JUNAID GIRKAR
60004190057
TE COMPS A4

DMW
EXPERIMENT 2

THEORY

Naive Bayes:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training dataset. It is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object. Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for text classification problems.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

DECISION TREE:

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

Some **advantages** of decision trees are:

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. However scikit-learn implementation does not support categorical variables for now. Other techniques are usually specialised in analysing datasets that have only one type of variable.
- Able to handle multi-output problems.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The **disadvantages** of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- Predictions of decision trees are neither smooth nor continuous, but piecewise constant approximations as seen in the above figure. Therefore, they are not good at extrapolation.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.

- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

DATASET 1: Diabetes

CODE:

```
import pandas as pd
df = pd.read_csv('diabetes.csv')

df.head()

df.isnull().sum()

from sklearn.model_selection import train_test_split

X=df.drop(columns=['Outcome'])
y=df['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

print("NAIVE BAYERS CLASSIFICATION")

from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()

nb.fit(X_train,y_train)

nb.score(X_test,y_test)

y_pred = nb.predict(X_test)

from sklearn.metrics import confusion_matrix,classification_report

print("Confusion Matrix")
confusion_matrix(y_test,y_pred)

print("Classification Report")
print(classification_report(y_test,y_pred))

X=df.drop(columns=['Outcome'])
```

```
y=df['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

from sklearn import tree
dt = tree.DecisionTreeClassifier()

print("\nDECISION TREE CLASSIFICATION")
dt.fit(X_train,y_train)
print("Testing Score")
dt.score(X_test,y_test)

y_pred_dt = dt.predict(X_test)

print("Confusion Matrix")
confusion_matrix(y_test,y_pred_dt)

print("Classification Report")
print(classification_report(y_test,y_pred_dt))

nb_probs = nb.predict_proba(X_test)
dt_probs = dt.predict_proba(X_test)

dt_probs = dt_probs[:, 1]
nb_probs = nb_probs[:, 1]
nb_probs

from sklearn.metrics import roc_curve, roc_auc_score

nb_auc = roc_auc_score(y_test, nb_probs)
dt_auc = roc_auc_score(y_test, dt_probs)

print('Decision Tree AUROC = ' + str(dt_auc))
print('Naive Bayes AUROC = ' + str(nb_auc))

nb_fpr, nb_tpr, _ = roc_curve(y_test, nb_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_probs)

import matplotlib.pyplot as plt

plt.plot(nb_fpr, nb_tpr, linestyle='--', label='Naive Bayes (AUROC
= %0.3f)' % nb_auc)
```

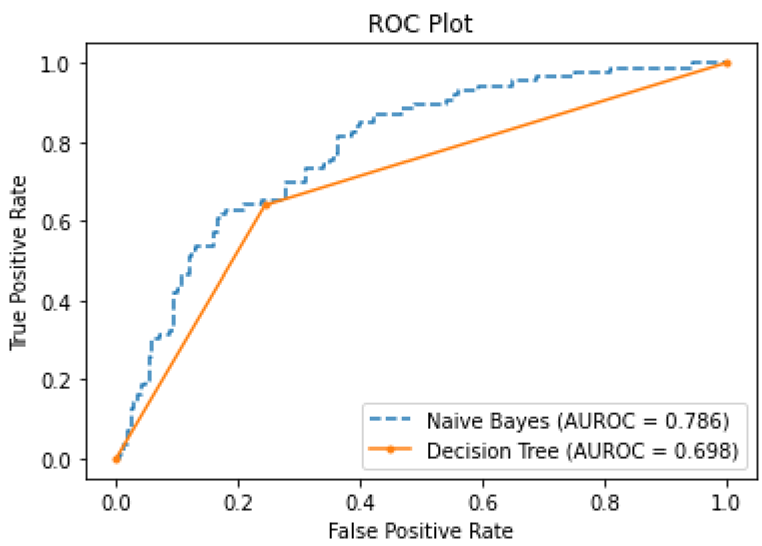
```
plt.plot(dt_fpr, dt_tpr, marker='.', label='Decision Tree (AUROC =
%0.3f)' % dt_auc)


# Title
plt.title('ROC Plot')
# Axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# Show legend
plt.legend() #
# Show plot
plt.show()
```

OUTPUT:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
NAIVE BAYERS CLASSIFICATION
Confusion Matrix
Classification Report
              precision    recall  f1-score   support

           0       0.81      0.79      0.80       168
           1       0.61      0.63      0.62        86

    accuracy                           0.74       254
   macro avg       0.71      0.71      0.71       254
weighted avg       0.74      0.74      0.74       254


