**Assignment 1 (K-Nearest Neighbour Classifier)**

**Data Set : Iris Plants Database**

Number of Instances: 150 (50 in each of three classes)
Number of Attributes: 4 numeric, predictive attributes and the class

Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
   -- Iris Setosa
   -- Iris Versicolour
   -- Iris Virginica

**5- Fold Cross Validation (k=1):**

| #Run | Mean (%) | Standard Deviation |
|---|---|---|
| 1 | 96.0 | 4.89 |
| 2 | 94.6 | 3.39 |
| 3 | 96.0 | 3.26 |
| 4 | 95.3 | 4.0 |
| 5 | 96.0 | 1.33 |
| 6 | 96.0 | 1.33 |
| 7 | 96.0 | 3.88 |
| 8 | 96.0 | 2.49 |
| 9 | 95.3 | 3.39 |
| 10 | 96.0 | 3.88 |

**Grand Mean -> 95.73**

**Confusion Matrix:**

| | **Iris-setosa** | **Iris-versicolor** | **Iris-virginica** |
|---|---|---|---|
| **Iris-setosa** | 26 | 0 | 0 |
| **Iris-versicolor** | 0 | 23 | 2 |
| **Iris-virginica** | 0 | 3 | 21 |

**Random Subsampling Approach (k=1):**

| **#Run** | **Accuracy** |
|---|---|
| 1 | 93.3 |
| 2 | 96.0 |
| 3 | 96.0 |
| 4 | 96.0 |
| 5 | 97.3 |
| 6 | 93.3 |
| 7 | 89.3 |
| 8 | 98.6 |
| 9 | 93.4 |
| 10 | 96.0 |

**Mean: 94.9**
**Standard Deviation: 2.515**

**5- Fold Cross Validation (k=3):**

| #Run | Mean (%) | Standard Deviation |
|---|---|---|
| 1 | 96.0 | 3.88 |
| 2 | 96.0 | 2.49 |
| 3 | 96.0 | 1.33 |
| 4 | 96.6 | 2.98 |
| 5 | 97.3 | 1.33 |
| 6 | 95.3 | 1.63 |
| 7 | 96.0 | 2.49 |
| 8 | 96.0 | 1.33 |
| 9 | 96.6 | 2.10 |
| 10 | 96.0 | 2.49 |

**Grand Mean -> 96.2**

**Confusion Matrix:**

| | **Iris-setosa** | **Iris-versicolor** | **Iris-virginica** |
|---|---|---|---|
| **Iris-setosa** | 19 | 0 | 0 |
| **Iris-versicolor** | 0 | 29 | 0 |
| **Iris-virginica** | 0 | 1 | 26 |

**Random Subsampling Approach (k=3):**

| #Run | Accuracy |
|---|---|
| 1 | 98.6 |
| 2 | 96.0 |
| 3 | 96.0 |
| 4 | 98.6 |
| 5 | 96.0 |
| 6 | 96.0 |
| 7 | 96.0 |
| 8 | 94.6 |
| 9 | 94.6 |
| 10 | 97.3 |

**Mean: 96.4**
**Standard Deviation: 1.339**

**Confusion Matrix:**

| | Iris-setosa | Iris-versicolor | Iris-virginica |
|---|---|---|---|
| **Iris-setosa** | 24 | 0 | 0 |
| **Iris-versicolor** | 0 | 26 | 3 |
| **Iris-virginica** | 0 | 0 | 22 |

**Data Set : Wisconsin Breast Cancer Database**
**(file name: breast-cancer-wisconsin.data) as multiple datafiles of available**

Attributes 2 through 10 have been used to represent instances.
Each instance has one of 2 possible classes: benign or malignant.

