

```

# -*- coding: utf-8 -*-
"""
Assignment Solution: Machine Learning Pipeline with Feature Selection
Student Name: [Merwa Omer]
"""

# =====
# Step 1: Import Necessary Libraries
# =====
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
from sklearn.feature_selection import RFE
import warnings
warnings.filterwarnings('ignore')

# =====
# Helper Function to Evaluate and Display Model Performance
# =====
def evaluate_model(y_true, y_pred, model_name="Model"):
    """
    Calculates metrics, prints a report, and plots a confusion matrix.
    """
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

    print(f"\n{'='*30}\nModel: {model_name}\n{'='*30}")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

    # Plot Confusion Matrix
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=np.unique(y_true), yticklabels=np.unique(y_true))
    plt.title(f'Confusion Matrix for {model_name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

    return {'accuracy': accuracy, 'precision': precision, 'recall': recall, 'f1': f1}

# =====
# Step 2: Load and Prepare the Dataset (Download from Kaggle)
# =====
# Note: Please download 'breast-cancer.xls' from Kaggle and place it in the same folder.
# Link: https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset
df = pd.read_csv("breast-cancer.xls")
print("Dataset loaded successfully.")
print(f"Initial dataset shape: {df.shape}")
print(f"Missing values before handling:\n{df.isnull().sum().sum()}") # Check for nulls

# =====
# Step 3: Handle Missing Values and Preprocess Data
# =====
# The breast cancer dataset is clean and has no missing values.
# However, we still need to encode the target variable 'diagnosis' (M/B) to numbers.
le = LabelEncoder()
df['diagnosis'] = le.fit_transform(df['diagnosis']) # Malignant (M) -> 1, Benign (B) -> 0

# Define features (X) and target (y). Drop 'id' column as it's not a feature.
X = df.drop(['diagnosis', 'id'], axis=1)
y = df['diagnosis']

print(f"Features shape: {X.shape} Target shape: {y.shape}")

```

```

print(f"Features shape: {X.shape}, Target shape: {y.shape}")
print(f"Target distribution:\n{y.value_counts()}") # Check for imbalanced classes

# Split data into training and testing sets (80/20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
print("Data split into training and testing sets.")

# =====
# Step 4: Apply Machine Learning Models with GridSearchCV (No Feature Selection)
# =====
# We will train and evaluate a few common classifiers first.

models = {
    'Random Forest': RandomForestClassifier(random_state=42),
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
    'SVM': SVC(random_state=42),
    'K-Neighbors': KNeighborsClassifier()
}

# Hyperparameter grids for GridSearchCV
param_grids = {
    'Random Forest': {
        'n_estimators': [100, 200],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5]
    },
    'Logistic Regression': {
        'C': [0.1, 1, 10],
        'solver': ['liblinear', 'lbfgs']
    },
    'SVM': {
        'C': [0.1, 1, 10],
        'gamma': ['scale', 'auto'],
        'kernel': ['linear', 'rbf']
    },
    'K-Neighbors': {
        'n_neighbors': [3, 5, 7, 9],
        'weights': ['uniform', 'distance']
    }
}

# Store initial results
initial_results = {}

print("\n" + "="*60)
print("STEP 4: MODEL TRAINING & EVALUATION (with Hyperparameter Tuning)")
print("="*60)

for name, model in models.items():
    print(f"\n--- Tuning {name} ---")
    # Create GridSearchCV object
    grid_search = GridSearchCV(estimator=model,
                               param_grid=param_grids[name],
                               cv=5,
                               scoring='accuracy',
                               n_jobs=-1,
                               verbose=1)

    # Fit the grid search to the training data
    grid_search.fit(X_train, y_train)

    # Get the best model
    best_model = grid_search.best_estimator_
    print(f"Best parameters: {grid_search.best_params_}")

    # Make predictions on the test set
    y_pred = best_model.predict(X_test)

    # Evaluate the model
    print(f"\nEvaluating optimized {name}:")
    metrics = evaluate_model(y_test, y_pred, model_name=name)
    initial_results[name] = metrics

# =====
# Step 5: Feature Selection Method 1 - Recursive Feature Elimination (RFE)
# =====
print("\n" + "="*60)
print("STEP 5: FEATURE SELECTION WITH RFE")
print("="*60)

# Choose a model to guide the RFE process
estimator_for_rfe = RandomForestClassifier(n_estimators=100, random_state=42)
# Select top 15 features
rfe = RFE(estimator=estimator_for_rfe, n_features_to_select=15)
rfe.fit(X_train, y_train)