DECISION TREE CLASSIFICATION
Testing Score
Confusion Matrix
Classification Report
              precision    recall  f1-score   support

           0       0.80      0.76      0.78       168
           1       0.57      0.64      0.60        86

    accuracy                           0.72       254
   macro avg       0.69      0.70      0.69       254
weighted avg       0.73      0.72      0.72       254

Decision Tree AUROC = 0.6977436323366556
Naive Bayes AUROC = 0.7857834994462902
```



DATASET 2: Sonar

CODE:

```
import pandas as pd
df = pd.read_csv('sonar.csv',header=None)
print("Showing First 5 rows of the database")
df.head()

print("Checking null fields in the dataset")
df.isnull().sum()

from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()

print("Data before using LavelEncoder")
df[60]

df[60]=le.fit_transform(df[60])
print("Data after using LavelEncoder")
df[60]


from sklearn.model_selection import train_test_split

X=df.drop(columns=[60])
y=df[60]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()

nb.fit(X_train,y_train)
print("Testing Score")
nb.score(X_test,y_test)

y_pred = nb.predict(X_test)

from sklearn.metrics import confusion_matrix,classification_report

print("Confusion Matrix for Naive Bayers")
confusion_matrix(y_test,y_pred)
```

```
print("Classification Report")
print(classification_report(y_test,y_pred))

X=df.drop(columns=[60])
y=df[60]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

print("\n\n DECISION TREE CLASSIFIER")
from sklearn import tree
dt = tree.DecisionTreeClassifier()

dt.fit(X_train,y_train)
print("Testing Score")
dt.score(X_test,y_test)

y_pred_dt = dt.predict(X_test)
print("Classification Report")
print(classification_report(y_test,y_pred_dt))

print("Confusion Matrix for Decision Tree")
confusion_matrix(y_test,y_pred_dt)
nb_probs = nb.predict_proba(X_test)
dt_probs = dt.predict_proba(X_test)

dt_probs = dt_probs[:, 1]
nb_probs = nb_probs[:, 1]
nb_probs
from sklearn.metrics import roc_curve, roc_auc_score

nb_auc = roc_auc_score(y_test, nb_probs)
dt_auc = roc_auc_score(y_test, dt_probs)

print('Decision Tree AUROC = ' + str(dt_auc))
print('Naive Bayes AUROC = ' + str(nb_auc))

nb_fpr, nb_tpr, _ = roc_curve(y_test, nb_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_probs)

import matplotlib.pyplot as plt

plt.plot(nb_fpr, nb_tpr, linestyle='--', label='Naive Bayes (AUROC
```

```
= %0.3f)' % nb_auc)
plt.plot(dt_fpr, dt_tpr, marker='.', label='Decision Tree (AUROC =
%0.3f)' % dt_auc)


# Title
plt.title('ROC Plot')
# Axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# Show legend
plt.legend() #
# Show plot
plt.show()
```

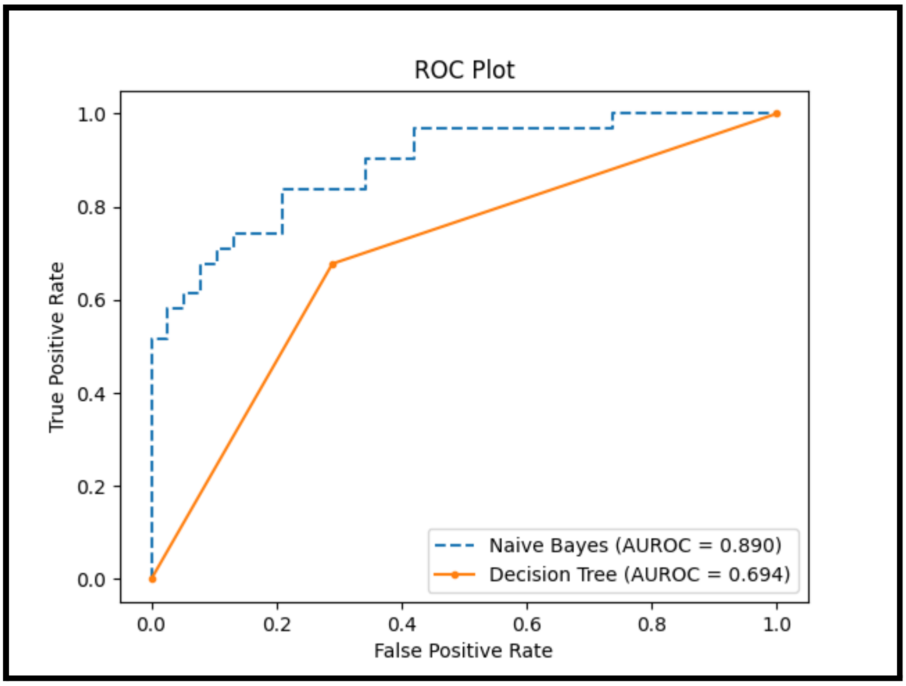
OUTPUT:

	0	1	2	3	4	5	6	7	8	9	...	51	52	53	54	55	56	57	58	59	60
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0.0065	0.0159	0.0072	0.0167	0.0180	0.0084	0.0090	0.0032	R
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0084	0.0089	0.0048	0.0094	0.0191	0.0140	0.0049	0.0052	0.0044	R
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0232	0.0166	0.0095	0.0180	0.0244	0.0316	0.0164	0.0095	0.0078	R
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0121	0.0036	0.0150	0.0085	0.0073	0.0050	0.0044	0.0040	0.0117	R
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0031	0.0054	0.0105	0.0110	0.0015	0.0072	0.0048	0.0107	0.0094	R

5 rows × 61 columns

