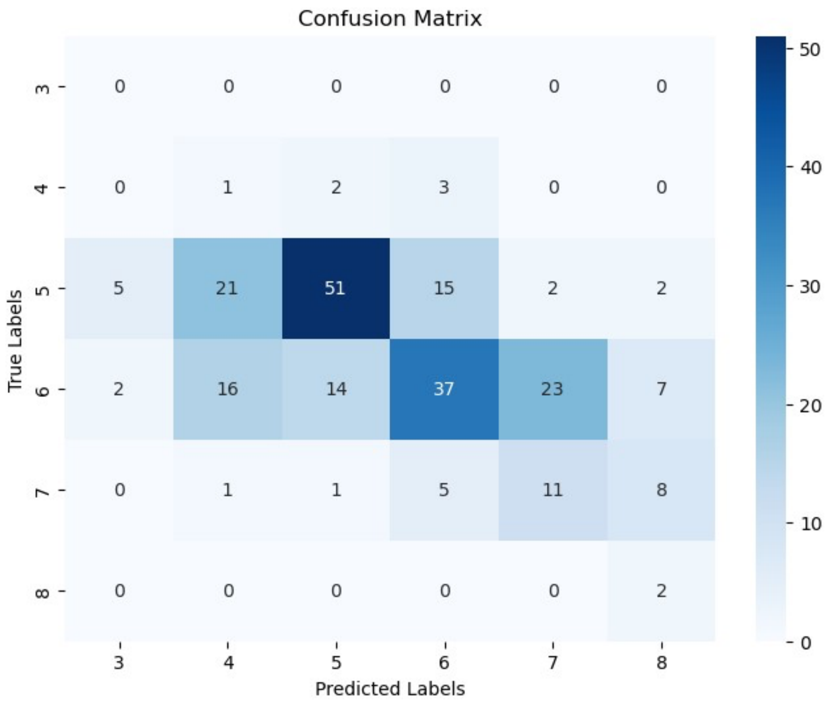
This image displays histograms for various features likely used in the logistic regression model, including fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, and quality. Each histogram shows the distribution of values for these features within the dataset.



The image contains a heatmap showing the correlation coefficients between different features used in the model, such as fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, and quality. The color scale ranges from blue (negative correlation) to red (positive correlation), indicating the strength and direction of the relationships between variables.



This image shows a confusion matrix for the logistic regression model, illustrating the true labels versus the predicted labels for classes ranging from 3 to 8. This matrix helps in visualizing the accuracy of predictions across different classes, highlighting where the model performs well and where it confuses between classes.



Support Vector Machine:

Description

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that separates different classes in the feature space while maximizing the margin between them. SVM aims to find the hyperplane that best distinguishes between different wine quality categories based on their physicochemical properties. The optimal hyperplane is the one that maximizes the margin, which is the distance between the hyperplane and the closest data points from each class, known as support vectors. SVM can handle both linearly separable and non-linearly separable data by using kernel functions to map the input features into a higher-dimensional space where the classes become separable.

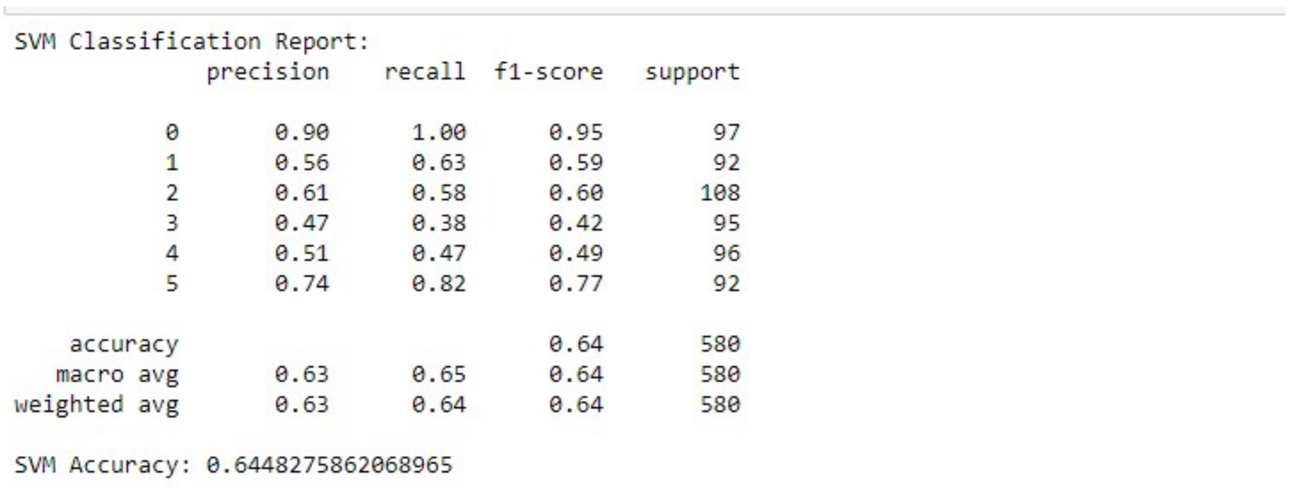
Performance Metric and Data Splitting:

In the SVM implementation, the model's performance is evaluated primarily using the classification report, a comprehensive summary of the model's performance across different wine quality categories. This report is generated using the classification_report function from the sklearn.metrics module, which calculates precision, recall, F1-score, and support for each class in the dataset based on the true and predicted labels. Precision represents the proportion of true positive predictions among all positive predictions, while recall measures the proportion of true positive predictions among all actual positive instances. The F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. Support indicates the number of occurrences of each class in the dataset. By analyzing these metrics, stakeholders gain insights into the SVM model's ability to correctly classify instances of each wine quality category based on their physicochemical properties.

Data splitting is a crucial step in model training and evaluation, ensuring that the SVM model is trained on a subset of the data and evaluated on unseen data for an unbiased assessment of its performance. The dataset was split into training and testing sets using the train_test_split function from the sklearn.model_selection module. This function divides the dataset into training and testing sets, with 80% of the data allocated for training and 20% reserved for testing. The test_size=0.2 parameter specifies the proportion of the dataset to include in the testing set, while the random_state=42 parameter sets the random seed for reproducibility. This splitting strategy ensures that the SVM model is trained on a sufficiently large dataset while still having unseen data for evaluation, facilitating an unbiased assessment of its performance.

Results:

The report includes metrics such as precision, recall, f1-score, and support for several classes labeled from 0 to 5. The SVM model has an accuracy of approximately 0.6448.



```
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.90      1.00      0.95        97
           1       0.56      0.63      0.59        92
           2       0.61      0.58      0.60       108
           3       0.47      0.38      0.42        95
           4       0.51      0.47      0.49        96
           5       0.74      0.82      0.77        92

    accuracy                           0.64       580
   macro avg       0.63      0.65      0.64       580
weighted avg       0.63      0.64      0.64       580

SVM Accuracy: 0.6448275862068965
```

Neural Network:

Description

Neural Networks are a class of deep learning algorithms inspired by the structure and functioning of the human brain. They consist of interconnected layers of nodes (neurons) and are capable of automatically learning complex patterns and representations from raw data. In our context, Neural Networks learn to classify wine samples into different quality categories based on their physicochemical properties. The input layer receives the features of the wine samples, which are then passed through one or more hidden layers before reaching the output layer, which provides the predictions. Each neuron in the network performs a linear transformation followed by a non-linear activation function, allowing the network to capture intricate relationships in the data.

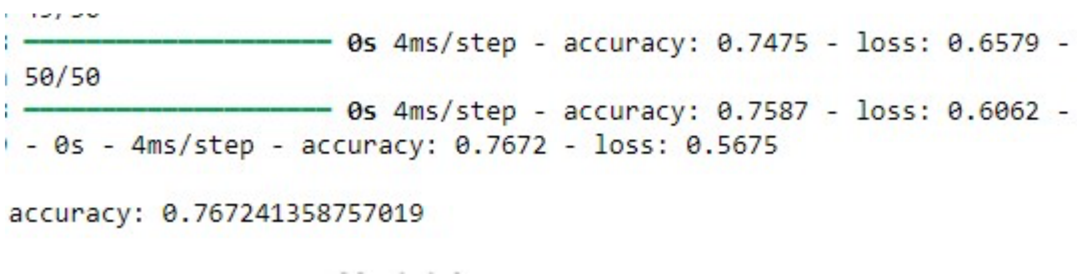
Performance Metric and Data Splitting:

In the Neural Network implementation, the primary performance metric used for evaluation is the accuracy score, which measures the proportion of correctly classified instances in the test set. This metric is computed using the evaluate method of the trained model, which takes the test features (X_test_scaled) and labels (y_test) as inputs. After training the Neural Network model on the training set, the evaluate method is called to assess its performance on the test set. The evaluate method returns the test loss and accuracy, with the latter representing the model's overall performance in correctly classifying wine samples into different quality categories. This accuracy score provides stakeholders with a straightforward measure of the model's classification capabilities, indicating the percentage of correctly predicted instances out of the total number of instances in the test set.

