Report Assignment 3: Intelligent Cancer Diagnose

1820212030 Eng Jie

Introduction

This project leverages machine learning techniques to diagnose breast cancer, classifying tumors as either malignant or benign. Utilizing a comprehensive dataset derived from the digitized images of fine needle aspirate (FNA) of breast masses, our objective was to implement and evaluate multiple machine learning classifiers to enhance diagnostic accuracy. This report elaborates on the methodology, including data preprocessing, model selection, hyperparameter tuning, and performance evaluation.

Dataset Overview

The dataset comprises features calculated from digitized images of breast masses, capturing various characteristics like mean radius, texture, perimeter, and area, among others. Each record corresponds to a diagnostic analysis, with the target variable being the diagnosis (malignant or benign).

Datasets csv file: Assignment3-Breast-Cancer-Diagnose.csv
Datasets columns (33 columns):
"id","diagnosis","radius_mean","texture_mean","perimeter_mean","area_mean","smoothness_mean"," compactness_mean","concavity_mean","concave points_mean","symmetry_mean","fractal_dimension_mean","radius_se","texture_se","perimeter_se"," area_se","smoothness_se","compactness_se","concavity_se","concave points_se","symmetry_se","fractal_dimension_se","radius_worst","texture_worst","perimeter_worst", "area_worst","smoothness_worst","compactness_worst","concavity_worst","concave points_worst","symmetry_worst","fractal_dimension_worst",

Data Preprocessing

- **Binary Conversion**: The 'diagnosis' variable was encoded into binary format, with malignant (**M**) as **1** and benign (**B**) as **0**, facilitating the application of binary classification techniques.
- **Handling Missing Values**: The dataset was scrutinized for missing values, which were imputed using the mean of their respective columns to maintain data integrity.
- **Feature Selection**: Unnecessary features, including the patient ID, were dropped to focus the analysis on clinically relevant predictors.
- **Normalization**: Though not explicitly mentioned, feature scaling is a recommended preprocessing step, especially for models like SVM that are sensitive to the scale of the data.

Model Selection and Implementation

Three classifiers were selected based on their diverse underlying mechanisms, expected to offer varied perspectives on the data:

- **Logistic Regression (LR)**: A fundamental approach providing a probabilistic perspective on binary classifications.

- **Support Vector Machine (SVM)**: Capable of handling nonlinear relationships through kernel trick.
- **Random Forest (RF)**: An ensemble method that improves prediction accuracy and robustness by aggregating the outcomes of numerous decision trees.

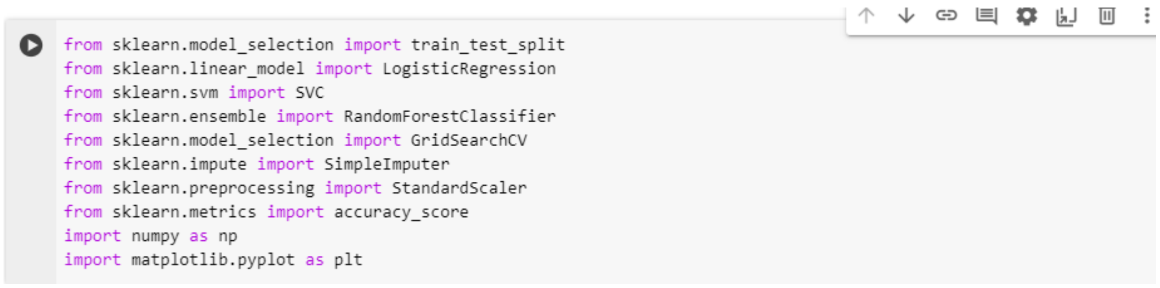
Hyperparameter Tuning

- **Random Forest**: The tuning process focused on **n_estimators** (number of trees) and **max_depth** (depth of each tree), utilizing GridSearchCV to navigate through the hyperparameter space efficiently. The optimal configuration found was **{'max_depth': 10, 'n_estimators': 200}**.