Number of Instances: 699
Number of Attributes: 10 plus the class attribute

Attribute Information:

| # | Attribute | Domain |
|---|---|---|
| 1. | Sample code number | id number |
| 2. | Clump Thickness | 1 - 10 |
| 3. | Uniformity of Cell Size | 1 - 10 |
| 4. | Uniformity of Cell Shape | 1 - 10 |
| 5. | Marginal Adhesion | 1 - 10 |
| 6. | Single Epithelial Cell Size | 1 - 10 |
| 7. | Bare Nuclei | 1 - 10 |
| 8. | Bland Chromatin | 1 - 10 |
| 9. | Normal Nucleoli | 1 - 10 |
| 10. | Mitoses | 1 - 10 |
| 11. | Class: | (2 for benign, 4 for malignant) |

Missing attribute values: 16

**They are denoted as “?”. Since the value of the missing attributes ranges from 1-10, assigned them value “5” to use the information instead of discarding the row.**

**Note: To keep track of this and also to drop the 0th column which is “Sample Code Number”, had to hardcode with a few if conditions for replacing missing data.**

**5- Fold Cross Validation (k=1):**

| #Run | Mean (%) | Standard Deviation |
|---|---|---|
| 1 | 95.1 | 1.63 |
| 2 | 95.1 | 2.16 |
| 3 | 94.5 | 0.53 |
| 4 | 95.7 | 1.59 |
| 5 | 95.4 | 1.72 |
| 6 | 96.1 | 2.15 |
| 7 | 94.8 | 1.66 |
| 8 | 95.4 | 1.32 |
| 9 | 94.9 | 0.83 |
| 10 | 95.2 | 1.71 |

**Grand Mean -> 95.26**

**Confusion Matrix:**

| | **2 (benign)** | **4 (malignant)** |
|---|---|---|
| **2 (benign)** | 167 | 49 |
| **4 (malignant)** | 80 | 54 |

**Random Subsampling Approach (k=1):**

| #Run | Accuracy |
|---|---|
| 1 | 96.1 |
| 2 | 96.1 |
| 3 | 94.1 |
| 4 | 96.4 |
| 5 | 95.8 |
| 6 | 95.5 |
| 7 | 96.1 |
| 8 | 94.7 |
| 9 | 96.7 |
| 10 | 97.2 |

**Mean: 95.9**
**Standard Deviation: 0.875**

**Confusion Matrix:**

| | **2 (benign)** | **4 (malignant)** |
|---|---|---|
| **2 (benign)** | 229 | 7 |
| **4 (malignant)** | 3 | 111 |

**5- Fold Cross Validation (k=3):**

| #Run | Mean (%) | Standard Deviation |
|---|---|---|
| 1 | 96.1 | 0.85 |
| 2 | 96.2 | 1.47 |
| 3 | 96.8 | 1.40 |
| 4 | 96.4 | 0.94 |
| 5 | 96.4 | 1.14 |
| 6 | 96.5 | 0.63 |
| 7 | 96.2 | 1.07 |
| 8 | 96.8 | 1.32 |
| 9 | 96.1 | 1.95 |
| 10 | 96.5 | 1.01 |

**Grand Mean -> 96.45**

**Confusion Matrix:**

| | **2 (benign)** | **4 (malignant)** |
|---|---|---|
| **2 (benign)** | 233 | 3 |
| **4 (malignant)** | 8 | 106 |

**Random Subsampling Approach (k=3):**

| #Run | Accuracy |
|---|---|
| 1 | 96.9 |
| 2 | 96.7 |
| 3 | 96.4 |
| 4 | 96.7 |
| 5 | 96.9 |
| 6 | 95.8 |
| 7 | 96.7 |
| 8 | 97.5 |
| 9 | 97.5 |
| 10 | 96.7 |

**Mean: 96.8**
**Standard Deviation: 0.482**

**Confusion Matrix:**

| | **2 (benign)** | **4 (malignant)** |
|---|---|---|
| **2 (benign)** | 215 | 5 |
| **4 (malignant)** | 7 | 123 |

**Data Set : Wine recognition data**

Number of Instances
- class 1 59
- class 2 71
- class 3 48

Number of Attributes 13

1st attribute is class identifier (1-3)

