Rishabh Patil Sapid:60009200056 batch:K3

Q1)Perform RandomForest from scratch on dataset 1.

```
from random import seed
from random import randrange
from csv import reader
from math import sqrt


# Load a CSV file
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

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

# Convert string column to integer
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

# Split a dataset into k folds
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
```

```
# Calculate accuracy percentage
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

# Evaluate an algorithm using a cross validation split
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


# Split a dataset based on an attribute and an attribute value
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

# Calculate the Gini index for a split dataset
def gini_index(groups, classes):
    # count all samples at split point
    n_instances = float(sum([len(group) for group in groups]))
    # sum weighted Gini index for each group
    gini = 0.0
    for group in groups:
        size = float(len(group))
        # avoid divide by zero
        if size == 0:
            continue
        score = 0.0
        # score the group based on the score for each class
        for class_val in classes:
            p = [row[-1] for row in group].count(class_val) / size
            score += p * p
        # weight the group score by its relative size
        gini += (1.0 - score) * (size / n_instances)
    return gini

# Select the best split point for a dataset
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}

# Create a terminal node value
def to_terminal(group):
    outcomes = [row[-1] for row in group]
    return max(set(outcomes), key=outcomes.count)

# Create child splits for a node or make terminal
def split(node, max_depth, min_size, n_features, depth):
    left, right = node['groups']
    del(node['groups'])
    # check for a no split
    if not left or not right:
        node['left'] = node['right'] = to_terminal(left + right)
        return
    # check for max depth
    if depth >= max_depth:
        node['left'], node['right'] = to_terminal(left),
to_terminal(right)
        return
    # process 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)
    # process 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)

# Build a decision tree
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

# Make a prediction with a decision tree
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']

# Create a random subsample from the dataset with replacement
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

# Make a prediction with a list of bagged trees
def bagging_predict(trees, row):
    predictions = [predict(tree, row) for tree in trees]
    return max(set(predictions), key=predictions.count)

# Random Forest Algorithm
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)
```

```
# Test the random forest algorithm
seed(2)
# load and prepare data
filename = 'sonar.all-data.csv'
dataset = load_csv(filename)
# convert string attributes to integers
for i in range(0, len(dataset[0])-1):
	str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# evaluate algorithm
n_folds = 5
max_depth = 10
min_size = 1
sample_size = 1.0
n_features = int(sqrt(len(dataset[0])-1))
for n_trees in [1, 5, 10]:
	scores = evaluate_algorithm(dataset, random_forest, n_folds,
max_depth, min_size, sample_size, n_trees, n_features)
	print('Trees: %d' % n_trees)
	print('Scores: %s' % scores)
	print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

```
Trees: 1
Scores: [56.09756097560976, 63.41463414634146, 60.97560975609756,
58.536585365853654, 73.17073170731707]
Mean Accuracy: 62.439%
Trees: 5
Scores: [70.73170731707317, 58.536585365853654, 85.36585365853658,
75.60975609756098, 63.41463414634146]
Mean Accuracy: 70.732%
Trees: 10
Scores: [75.60975609756098, 80.48780487804879, 92.6829268292683,
73.17073170731707, 70.73170731707317]
Mean Accuracy: 78.537%
```

Rishabh Patil Sapid:60009200056 batch:K3

Q2)Compare the results of decision tree and random forest classifier for dataset 2and 3

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
df=pd.read_csv('/content/Iris (2).csv')
```

```
df.head()
```

```
   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
```

```
df.drop(['Id'],axis=1,inplace=True)
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df["Species"] = le.fit_transform(df["Species"])
```

```
df.head()
```

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

```
#decsision tree
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
y=df.iloc[:,-1]
x=df.drop(['Species'],axis=1)
```

```
feature_col=list(x.columns)
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(x,y,test_size=0.2,rando
m_state=55)
```

```
model=DecisionTreeClassifier()
model.fit(X_train,Y_train)
train_pred=model.predict(X_train)
test_pred=model.predict(X_test)
print(train_pred)
```

```
[0 0 2 2 0 1 1 0 2 1 1 0 1 0 1 2 2 2 0 2 2 0 1 2 1 0 0 1 0 0 1 2 0 2 1
2 0
 2 0 0 0 0 2 1 0 0 2 2 2 1 1 2 2 0 0 2 1 0 1 1 0 2 2 1 1 0 1 1 2 2 0 2
1 0
 0 0 0 2 1 2 2 2 1 0 2 1 1 2 0 2 1 2 1 1 0 2 1 0 1 0 1 0 1 2 2 1 1 0 2
1 1
 0 2 1 1 0 1 0 2 0]
```

```
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_train,train_pred))
print(accuracy_score(Y_test,test_pred))
```

```
1.0
0.9666666666666667
```

```
#Random Forest
```

```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_estimators=50)
models=clf.fit(X_train,Y_train)
train_random_pred=models.predict(X_train)
test_random_pred=models.predict(X_test)
print(accuracy_score(Y_train,train_random_pred))
print(accuracy_score(Y_test,test_random_pred))
```

```
1.0
0.9666666666666667
```

```
#DT+FI
```

```
feature_col=['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm',
'PetalWidthCm']
feature_imp =
pd.Series(clf.feature_importances_,index=feature_col).sort_values(asce
nding=False)
feature_imp
```

```
PetalLengthCm    0.430991
PetalWidthCm     0.426522
SepalLengthCm    0.120873
SepalWidthCm     0.021614
dtype: float64
```

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

```
df.head()
```

```
   SepalLengthCm  PetalLengthCm  PetalWidthCm  Species
0            5.1            1.4           0.2        0
1            4.9            1.4           0.2        0
2            4.7            1.3           0.2        0
3            4.6            1.5           0.2        0
4            5.0            1.4           0.2        0
```

```
y_fi=df.iloc[:,-1]
x_fi=df.drop(['Species'],axis=1)


from sklearn.model_selection import train_test_split
X_train_fi,X_test_fi,Y_train_fi,Y_test_fi=train_test_split(x_fi,y_fi,t
est_size=0.2,random_state=55)
```

```
model=DecisionTreeClassifier()
model.fit(X_train_fi,Y_train_fi)
train_pred_fi=model.predict(X_train_fi)
test_pred_fi=model.predict(X_test_fi)
print(train_pred_fi)
```

```
[0 0 2 2 0 1 1 0 2 1 1 0 1 0 1 2 2 2 0 2 2 0 1 2 1 0 0 1 0 0 1 2 0 2 1
2 0
 2 0 0 0 0 2 1 0 0 2 2 2 1 1 2 2 0 0 2 1 0 1 1 0 2 2 1 1 0 1 1 2 2 0 2
1 0
 0 0 0 2 1 2 2 2 1 0 2 1 1 2 0 2 1 2 1 1 0 2 1 0 1 0 1 0 1 2 2 1 1 0 2
1 1
 0 2 1 1 0 1 0 2 0]
```

```
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_train_fi,train_pred_fi))
print(accuracy_score(Y_test_fi,test_pred_fi))
```

```
1.0
0.9666666666666667
```

```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_estimators=50)
models=clf.fit(X_train_fi,Y_train_fi)
train_random_pred_fi=models.predict(X_train_fi)
test_random_pred_fi=models.predict(X_test_fi)
print(accuracy_score(Y_train_fi,train_random_pred_fi))
print(accuracy_score(Y_test_fi,test_random_pred_fi))
```

```
1.0
0.9666666666666667
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
l=[]
for i in range(1,62):
  l.append(i)
df=pd.read_csv('/content/sonar.all-data.csv',names=l)
```

```
df.head()
```

```
       1       2       3       4       5       6       7       8
9   \
0  0.0200  0.0371  0.0428  0.0207  0.0954  0.0986  0.1539  0.1601
0.3109
1  0.0453  0.0523  0.0843  0.0689  0.1183  0.2583  0.2156  0.3481
0.3337
2  0.0262  0.0582  0.1099  0.1083  0.0974  0.2280  0.2431  0.3771
0.5598
3  0.0100  0.0171  0.0623  0.0205  0.0205  0.0368  0.1098  0.1276
0.0598
4  0.0762  0.0666  0.0481  0.0394  0.0590  0.0649  0.1209  0.2467
0.3564

       10  ...      52      53      54      55      56      57      58
\
0  0.2111  ...  0.0027  0.0065  0.0159  0.0072  0.0167  0.0180  0.0084

1  0.2872  ...  0.0084  0.0089  0.0048  0.0094  0.0191  0.0140  0.0049

2  0.6194  ...  0.0232  0.0166  0.0095  0.0180  0.0244  0.0316  0.0164

3  0.1264  ...  0.0121  0.0036  0.0150  0.0085  0.0073  0.0050  0.0044

4  0.4459  ...  0.0031  0.0054  0.0105  0.0110  0.0015  0.0072  0.0048


       59      60  61
0  0.0090  0.0032   R
1  0.0052  0.0044   R
2  0.0095  0.0078   R
3  0.0040  0.0117   R
4  0.0107  0.0094   R

[5 rows x 61 columns]
```

```
df.isnull().sum()
```

```
1     0
2     0
```

```
3     0
4     0
5     0
     ..
57    0
58    0
59    0
60    0
61    0
Length: 61, dtype: int64
```

```
df[61].value_counts()
```

```
M    111
R     97
Name: 61, dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df[61] = le.fit_transform(df[61])
```

```
df.head()
```

```
       1       2       3       4       5       6       7       8
9   \
0  0.0200  0.0371  0.0428  0.0207  0.0954  0.0986  0.1539  0.1601
0.3109
1  0.0453  0.0523  0.0843  0.0689  0.1183  0.2583  0.2156  0.3481
0.3337
2  0.0262  0.0582  0.1099  0.1083  0.0974  0.2280  0.2431  0.3771
0.5598
3  0.0100  0.0171  0.0623  0.0205  0.0205  0.0368  0.1098  0.1276
0.0598
4  0.0762  0.0666  0.0481  0.0394  0.0590  0.0649  0.1209  0.2467
0.3564

       10  ...      52      53      54      55      56      57      58
\
0  0.2111  ...  0.0027  0.0065  0.0159  0.0072  0.0167  0.0180  0.0084

1  0.2872  ...  0.0084  0.0089  0.0048  0.0094  0.0191  0.0140  0.0049

2  0.6194  ...  0.0232  0.0166  0.0095  0.0180  0.0244  0.0316  0.0164

3  0.1264  ...  0.0121  0.0036  0.0150  0.0085  0.0073  0.0050  0.0044

4  0.4459  ...  0.0031  0.0054  0.0105  0.0110  0.0015  0.0072  0.0048


       59      60  61
0  0.0090  0.0032   1
```

```
1  0.0052  0.0044   1
2  0.0095  0.0078   1
3  0.0040  0.0117   1
4  0.0107  0.0094   1

[5 rows x 61 columns]
```

```
df[61].value_counts()
```

```
0    111
1     97
Name: 61, dtype: int64
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
y=df.iloc[:,-1]
x=df.drop([61],axis=1)
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(x,y,test_size=0.2,rando
m_state=25)
```

```
model=DecisionTreeClassifier()
model.fit(X_train,Y_train)
train_pred=model.predict(X_train)
test_pred=model.predict(X_test)
print(train_pred)
```

```
[0 1 0 1 1 1 0 0 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 1 1 0
0 0
 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 0 0 1 0 1 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1
1 1
 0 0 1 0 1 1 0 1 1 0 1 1 1 0 0 1 1 1 0 1 0 1 0 1 1 1 0 0 1 1 0 1 1 0 1
0 0
 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 1 0 0 1 0 1
1 0
 0 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0]
```

```
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_train,train_pred))
print(accuracy_score(Y_test,test_pred))
```

```
1.0
0.7380952380952381
```

```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_estimators=50)
models=clf.fit(X_train,Y_train)
train_random_pred=models.predict(X_train)
test_random_pred=models.predict(X_test)
```

```
print(accuracy_score(Y_train,train_random_pred))
print(accuracy_score(Y_test,test_random_pred))
```

```
1.0
0.8571428571428571
```

```
feature_col=list(x.columns)
feature_imp =
pd.Series(clf.feature_importances_,index=feature_col).sort_values(asce
nding=False)
feature_imp
```

```
11    0.071702
9     0.050912
10    0.043391
12    0.032228
47    0.030840
45    0.028840
36    0.024718
28    0.024538
13    0.024148
21    0.024134
17    0.023270
49    0.022797
2     0.021733
32    0.021601
51    0.021025
4     0.020827
16    0.020419
14    0.019385
20    0.019162
22    0.018935
48    0.018051
1     0.016490
37    0.016347
18    0.015807
5     0.015693
46    0.015113
34    0.013571
6     0.013567
31    0.013193
43    0.012970
15    0.012771
23    0.012378
39    0.012335
41    0.011997
35    0.011525
59    0.011359
60    0.011224
53    0.011162
40    0.010850
```

```
24    0.010804
55    0.010724
7     0.010521
52    0.010481
29    0.010414
19    0.010264
30    0.009630
27    0.009555
44    0.009481
50    0.009272
54    0.009113
57    0.009112
38    0.008506
58    0.008469
8     0.008399
42    0.007833
3     0.007375
33    0.006811
26    0.005852
56    0.003641
25    0.002735
dtype: float64
```

```
df.drop([25,56,26,33,3],axis=1,inplace=True)
```

```
df.drop([42,8,58,38,57],axis=1,inplace=True)
```

```
y_fi=df.iloc[:,-1]
x_fi=df.drop([61],axis=1)
```

```
X_train_fi,X_test_fi,Y_train_fi,Y_test_fi=train_test_split(x_fi,y_fi,t
est_size=0.2,random_state=30)
```

```
model=DecisionTreeClassifier()
model.fit(X_train_fi,Y_train_fi)
train_pred_fi=model.predict(X_train_fi)
test_pred_fi=model.predict(X_test_fi)
print(train_pred_fi)
```

```
[0 0 1 0 0 1 0 0 1 1 1 0 1 0 1 1 1 0 0 0 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0
1 1
 1 0 1 1 1 1 1 0 0 1 1 0 0 1 1 1 0 0 0 0 1 0 1 0 1 1 0 0 1 1 0 0 0 1 1
1 0
 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0 1 0 1 1 0 1 1 1 1 1 1
1 1
 0 0 0 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 0 1 0 1 1 0 1 0 1 1 1 0 0 0 0 0
1 0
 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 1]
```

```
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_train_fi,train_pred_fi))
print(accuracy_score(Y_test_fi,test_pred_fi))
```

```
1.0
0.6904761904761905
```

```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_estimators=150)
models=clf.fit(X_train_fi,Y_train_fi)
train_random_pred_fi=models.predict(X_train_fi)
test_random_pred_fi=models.predict(X_test_fi)
print(accuracy_score(Y_train_fi,train_random_pred_fi))
print(accuracy_score(Y_test_fi,test_random_pred_fi))
```

```
1.0
0.8571428571428571
```