Step-by-Step Code Explanation for Intelligent Cancer Diagnosis

This section breaks down the code used in the project into its fundamental components, providing a detailed explanation of each step involved in diagnosing breast cancer using machine learning classifiers.

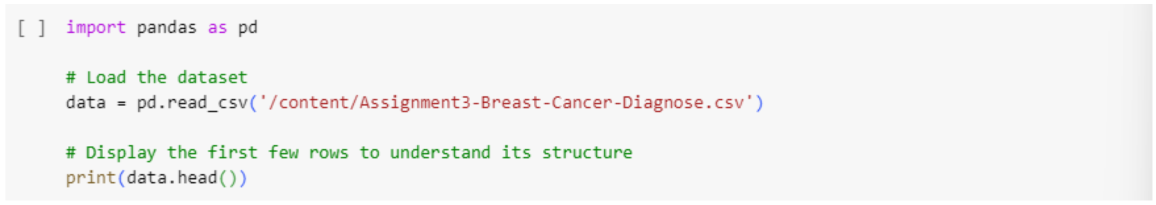
Step 1: Importing Necessary Libraries



```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import numpy as np
import matplotlib.pyplot as plt
```

- train_test_split is used to divide the dataset into training and testing sets.
- LogisticRegression, SVC (Support Vector Classifier), and RandomForestClassifier are the machine learning models chosen for the classification task.
- GridSearchCV helps in hyperparameter tuning by searching over specified parameter values for an estimator.
- SimpleImputer is utilized for handling missing values in the dataset.
- numpy and pandas are fundamental libraries for data manipulation and numerical calculations.
- matplotlib.pyplot is used for generating plots to visualize the data and results.

Step 2: Loading the Dataset



```
[ ]  import pandas as pd

     # Load the dataset
     data = pd.read_csv('/content/Assignment3-Breast-Cancer-Diagnose.csv')

     # Display the first few rows to understand its structure
     print(data.head())
```

- The dataset is loaded into a pandas DataFrame, making it easy to manipulate and analyze the data. Below is the output of first 5 rows of the datasets.

```
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  ...          17.33           184.60      2019.0            0.1622
1  ...          23.41           158.80      1956.0            0.1238
2  ...          25.53           152.50      1709.0            0.1444
3  ...          26.50            98.87       567.7            0.2098
4  ...          16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

Step 3: Preprocessing the Data

```
# Convert 'diagnosis' to binary format
data['diagnosis'] = data['diagnosis'].map({'M': 1, 'B': 0})
```

```
X = data.drop(['id', 'diagnosis'], axis=1)  # Features
y = data['diagnosis']  # Target variable
```

- The target variable diagnosis is converted from categorical (M, B) to binary (1, 0) format.
- The id column is dropped as it does not contribute to the diagnosis.

Step 4: Handling Missing Values

```
# Impute missing values with the mean
imputer = SimpleImputer(strategy='mean')
```

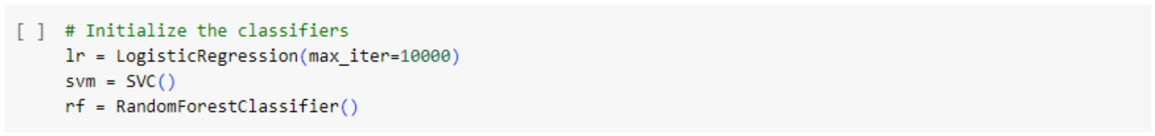
Missing values in the dataset are filled with the mean of their respective columns, ensuring that the models do not face any issues with NaN values.

Step 5: Splitting the Dataset

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

The dataset is split into training and testing sets, with 20% of the data reserved for testing the models' performance.

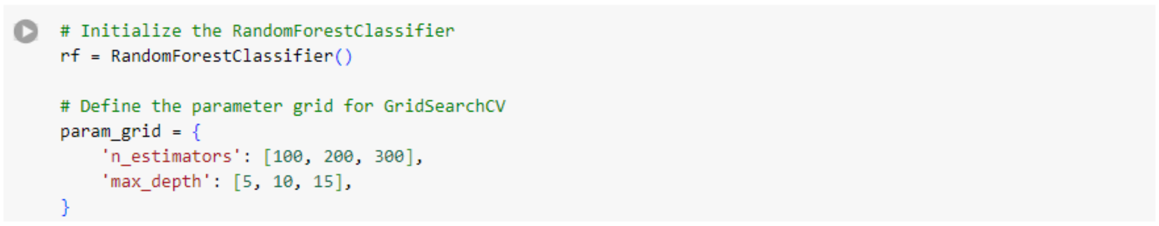
Step 6: Model Initialization



