Wine Quality Prediction Project

Predicting wine quality can be a fascinating project! Typically, this involves using machine learning techniques to analyze features of the wine (like acidity, pH, alcohol content, etc.) and predict its quality rating based on historical data.

Here’s a step-by-step outline to guide you through the process in Python:

1. Dataset

First, you need a dataset that contains information about various wines along with their quality ratings. The most commonly used dataset for this task is the Wine Quality dataset from the UCI Machine Learning Repository.

Wine Quality Dataset (Red Wine)

2. Loading the Data

Use Pandas to load and inspect the dataset:

```
import pandas as pd


# Load the dataset (change file path as necessary)

df = pd.read_csv('winequality-red.csv', sep=';')
```

3. Exploratory Data Analysis (EDA)

Explore the dataset to understand its structure and characteristics:

```
# Display the first few rows of the dataset

print(df.head())


# Check for missing values

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

```
# Summary statistics
print(df.describe())
```

4. Data Preprocessing

Prepare the data for modeling:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


# Separate features and target variable
X = df.drop('quality', axis=1)
y = df['quality']


# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

5.Model Selection and Training

Choose a machine learning model and train it on the data:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
# Initialize the model
model = LinearRegression()

# Train the model
model.fit(X_train_scaled, y_train)

# Predictions
train_preds = model.predict(X_train_scaled)
test_preds = model.predict(X_test_scaled)

# Evaluate model performance
train_rmse = mean_squared_error(y_train, train_preds, squared=False)
test_rmse = mean_squared_error(y_test, test_preds, squared=False)

print(f'Train RMSE: {train_rmse}')
print(f'Test RMSE: {test_rmse}')
```

6. Model Evaluation

Evaluate the model's performance using appropriate metrics:

```
from sklearn.metrics import r2_score

# R-squared score
train_r2 = r2_score(y_train, train_preds)
test_r2 = r2_score(y_test, test_preds)

print(f'Train R^2 score: {train_r2}')
print(f'Test R^2 score: {test_r2}')
```

7. Hyperparameter Tuning (Optional)

If using models that have hyperparameters (like Random Forests or Gradient Boosting), consider tuning them using techniques like GridSearchCV or RandomizedSearchCV.

8. Conclusion

Summarize your findings and discuss the performance of the model. You can also visualize results using plots like feature importance (for tree-based models) or residual plots.

Additional Tips:

Feature Engineering: Experiment with creating new features or transforming existing ones to potentially improve model performance.

Model Selection: Try different algorithms (e.g., Random Forests, Gradient Boosting, SVMs) to see which one performs best for your dataset.

Cross-validation: Use cross-validation for more robust model evaluation, especially if your dataset is relatively small.

SUBMITTED BY

PUSHPANSH PANDEY