Classifying with k-Nearest Neighbors (kNN) Algorithm

1. Requirements

1. In this experiment, we have two Examples and one Exercise to help you learn Supervised Learning. Follow the guides in Example 1 and Example 2 to write and execute the codes. For the Exercise, you need to do the coding by yourself.

2. Please take screenshots of the codes you wrote and the output results you obtained, then put them into your lab report. In addition, you need to clearly explain to us in your report what each block of code does and what the output results mean.

3. The lab report must be completed independently. Any similarities to the reports submitted by others will affect the final score.

2. Example 1: Improving Matches for A Dating Site with kNN

2.1 Background

My friend Hellen has been using some online dating sites to find different people to go out with. She realized that despite the site's recommendations, she didn't like everyone she was matched with. After some introspection, she realized there were three types of people she went out with 1) People she didn't like; 2) People she liked in small doses; 3) People she liked in large doses. Despite this discovery, Hellen still couldn't figure out what made a new matched person fit into any of these categories, because they were all recommended to her by the dating site. Thus, Hellen asked us to help her filter future matches and categorize them. In addition, Hellen has collected some data that isn't recorded by the dating site, but she feels it's useful in selecting people to go out with.

2.2 Data Processing

The data Hellen collected is in a text file called datingTestSet2.txt. Hellen has been collecting this data for a while and has 1,000 entries, where each entry is on each line. For each entry, the three leftmost elements represent three features of a person, and the rightmost column is the label of its category. Before we can use this data, we need to change it to the format that our classifier can accept.

Now, let's create example_1.py and then add the python packages:

```
from numpy import *
import operator
from os import listdir
```

After that, add the following function to read in and format the data:

```
def file2matrix(filename):
    fr = open(filename)
    numberOfLines = len(fr.readlines())
    returnMat = zeros((numberOfLines,3))
    classLabelVector = []
    fr = open(filename)
    index = 0
    for line in fr.readlines():
        line = line.strip()
        listFromLine = line.split('\t')
        returnMat[index,:] = listFromLine[0:3]
        classLabelVector.append(int(listFromLine[-1]))
        index += 1
    return returnMat,classLabelVector
```

This function reads in the file and counts the number of lines, and then creates a NumPy matrix to populate and return. It loops over all the lines in the file and strips off the return line character with line.strip(). Next, it splits the line into a list of elements delimited by the tab character '\t'. It takes the left three elements (i.e., the features) and shoves them into a row of the matrix, and then puts the last item (i.e., the label) into classLabelVector.

To execute this function, we need to add the following lines into example_1.py:

```
datingDataMat,datingLabels = file2matrix('datingTestSet2.txt')
print("\n Original features:")
print(datingDataMat)
print("\n Labels:")
print(datingLabels[0:20])
```

where the input is datingTestSet2.txt and the outputs are the extracted features as well as the labels. Now save example_1.py, launch a command console (i.e., CMD.exe Prompt in ANACONDA), and change the directory to where you've stored example_1.py. To run the code, we need to type python example_1.py in the console, then the features and the labels will be printed.

Please take screenshots of the codes you wrote and the output results you obtained, then put them into your lab report. In addition, you need to clearly explain to us in your report what each block of code does and what the output results mean.

For instance, in your report, you can say "function file2matrix() helps to process and format the initial sample data, where its input filename reads the initial sample files datingTestSet2.txt, and its output returnMat contains the formatted features and classLabelVector contains the formatted labels."

For these three features, the range of their values is not in the same order of magnitude (comparing the values in different columns), which will lead to different impacts on the classifier. However, we believe that these three features should be equally important to the classification. When dealing with values that lie in different ranges, it's common to

normalize them. To scale/normalize everything from 0 to 1, we need to apply the following formula:

```
newValue = (oldValue-min)/(max-min)
```

where the variables "min" and "max" are the smallest and largest values in each feature. This scaling adds some complexity to our classifier, but it can bring good results. Let's create a new function in example_1.py called autoNorm() to automatically normalize the data to values within 0 to 1:

```
def autoNorm(dataSet):
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    ranges = maxVals - minVals
    normDataSet = zeros(shape(dataSet))
    m = dataSet.shape[0]
    normDataSet = dataSet - tile(minVals, (m,1))
    normDataSet = normDataSet/tile(ranges, (m,1))
    return normDataSet, ranges, minVals
```

To execute this function and show its effectiveness, we need to add the following lines:

```
normMat, ranges, minVals = autoNorm(datingDataMat)
print("\n Normalized features:")
print(normMat)
```

Again, save example_1.py and type "python example_1.py" to run the code, then the normalized features can be obtained. We can compare the normalized features with the original features to see the differences. Please take screenshots of the codes you wrote and the output results you obtained, then put them into your lab report. In addition, you need to clearly explain to us in your report what each block of code does and what the output results mean.

