ASSIGNMENT 4

Name: Vaishnavi Gosavi **Roll No:** 71
Year & Class: TY CSAI-A **Batch:** 2

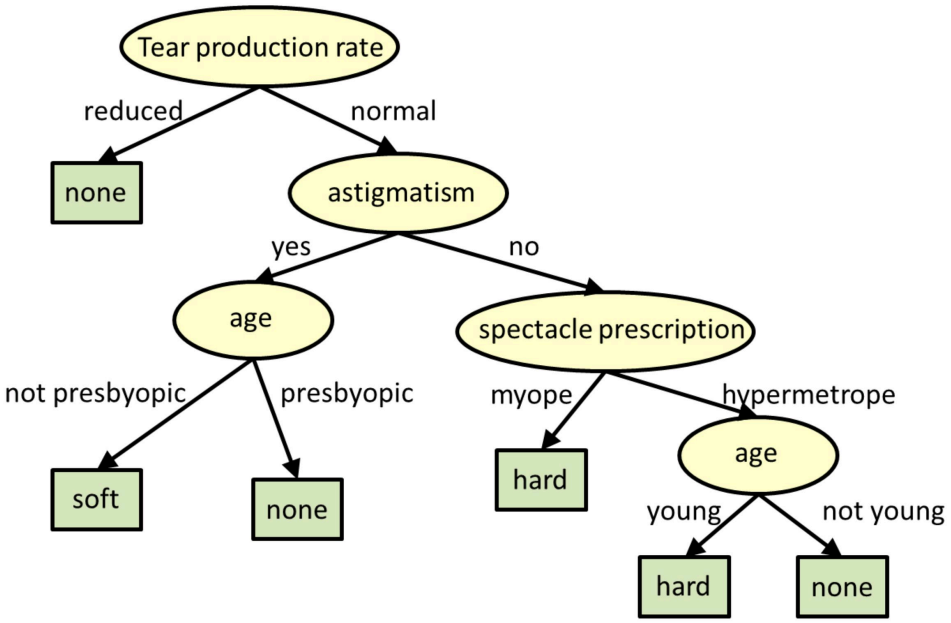
Title: Learn Decision trees for regression and classification problem

a. Split the data set into training and test sets.
b. Build the decision tree
c. Check model performances on training and test data sets.
d. Apply cost complexity pruning to overcome overfitting problem
e. Apply Random Forest algorithm to overcome overfitting problem.
f. Apply Ada-boost ensemble method on Decision stumps.

Description:-

1. Introduction

Decision Trees are supervised learning models used for both classification and regression tasks. They work by splitting the data into subsets based on feature values, forming a tree-like structure. The tree consists of decision nodes, branches, and leaf nodes, where each internal node represents a decision based on a feature, branches represent possible outcomes, and leaf nodes contain the final predictions.



2. Steps Involved in the Practical

a. Split the Dataset into Training and Test Sets

Before training a Decision Tree, the dataset is divided into training and testing sets. This ensures that the model is trained on one portion of the data and evaluated on unseen data to check its generalization capability. Typically, the data is split in an 80:20 or 70:30 ratio for training and testing.

b. Build the Decision Tree

A Decision Tree splits the dataset into subsets based on conditions at each node using:

- **Gini Index / Entropy (for classification)**
- **Mean Squared Error (MSE) (for regression)**

c. Check Model Performance on Training and Test Data

After training the Decision Tree, its performance is evaluated using appropriate metrics:

- **For classification:** Accuracy, Precision, Recall, F1-score
- **For regression:** Mean Squared Error (MSE), R-squared (R^2)

A well-performing model should generalize well to unseen data. If a model performs well on training data but poorly on test data, it indicates *overfitting*. Conversely, if the model is too simple and fails to capture patterns, it leads to *underfitting*.