Similarly, the dataset was split into training and testing sets using the train_test_split function from the sklearn.model_selection module. The dataset was divided with 80% of the data allocated for training and 20% reserved for testing. This splitting strategy ensures that the Neural Network model is trained on a sufficiently large dataset while still having unseen data for evaluation. The test_size=0.2 parameter specifies the proportion of the dataset to include in the testing set, while the random_state=42 parameter sets the random seed for reproducibility. By splitting the data in this manner, stakeholders can train the Neural Network model on a subset of the data and evaluate its performance on unseen data, facilitating an unbiased assessment of its classification capabilities.

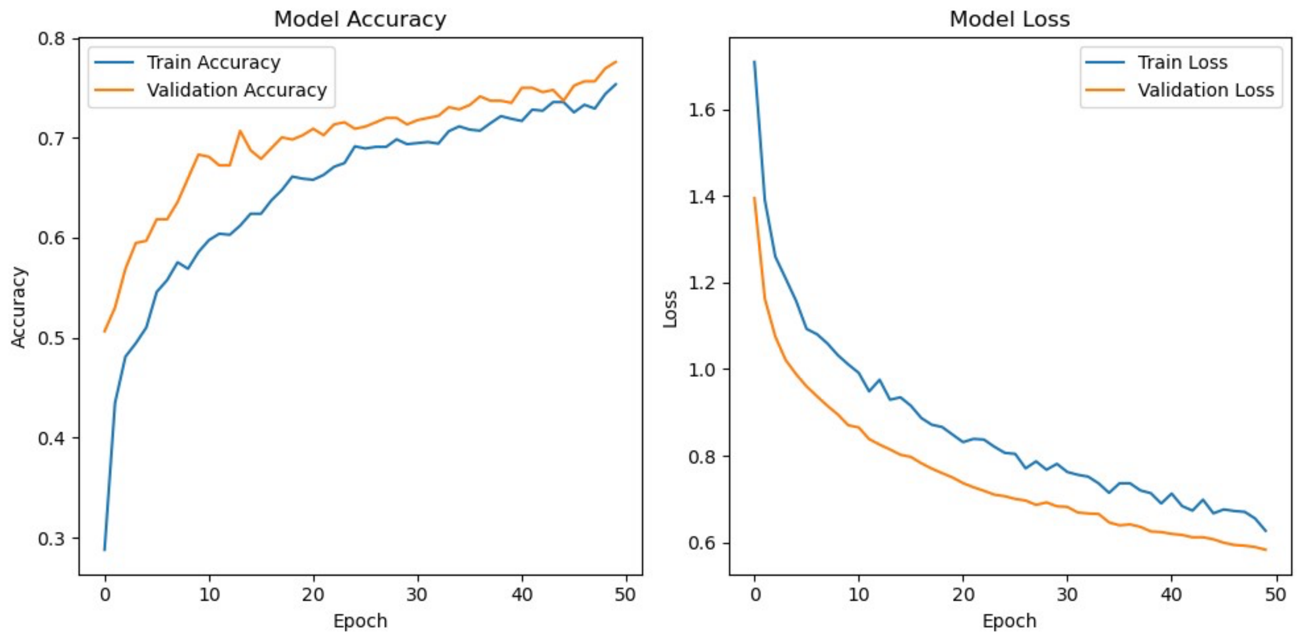
Results and plots:

The final calculated accuracy for is 0.7672. The Neural Network achieved the highest accuracy.



```
━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7475 - loss: 0.6579 -
50/50
━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7587 - loss: 0.6062 -
- 0s - 4ms/step - accuracy: 0.7672 - loss: 0.5675

accuracy: 0.767241358757019
```

This image contains two-line graphs plotting the accuracy and loss of the model over 50 epochs of training. The left graph shows "Model Accuracy," comparing training accuracy (orange line) and validation accuracy (blue line). The right graph depicts "Model Loss," comparing training loss (orange line) and validation loss (blue line). Both graphs illustrate an increase in accuracy and a decrease in loss over time, indicating that the model is learning effectively from the training process.