2.3 Classification

Now we'll build a function to run the kNN algorithm on one piece of data. I'll first show the function in pseudocode and then in actual Python code, followed by a detailed explanation of what everything in the code does. Remember, the goal of this function is to use the kNN algorithm to classify one piece of data called inX. Pseudocode for this function would look like this:

For every point in our dataset:
calculate the distance between inX and the current point
sort the distances in increasing order
take k items with lowest distances to inX
find the majority class among these items
return the majority class as our prediction for the class of inX

The Python code called classify0() function is in the following, we need to add it into "example_1.py":

```
def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    diffMat = tile(inX, (dataSetSize,1)) - dataSet
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances**0.5
    sortedDistIndicies = distances.argsort()
    classCount={}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
    sortedClassCount = sorted(classCount.iteritems(),
     key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]
```

1 Distance calculation

Voting with lowest k distances 2

3 Sort dictionary

The function classify0() takes four inputs: the input vector to be classified called inX, the features of training samples called dataSet, a vector of labels called labels, and, finally, k, the number of nearest neighbors to use in the voting. The labels vector should have as many elements in it as there are rows in the dataSet matrix. The code first calculates the distances i.e., ①, using the Euclidian distance. Following the distance calculation, the distances are sorted from least to greatest (this is the default). Next, ② the lowest "k" distances are used to vote on the class of inX. The input k should always be a positive integer. Lastly, ③ the code takes the classCount dictionary and decomposes it into a list of tuples and then sorts the tuples by the second item in the tuple using the itemgetter method from the operator module imported in the second line of the program. This sort is done in reverse so you have largest to smallest. Finally, it can return the label of the item occurring the most frequently.

Now it's time to use the classifier to actually classify people. Add the following testing code into example_1.py:

```
def classifyPerson():
    resultList = ['not at all','in small doses', 'in large doses']
    ffMiles = float(input("\n Feature 1:"))
    percentTats = float(input("\n Feature 2:"))
    iceCream = float(input("\n Feature 3:"))
    datingDataMat,datingLabels = file2matrix('datingTestSet2.txt')
    normMat, ranges, minVals = autoNorm(datingDataMat)
    inArr = array([ffMiles, percentTats, iceCream])
    classifierResult = classify0((inArr-minVals)/ranges,normMat,datingLabels,3)
    print("\n You will probably like this person: ",resultList[classifierResult - 1])
```

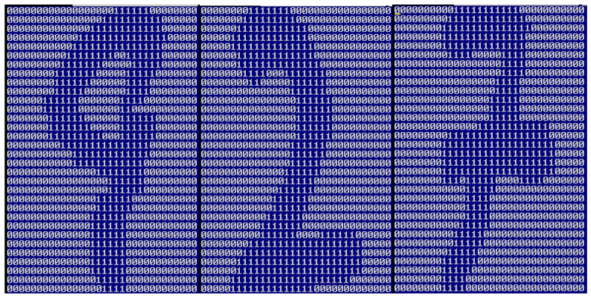
Again, save example_1.py and type "python example_1.py" to execute the code. The command console will require us to enter the values of the three features, say, Feature 1: 10000, Feature 2: 10, Feature 3: 0.5. After that, we can get the final classification result. Please take screenshots of the codes you wrote and the output results you

obtained, then put them into your lab report. In addition, you need to clearly explain to us in your report what each block of code does and what the output results mean.

3. Example 2: A Handwriting Recognition System

3.1 Background

We're going to work through an example of handwriting recognition with our kNN classifier. We'll be working only with the digits 0–9. Some samples are shown in the below figure.



3.2 Data Processing

The image samples are stored in two directories, namely trainingDigits and testDigits. The digits in the samples have already been processed through image-processing software to make them have the same size and color, i.e., 32x32 in black and white (i.e., binary characters). The trainingDigits directory contains about 2,000 samples and the testDigits directory contains about 900 samples. We'll use the trainingDigits directory to train our classifier and testDigits to test it. There'll be no overlap between the two groups. Feel free to take a look at the sample files in these folders.