**5- Fold Cross Validation (k=1):**

| #Run | Mean (%) | Standard Deviation |
|---|---|---|
| 1 | 76.4 | 4.83 |
| 2 | 74.1 | 7.45 |
| 3 | 73.5 | 2.10 |
| 4 | 77.5 | 6.05 |
| 5 | 75.8 | 6.15 |
| 6 | 77.5 | 5.78 |
| 7 | 72.4 | 10.11 |
| 8 | 75.2 | 5.99 |
| 9 | 71.9 | 2.47 |
| 10 | 74.1 | 8.06 |

**Grand Mean -> 74.88**

**Confusion Matrix:**

| | **1** | **2** | **3** |
|---|---|---|---|
| **Iris-setosa** | 26 | 0 | 0 |
| **Iris-versicolor** | 1 | 26 | 12 |
| **Iris-virginica** | 1 | 7 | 10 |

**Random Subsampling Approach (k=1):**

| **#Run** | **Accuracy** |
|---|---|
| 1 | 69.6 |
| 2 | 71.9 |
| 3 | 66.2 |
| 4 | 65.1 |
| 5 | 76.4 |
| 6 | 73.0 |
| 7 | 64.0 |
| 8 | 75.2 |
| 9 | 73.0 |
| 10 | 66.2 |

**Mean: 70.11**
**Standard Deviation: 4.210**

**5- Fold Cross Validation (k=3):**

| #Run | Mean (%) | Standard Deviation |
|---|---|---|
| 1 | 71.3 | 3.81 |
| 2 | 71.3 | 8.98 |
| 3 | 69.6 | 6.25 |
| 4 | 68.5 | 9.82 |
| 5 | 71.9 | 5.20 |
| 6 | 69.1 | 6.30 |
| 7 | 70.7 | 4.12 |
| 8 | 69.1 | 6.30 |
| 9 | 71.3 | 3.81 |
| 10 | 71.9 | 3.81 |

**Grand Mean -> 70.505**

**Confusion Matrix:**

| | **1** | **2** | **3** |
|---|---|---|---|
| **1** | 30 | 0 | 2 |
| **2** | 6 | 24 | 9 |
| **3** | 2 | 6 | 10 |

**Random Subsampling Approach (k=3):**

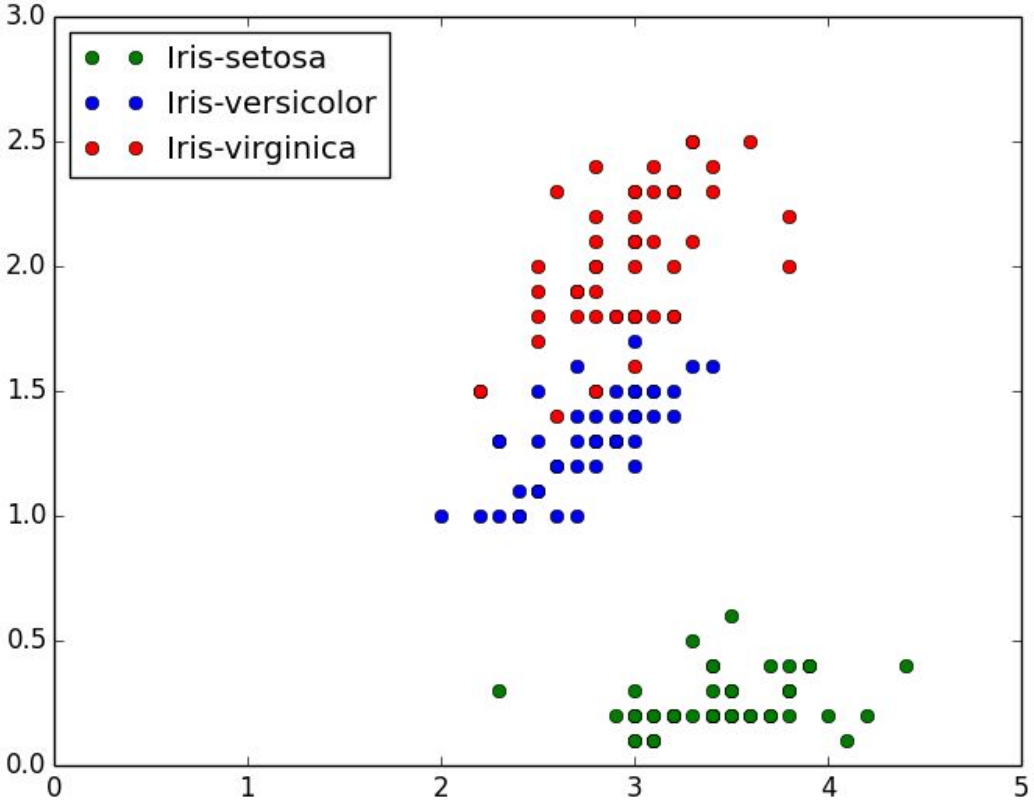
| #Run | Accuracy |
|---|---|
| 1 | 71.9 |
| 2 | 70.7 |
| 3 | 68.5 |
| 4 | 73.0 |
| 5 | 69.6 |
| 6 | 68.5 |
| 7 | 66.2 |
| 8 | 74.1 |
| 9 | 62.9 |
| 10 | 69.6 |