```
Checking null fields in the dataset

0     0
1     0
2     0
3     0
4     0
     ..
56    0
57    0
58    0
59    0
60    0
Length: 61, dtype: int64
```

```
Showing First 5 rows of the database
Checking null fields in the dataset
Data before using LavelEncoder
Data after using LavelEncoder
Testing Score
Confusion Matrix for Naive Bayers
Classification Report
              precision    recall  f1-score   support

           0       0.86      0.66      0.75        38
           1       0.68      0.87      0.76        31

    accuracy                           0.75        69
   macro avg       0.77      0.76      0.75        69
weighted avg       0.78      0.75      0.75        69



 DECISION TREE CLASSIFIER
Testing Score
Classification Report
              precision    recall  f1-score   support

           0       0.76      0.82      0.78        38
           1       0.75      0.68      0.71        31

    accuracy                           0.75        69
   macro avg       0.75      0.75      0.75        69
weighted avg       0.75      0.75      0.75        69

Confusion Matrix for Decision Tree
Decision Tree AUROC = 0.74660441426146
Naive Bayes AUROC = 0.8904923599320883
```


ROC Plot
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Naive Bayes (AUROC = 0.890)
Decision Tree (AUROC = 0.694)

DATASET 3: Haberman

CODE:

```
import pandas as pd

df = pd.read_csv('haberman.csv',header=None)

df.head()
df.isnull().sum()

from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()
df[3]

df[3]=le.fit_transform(df[3])
df[3]

from sklearn.model_selection import train_test_split

X=df.drop(columns=[3])
y=df[3]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

print("NAIVE BAYERS CLASSIFICATION")

from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()

nb.fit(X_train,y_train)
print("Testing Score")
nb.score(X_test,y_test)
y_pred = nb.predict(X_test)
from sklearn.metrics import confusion_matrix,classification_report

print("Naive Bayers Confusion Matrix")
confusion_matrix(y_test,y_pred)
print("Classification Report")
print(classification_report(y_test,y_pred))

X=df.drop(columns=[3])
y=df[3]
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size=0.33, random_state=42)
print("DECISION TREE CLASSIFIER")
from sklearn import tree
dt = tree.DecisionTreeClassifier()

dt.fit(X_train,y_train)

print("Testing Score")
dt.score(X_test,y_test)

y_pred_dt = dt.predict(X_test)

print(classification_report(y_test,y_pred_dt))
confusion_matrix(y_test,y_pred_dt)

nb_probs = nb.predict_proba(X_test)
dt_probs = dt.predict_proba(X_test)

dt_probs = dt_probs[:, 1]
nb_probs = nb_probs[:, 1]
nb_probs

from sklearn.metrics import roc_curve, roc_auc_score

nb_auc = roc_auc_score(y_test, nb_probs)
dt_auc = roc_auc_score(y_test, dt_probs)

print('Decision Tree AUROC = ' + str(dt_auc))
print('Naive Bayes AUROC = ' + str(nb_auc))

nb_fpr, nb_tpr, _ = roc_curve(y_test, nb_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_probs)

import matplotlib.pyplot as plt

plt.plot(nb_fpr, nb_tpr, linestyle='--', label='Naive Bayes (AUROC
= %0.3f)' % nb_auc)
plt.plot(dt_fpr, dt_tpr, marker='.', label='Decision Tree (AUROC =
%0.3f)' % dt_auc)


