CSE 151: Programming Assignment #2

Due on Monday, April 18, 2016

Mangione-Tran,Carmine 9AM

Kyle Lee, Jaehee Park

A01614951, A11287366

Homework Code

```
import random
import csv
import math
import operator
from decimal import *
import numpy as np
import matplotlib.pyplot as plt


#method for generating hits
#@param:x => our data
#       :testSize => hit rate
#       :genList => list of # of hits per index
#@return:genList => updated list with hits per index
def count (x, testSize, genList):
    size = len(x)
    expectedDraws = int(round(size*testSize))
    x1 = Decimal(expectedDraws)/Decimal(size)

    j = 0

    for i in range(0, size):
        x2 = random.uniform(0,1)
        if x2 < x1:
            genList[i] = genList[i] + 1
            j = j+1
        x1 = Decimal(expectedDraws-j)/Decimal(size-i)
    return genList

# Performs z-scaling on matrix
# @param: matrix - The matrix/array to scale
# @return: matrix - The matrix that has been scaled by the mean of each
#                    column and the standard deviation of each column
def zscale(matrix, means,stdev):
    offset = len(means)-2
    matrix[:,0:offset] = (matrix[:,0:offset] - means[0:offset])/(stdev[0:offset])
    return matrix

# Calculate the distance between two instances of data
# @param: data1 - First data to compare to
# @param: data2 - Second data to compare to
# @param: length - Size of data
def calculateDistance(data1, data2, length):
    distance = 0
    for x in range(length):
        distance+=pow((data1[x] - data2[x]),2)
    return math.sqrt(distance)

# Find the nearest neighbor
# @param: trainingSet - The set to be trained
# @param: testInstance - The test item to be categorized
```

```
# @param: k - the k-nearest neighbors
def findNeighbor(training, testI,k):
    d = []
    length = len(testI)-1

    for x in range(len(training)):
        dist = calculateDistance(testI, training[x], length)
        d.append((training[x], dist))

    d.sort(key=operator.itemgetter(1))
    k_neighbors = []

    for x in range(k):
        k_neighbors.append(d[x][0])
    return k_neighbors

# Voting process
# @param: neighbors - neighbors already chosen
# @return: sortedVotes - the nearest neighbor
def getResponse(k_neighbors):
     votes = {}
     for x in range(len(k_neighbors)):
          est = k_neighbors[x][-1]

          if est in votes:
               votes[est] += 1
          else:
               votes[est] = 1

     sVotes = sorted(votes.items(), key=operator.itemgetter(1), reverse=True)
     return sVotes[0][0]


# Create a confusion matrix
# @param: predicted - Predictions in array
# @param: actual - Actual values
# @param: size - Number of classes
def confuseMe(predicted, actual,size):
    # Initialize confusion matrix
    confusionMatrix = [[0 for x in range(size)] for x in range(size)]
    confusionMatrix = np.array(confusionMatrix).astype(np.float)
    # Count true positives and false positives
    for i in range(len(predicted)):
        # If actual is equal to predicted, increase the diagonal by 1
        if actual[i] == predicted[i]:
            confusionMatrix[int(actual[i])][int(actual[i])] = confusionMatrix[int(actual[i])][int(act

        # If actual does not equal predicted, increase that respective spot by 1
        elif actual[i] is not predicted[i]:
            confusionMatrix[int(actual[i])][int(predicted[i])] = confusionMatrix[int(actual[i])][int

    return confusionMatrix
```

```
# Change a confusion matrix into a series of ratios
# @param: confusionMatrix - The Confusion Matrix to change
def decimateConfusionMatrix(confusionMatrix):
    for k in range(len(confusionMatrix)):
        weight = np.sum(confusionMatrix[k,:])
        if (weight > 0):
            confusionMatrix[k,:] = confusionMatrix[k,:]/weight
