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:

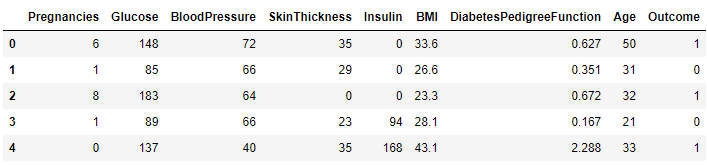
* Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
* Decision trees 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.

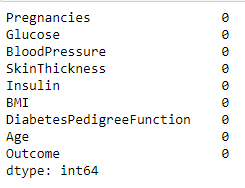
**DATASET 1: Diabetes**

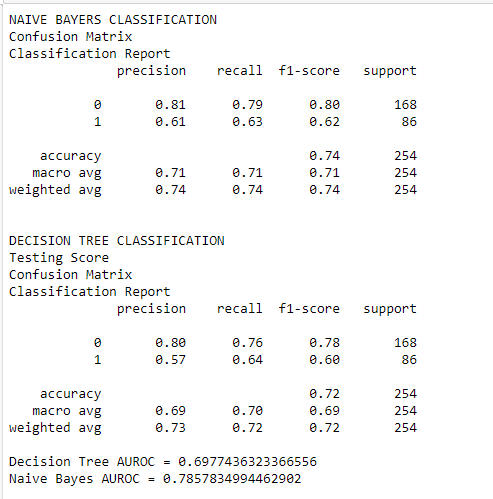
CODE:

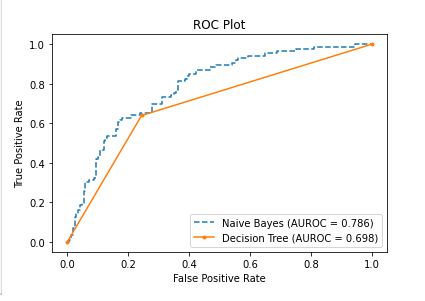
| import pandas as pd df = pd.read\_csv('diabetes.csv')  df.head()  df.isnull().sum()  from sklearn.model\_selection import train\_test\_split  X=df.drop(columns=['Outcome']) y=df['Outcome'] X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)  print("NAIVE 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:







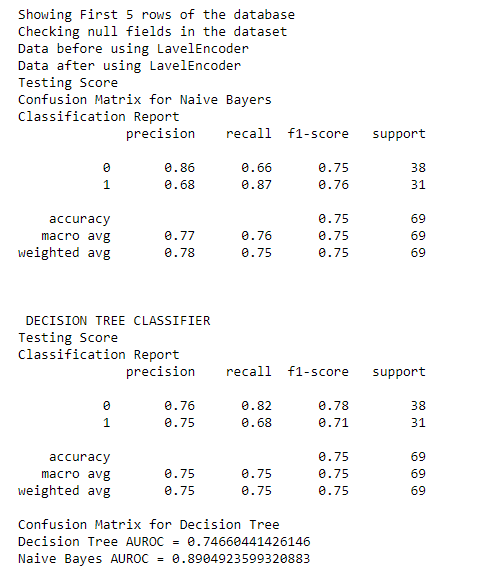
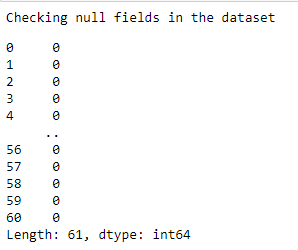
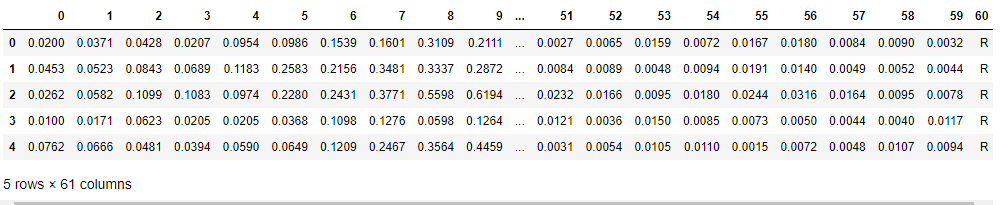


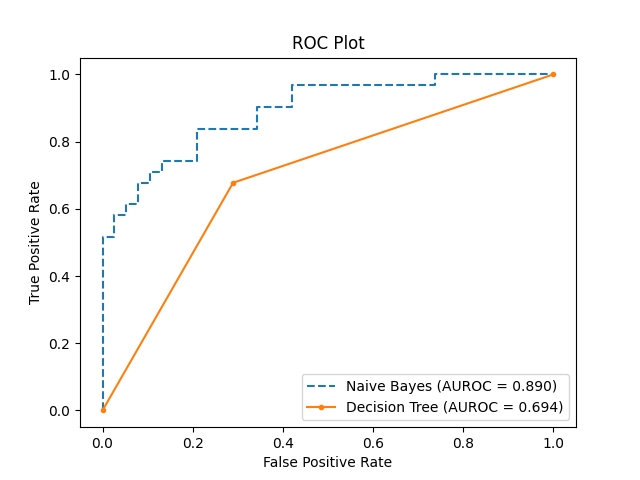
**DATASET 2: Sonar**

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:



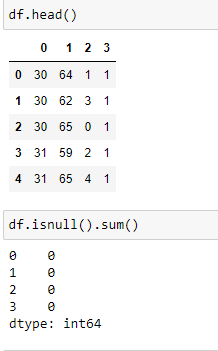


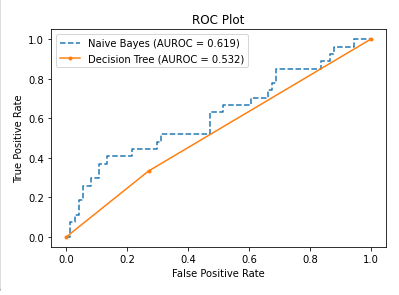
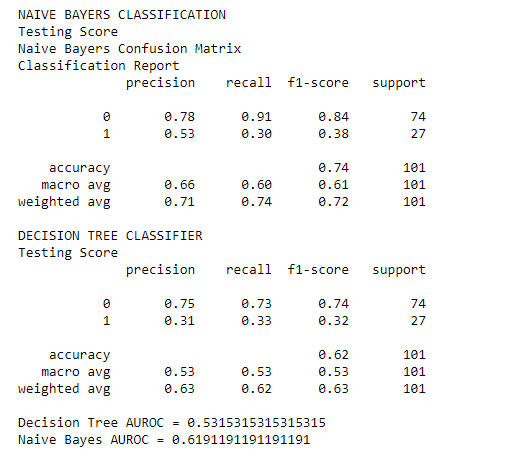
**DATASET 3: Haberman**

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:



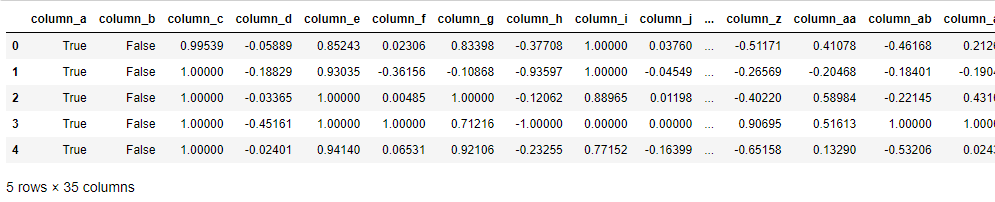


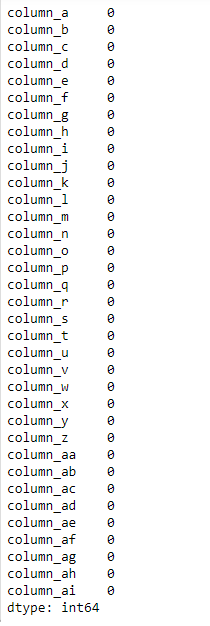
**DATASET 4: Ionosphere**

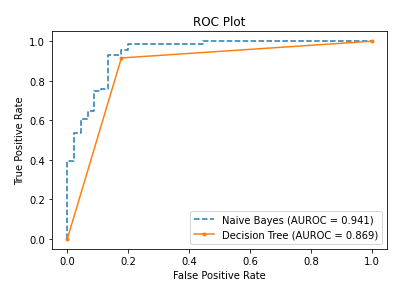
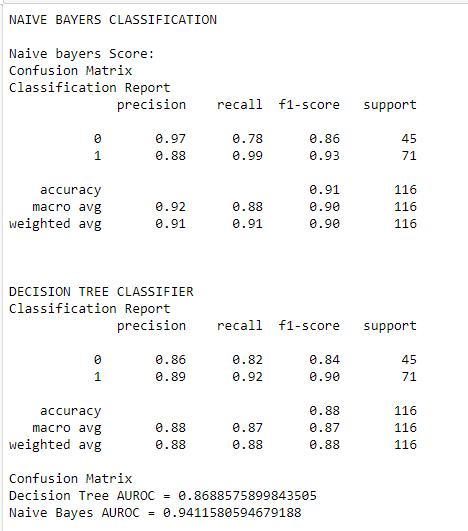
CODE:

| 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:





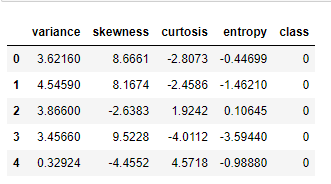


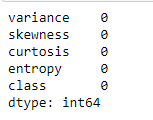
**DATASET 5: BankNote Authentication**

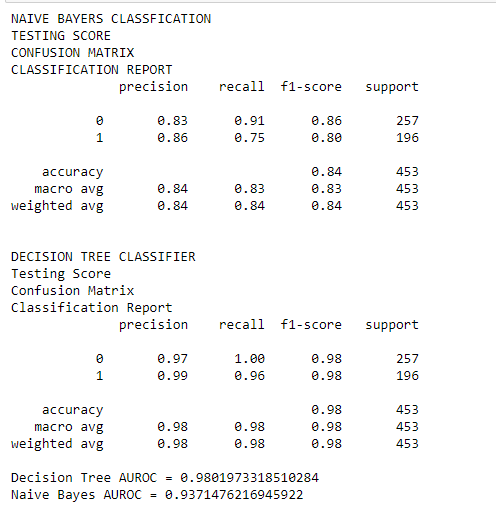
CODE:

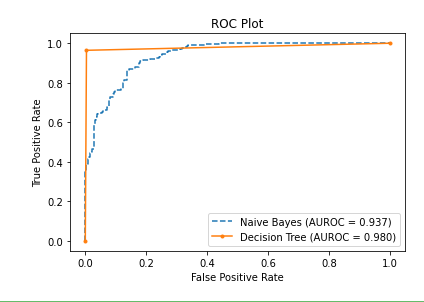
| import pandas as pd  df = pd.read\_csv('BankNoteAuthentication.csv')  df.head()  df.isnull().sum()  from sklearn.model\_selection import train\_test\_split  X=df.drop(columns=['class']) y=df['class'] X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)  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:







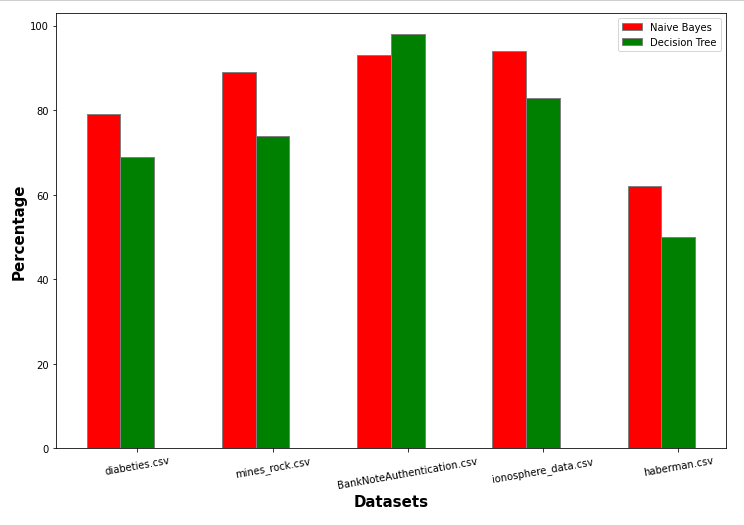


**Comparison:**

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:



**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:  