At first, let's create example_2.py and then add the python packages:

```
from numpy import *
import operator
from os import listdir
```

Next, we need to reformat the images to a single vector that can be accepted by the classifier. We'll convert the 32x32 matrix in each image sample to a 1x1024 vector, which can be done by the following function called img2vector(). We need to add it into example_2.py:

```
def img2vector(filename):
    returnVect = zeros((1,1024))
    fr = open(filename)
    for i in range(32):
        lineStr = fr.readline()
        for j in range(32):
            returnVect[0,32*i+j] = int(lineStr[j])
    return returnVect
```

This function creates a 1x1024 NumPy array, then opens the given file, loops over the first 32 lines in the file, and stores the integer value of the first 32 characters on each line in the NumPy array. This array is finally returned.

We can add the following line to test the function,

```
print(img2vector("testDigits/0_13.txt")[0][0:32])
```

where the input testDigits/0_13.txt is an image sample file, and we only print the characters in its first line, i.e., [0:32]. Type "python example_2.py" in the command console to run the code. Please take screenshots of the codes you wrote and the output results you obtained, then put them into your lab report. In addition, you need to clearly explain to us in your report what each block of code does and what the output results mean.

Based on img2vector(), we can reformat all the image samples in trainingDigits with the following function:

```
def formulateTrainingData():
    hwLabels = []
    trainingFileList = listdir('trainingDigits')
    m = len(trainingFileList)
    trainingMat = zeros((m,1024))
    for i in range(m):
        fileNameStr = trainingFileList[i]
        fileStr = fileNameStr.split('.')[0]
        classNumStr = int(fileStr.split('_')[0])
        hwLabels.append(classNumStr)
        trainingMat[i,:] = img2vector('trainingDigits/%s' % fileNameStr)

    return trainingMat,hwLabels
```

where the output trainingMat contains all the samples in 1x1024 vector format, while hwLabels are extracted from the names of the sample files, which include the actual digits represented by the image samples. In the subsequent classification, trainingMat will function as features and hwLabels will be the corresponding labels, both of which will be employed by the kNN classifier as the training data. We can now add the following line to test the function

```
trainingMat,hwLabels = formulateTrainingData()
```

3.3 Classification

Now that we have processed the training data in a format that can be plugged into our classifier, so we are ready to test out the classifier and see how well it works. Before the testing, we need to introduce the classifier function, i.e., classify0(), that has been utilized in example 1.

Then, we can write a testing function handwritingClassTest() to introduce and reformat the testing data in testDigits:

```
def handwritingClassTest(trainingMat,hwLabels):

    testFileList = listdir('testDigits')
    errorCount = 0.0
    mTest = len(testFileList)
    for i in range(mTest):
        fileNameStr = testFileList[i]
        fileStr = fileNameStr.split('.')[0]
        classNumStr = int(fileStr.split('_')[0])
        vectorUnderTest = img2vector('testDigits/%s' % fileNameStr)
        classifierResult = classify0(vectorUnderTest, trainingMat, hwLabels, 3)
        if (classifierResult != classNumStr): errorCount += 1.0
    print("\n The total number of errors is: %d" % errorCount)
    print("\n The total error rate is: %f" % (errorCount/float(mTest)))
```

where the training data is imported as well. Finally, the classification can be executed via:

```
handwritingClassTest(trainingMat,hwLabels)
```

by typing "python example_2.py" in the command console. After the code is run, the number of error classification and the total error rate will be printed by the console. Please take screenshots of the codes you wrote and the output results you obtained, then put them into your lab report. In addition, you need to clearly explain to us in your report what each block of code does and what the output results mean.

4. Exercise

Now I will give you an exercise to test whether you have learned to use kNN classification to solve real problems.

In the following table, there are some four-dimensional points where the coordinates and labels of P1-P7 are given, while P8 and P9 are only given the coordinates and their labels are missing.

Point	Coordinate	Label
P1	(10, 10, 12, 7.4)	'A'
P2	(15, 20, 5, 59)	'A'
P3	(30, 40, 71, 10)	'B'

P4	(50, 70, 63, 27)	'B'
P5	(35, 50, 50, 65)	'B'
P6	(45, 50, 24, 99)	'C'
P7	(35, 45, 32, 4)	'C'
P8	(18, 23, 46, 62)	
P9	(41, 24, 87, 77)	

Task 1: Please employ function classify0() with k=3 (this function is introduced in Example 1) and write a Python program to get the labels of P8 and P9.

Task 2: If use classify0() with k=5, will their labels be different? Please change the code and give the results.

Please take screenshots of the code you wrote and the output results you obtained, and put them in your lab report. In addition, you need to clearly explain to us in your report what each block of code does and what the output results mean.