    return confusionMatrix


#### BEGIN MAIN METHOD HERE ####

# Read CSV Files
inputFile = open('abalone.data')
inputReader = csv.reader(inputFile)
inputData = list(inputReader)   #inputData = list of our data (which is in lists)


# Initiaize test Size
trainingSize = 0.9

# Length of the data
size = len(inputData)

# Set a random seed
random.seed(123451)

# Initialize counter
counter = [] #stores hit rates on index

# initialize counter array
for x in range(size):
    counter.append(0)

# Find test size
for i in range(1,2):
    counter = count(inputData, trainingSize, counter)

for i in range(size):
    if inputData[i][0] == 'M':
        inputData[i][0] = 0
    elif inputData[i][0] == 'F':
        inputData[i][0] = 1
    elif inputData[i][0] == 'I':
        inputData[i][0] = 2

# Add 3 columns for classification of sex
proxiedData = np.zeros((size,len(inputData[0])+3))
proxiedData[:,:-3] = inputData
# Fix data for Euclidean distance
for i in range(size):
    if proxiedData[i][0] == 0:
        proxiedData[i][9] = 1
```

```python
    elif proxiedData[i][0] == 1:
        proxiedData[i][10] = 1
    elif proxiedData[i][0] == 2:
        proxiedData[i][11] = 1

# Remove first column
proxiedData = proxiedData[:,1:12]

# Swap the actual to the predictions
proxiedData[:,[7,8,9,10]] = proxiedData[:,[8,9,10,7]]

# Keep track of counter and create training and testing sets
trainIndex = []
testIndex = []
trainingSet = []
testSet = []

# Create training and testing sets
for i in range(size):
    if counter[i] == 1:
        trainIndex.append(i)
        trainingSet.append(proxiedData[i])
    else:
        testIndex.append(i)
        testSet.append(proxiedData[i])


# Change to numeric arrays
trainingSet = np.array(trainingSet)
testSet = np.array(testSet)

# Calculate mean and standard deviation of the training set
means = trainingSet.mean(axis=0)
stdevs = trainingSet.std(axis=0)

# Standard normalization
proxiedData = zscale(proxiedData,means,stdevs)

predictions1 = []
predictions2 = []
predictions3 = []
predictions4 = []
predictions5 = []
############## Start k-NN clustering
### K=1 Testing ###
k=1
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions1.append(result)

# Get minimum and maximum of age to create confusion matrix
ages = set(proxiedData[:,10])
```

```
sizeOfAges = len(ages)
actualAges = testSet[:,10]
predictions1 = np.array(predictions1)

# Calculate confusion matrix
confusionMatrix1 =[]
confusionMatrix1 = confuseMe(predictions1, actualAges, sizeOfAges)

# Print accuracy
accuracy1 = sum(confusionMatrix1.diagonal())/len(actualAges)
print('Accuracy for Abalone Dataset for k = ',k,":", accuracy1)

### K=3 Testing ###
k=3
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions2.append(result)

predictions2 = np.array(predictions2)

# Calculate confusion matrix
confusionMatrix2 =[]
confusionMatrix2 = confuseMe(predictions2, actualAges, sizeOfAges)

# Print accuracy
accuracy2 = sum(confusionMatrix2.diagonal())/len(actualAges)
print('Accuracy for Abalone Dataset for k = ',k,":", accuracy2)

### K=5 Testing ###
k=5
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions3.append(result)

predictions3 = np.array(predictions3)

# Calculate confusion matrix
confusionMatrix3 =[]
confusionMatrix3 = confuseMe(predictions3, actualAges, sizeOfAges)

# Print accuracy
accuracy3 = sum(confusionMatrix3.diagonal())/len(actualAges)
print('Accuracy for Abalone Dataset for k = ',k,":", accuracy3)

### K=7 Testing ###
k=7
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions4.append(result)
```

```
predictions4 = np.array(predictions4)

# Calculate confusion matrix
confusionMatrix4 =[]
confusionMatrix4 = confuseMe(predictions4, actualAges, sizeOfAges)

# Print accuracy
accuracy4 = sum(confusionMatrix4.diagonal())/len(actualAges)