# Title
```

```
plt.title('ROC Plot')
# Axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# Show legend
plt.legend() #
# Show plot
plt.show()
```

OUTPUT:

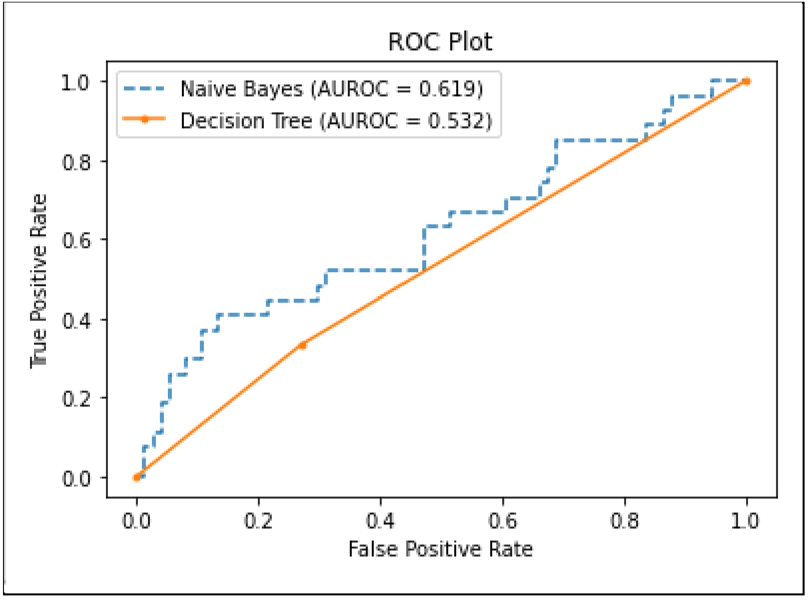
```
df.head()
```

	0	1	2	3
0	30	64	1	1
1	30	62	3	1
2	30	65	0	1
3	31	59	2	1
4	31	65	4	1

```
df.isnull().sum()
```

```
0    0
1    0
2    0
3    0
dtype: int64
```

```
NAIVE BAYERS CLASSIFICATION
Testing Score
Naive Bayers Confusion Matrix
Classification Report
              precision    recall  f1-score   support

           0       0.78      0.91      0.84        74
           1       0.53      0.30      0.38        27

    accuracy                           0.74       101
   macro avg       0.66      0.60      0.61       101
weighted avg       0.71      0.74      0.72       101

DECISION TREE CLASSIFIER
Testing Score
              precision    recall  f1-score   support

           0       0.75      0.73      0.74        74
           1       0.31      0.33      0.32        27

    accuracy                           0.62       101
   macro avg       0.53      0.53      0.53       101
weighted avg       0.63      0.62      0.63       101

Decision Tree AUROC = 0.5315315315315315
Naive Bayes AUROC = 0.6191191191191191
```



DATASET 4: Ionosphere

CODE:

```
import pandas as pd

df = pd.read_csv('ionosphere_data.csv')

df.head()

df.isnull().sum()

from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()
df['column_ai']

df['column_ai']=le.fit_transform(df['column_ai'])
df['column_ai']

from sklearn.model_selection import train_test_split

X=df.drop(columns=['column_ai'])
y=df['column_ai']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

print("NAIVE BAYERS CLASSIFICATION\n")

from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()

nb.fit(X_train,y_train)

print("Naive bayers Score:")
nb.score(X_test,y_test)

y_pred = nb.predict(X_test)

from sklearn.metrics import confusion_matrix,classification_report

print("Confusion Matrix")
confusion_matrix(y_test,y_pred)

print("Classification Report")
```

```
print(classification_report(y_test,y_pred))

X=df.drop(columns=['column_ai'])
y=df['column_ai']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

from sklearn import tree
dt = tree.DecisionTreeClassifier()

print("\n\nDECISION TREE CLASSIFIER")
dt.fit(X_train,y_train)

dt.score(X_test,y_test)

y_pred_dt = dt.predict(X_test)

print("Classification Report")
print(classification_report(y_test,y_pred_dt))

print("Confusion Matrix")
confusion_matrix(y_test,y_pred_dt)

nb_probs = nb.predict_proba(X_test)
dt_probs = dt.predict_proba(X_test)

dt_probs = dt_probs[:, 1]
nb_probs = nb_probs[:, 1]
nb_probs

from sklearn.metrics import roc_curve, roc_auc_score

nb_auc = roc_auc_score(y_test, nb_probs)
dt_auc = roc_auc_score(y_test, dt_probs)

print('Decision Tree AUROC = ' + str(dt_auc))
print('Naive Bayes AUROC = ' + str(nb_auc))

nb_fpr, nb_tpr, _ = roc_curve(y_test, nb_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_probs)

import matplotlib.pyplot as plt
```

```
plt.plot(nb_fpr, nb_tpr, linestyle='--', label='Naive Bayes (AUROC
= %0.3f)' % nb_auc)
plt.plot(dt_fpr, dt_tpr, marker='.', label='Decision Tree (AUROC =
%0.3f)' % dt_auc)


