Classifying with k-Nearest Neighbors (kNN)

葉建華

jhyeh@mail.au.edu.tw

http://jhyeh.csie.au.edu.tw/

Movie Genre?

Movie Genre

- Definition of genre, how?
- Movies in one genre are similar base on…?
- “Die Hard” is similar to “Mission Impossible” but dissimilar to “Life of Pi” or “American Pie”, why?
  - Measure kisses, kicks, other things?
- The key: count and distance

k-Nearest Neighbor, kNN

- Count kisses and kicks

Table 2.1 **Movies with the number of kicks and number of kisses shown for each movie, along with our assessment of the movie type**

Movie title	# of kicks	# of kisses	Type of movie
California Man	3	104	Romance
He's Not Really into Dudes	2	100	Romance
Beautiful Woman	1	81	Romance
Kevin Longblade	101	10	Action
Robo Slayer 3000	99	5	Action
Amped II	98	2	Action
?	18	90	Unknown

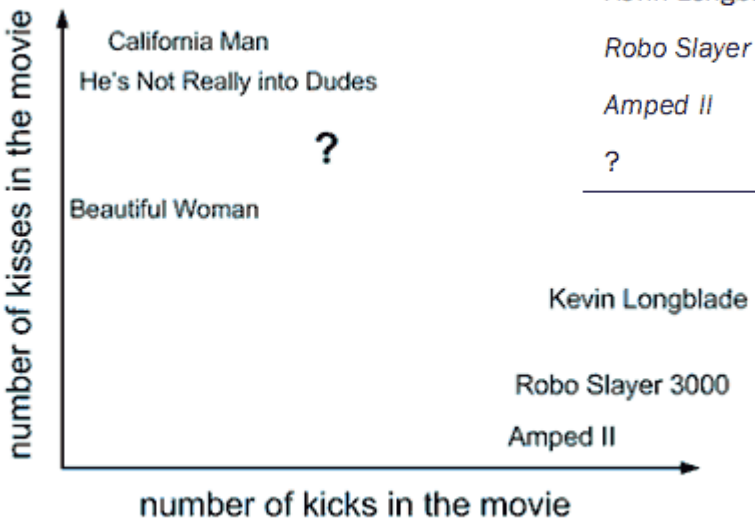

Figure 2.1 **Classifying movies by plotting the number of kicks and kisses in each movie**

Distance Measure

- Now, find k-nearest movies by sorting the distances in decreasing order

Movie title	Distance to movie "?"
California Man	20.5
He's Not Really into Dudes	18.7
Beautiful Woman	19.2
Kevin Longblade	115.3
Robo Slayer 3000	117.4
Amped II	118.9

Table 2.2 **Distances between each movie and the unknown movie**

- Assume k=3, then?
  - Get majority vote!

General Approach

General approach to kNN

1. Collect: Any method.
2. Prepare: Numeric values are needed for a distance calculation. A structured data format is best.
3. Analyze: Any method.
4. Train: Does not apply to the kNN algorithm.
5. Test: Calculate the error rate.
6. Use: This application needs to get some input data and output structured numeric values. Next, the application runs the kNN algorithm on this input data and determines which class the input data should belong to. The application then takes some action on the calculated class.

Importing Data

```
def createDataSet():
    group = array([[1.0,1.1],[1.0,1.0],[0,0],[0,0.1]])
    labels = ['A','A','B','B']
    return group, labels
```