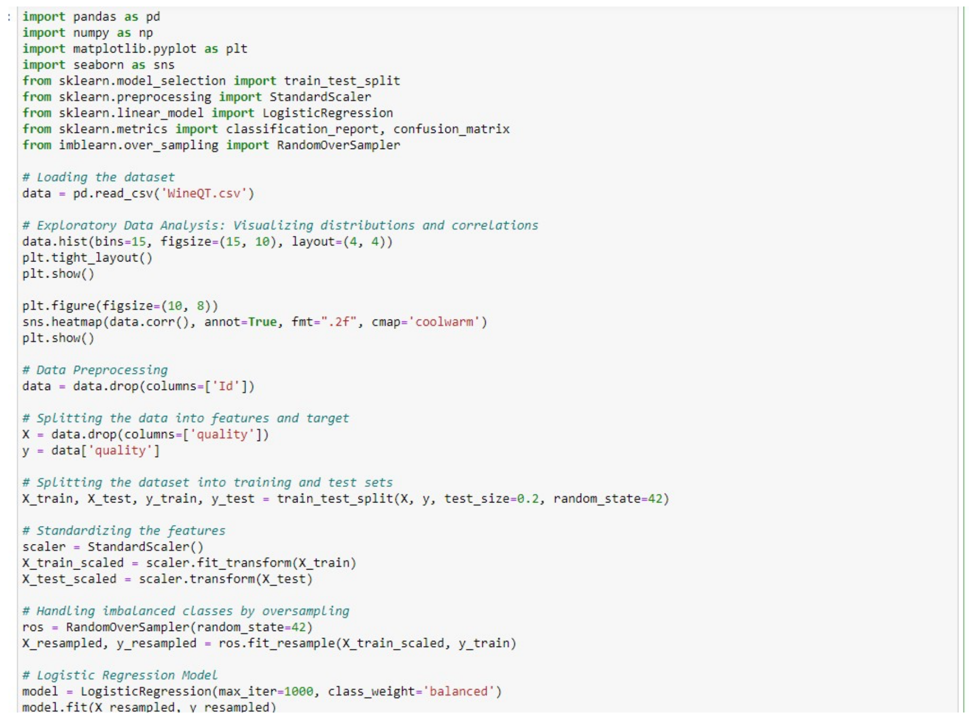


Code:

Logistic Regression

import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns from sklearn.model_selection import train_test_split from sklearn.preprocessing import StandardScaler from sklearn.linear_model import LogisticRegression from sklearn.metrics import classification_report, confusion_matrix from imblearn.over_sampling import RandomOverSampler

```
# Loading the dataset data =
pd.read_csv('WineQT.csv')
# Exploratory Data Analysis: Visualizing distributions and correlations
data.hist(bins=15, figsize=(15, 10), layout=(4, 4)) plt.tight_layout()
plt.show() plt.figure(figsize=(10, 8)) sns.heatmap(data.corr(),
annot=True, fmt=".2f", cmap='coolwarm') plt.show() # Data
Preprocessing data = data.drop(columns=['Id']) # Splitting the data into
features and target X = data.drop(columns=['quality']) y =
data['quality']
# Splitting the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Standardizing the features scaler
= StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test) #
Handling imbalanced classes by oversampling
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X_train_scaled, y_train)
# Logistic Regression Model model =
LogisticRegression(max_iter=1000, class_weight='balanced')
model.fit(X_resampled, y_resampled) y_pred =
model.predict(X_test_scaled)
# Model Evaluation print(classification_report(y_test,
y_pred, zero_division=1)) print(confusion_matrix(y_test,
y_pred)) # Visualizing the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True,
fmt="d", cmap='Blues', xticklabels=np.unique(y),
yticklabels=np.unique(y)) plt.xlabel('Predicted Labels')
plt.ylabel('True Labels') plt.title('Confusion Matrix')
plt.show()
```



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import RandomOverSampler

# Loading the dataset
data = pd.read_csv('WineQT.csv')

# Exploratory Data Analysis: Visualizing distributions and correlations
data.hist(bins=15, figsize=(15, 10), layout=(4, 4))
plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.show()

# Data Preprocessing
data = data.drop(columns=['Id'])

# Splitting the data into features and target
X = data.drop(columns=['quality'])
y = data['quality']

# Splitting the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Handling imbalanced classes by oversampling
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X_train_scaled, y_train)

# Logistic Regression Model
model = LogisticRegression(max_iter=1000, class_weight='balanced')
model.fit(X_resampled, y_resampled)
```

Support Vector Machine