# Title
plt.title('ROC Plot')
# Axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# Show legend
plt.legend() #
# Show plot
plt.show()
```

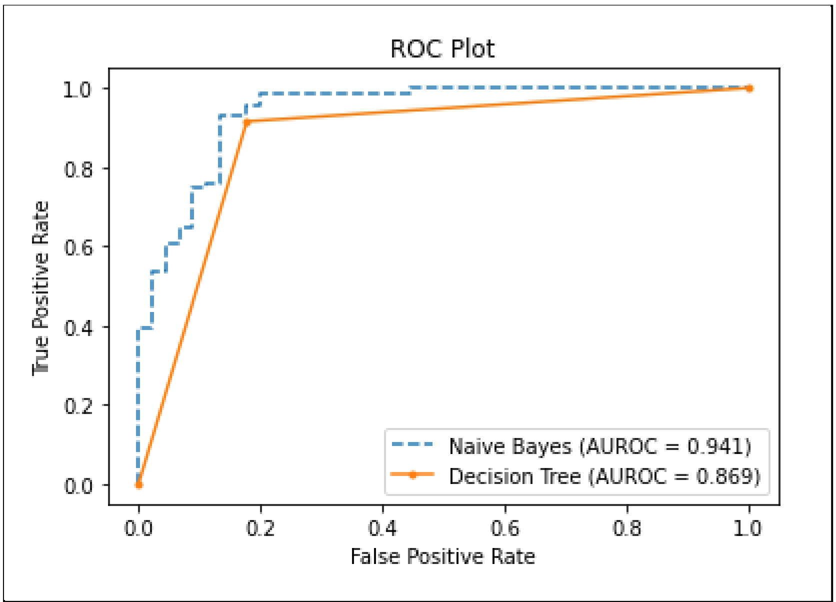
OUTPUT:

	column_a	column_b	column_c	column_d	column_e	column_f	column_g	column_h	column_i	column_j	...	column_z	column_aa	column_ab	column_
0	True	False	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.03760	...	-0.51171	0.41078	-0.46168	0.212
1	True	False	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...	-0.26569	-0.20468	-0.18401	-0.190
2	True	False	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...	-0.40220	0.58984	-0.22145	0.431
3	True	False	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	...	0.90695	0.51613	1.00000	1.000
4	True	False	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	...	-0.65158	0.13290	-0.53206	0.024

5 rows × 35 columns

```
column_a     0
column_b     0
column_c     0
column_d     0
column_e     0
column_f     0
column_g     0
column_h     0
column_i     0
column_j     0
column_k     0
column_l     0
column_m     0
column_n     0
column_o     0
column_p     0
column_q     0
column_r     0
column_s     0
column_t     0
column_u     0
column_v     0
column_w     0
column_x     0
column_y     0
column_z     0
column_aa    0
column_ab    0
column_ac    0
column_ad    0
column_ae    0
column_af    0
column_ag    0
column_ah    0
column_ai    0
dtype: int64
```

```
NAIVE BAYERS CLASSIFICATION

Naive bayers Score:
Confusion Matrix
Classification Report
              precision    recall  f1-score   support

           0       0.97      0.78      0.86        45
           1       0.88      0.99      0.93        71

    accuracy                           0.91       116
   macro avg       0.92      0.88      0.90       116
weighted avg       0.91      0.91      0.90       116



DECISION TREE CLASSIFIER
Classification Report
              precision    recall  f1-score   support

           0       0.86      0.82      0.84        45
           1       0.89      0.92      0.90        71

    accuracy                           0.88       116
   macro avg       0.88      0.87      0.87       116
weighted avg       0.88      0.88      0.88       116

Confusion Matrix
Decision Tree AUROC = 0.8688575899843505
Naive Bayes AUROC = 0.9411580594679188
```



DATASET 5: BankNote Authentication

CODE:

```
import pandas as pd

df = pd.read_csv('BankNoteAuthentication.csv')

df.head()

df.isnull().sum()

from sklearn.model_selection import train_test_split

X=df.drop(columns=['class'])
y=df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

print("NAIVE BAYERS CLASSFICATION")

from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()

nb.fit(X_train,y_train)

print("TESTING SCORE")
nb.score(X_test,y_test)

y_pred = nb.predict(X_test)

from sklearn.metrics import confusion_matrix,classification_report

print("CONFUSION MATRIX")
confusion_matrix(y_test,y_pred)

print("CLASSIFICATION REPORT")
print(classification_report(y_test,y_pred))

X=df.drop(columns=['class'])
y=df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)
```

```
print("\nDECISION TREE CLASSIFIER")
from sklearn import tree
dt = tree.DecisionTreeClassifier()

dt.fit(X_train,y_train)

print("Testing Score")
dt.score(X_test,y_test)

y_pred_dt = dt.predict(X_test)

print("Confusion Matrix")
confusion_matrix(y_test,y_pred_dt)

print("Classification Report")
print(classification_report(y_test,y_pred_dt))

nb_probs = nb.predict_proba(X_test)
dt_probs = dt.predict_proba(X_test)

dt_probs = dt_probs[:, 1]
nb_probs = nb_probs[:, 1]
nb_probs

from sklearn.metrics import roc_curve, roc_auc_score

nb_auc = roc_auc_score(y_test, nb_probs)
dt_auc = roc_auc_score(y_test, dt_probs)

print('Decision Tree AUROC = ' + str(dt_auc))
print('Naive Bayes AUROC = ' + str(nb_auc))

nb_fpr, nb_tpr, _ = roc_curve(y_test, nb_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_probs)

import matplotlib.pyplot as plt

plt.plot(nb_fpr, nb_tpr, linestyle='--', label='Naive Bayes (AUROC
= %0.3f)' % nb_auc)
plt.plot(dt_fpr, dt_tpr, marker='.', label='Decision Tree (AUROC =
%0.3f)' % dt_auc)
```

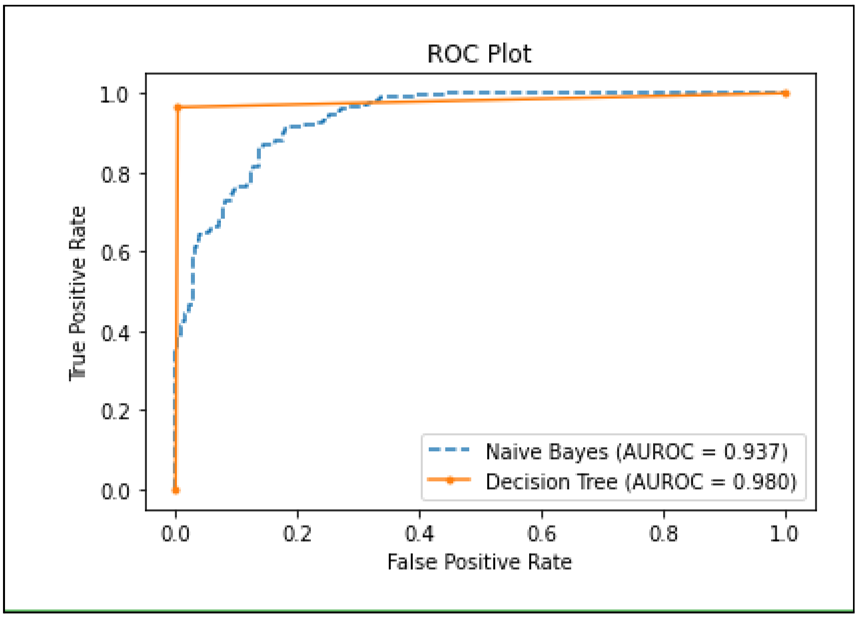
```
# Title
plt.title('ROC Plot')
# Axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# Show legend
plt.legend() #
# Show plot
plt.show()
```

OUTPUT:

	variance	skewness	curtosis	entropy	class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

```
variance    0
skewness    0
curtosis    0
entropy     0
class       0
dtype: int64
```

```
NAIVE BAYERS CLASSFICATION
TESTING SCORE
CONFUSION MATRIX
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.83      0.91      0.86       257
           1       0.86      0.75      0.80       196

    accuracy                           0.84       453
   macro avg       0.84      0.83      0.83       453
weighted avg       0.84      0.84      0.84       453


DECISION TREE CLASSIFIER
Testing Score
Confusion Matrix
Classification Report
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       257
           1       0.99      0.96      0.98       196

    accuracy                           0.98       453
   macro avg       0.98      0.98      0.98       453
weighted avg       0.98      0.98      0.98       453

