机器学习实战》学习笔记（二）：k-近邻算法

原创 我是管小亮 2019-08-19 22:36:02 4411 收藏 28 版权
分类专栏： Machine Learning♡ 文章标签： 机器学习 机器学习实战 读书笔记 k-近邻算法

欢迎关注WX公众号：【程序员管小亮】

【机器学习】《机器学习实战》读书笔记及代码 总目录

- https://blog.csdn.net/TeFuirnever/article/details/99701256

GitHub代码地址：

- https://github.com/TeFuirnever/Machine-Learning-in-Action

目录

本章内容

- k-近邻分类算法
- 从文本文件中解析和导入数据
- 使用Matplotlib创建扩散图
- 归一化数值

这一节早在【CS231n】斯坦福大学李飞飞视觉识别课程笔记中就讲过了，感兴趣的同学可以去学一学这个课程，同时还有python的相关课程【Python - 100天从新手到大师】，但是没有具体的实现代码，所以今天来搞一下。

1、k-近邻算法概述

简单地说，k-近邻算法采用测量不同特征值之间的距离方法进行分类。

> k-近邻算法
> 优点：精度高、对异常值不敏感、无数据输入假定。
> 缺点：计算复杂度高、空间复杂度高。
> 适用数据范围：数值型和标称型。

k-近邻算法（kNN），它的工作原理是：存在一个样本数据集合，也称作训练样本集，并且样本集中每个数据都存在标签，即我们知道样本集中每一数据与所属分类的对应关系。输入没有标签的新数据后，将新数据的每个特征与样本集中数据对应的特征进行比较，然后算法提取样本集中特征最相似数据（最近邻）的分类标签。最后，选择k个最相似数据中出现次数最多的分类，作为新数据的分类。

一般来说，只选择样本数据集中前k个最相似的数据，这就是 **k-近邻** 算法中k的出处，通常k是不大于20的整数。

书上举了一个简单但是经典的例子——电影分类，有人曾经统计过很多电影的打斗镜头和接吻镜头，图2-1显示了6部电影的打斗和接吻镜头数。

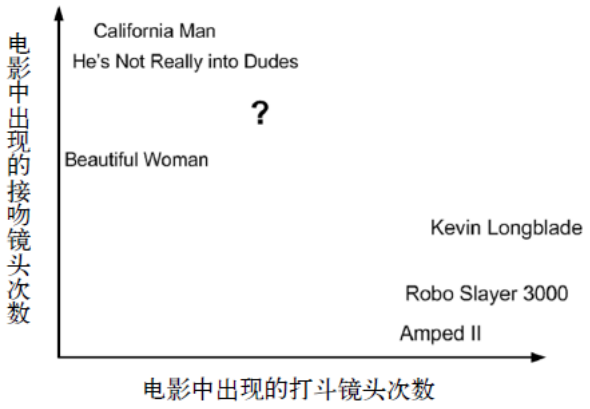


图2-1　使用打斗和接吻镜头数分类电影

首先需要知道这个未知电影存在多少个打斗镜头和接吻镜头，图2-1中问号位置是该未知电影出现的镜头数图形化展示，具体数字参见表2-1。

表2-1　每部电影的打斗镜头数、接吻镜头数以及电影评估类型

电影名称	打斗镜头	接吻镜头	电影类型
California Man	3	104	爱情片
He's Not Really into Dudes	2	100	爱情片
Beautiful Woman	1	81	爱情片
Kevin Longblade	101	10	动作片
Robo Slayer 3000	99	5	动作片
Amped II	98	2	动作片
?	18	90	未知

首先计算未知电影与样本集中其他电影的距离，如表2-2所示。

表2-2　已知电影与未知电影的距离

电影名称	与未知电影的距离
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

按照距离递增排序，可以找到k个距离最近的电影。假定k=3，则三个最靠近的电影依次是He's Not Really into Dudes、Beautiful Woman和California Man。k-近邻算法按照距离最近的三部电影的类型，决定未知电影的类型，而这三部电影全是爱情片，因此我们判定未知电影是爱情片。（当然根据你的经验应该也是认为爱情片的概率更高才是，因为接吻镜头90个，但是打斗镜头才18个，额，不可以开车，谁说的爱情动作片？？？）

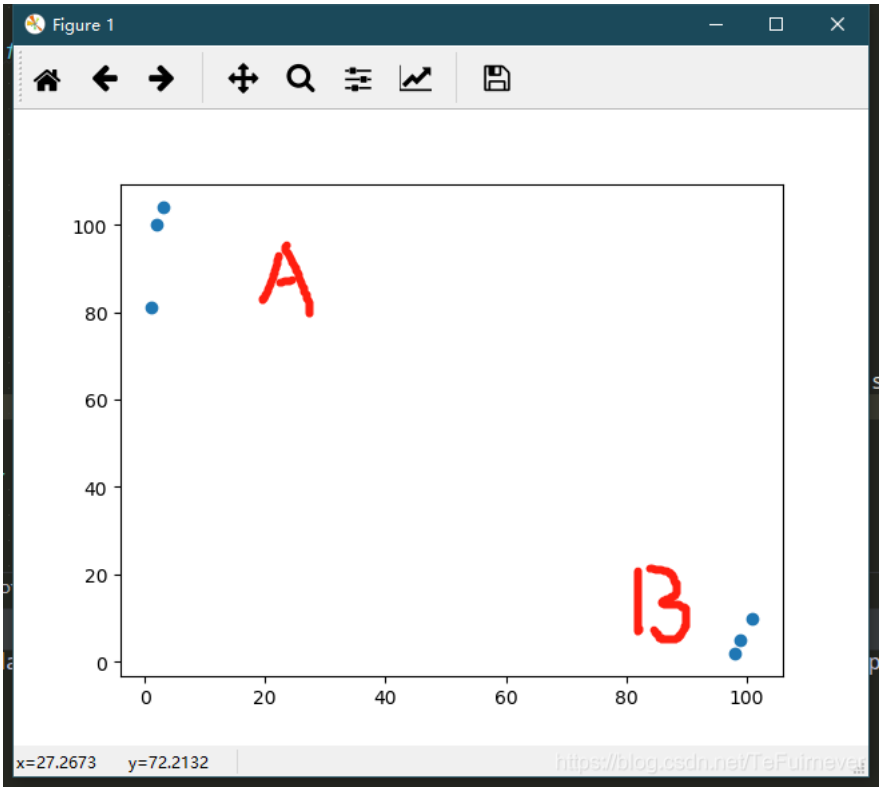
> k-近邻算法的一般流程
> (1) 收集数据：可以使用任何方法。
> (2) 准备数据：距离计算所需要的数值，最好是结构化的数据格式。
> (3) 分析数据：可以使用任何方法。
> (4) 训练算法：此步骤不适用于k-近邻算法。
> (5) 测试算法：计算错误率。
> (6) 使用算法：首先需要输入样本数据和结构化的输出结果，然后运行k-近邻算法判定输入数据分别属于哪个分类，最后应用对计算出的分类执行后续的处理。

