EX.NO: 6 BUILD DECISION TREES AND RANDOM FORESTS

Aim:

The aim of building decision trees and random forests is to create models that can be used to predict a target variable based on a set of input features. Decision trees and randomforests are both popular machine learning algorithms for building predictive models.

Algorithm:

Decision Trees.

1. Select the feature that best splits the data: The first step is to select the feature that best separates the data into groups with different target values.
2. Recursively split the data: For each group created in step 1, repeat the process of selecting the best feature to split the data until a stopping criterion is met. The stopping criterion may be a maximum tree depth, a minimum number of samples in a leaf node, or another condition.
3. Assign a prediction value to each leaf node: Once the tree is built, assign a prediction value to each leaf node. This value may be the mean or median target value of the samples in the leaf node.

Random Forest

1. Randomly select a subset of features: Before building each decision tree, randomly select a subset of features to consider for splitting the data.
2. Build multiple decision trees: Build multiple decision trees using the process described above, each with a different subset of features.
3. Aggregate the predictions: When making predictions on new data, aggregate the predictions from all decision trees to obtain a final prediction value. This can be done by taking the average or majority vote of the predictions.

Program:

```
import pandas as pd
fromsklearn.tree import DecisionTreeRegressor
fromsklearn.ensemble import RandomForestRegressor
fromsklearn.model_selection import train_test_split
fromsklearn.metrics import mean_squared_error

# Load data
data = pd.read_csv('data.csv')

# Split data into training and test setsX
= data.drop(['target'], axis=1)
y = data['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build decision tree
dt = DecisionTreeRegressor()
dt.fit(X_train, y_train)

# Predict on test set
y_pred = dt.predict(X_test)

# Evaluate performance
mse = mean_squared_error(y_test, y_pred)
print(f"Decision Tree Mean Squared Error: {mse:.4f}")

# Build random forest
rf = RandomForestRegressor()
rf.fit(X_train, y_train)

# Predict on test set y_pred
= rf.predict(X_test)

# Evaluate performance
mse = mean_squared_error(y_test, y_pred)
print(f"Random Forest Mean Squared Error: {mse:.4f}")
```

Output:
Decision Tree Classifier Accuracy: 1.0
Random Forest Classifier Accuracy: 1.0

Result:

Thus the program for decision trees is executed successfully and output is verified.

EX.NO: 7 BUILD SVM MODELS

Aim:

The aim of this Python code is to demonstrate how to use the scikit-learn library to train support vector machine (SVM) models for classification tasks.

Algorithm:

1. Load a dataset using the pandas library
2. Split the dataset into training and testing sets **train_test_split** usingscikit-learn
3. Train three SVM models with different **SVC** kernels (linear, polynomial, and RBF) usingfunction from scikit-learn
4. Predict the test set labels using the trained model
5. Evaluate the accuracy of the models using **accuracy_score** thelearn
6. Print the accuracy of each model

Program:

```
import pandas as pd
fromsklearn.model_selection import train_test_split
fromsklearn.svm import SVC
fromsklearn.metrics import accuracy_score

# Load the dataset
data = pd.read_csv('data.csv')

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data.drop('target', axis=1), data['target'],
test_size=0.3, random_state=42)

# Train an SVM model with a linear kernel
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)

# Predict the test set labels
y_pred = svm_linear.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Linear SVM accuracy: {accuracy:.2f}')

# Train an SVM model with a polynomial kernel
svm_poly = SVC(kernel='poly', degree=3)
svm_poly.fit(X_train, y_train)
```

```
# Predict the test set labels
y_pred = svm_poly.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Polynomial SVM accuracy: {accuracy:.2f}')



# Train an SVM model with an RBF kernel
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train, y_train)

# Predict the test set labels y_pred
= svm_rbf.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'RBF SVM accuracy: {accuracy:.2f}')
```

Output:

```
Accuracy: 0.9777777777777777
```

Result:

Thus the program for Build SVM Model has been executed successfully and output is verified.