**Mean: 69.550**
**Standard Deviation: 3.115**

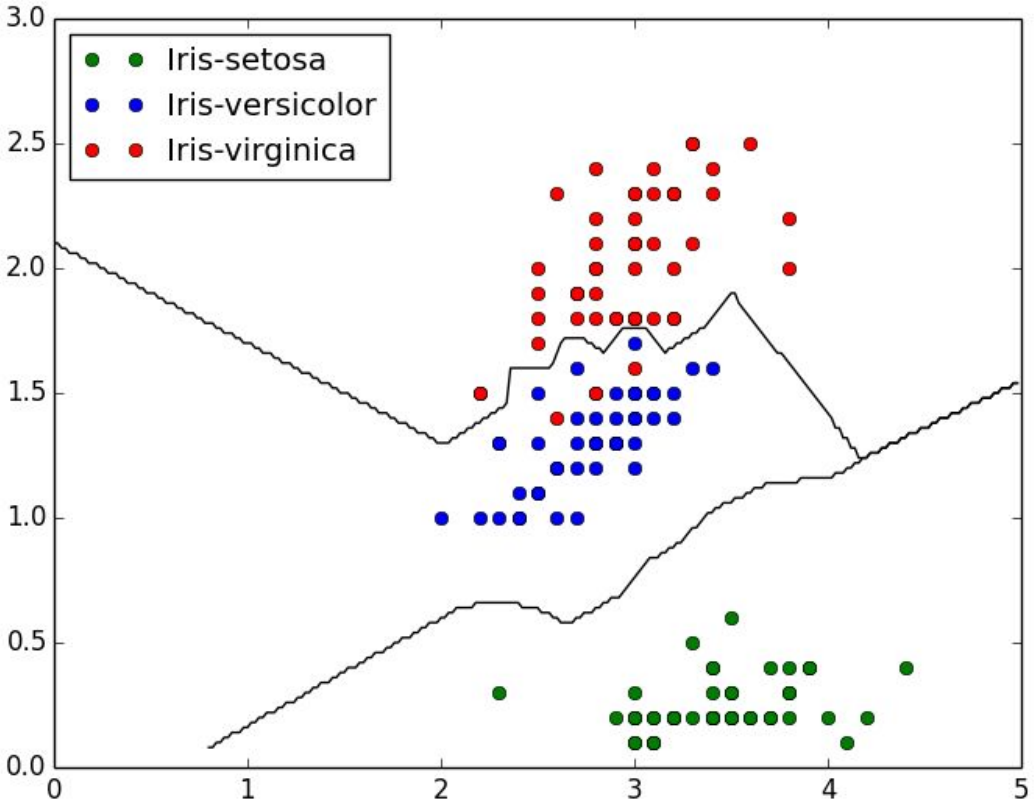
**Confusion Matrix:**

| | **1** | **2** | **3** |
|---|---|---|---|
| **1** | 29 | 4 | 4 |
| **2** | 0 | 24 | 7 |
| **3** | 0 | 8 | 13 |

