Wine Quality Prediction

Original Dataset Description

The dataset used is the **Wine Quality (Red Wine) dataset** from the UCI Machine Learning Repository. It contains physicochemical properties of red wines along with a quality rating given by experts.

Dataset Features:

1. **Fixed acidity** – Non-volatile acids (tartaric, malic, citric)
2. **Volatile acidity** – Acetic acid (too much leads to vinegar taste)
3. **Citric acid** – Adds freshness and flavor
4. **Residual sugar** – Sugar remaining after fermentation
5. **Chlorides** – Salt content
6. **Free sulfur dioxide** – Prevents microbial growth
7. **Total sulfur dioxide** – Preservative effects
8. **Density** – Wine density (affected by alcohol/sugar content)
9. **pH** – Acidity level (0-14 scale)
10. **Sulphates** – Additive affecting sulfur dioxide levels
11. **Alcohol** – Percentage of alcohol content
12. **Quality** – Target variable (score from 3 to 9, given by experts)

Dataset Statistics:

- **Number of samples:** 1,599 red wines
- **No missing values** (complete dataset)
- **Quality distribution:** Most wines are rated 5 or 6 (skewed distribution)

Step-by-Step Data Processing

1. Loading the Dataset

```
import pandas as pd
data = pd.read_csv('data/orignal/winequality-red.csv')
```

2. Splitting into Features (X) and Target (y)

- **Features (X):** All columns except `quality`
- **Target (y):** Only the `quality` column

```
X = data.drop('quality', axis=1)  # Features (11 columns)
y = data['quality']               # Target (1 column)
```

3. Train-Test Split (80% Training, 20% Testing)

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- `random_state=42` ensures reproducibility (same split every time).

4. Feature Scaling (Standardization)

- Machine learning models perform better when features are on the same scale.
- **StandardScaler** transforms data to have **mean=0** and **std=1**.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)  # Fit on training data
X_test = scaler.transform(X_test)       # Apply same scaling to test data
```

5. Saving Processed Data

```
pd.DataFrame(X_train).to_csv('data/preprocessed/X_train.csv', index=False)
pd.DataFrame(y_train).to_csv('data/preprocessed/y_train.csv', index=False)
pd.DataFrame(X_test).to_csv('data/preprocessed/X_test.csv', index=False)
pd.DataFrame(y_test).to_csv('data/preprocessed/y_test.csv', index=False)
```

Step-by-Step Machine Learning Model Building Process

1. Random Forest Classifier

Steps:

1. **Import & Initialize Model**

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100)  # 100 decision trees
```

2. **Train the Model**

```
rf.fit(X_train, y_train)  # Uses standardized X_train
```

3. **Make Predictions**

```
predictions_RF = rf.predict(X_test)
```

4. **Save Predictions**

```
pd.DataFrame(predictions_RF).to_csv('data/results/predictions_RF.csv')
```

2. Support Vector Machine (SVM)

Steps:

1. **Import & Initialize Model**

```
from sklearn.svm import SVC
svm = SVC(kernel='rbf')  # Radial Basis Function kernel
```

2. **Train the Model**

```
svm.fit(X_train, y_train)
```

3. **Make Predictions**

```
predictions_SVM = svm.predict(X_test)
```

4. **Save Predictions**

```
pd.DataFrame(predictions_SVM).to_csv('data/results/predictions_SVM.csv')
```

3. Naive Bayes (Gaussian)

Steps:

1. **Import & Initialize Model**

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()  # Assumes Gaussian-distributed features
```

2. **Train the Model**

```
nb.fit(X_train, y_train)
```

3. **Make Predictions**

```
predictions_NB = nb.predict(X_test)
```

4. **Save Predictions**

```
pd.DataFrame(predictions_NB).to_csv('data/results/predictions_NB.csv')
```

4. Artificial Neural Network (ANN)

Steps:

1. **Import Libraries**

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

2. **Adjust Labels (3-9 → 0-6)**

```
y_train_adj = y_train - 3
y_test_adj = y_test - 3
```

3. **Build Model Architecture**

```
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(7, activation='softmax')  # 7 classes (0-6)
])
```

4. **Compile the Model**

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

```
```

5. **Train the Model**

```
history = model.fit(
    X_train, y_train_adj,
    epochs=20,
    batch_size=32,
    validation_split=0.2
)
```

6. **Make Predictions**

```
predictions_ANN = model.predict(X_test).argmax(axis=1) + 3
```

7. **Save Predictions**

```
pd.DataFrame(predictions_ANN).to_csv('data/results/predictions_ANN.csv')
```

5. Linear Regression (Baseline)

Steps:

1. **Import & Initialize Model**

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

2. **Train the Model**

```
lr.fit(X_train, y_train)
```

3. **Make Predictions**

```
predictions_LR = lr.predict(X_test).round().astype(int)  # Rounded to nearest integer
```

4. **Save Predictions**

```
pd.DataFrame(predictions_LR).to_csv('data/results/predictions_LR.csv')
```

6. Model Evaluation

All models were evaluated using:

```
from sklearn.metrics import classification_report, accuracy_score

for name, pred in models.items():
    print(f"----- {name} -----")
    print(classification_report(y_test, pred))
    print(f"Accuracy: {accuracy_score(y_test, pred):.2f}\n")
```

Metrics Reported:

- **Accuracy**: Overall correctness.
- **Precision/Recall/F1**: Per-class performance.

Summary Table of Models

Model	Type	Key Features	Best For
Random Forest	Ensemble	Handles non-linearity, robust	General-purpose
SVM	Kernel-based	High-dimensional spaces	Small/medium datasets
Naive Bayes	Probabilistic	Fast, simple	Baseline
ANN	Deep Learning	Complex patterns	Large/structured data
Linear Regression	Linear	Interpretable	Baseline (regression)