```

```

rfe.fit(X_train, y_train)

# Get the selected features
selected_features_rfe = X.columns[rfe.support_]
print(f"Selected {len(selected_features_rfe)} features using RFE:\n{list(selected_features_rfe)}")

# Transform the dataset to only include the selected features
X_train_rfe = rfe.transform(X_train)
X_test_rfe = rfe.transform(X_test)

# Re-train and evaluate models on RFE-selected features
print("\n--- Evaluating Models on RFE-Selected Features ---")
rfe_results = {}
for name, model in models.items():
    # Re-initialize the model
    model_inst = model
    # Perform GridSearchCV again on the reduced feature set
    grid_search = GridSearchCV(estimator=model_inst,
                               param_grid=param_grids[name],
                               cv=5,
                               scoring='accuracy',
                               n_jobs=-1,
                               verbose=0)
    grid_search.fit(X_train_rfe, y_train)
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test_rfe)

    print(f"\nEvaluating {name} on RFE features:")
    metrics = evaluate_model(y_test, y_pred, model_name=f"{name} (RFE)")
    rfe_results[name] = metrics

# =====
# Step 6: Feature Selection Method 2 - Random Forest Feature Importance (Embedded)
# =====
print("\n" + "="*60)
print("STEP 6: FEATURE SELECTION WITH RANDOM FOREST IMPORTANCE")
print("="*60)

# Train a Random Forest on the full training set to get feature importances
rf_for_importance = RandomForestClassifier(n_estimators=200, random_state=42)
rf_for_importance.fit(X_train, y_train)

# Get feature importances
importances = rf_for_importance.feature_importances_
indices = np.argsort(importances)[::-1] # Sort in descending order

# Select top 15 features
k = 15
top_k_indices = indices[:k]
selected_features_rf = X.columns[top_k_indices]
print(f"Selected {k} features using RF Importance:\n{list(selected_features_rf)}")

# Transform the dataset to only include the selected features
X_train_rf_imp = X_train.iloc[:, top_k_indices]
X_test_rf_imp = X_test.iloc[:, top_k_indices]

# Re-train and evaluate models on RF Importance-selected features
print("\n--- Evaluating Models on RF Importance-Selected Features ---")
rf_imp_results = {}
for name, model in models.items():
    # Re-initialize the model
    model_inst = model
    # Perform GridSearchCV again on the reduced feature set
    grid_search = GridSearchCV(estimator=model_inst,
                               param_grid=param_grids[name],
                               cv=5,
                               scoring='accuracy',
                               n_jobs=-1,
                               verbose=0)
    grid_search.fit(X_train_rf_imp, y_train)
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test_rf_imp)

    print(f"\nEvaluating {name} on RF Importance features:")
    metrics = evaluate_model(y_test, y_pred, model_name=f"{name} (RF-Imp)")
    rf_imp_results[name] = metrics

# =====
# Step 7: Compare and Summarize All Results
# =====
print("\n" + "="*60)
print("FINAL: COMPARISON OF MODEL PERFORMANCE")
print("="*60)

```

```

# Create a summary dataframe for comparison
comparison_data = []
for model_name in initial_results:
    comparison_data.append({
        'Model': model_name,
        'Feature Set': 'All Features',
        'Accuracy': initial_results[model_name]['accuracy']
    })
    comparison_data.append({
        'Model': model_name,
        'Feature Set': 'RFE (15 Features)',
        'Accuracy': rfe_results[model_name]['accuracy']
    })
    comparison_data.append({
        'Model': model_name,
        'Feature Set': 'RF Importance (15 Features)',
        'Accuracy': rf_imp_results[model_name]['accuracy']
    })

comparison_df = pd.DataFrame(comparison_data)
pivot_table = comparison_df.pivot(index='Model', columns='Feature Set', values='Accuracy')
print("\nAccuracy Comparison Table:")
print(pivot_table.to_string(float_format="%.4f"))

# Plot the comparison
pivot_table.plot(kind='bar', figsize=(10, 6), colormap='viridis')
plt.title('Model Accuracy Comparison Across Different Feature Sets')
plt.ylabel('Accuracy')
plt.xlabel('Machine Learning Model')
plt.xticks(rotation=0)
plt.ylim(0.85, 1.0) # Adjust as needed
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

print("\n" + "="*60)
print("Assignment Completed Successfully.")
print("="*60)

```

```

-----
FileNotFoundError                                Traceback (most recent call last)
/tmp/ipykernel_3203/1109751206.py in <cell line: 0>()
    62 # Note: Please download 'breast-cancer.xls' from Kaggle and place it in the same folder.
    63 # Link: https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset
--> 64 df = pd.read_csv("breast-cancer.xls")
    65 print("Dataset loaded successfully.")
    66 print(f"Initial dataset shape: {df.shape}")

----- 4 frames -----
/usr/local/lib/python3.12/dist-packages/pandas/io/common.py in get_handle(path_or_buf, mode, encoding,
compression, memory_map, is_text, errors, storage_options)
    871     if ioargs.encoding and "b" not in ioargs.mode:
    872         # Encoding
--> 873         handle = open(
    874             handle,
    875             ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory: 'breast-cancer.xls'

```

Next steps: [Explain error](#)