1. 准备：使用Python 导入数据

对于表2.1中的数据，我们可以使用numpy直接创建，代码如下：

```python
import numpy as np

"""
Parameters:
    无
Returns:
    group - 数据集
        labels - 分类标签
"""
# 函数说明:创建数据集
def createDataSet():
    #六组二维特征
    group = np.array([[3,104],[2,100],[1,81],[101,10],[99,5],[98,2]])
    #六组特征的标签
    labels = ['爱情片','爱情片','爱情片','动作片','动作片','动作片']
    return group, labels

def showDataSet(dataMat, labelMat):
    data_plus = []
    for i in range(len(group)):
        data_plus.append(dataMat[i])
    data_plus_np = np.array(data_plus)# 转换为numpy矩阵
    plt.scatter(np.transpose(data_plus_np)[0],np.transpose(data_plus_np)[1])#散点图
    plt.show()


if __name__ == '__main__':
    #创建数据集
    group, labels = createDataSet()
    #打印数据集
    print(group)
    print(labels)
```

```
>>>
[[  3 104]
 [  2 100]
 [  1  81]
 [101  10]
 [ 99   5]
 [ 98   2]]
['爱情片', '爱情片', '爱情片', '动作片', '动作片', '动作片']
```



这里将数据点左上方定义为类A（也就是爱情片），数据点右下方定义为类B（也就是动作片）。

一共有6组数据，每组数据有两个已知的属性或者特征值。上面的group矩阵每行包含一个不同的数据，我们可以把它想象为某个日志文件中不同的测量点或者入口。由于人类大脑的限制，通常只能可视化处理三维以下的事务。因此为了简单地实现数据可视化，对于每个数据点通常只使用两个特征。向量labels包含了每个数据点的标签信息，labels包含的元素个数等于group矩阵行数。

2. 实施kNN 算法

> 对未知类别属性的数据集中的每个点依次执行以下操作:
> (1) 计算已知类别数据集中的点与当前点之间的距离;
> (2) 按照距离递增次序排序;
> (3) 选取与当前点距离最小的k个点;
> (4) 确定前k个点所在类别的出现频率;
> (5) 返回前k个点出现频率最高的类别作为当前点的预测分类。

根据两点距离公式（欧氏距离公式），计算两个向量点xA和xB之间的距离。

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

按照步骤，接下来对数据按照从小到大的次序排序。选择距离最小的前k个点，并返回分类结果。

```
import numpy as np
import operator

"""
Parameters:
    inX - 用于分类的数据(测试集)
    dataSet - 用于训练的数据(训练集)
    labes - 分类标签
    k - kNN算法参数,选择距离最小的k个点
Returns:
    sortedClassCount[0][0] - 分类结果
"""
# 函数说明:kNN算法,分类器
def classify0(inX, dataSet, labels, k):、
    #numpy函数shape[0]返回dataSet的行数
    dataSetSize = dataSet.shape[0]
    #在列向量方向上重复inX共1次(横向)，行向量方向上重复inX共dataSetSize次(纵向)
    diffMat = np.tile(inX, (dataSetSize, 1)) - dataSet
    #二维特征相减后平方
```

```
    #二维特征相减后平方
    sqDiffMat = diffMat**2
    #sum()所有元素相加，sum(0)列相加，sum(1)行相加
    sqDistances = sqDiffMat.sum(axis=1)
    #开方，计算出距离
    distances = sqDistances**0.5
    #返回distances中元素从小到大排序后的索引值
    sortedDistIndices = distances.argsort()
    #定一个记录类别次数的字典
    classCount = {}
    for i in range(k):
        #取出前k个元素的类别
        voteIlabel = labels[sortedDistIndices[i]]
        #dict.get(key,default=None),字典的get()方法,返回指定键的值,如果值不在字典中返回默认值。
        #计算类别次数
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
    #python3中用items()替换python2中的iteritems()
    #key=operator.itemgetter(1)根据字典的值进行排序
    #key=operator.itemgetter(0)根据字典的键进行排序
    #reverse降序排序字典
    sortedClassCount = sorted(classCount.items(),key=operator.itemgetter(1),reverse=True)
    #返回次数最多的类别,即所要分类的类别
    return sortedClassCount[0][0]
```