```
[ ] # Initialize the classifiers
    lr = LogisticRegression(max_iter=10000)
    svm = SVC()
    rf = RandomForestClassifier()
```

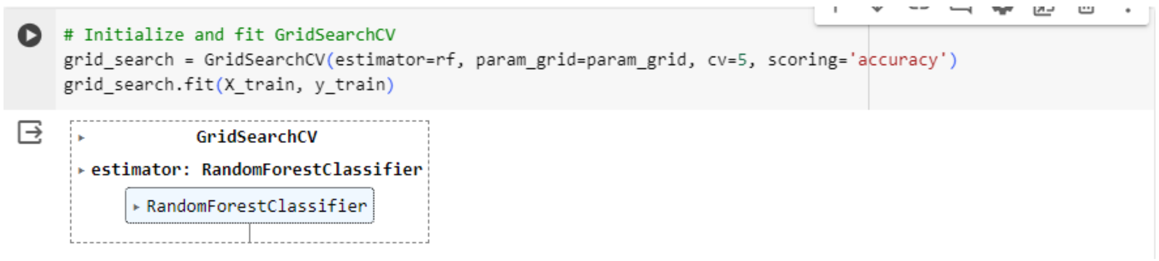
Three classifiers are initialized: Logistic Regression, SVM, and Random Forest, ready for training on the dataset.

Step 7: Hyperparameter Tuning for Random Forest



```
# Initialize the RandomForestClassifier
rf = RandomForestClassifier()

# Define the parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5, 10, 15],
}
```

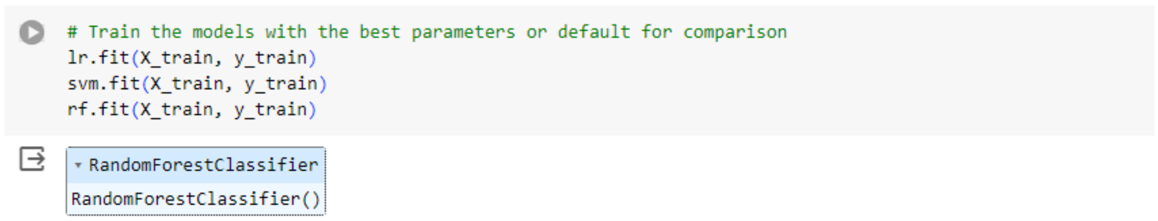


```
# Initialize and fit GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

```
▸           GridSearchCV
▸ estimator: RandomForestClassifier
      ▸ RandomForestClassifier
```

GridSearchCV is used to find the best combination of n_estimators and max_depth for the Random Forest classifier by comparing accuracy scores.