d. Apply Cost Complexity Pruning to Overcome Overfitting

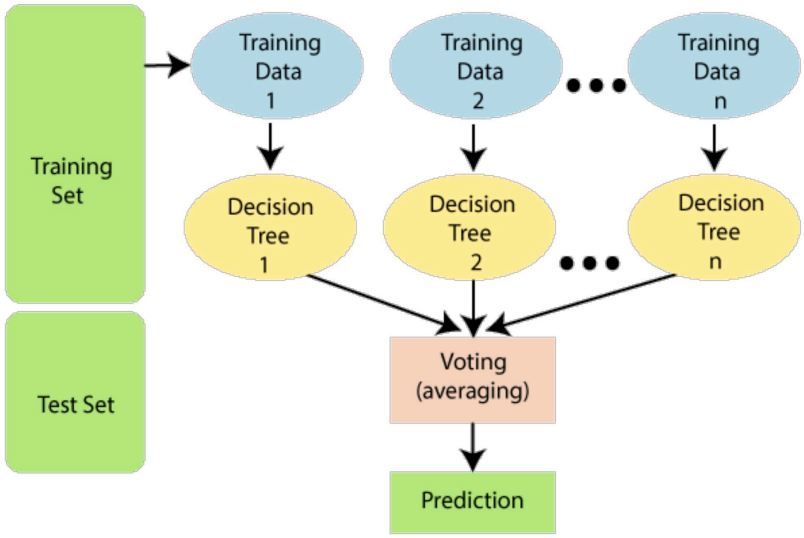
Overfitting occurs when the Decision Tree becomes too complex, capturing noise instead of meaningful patterns. *Pruning* helps reduce overfitting by removing unnecessary nodes.

- **Pre-Pruning (Early Stopping):** Stops tree growth if further splitting does not improve performance.
- **Post-Pruning:** Removes weak nodes after training based on a pruning parameter (ccp_alpha in scikit-learn).

e. Apply Random Forest Algorithm to Overcome Overfitting

Random Forest is an ensemble learning method that constructs multiple Decision Trees and combines their outputs. It reduces overfitting by using:

- **Bootstrap Aggregation (Bagging):** Training each tree on random subsets of data.
- **Feature Randomization:** Selecting a random subset of features for splitting. Final prediction is by *majority voting (classification)* or *averaging (regression)*.



f. Apply AdaBoost Ensemble Method on Decision Stumps

AdaBoost (Adaptive Boosting) is an ensemble method that combines multiple weak learners (Decision Stumps – trees with only one split) to create a strong learner. It assigns higher weights to misclassified instances, allowing subsequent trees to focus on difficult cases. The final prediction is based on a weighted majority vote.

Algorithm

1. Load the dataset and split it into training and test sets.
2. Build a Decision Tree for regression and classification.
3. Evaluate model performance on both training and test datasets.
4. Apply Cost Complexity Pruning to avoid overfitting.
5. Use Random Forest to improve performance.
6. Apply AdaBoost using Decision Stumps.

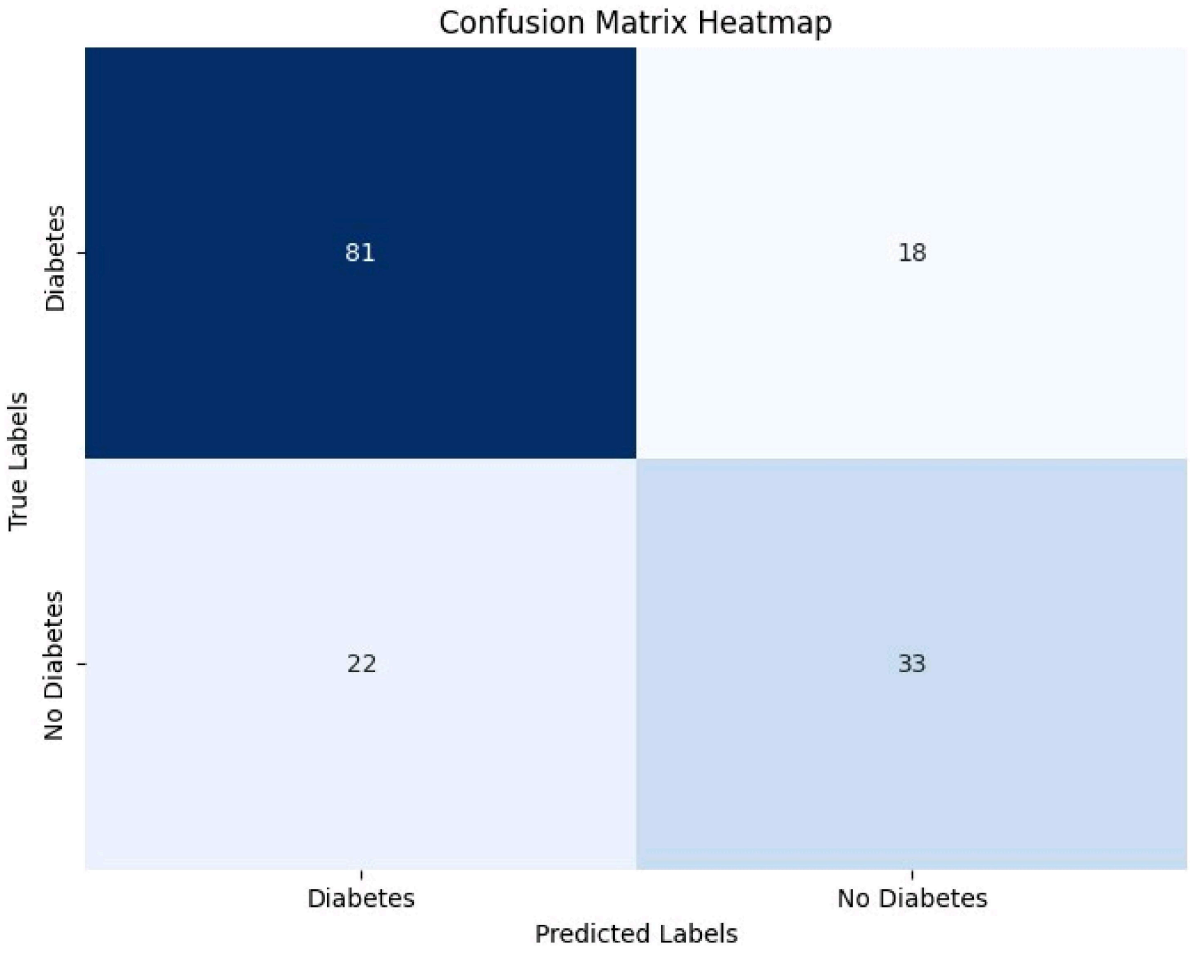
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

df = pd.read_csv("diabetes.csv")
df

df.head()

feature_cols =
['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BM
I', 'DiabetesPedigreeFunction','Age']
X = df[feature_cols]
y = df.Outcome

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

X_train

X_test

y_train

y_test

clf = DecisionTreeClassifier(criterion='entropy', max_depth = 3)

clf = clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print("Accuracy: ", metrics.accuracy_score(y_test, y_pred))

print("Training accuracy:", clf.score(X_train, y_train))
print("Testing accuracy:", clf.score(X_test, y_test))

from sklearn.metrics import confusion_matrix, precision_score,
recall_score, f1_score