完整代码：

```
import numpy as np
import operator

"""
Parameters:
    无
Returns:
    group - 数据集
    labels - 分类标签
"""
# 函数说明:创建数据集
def createDataSet():
    #六组二维特征
    group = np.array([[3,104],[2,100],[1,81],[101,10],[99,5],[98,2]])
    #六组特征的标签
    labels = ['爱情片','爱情片','爱情片','动作片','动作片','动作片']
    return group, labels

"""
Parameters:
    inX - 用于分类的数据(测试集)
    dataSet - 用于训练的数据(训练集)
    labes - 分类标签
    k - kNN算法参数,选择距离最小的k个点
Returns:
    sortedClassCount[0][0] - 分类结果
"""
# 函数说明:kNN算法,分类器
def classify0(inX, dataSet, labels, k):
    #numpy函数shape[0]返回dataSet的行数
    dataSetSize = dataSet.shape[0]
    #在列向量方向上重复inX共1次(横向)，行向量方向上重复inX共dataSetSize次(纵向)
    diffMat = np.tile(inX, (dataSetSize, 1)) - dataSet
    #二维特征相减后平方
    sqDiffMat = diffMat**2
    #sum()所有元素相加，sum(0)列相加，sum(1)行相加
    sqDistances = sqDiffMat.sum(axis=1)
    #开方，计算出距离
    distances = sqDistances**0.5
    #返回distances中元素从小到大排序后的索引值
    sortedDistIndices = distances.argsort()
    #定一个记录类别次数的字典
    classCount = {}
    for i in range(k):
        #取出前k个元素的类别
        voteIlabel = labels[sortedDistIndices[i]]
        #dict.get(key,default=None),字典的get()方法,返回指定键的值,如果值不在字典中返回默认值。
        #计算类别次数
```

```
        #计算类别次数
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
    #python3中用items()替换python2中的iteritems()
    #key=operator.itemgetter(1)根据字典的值进行排序
    #key=operator.itemgetter(0)根据字典的键进行排序
    #reverse降序排序字典
    sortedClassCount = sorted(classCount.items(),key=operator.itemgetter(1),reverse=True)
    #返回次数最多的类别,即所要分类的类别
    return sortedClassCount[0][0]

if __name__ == '__main__':
    #创建数据集
    group, labels = createDataSet()
    #测试集
    test = [101,20]
    #kNN分类
    test_class = classify0(test, group, labels, 3)
    #打印分类结果
    print(test_class)
```

```
>>>
动作片
```

3. 如何测试分类器

有点那种我们依据自己的经验进行判断的意思，也就是上面我说的看“接吻动作”和“打斗动作”，然后进行判断的意味。

看到这，你可能会问：“分类器何种情况下会出错？”或者“答案是否总是正确的？”答案是否定的，分类器并不会得到百分百正确的结果，我们可以使用多种方法检测分类器的正确率。此外分类器的性能也会受到多种因素的影响，如分类器设置和数据集等。除此之外，不同的算法在不同数据集上的表现可能完全不同。

为了测试分类器的效果，我们可以使用已知答案的数据，当然答案不能告诉分类器，检验分类器给出的结果是否符合预期结果。通过大量的测试数据，我们可以得到分类器的错误率——分类器给出错误结果的次数除以测试执行的总数。错误率是常用的评估方法，主要用于评估分类器在某个数据集上的执行效果。完美分类器的错误率为0，最差分类器的错误率是1.0。同时，我们也不难发现，k-近邻算法没有进行数据的训练，直接使用未知的数据与已知的数据进行比较，得到结果。因此，可以说k-邻近算法不具有显式的学习过程。

2、使用k-近邻算法改进约会网站的配对效果

这里是一个比较有意思的话题产生的例子，也就是婚介网站等等的约会网站帮助你相亲，我的朋友海伦就是这样一个人，她一直使用在线约会网站寻找适合自己的约会对象，但是她发现尽管约会网站会推荐不同的人选，但并不是每一个人她都喜欢。经过一番总结，她发现曾交往过三种类型的人：不喜欢的人、魅力一般的人和极具魅力的人。

她希望我们的分类软件可以更好地帮助她将匹配对象划分到确切的分类中。此外，海伦自己还收集了一些约会网站未曾记录的数据信息，她认为这些数据更有助于匹配对象的归类。

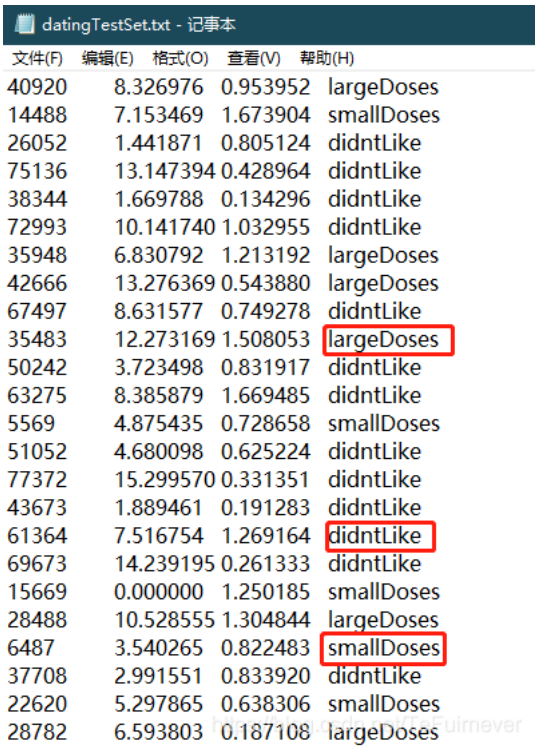
> 在约会网站上使用k-近邻算法
> (1) 收集数据：提供文本文件。
> (2) 准备数据：使用Python解析文本文件。
> (3) 分析数据：使用Matplotlib画二维扩散图。
> (4) 训练算法：此步骤不适用于k-近邻算法。
> (5) 测试算法：使用海伦提供的部分数据作为测试样本。测试样本和非测试样本的区别在于：测试样本是已经完成分类的数据，如果预测分类与实际类别不同，则标记为一个错误。
> (6) 使用算法：产生简单的命令行程序，然后海伦可以输入一些特征数据以判断对方是否为自己喜欢的类型。

1. 准备数据：从文本文件中解析数据

海伦收集约会数据已经有了一段时间，她把这些数据存放在文本文件datingTestSet.txt中，每个样本数据占据一行，总共有1000行。海伦的样本主要包含以下3种特征：