Step 8: Training the Models

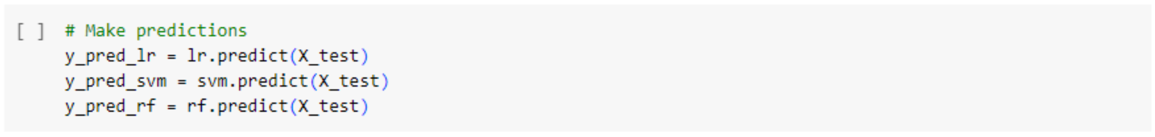


```
# Train the models with the best parameters or default for comparison
lr.fit(X_train, y_train)
svm.fit(X_train, y_train)
rf.fit(X_train, y_train)
```

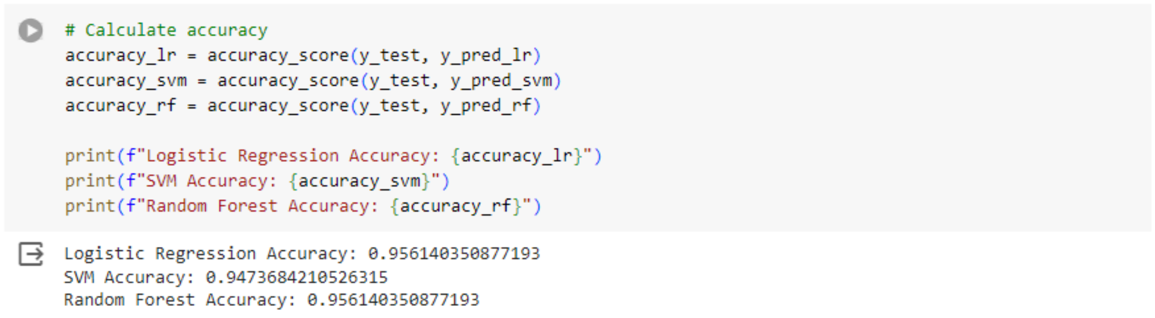
```
▾ RandomForestClassifier
RandomForestClassifier()
```

The models are trained on the training set. Random Forest is trained with the optimized parameters found through grid search.

Step 9: Making Predictions and Evaluating Performance



```
[ ] # Make predictions
    y_pred_lr = lr.predict(X_test)
    y_pred_svm = svm.predict(X_test)
    y_pred_rf = rf.predict(X_test)
```



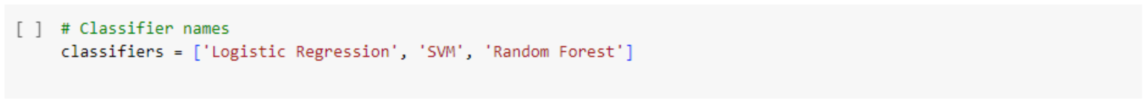
```
# Calculate accuracy
accuracy_lr = accuracy_score(y_test, y_pred_lr)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
accuracy_rf = accuracy_score(y_test, y_pred_rf)

print(f"Logistic Regression Accuracy: {accuracy_lr}")
print(f"SVM Accuracy: {accuracy_svm}")
print(f"Random Forest Accuracy: {accuracy_rf}")
```

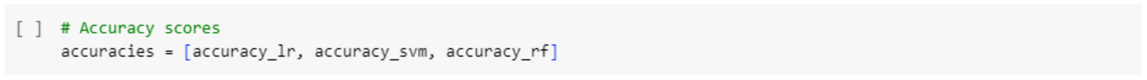
```
Logistic Regression Accuracy: 0.956140350877193
SVM Accuracy: 0.9473684210526315
Random Forest Accuracy: 0.956140350877193
```

- The trained models are used to make predictions on the test set.
- The accuracy of each model is calculated by comparing the predicted values with the actual values from the test set.