**Plot of the Iris Dataset (using sepal width as x and petal width as y)**



**Plot of decision boundary for the above using 1-nearest neighbour classifier**




**Some Conclusions from the results:**

- As my code is generalized to any value of ‘k’, which is taken as an arguement, when comparing k=1 and k=3, it is clearly giving a better accuracy. Reason: More information (neighbours) is considered before classifying the current data unit. Results show the accuracy more or less staying the same with increasing the value of ‘k’.
- But i believe increasing the value of ‘k’ might also decrease the accuracy after a certain point if the data is too sparse and as ‘k’ increases, weightage of data points of a different class might get more weightage thereby giving a “false” classification.

**Selecting Data Set:**
Data Sets, which are simple in getting the data so that good amount of time can be spent understanding the working of algorithms rather than extracting data. Also to maintain diversity and to check the working of algorithms on data of different size, one large (cancer) and 2 medium sized data sets are chosen.

*** Will the decision boundary of a 3-NN classifier be piecewise linear?**

The answer is “YES”.

Lets consider a 1-NN classifier to get some better understanding. Since at any instance only 1 closest neighbour is considered to classify a “data point”, there is a really high possibility of getting a highly zig-zag/non-smooth graph (assuming data is random nd dense). Cause there is effect only from one point at the back or front determining its classification, making it highly probable to move in any direction.

But in case of a 3-NN classifier, decision of classifying a ‘data point’ is determined by 3 closest neighbours, thereby a drastic change can only occur if all 3 closest neighbours move away from current classification, thereby creating a sudden turn in the graph. Else, any drastic change forced by one neighbour will be lessened by the other two or vice versa keeping the graph “piecewise” linear.

