Subject: Machine Learning – I (DJ19DSC402)

AY: 2021-22

Experiment 6

(Random Forest)

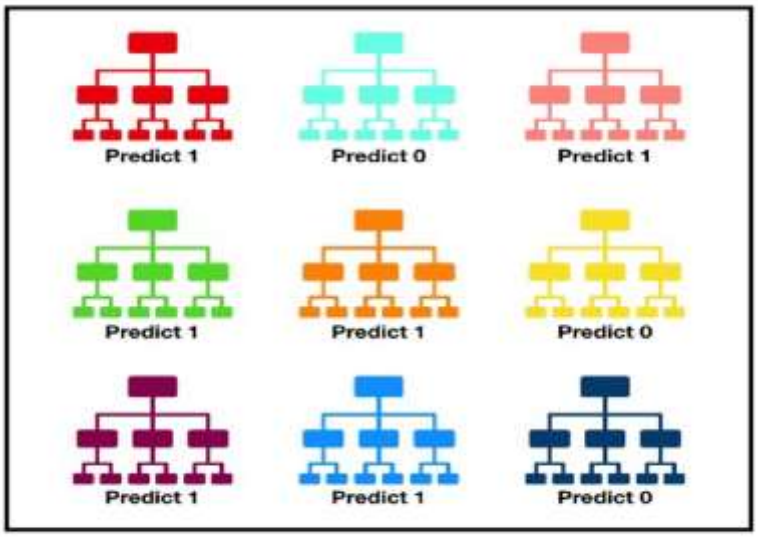
Name: Dev Patel **SAP ID: 60009200016**

Batch: K/K1 **Date: 02/05/2022**

Aim: Implement Random Forest algorithm on given datasets and compare the results with Decision Tree classifiers for the same datasets.

Theory:

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction (see figure below).



Tally: Six 1s and Three 0s
Prediction: 1

A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. **The reason for this wonderful effect is that the trees protect each other from their individual errors** (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So, the prerequisites for random forest to perform well are:

1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.
2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:

Dataset 1: SonarAllDataset.csv

Dataset 2: IRIS.csv

Dataset 3: BehaviouralRskFactorSurvillanceSystem.csv (The objective of the BRFSS is to collect uniform, state-specific data on preventive health practices and risk behaviors that are linked to chronic diseases, injuries, and preventable infectious diseases in the adult population. Factors assessed by the BRFSS include tobacco use, health care coverage, HIV/AIDS knowledge or prevention, physical activity, and fruit and vegetable consumption. Data are collected from a random sample of adults (one per household) through a telephone survey. The Behavioral Risk Factor Surveillance System (BRFSS) is the nation's premier system of health-related telephone surveys that collect state data about U.S. residents regarding their health-related risk behaviors, chronic health conditions, and use of preventive services. Established in 1984 with 15 states, BRFSS now collects data in all 50 states as well as the District of Columbia and three U.S. territories. BRFSS completes more than 400,000 adult interviews each year, making it the largest continuously conducted health survey system in the world.)

1. Perform Random Forest from scratch on dataset 1.

2. Compare the results of decision tree and random forest classifier for dataset 2 and 3.

3. Compare the results of random forest with and without selecting important features only for building the classifier on dataset 2 and 3.

Results:

The accuracy from 5 readings for each dataset and its average are given below:

For Dataset 1: Sonar Dataset

	Using Decision Tree	**Using Random Forest**	**Using Random Forest after Feature Importance**
Reading 1	0.67	0.76	0.78
Reading 2	0.71	0.79	0.86
Reading 3	0.64	0.83	0.88
Reading 4	0.79	0.90	0.91
Reading 5	0.71	0.76	0.81
Average	0.704	0.808	0.848

For Dataset 2: Iris Dataset

	Using Decision Tree	**Using Random Forest**	**Using Random Forest after Feature Importance**
Reading 1	0.97	1.00	1.00
Reading 2	0.97	1.00	1.00
Reading 3	1.00	1.00	1.00
Reading 4	0.93	0.97	1.00
Reading 5	1.00	1.00	1.00
Average	0.974	0.994	1.00

We can observe that the performances of the models improve. Random Forest, which is an ensemble of several decision trees, gives better results than individual decision trees. Also, Random Forest after removing columns based on Feature Importance provide even better results than when we applied Random Forest directly.