print('Accuracy for Abalone Dataset for k = ',k,":", accuracy4)

### K=9 Testing ###
k=9
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions5.append(result)

predictions5 = np.array(predictions5)

# Calculate confusion matrix
confusionMatrix5 =[]
confusionMatrix5 = confuseMe(predictions5, actualAges, sizeOfAges)

# Print accuracy
accuracy5 = sum(confusionMatrix5.diagonal())/len(actualAges)
print('Accuracy for Abalone Dataset for k = ',k,":", accuracy5)

# We observe that we will have k=9 be the best choice
print(confusionMatrix4)


########## Testing 10% Categorization
from numpy import genfromtxt
inputData2 = genfromtxt(r'C:\Users\Kyle Lee\Google Drive\School Doc. 2015-2016\CSE 151\Programming Ho
inputData2 = np.array(inputData2)
# Length of the data

size = len(inputData2)

# Initialize counter
counter = [] #stores hit rates on index

# initialize counter array
for x in range(size):
    counter.append(0)

# Find test size
for i in range(1,2):
    counter = count(inputData2, trainingSize, counter)

# Keep track of counter and create training and testing sets
trainingSet = []
testSet = []
```

```
# Create training and testing sets
for i in range(size):
    if counter[i] == 1:
        trainingSet.append(inputData2[i])
    else:
        testSet.append(inputData2[i])

# Change to numeric arrays
trainingSet = np.array(trainingSet)
testSet = np.array(testSet)

# Calculate mean and standard deviation of the training set
means = trainingSet.mean(axis=0)
stdevs = trainingSet.std(axis=0)

# Standard normalization
inputData2 = zscale(inputData2,means,stdevs)

predictions1 = []
predictions2 = []
predictions3 = []
predictions4 = []
predictions5 = []
############## Start k-NN clustering
### K=1 Testing ###
k=1
inputData2 = np.array(inputData2)
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions1.append(result)

# Get minimum and maximum of age to create confusion matrix
ages = set(inputData2[:,9])
n = len(ages)
actualAges = testSet[:,9]
predictions1 = np.array(predictions1)

# Calculate confusion matrix
confusionMatrix1 =[]
confusionMatrix1 = confuseMe(predictions1, actualAges, n)

# Print accuracy
accuracy1 = sum(confusionMatrix1.diagonal())/len(actualAges)
print('Accuracy for 10% Dataset for k = ',k,":", accuracy1)

### K=3 Testing ###
k=3
inputData2 = np.array(inputData2)
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
```

```
    predictions2.append(result)

# Get predictions
predictions2 = np.array(predictions2)

# Calculate confusion matrix
confusionMatrix2 =[]
confusionMatrix2 = confuseMe(predictions2, actualAges, n)

# Print accuracy
accuracy2 = sum(confusionMatrix2.diagonal())/len(actualAges)
print('Accuracy for 10% Dataset for k = ',k,":", accuracy2)

### K=5 Testing ###
k=5
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions3.append(result)

# Get minimum and maximum of age to create confusion matrix
ages = set(inputData2[:,9])
n = len(ages)
actualAges = testSet[:,9]
predictions3 = np.array(predictions3)

# Calculate confusion matrix
confusionMatrix3 =[]
confusionMatrix3 = confuseMe(predictions3, actualAges, n)

# Print accuracy
accuracy3 = sum(confusionMatrix3.diagonal())/len(actualAges)
print('Accuracy for 10% Dataset for k = ',k,":", accuracy3)


### K=7 Testing ###
k=7
inputData2 = np.array(inputData2)
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions4.append(result)

# Get minimum and maximum of age to create confusion matrix
ages = set(inputData2[:,9])
n = len(ages)
actualAges = testSet[:,9]
predictions4 = np.array(predictions4)

# Calculate confusion matrix
confusionMatrix4 =[]
confusionMatrix4 = confuseMe(predictions4, actualAges, n)
```

```
# Print accuracy
accuracy4 = sum(confusionMatrix4.diagonal())/len(actualAges)
print('Accuracy for 10% Dataset for k = ',k,":", accuracy4)

