Dikshita Kambri

118A2044

BE EXTC A2

EXPERIMENT 7

Implementation of Random forest

EXPERIMENT 7

RANDOM FOREST

AIM: Implement of random forest algorithm

APPARATUS: python

THEORY:

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.
random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.
One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems

Random forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, there's no need to combine a decision tree with a bagging classifier because you can easily use the classifier-class of random forest. With random forest, you can also deal with regression tasks by using the algorithm's regressor.

Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

The hyperparameters in random forest are either used to increase the predictive power of the model or to make the model faster. Let's look at the hyperparameters of sklearns built-in random forest function.

1. Increasing the predictive power
Firstly, there is the n_estimators hyperparameter, which is just the number of trees the algorithm builds before taking the maximum voting or taking the averages of predictions. In general, a higher number of trees increases the performance and makes the predictions more stable, but it also slows down the computation.
Another important hyperparameter is max_features, which is the maximum number of features random forest considers to split a node. Sklearn provides several options, all described in the documentation.
The last important hyperparameter is min_sample_leaf. This determines the minimum number of leafs required to split an internal node.

2. Increasing the model's speed

The n_jobs hyperparameter tells the engine how many processors it is allowed to use. If it has a value of one, it can only use one processor. A value of "-1" means that there is no limit.
The random_state hyperparameter makes the model's output replicable. The model will always produce the same results when it has a definite value of random_state and if it has been given the same hyperparameters and the same training data.

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn import tree
from google.colab import files
uploaded = files.upload()
dataset = pd.read_csv('Churn_Modelling1.csv')
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, 7]
#print (X)
#print(y)
print(X.shape)
print(y.shape)
dataset.shape
print(X)
print(y)
print(X['Geography'])
#Convert the column into categorical columns
states=pd.get_dummies(X['Geography'], drop_first= True)
print(states)
# Drop the state coulmn
X = X.iloc[:, 1:7]
print(X)
# concat the dummy variables
X=pd.concat([X,states],axis=1)
print(X)
#Convert the column into categorical columns
states=pd.get_dummies(X['Gender'], drop_first= True)
print(states)
# Drop the state coulmn
X = X.iloc[:, 1:8]
print(X)
# concat the dummy variables
X=pd.concat([X,states],axis=1)
```

```
print(X)
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_si
ze = 0.2, random_state = 2)
print(X_train)
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(criterion='entropy', n_estimators
=40, random_state=2, n_jobs=2)
forest.fit(X_train, y_train)
y_pred = forest.predict(X_test)
print(y_pred)
print('percentage accuracy',100*accuracy_score(y_test, y_pred))
```

OUTPUT:

```
[ ] from sklearn.ensemble import RandomForestClassifier
    forest = RandomForestClassifier(criterion='entropy', n_estimators=40, random_state=2, n_jobs=2)
    forest.fit(X_train, y_train)
    y_pred = forest.predict(X_test)
    print(y_pred)

    [0 0 0 ... 0 0 0]

[ ] print('percentage accuracy',100*accuracy_score(y_test, y_pred))

    percentage accuracy 82.8
```

CONCLUSION:

We trained models using random forest algorithm with the help of churn modeling dataset with accuracy of 82.8%. We also observed that, random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.