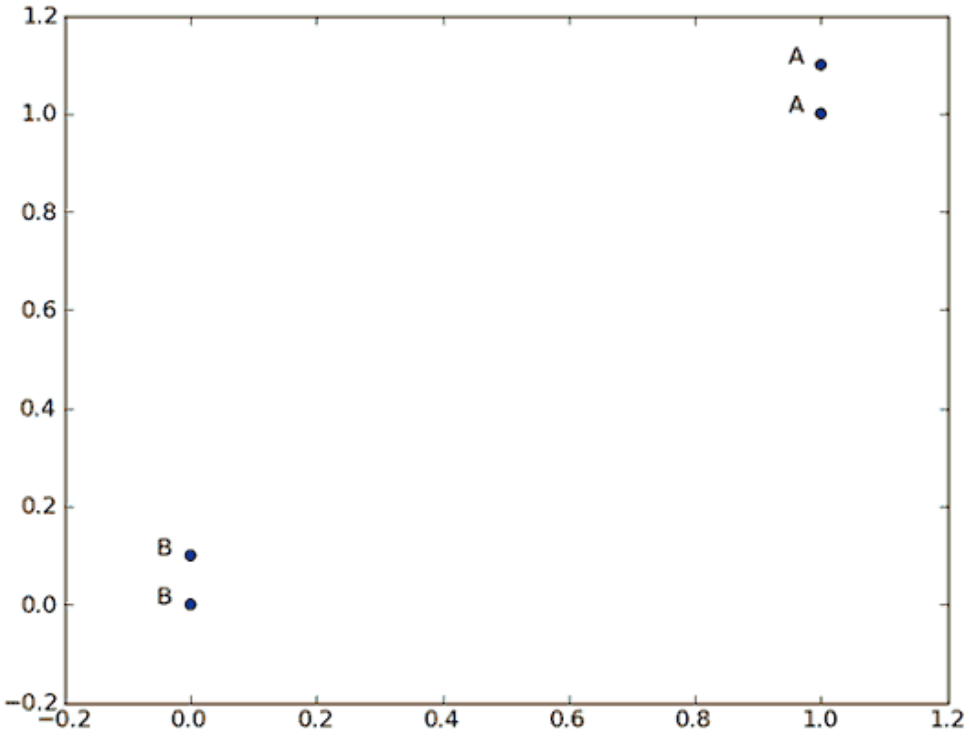

Figure 2.2 The four data points of our very simple kNN example

Pseudo Code for kNN

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

Python Code for kNN

Listing 2.1 k-Nearest Neighbors algorithm

```
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

$$d = \sqrt{(xA_0 - xB_0)^2 + (xA_1 - xB_1)^2}$$

Is It Always Right?

- Is it always right?
- At what point does this break?
- Error rate: (# of wrong classification)/(# of tests)
  - 0: perfect classifier, 1.0: always wrong

Improving Matches with kNN

- The dating site example

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:

- People she didn't like
- People she liked in small doses
- People she liked in large doses

After discovering this, Hellen couldn't figure out what made a person fit into any of these categories. They all were recommended to her by the dating site. The people whom she liked in small doses were good to see Monday through Friday, but on the weekend she'd rather spend time with the people she liked in large doses.

Processing Flow

Example: using kNN on results from a dating site

1. Collect: Text file provided.
2. Prepare: Parse a text file in Python.
3. Analyze: Use Matplotlib to make 2D plots of our data.
4. Train: Doesn't apply to the kNN algorithm.
5. Test: Write a function to use some portion of the data Hellen gave us as test examples. The test examples are classified against the non-test examples. If the predicted class doesn't match the real class, we'll count that as an error.
6. Use: Build a simple command-line program Hellen can use to predict whether she'll like someone based on a few inputs.

Collect and Prepare

The data Hellen collected is in a text file called datingTestSet.txt. Hellen has been collecting this data for a while and has 1,000 entries. A new sample is on each line, and Hellen has recorded the following features:

- Number of frequent flyer miles earned per year
- Percentage of time spent playing video games
- Liters of ice cream consumed per week

