SMAI Spring 2016

Assignment 1

Submitted By - Akash Agarwal (201406593)

Datasets Used

1) Iris Data Set
Link : http://archive.ics.uci.edu/ml/datasets/Iris

Number of features: 5
Number of Instances: 150
Number of Classes: 3

2) Wine Data Set
Link: http://archive.ics.uci.edu/ml/datasets/Wine

Number of features: 13
Number of Instances: 178
Number of Classes: 3

3) Banknote authentication Data Set
Link: http://archive.ics.uci.edu/ml/datasets/banknote+authentication

Number of features: 5
Number of Instances: 1372
Number of Classes: 2

Criteria Kept in mind:

All the values should be real values. No missing values should be there.

Code Written

Note: Numpy is used to calculate only mean and standard deviation. All other features are implemented in full without any use of external libraries.

```
#!/usr/bin/python

import csv
import sys
import random
import math
import numpy

def randomSubSampling(dataset):
    Testdataset = []
    Traindataset = []
    random.shuffle(dataset)
    for i in range(len(dataset)):
        if i%2 ==0:
            Testdataset.append(dataset[i])
        else:
            Traindataset.append(dataset[i])
    return [Testdataset, Traindataset]

def fiveFoldVerif(dataset,kthfold):
    Testdataset = []
    Traindataset = []
    random.shuffle(dataset)
    dim = len(dataset)

    foldsize = dim/5

    for i in range(len(dataset)):
        if i > foldsize*kthfold and i < foldsize*(kthfold+1):
            Testdataset.append(dataset[i])
        else:
            Traindataset.append(dataset[i])
    return [Testdataset, Traindataset]

def classifyInstanceNN(Traindataset, inst, classes,k):
    distVect = {}
    dist = 0
    for row in Traindataset:
        for i in range(len(row)):
            try:
                dist = dist + (row[i]-inst[i])*(row[i]-inst[i])
            except:
                className = row[i]
        dist = math.sqrt(dist)
        distVect[dist] = className
        dist = 0
    keys = sorted(distVect)
    counts={}

    for i in range(k):
        counts[distVect[keys[i]]] = 0

    for i in range(k):
        counts[distVect[keys[i]]] = counts[distVect[keys[i]]] +1

    mx = -1
    clas = None

    for key in counts.keys():
        if counts[key] > mx :
            mx = counts[key]
            clas = key
    return clas

def applyKNN(dataset, classes, classColumnNum, k, flag, kthfold):
    if flag:
        Testdataset, Traindataset = randomSubSampling(dataset)
    else:
        Testdataset, Traindataset = fiveFoldVerif(dataset, kthfold)

    confusionMatrix = [[key,[0,0,0]] for key in classes.keys()]
    confusionMatrix = dict(confusionMatrix)

    for row in Testdataset:
        clas = classifyInstanceNN(Traindataset, row, classes,k)
        confusionMatrix[row[classColumnNum]][classes[clas]] =
confusionMatrix[row[classColumnNum]][classes[clas]]+1

    return confusionMatrix

def extractClasses(dataset, classColumnNum):
    col = [row[classColumnNum] for row in dataset]
    col = list(set(col))
    col = [[col[i],i] for i in range(len(col))]
    return col
```

```
def getInputArgs():
        try:
                fileName = sys.argv[1]
                classColumnNum = int(sys.argv[2])
        except:
                print "arguments : <fileName> <classColumnNumber>"
                sys.exit()

        return [fileName, classColumnNum]

def getDataset(fileName,classColumnNum):
        dataset = []
        with open(fileName, 'rb') as f:
            reader = csv.reader(f)

            for row in reader:
                tmp = []
                if len(row) == 0:
                        continue
                for i in range(len(row)):
                        try:
                                if i == classColumnNum:
                                        tmp.append(row[i])
                                else:
                                        tmp.append(float(row[i]))
                        except:
                                tmp.append(row[i])
                dataset.append(tmp)
        f.close()
        return dataset

def getAccuracy(confusionMatrix,classes):
        correct=0
        incorrect=0
        for key in confusionMatrix.keys():
                for i in range(len(classes.keys())):