### K=9 Testing ###
k=9
inputData2 = np.array(inputData2)
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions5.append(result)

# Get minimum and maximum of age to create confusion matrix
ages = set(inputData2[:,9])
n = len(ages)
actualAges = testSet[:,9]
predictions5 = np.array(predictions5)

# Calculate confusion matrix
confusionMatrix5 =[]
confusionMatrix5 = confuseMe(predictions5, actualAges, n)

# Print accuracy
accuracy5 = sum(confusionMatrix5.diagonal())/len(actualAges)
print('Accuracy for 10% Dataset for k = ',k,":", accuracy5)

# Print k=9 confusion matrix since we know the best confusion matrix
print(confusionMatrix5)

######################## Testing 3% Categorization
from numpy import genfromtxt
inputData2 = genfromtxt(r'C:\Users\Kyle Lee\Google Drive\School Doc. 2015-2016\CSE 151\Programming Ho
inputData2 = np.array(inputData2)
# Length of the data

size = len(inputData2)

# Initialize counter
counter = [] #stores hit rates on index

# initialize counter array
for x in range(size):
    counter.append(0)

# Find test size
for i in range(1,2):
    counter = count(inputData2, trainingSize, counter)

# Keep track of counter and create training and testing sets
trainingSet = []
testSet = []

# Create training and testing sets
```

```python
for i in range(size):
    if counter[i] == 1:
        trainingSet.append(inputData2[i])
    else:
        testSet.append(inputData2[i])

# Change to numeric arrays
trainingSet = np.array(trainingSet)
testSet = np.array(testSet)

# Calculate mean and standard deviation of the training set
means = trainingSet.mean(axis=0)
stdevs = trainingSet.std(axis=0)

# Standard normalization
inputData2 = zscale(inputData2,means,stdevs)

predictions1 = []
predictions2 = []
predictions3 = []
predictions4 = []
predictions5 = []
############## Start k-NN clustering
### K=1 Testing ###
k=1
inputData2 = np.array(inputData2)
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions1.append(result)

# Get minimum and maximum of age to create confusion matrix
ages = set(inputData2[:,9])
n = len(ages)
actualAges = testSet[:,9]
predictions1 = np.array(predictions1)

# Calculate confusion matrix
confusionMatrix1 =[]
confusionMatrix1 = confuseMe(predictions1, actualAges, n)

# Print accuracy
accuracy1 = sum(confusionMatrix1.diagonal())/len(actualAges)
print('Accuracy for 3% Dataset for k = ',k,":", accuracy1)

### K=3 Testing ###
k=3
inputData2 = np.array(inputData2)
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions2.append(result)
```

```
# Get predictions
predictions2 = np.array(predictions2)

# Calculate confusion matrix
confusionMatrix2 =[]
confusionMatrix2 = confuseMe(predictions2, actualAges, n)

# Print accuracy
accuracy2 = sum(confusionMatrix2.diagonal())/len(actualAges)
print('Accuracy for 3% Dataset for k = ',k,":", accuracy2)

### K=5 Testing ###
k=5
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions3.append(result)

# Get minimum and maximum of age to create confusion matrix
ages = set(inputData2[:,9])
n = len(ages)
actualAges = testSet[:,9]
predictions3 = np.array(predictions3)

# Calculate confusion matrix
confusionMatrix3 =[]
confusionMatrix3 = confuseMe(predictions3, actualAges, n)

# Print accuracy
accuracy3 = sum(confusionMatrix3.diagonal())/len(actualAges)
print('Accuracy for 3% Dataset for k = ',k,":", accuracy3)


### K=7 Testing ###
k=7
inputData2 = np.array(inputData2)
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions4.append(result)

# Get minimum and maximum of age to create confusion matrix
ages = set(inputData2[:,9])
n = len(ages)
actualAges = testSet[:,9]
predictions4 = np.array(predictions4)

# Calculate confusion matrix
confusionMatrix4 =[]
confusionMatrix4 = confuseMe(predictions4, actualAges, n)