Code and Output:

In [1]:

```
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import sklearn
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score , confusion_matrix
from random import seed
from random import randrange
from csv import reader
from math import sqrt
```

1. Perform Random Forest from scratch on SONAR dataset.

From Scratch Code:

In [2]:

```
def load_csv(filename):
 dataset = list()
 with open(filename, 'r') as file:
  csv_reader = reader(file)
  for row in csv_reader:
   if not row:
    continue
   dataset.append(row)
 return dataset

def str_column_to_float(dataset, column):
 for row in dataset:
  row[column] = float(row[column].strip())

def str_column_to_int(dataset, column):
 class_values = [row[column] for row in dataset]
 unique = set(class_values)
 lookup = dict()
 for i, value in enumerate(unique):
  lookup[value] = i
 for row in dataset:
  row[column] = lookup[row[column]]
 return lookup

def cross_validation_split(dataset, n_folds):
 dataset_split = list()
 dataset_copy = list(dataset)
 fold_size = int(len(dataset) / n_folds)
 for i in range(n_folds):
  fold = list()
  while len(fold) < fold_size:
   index = randrange(len(dataset_copy))
   fold.append(dataset_copy.pop(index))
  dataset_split.append(fold)
 return dataset_split

def accuracy_metric(actual, predicted):
 correct = 0
 for i in range(len(actual)):
  if actual[i] == predicted[i]:
   correct += 1
 return correct / float(len(actual)) * 100.0
```

```
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
 folds = cross_validation_split(dataset, n_folds)
 scores = list()
 for fold in folds:
  train_set = list(folds)
  train_set.remove(fold)
  train_set = sum(train_set, [])
  test_set = list()
  for row in fold:
   row_copy = list(row)
   test_set.append(row_copy)
   row_copy[-1] = None
  predicted = algorithm(train_set, test_set, *args)
  actual = [row[-1] for row in fold]
  accuracy = accuracy_metric(actual, predicted)
  scores.append(accuracy)
 return scores

def test_split(index, value, dataset):
 left, right = list(), list()
 for row in dataset:
  if row[index] < value:
   left.append(row)
  else:
   right.append(row)
 return left, right

def gini_index(groups, classes):
 n_instances = float(sum([len(group) for group in groups]))
 gini = 0.0
 for group in groups:
  size = float(len(group))
  if size == 0:
   continue
  score = 0.0
  for class_val in classes:
   p = [row[-1] for row in group].count(class_val) / size
   score += p * p
  gini += (1.0 - score) * (size / n_instances)
 return gini

def get_split(dataset, n_features):
 class_values = list(set(row[-1] for row in dataset))
 b_index, b_value, b_score, b_groups = 999, 999, 999, None
 features = list()
 while len(features) < n_features:
  index = randrange(len(dataset[0])-1)
  if index not in features:
   features.append(index)
 for index in features:
  for row in dataset:
   groups = test_split(index, row[index], dataset)
   gini = gini_index(groups, class_values)
   if gini < b_score:
    b_index, b_value, b_score, b_groups = index, row[index], gini, groups
 return {'index':b_index, 'value':b_value, 'groups':b_groups}

def to_terminal(group):
 outcomes = [row[-1] for row in group]
 return max(set(outcomes), key=outcomes.count)

def split(node, max_depth, min_size, n_features, depth):
 left, right = node['groups']
 del(node['groups'])
 if not left or not right:
  node['left'] = node['right'] = to_terminal(left + right)
  return

 if depth >= max_depth:
  node['left'], node['right'] = to_terminal(left), to_terminal(right)
  return
```

```
 # For left child
 if len(left) <= min_size:
  node['left'] = to_terminal(left)
 else:
  node['left'] = get_split(left, n_features)
  split(node['left'], max_depth, min_size, n_features, depth+1)
 # For right child
 if len(right) <= min_size:
  node['right'] = to_terminal(right)
 else:
  node['right'] = get_split(right, n_features)
  split(node['right'], max_depth, min_size, n_features, depth+1)

def build_tree(train, max_depth, min_size, n_features):
 root = get_split(train, n_features)
 split(root, max_depth, min_size, n_features, 1)
 return root

def predict(node, row):
 if row[node['index']] < node['value']:
  if isinstance(node['left'], dict):
   return predict(node['left'], row)
  else:
   return node['left']
 else:
  if isinstance(node['right'], dict):
   return predict(node['right'], row)
  else:
   return node['right']

def subsample(dataset, ratio):
 sample = list()
 n_sample = round(len(dataset) * ratio)
 while len(sample) < n_sample:
  index = randrange(len(dataset))
  sample.append(dataset[index])
 return sample

def bagging_predict(trees, row):
 predictions = [predict(tree, row) for tree in trees]
 return max(set(predictions), key=predictions.count)

def random_forest(train, test, max_depth, min_size, sample_size, n_trees, n_features):
 trees = list()
 for i in range(n_trees):
  sample = subsample(train, sample_size)
  tree = build_tree(sample, max_depth, min_size, n_features)
  trees.append(tree)
 predictions = [bagging_predict(trees, row) for row in test]
 return(predictions)
```

In [3]:

```
sonar = load_csv('/content/sample_data/sonar.all-data.csv')
```

In [4]:

```
for i in range(0, len(sonar[0])-1):
 str_column_to_float(sonar, i)
```

In [5]:

```
str_column_to_int(sonar, len(sonar[0])-1)
```

Out[5]:

```
{'M': 1, 'R': 0}
```

In [6]:

```
n_folds = 5
```

```
max_depth = 10
min_size = 1
sample_size = 1.0
n_trees=10
n_features = int(sqrt(len(sonar[0])-1))

scores = evaluate_algorithm(sonar, random_forest, n_folds, max_depth, min_size, sample_si
ze, n_trees, n_features)
print('For %d Trees:' % n_trees)
print('The scores are: %s' % scores)
print('Mean Accuracy Percentage: %f' %(sum(scores)/float(len(scores))))
```

```
For 10 Trees:
The scores are: [68.29268292682927, 85.36585365853658, 70.73170731707317, 80.487804878048
79, 85.36585365853658]
Mean Accuracy Percentage: 78.048780
```

The percentage accuracy of the Random Forest from scratch model with 10 trees is 77.56%

Using libraries on SONAR dataset

In [58]:

```
sonar = pd.read_csv('/content/sample_data/sonar.all-data.csv',header = None)
sonar.head()
```

Out[58]:

	0	1	2	3	4	5	6	7	8	9	...	51	52	53	54	55	
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0.0065	0.0159	0.0072	0.0167	
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0084	0.0089	0.0048	0.0094	0.0191	
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0232	0.0166	0.0095	0.0180	0.0244	
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0121	0.0036	0.0150	0.0085	0.0073	
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0031	0.0054	0.0105	0.0110	0.0015	

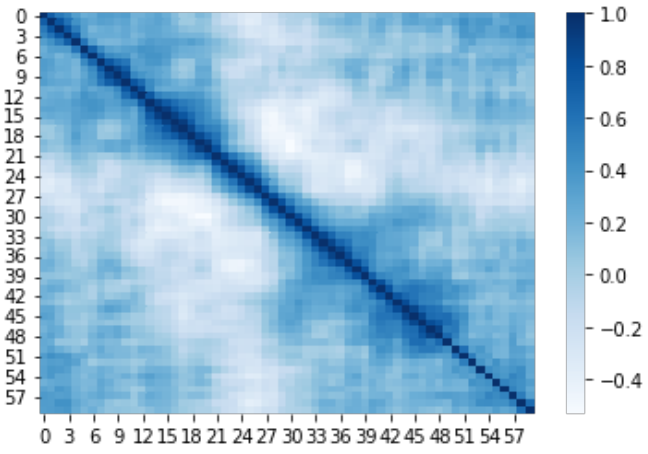
5 rows × 61 columns

In [59]:

```
sns.heatmap(sonar.corr(),cmap='Blues')
```

Out[59]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa0ec81e050>
```



Decision Tree

In [60]:

```
X = sonar.iloc[:,:-1]
y = sonar.iloc[:,-1]
from sklearn.model_selection import train_test_split as tts
X_train, X_test, y_train, y_test = tts(X,y,test_size =0.20,random_state = 1)
DT = DecisionTreeClassifier()
DT.fit(X_train , y_train)
y_pred = DT.predict(X_test)
y_trainpred = DT.predict(X_train)
```

In [61]:

```
print("Accuracy in train_set= ",accuracy_score(y_train,y_trainpred))
print("Accuracy in test_set= ",accuracy_score(y_test,y_pred))
```