                        if classes[key] == i:
                                correct = correct + confusionMatrix[key][i]
                        else:
                                incorrect = incorrect + confusionMatrix[key][i]
        total = correct+incorrect
        return (correct/float(total))*100

def calculate1NNRand(dataset, classes, classColumnNum):
        FinConfusionMat = []
        dimension = 0
        stats = []
```

```
    for k in range(10):
        confusionMatrix = applyKNN(dataset, classes, classColumnNum,1, True,0)
        stats.append(getAccuracy(confusionMatrix,classes))
        if k == 0:
            FinConfusionMat = confusionMatrix
        else:
            for key in confusionMatrix.keys():
                for i in range(len(classes.keys())):
                    FinConfusionMat[key][i] = FinConfusionMat[key][i]+confusionMatrix[key][i]

    arr = numpy.array(stats)
    print "\n**************************"
    print "Final confusion Matrix for 1NN with 10 iterations of randomSubSampling:\n"
    print FinConfusionMat

    print ""
    print "Mean: " + str(arr.mean())
    print "Standard Deviation: " + str(arr.std())
    print "**************************\n"

def calculate3NNRand(dataset, classes, classColumnNum):
    FinConfusionMat = []
    dimension = 0
    stats = []

    for k in range(10):
        confusionMatrix = applyKNN(dataset, classes, classColumnNum,3, True,0)
        stats.append(getAccuracy(confusionMatrix,classes))
        if k == 0:
            FinConfusionMat = confusionMatrix
        else:
            for key in confusionMatrix.keys():
                for i in range(len(classes.keys())):
                    FinConfusionMat[key][i] = FinConfusionMat[key][i]+confusionMatrix[key][i]

    arr = numpy.array(stats)
    print "\n**************************"
    print "Final confusion Matrix for 3NN with 10 iterations of randomSubSampling:\n"
    print FinConfusionMat

    print ""
    print "Mean: " + str(arr.mean())
    print "Standard Deviation: " + str(arr.std())
    print "**************************\n"

def calculate1NNfold(dataset, classes, classColumnNum):
    FinConfusionMat = []
```

```
	dimension = 0
	stats = []

	for k in range(5):
		confusionMatrix = applyKNN(dataset, classes, classColumnNum,1, False, k)
		stats.append(getAccuracy(confusionMatrix,classes))
		if k == 0:
			FinConfusionMat = confusionMatrix
		else:
			for key in confusionMatrix.keys():
				for i in range(len(classes.keys())):
					FinConfusionMat[key][i] = FinConfusionMat[key][i]+confusionMatrix[key][i]

	arr = numpy.array(stats)
	print "\n*************************"
	print "Final confusion Matrix for 1NN with 5 fold cross validation:\n"
	print FinConfusionMat

	print ""
	print "Mean: " + str(arr.mean())
	print "Standard Deviation: " + str(arr.std())
	print "*************************\n"

def calculate3NNfold(dataset, classes, classColumnNum):
	FinConfusionMat = []
	dimension = 0
	stats = []

	for k in range(5):
		confusionMatrix = applyKNN(dataset, classes, classColumnNum,3, False, k)
		stats.append(getAccuracy(confusionMatrix,classes))
		if k == 0:
			FinConfusionMat = confusionMatrix
		else:
			for key in confusionMatrix.keys():
				for i in range(len(classes.keys())):
					FinConfusionMat[key][i] = FinConfusionMat[key][i]+confusionMatrix[key][i]

	arr = numpy.array(stats)
	print "\n*************************"
	print "Final confusion Matrix for 3NN with 5 fold cross validation:\n"
	print FinConfusionMat

	print ""
	print "Mean: " + str(arr.mean())
	print "Standard Deviation: " + str(arr.std())
	print "*************************\n"
def main():
    fileName, classColumnNum = getInputArgs()
    dataset = getDataset(fileName,classColumnNum)
    classes = extractClasses(dataset, classColumnNum)
    classes = dict(classes)

    calculate1NNRand(dataset, classes, classColumnNum)
    calculate3NNRand(dataset, classes, classColumnNum)

    calculate1NNfold(dataset, classes, classColumnNum)
    calculate3NNfold(dataset, classes, classColumnNum)

    del dataset


main()

#############  Code Ends Here  ################
```

Results

Note: Mentioned confusion matrix are the sum total of each element of 10 iterations of sub confusion matrices.

