Lab Report 9

Dataset: Breast Cancer (Binary Classification)

Name MaazAhmad
Reg No B23F722AI170
Section Ai yellow
Submitted to Abdullah Sajid

Part A — Exploratory Data Analysis (EDA)

```
import pandas as pd
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target, name="target")

print("Dataset Shape:", X.shape)
print("\nMissing Values:\n", X.isnull().sum())
print("\nTarget Distribution:\n", y.value_counts())

X.describe()
```

Show hidden output

Description

This code loads the Breast Cancer dataset, converts it to a DataFrame, and checks its shape, missing values, and target distribution. It gives a basic understanding of how many samples/features you have and whether the data needs cleaning. It also describes the dataset statistically using describe().

Train/Test Split

```
from sklearn.model_selection import train_test_split

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

Description

This code separates the dataset into training and testing sets using an 80/20 split. Stratification ensures both sets have the same class distribution (malignant/benign), improving fairness and evaluation quality.

Part B — Baseline Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
```

```
clf = RandomForestClassifier(
    n_estimators=100,
    max_features='sqrt',
    oob_score=True,
    random_state=42
)

clf.fit(X_train, y_train)

print("OOB Score:", clf.oob_score_)
print("Train Accuracy:", accuracy_score(y_train, clf.predict(X_train)))
print("Test Accuracy:", accuracy_score(y_test, clf.predict(X_test)))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, clf.predict(X_test)))
print("\nClassification Report:\n", classification_report(y_test, clf.predict(X_test)))
```

Description

This code trains a Random Forest with default good settings. It prints the OOB score, training accuracy, test accuracy, confusion matrix, and classification report. This gives a baseline model to compare against and helps evaluate precision, recall, and F1 score.

Feature Importance (MDI)

```
importances = clf.feature_importances_
fi_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
}).sort_values('Importance', ascending=False)

plt.figure(figsize=(10,6))
plt.barh(fi_df['Feature'].head(15), fi_df['Importance'].head(15))
plt.title("Top 15 Feature Importances (MDI)")
plt.gca().invert_yaxis()
plt.show()
```

Description

This code computes Mean Decrease in Impurity (MDI) feature importance and plots the top features. It shows which variables the Random Forest relies on the most during splitting. Higher values mean more influence on decisions.

Part C — Permutation Importance

```
from sklearn.inspection import permutation_importance

perm = permutation_importance(
    clf, X_test, y_test, n_repeats=10, random_state=42
)

perm_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': perm.importances_mean
}).sort_values('Importance', ascending=False)

plt.figure(figsize=(10,6))
plt.barh(perm_df['Feature'].head(15), perm_df['Importance'].head(15))
plt.title("Permutation Importance (Top 15)")
plt.gca().invert_yaxis()
plt.show()

print(perm_df)
```

Description

This code measures the drop in model accuracy when each feature is randomly shuffled. It ranks features based on how much shuffling harms performance. It helps identify true predictive features and compare with MDI results.

Part D — Hyperparameter Tuning (GridSearchCV)

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10],
    'max_features': ['sqrt', 'log2'],
    'min_samples_leaf': [1, 2, 3]
}

grid = GridSearchCV(
    RandomForestClassifier(oob_score=True, random_state=42),
    param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train, y_train)

best = grid.best_estimator_

print("\nBest Params:", grid.best_params_)
print("Best CV Score:", grid.best_score_)
print("Best OOB Score:", best.oob_score_)
print("Test Accuracy:", accuracy_score(y_test, best.predict(X_test)))
```

Description

This code performs a GridSearch over multiple Random Forest parameters like n_estimators, max_depth, max_features, and min_samples_leaf. It finds the best combination using cross-validation. It then prints best parameters, CV accuracy, OOB score, and final test accuracy.

Validation Plot (optional)

```
results = pd.DataFrame(grid.cv_results_)
plt.plot(results['param_n_estimators'], results['mean_test_score'])
plt.xlabel("n_estimators")
plt.ylabel("CV Accuracy")
plt.title("n_estimators vs Accuracy")
plt.show()
```

Show hidden output

Next steps: Explain error

Description

This code plots how CV accuracy changes with different n_estimators values, helping understand if more trees improve the model or not. It visually shows learning behavior.

Part E — Bagging vs Random Forest vs Boosting

Bagging

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

bag = BaggingClassifier(
    estimator=DecisionTreeClassifier(random_state=42),
    n_estimators=100,
    bootstrap=True,
    oob_score=True,
    random_state=42,
    n_jobs=-1
)

bag.fit(X_train, y_train)

print("Bagging Test Accuracy:", accuracy_score(y_test, bag.predict(X_test)))
print("Bagging OOB Score:", bag.oob_score_)
```

Show hidden output

Next steps: Explain error

Description

This code trains a BaggingClassifier using a Decision Tree. It provides a comparison point showing how pure bagging performs without feature randomness. Useful to see RF improvements.

AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier

ada = AdaBoostClassifier(n_estimators=100, random_state=42)
ada.fit(X_train, y_train)

print("AdaBoost Test Accuracy:", accuracy_score(y_test, ada.predict(X_test)))
```

Show hidden output

Next steps: Explain error

Description

This code trains an AdaBoost classifier with 100 weak learners. It shows how boosting focuses on misclassified samples and compares its accuracy with bagging and RF.

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(random_state=42)
gb.fit(X_train, y_train)

print("Gradient Boosting Test Accuracy:", accuracy_score(y_test, gb.predict(X_test)))
```

Show hidden output

Next steps: Explain error

Description

This code trains a GradientBoostingClassifier, another boosting technique. It compares performance with AdaBoost, Bagging, and RF, useful for evaluating bias–variance trade-offs.

Lab Summary

1. The lab introduced Random Forests and demonstrated how they reduce overfitting through bootstrap sampling and feature randomness.
2. The Breast Cancer dataset was explored through EDA, checking shape, missing values, and class distribution.
3. A baseline Random Forest model was trained using default parameters and evaluated using OOB score, training accuracy, and test accuracy.
4. Performance metrics such as confusion matrix and classification report (precision, recall, F1 score) were generated for deeper evaluation.
5. MDI (Mean Decrease in Impurity) feature importance was computed and visualized to identify the most influential features.
6. Permutation Importance was applied to further validate feature importance by measuring accuracy drop when features were shuffled.
7. Hyperparameter tuning was performed using GridSearchCV to optimize parameters like n_estimators, max_depth, max_features, and min_samples_leaf.
8. The best model's cross-validation accuracy, OOB score, and final test accuracy were compared to the baseline model.
9. Bagging, Random Forest, and Boosting (AdaBoost/Gradient Boosting) were compared in terms of accuracy, learning behavior, and interpretability to understand the strengths of each ensemble method.