Decision Tree AUROC = 0.9801973318510284
Naive Bayes AUROC = 0.9371476216945922
```

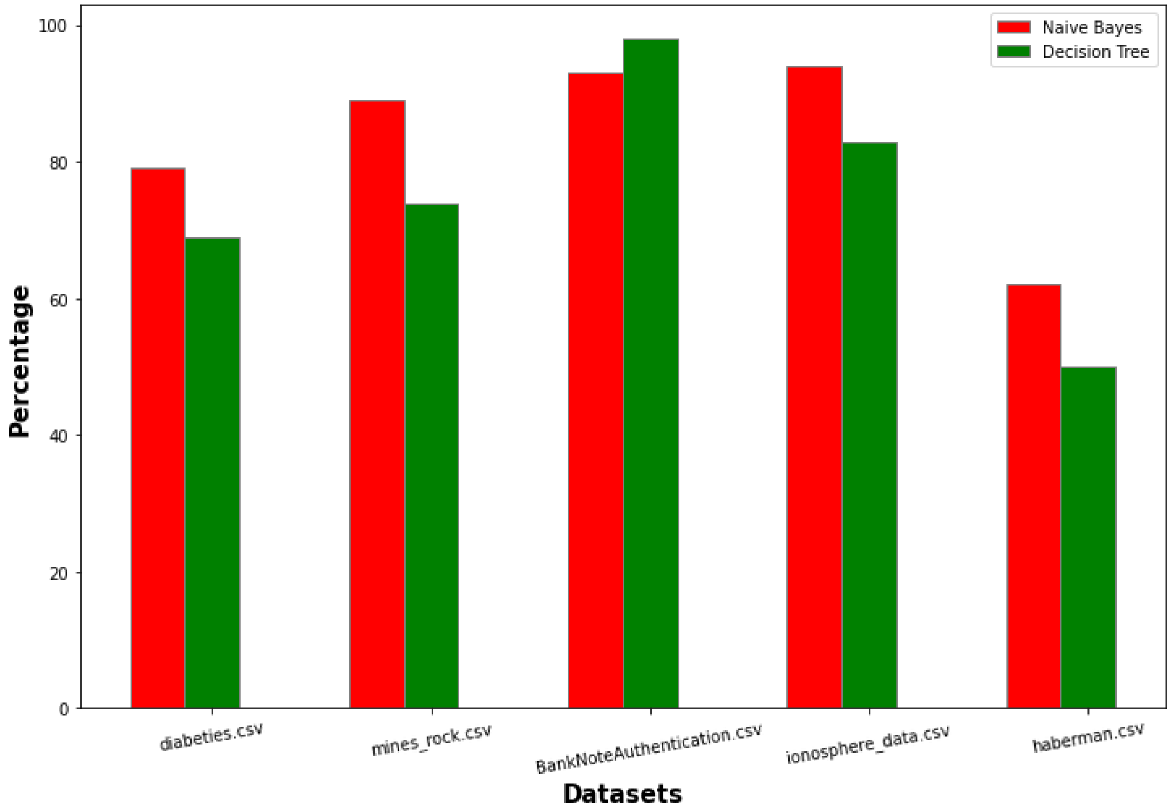


Comparison:

CODE:

```
import numpy as np
import matplotlib.pyplot as plt

barWidth = 0.25
fig = plt.subplots(figsize =(12, 8))

naive_bayes = [79, 89, 93, 94, 62]
decision_tree = [69,74, 98, 83, 50]

br1 = np.arange(len(naive_bayes))
br2 = [x + barWidth for x in br1]

plt.bar(br1, naive_bayes, color ='b', width = barWidth,
        edgecolor ='grey', label ='Naive Bayes')
plt.bar(br2, decision_tree, color ='y', width = barWidth,
        edgecolor ='grey', label ='Decision Tree')
plt.xlabel('Datasets', fontweight ='bold', fontsize = 15)
plt.ylabel('Percentage', fontweight ='bold', fontsize = 15)
plt.xticks([r+ barWidth for r in range(len(naive_bayes))],
        ['diabeties.csv', 'mines_rock.csv',
'BankNoteAuthentication.csv', 'ionosphere_data.csv',
'haberman.csv'],rotation=30)

