

Project 2

Feature Selection Methods in Machine Learning

Dataset: WineQT (Red Wine Quality)

Course: Machine Learning

Task	Description
1	Download dataset from Kaggle
2	Handle missing values
3	Apply ML models with Grid Search
4	Feature Selection (2 methods)
5	Apply ML models to selected features
6	Save and export as PDF

Step 1: Download Dataset from Kaggle

The WineQT dataset was downloaded from Kaggle. It contains 1,143 samples of red wine with 11 chemical features and a quality score as the target variable.

Dataset Link: <https://www.kaggle.com/datasets/yasserh/wine-quality-dataset>

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

# Load dataset
df = pd.read_csv("WineQT.csv")
print("Shape:", df.shape)
print(df.head())
```

Step 2: Handle Missing Values

Before applying any machine learning model, the dataset must be cleaned. The following steps were performed:

```
# Check missing values
print("Missing Values:")
print(df.isnull().sum())

# Check duplicates
print("Duplicate Rows:", df.duplicated().sum())

# Check data types
print(df.dtypes)

# Statistical summary
print(df.describe())

# Handle missing values (median imputation as best practice)
for col in df.select_dtypes(include=np.number).columns:
    df[col].fillna(df[col].median(), inplace=True)

# Remove duplicate rows
df.drop_duplicates(inplace=True)
print("Shape after cleaning:", df.shape)

# Drop Id column (not a feature)
df.drop(columns=['Id'], inplace=True)

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

```
print("Features:", list(X.columns))
print("Target: quality")
```

Step 3: ML Models with Grid Search Optimizer

Three regression models were trained using GridSearchCV with 5-fold cross-validation to find the optimal hyperparameters for each model.

[illegible]

```

grid.fit(X_tr, y_tr)
print("Best Params:", grid.best_params_)
print("Best CV R2:", round(grid.best_score_, 4))

y_pred = grid.best_estimator_.predict(X_te)
mae = mean_absolute_error(y_te, y_pred)
rmse = np.sqrt(mean_squared_error(y_te, y_pred))
r2 = r2_score(y_te, y_pred)
print(f"MAE:{mae:.4f} | RMSE:{rmse:.4f} | R2:{r2:.4f}")
return {'Model':name, 'MAE':round(mae,4),
        'RMSE':round(rmse,4), 'R2':round(r2,4)}

results_all = []
results_all.append(run_grid_search(
DecisionTreeRegressor(random_state=42), dt_params,
X_train, X_test, y_train, y_test, "Decision Tree"))

results_all.append(run_grid_search(
RandomForestRegressor(random_state=42), rf_params,
X_train, X_test, y_train, y_test, "Random Forest"))

results_all.append(run_grid_search(
SVR(), svr_params,
X_train_scaled, X_test_scaled, y_train, y_test, "SVR"))

results_all_df = pd.DataFrame(results_all)
print("\nAll Features Performance:")
print(results_all_df)

```

Step 4: Feature Selection (Two Methods)

Method 1: Mutual Information (Filter Method)

Mutual Information measures the statistical dependency between each feature and the target variable. Higher score = more important feature.

```

from sklearn.feature_selection import SelectKBest, mutual_info_regression

# Calculate MI scores for all features
selector_mi_all = SelectKBest(mutual_info_regression, k='all')
selector_mi_all.fit(X_train, y_train)

mi_df = pd.DataFrame({
    'Feature': X.columns,
    'MI Score': selector_mi_all.scores_
}).sort_values('MI Score', ascending=False)

print("Mutual Information Scores:")
print(mi_df)

```

```

# Plot
plt.figure(figsize=(10, 5))
plt.bar(mi_df['Feature'], mi_df['MI Score'], color='steelblue')
plt.xlabel("Feature")
plt.ylabel("MI Score")
plt.title("Feature Importance - Mutual Information")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Select top 5 features
k = 5
selector_mi = SelectKBest(mutual_info_regression, k=k)
X_train_mi = selector_mi.fit_transform(X_train, y_train)
X_test_mi = selector_mi.transform(X_test)
selected_mi = X.columns[selector_mi.get_support()]
print("Selected Features (MI):", list(selected_mi))

```

Method 2: RFE + Random Forest (Wrapper Method)

Recursive Feature Elimination trains a Random Forest model iteratively, removing the least important feature at each step until only k features remain.

```

from sklearn.feature_selection import RFE

# Apply RFE with Random Forest
rf_rfe = RandomForestRegressor(n_estimators=200, random_state=42)
k = 5
rfe = RFE(estimator=rf_rfe, n_features_to_select=k)
rfe.fit(X_train, y_train)

rfe_df = pd.DataFrame({
    'Feature': X.columns,
    'Ranking': rfe.ranking_,
    'Selected': rfe.support_
}).sort_values('Ranking')

print("RFE Feature Ranking:")
print(rfe_df)

# Plot
colors_bar = ['green' if s else 'steelblue' for s in rfe_df['Selected']]
plt.figure(figsize=(10, 5))
plt.bar(rfe_df['Feature'], rfe_df['Ranking'], color=colors_bar)
plt.xlabel("Feature")
plt.ylabel("RFE Ranking (1 = Selected)")
plt.title("Feature Ranking - RFE + Random Forest (Green = Selected)")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

```

```
X_train_rfe = rfe.transform(X_train)
X_test_rfe = rfe.transform(X_test)
selected_rfe = X.columns[rfe.get_support()]
print("Selected Features (RFE):", list(selected_rfe))

# Compare
print("\nCommon Features:", list(set(selected_mi) & set(selected_rfe)))
```

Step 5: Apply ML Models to Selected Features

The three regression models (Decision Tree, Random Forest, SVR) are applied to features selected by each method and evaluated using MAE, RMSE, and R2 Score.

[illegible]

```
# ■■■ Models on RFE-Selected Features
print("MODELS ON RFE FEATURES")
scaler_rfe = StandardScaler()
X_train_rfe_sc = scaler_rfe.fit_transform(X_train_rfe)
X_test_rfe_sc = scaler_rfe.transform(X_test_rfe)

results_rfe = []
for name, model in models.items():
```


Summary: Evaluation Metrics Explained

Metric	Formula	Best Value	Meaning
MAE	$\text{mean} y - y_{\text{pred}} $	Lower is better	Average absolute error
MSE	$\text{mean}(y - y_{\text{pred}})^2$	Lower is better	Penalizes large errors
RMSE	$\sqrt{\text{MSE}}$	Lower is better	Same unit as target
R2	$1 - \text{SS}_{\text{res}}/\text{SS}_{\text{tot}}$	Closer to 1	Variance explained by model