- 每年获得的飞行常客里程数
- 玩视频游戏所耗时间百分比
- 每周消费的冰淇淋公升数

观察数据，可以看到有largeDoses、smallDoses、didntLike三类标记。


datingTestSet.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
40920   8.326976 0.953952 largeDoses
14488   7.153469 1.673904 smallDoses
26052   1.441871 0.805124 didntLike
75136   13.147394 0.428964 didntLike
38344   1.669788 0.134296 didntLike
72993   10.141740 1.032955 didntLike
35948   6.830792 1.213192 largeDoses
42666   13.276369 0.543880 largeDoses
67497   8.631577 0.749278 didntLike
35483   12.273169 1.508053 largeDoses
50242   3.723498 0.831917 didntLike
63275   8.385879 1.669485 didntLike
5569    4.875435 0.728658 smallDoses
51052   4.680098 0.625224 didntLike
77372   15.299570 0.331351 didntLike
43673   1.889461 0.191283 didntLike
61364   7.516754 1.269164 didntLike
69673   14.239195 0.261333 didntLike
15669   0.000000 1.250185 smallDoses
28488   10.528555 1.304844 largeDoses
6487    3.540265 0.822483 smallDoses
37708   2.991551 0.833920 didntLike
22620   5.297865 0.638306 smallDoses
28782   6.593803 0.187108 largeDoses
```

在将上述特征数据输入到分类器前，必须将待处理的数据的格式改变为分类器可以接收的格式。分类器接收的数据是什么格式的？从前面讲的你已经知道，要将数据分类两部分，即 **特征矩阵** 和对应的 **分类标签** 向量。在kNN.py中创建名为file2matrix的函数，以此来处理输入格式问题。该函数的输入为文件名字符串，输出为 **训练样本矩阵** 和 **类标签向量**。

```
import numpy as np

"""
Parameters:
    filename - 文件名
Returns:
    returnMat - 特征矩阵
    classLabelVector - 分类Label向量
"""
# 函数说明:打开并解析文件，对数据进行分类：1代表不喜欢,2代表魅力一般,3代表极具魅力
def file2matrix(filename):
    #打开文件
    fr = open(filename)
    #读取文件所有内容
    arrayOLines = fr.readlines()
    #得到文件行数
    numberOfLines = len(arrayOLines)
    #返回的NumPy矩阵,解析完成的数据:numberOfLines行,3列
    returnMat = np.zeros((numberOfLines,3))
    #返回的分类标签向量
    classLabelVector = []
    #行的索引值
    index = 0
    for line in arrayOLines:
        #s.strip(rm)，当rm空时,默认删除空白符(包括'\n','\r','\t',' ')
        line = line.strip()
        #使用s.split(str="",num=string,cout(str))将字符串根据'\t'分隔符进行切片。
        listFromLine = line.split('\t')
        #将数据前三列提取出来,存放到returnMat的NumPy矩阵中,也就是特征矩阵
        returnMat[index,:] = listFromLine[0:3]
        #根据文本中标记的喜欢的程度进行分类,1代表不喜欢,2代表魅力一般,3代表极具魅力
        if listFromLine[-1] == 'didntLike':
            classLabelVector.append(1)
        elif listFromLine[-1] == 'smallDoses':
            classLabelVector.append(2)
        elif listFromLine[-1] == 'largeDoses':
            classLabelVector.append(3)
        index += 1
    return returnMat, classLabelVector
```

```
if __name__ == '__main__':
    #打开的文件名
    filename = "datingTestSet.txt"
    #打开并处理数据
    datingDataMat, datingLabels = file2matrix(filename)
    print(datingDataMat)
    print(datingLabels)
```

```
>>>
[[  4.09200000e+04   8.32697600e+00   9.53952000e-01]
 [  1.44880000e+04   7.15346900e+00   1.67390400e+00]
 [  2.60520000e+04   1.44187100e+00   8.05124000e-01]
 ...,
 [  2.65750000e+04   1.06501020e+01   8.66627000e-01]
 [  4.81110000e+04   9.13452800e+00   7.28045000e-01]
 [  4.37570000e+04   7.88260100e+00   1.33244600e+00]]
[3, 2, 1, 1, 1, 1, 3, 3, 1, 3, 1, 1, 2, 1, 1, 1, 1, 1, 2, 3, 2, 1, 2, 3, 2, 3, 2, 3, 2, 1, 3, 1, 3, 1, 2, 1, 1, 2, 3, 3, 1, 2, 3
```

上面是特征矩阵，下面是标签向量。

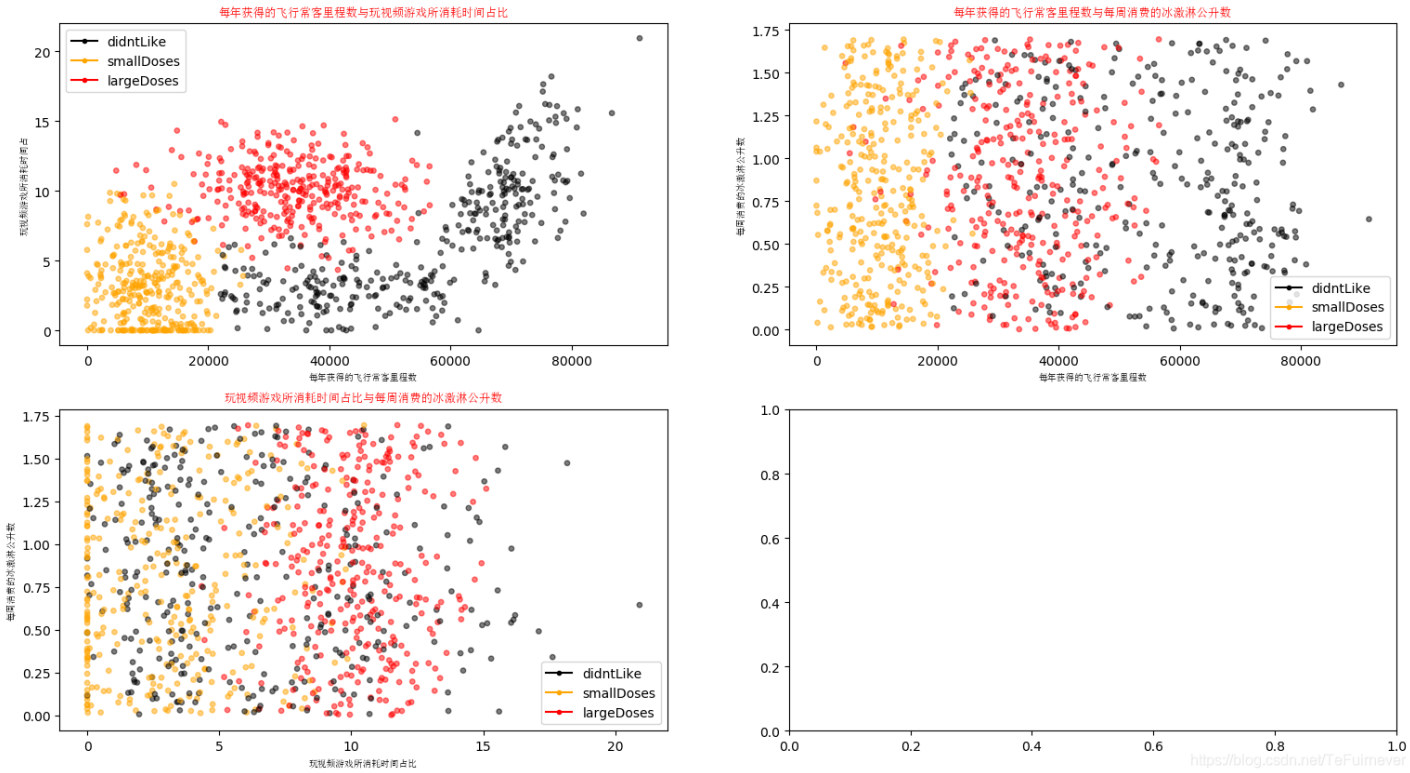
2. 分析数据：使用Matplotlib 创建散点图

现在已经从文本文件中导入了数据，并将其格式化为想要的格式，接着我们需要了解数据的真实含义。当然我们可以直接浏览文本文件，但是这种方法非常不友好，一般来说，我们会采用图形化的方式直观地展示数据。下面就用Python工具来图形化展示数据内容，以便辨识出一些数据模式。