plt.legend()
plt.show()
```

OUTPUT:


100
80
60
40
20
0
Percentage
Naive Bayes
Decision Tree
diabeties.csv
mines_rock.csv
BankNoteAuthentication.csv
ionosphere_data.csv
haberman.csv
Datasets

PART C

THEORY:

k-Fold Cross-Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.
The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.
Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.
It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.
The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
    1. Take the group as a hold out or test data set
    2. Take the remaining groups as a training data set
    3. Fit a model on the training set and evaluate it on the test set
    4. Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

Configuration of k

The k value must be chosen carefully for your data sample.
A poorly chosen value for k may result in a mis-representative idea of the skill of the model, such as a score with a high variance (that may change a lot based on the data used to fit the model), or a high bias, (such as an overestimate of the skill of the model).
Three common tactics for choosing a value for k are as follows:

- Representative: The value for k is chosen such that each train/test group of data samples is large enough to be statistically representative of the broader dataset.
- k=10: The value for k is fixed to 10, a value that has been found through experimentation to generally result in a model skill estimate with low bias a modest variance.
- k=n: The value for k is fixed to n, where n is the size of the dataset to give each test sample an opportunity to be used in the hold out dataset. This approach is called leave-one-out cross-validation.

Ensemble Learning

Ensemble learning is a general meta approach to machine learning that seeks better predictive performance by combining the predictions from multiple models.
Although there are a seemingly unlimited number of ensembles that you can develop for your predictive modeling problem, there are three methods that dominate the field of ensemble learning. So much so, that rather than algorithms per se, each is a field of study that has spawned many more specialized methods.
The three main classes of ensemble learning methods are bagging, stacking, and boosting

CODE:

```
from sklearn.model_selection import KFold, train_test_split,
cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import AdaBoostClassifier, VotingClassifier
from numpy import mean
from sklearn.metrics import accuracy_score

cv = KFold(n_splits=10, shuffle=True, random_state=1)
model = AdaBoostClassifier()
def evaluate_model(cv, model):
    X, y = get_dataset()
    scores = cross_val_score(model, X, y, scoring='accuracy',
cv=cv, n_jobs=-1)
    return np.mean(scores), scores.min(), scores.max()


def naive_bayes_classification(X_train, X_test, y_train, y_test) :
    #Training gaussian model
```

```
    gnb = GaussianNB()
    gnb.fit(X_train, y_train)
    #Getting predictions
    y_pred = gnb.predict(X_test)
    return accuracy_score(y_test, y_pred)

def decision_tree_classification(X_train, X_test, y_train, y_test)
:
    #Training decision tree
    dtc = tree.DecisionTreeClassifier(
        criterion="entropy",
        max_depth=4,
        max_features=2,
        max_leaf_nodes=None,
        min_samples_leaf=1,
        min_samples_split=2,
        min_weight_fraction_leaf=0.0,
        random_state=None,
        splitter="best",
    )
    dtc.fit(X_train, y_train)
    #Getting predictions
    y_pred = dtc.predict(X_test)
    return accuracy_score(y_test, y_pred)

n_splits=10
#K-Fold Cross Validation
kf = KFold(n_splits=n_splits)
avg_score = [0, 0]
for trainIndex, testIndex in kf.split(df) :
    avg_score[0] += naive_bayes_classification(X_train, X_test,
y_train, y_test)
    avg_score[1] += decision_tree_classification(X_train, X_test,
y_train, y_test)
print(f"Naive Bayes Avg. Accuracy = {avg_score[0]*100/10} %")
print(f"Decision Tree Avg. Accuracy = {avg_score[1]*100/10} %")


#Bagging Ensemble model
estimators = [("naiveBayes", GaussianNB()), ("decisionTree",
tree.DecisionTreeClassifier())]
baggingEnsemble = VotingClassifier(estimators)
```

```
baggingEnsemble.fit(X_train, y_train)
y_pred = baggingEnsemble.predict(X_test)
baggingAccuracy = accuracy_score(y_test, y_pred)
print(f"Bagging Accuracy: {baggingAccuracy*100} %")


#Adaboost Ensemble model
adaboostEnsemble = AdaBoostClassifier(n_estimators=3)
adaboostEnsemble.fit(X_train, y_train)
y_pred = adaboostEnsemble.predict(X_test)
adaboostAccuracy = accuracy_score(y_test, y_pred)
print(f"Adaboost Accuracy: {adaboostAccuracy*100} %")


#Plotting
plt.bar([1,2,3,4],
[avg_score[0]/10,avg_score[1]/10,baggingAccuracy,adaboostAccuracy]
,
color=["red","green","pink","blue"])
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.xticks([1,2,3,4],["Naive Bayes", "Decision Tree", "Bagging",
"AdaBoost"])
plt.show()
```

OUTPUT: