D D M M Y Y Y Y

PROGRAM - 7

AdaBoost Classifier

Definition:

AdaBoost or Adaptive Boosting is one of ensemble boosting classifier. It combines multiple classifiers to increase the accuracy of classifiers.

Working:

→ The algorithm is trained on the training data using weights. Initially all the weights are set to the same value

→ A weak learner is trained on the training data.

→ The algorithm calculates the error rate of the weak learner

→ The weights of the incorrectly predicted instances are increased.

→ Another weak learner is trained on the training data using the updated weights

→ The process is repeated until the desired number of weak learners has been created

→ The weak learners are then combined to create the final strong learner

→ To make a prediction the strong learner combines the predictions of the weak learners using a weighted majority vote, where the weights are determined by the accuracy of each weak learner

D D M M Y Y Y Y

Advantages:

- It is fast, since it only requires the weak learners to perform simple tasks
- It is versatile, because it can be used with a variety of weak learners.
- It can handle high dimensional data
- It can handle noise and outliers in the data
- It is relatively easy to interpret.

Disadvantages:

- It can be slow to predict, because it requires the combined prediction of all the weak learners.
- It is sensitive to the scale of the data
- It is sensitive to the choice of hyperparameter
- It requires careful selection of the weak learners
- It is prone to overfitting

Program:

```
#Import libraries and data

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

data = pd.read_csv("Bank-marketing.csv")
data.head()
```

D D M M Y Y Y Y

```
# Choose the x and y
x = data[['age', 'marital', 'ever_defaulted', 'housing_loan', 'Personal_loan']]
y = data[['y']]

# Train, test validation split
X_train, X_test, Y_train, y_test = train_test_split
(x, y, test_size = 0.20, random_state = 42)
test_df = pd.concat([x_test, y_test], axis = 1)

# Initialize and train your model
abc = AdaBoostClassifier(random_state = 0)
abc_model = abc.fit(x_train, y_train)

# Predict the test data based on trained model
abc_pred = abc_model.predict(x_test)
accuracy_score(y_test, abc_pred)
```

DSC 8

D.S.C.

D D M M Y Y Y Y

PROGRAM - II

Random Forest Classifier

Definition:

Random Forests are a type of ensemble learning method for classification, regression, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Working:

→ Random subsets of the training data are selected

→ For each subset of the training data, a decision tree is constructed.

→ Among all the decision trees constructed, choose the decision tree, that performs the best.

→ The process is repeated until the desired number of decision trees has been built.

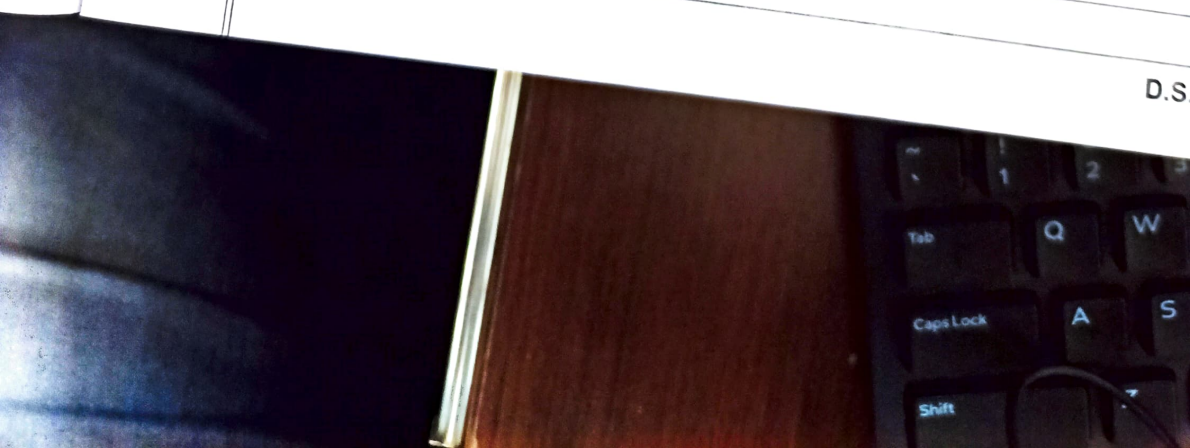
→ To classify a new object based on attributes, each tree gives a classification and we chose the classification with most votes.

Advantages:

→ They are accurate and robust because they average the predictions of many decision trees.

→ They can handle high dimensional data, because they do not require the features to be linearly co-related.

DSC (11)



D D M M Y Y Y Y

→ They can handle missing values, trees are trained on the existent available data.
→ They can handle unbalanced datasets.
→ They are easy to use, because they require little pre-processing of the data and have a few hyper-parameters to tune.

Disadvantages:
→ They can be slow to predict, because they require the prediction of all the trees in the forest.
→ They are difficult to interpret.
→ They may be less accurate than some other models, if the data is not sufficiently random.
→ They are prone to overfitting, if the number of trees in the forest is too large.
→ They are sensitive to the scale of the data

Program:

```
#Import libraries and data

import pandas as pd
from sklearn.model_selection import train_test_spl
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

data=pd.read_csv("Bank_Marketing.csv")
data.head()
```

D D M M Y Y Y Y

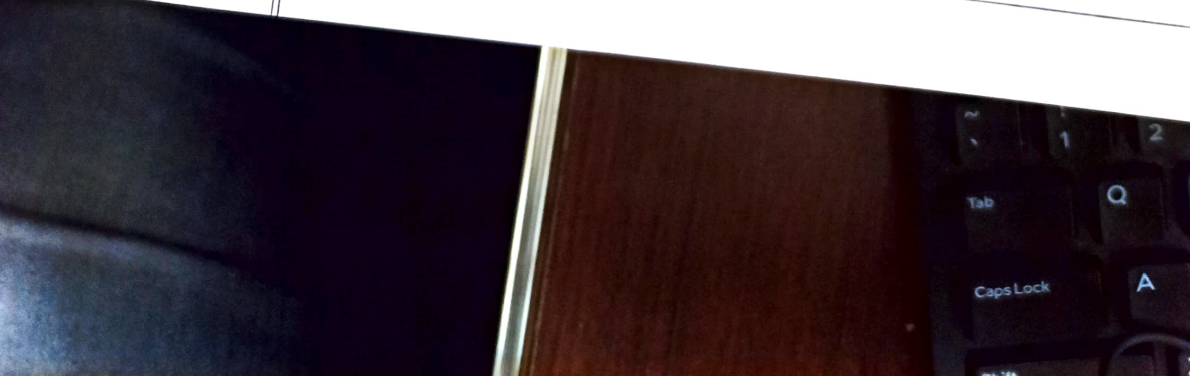
```
# Chose the dependent and independent variables
x = data[['age', 'marital', 'ever_defaulted', 'housing_
loan', 'Personal_loan']]
y = data[['y']]

# Train, test validation Split
X_train, X_test, Y_train, Y_test = train_test_split
(x, y, test_size=0.20, random_state=42)
test_df = pd.concat([x_test, y_test, axis=1)

# Initialize and train your model
clf = RandomForestClassifier(random_state=42)
clf.fit(x_train, y_train)

# Predict the test based on the trained model
rf_pred = clf.predict(x_test)
accuracy_score(y_test, rf_pred)
```


E.



PROGRAM - III

Gradient Boosting Classifies

Definition:

Gradient Boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

Working:

→ Initialize the model with a constant prediction. This can be the mean of the target values in the training set or a constant value.

→ Train a weak learner to predict the residual error of the current model. The residual error is the difference between the actual target value and current model's prediction.

→ Add the weak learner to the model.

→ Update the model by adding the weak learner multiplied by the learning rate alpha.

→ Repeat steps 2-4 until the desired number of weak learners have been trained.

→ To make a prediction, the model combines the predictions of individual weak learners using a weighted sum.

Advantages:

→ It can handle a variety of loss functions,

because it can optimize an arbitary differentiable loss function

→ It can provide a good prediction accuracy, because it can learn complex non-linear relationships in data.

→ It is resistant to overfitting

→ It can handle missing values, because the weak learners can be trained on the available data and the missing values are ignored.

→ It is parallelizable, weak learners can be trained in parallel.

Disadvantages:

→ It can be slow to train, because it requires training of many weak learners.

→ It is sensitive to the scale of the data, weak learners may be sensitive to the scale of features

→ It is difficult to interpret, as model is complex ensemble of weak learners.

→ It is sensitive to noisy data and outliers, as it gives more weight to misclassified instances.

→ It requires careful tuning of hyperparameters

Program:

#Import libraries and data

import pandas as pd

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

data= pd.read_csv("Bank_marketing.csv")
data.head()

#Choose dependent and independent variables
x= data[['age','marital','ever_defaulted','housing_loan','Personal_loan']]
y=data[['y']]

#Train test validation split
x_train, x_test, y_train, y_test= train_test_split(x,y, test_size=0.20, random_state=42)
test_df = pd.concat([x_test, y_test], axis=1)

#Initialize and train your model
gbcl=GradientBoostingClassifier(random_state=42)
gbcl.fit(x_train, y_train)

#Predict the test data based on the trained model
gbcl_pred= gbcl.predict(x_test)
accuracy_score(y_test, gbcl_pred)
```

PROGRAM - IV

Extreme Gradient Boosting Classifier

Definition:

XGBoost (Extreme gradient Boosting) is an open-source implementation of gradient boosting that is designed to very efficient flexible and portable. It provides a high performance implementation of gradient-boosting that is optimised for speed and memory efficiency

Working:

→ Initialize the model with a constant prediction. This can be the mean of the target values in the training set.

→ Train a weak learner to predict the residual errors (difference of actual target value and current model prediction).

→ Add the weak learner to the model.

→ Update the model by adding the weak learner multiplied by the learning rate alpha.

→ To make a prediction, the model combines the predictions of the individual weak learners using a weighted sum

Advantages:

→ It is faster and more memory-efficient than traditional gradient boosting

→ It supports parallelization and distributed

computation, making it suitable for large scale problems.

→ It has large number of hyperparameters that can be tuned to optimize the model's performance.

→ It has built-in support for missing values.

→ It has built in support for regularization, which can help prevent over-fitting.

Disadvantages:

→ It is difficult to interpret, because the model is complex ensemble of weak learners.

→ It may not be suitable for very large dataset, as it requires more memory and computational resources.

→ It is sensitive to noisy data and outliers.

→ It is sensitive to the scale of the data, as weak learners may be sensitive to the scale of the features.

→ It requires careful tuning of hyperparameters, such as learning rate and number of weak learners.

Program:

```
#Import libraries and data

import pandas as pd
from sklearn.model_selection import train_test_
split
```

DD MM YYYY

```
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier

data = pd.read_csv("Bank_marketing.csv")
data.head()

#choose the dependent and independent variables
X = data[['age', 'marital', 'ever_defaulted',
'housing_loan', 'Personal_loan']]
y = data['y']

#Train test validation split
X_train, X_test, y_train, y_test = train_test_split
(X, y, test_size=0.20, random_state=42)
test_df = pd.concat([X_test, y_test], axis=1)

#Initialize and train your model
model = XGBClassifier()
model.fit(X_train, y_train)

#Predict the test data based on trained model
xgbc_pred = model.predict(X_test)
accuracy_score(y_test, xgbc_pred)
```

D D M M Y Y Y Y

PROGRAM - V

Artificial Neural Network (ANN) Classifier

Definition:

An artificial neural network (ANN) is a machine learning model inspired by the structure and function of the human brain. It is composed of layers of interconnected "neurons" which process and transmit information.

Working:

→ The ANN is provided with a dataset which consists of input data and corresponding labels.
→ The input data is processed through the input layer, which passes it on to the hidden layers.
→ The hidden layers use weights and biases to transform the input data and pass it on to the next layer.
→ The output layer processes the data from the hidden layers and produces a prediction.
→ The error between the predicted output and the true labels is calculated.
→ The process is repeated until the model reaches the desired level of accuracy.

Advantages:

→ They can learn to recognize complex patterns and relationships in data.
→ They can process noisy datasets with high

dimensionality.
→ They can handle missing values in the data
→ They can be trained using a variety of optimization algorithms.
→ They can adapt to changing circumstances and learn from new data.

Disadvantages:
→ They can be difficult to design and train because they require and need large amount of data and computational resources.
→ They can be prone to overfitting, if the model is very complex.
→ They can be difficult to interpret, as internal workings of the model are not transparent.
→ They may require large amount of memory and computational resources.
→ They can be sensitive to the choice of hyper-parameters.

Program:

```
#Import libraries and data

import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.metrics import accuracy_score
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

data = pd.read_csv("Bank_marketing.csv")
data.head()

#Scale the numeric columns
col_to_scale = ['age']
data[col_to_scale] = scaler.fit_transform(data[col_to_scale])
data.head()

#Choose the dependent and independent variable
X = data.drop('y', axis=1)
Y = data['y']

#Train test validation split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state=42)
test_df = pd.concat([X_test, y_test], axis=1)

#Create the network

model = Keras.Sequential([
Keras.layers.Dense(3, input_shape=(5,), activation='relu'),
Keras.layers.Dense(2, input_shape=(3,), activation='relu'),
Keras.layers.Dense(1, activation='sigmoid'),])
```

DSC (15)

```
#Compile the network

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

#Train the model
model.fit(x_train, y_train, epochs=5)

#Evaluate the model
model.evaluate(x_test, y_test)

#Predict based on ANN model
yp = model.predict(x_test)

y_pred = []
for element in yp:
    if element > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)

#Check the accuracy of the model
accuracy_score(y_test, y_pred)
```