```
from matplotlib.font_manager import FontProperties
import matplotlib.lines as mlines
import matplotlib.pyplot as plt
import numpy as np

"""
Parameters:
    filename - 文件名
Returns:
    returnMat - 特征矩阵
    classLabelVector - 分类Label向量
"""
# 函数说明:打开并解析文件，对数据进行分类：1代表不喜欢,2代表魅力一般,3代表极具魅力
def file2matrix(filename):
    #打开文件
    fr = open(filename)
    #读取文件所有内容
    arrayOLines = fr.readlines()
    #得到文件行数
    numberOfLines = len(arrayOLines)
    #返回的NumPy矩阵,解析完成的数据:numberOfLines行,3列
    returnMat = np.zeros((numberOfLines,3))
    #返回的分类标签向量
    classLabelVector = []
    #行的索引值
    index = 0
    for line in arrayOLines:
        #s.strip(rm)，当rm空时,默认删除空白符(包括'\n','\r','\t',' ')
        line = line.strip()
        #使用s.split(str="",num=string,cout(str))将字符串根据'\t'分隔符进行切片。
        listFromLine = line.split('\t')
        #将数据前三列提取出来,存放到returnMat的NumPy矩阵中,也就是特征矩阵
        returnMat[index,:] = listFromLine[0:3]
        #根据文本中标记的喜欢的程度进行分类,1代表不喜欢,2代表魅力一般,3代表极具魅力
        if listFromLine[-1] == 'didntLike':
            classLabelVector.append(1)
        elif listFromLine[-1] == 'smallDoses':
            classLabelVector.append(2)
        elif listFromLine[-1] == 'largeDoses':
            classLabelVector.append(3)
```

```
        index += 1
    return returnMat, classLabelVector

"""
Parameters:
    datingDataMat - 特征矩阵
    datingLabels - 分类Label
Returns:
    无
"""
# 函数说明:可视化数据
def showdatas(datingDataMat, datingLabels):
    #设置汉字格式
    font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc", size=14)
    #将fig画布分隔成1行1列,不共享x轴和y轴,fig画布的大小为(13,8)
    #当nrow=2,nclos=2时,代表fig画布被分为四个区域,axs[0][0]表示第一行第一个区域
    fig, axs = plt.subplots(nrows=2, ncols=2,sharex=False, sharey=False, figsize=(13,8))

    numberOfLabels = len(datingLabels)
    LabelsColors = []
    for i in datingLabels:
        if i == 1:
            LabelsColors.append('black')
        if i == 2:
            LabelsColors.append('orange')
        if i == 3:
            LabelsColors.append('red')
    #画出散点图,以datingDataMat矩阵的第一(飞行常客例程)、第二列(玩游戏)数据画散点数据,散点大小为15,透明度为0.5
    axs[0][0].scatter(x=datingDataMat[:,0], y=datingDataMat[:,1], color=LabelsColors,s=15, alpha=.5)
    #设置标题,x轴label,y轴label
    axs0_title_text = axs[0][0].set_title(u'每年获得的飞行常客里程数与玩视频游戏所消耗时间占比',FontProperties=font)
    axs0_xlabel_text = axs[0][0].set_xlabel(u'每年获得的飞行常客里程数',FontProperties=font)
    axs0_ylabel_text = axs[0][0].set_ylabel(u'玩视频游戏所消耗时间占',FontProperties=font)
    plt.setp(axs0_title_text, size=9, weight='bold', color='red')
    plt.setp(axs0_xlabel_text, size=7, weight='bold', color='black')
    plt.setp(axs0_ylabel_text, size=7, weight='bold', color='black')

    #画出散点图,以datingDataMat矩阵的第一(飞行常客例程)、第三列(冰激凌)数据画散点数据,散点大小为15,透明度为0.5
    axs[0][1].scatter(x=datingDataMat[:,0], y=datingDataMat[:,2], color=LabelsColors,s=15, alpha=.5)
    #设置标题,x轴label,y轴label
    axs1_title_text = axs[0][1].set_title(u'每年获得的飞行常客里程数与每周消费的冰激淋公升数',FontProperties=font)
    axs1_xlabel_text = axs[0][1].set_xlabel(u'每年获得的飞行常客里程数',FontProperties=font)
    axs1_ylabel_text = axs[0][1].set_ylabel(u'每周消费的冰激淋公升数',FontProperties=font)
    plt.setp(axs1_title_text, size=9, weight='bold', color='red')
    plt.setp(axs1_xlabel_text, size=7, weight='bold', color='black')
    plt.setp(axs1_ylabel_text, size=7, weight='bold', color='black')