```
import pandas as pd import
numpy as np
from sklearn.model_selection import train_test_split from
sklearn.preprocessing import StandardScaler from
imblearn.over_sampling import SMOTE from sklearn.svm
import SVC from sklearn.metrics import classification_report,
accuracy_score
# Load and prepare the dataset data
= pd.read_csv('WineQT.csv')
data.drop(columns='Id',
inplace=True)  # Assuming 'Id' is a
column to be dropped

# Splitting the dataset into features and labels
X = data.drop('quality', axis=1) y =
data['quality']
# Adjust labels to be zero-indexed if necessary y
-= y.min()
# Handling imbalanced classes with SMOTE smote
= SMOTE(random_state=42)
```

X_res, y_res = smote.fit_resample(X, y)

Splitting data into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42)

# Feature scaling scaler

= StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

# SVM model svm_model = SVC(kernel='linear',

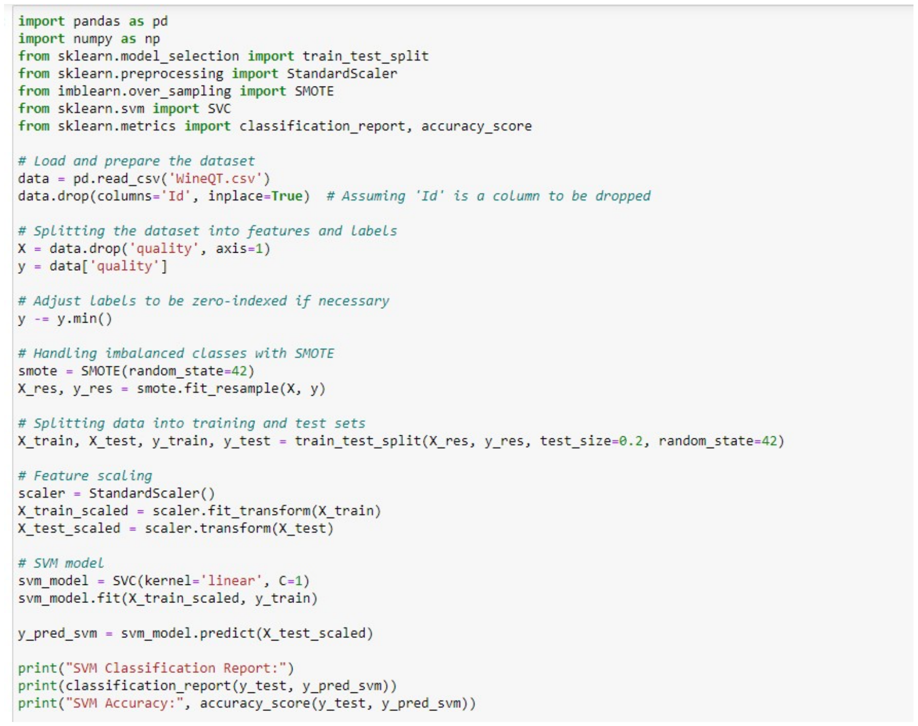
C=1) svm_model.fit(X_train_scaled, y_train)

y_pred_svm = svm_model.predict(X_test_scaled)

print("SVM Classification Report:")

print(classification_report(y_test, y_pred_svm)) print("SVM

Accuracy:", accuracy_score(y_test, y_pred_svm))



```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score

# Load and prepare the dataset
data = pd.read_csv('WineQT.csv')
data.drop(columns='Id', inplace=True)  # Assuming 'Id' is a column to be dropped

# Splitting the dataset into features and labels
X = data.drop('quality', axis=1)
y = data['quality']

# Adjust labels to be zero-indexed if necessary
y -= y.min()

# Handling imbalanced classes with SMOTE
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)

# Splitting data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# SVM model
svm_model = SVC(kernel='linear', C=1)
svm_model.fit(X_train_scaled, y_train)

y_pred_svm = svm_model.predict(X_test_scaled)