```
Accuracy in train_set=  1.0
Accuracy in test_set=  0.7142857142857143
```

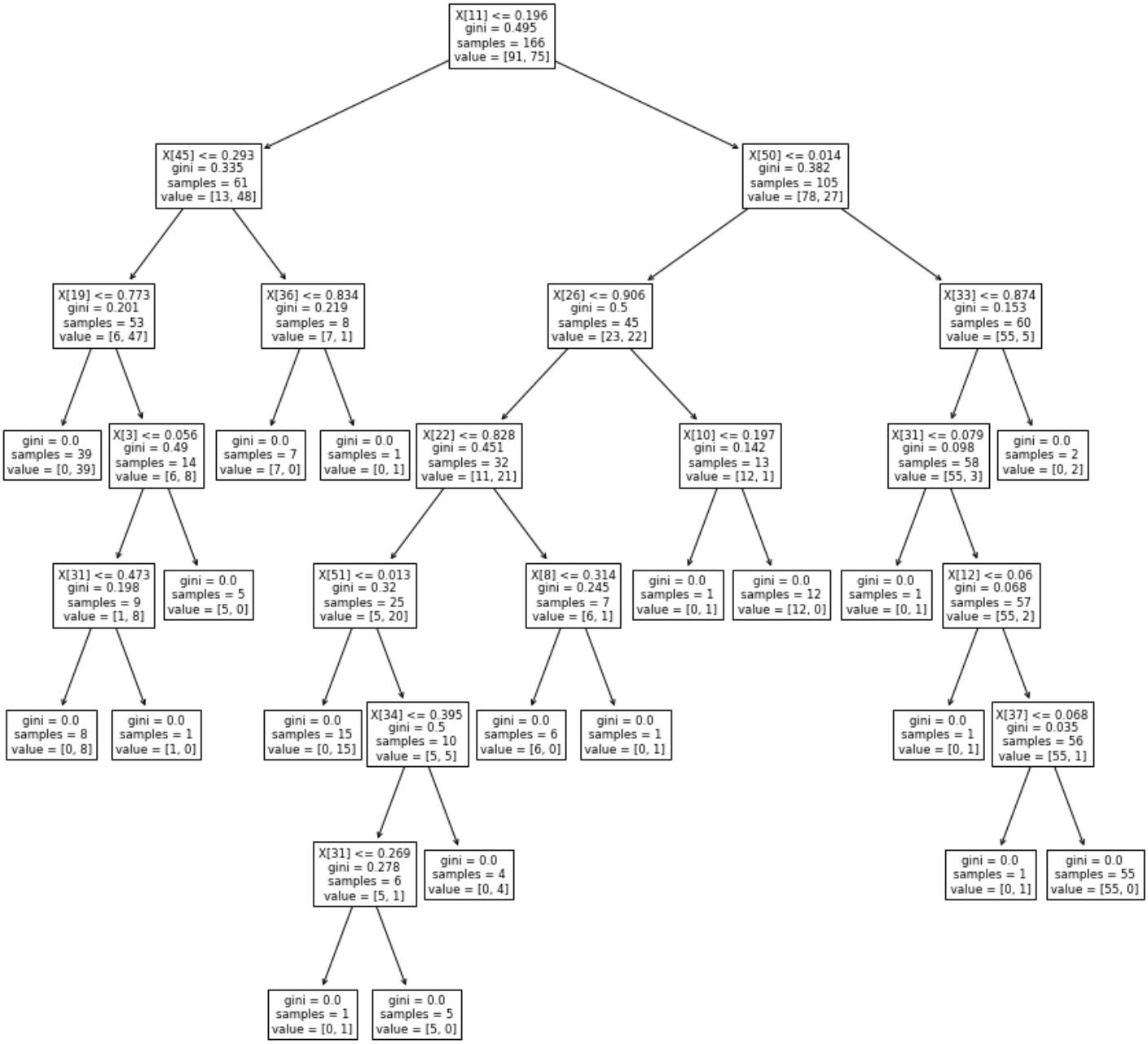
In [62]:

```
plt.figure(figsize=(15,15))
sklearn.tree.plot_tree(DT)
```

Out[62]:

```
[Text(0.4375, 0.9375, 'X[11] <= 0.196\ngini = 0.495\nsamples = 166\nvalue = [91, 75]'),
 Text(0.18181818181818182, 0.8125, 'X[45] <= 0.293\ngini = 0.335\nsamples = 61\nvalue = [
13, 48]'),
 Text(0.09090909090909091, 0.6875, 'X[19] <= 0.773\ngini = 0.201\nsamples = 53\nvalue = [
6, 47]'),
 Text(0.045454545454545456, 0.5625, 'gini = 0.0\nsamples = 39\nvalue = [0, 39]'),
 Text(0.13636363636363635, 0.5625, 'X[3] <= 0.056\ngini = 0.49\nsamples = 14\nvalue = [6,
8]'),
 Text(0.09090909090909091, 0.4375, 'X[31] <= 0.473\ngini = 0.198\nsamples = 9\nvalue = [1
, 8]'),
 Text(0.045454545454545456, 0.3125, 'gini = 0.0\nsamples = 8\nvalue = [0, 8]'),
 Text(0.13636363636363635, 0.3125, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.18181818181818182, 0.4375, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(0.2727272727272727, 0.6875, 'X[36] <= 0.834\ngini = 0.219\nsamples = 8\nvalue = [7,
1]'),
 Text(0.22727272727272727, 0.5625, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]'),
 Text(0.3181818181818182, 0.5625, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.6931818181818182, 0.8125, 'X[50] <= 0.014\ngini = 0.382\nsamples = 105\nvalue = [
78, 27]'),
 Text(0.5227272727272727, 0.6875, 'X[26] <= 0.906\ngini = 0.5\nsamples = 45\nvalue = [23,
22]'),
 Text(0.4090909090909091, 0.5625, 'X[22] <= 0.828\ngini = 0.451\nsamples = 32\nvalue = [1
1, 21]'),
 Text(0.3181818181818182, 0.4375, 'X[51] <= 0.013\ngini = 0.32\nsamples = 25\nvalue = [5,
20]'),
 Text(0.2727272727272727, 0.3125, 'gini = 0.0\nsamples = 15\nvalue = [0, 15]'),
 Text(0.36363636363636365, 0.3125, 'X[34] <= 0.395\ngini = 0.5\nsamples = 10\nvalue = [5,
5]'),
 Text(0.3181818181818182, 0.1875, 'X[31] <= 0.269\ngini = 0.278\nsamples = 6\nvalue = [5,
1]'),
 Text(0.2727272727272727, 0.0625, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.36363636363636365, 0.0625, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(0.4090909090909091, 0.1875, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.5, 0.4375, 'X[8] <= 0.314\ngini = 0.245\nsamples = 7\nvalue = [6, 1]'),
 Text(0.45454545454545453, 0.3125, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.5454545454545454, 0.3125, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.6363636363636364, 0.5625, 'X[10] <= 0.197\ngini = 0.142\nsamples = 13\nvalue = [1
2, 1]'),
 Text(0.5909090909090909, 0.4375, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.6818181818181818, 0.4375, 'gini = 0.0\nsamples = 12\nvalue = [12, 0]'),
 Text(0.8636363636363636, 0.6875, 'X[33] <= 0.874\ngini = 0.153\nsamples = 60\nvalue = [5
5, 5]'),
 Text(0.8181818181818182, 0.5625, 'X[31] <= 0.079\ngini = 0.098\nsamples = 58\nvalue = [5
5, 3]'),
 Text(0.7727272727272727, 0.4375, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.8636363636363636, 0.4375, 'X[12] <= 0.06\ngini = 0.068\nsamples = 57\nvalue = [55
, 2]'),
 Text(0.8181818181818182, 0.3125, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
```

```
 Text(0.9090909090909091, 0.3125, 'X[37] <= 0.068\ngini = 0.035\nsamples = 56\nvalue = [5
5, 1]'),
 Text(0.8636363636363636, 0.1875, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.9545454545454546, 0.1875, 'gini = 0.0\nsamples = 55\nvalue = [55, 0]'),
 Text(0.9090909090909091, 0.5625, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]')]
```



Random Forest

In [63]:

```
X_train, X_test, y_train, y_test = tts(X,y,test_size =0.20,random_state = 1)
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)
y_trainpred = clf.predict(X_train)
```

In [64]:

```
print("Accuracy in train_set= ",accuracy_score(y_train,y_trainpred))
print("Accuracy in test_set= ",accuracy_score(y_test,y_pred))
```

```
Accuracy in train_set=  1.0
Accuracy in test_set=  0.7619047619047619
```

Random Forest after Feature Importance

In [14]:

```
clf.fit(X,y)
feature_imp = pd.Series(clf.feature_importances_)
feature_imp.sort_values(ascending = False)
```

Out[14]:

```
10    0.067452
11    0.058207
9     0.039133
8     0.038080
48    0.034038
47    0.033979
46    0.028556
12    0.027759
4     0.026854
36    0.023309
44    0.023091
19    0.022571
50    0.021429
16    0.020443
45    0.020373
51    0.019479
27    0.018905
35    0.018435
15    0.017587
22    0.016906
20    0.016684
26    0.015889
0     0.015102
21    0.014625
30    0.014459
7     0.014173
5     0.014099
38    0.013400
43    0.012915
1     0.012858
17    0.012785
3     0.012750
33    0.012584
18    0.012189
57    0.011819
25    0.011318
29    0.011225
52    0.011019
13    0.010848
40    0.010362
42    0.010241
39    0.010199
6     0.010041
34    0.010036
59    0.009681
54    0.009243
2     0.008635
32    0.008533
55    0.008287
58    0.008220
14    0.008062
23    0.007410
41    0.007377
28    0.007346
31    0.007294
53    0.006908
37    0.006726
56    0.006674
24    0.005892
49    0.005505
dtype: float64
```

In [53]:

```
sonar.drop(49, axis=1, inplace = True)
sonar.drop(24, axis=1, inplace = True)
sonar.drop(56, axis=1, inplace = True)
sonar.drop(37, axis=1, inplace = True)
sonar.drop(53, axis=1, inplace = True)
sonar.drop(31, axis=1, inplace = True)
sonar.drop(28, axis=1, inplace = True)
sonar.drop(41, axis=1, inplace = True)
sonar.drop(23, axis=1, inplace = True)
sonar.drop(14, axis=1, inplace = True)
sonar.drop(58, axis=1, inplace = True)
sonar.drop(55, axis=1, inplace = True)
sonar.drop(32, axis=1, inplace = True)
sonar.drop(2, axis=1, inplace = True)
sonar.drop(54, axis=1, inplace = True)
sonar.drop(59, axis=1, inplace = True)
```

In [54]:

```
X = sonar.iloc[:,:-1]
y = sonar.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =0.20,random_state = 1
)
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train , y_train)
y_pred = clf.predict(X_test)
y_trainpred = clf.predict(X_train)
```

In [55]:

```
print("Accuracy in train_set= ",accuracy_score(y_train,y_trainpred))
print("Accuracy in test_set= ",accuracy_score(y_test,y_pred))
```

```
Accuracy in train_set=  1.0
Accuracy in test_set=  0.8095238095238095
```

Using libraries on Iris dataset

In [18]:

```
iris = pd.read_csv('/content/sample_data/Iris.csv')
iris.head()
```

Out[18]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [19]:

```
X = iris.iloc[:,:-1]
y = iris.iloc[:,-1]
```

Decision Tree

In [20]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =0.20,random_state= 1)
DT = DecisionTreeClassifier()
```

```
DT.fit(X_train , y_train)
y_pred = DT.predict(X_test)
y_trainpred = DT.predict(X_train)
```

In [21]:

```
print("Accuracy in train_set= ",accuracy_score(y_train,y_trainpred))
print("Accuracy in test_set= ",accuracy_score(y_test,y_pred))
```

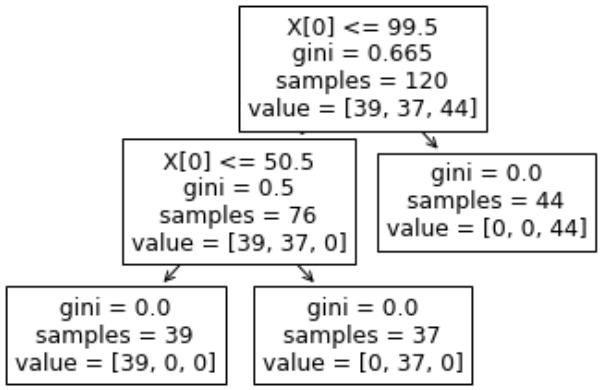
```
Accuracy in train_set=  1.0
Accuracy in test_set=  0.9666666666666667
```

In [22]:

```
sklearn.tree.plot_tree(DT)
```

Out[22]:

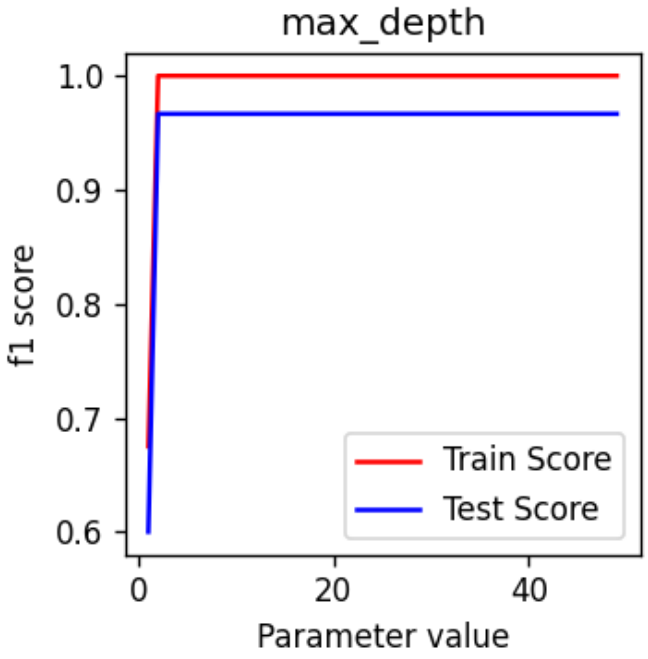
```
[Text(0.6, 0.8333333333333334, 'X[0] <= 99.5\ngini = 0.665\nsamples = 120\nvalue = [39, 3
7, 44]'),
 Text(0.4, 0.5, 'X[0] <= 50.5\ngini = 0.5\nsamples = 76\nvalue = [39, 37, 0]'),
 Text(0.2, 0.16666666666666666, 'gini = 0.0\nsamples = 39\nvalue = [39, 0, 0]'),
 Text(0.6, 0.16666666666666666, 'gini = 0.0\nsamples = 37\nvalue = [0, 37, 0]'),
 Text(0.8, 0.5, 'gini = 0.0\nsamples = 44\nvalue = [0, 0, 44]')]
```



Checking for overfitting

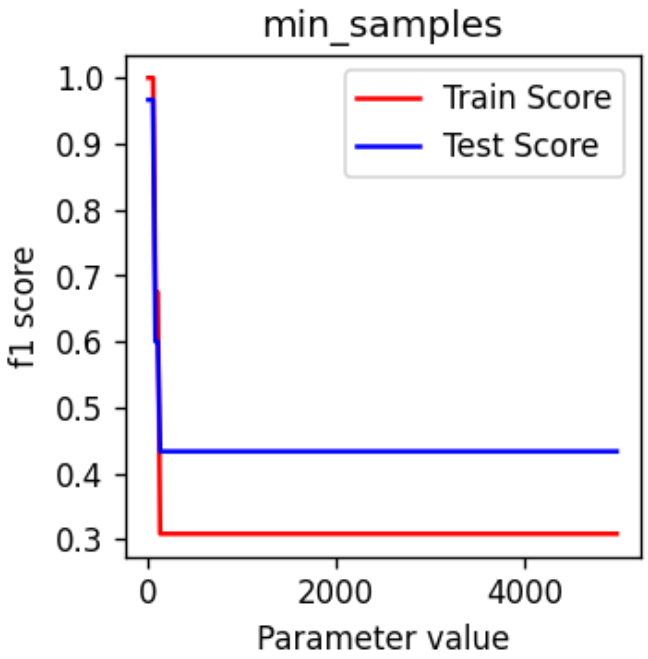
In [23]:

```
from sklearn.metrics import accuracy_score
def calc_score(model,x1,y1,x2,y2):
  model.fit(x1,y1)
  predict = model.predict(x1)
  f1 = accuracy_score(y1,predict)
  predict = model.predict(x2)
  f2 = accuracy_score(y2,predict)
  return f1,f2
def effect(train_score,test_score,x_axis, title):
  plt.figure(figsize =(3,3),dpi =120)
  plt.plot(x_axis, train_score, color='red',label = 'Train Score')
  plt.plot(x_axis, test_score, color='blue',label = 'Test Score')
  plt.title(title)
  plt.legend()
  plt.xlabel("Parameter value")
  plt.ylabel("f1 score")
  plt.show()
maxdepth = [i for i in range(1,50)]
train = []
test =[]

for i in maxdepth:
  model = DecisionTreeClassifier(class_weight = 'balanced', max_depth = i)
  f1,f2 = calc_score(model,X_train,y_train,X_test,y_test)
  train.append(f1)
  test.append(f2)
effect(train,test,range(1,50),'max_depth')
```