Iris DataSet

- Confusion Matrix for 1NN with 10 iterations of random SubSampling:

Actual/Predicted	**Iris-virginica**	**Iris-setosa**	**Iris-versicolor**
Iris-virginica	233	0	22
Iris-setosa	0	246	0
Iris-versicolor	16	0	223

Mean: 94.9333333333
Standard Deviation: 1.55492050529

- Confusion Matrix for 3NN with 10 iterations of random SubSampling:

Actual/Predicted	**Iris-virginica**	**Iris-setosa**	**Iris-versicolor**
Iris-virginica	229	0	15
Iris-setosa	0	259	0
Iris-versicolor	22	0	225

Mean: 95.0666666667
Standard Deviation: 1.46666666667

- Confusion Matrix for 1NN with 5 fold cross validation:

Actual/Predicted	**Iris-virginica**	**Iris-setosa**	**Iris-versicolor**
Iris-virginica	43	0	5
Iris-setosa	0	48	0
Iris-versicolor	3	0	46

Mean: 94.4827586207
Standard Deviation: 3.51656518179

- Confusion Matrix for 3NN with 5 fold cross validation:

Actual/Predicted	**Iris-virginica**	**Iris-setosa**	**Iris-versicolor**
Iris-virginica	43	0	0
Iris-setosa	0	52	0
Iris-versicolor	4	0	46

Mean: 97.2413793103
Standard Deviation: 2.58045337019

Observations:

- Class distribution is even in the above dataset.
- Almost all the KNN algorithms are giving >90% accurate result which is fairly acceptable.
- Variance of Random SubSampling is lower than variance of 5-fold cross validation.
- Accuracy of 5-fold cross validation is somewhat higher than random subsampling as more amount of train data is used to train the classifier.

Wine Data Set

- Confusion Matrix for 1NN with 10 iterations of random SubSampling:

Actual/Predicted	**1**	**3**	**2**

1	247	16	23
3	28	127	90
2	25	77	257

Mean: 70.8988764045
Standard Deviation: 3.84164516793

- Confusion Matrix for 3NN with 10 iterations of random SubSampling:

Actual/Predicted	**1**	**3**	**2**
1	246	19	16
3	39	138	79
2	27	109	217

Mean: 67.5280898876
Standard Deviation: 4.06518034048

- Confusion Matrix for 1NN with 5 fold cross validation:

Actual/Predicted	**1**	**3**	**2**
1	51	4	1
3	3	26	13
2	5	9	58

Mean: 79.4117647059
Standard Deviation: 4.15945165404

- Confusion Matrix for 3NN with 5 fold cross validation:

Actual/Predicted	**1**	**3**	**2**
1	42	4	2
3	9	18	19
2	8	17	51

Mean: 65.2941176471
Standard Deviation: 2.88175263857

Observations:

- Class distribution is uneven in the above dataset but with lower variance.
- Huge variation in accuracy and variance can be seen.
- Only limited iterations are shown above, but upon multiple iterations, accuracies of all the techniques seem to differ largely and no clear winner is there.
- Comparing Random SubSampling and 5 fold cross validation, 5 fold cross validation performed better on an overall basis.
- Above reading indicate, there is no clear cut feature which could help in achieving higher accuracies.
- Use of another classifier or ensembling technique is recommended to achieve higher accuracies.

Banknote Authentication Data Set

- Confusion Matrix for 1NN with 10 iterations of random SubSampling:

Actual/Predicted	**1**	**0**
1	3063	0
0	5	3792

Mean: 99.9271137026
Standard Deviation: 0.0728862973761

- Confusion Matrix for 3NN with 10 iterations of random SubSampling:

Actual/Predicted	**1**	**0**
1	3080	0
0	16	3764

Mean: 99.7667638484
Standard Deviation: 0.509995792552

- Confusion Matrix for 1NN with 5 fold cross validation:

Actual/Predicted	**1**	**0**
1	612	0

0	1	752

Mean: 99.9267399267
Standard Deviation: 0.14652014652

- Confusion Matrix for 3NN with 5 fold cross validation:

Actual/Predicted	**1**	**0**
1	598	0
0	0	767

Mean: 100.0
Standard Deviation: 0.0

Observations:

- Class distribution is uneven in the above dataset but with lower variance.
- Almost no variation in accuracy and variance can be seen.
- Only limited iterations are shown above, but upon multiple iterations, accuracies of all the techniques seem to differ minutely and KNN is the clear winner as a reliable classifier.
- Comparing Random SubSampling and 5 fold cross validation, both of them performed almost equally.
- Feature set is clearly good and helped in classifying with higher accuracies.

Graphs

Figures

X Axis: Sepal Width
Y Axis: Petal Width

Observations

Fig 1 : It depicts the overall distribution of classes. Class in green color is clearly separated from the other two classes denoting good choice of feature vector. Other two classes have some clash and linear will be suitable to separate the two classes in the highest accurate form.

Fig 2: This figure shows the decision boundary in the most sophisticated form. Simply, a linear curve would also satisfy the need of most of the classification decision.