cm = confusion_matrix(y_test,y_pred)
cm

#Precision
precision = precision_score(y_test, y_pred)
precision

#Recall
recall = recall_score(y_test, y_pred)
recall
```

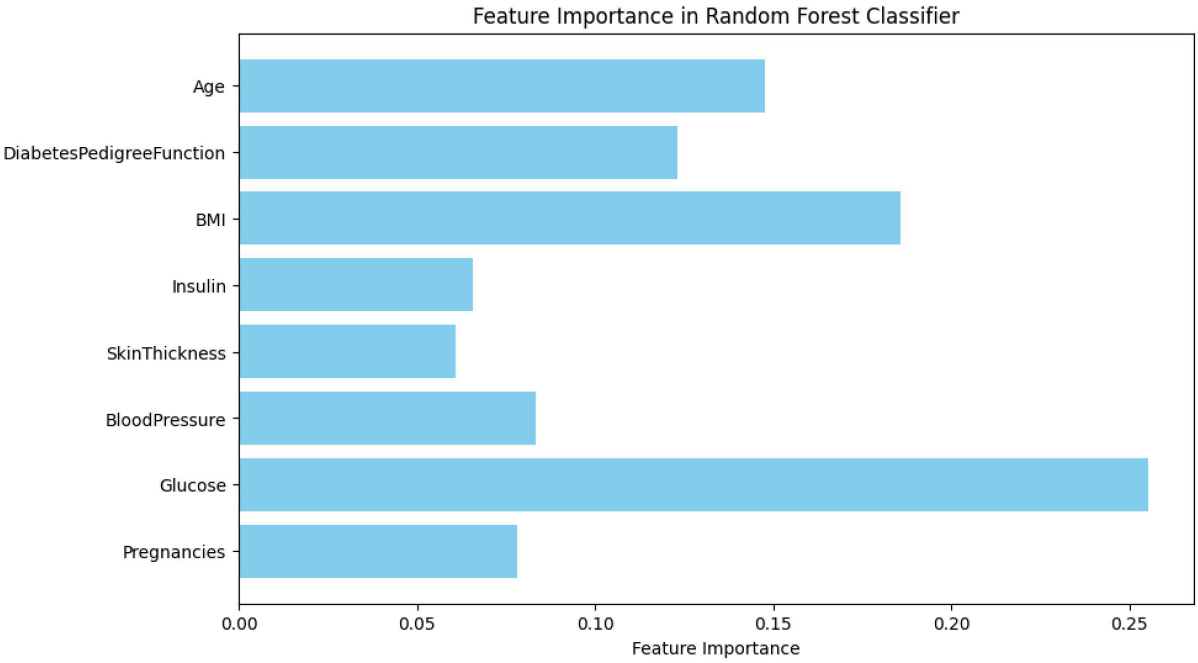
```python
#F1 Score
f1 = f1_score(y_test, y_pred)
f1

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, roc_auc_score
from collections import Counter
from sklearn.model_selection import GridSearchCV

#Alpha Pruning
alpha_congifs = {
    'No Pruning (alpha=0)':0,
    '(alpha=0.001)':0.003,
    '(alpha=0.01)':0.01,
    '(alpha=0.1)':0.1
}

param_grid = {'max_depth': range(1, 16)}
grid_search = GridSearchCV(DecisionTreeClassifier(random_state=42),
param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_depth = grid_search.best_params_['max_depth']

clf_pruned = DecisionTreeClassifier(max_depth =
best_depth,min_samples_split=4, random_state=42)
clf_pruned.fit(X_train, y_train)

train_accuracy_pruned = metrics.accuracy_score(y_train,
clf_pruned.predict(X_train))
test_accuracy_pruned = metrics.accuracy_score(y_test,
clf_pruned.predict(X_test))

print(f"Optimal Depth: {best_depth}")
print(f"Training Accuracy (Pruned): {train_accuracy_pruned}")
print(f"Test Accuracy (Pruned): {test_accuracy_pruned}")

#Applying Random Forest Algorithm to overcome overfitting
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators = 10, criterion =
'entropy', random_state = 42)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

```
import seaborn as sns

accuracy = metrics.accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

class_labels = ['Diabetes', 'No Diabetes']

conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues',
cbar=False,
            xticklabels=class_labels, yticklabels=class_labels)

plt.title('Confusion Matrix Heatmap')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

import matplotlib.pyplot as plt

feature_importances = classifier.feature_importances_

plt.figure(figsize=(10, 6))
plt.barh(feature_cols, feature_importances, color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Feature Importance in Random Forest Classifier')
plt.show()


#Apply Ada Boost Ensemble Method on Decision Stumps
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.inspection import DecisionBoundaryDisplay

base_estimator = DecisionTreeClassifier(max_depth=1, random_state=42)

ada = AdaBoostClassifier(estimator=base_estimator, n_estimators=50,
learning_rate=1.0, random_state=42)
ada.fit(X_train, y_train)

y_pred = ada.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Test Accuracy: {:.2f}%".format(accuracy * 100))
print("\nClassification Report:\n", classification_report(y_test,
y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix
```

