Dating classification with KNN algorithm.

Euclidean distance $d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$

Classification

```
from sklearn import *
import operator

def classify(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]
```

Read into matrix

```
def file2matrix(filename):
    fr = open(filename)
    arrayOLines = fr.readlines()  # get the number of lines in the file
    returnMat = zeros(len(arrayOLines), 3)  # prepare matrix to return
    classLabelVector = []  # prepare labels return
    index = 0
    for line in arrayOLines:
        line = line.strip()
        listFromLine = line.split('\t')
        returnMat[index,:] = listFromLine[0:3]
        classLabelVector.append(int(listFromLine[-1]))
        index += 1
    return returnMat, classLabelVector
```

Norming

```
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))  # element wise divide
    return normDataSet, ranges, minVals
```

Dating test

```
def datingClassTest():
    hoRatio = 0.50  # hold out 10%
    datingDataMat, datingDataLabels = file2matrix('
    normMat, ranges, minVals = autoNorm(datingDataMat)
    m = normMat.shape[0]
    numTestVecs = int(m * hoRatio)
    errorCount = 0.0
    for i in range(numTestVecs):
        classifierResult = classify(
            normMat[i, :], normMat[numTestVecs:m, :], datingDataLabels[numTestVecs:m], 3
        print("the classifier came back with: %d, the real answer is: %d" % (
            classifierResult,
            datingDataLabels[i]))
        if (classifierResult != datingDataLabels[i]):
            errorCount += 1.
    print("the total error rate is: %f" % (errorCount / float(numTestVecs)))
    print(errorCount)

datingClassTest()
```

```
the classifier came back with: 3, the real answer is: 2
the classifier came back with: 3, the real answer is: 3
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 2, the real answer is: 2
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 3, the real answer is: 3
the classifier came back with: 3, the real answer is: 3
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 3, the real answer is: 3
the classifier came back with: 3, the real answer is: 3
the classifier came back with: 2, the real answer is: 2
the classifier came back with: 2, the real answer is: 2
the classifier came back with: 2, the real answer is: 2
```