print("SVM Classification Report:")
print(classification_report(y_test, y_pred_svm))
print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
```

Neural Network

import pandas as pd import numpy as np from

sklearn.model_selection import train_test_split from

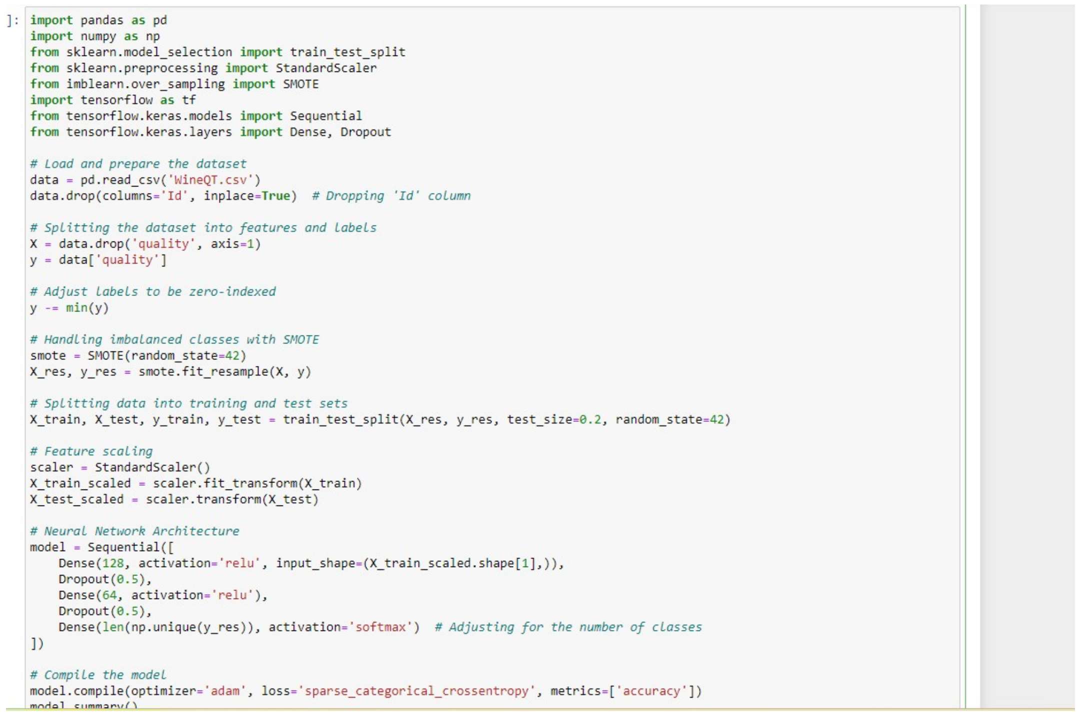
sklearn.preprocessing import StandardScaler from

imblearn.over_sampling import SMOTE import

tensorflow as tf from tensorflow.keras.models

```
import Sequential from tensorflow.keras.layers
import Dense, Dropout

# Load and prepare the dataset data
= pd.read_csv('WineQT.csv')
data.drop(columns='Id', inplace=True)  # Dropping 'Id' column
# Splitting the dataset into features and labels
X = data.drop('quality', axis=1) y =
data['quality']
# Adjust labels to be zero-indexed y
-= min(y)
# Handling imbalanced classes with SMOTE smote
= SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)
# Splitting data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42)
# Feature scaling scaler
= StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Neural Network Architecture model =
Sequential([
    Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(len(np.unique(y_res)), activation='softmax')  # Adjusting for the number of classes
])
# Compile the model model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy']) model.summary() # Train the model history = model.fit(X_train_scaled, y_train,
epochs=50, validation_split=0.2, batch_size=32, verbose=1)
# Evaluate the model on the test set test_loss, test_acc =
model.evaluate(X_test_scaled, y_test, verbose=2) print('\nTest
accuracy:', test_acc)
# Visualizing Training and Validation Metrics import
matplotlib.pyplot as plt
```

```
# Accuracy plot plt.figure(figsize=(10,
5)) plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy') plt.xlabel('Epoch') plt.ylabel('Accuracy')
plt.legend()
# Loss plot plt.subplot(1,
2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss') plt.xlabel('Epoch') plt.ylabel('Loss')
plt.legend() plt.tight_layout() plt.show()
```



```
]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Load and prepare the dataset
data = pd.read_csv('WineQT.csv')
data.drop(columns='Id', inplace=True)  # Dropping 'Id' column

# Splitting the dataset into features and labels
X = data.drop('quality', axis=1)
y = data['quality']

# Adjust labels to be zero-indexed
y -= min(y)

# Handling imbalanced classes with SMOTE
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)

# Splitting data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Neural Network Architecture
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(len(np.unique(y_res)), activation='softmax')  # Adjusting for the number of classes
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.summary()
```