

Project 2

Title: Machine Learning Pipeline with Grid Search

Dataset: Titanic (Kaggle)

Task:

1. Handle missing values

2. Apply ML models with GridSearch optimization

3. Use two feature selection methods

4. Evaluate model performance

=====

Import required libraries

import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.impute import SimpleImputer

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.feature_selection import SelectKBest, f_classif, RFE

```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix,  
classification_report
```

```
# -----
```

```
# 1. Load Kaggle dataset
```

```
# -----
```

```
# The Titanic dataset is used for this project.
```

```
# The target variable is "Survived".
```

```
df = pd.read_csv("train.csv")
```

```
target_column = "Survived"
```

```
# Display dataset information
```

```
print("Dataset shape:", df.shape)
```

```
print("\nMissing values per column:\n", df.isnull().sum())
```

```
# -----
```

```
# 2. Separate features and target
```

```
# -----
```

```
# X contains input features
```

```
# y contains the target labels
```

```
X = df.drop(columns=[target_column])
```

```
y = df[target_column]
```

```
# Split the data into training and testing sets
```

```
# Stratification is used to preserve class distribution
```

```
X_train, X_test, y_train, y_test = train_test_split(
```

```

X,
y,
test_size=0.2,
random_state=42,
stratify=y
)

# -----
# 3. Handle missing values and preprocessing
# -----

# Identify numerical and categorical columns
numeric_features = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_features = [col for col in X.columns if col not in numeric_features]

# Pipeline for numerical data:
# - Replace missing values with median
# - Scale features using standardization
numeric_pipeline = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

# Pipeline for categorical data:
# - Replace missing values with most frequent value
# - Convert categories into numerical values using One-Hot Encoding
categorical_pipeline = Pipeline(steps=[

```

```
    ("imputer", SimpleImputer(strategy="most_frequent")),  
    ("encoder", OneHotEncoder(handle_unknown="ignore"))  
])
```

Combine numerical and categorical preprocessing

```
preprocessor = ColumnTransformer(  
    transformers=[  
        ("num", numeric_pipeline, numeric_features),  
        ("cat", categorical_pipeline, categorical_features)  
    ]  
)
```

Evaluation function

This function prints common classification metrics

```
def evaluate(y_true, y_pred, title):  
    print("\n", title)  
    print("Accuracy:", accuracy_score(y_true, y_pred))  
    print("F1 Score:", f1_score(y_true, y_pred))  
    print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))  
    print("Classification Report:\n", classification_report(y_true, y_pred))
```

4. Apply ML models with GridSearch

```

# Two models are used:

# - Logistic Regression

# - Random Forest Classifier

models = {

    "Logistic Regression": (
        LogisticRegression(max_iter=500),
        {"model__C": [0.1, 1, 10]}
    ),
    "Random Forest": (
        RandomForestClassifier(random_state=42),
        {
            "model__n_estimators": [200, 500],
            "model__max_depth": [None, 5, 10]
        }
    )
}

print("\nGrid Search on All Features")

# Train and evaluate each model using GridSearch
for name, (model, params) in models.items():

    # Create a full pipeline including preprocessing and model
    pipeline = Pipeline(steps=[
        ("preprocess", preprocessor),
        ("model", model)
    ])

```

```
])
```

```
# Apply GridSearch to find optimal hyperparameters
```

```
grid = GridSearchCV(
```

```
    pipeline,
```

```
    params,
```

```
    cv=5,
```

```
    scoring="f1",
```

```
    n_jobs=-1
```

```
)
```

```
# Train the model
```

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

```
# Predict on test data
```

```
predictions = grid.best_estimator_.predict(X_test)
```

```
# Display results
```

```
print("\nModel:", name)
```

```
print("Best Parameters:", grid.best_params_)
```

```
evaluate(y_test, predictions, "Evaluation Using All Features")
```

```
# -----
```

```
# 5. Feature Selection Method 1: SelectKBest
```

```
# -----
```

```
# Select the top 10 most important features using ANOVA F-test
```

```
print("\nFeature Selection Using SelectKBest")
```

```
pipeline_kbest = Pipeline(steps=[  
    ("preprocess", preprocessor),  
    ("select", SelectKBest(score_func=f_classif, k=10)),  
    ("model", RandomForestClassifier(random_state=42))  
])
```

```
# Train and evaluate the model
```

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

```
pred_kbest = pipeline_kbest.predict(X_test)
```

```
evaluate(y_test, pred_kbest, "Evaluation with SelectKBest")
```

```
# -----
```

```
# 6. Feature Selection Method 2: RFE
```

```
# -----
```

```
# Recursive Feature Elimination selects features by recursively
```

```
# removing the least important features
```

```
print("\nFeature Selection Using RFE")
```

```
pipeline_rfe = Pipeline(steps=[  
    ("preprocess", preprocessor),  
    ("rfe", RFE(  
        estimator=LogisticRegression(max_iter=500),  
        n_features_to_select=10
```

```
)),  
("model", LogisticRegression(max_iter=500))  
])  
  
# Train and evaluate the model  
pipeline_rfe.fit(X_train, y_train)  
pred_rfe = pipeline_rfe.predict(X_test)  
  
evaluate(y_test, pred_rfe, "Evaluation with RFE")  
  
print("\nProject execution completed.")
```