    #画出散点图,以datingDataMat矩阵的第二(玩游戏)、第三列(冰激凌)数据画散点数据,散点大小为15,透明度为0.5
    axs[1][0].scatter(x=datingDataMat[:,1], y=datingDataMat[:,2], color=LabelsColors,s=15, alpha=.5)
    #设置标题,x轴label,y轴label
    axs2_title_text = axs[1][0].set_title(u'玩视频游戏所消耗时间占比与每周消费的冰激淋公升数',FontProperties=font)
    axs2_xlabel_text = axs[1][0].set_xlabel(u'玩视频游戏所消耗时间占比',FontProperties=font)
    axs2_ylabel_text = axs[1][0].set_ylabel(u'每周消费的冰激淋公升数',FontProperties=font)
    plt.setp(axs2_title_text, size=9, weight='bold', color='red')
    plt.setp(axs2_xlabel_text, size=7, weight='bold', color='black')
    plt.setp(axs2_ylabel_text, size=7, weight='bold', color='black')
    #设置图例
    didntLike = mlines.Line2D([], [], color='black', marker='.',
                      markersize=6, label='didntLike')
    smallDoses = mlines.Line2D([], [], color='orange', marker='.',
                      markersize=6, label='smallDoses')
    largeDoses = mlines.Line2D([], [], color='red', marker='.',
                      markersize=6, label='largeDoses')
    #添加图例
    axs[0][0].legend(handles=[didntLike,smallDoses,largeDoses])
    axs[0][1].legend(handles=[didntLike,smallDoses,largeDoses])
    axs[1][0].legend(handles=[didntLike,smallDoses,largeDoses])
    #显示图片
    plt.show()


if __name__ == '__main__':
    #打开的文件名
```

```
111     #打开的文件名
112     filename = "datingTestSet.txt"
113     #打开并处理数据
114     datingDataMat, datingLabels = file2matrix(filename)
115     showdatas(datingDataMat, datingLabels)
```



通过数据可以很直观的发现数据的规律，比如横纵坐标是玩游戏所消耗时间占比与每年获得的飞行常客里程数，只考虑这二维的特征信息，就会发现，海伦很喜欢生活质量高的男人。为什么这么说呢?

每年获得的飞行常客里程数这一个特征表明，海伦喜欢能享受飞行常客奖励计划的男人，但是不能经常坐飞机，疲于奔波，满世界飞。同时，这个男人也要玩视频游戏，并且占一定时间比例。能到处飞，又能经常玩游戏的男人是什么样的男人？很显然，有生活质量，并且生活悠闲的人。哈哈，兄弟们，姐妹们，是不是发现了什么了不得的事啊 😃

3. 准备数据：归一化数值

表2-3给出了提取的四组数据，如果想要计算样本3和样本4之间的距离，可以使用下面的方法：

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

很容易发现，上面方程中数字差值最大的属性对计算结果的影响最大，也就是说，每年获取的飞行常客里程数对于计算结果的影响将远远大于表2-3中其他两个特征——玩视频游戏的和每周消费冰淇淋公升数——的影响。而产生这种现象的唯一原因，仅仅是因为飞行常客里程数远大于其他特征值。但海伦认为这三种特征是同等重要的，因此作为三个等权重的特征之一，飞行常客里程数并不应该如此严重地影响到计算结果。

表2-3　约会网站原始数据改进之后的样本数据

	玩视频游戏所耗时间百分比	每年获得的飞行常客里程数	每周消费的冰淇淋公升数	样本分类
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

在处理这种不同取值范围的特征值时，通常采用的方法是将数值归一化，如将取值范围处理为0到1或者□1到1之间。下面的公式可以将任意取值范围的特征值转化为0到1区间内的值：

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

其中min和max分别是数据集中的最小特征值和最大特征值。虽然改变数值取值范围增加了分类器的复杂度，但为了得到准确结果，我们必须这样做。

```python
import numpy as np

"""
Parameters:
    filename - 文件名
Returns:
    returnMat - 特征矩阵
    classLabelVector - 分类Label向量
"""
# 函数说明:打开并解析文件，对数据进行分类：1代表不喜欢,2代表魅力一般,3代表极具魅力
def file2matrix(filename):
    #打开文件
    fr = open(filename)
    #读取文件所有内容
    arrayOLines = fr.readlines()
    #得到文件行数
    numberOfLines = len(arrayOLines)
    #返回的NumPy矩阵,解析完成的数据:numberOfLines行,3列
    returnMat = np.zeros((numberOfLines,3))
    #返回的分类标签向量
    classLabelVector = []
    #行的索引值
    index = 0
    for line in arrayOLines:
        #s.strip(rm)，当rm空时,默认删除空白符(包括'\n','\r','\t',' ')
        line = line.strip()
        #使用s.split(str="",num=string,cout(str))将字符串根据'\t'分隔符进行切片。
        listFromLine = line.split('\t')
        #将数据前三列提取出来,存放到returnMat的NumPy矩阵中,也就是特征矩阵
        returnMat[index,:] = listFromLine[0:3]
        #根据文本中标记的喜欢的程度进行分类,1代表不喜欢,2代表魅力一般,3代表极具魅力
        if listFromLine[-1] == 'didntLike':
            classLabelVector.append(1)
        elif listFromLine[-1] == 'smallDoses':
            classLabelVector.append(2)
        elif listFromLine[-1] == 'largeDoses':
            classLabelVector.append(3)
        index += 1
    return returnMat, classLabelVector

"""
Parameters:
    dataSet - 特征矩阵
Returns:
    normDataSet - 归一化后的特征矩阵
    ranges - 数据范围
    minVals - 数据最小值
"""
# 函数说明:对数据进行归一化
def autoNorm(dataSet):
    #获得数据的最小值
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    #最大值和最小值的范围
    ranges = maxVals - minVals
    #shape(dataSet)返回dataSet的矩阵行列数
    normDataSet = np.zeros(np.shape(dataSet))
    #返回dataSet的行数
    m = dataSet.shape[0]
    #原始值减去最小值
    normDataSet = dataSet - np.tile(minVals, (m, 1))
    #除以最大和最小值的差,得到归一化数据
    normDataSet = normDataSet / np.tile(ranges, (m, 1))
    #返回归一化数据结果,数据范围,最小值
    return normDataSet, ranges, minVals

```

```
64
65 if __name__ == '__main__':
66     #打开的文件名
67     filename = "datingTestSet.txt"
68     #打开并处理数据
69     datingDataMat, datingLabels = file2matrix(filename)
70     normDataSet, ranges, minVals = autoNorm(datingDataMat)
71     print(normDataSet)
72     print(ranges)
73     print(minVals)
74
```

```
1  >>>
2  [[ 0.44832535  0.39805139  0.56233353]
3   [ 0.15873259  0.34195467  0.98724416]
4   [ 0.28542943  0.06892523  0.47449629]
5   ...,
6   [ 0.29115949  0.50910294  0.51079493]
7   [ 0.52711097  0.43665451  0.4290048 ]
8   [ 0.47940793  0.3768091   0.78571804]]
9  [  9.12730000e+04   2.09193490e+01   1.69436100e+00]
10 [ 0.          0.          0.001156]
```

从运行结果可以看到，数据顺利地进行了归一化，并且求出了数据的取值范围和数据的最小值，这两个值是在分类的时候需要用到的，直接先求解出来，也算是对数据预处理了。

4. 测试算法：作为完整程序验证分类器

机器学习算法一个很重要的工作就是评估算法的正确率，通常我们只提供已有数据的90%作为训练样本来训练分类器，而使用其余的10%数据去测试分类器，检测分类器的正确率。需要注意的是，10%的测试数据应该是随机选择的，由于海伦提供的数据并没有按照特定目的来排序，所以我么你可以随意选择10%数据而不影响其随机性。

前面已经提到可以使用错误率来检测分类器的性能。对于分类器来说，错误率就是分类器给出错误结果的次数除以测试数据的总数，完美分类器的错误率为0，而错误率为1.0的分类器不会给出任何正确的分类结果。代码里我们定义一个计数器变量，每次分类器错误地分类数据，计数器就加1，程序执行完成之后计数器的结果除以数据点总数即是错误率。

```
1  import numpy as np
2  import operator
3
4  """
5  Parameters:
6      inX - 用于分类的数据(测试集)
7      dataSet - 用于训练的数据(训练集)
8      labes - 分类标签
9      k - kNN算法参数,选择距离最小的k个点
10 Returns:
11     sortedClassCount[0][0] - 分类结果
12 """
13 # 函数说明:kNN算法,分类器
14 def classify0(inX, dataSet, labels, k):
15     #numpy函数shape[0]返回dataSet的行数
16     dataSetSize = dataSet.shape[0]
17     #在列向量方向上重复inX共1次(横向),行向量方向上重复inX共dataSetSize次(纵向)
18     diffMat = np.tile(inX, (dataSetSize, 1)) - dataSet
19     #二维特征相减后平方
20     sqDiffMat = diffMat**2
21     #sum()所有元素相加,sum(0)列相加,sum(1)行相加
22     sqDistances = sqDiffMat.sum(axis=1)
23     #开方,计算出距离
24     distances = sqDistances**0.5
25     #返回distances中元素从小到大排序后的索引值
26     sortedDistIndices = distances.argsort()
27     #定一个记录类别次数的字典
28     classCount = {}
29     for i in range(k):
30         #取出前k个元素的类别
31         voteIlabel = labels[sortedDistIndices[i]]
32         #dict.get(key,default=None),字典的get()方法,返回指定键的值,如果值不在字典中返回默认值。
33         #计算类别次数
           classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
```

```
    #python3中用items()替换python2中的iteritems()
    #key=operator.itemgetter(1)根据字典的值进行排序
    #key=operator.itemgetter(0)根据字典的键进行排序
    #reverse降序排序字典
    sortedClassCount = sorted(classCount.items(),key=operator.itemgetter(1),reverse=True)
    #返回次数最多的类别,即所要分类的类别
    return sortedClassCount[0][0]

"""
Parameters:
    filename - 文件名
Returns:
    returnMat - 特征矩阵
    classLabelVector - 分类Label向量
"""
# 函数说明:打开并解析文件，对数据进行分类：1代表不喜欢,2代表魅力一般,3代表极具魅力
def file2matrix(filename):
    #打开文件
    fr = open(filename)
    #读取文件所有内容
    arrayOLines = fr.readlines()
    #得到文件行数
    numberOfLines = len(arrayOLines)
    #返回的NumPy矩阵,解析完成的数据:numberOfLines行,3列
    returnMat = np.zeros((numberOfLines,3))
    #返回的分类标签向量
    classLabelVector = []
    #行的索引值
    index = 0
    for line in arrayOLines:
        #s.strip(rm)，当rm空时,默认删除空白符(包括'\n','\r','\t',' ')
        line = line.strip()
        #使用s.split(str="",num=string,cout(str))将字符串根据'\t'分隔符进行切片。
        listFromLine = line.split('\t')
        #将数据前三列提取出来,存放到returnMat的NumPy矩阵中,也就是特征矩阵
        returnMat[index,:] = listFromLine[0:3]
        #根据文本中标记的喜欢的程度进行分类,1代表不喜欢,2代表魅力一般,3代表极具魅力
        if listFromLine[-1] == 'didntLike':
            classLabelVector.append(1)
        elif listFromLine[-1] == 'smallDoses':
            classLabelVector.append(2)
        elif listFromLine[-1] == 'largeDoses':
            classLabelVector.append(3)
        index += 1
    return returnMat, classLabelVector

"""
Parameters:
    dataSet - 特征矩阵
Returns:
    normDataSet - 归一化后的特征矩阵
    ranges - 数据范围
    minVals - 数据最小值
"""
# 函数说明:对数据进行归一化
def autoNorm(dataSet):
    #获得数据的最小值
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    #最大值和最小值的范围
    ranges = maxVals - minVals
    #shape(dataSet)返回dataSet的矩阵行列数
    normDataSet = np.zeros(np.shape(dataSet))
    #返回dataSet的行数
    m = dataSet.shape[0]
    #原始值减去最小值
    normDataSet = dataSet - np.tile(minVals, (m, 1))
    #除以最大和最小值的差,得到归一化数据
    normDataSet = normDataSet / np.tile(ranges, (m, 1))
    #返回归一化数据结果,数据范围,最小值
    return normDataSet, ranges, minVals