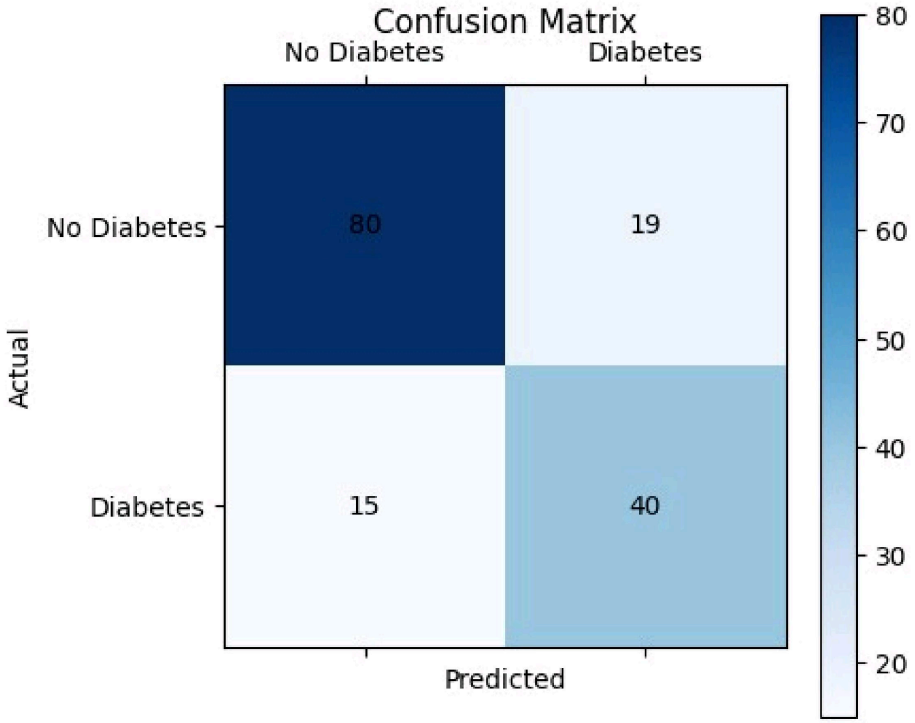
```
conf_matrix = confusion_matrix(y_test, y_pred)

labels = ['No Diabetes', 'Diabetes']

plt.figure(figsize=(6, 4))

cax = plt.matshow(conf_matrix, cmap='Blues')
plt.title("Confusion Matrix", pad=20)

plt.colorbar(cax)

plt.xlabel("Predicted")
plt.ylabel("Actual")

plt.xticks([0, 1], labels)
plt.yticks([0, 1], labels)

for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        plt.text(j, i, str(conf_matrix[i, j]),
                 ha='center', va='center', color='black')

plt.show()
```

```
Accuracy:  0.7662337662337663
Training accuracy: 0.7752442996742671
Testing accuracy: 0.7662337662337663
Optimal Depth: 3
Training Accuracy (Pruned): 0.7768729641693811
Test Accuracy (Pruned): 0.7597402597402597
Accuracy: 74.03%
```


Confusion Matrix Heatmap
81
18
22
33
Diabetes
No Diabetes
True Labels
Diabetes
No Diabetes
Predicted Labels


Feature Importance in Random Forest Classifier
Age
DiabetesPedigreeFunction
BMI
Insulin
SkinThickness
BloodPressure
Glucose
Pregnancies
0.00
0.05
0.10
0.15
0.20
0.25
Feature Importance

```
Test Accuracy: 77.92%

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.81      0.82        99
           1       0.68      0.73      0.70        55

    accuracy                           0.78       154
   macro avg       0.76      0.77      0.76       154
weighted avg       0.78      0.78      0.78       154


Confusion Matrix:
 [[80 19]
 [15 40]]

<Figure size 600x400 with 0 Axes>
```



Conclusion

Decision Trees are powerful models for classification and regression tasks. However, they can suffer from overfitting, which can be mitigated using techniques such as pruning, Random Forest, and AdaBoost. By applying these methods, we can build more robust and generalizable models for real-world problems.