Wine Quality Prediction Using KDD Methodology

Jayanth Kalyanam
San Jose State University
jayanth.kalyanam@sjsu.edu

October 18, 2024

Abstract

This paper presents a machine learning approach to predicting the quality of red wines using the KDD (Knowledge Discovery in Databases) methodology. The dataset consists of 1,599 red wine samples, each described by 11 features. Several models were trained, including Logistic Regression, K-Nearest Neighbors, Decision Tree, Random Forest, Support Vector Machine, and Gradient Boosting. The Random Forest model, after optimization, achieved an accuracy of 81%, with the most important features being density_diff, sulphates, and alcohol content. This paper also includes the code used to implement each phase of the project along with the relevant visualizations, such as the ROC curve, confusion matrix, and feature importance plot.

1 Introduction

Wine quality is traditionally evaluated through sensory analysis by experts, which can be subjective and costly. In this project, we apply machine learning techniques to predict wine quality based on physicochemical properties. The project follows the KDD (Knowledge Discovery in Databases) methodology to ensure a structured approach, from data selection to knowledge representation. Our goal is to build an accurate model for predicting wine quality and provide insights into the chemical properties that most influence wine quality.

2 KDD Methodology

The **Knowledge Discovery in Databases (KDD)** methodology consists of the following phases:

- **Data Selection**: Identifying and selecting the relevant dataset.
- **Data Preprocessing**: Cleaning, handling missing values, and scaling the data.

- **Data Transformation**: Creating new features or modifying existing features to enhance predictive power.
- **Data Mining**: Applying machine learning algorithms to build predictive models.
- **Pattern Evaluation**: Evaluating model performance through metrics and visualizations.
- **Knowledge Representation**: Interpreting results and providing insights based on the model's performance.

3 Phase 1: Data Selection

The dataset used in this study is the Red Wine Quality dataset from the UCI Machine Learning Repository. It contains 1,599 samples, each described by 11 physicochemical features, and the target variable is the wine quality score.

Features:

- Fixed Acidity
- Volatile Acidity
- Citric Acid
- Residual Sugar
- Chlorides
- Free Sulfur Dioxide
- Total Sulfur Dioxide
- Density
- pH
- Sulphates
- Alcohol

Data Selection Code:

Listing 1: Loading the Dataset

```
import pandas as pd

# Load the Wine Quality dataset
url = 'https://archive.ics.uci.edu/ml/machine-learning-
    databases/wine-quality/winequality-red.csv'
data = pd.read_csv(url, sep=';')

```

```
# Display the first five rows
data.head()

# Check the shape of the dataset
print(f"Dataset shape: {data.shape}")
```

The target variable, wine quality, was converted into a binary classification task:

- **Good Quality**: Quality score ≥ 6
- **Bad Quality**: Quality score < 6

4 Phase 2: Data Preprocessing

The dataset was preprocessed by handling missing values, scaling features, and creating a new feature called *density_diff* to enhance the predictive power.

4.1 Handling Missing Values

The dataset had no missing values, so no imputation was necessary.

4.2 Scaling and Feature Engineering

All features were scaled using the **RobustScaler** to account for the presence of outliers. Additionally, we created the following new feature:

$$density\_diff = density - \left(\frac{alcohol}{100}\right)$$

Data Preprocessing Code:

Listing 2: Data Preprocessing and Feature Engineering

```
from sklearn.preprocessing import RobustScaler

# Initialize the scaler
scaler = RobustScaler()

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

# Fit and transform the features
X_scaled = scaler.fit_transform(X)

# Binarize the target variable
y_binary = y.apply(lambda x: 1 if x >= 6 else 0)

```

```
# Create a new feature 'density_diff'
data['density_diff'] = data['density'] - data['alcohol'] *
    0.01

# Update features and apply scaling
X = data.drop('quality', axis=1)
X_scaled = scaler.fit_transform(X)

# Split into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y_binary, test_size=0.2, random_state=42,
        stratify=y_binary
)
```

5 Phase 3: Data Transformation

In this phase, we ensured that the data was ready for machine learning by scaling all features and transforming the target variable into a binary classification task. No further transformations were applied beyond feature engineering and scaling.

6 Phase 4: Data Mining

In the data mining phase, we trained several machine learning models on the processed data to predict wine quality. The models used were:

- Logistic Regression
- K-Nearest Neighbors
- Decision Tree
- Random Forest
- Support Vector Machine
- Gradient Boosting

Code for Model Training:

Listing 3: Training Multiple Models

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
    GradientBoostingClassifier
from sklearn.svm import SVC

```

```
# Define models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'Support Vector Machine': SVC(probability=True, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42)
}

# Function to evaluate models
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

def evaluate_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else None

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_proba) if y_proba is not None else 'N/A'

    return {
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,
        'ROC AUC': roc_auc
    }

# Evaluate each model
results = {}
for model_name, model in models.items():
    metrics = evaluate_model(model, X_train, y_train, X_test, y_test)
    results[model_name] = metrics

# Display the results
import pandas as pd
results_df = pd.DataFrame(results).T
```

```
print(results_df)
```

The Random Forest model performed the best and was further optimized using **GridSearchCV**.

7 Phase 5: Pattern Evaluation

The optimized Random Forest model was evaluated using several performance metrics on the test set, achieving the following results:

- Accuracy: 81%
- Precision: 83% (Good Quality), 78% (Bad Quality)
- Recall: 80% (Good Quality), 82% (Bad Quality)
- F1 Score: 81% (Good Quality), 80% (Bad Quality)
- ROC AUC Score: 0.90

Code for Evaluation Metrics and Confusion Matrix:

Listing 4: Evaluation Metrics and Confusion Matrix

