

```

# =====

# Project 2: Machine Learning Pipeline with Feature Selection
# Author: Rayn
# Dataset: Breast Cancer Wisconsin (sklearn built-in)
# =====

import pandas as pd
import numpy as np
import matplotlib
matplotlib.use('Agg')
import warnings
warnings.filterwarnings('ignore')

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.inspection import permutation_importance

# — STEP 1: Load Dataset —————
print("=" * 65)

```

```
print("STEP 1 - Loading Dataset (Breast Cancer Wisconsin)")  
print("=" * 65)
```

```
data = load_breast_cancer()  
df = pd.DataFrame(data.data, columns=data.feature_names)  
df['target'] = data.target
```

```
print(f"Dataset shape : {df.shape}")  
print(f"Class distribution:\n{df['target'].value_counts()}")  
print("\nFirst 3 rows:")  
print(df.head(3).to_string())
```

```
# — STEP 2: Introduce & Handle Missing Values —————
```

```
print("\n" + "=" * 65)  
print("STEP 2 - Handling Missing Values")  
print("=" * 65)
```

```
np.random.seed(42)  
missing_cols = ['mean radius', 'mean texture', 'mean perimeter',  
                'mean area', 'mean smoothness']  
for col in missing_cols:  
    idx = np.random.choice(df.index, size=int(0.10 * len(df)), replace=False)  
    df[col] = df[col].copy()  
    df.loc[idx, col] = np.nan  
  
print("Missing values BEFORE handling:")
```

```

print(df.isnull().sum()[df.isnull().sum() > 0])

# Fix: use df[col] = ... to avoid ChainedAssignment with pandas CoW
for col in df.columns[:-1]:
    if df[col].isnull().any():
        median_val = df[col].median()
        df[col] = df[col].fillna(median_val)

print("\nMissing values AFTER handling (median imputation):")
print(df.isnull().sum().sum(), "total remaining")
print(f"Dataset shape: {df.shape}")

# — Prepare Features & Split —————
X = df.drop(columns=['target'])
y = df['target']

scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

assert X_scaled.isnull().sum().sum() == 0, "NaN found after scaling!"

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.20, random_state=42, stratify=y)

print(f"\nTrain: {X_train.shape[0]} samples | Test: {X_test.shape[0]} samples")

```

```

# — STEP 3: ML Models with GridSearchCV —————

print("\n" + "=" * 65)

print("STEP 3 - ML Models with GridSearchCV (Full Feature Set)")

print("=" * 65)


models_params = {

    "Logistic Regression": {

        "model" : LogisticRegression(max_iter=2000, random_state=42),

        "params": {"C": [0.01, 0.1, 1, 10], "solver": ["lbfgs", "liblinear"]}

    },

    "Random Forest": {

        "model" : RandomForestClassifier(random_state=42),

        "params": {"n_estimators": [50, 100], "max_depth": [None, 5, 10]}

    },

    "Decision Tree": {

        "model" : DecisionTreeClassifier(random_state=42),

        "params": {"max_depth": [3, 5, 10, None], "criterion": ["gini", "entropy"]}

    },

    "SVM": {

        "model" : SVC(random_state=42),

        "params": {"C": [0.1, 1, 10], "kernel": ["rbf", "linear"]}

    },

    "Gradient Boosting": {

        "model" : GradientBoostingClassifier(random_state=42),

        "params": {"n_estimators": [50, 100], "learning_rate": [0.05, 0.1]}

    }

}

```

```
}
```

```
results_full = {}
```

```
for name, mp in models_params.items():
```

```
    gs = GridSearchCV(mp["model"], mp["params"], cv=5, scoring='accuracy', n_jobs=-1)
```