## Code:

```
#!/usr/bin/python

import sys
import operator
import math
import random

if len(sys.argv)!=4:
        print "invalid no of args"
        sys.exit()

myData=[]
fh=open(sys.argv[1],'r')
kVal=int(sys.argv[2]) #k value for kth nearest neighbour
classIndex=int(sys.argv[3])
```

```
for i in fh:
	if len(i)!=1:
		i=i.strip()
		row=i.split(',')
		myData.append(row)


classList=[]
for i in myData:
	if i[classIndex] not in classList:
		classList.append(i[classIndex])


print "ClassList: "
print classList


meanList=[] #mean list
print "\n###  5-Fold Cross Validation  ###"

for count in range(10):
	random.shuffle(myData)

	accuracyList=[]
	for fold in range(5):
		trainSet=[]
		testSet=[]

		size=len(myData)
		dim=len(myData[0])
		#print size

		i=0
		while i<size:
			if i%5==fold:
				testSet.append(myData[i])
			else:
				trainSet.append(myData[i])
			i=i+1

		#print len(trainSet)
		#print len(testSet)
```

```
		#iterate over the testData nd use k to test get accuracy
		#print len(trainSet)
		#print len(testSet)
		matchCount=0
		for row1 in testSet:
			dList=[]
			for row2 in trainSet:
				temp=[]
				dist=0
				for k in range(dim):
					if ((sys.argv[1]!='breast-cancer-wisconsin.data' and
k!=classIndex) or (sys.argv[1]=='breast-cancer-wisconsin.data' and k!=classIndex and  k!=0)):
						if k==0:
							print row1[k],row2[k]
						val1=row1[k]
						val2=row2[k]

						#handle unavailability in cancer.data, give average
						if val1=='?':
							val1=5
						if val2=='?':
							val2=5

						dist=dist+math.pow((float(val1)-float(val2)),2)
				dist=math.sqrt(dist)
				temp.append(dist)
				temp.append(row2[classIndex]) #append class to list
				dList.append(temp)
			dList=sorted(dList)
	#		for i in dList:
	#			print i

			myDict={}
			for i in range(kVal):
				key=dList[i][1]
				if key not in myDict:
					myDict[key]=1
				else:
					myDict[key]=myDict[key]+1

			#now sort the dictionary based on the value (descending)
			myDict=sorted(myDict.items(),key=operator.itemgetter(1),reverse=True)
		#	print "\n***************"
```

```
                #       for i in myDict:
                #               print i[0],i[1]

                        #randomly assign a class,  assign class at zeroth key

                        actual=row1[classIndex]
                        prediction=myDict[0][0]

        #               print classMap
        #               print actual,prediction
                        if len(myDict)==dim: #if all r discrete, assign the one at the top of the dict
as the predicted class
                                if actual==prediction:
                                        matchCount=matchCount+1
                        else:
                                if actual==prediction:
                                        matchCount=matchCount+1


                accuracy=(float(matchCount)/(float(size)/5.0))*100
                accuracyList.append(accuracy)

        sum=0.0
        for i in accuracyList:
                sum=sum+i
        foldMean=(sum/5.0)
        meanList.append(foldMean)


        #standard deviation of the current fold
        row="Test Run #"+str(count+1)
        temp=0.0
        for i in accuracyList:
                temp=temp+math.pow(foldMean-i,2)
        foldDev=math.sqrt(temp/5.0)
        row=row+"       " + "Mean: "+str(foldMean)+ " "
        row=row+ "Standard Deviation: "+str(foldDev)+"\n"
        print row

tempSum=0.0
for i in meanList:
        tempSum=tempSum+i
print "Grand Mean: "+str(tempSum/10.0)
```

```
print "\n###   Random Subsampling Approach  ###"
accuList=[]
for count in range(10):
	random.shuffle(myData)

	trainSet=[]
	testSet=[]

	size=len(myData)
	dim=len(myData[0])
	#print size

	i=0
	while i<size/2:
		trainSet.append(myData[i])
		i=i+1

	while i<size:
		testSet.append(myData[i])
		i=i+1

	#print len(trainSet)
	#print len(testSet)

	#iterate over the testData nd use k to test get accuracy
	#confusion Matrix
	cMatrix={}


	matchCount=0
	for row1 in testSet:
		dList=[]
		for row2 in trainSet:
			temp=[]
			dist=0
			for k in range(dim):
				if ((sys.argv[1]!='breast-cancer-wisconsin.data' and k!=classIndex)
or (sys.argv[1]=='breast-cancer-wisconsin.data' and k!=classIndex and  k!=0)):
```

```
                                        val1=row1[k]
                                        val2=row2[k]

                                        #handle unavailability in cancer.data, give average
                                        if val1=='?':
                                                val1=5
                                        if val2=='?':
                                                val2=5

                                        dist=dist+math.pow((float(val1)-float(val2)),2)
                        dist=math.sqrt(dist)
                        temp.append(dist)
                        temp.append(row2[classIndex]) #append class to list
                        dList.append(temp)
                dList=sorted(dList)
#               for i in dList:
#                       print i

                myDict={}
                for i in range(kVal):
                        key=dList[i][1]
                        if key not in myDict:
                                myDict[key]=1
                        else:
                                myDict[key]=myDict[key]+1

                #now sort the dictionary based on the value (descending)
                myDict=sorted(myDict.items(),key=operator.itemgetter(1),reverse=True)
        #       print "\n***************"
        #       for i in myDict:
        #               print i[0],i[1]

                #randomly assign a class,  assign class at zeroth key

                actual=row1[classIndex]
                prediction=myDict[0][0]

#               print classMap
#               print actual,prediction
                if len(myDict)==dim: #if all r discrete, assign the one at the top of the dict as the
predicted class
                        if actual==prediction:
                                matchCount=matchCount+1
```

```
                else:
                        if actual==prediction:
                                matchCount=matchCount+1

                #update confusion matrix
                if actual in cMatrix:
                        if prediction in cMatrix[actual]:
                                cMatrix[actual][prediction]=cMatrix[actual][prediction]+1
                        else:
                                cMatrix[actual][prediction]=1
                else:
                        cMatrix[actual]={}
                        cMatrix[actual][prediction]=1



        accuracy=(float(matchCount)/(float(size)/2.0))*100
        print '*****************'
        print "Test Run: "+str(count+1)

        print 'Accuracy: ',accuracy
        accuList.append(accuracy)

#       print cMatrix
        #print dictionary
        print "\nConfusion Matrix"

        header=""
        for i in classList:
                header+="\t\t"+i

        print header
        for i in classList:
                col=cMatrix[i]
                row=i
                for j in classList:
                        if col.has_key(j):
                                row+="\t\t"+str(col[j])
                        else:
                                row+="\t\t"+"0"

                print row
        print '*****************\n'
```

```
#calculate mean
sum=0.0
for i in  accuList:
        sum=sum+i
mean=sum/10.0
print "Mean: ",
print mean

#calculate standard deviation
temp=0.0
for i in accuList:
        temp=temp+math.pow(mean-i,2)
sd=math.sqrt(temp/10)
print "Standard Deviation: ",
print sd
```