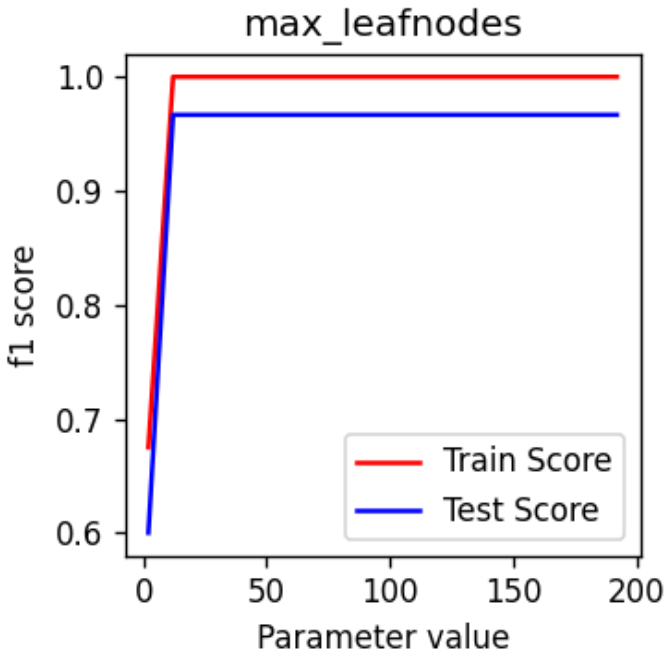
In [24]:

```
min_samples = [i for i in range(2,5000,25)]
train = []
test =[]

for i in min_samples:
  model = DecisionTreeClassifier(class_weight = 'balanced', min_samples_split = i)
  f1,f2 = calc_score(model,X_train,y_train,X_test,y_test)
  train.append(f1)
  test.append(f2)
effect(train,test,range(2,5000,25),'min_samples')
```



In [25]:

```
max_leafnodes = [i for i in range(2,200,10)]
train = []
test =[]

for i in max_leafnodes:
  model = DecisionTreeClassifier(class_weight = 'balanced', max_leaf_nodes = i)
  f1,f2 = calc_score(model,X_train,y_train,X_test,y_test)
  train.append(f1)
  test.append(f2)
effect(train,test,range(2,200,10),'max_leafnodes')
```



Random Forest

In [26]:

```
X = iris.iloc[:,:-1]
y = iris.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =0.20,random_state=1)
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)
y_trainpred = clf.predict(X_train)
```

In [27]:

```
print("Accuracy in train_set= ",accuracy_score(y_train,y_trainpred))
print("Accuracy in test_set= ",accuracy_score(y_test,y_pred))
```

```
Accuracy in train_set=  1.0
Accuracy in test_set=  1.0
```

Random Forest after Feature Importance

In [28]:

```
feature_names = ['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm','Species']
feature_imp = pd.Series(clf.feature_importances_,index=feature_names)
feature_imp.sort_values(ascending = False)
```

Out[28]:

```
SepalLengthCm    0.431627
Species          0.250332
PetalWidthCm     0.238259
SepalWidthCm     0.068720
PetalLengthCm    0.011061
dtype: float64
```

In [29]:

```
iris.drop(['SepalWidthCm'],axis =1, inplace = True)
iris.drop(['PetalLengthCm'],axis =1, inplace = True)

iris.head()
```

Out[29]:

	Id	SepalLengthCm	PetalWidthCm	Species
0	**1**	**5.1**	**0.2**	**Iris-setosa**
1	**2**	**4.9**	**0.2**	**Iris-setosa**
2	**3**	**4.7**	**0.2**	**Iris-setosa**
3	**4**	**4.6**	**0.2**	**Iris-setosa**
4	**5**	**5.0**	**0.2**	**Iris-setosa**

In [30]:

```
X = iris.iloc[:,:-1]
y = iris.iloc[:,-1]
```

In [31]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =0.20,random_state = 1
)
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)
y_trainpred = clf.predict(X_train)
```

In [32]:

```
print("Accuracy in train_set= ",accuracy_score(y_train,y_trainpred))
print("Accuracy in test_set= ",accuracy_score(y_test,y_pred))
```

```
Accuracy in train_set=  1.0
Accuracy in test_set=  1.0
```