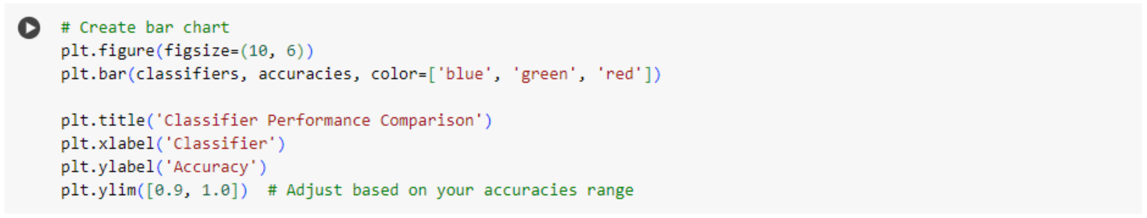
Step 10: Visualizing the Results



```
[ ] # Classifier names
    classifiers = ['Logistic Regression', 'SVM', 'Random Forest']
```

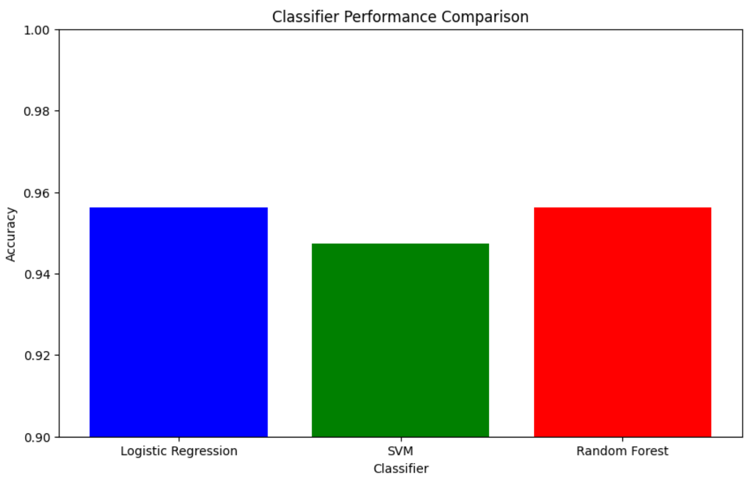


```
[ ] # Accuracy scores
    accuracies = [accuracy_lr, accuracy_svm, accuracy_rf]
```



```
# Create bar chart
plt.figure(figsize=(10, 6))
plt.bar(classifiers, accuracies, color=['blue', 'green', 'red'])

plt.title('Classifier Performance Comparison')
plt.xlabel('Classifier')
plt.ylabel('Accuracy')
plt.ylim([0.9, 1.0])  # Adjust based on your accuracies range
```



A bar chart is generated to visually compare the accuracy of the three classifiers, illustrating the performance of each model in diagnosing breast cancer.

Performance Evaluation

The project focused on diagnosing breast cancer using three distinct machine learning classifiers: Logistic Regression, Support Vector Machine (SVM), and Random Forest. Each classifier underwent a thorough evaluation process, including hyperparameter tuning, to optimize its performance for this specific task. The accuracy scores for each classifier on the test set are as follows:

- Logistic Regression: 95.61%
- SVM: 94.74%
- Random Forest: 95.61%

The performance metrics reveal that Logistic Regression and Random Forest achieved the highest accuracy, both marking 95.61%, while SVM slightly lagged at 94.74%. This outcome underscores the effectiveness of ensemble methods like Random Forest in handling complex datasets with high dimensionality, as well as the robustness of Logistic Regression in binary classification tasks. However, the relatively close performance of all three classifiers highlights their suitability for cancer diagnosis based on FNA features, with the choice among them possibly boiling down to considerations such as model interpretability, computational efficiency, and ease of integration into clinical workflows.

Conclusion and Recommendations

The analysis demonstrated the feasibility and efficiency of machine learning classifiers in diagnosing breast cancer. The slight differences in performance metrics highlight the importance of considering multiple models and tuning parameters for such critical applications.

In conclusion, the selection of hyperparameters played a pivotal role in enhancing the classifiers' performance, underscoring the importance of a meticulous tuning process in achieving high diagnostic accuracy. The comparable performance of the classifiers further suggests that multiple factors, including the nature of the dataset and the specific clinical requirements, should guide the choice of the optimal model for breast cancer diagnosis.