## Code for plotting decision boundaries of the 1-NN classifier:

```
#!/usr/bin/python

import math
import sys
import matplotlib.pyplot as plt
import numpy as np

fh=open('iris.data','r')

myData=[]
for i in fh:
        if len(i)!=1:
                i=i.strip()
                row=i.split(',')
                myData.append(row)

trainData={}
for row in myData:
        key=(float(row[1]),float(row[3]))
        trainData[key]=row[4]
```

```
setx=[]
sety=[]
virx=[]
viry=[]
verx=[]
very=[]

for row in myData:
	if row[4]=='Iris-setosa':
		setx.append(row[1])
		sety.append(row[3])
	elif row[4]=='Iris-versicolor':
		verx.append(row[1])
		very.append(row[3])
	elif row[4]=='Iris-virginica':
		virx.append(row[1])
		viry.append(row[3])

prec = 0.02
xCord=np.arange(0,5.1,prec)
yCord=np.arange(0,3.1,prec)

testData={}
for i in xCord:
	for j in yCord:
		newi=float('%.2f' %(i))
		newj=float('%.2f' %(j))
		testData[(newi,newj)]='?'


for row1 in testData.keys():
	minDist=10000
	tClass=''
	for row2 in trainData.keys():
		dist=math.pow((row2[0]-row1[0]),2)+math.pow((row2[1]-row1[1]),2)
		dist=math.sqrt(dist)
		if dist<minDist:
			minDist=dist
			tClass=trainData[row2]
	testData[row1]=tClass


setosax=[]
```

```
setosay=[]
virginicax=[]
virginicay=[]
versix=[]
versiy=[]

for row in testData.keys():
	if testData[row]=='Iris-setosa':
		setosax.append(row[0])
		setosay.append(row[1])
	elif testData[row]=='Iris-versicolor':
		versix.append(row[0])
		versiy.append(row[1])
	elif testData[row]=='Iris-virginica':
		virginicax.append(row[0])
		virginicay.append(row[1])

line1=[]
for x in np.arange(0,5,prec):
	for y in np.arange(3,0,-prec):
		newx=float('%.2f' %(x))
		newy=float('%.2f' %(y))
		if testData[(newx,newy)]!=testData[(newx,float('%.2f' %(newy-prec)))]:
			line1.append((newx,newy))
			break

line1x=[]
line1y=[]
for i in line1:
	line1x.append(i[0])
	line1y.append(i[1])

line2=[]
for x in np.arange(0.8,5,prec):
	for y in np.arange(0,2.9,prec):
		newx=float('%.2f' %(x))
		newy=float('%.2f' %(y))
		if testData[(newx,newy)]!=testData[(newx,float('%.2f' %(newy+prec)))]:
			line2.append((newx,float('%.2f' %(newy+prec))))
			break

line2x=[]
line2y=[]
```

```
for i in line2:
        line2x.append(i[0])
        line2y.append(i[1])


plt.plot(setx,sety,'go',label='Iris-setosa')
plt.plot(verx,very,'bo',label='Iris-versicolor')
plt.plot(virx,viry,'ro',label='Iris-virginica')
plt.axis([0,5,0,3])
plt.plot(line1x,line1y,'black')
plt.plot(line2x,line2y,'black')
plt.legend(loc=2)
plt.show()
```