```
from sklearn.metrics import confusion_matrix,
    ConfusionMatrixDisplay, classification_report

# Predict on the test set using the optimized Random Forest
y_pred = best_rf.predict(X_test)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
    display_labels=['Bad Quality', 'Good Quality'])
disp.plot(cmap='Blues')
plt.title('Confusion Matrix for Random Forest Model')
plt.show()

# Classification report
report = classification_report(y_test, y_pred, target_names
    =['Bad Quality', 'Good Quality'])
print(report)
```

ROC Curve Code:

Listing 5: ROC Curve for Optimized Random Forest Model

```
from sklearn.metrics import roc_curve, auc

# Compute ROC curve and ROC area
y_scores = best_rf.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
```

```
roc_auc = auc(fpr, tpr)

# Plot ROC curve
import matplotlib.pyplot as plt
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC
    curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

The ROC curve for the Random Forest model, with an AUC score of 0.90, is shown below:

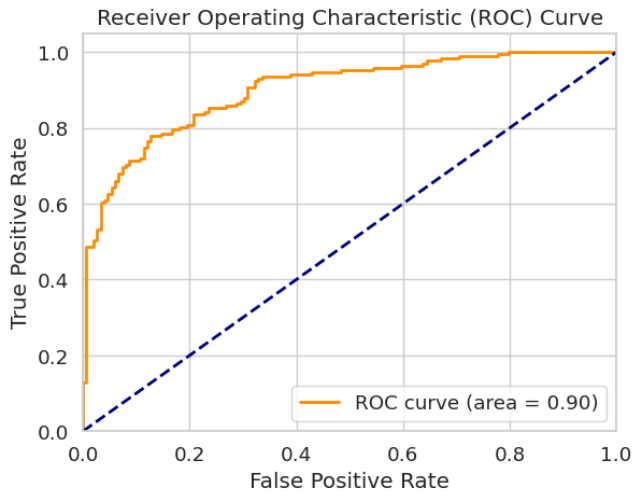

Figure 1: ROC Curve for Random Forest Model with AUC = 0.90

7.1 Feature Importance

The most important features in the Random Forest model were:

Feature Importance Code:

Listing 6: Feature Importance Plot

```
# Retrieve feature importances
importances = best_rf.feature_importances_
feature_names = data.drop('quality', axis=1).columns
```

```
4
5  # Create a DataFrame for feature importances
6  feature_importance_df = pd.DataFrame({
7      'Feature': feature_names,
8      'Importance': importances
9  }).sort_values(by='Importance', ascending=False)
10
11 # Plot Feature Importances
12 import seaborn as sns
13 plt.figure(figsize=(10, 6))
14 sns.barplot(x='Importance', y='Feature', data=
       feature_importance_df)
15 plt.title('Feature Importances from Optimized Random Forest'
       )
16 plt.show()
```

The **Feature Importance Plot** is shown below:

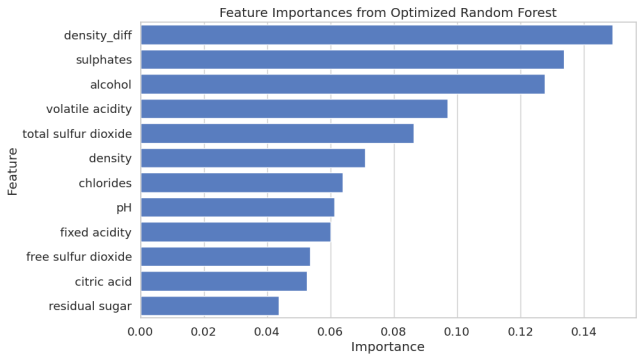

Figure 2: Feature Importance Plot for Random Forest Model

8 Phase 6: Knowledge Representation

Based on the classification metrics and visualizations, we derived several key insights:

Classification Report:

- **Accuracy**: The model achieved an accuracy of **81%**, indicating that it correctly classified the wine quality for most samples.
- **Precision**: The precision for **Good Quality** wines was **83%**, meaning that 83% of the wines predicted as good were actually good.
- **Recall**: The recall for **Bad Quality** wines was **82%**, meaning that 82% of the actual bad quality wines were correctly identified.

- **F1-Score**: Balanced F1-scores of **0.80** and **0.81** for bad and good quality wines, respectively, indicate the model handles both classes effectively.

Top Features:

- **Density_diff** was the most important feature, highlighting the influence of both density and alcohol content on wine quality.
- **Sulphates** played a significant role in determining wine preservation and quality.
- **Alcohol** content had a strong correlation with wine quality, with higher alcohol levels generally linked to higher quality.

ROC Curve: The **AUC score of 0.90** shows that the Random Forest model has a strong ability to distinguish between good and bad quality wines across different thresholds.

9 Conclusion

Using the KDD methodology, we successfully built a machine learning model to predict wine quality with an accuracy of 81% and an AUC score of 0.90. The most influential features were identified as density_diff, sulphates, and alcohol content. These insights can help winemakers optimize the quality of their products by focusing on these key factors.

10 References

- Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4), 547-553. `https://doi.org/10.1016/j.dss.2009.05.016`.
- Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32. `https://doi.org/10.1023/A:1010933404324`.