# Print accuracy
accuracy4 = sum(confusionMatrix4.diagonal())/len(actualAges)
```

```
print('Accuracy for 3% Dataset for k = ',k,":", accuracy4)

### K=9 Testing ###
k=9
inputData2 = np.array(inputData2)
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions5.append(result)

# Get minimum and maximum of age to create confusion matrix
ages = set(inputData2[:,9])
n = len(ages)
actualAges = testSet[:,9]
predictions5 = np.array(predictions5)

# Calculate confusion matrix
confusionMatrix5 =[]
confusionMatrix5 = confuseMe(predictions5, actualAges, n)

# Print accuracy
accuracy5 = sum(confusionMatrix5.diagonal())/len(actualAges)
print('Accuracy for 3% Dataset for k = ',k,":", accuracy5)

# Print k=9 confusion matrix
print(confusionMatrix5)


######################## Testing separable Categorization
from numpy import genfromtxt
inputData2 = genfromtxt(r'C:\Users\Kyle Lee\Google Drive\School Doc. 2015-2016\CSE 151\Programming Ho
inputData2 = np.array(inputData2)
# Length of the data

size = len(inputData2)

# Initialize counter
counter = [] #stores hit rates on index

# initialize counter array
for x in range(size):
    counter.append(0)

# Find test size
for i in range(1,2):
    counter = count(inputData2, trainingSize, counter)

# Keep track of counter and create training and testing sets
trainingSet = []
testSet = []

# Create training and testing sets
for i in range(size):
```

```
    if counter[i] == 1:
        trainingSet.append(inputData2[i])
    else:
        testSet.append(inputData2[i])

# Change to numeric arrays
trainingSet = np.array(trainingSet)
testSet = np.array(testSet)

# Calculate mean and standard deviation of the training set
means = trainingSet.mean(axis=0)
stdevs = trainingSet.std(axis=0)

# Standard normalization
inputData2 = zscale(inputData2,means,stdevs)

predictions1 = []
predictions2 = []
predictions3 = []
predictions4 = []
predictions5 = []
############## Start k-NN clustering
### K=1 Testing ###
k=1
inputData2 = np.array(inputData2)
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions1.append(result)

# Get minimum and maximum of age to create confusion matrix
ages = set(inputData2[:,9])
n = len(ages)
actualAges = testSet[:,9]
predictions1 = np.array(predictions1)

# Calculate confusion matrix
confusionMatrix1 =[]
confusionMatrix1 = confuseMe(predictions1, actualAges, n)

# Print accuracy
accuracy1 = sum(confusionMatrix1.diagonal())/len(actualAges)
print('Accuracy for separable Dataset for k = ',k,":", accuracy1)

### K=3 Testing ###
k=3
inputData2 = np.array(inputData2)
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions2.append(result)

# Get predictions
```

```
predictions2 = np.array(predictions2)

# Calculate confusion matrix
confusionMatrix2 =[]
confusionMatrix2 = confuseMe(predictions2, actualAges, n)

# Print accuracy
accuracy2 = sum(confusionMatrix2.diagonal())/len(actualAges)
print('Accuracy for separable Dataset for k = ',k,":", accuracy2)

### K=5 Testing ###
k=5
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions3.append(result)

# Get minimum and maximum of age to create confusion matrix
ages = set(inputData2[:,9])
n = len(ages)
actualAges = testSet[:,9]
predictions3 = np.array(predictions3)

# Calculate confusion matrix
confusionMatrix3 =[]
confusionMatrix3 = confuseMe(predictions3, actualAges, n)

# Print accuracy
accuracy3 = sum(confusionMatrix3.diagonal())/len(actualAges)
print('Accuracy for separable Dataset for k = ',k,":", accuracy3)


### K=7 Testing ###
k=7
inputData2 = np.array(inputData2)
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions4.append(result)

# Get minimum and maximum of age to create confusion matrix
ages = set(inputData2[:,9])
n = len(ages)
actualAges = testSet[:,9]
predictions4 = np.array(predictions4)

# Calculate confusion matrix
confusionMatrix4 =[]