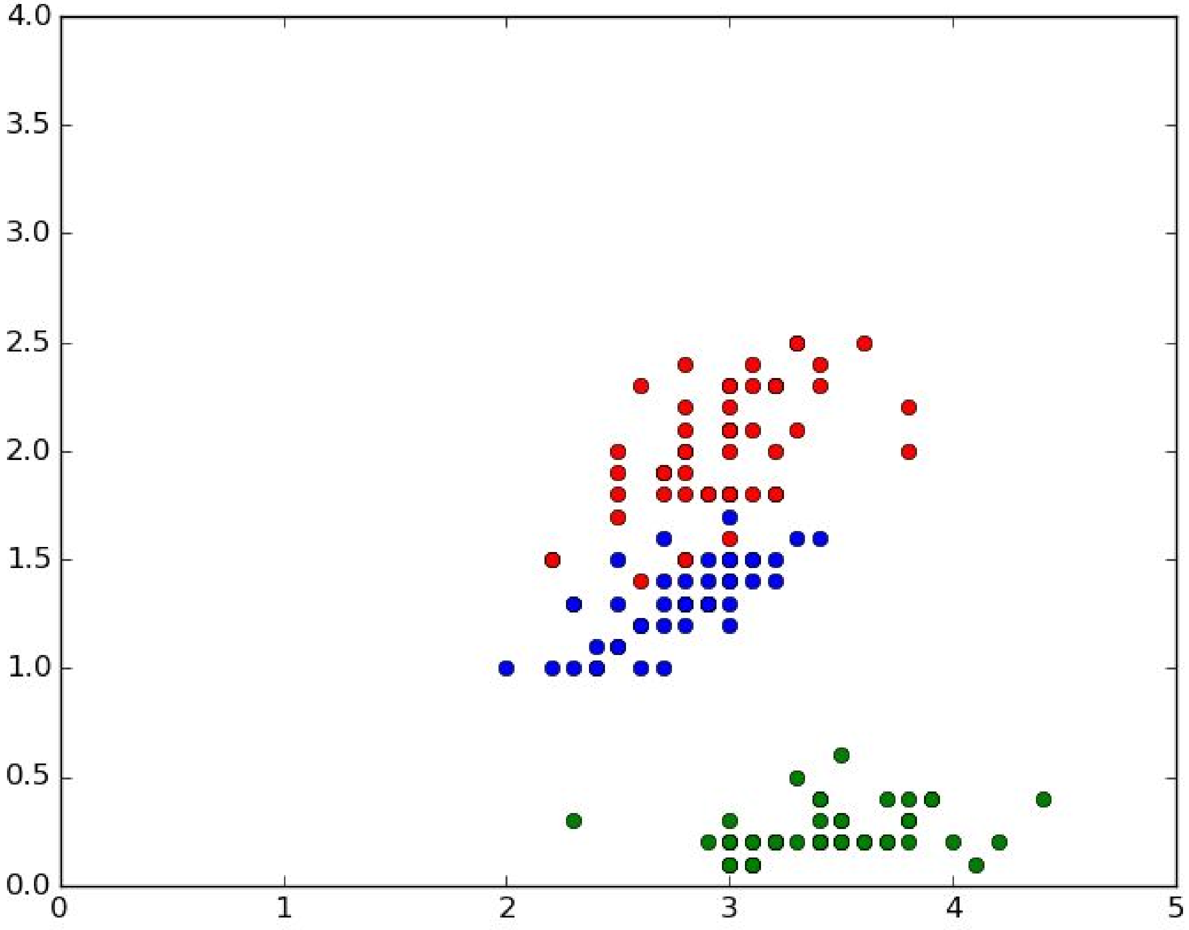

Fig 1

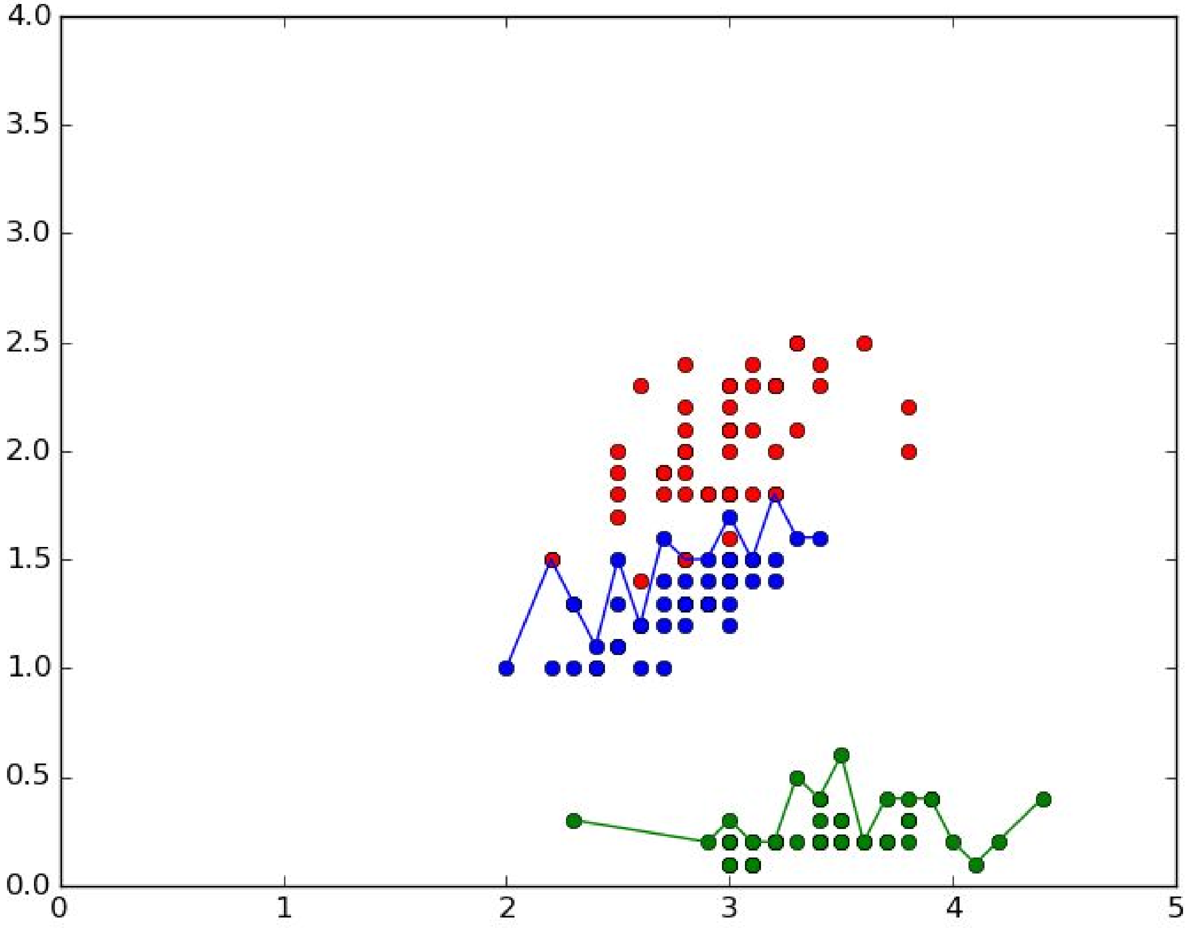

Fig 2

Code

```
#!/usr/bin/python

import csv
import sys
import matplotlib.pyplot as plt

def getInputArgs():
	try:
		fileName = sys.argv[1]
		classColumnNum = int(sys.argv[2])
	except:
		print "arguments : <fileName> <classColumnNumber>"
		sys.exit()
```

```
        return [fileName, classColumnNum]

def getDataset(fileName,classColumnNum):
        dataset = []
        with open(fileName, 'rb') as f:
           reader = csv.reader(f)

           for row in reader:
                tmp = []
                if len(row) == 0:
                        continue
                for i in range(len(row)):
                        try:
                                if i == classColumnNum:
                                        tmp.append(row[i])
                                else:
                                        tmp.append(float(row[i]))
                        except:
                                tmp.append(row[i])
                dataset.append(tmp)
        f.close()
        return dataset

fileName, classColumnNum = getInputArgs()
dataset = getDataset(fileName,classColumnNum)

x = []
y = []

for i in range(len(dataset)):
        for j in range(len(dataset[i])):
                if j ==1:
                        x.append(dataset[i][j])
                if j == 3:
                        y.append(dataset[i][j])

plt.plot(x, y, 'ro')

plt.show()
```

Answer 4:

Fig 1 : It depicts the overall distribution of classes. Class in green color is clearly separated from the other two classes denoting good choice of feature vector. Other two classes have some clash and linear will be suitable to separate the two classes in the highest accurate form.

Fig 2: This figure shows the decision boundary in the most sophisticated form. Simply, a linear curve would also satisfy the need of most of the classification decision.

Therefore, yes, the decision boundary for iris dataset will be piecewise linear. Classification of points is done into regions as they are located clearly in different regions with little overlapping on the boundary. We can clearly infer from the figure and hence the classifier will be piecewise linear.