

```
# Project 2 - Machine Learning + Missing Values + Feature Selection
# Student: Yusra Elfaran
# Dataset: Wine Quality (WineQT.csv) (from Kaggle)
# Submission deadline: 21/12/2025

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
confusion_matrix

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

from sklearn.feature_selection import SelectKBest, mutual_info_classif, RFE

# 1) Load dataset (downloaded from Kaggle)
df_raw = pd.read_csv(r"/mnt/data/WineQT.csv")

# 2) Prepare target + handle missing values
# We convert quality into a binary classification:
# good wine = quality >= 6, otherwise not good.
df = df_raw.copy()

y = (df["quality"] >= 6).astype(int)
X = df.drop(columns=["quality"])

# Drop ID-like column if exists
if "Id" in X.columns:
    X = X.drop(columns=["Id"])

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

preprocess = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")), # missing values
    ("scaler", StandardScaler())
])

def evaluate(estimator, Xte, yte):
    pred = estimator.predict(Xte)

    if hasattr(estimator, "predict_proba"):
        proba = estimator.predict_proba(Xte)[:, 1]
    else:
        scores = estimator.decision_function(Xte)
        proba = (scores - scores.min()) / (scores.max() - scores.min() + 1e-9)

    return {
        "Accuracy": accuracy_score(yte, pred),
        "Precision": precision_score(yte, pred, zero_division=0),
        "Recall": recall_score(yte, pred, zero_division=0),
        "F1": f1_score(yte, pred, zero_division=0),
        "ROC_AUC": roc_auc_score(yte, proba),
        "ConfusionMatrix": confusion_matrix(yte, pred)
    }

# 3) Apply ML models with GridSearch optimizer
models = {
    "LogReg": (LogisticRegression(max_iter=3000), {
        "clf__C": [0.1, 1, 10]
    }),
    "SVM": (SVC(probability=True), {
        "clf__C": [0.5, 1, 5],
        "clf__gamma": ["scale", "auto"]
    }),
    "RF": (RandomForestClassifier(random_state=42), {
        "clf__n_estimators": [200, 400],
```

```

        "clf__max_depth": [None, 8, 12]
    })
}

full_results = []
best_estimators = {}

for name, (clf, grid) in models.items():
    pipe = Pipeline(steps=[("prep", preprocess), ("clf", clf)])
    gs = GridSearchCV(pipe, grid, cv=3, scoring="f1", n_jobs=1)
    gs.fit(X_train, y_train)

    best_estimators[name] = gs.best_estimator_
    m = evaluate(gs.best_estimator_, X_test, y_test)

    full_results.append({
        "Model": name,
        "BestParams": gs.best_params_,
        "Accuracy": m["Accuracy"],
        "Precision": m["Precision"],
        "Recall": m["Recall"],
        "F1": m["F1"],
        "ROC_AUC": m["ROC_AUC"]
    })

full_results_df = pd.DataFrame(full_results).sort_values("F1", ascending=False)
print("GridSearch results (Full Features):")
print(full_results_df)

# 4) Feature Selection Method 1: SelectKBest (Mutual Information)
pipe_kbest = Pipeline(steps=[
    ("prep", preprocess),
    ("fs", SelectKBest(score_func=mutual_info_classif, k=8)),
    ("clf", RandomForestClassifier(random_state=42))
])

grid_kbest = {
    "fs__k": [5, 8, 10],
    "clf__n_estimators": [200, 400],
    "clf__max_depth": [None, 8, 12]
}

gs_kbest = GridSearchCV(pipe_kbest, grid_kbest, cv=3, scoring="f1", n_jobs=1)
gs_kbest.fit(X_train, y_train)

kbest_metrics = evaluate(gs_kbest.best_estimator_, X_test, y_test)

selector = gs_kbest.best_estimator_.named_steps["fs"]
mask = selector.get_support()
selected_kbest = pd.DataFrame({
    "Feature": X.columns[mask],
    "MI_Score": selector.scores_[mask]
}).sort_values("MI_Score", ascending=False)

print("\nBest params (SelectKBest):", gs_kbest.best_params_)
print("Top selected features (SelectKBest):")
print(selected_kbest)
print("Metrics (SelectKBest):", kbest_metrics)

# 5) Feature Selection Method 2: RFE (Recursive Feature Elimination)
base_lr = LogisticRegression(max_iter=5000, solver="liblinear")

pipe_rfe = Pipeline(steps=[
    ("prep", preprocess),
    ("fs", RFE(estimator=base_lr, n_features_to_select=8, step=1)),
    ("clf", LogisticRegression(max_iter=5000, solver="liblinear"))
])

grid_rfe = {
    "fs__n_features_to_select": [5, 8, 10],
    "clf__C": [0.5, 1, 5]
}

gs_rfe = GridSearchCV(pipe_rfe, grid_rfe, cv=3, scoring="f1", n_jobs=1)

```

```
gs_rfe.fit(X_train, y_train)

rfe_metrics = evaluate(gs_rfe.best_estimator_, X_test, y_test)

rfe = gs_rfe.best_estimator_.named_steps["fs"]
mask2 = rfe.support_

selected_rfe = pd.DataFrame({
    "Feature": X.columns[mask2],
    "Ranking": rfe.ranking_[mask2]
})

print("\nBest params (RFE):", gs_rfe.best_params_)
print("Selected features (RFE):")
print(selected_rfe)
print("Metrics (RFE):", rfe_metrics)
```

Results Summary (WineQT.csv)

Rows/Cols: 1143 x 13

Target: good wine = quality >= 6 (binary classification)

GridSearch (Full Features) - sorted by F1:

Model	BestParams	Accuracy	Precision	Recall	F1	ROC_AUC
RF	{'clf__max_depth': 8, 'clf__n_estimators': 400}	0.803493	0.796992	0.854839	0.824903	0.864900
LogReg	{'clf__C': 10}	0.777293	0.801653	0.782258	0.791837	0.823733
SVM	{'clf__C': 0.5, 'clf__gamma': 'scale'}	0.768559	0.784000	0.790323	0.787149	0.838671

Feature Selection #1: SelectKBest (Mutual Information)

Best params: {'clf__max_depth': 12, 'clf__n_estimators': 200, 'fs__k': 10}

Feature	MI_Score
alcohol	0.117047
volatile acidity	0.074315
sulphates	0.074023
free sulfur dioxide	0.045476
total sulfur dioxide	0.045360
chlorides	0.034942
residual sugar	0.032594
density	0.027323
citric acid	0.013521
fixed acidity	0.008154

Feature Selection #2: RFE (Logistic Regression)

Best params: {'clf__C': 0.5, 'fs__n_features_to_select': 10}

Feature	Ranking
fixed acidity	1
volatile acidity	1
citric acid	1
residual sugar	1
chlorides	1
free sulfur dioxide	1
total sulfur dioxide	1
density	1
sulphates	1
alcohol	1