```
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
```

1 Get number of lines in file

2 Create NumPy matrix to return

3 Parse line to a list

Analyze

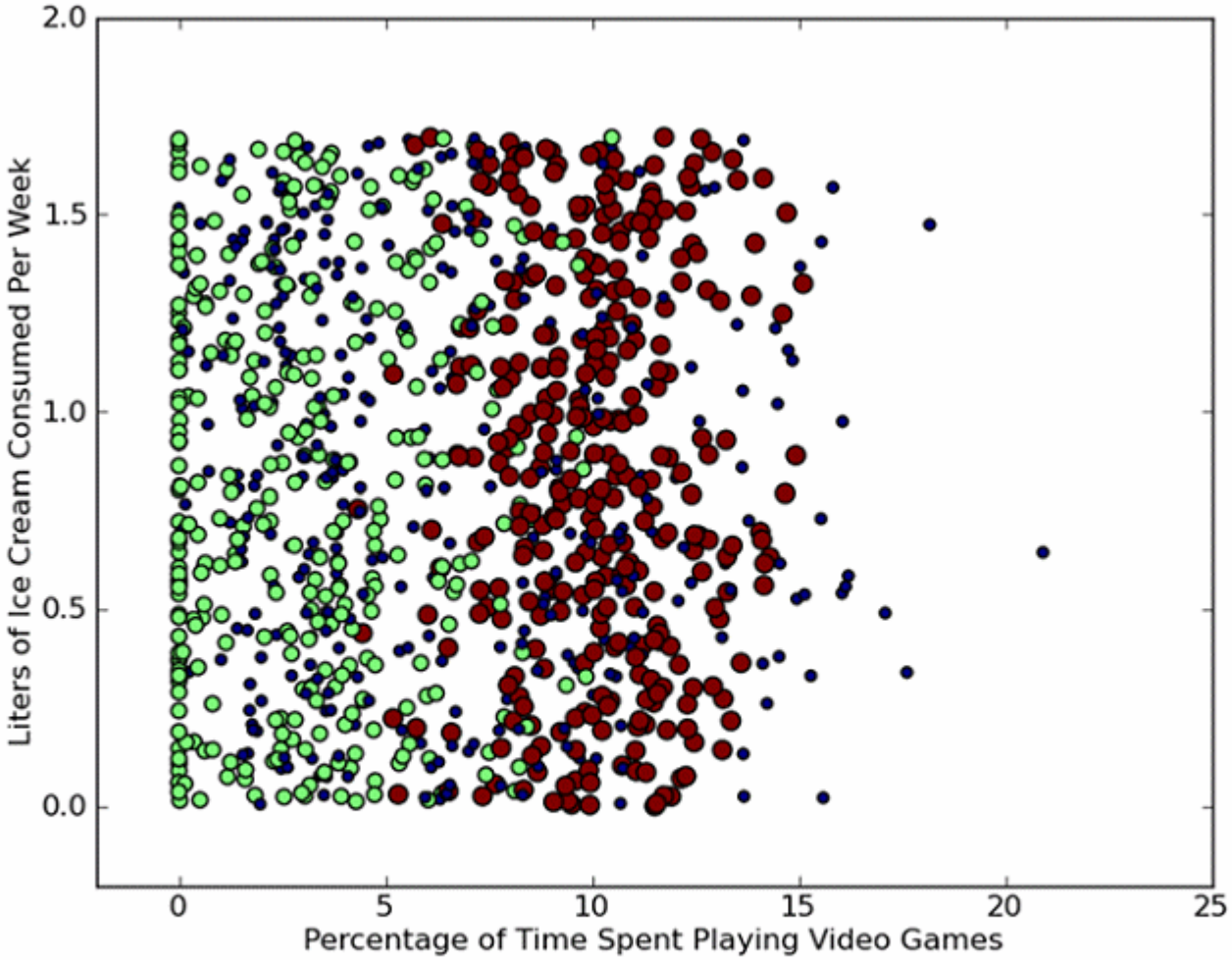

Figure 2.4 Dating data with markers changed by class label. It's easier to identify the different classes, but it's difficult to draw conclusions from looking at this data.

Analyze

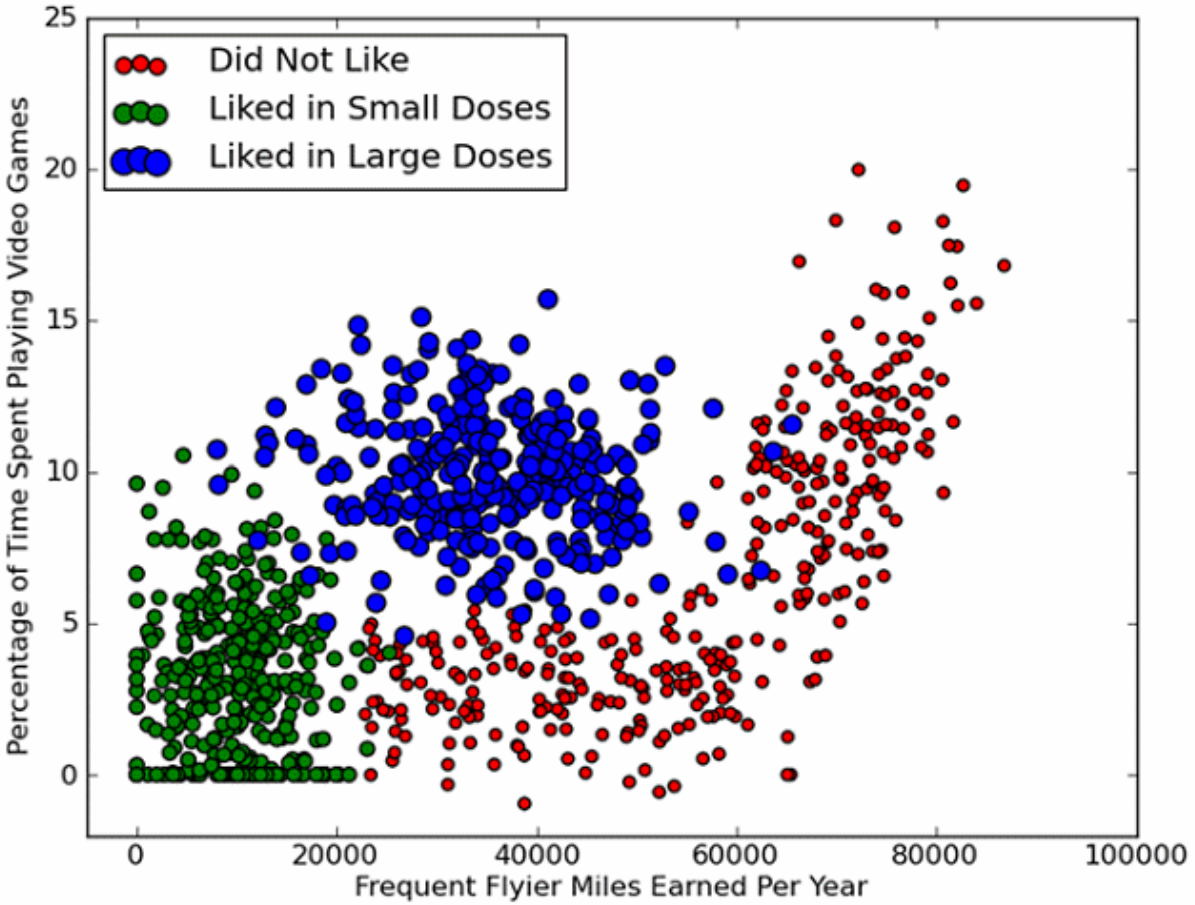

Figure 2.5 Dating data with frequent flier miles versus percentage of time spent playing video games plotted. The dating data has three features, and these two features show areas where the three different classes lie.

Be Careful to the Dominating Attribute

- Some ranges of attribute value dominate the others:

$$\sqrt{(0-67)^2+(20{,}000-32{,}000)^2+(1.1-0.1)^2}$$

Table 2.3 **Sample of data from improved results on a dating site**

	Percentage of time spent playing video games	Number of frequent flyer miles earned per year	Liters of ice cream consumed weekly	Category
1	0.8	400	0.5	1
2	12	134,000	0.9	3
3	0	20,000	1.1	2
4	67	32,000	0.1	2

Normalization of Attribute

- Common ranges to normalize them to are 0 to 1 or -1 to 1

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

Listing 2.3 Data-normalizing code

