**DMWA Lab Test**

Teghdeep Kapoor

180104050

B12

Q1

import csv

import math

import random

def encode\_class(mydata):

classes = []

for i in range(len(mydata)):

if mydata[i][-1] not in classes:

classes.append(mydata[i][-1])

for i in range(len(classes)):

for j in range(len(mydata)):

if mydata[j][-1] == classes[i]:

mydata[j][-1] = i

return mydata

def splitting(mydata, ratio):

train\_num = int(len(mydata) \* ratio)

train = []

test = list(mydata)

while len(train) < train\_num:

# index generated randomly from range 0

# to length of testset

index = random.randrange(len(test))

train.append(test.pop(index))

return train, test

def groupUnderClass(mydata):

dict = {}

for i in range(len(mydata)):

if (mydata[i][-1] not in dict):

dict[mydata[i][-1]] = []

dict[mydata[i][-1]].append(mydata[i])

return dict

def mean(numbers):

return sum(numbers) / float(len(numbers))

def std\_dev(numbers):

avg = mean(numbers)

variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers) - 1)

return math.sqrt(variance)

def MeanAndStdDev(mydata):

info = [(mean(attribute), std\_dev(attribute)) for attribute in zip(\*mydata)]

del info[-1]

return info

def MeanAndStdDevForClass(mydata):

info = {}

dict = groupUnderClass(mydata)

for classValue, instances in dict.items():

info[classValue] = MeanAndStdDev(instances)

return info

def calculateGaussianProbability(x, mean, stdev):

expo = math.exp(-(math.pow(x - mean, 2) / (2 \* math.pow(stdev, 2))))

return (1 / (math.sqrt(2 \* math.pi) \* stdev)) \* expo

def calculateClassProbabilities(info, test):

probabilities = {}

for classValue, classSummaries in info.items():

probabilities[classValue] = 1

for i in range(len(classSummaries)):

mean, std\_dev = classSummaries[i]

x = test[i]

probabilities[classValue] \*= calculateGaussianProbability(x, mean, std\_dev)

return probabilities

def predict(info, test):

probabilities = calculateClassProbabilities(info, test)

bestLabel, bestProb = None, -1

for classValue, probability in probabilities.items():

if bestLabel is None or probability > bestProb:

bestProb = probability

bestLabel = classValue

return bestLabel

def getPredictions(info, test):

predictions = []

for i in range(len(test)):

result = predict(info, test[i])

predictions.append(result)

return predictions

def accuracy\_rate(test, predictions):

correct = 0

for i in range(len(test)):

if test[i][-1] == predictions[i]:

correct += 1

return (correct / float(len(test))) \* 100.0

filename = r' link of csv file’

mydata = csv.reader(open(filename, "rt"))

mydata = list(mydata)

mydata = encode\_class(mydata)

for i in range(len(mydata)):

mydata[i] = [float(x) for x in mydata[i]]

ratio = 0.7

train\_data, test\_data = splitting(mydata, ratio)

print('Total number of examples are: ', len(mydata))

print('Out of these, training examples are: ', len(train\_data))

print("Test examples are: ", len(test\_data))

info = MeanAndStdDevForClass(train\_data)

predictions = getPredictions(info, test\_data)

accuracy = accuracy\_rate(test\_data, predictions)

print("Accuracy of your model is: ", accuracy)

Q2

Steps –

1) Open the arff file in weka

2) Since all the attributes are nominal we can apply ID3 algo

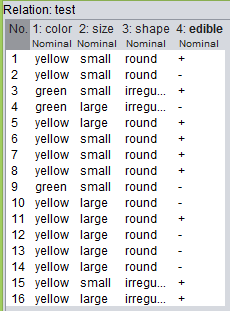
3) Go to classify tab

4) Choose J48 within the tree option

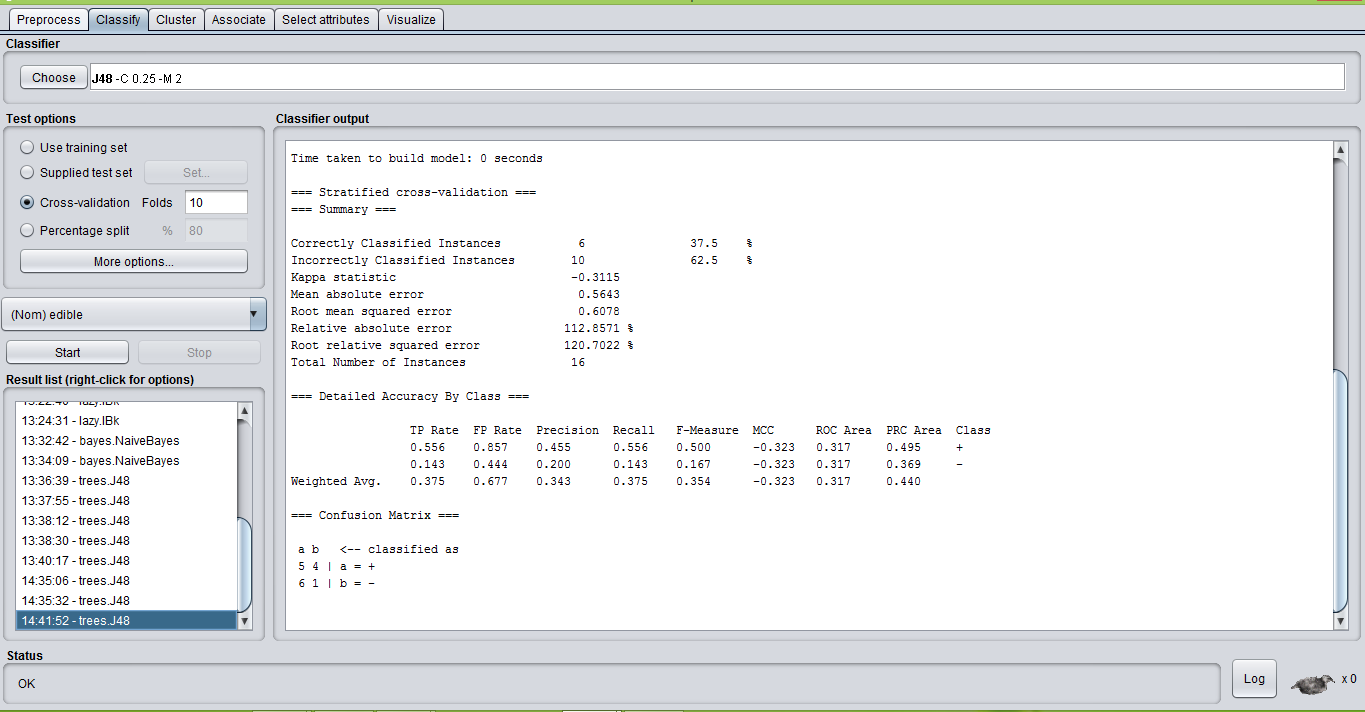
5) Select cross-validation with 10 folds. Then with 5 folds

6) Select Percentage Spilt as 20 %

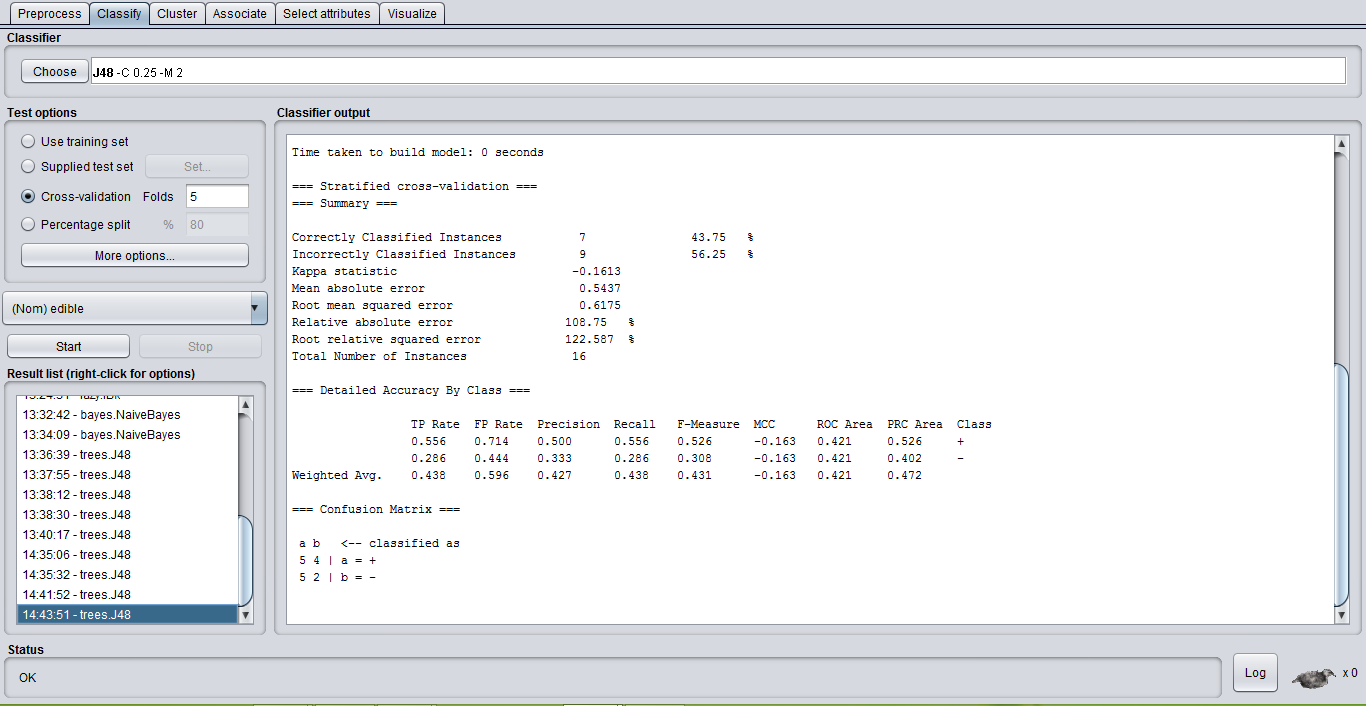
1. Making Dataset test.arff



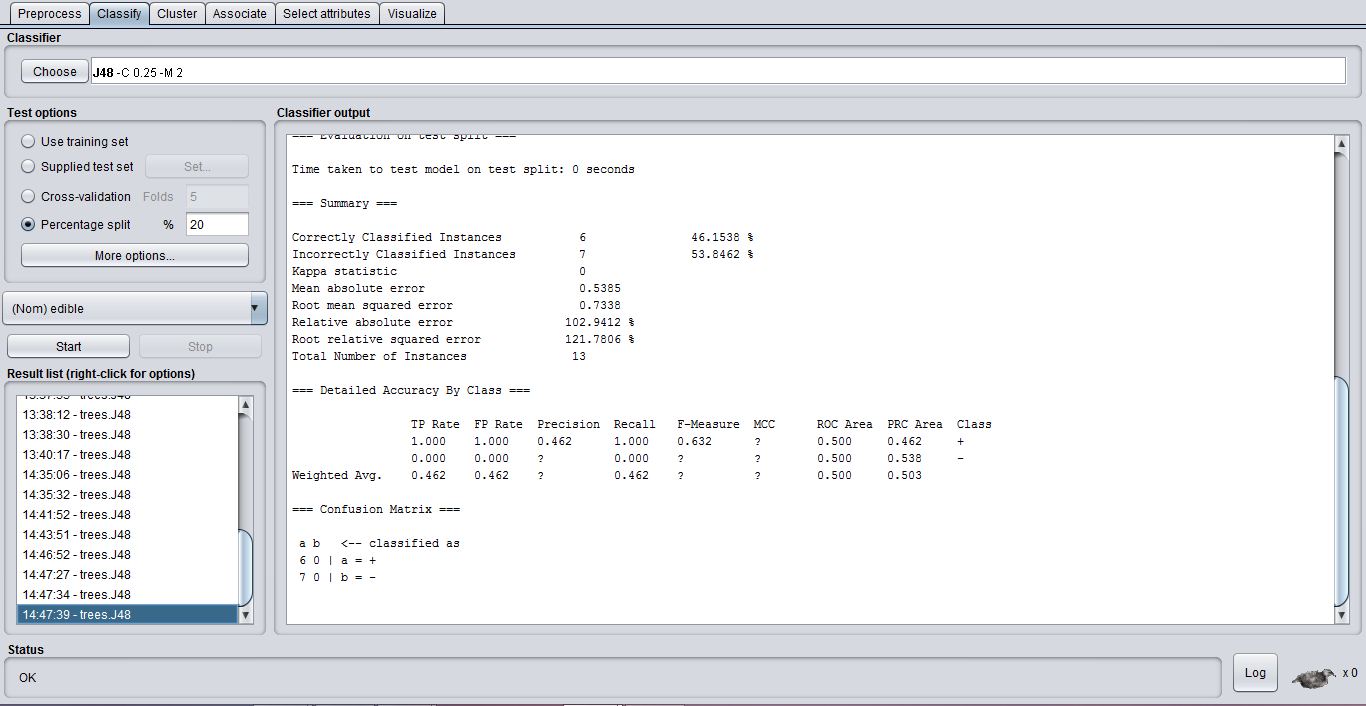
1. Applying J48 Classifier from Tree Classifier Using Validation Test(10 Folds)



1. Applying J48 Classifier from Tree Classifier Using Validation Test(5 Folds)



1. Applying J48 Classifier from Tree Classifier Using Split (20% Train and 80% Test)



Comparison Between Recall and Accuracy -:

1. Validation Test(10 Folds)

Recall – 0.556

Accuracy – 37.5%

1. Test Dataset(20 : 80)

Recall – 1.00

Accuracy – 46.1538%