St. Francis Institute of Technology, Mumbai-400 103
Department Of Information Technology

A.Y. 2025-2026
Class: BE-ITA/B, Semester: VII
Subject: Data Science Lab

Experiment – 8

1. **Aim:** To implement Supervised Learning algorithm - Random Forest.
2. **Objectives:** Students should be familiarize with Learning Architectures and Frameworks
3. **Prerequisite:** Python basics

4. **Pre-Experiment Exercise:**
Theory:
Random Forest Algorithm

Decision trees involve the greedy selection of the best split point from the dataset at each step.

This algorithm makes decision trees susceptible to high variance if they are not pruned. This high variance can be harnessed and reduced by creating multiple trees with different samples of the training dataset (different views of the problem) and combining their predictions. This approach is called bootstrap aggregation or bagging for short.

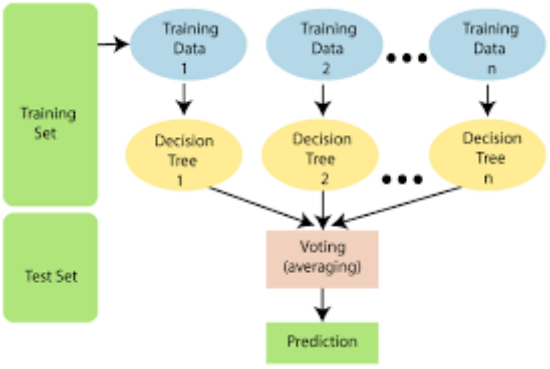
A limitation of bagging is that the same greedy algorithm is used to create each tree, meaning that it is likely that the same or very similar split points will be chosen in each tree making the different trees very similar (trees will be correlated). This, in turn, makes their predictions similar, mitigating the variance originally sought.

We can force the decision trees to be different by limiting the features (rows) that the greedy algorithm can evaluate at each split point when creating the tree. This is called the Random Forest algorithm.

Like bagging, multiple samples of the training dataset are taken and a different tree trained on each. The difference is that at each point a split is made in the data and added to the tree, only a fixed subset of attributes can be considered.

For classification problems, the type of problems we will look at in this tutorial, the number of attributes to be considered for the split is limited to the square root of the number of input features.

num_features_for_split = sqrt(total_input_features)
The result of this one small change are trees that are more different from each other (uncorrelated) resulting predictions that are more diverse and a combined prediction that often has better performance that single tree or bagging alone.



6. Laboratory Exercise

Procedure

i. Use google colab for programming.
ii. Import required packages.
iii. Demonstrate random forest classifier for any given dataset.
iv. Add relevant comments in your programs and execute the code. Test it for various cases.

Post-Experiments Exercise:

A. Extended Theory:

a. Write real life applications of Random Forest Classifier.

B. Conclusion:

1. Write what was performed in the program (s) .
2. What is the significance of program and what Objective is achieved?

C. References:

[1] https://machinelearningmastery.com/implement-random-forest-scratch-python/.
[2] https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/

Load a suitable dataset for classification from the available files.

```
import pandas as pd

df = pd.read_csv('/content/sample_data/mnist_train_small.csv')
display(df.head())
display(df.info())
```

	6	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	...	0.581	0.582	0.583	0.584	0.585	0.586	0.587	0.588	0.589	0.590
0	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19999 entries, 0 to 19998
Columns: 785 entries, 6 to 0.590
dtypes: int64(785)
memory usage: 119.8 MB
None
```

Split the dataset into training and testing sets.

```
from sklearn.model_selection import train_test_split

X = df.iloc[:, 1:]
y = df.iloc[:, 0]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (15999, 784)
Shape of X_test: (4000, 784)
Shape of y_train: (15999,)
Shape of y_test: (4000,)
```

Train both a Random Forest classifier and a Decision Tree classifier on the training data.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

rf_clf = RandomForestClassifier(random_state=42)
dt_clf = DecisionTreeClassifier(random_state=42)

rf_clf.fit(X_train, y_train)
dt_clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

Evaluate the accuracy of both models on the testing data.

```
from sklearn.metrics import accuracy_score

y_pred_rf = rf_clf.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)

y_pred_dt = dt_clf.predict(X_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)

print(f"Random Forest Classifier Accuracy: {accuracy_rf}")
print(f"Decision Tree Classifier Accuracy: {accuracy_dt}")
```

```
Random Forest Classifier Accuracy: 0.954
Decision Tree Classifier Accuracy: 0.83
```

Visualize the accuracy of both models.

```
import numpy as np

labels = ['Random Forest', 'Decision Tree']
accuracy_scores = [accuracy_rf, accuracy_dt]
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars
fig, ax = plt.subplots()
rects = ax.bar(x, accuracy_scores, width)
# Add some text for labels, title and axes ticks
ax.set_ylabel('Accuracy')
ax.set_title('Accuracy of Random Forest vs Decision Tree')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.set_ylim([0, 1]) # Accuracy is between 0 and 1

def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.3f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects)
fig.tight_layout()
plt.show()
```