```
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
```

1 Element-wise division

Test the Classifier

- With k=3

Listing 2.4 Classifier testing code for dating site

```
def datingClassTest():
    hoRatio = 0.10
    datingDataMat,datingLabels = file2matrix('datingTestSet.txt')
    normMat, ranges, minVals = autoNorm(datingDataMat)
    m = normMat.shape[0]
    numTestVecs = int(m*hoRatio)
    errorCount = 0.0
    for i in range(numTestVecs):
        classifierResult = classify0(normMat[i,:],normMat[numTestVecs:m,:],\
                    datingLabels[numTestVecs:m],3)
        print "the classifier came back with: %d, the real answer is: %d"\
                    % (classifierResult, datingLabels[i])
        if (classifierResult != datingLabels[i]): errorCount += 1.0
    print "the total error rate is: %f" % (errorCount/float(numTestVecs))
```

Remark

- Any improvements?

Handwriting Recognition Example

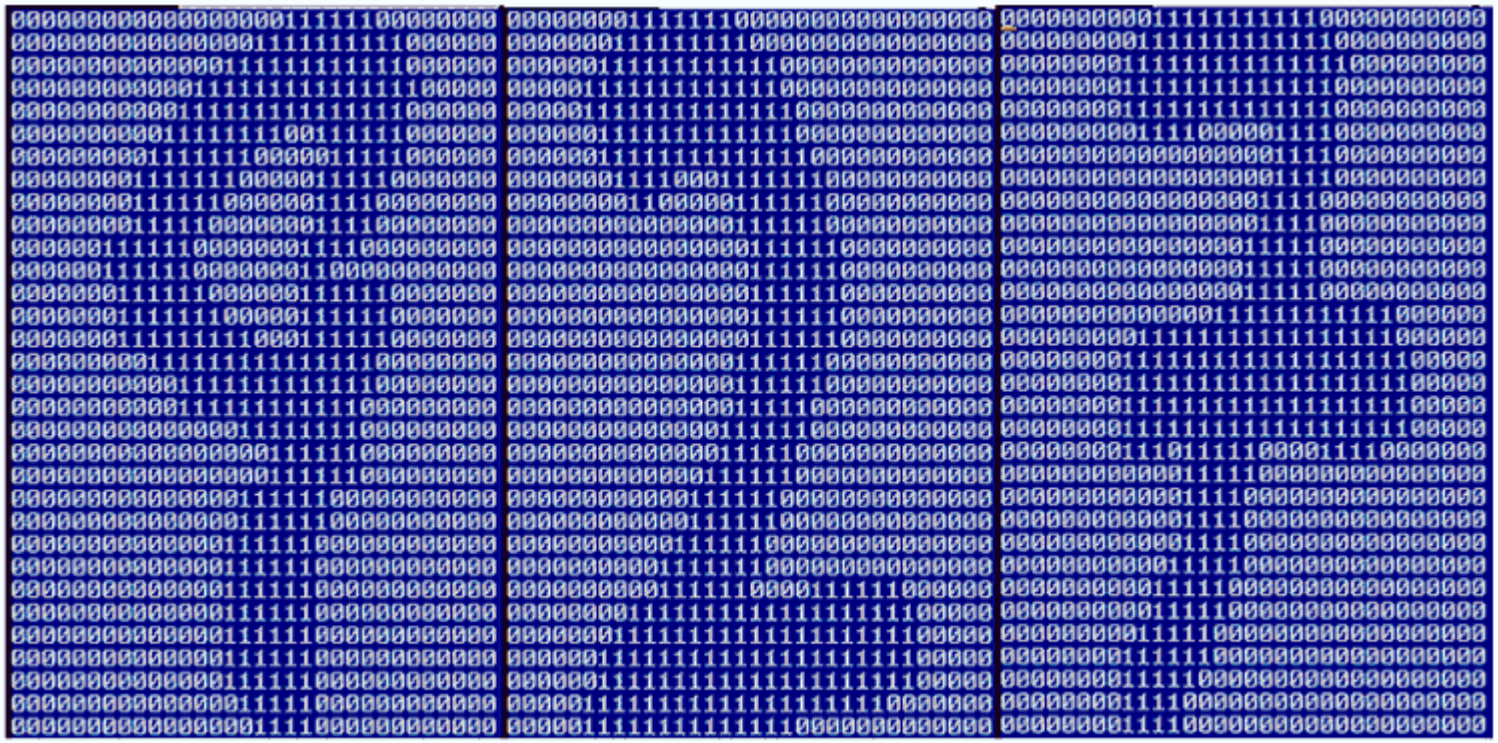

Figure 2.6 Examples of the handwritten digits dataset

Processing Flow

Example: using kNN on a handwriting recognition system

1. Collect: Text file provided.
2. Prepare: Write a function to convert from the image format to the list format used in our classifier, `classify0()`.
3. Analyze: We'll look at the prepared data in the Python shell to make sure it's correct.
4. Train: Doesn't apply to the kNN algorithm.
5. Test: Write a function to use some portion of the data as test examples. The test examples are classified against the non-test examples. If the predicted class doesn't match the real class, you'll count that as an error.
6. Use: Not performed in this example. You could build a complete program to extract digits from an image, such a system used to sort the mail in the United States.

Collect and Prepare

```
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
```

```
>>> testVector = kNN.img2vector('testDigits/0_13.txt')
>>> testVector[0,0:31]
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.])
>>> testVector[0,32:63]
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,
        1.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.])
```

Test the Classifier

- With k=3

Listing 2.6 Handwritten digits testing code

```
def handwritingClassTest():
    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)
    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)
        print "the classifier came back with: %d, the real answer is: %d"\
                                     % (classifierResult, classNumStr)
        if (classifierResult != classNumStr): errorCount += 1.0
    print "\nthe total number of errors is: %d" % errorCount
    print "\nthe total error rate is: %f" % (errorCount/float(mTest))
```

1 Get contents of directory

2 Process class num from filename

Remark

- Any improvements?

Summary

- kNN: a simple and effective way to classify data
- Drawbacks
  - Full dataset is necessary for calculation (large data?)
  - Calculate distance for every piece of data
  - No underlying structure known