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

```
    y_pred = gs.best_estimator_.predict(X_test)
```

```
    acc = accuracy_score(y_test, y_pred)
```

```
    f1 = f1_score(y_test, y_pred, average='weighted')
```

```
    results_full[name] = {"Best Params": gs.best_params_,
```

```
                          "Accuracy": round(acc, 4),
```

```
                          "F1-Score": round(f1, 4),
```

```
                          "CV Score": round(gs.best_score_, 4)}
```

```
    print(f"\n{name}")
```

```
    print(f" Best Params : {gs.best_params_}")
```

```
    print(f" Test Accuracy: {acc:.4f} | F1: {f1:.4f} | CV: {gs.best_score_:.4f}")
```

```
    print(" Classification Report:")
```

```
    print(classification_report(y_test, y_pred, target_names=["Malignant", "Benign"]))
```

```
# — STEP 4a: Feature Selection - SelectKBest (ANOVA) —————
```

```
print("\n" + "=" * 65)
```

```
print("STEP 4a - Feature Selection: SelectKBest (ANOVA F-test)")
```

```
print("=" * 65)
```

```
selector_k = SelectKBest(score_func=f_classif, k=10)
```

```

selector_k.fit(X_train, y_train)

scores_anova = pd.Series(selector_k.scores_,
                          index=X.columns).sort_values(ascending=False)

top10_anova = scores_anova.head(10).index.tolist()

print("\nANOVA F-scores:")
print(scores_anova.round(2).to_string())
print(f"\nTop-10 selected: {top10_anova}")

X_train_a = X_train[top10_anova]
X_test_a = X_test[top10_anova]

# — STEP 4b: Feature Selection - Permutation Importance —
print("\n" + "=" * 65)
print("STEP 4b - Feature Selection: Permutation Importance (RF)")
print("=" * 65)

rf_base = RandomForestClassifier(n_estimators=100, random_state=42)
rf_base.fit(X_train, y_train)

perm = permutation_importance(rf_base, X_test, y_test,
                              n_repeats=15, random_state=42, n_jobs=-1)

perm_scores = pd.Series(perm.importances_mean,
                        index=X.columns).sort_values(ascending=False)

top10_perm = perm_scores.head(10).index.tolist()

print("\nPermutation Importances:")

```

```
print(perm_scores.round(4).to_string())
print(f"\nTop-10 selected: {top10_perm}")
```

```
X_train_p = X_train[top10_perm]
X_test_p = X_test[top10_perm]
```

```
# — STEP 5: Models on Selected Features —————
```

```
print("\n" + "=" * 65)
print("STEP 5 - Models on Selected Features")
print("=" * 65)
```

```
def run_models(X_tr, X_te, label):
    res = {}
    for name, mp in models_params.items():
        gs = GridSearchCV(mp["model"], mp["params"], cv=5, scoring='accuracy', n_jobs=-1)
        gs.fit(X_tr, y_train)
        y_pred = gs.best_estimator_.predict(X_te)
        acc = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred, average='weighted')
        res[name] = {"Accuracy": round(acc, 4), "F1-Score": round(f1, 4)}
        print(f" [{label:22s}] {name:22s} Acc={acc:.4f} F1={f1:.4f}")
    return res
```

```
print("\n--- ANOVA SelectKBest (top-10) ---")
results_anova = run_models(X_train_a, X_test_a, "ANOVA SelectKBest")
```

```

print("\n--- Permutation Importance (top-10) ---")

results_perm = run_models(X_train_p, X_test_p, "Permutation Imp.")

# — Summary —————
print("\n" + "=" * 65)

print("FINAL SUMMARY - Accuracy Comparison")

print("=" * 65)


summary_df = pd.DataFrame({
    "Full (30 feat)": {k: v["Accuracy"] for k, v in results_full.items()},
    "ANOVA (10 feat)": {k: v["Accuracy"] for k, v in results_anova.items()},
    "Perm (10 feat)": {k: v["Accuracy"] for k, v in results_perm.items()}
})

print(summary_df.to_string())


best = summary_df.stack().idxmax()

print(f"\nBest: {best[0]} with {best[1]} -> Accuracy = {summary_df.stack().max():.4f}")

print("\nProject 2 completed successfully.")

```