confusionMatrix4 = confuseMe(predictions4, actualAges, n)

# Print accuracy
accuracy4 = sum(confusionMatrix4.diagonal())/len(actualAges)
print('Accuracy for separable Dataset for k = ',k,":", accuracy4)
```

```
### K=9 Testing ###
k=9
inputData2 = np.array(inputData2)
for x in range(len(testSet)):
    neighbors = findNeighbor(trainingSet, testSet[x],k)
    result = getResponse(neighbors)
    predictions5.append(result)

# Get minimum and maximum of age to create confusion matrix
ages = set(inputData2[:,9])
n = len(ages)
actualAges = testSet[:,9]
predictions5 = np.array(predictions5)

# Calculate confusion matrix
confusionMatrix5 =[]
confusionMatrix5 = confuseMe(predictions5, actualAges, n)

# Print accuracy
accuracy5 = sum(confusionMatrix5.diagonal())/len(actualAges)
print('Accuracy for separable Dataset for k = ',k,":", accuracy5)

# Print k=9 confusion matrix
print(confusionMatrix5)
```

Console Output

```
Accuracy for Abalone Dataset for k =  1 : 0.1861575179
Accuracy for Abalone Dataset for k =  3 : 0.219570405728
Accuracy for Abalone Dataset for k =  5 : 0.233890214797
Accuracy for Abalone Dataset for k =  7 : 0.252983293556
Accuracy for Abalone Dataset for k =  9 : 0.26968973747
[[  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   3.   0.   0.   0.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   2.   0.   4.   0.   1.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   1.   3.   3.   4.   1.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   2.  10.   7.   2.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   1.   6.  12.  13.   4.   1.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   3.   7.  17.  14.   7.   1.   0.   1.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   1.   1.   3.  16.  25.  17.  10.   1.   0.
```

```
    0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   2.  13.  22.  26.   9.   1.   2.
    0.   0.   1.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   1.   2.  14.  16.   9.   2.   1.
    1.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   1.   3.   4.   7.   5.   3.   0.
    1.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   2.   4.   4.   4.   0.   0.
    0.   0.   1.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   2.   3.   3.   1.   2.   2.
    0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   1.   1.   0.   2.   3.   0.   1.
    0.   1.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   2.   1.   1.   1.
    1.   0.   0.   2.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   1.   0.   1.
    0.   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   1.   1.   1.   0.
    0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   2.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   2.   0.   0.   0.
    0.   0.   1.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]]

Accuracy for 10% Dataset for k =  1 : 0.593406593407
Accuracy for 10% Dataset for k =  3 : 0.597402597403
Accuracy for 10% Dataset for k =  5 : 0.614385614386
Accuracy for 10% Dataset for k =  7 : 0.63036963037
Accuracy for 10% Dataset for k =  9 : 0.637362637363
[[ 255.  182.]
 [ 181.  383.]]

 Accuracy for 3% Dataset for k =  1 : 0.596403596404
Accuracy for 3% Dataset for k =  3 : 0.628371628372
Accuracy for 3% Dataset for k =  5 : 0.636363636364
Accuracy for 3% Dataset for k =  7 : 0.637362637363
Accuracy for 3% Dataset for k =  9 : 0.629370629371
[[ 262.  204.]
 [ 167.  368.]]
```

```
 Accuracy for separable Dataset for k =  1 : 0.713286713287
Accuracy for separable Dataset for k =  3 : 0.741258741259
Accuracy for separable Dataset for k =  5 : 0.754245754246
Accuracy for separable Dataset for k =  7 : 0.756243756244
Accuracy for separable Dataset for k =  9 : 0.755244755245
[[ 337.  115.]
 [ 130.  419.]]
```