"""
Parameters:
    无
```

```
    无
Returns:
    normDataSet - 归一化后的特征矩阵
    ranges - 数据范围
    minVals - 数据最小值
"""
# 函数说明:分类器测试函数
def datingClassTest():
    #打开的文件名
    filename = "datingTestSet.txt"
    #将返回的特征矩阵和分类向量分别存储到datingDataMat和datingLabels中
    datingDataMat, datingLabels = file2matrix(filename)
    #取所有数据的百分之十
    hoRatio = 0.10
    #数据归一化,返回归一化后的矩阵,数据范围,数据最小值
    normMat, ranges, minVals = autoNorm(datingDataMat)
    #获得normMat的行数
    m = normMat.shape[0]
    #百分之十的测试数据的个数
    numTestVecs = int(m * hoRatio)
    #分类错误计数
    errorCount = 0.0

    for i in range(numTestVecs):
        #前numTestVecs个数据作为测试集,后m-numTestVecs个数据作为训练集
        classifierResult = classify0(normMat[i,:], normMat[numTestVecs:m,:],
            datingLabels[numTestVecs:m], 4)
        print("分类结果:%d\t真实类别:%d" % (classifierResult, datingLabels[i]))
        if classifierResult != datingLabels[i]:
            errorCount += 1.0
    print("错误率:%f%%" %(errorCount/float(numTestVecs)*100))


if __name__ == '__main__':
    datingClassTest()
```

```
>>>
......
......
分类结果:2       真实类别:2
分类结果:2       真实类别:1
分类结果:1       真实类别:1
错误率:4.000000%
```

分类器处理约会数据集的错误率是4%，这是一个相当不错的结果。我们可以改变函数datingClassTest内变量hoRatio和变量k的值，检测错误率是否随着变量值的变化而增加。依赖于分类算法、数据集和程序设置，分类器的输出结果可能有很大的不同。

这个例子表明我们可以正确地预测分类，错误率仅仅是4%。海伦完全可以输入未知对象的属性信息，由分类软件来帮助她判定某一对象的可交往程度：讨厌、一般喜欢、非常喜欢。

5. 使用算法：构建完整可用系统

上面已经在数据上对分类器进行了测试，现在终于可以使用这个分类器为海伦来对人们分类。我们会给海伦一小段程序，通过该程序海伦会在约会网站上找到某个人并输入他的信息。程序会给出她对对方喜欢程度的预测值。

```
import numpy as np
import operator

"""
Parameters:
    inX - 用于分类的数据(测试集)
    dataSet - 用于训练的数据(训练集)
    labes - 分类标签
    k - kNN算法参数,选择距离最小的k个点
Returns:
    sortedClassCount[0][0] - 分类结果
"""
# 函数说明:kNN算法,分类器
def classify0(inX, dataSet, labels, k):
    #numpy函数shape[0]返回dataSet的行数
    dataSetSize = dataSet.shape[0]
    #在列向量方向上重复inX共1次(横向),行向量方向上重复inX共dataSetSize次(纵向)
```

```
    #在列向量方向上重复inX共1次(横向),行向量方向上重复inX共dataSetSize次(纵向)
    diffMat = np.tile(inX, (dataSetSize, 1)) - dataSet
    #二维特征相减后平方
    sqDiffMat = diffMat**2
    #sum()所有元素相加,sum(0)列相加,sum(1)行相加
    sqDistances = sqDiffMat.sum(axis=1)
    #开方,计算出距离
    distances = sqDistances**0.5
    #返回distances中元素从小到大排序后的索引值
    sortedDistIndices = distances.argsort()
    #定一个记录类别次数的字典
    classCount = {}
    for i in range(k):
        #取出前k个元素的类别
        voteIlabel = labels[sortedDistIndices[i]]
        #dict.get(key,default=None),字典的get()方法,返回指定键的值,如果值不在字典中返回默认值。
        #计算类别次数
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
    #python3中用items()替换python2中的iteritems()
    #key=operator.itemgetter(1)根据字典的值进行排序
    #key=operator.itemgetter(0)根据字典的键进行排序
    #reverse降序排序字典
    sortedClassCount = sorted(classCount.items(),key=operator.itemgetter(1),reverse=True)
    #返回次数最多的类别,即所要分类的类别
    return sortedClassCount[0][0]

"""
Parameters:
    filename - 文件名
Returns:
    returnMat - 特征矩阵
    classLabelVector - 分类Label向量
"""
# 函数说明:打开并解析文件，对数据进行分类：1代表不喜欢,2代表魅力一般,3代表极具魅力
def file2matrix(filename):
    #打开文件
    fr = open(filename)
    #读取文件所有内容
    arrayOLines = fr.readlines()
    #得到文件行数
    numberOfLines = len(arrayOLines)
    #返回的NumPy矩阵,解析完成的数据:numberOfLines行,3列
    returnMat = np.zeros((numberOfLines,3))
    #返回的分类标签向量
    classLabelVector = []
    #行的索引值
    index = 0
    for line in arrayOLines:
        #s.strip(rm)，当rm空时,默认删除空白符(包括'\n','\r','\t',' ')
        line = line.strip()
        #使用s.split(str="",num=string,cout(str))将字符串根据'\t'分隔符进行切片。
        listFromLine = line.split('\t')
        #将数据前三列提取出来,存放到returnMat的NumPy矩阵中,也就是特征矩阵
        returnMat[index,:] = listFromLine[0:3]
        #根据文本中标记的喜欢的程度进行分类,1代表不喜欢,2代表魅力一般,3代表极具魅力
        if listFromLine[-1] == 'didntLike':
            classLabelVector.append(1)
        elif listFromLine[-1] == 'smallDoses':
            classLabelVector.append(2)
        elif listFromLine[-1] == 'largeDoses':
            classLabelVector.append(3)
        index += 1
    return returnMat, classLabelVector

"""
Parameters:
    dataSet - 特征矩阵
Returns:
    normDataSet - 归一化后的特征矩阵
    ranges - 数据范围
    minVals - 数据最小值
"""
# 函数说明:对数据进行归一化
def autoNorm(dataSet):
    #获得数据的最小值
```

```
    #获得数据的最小值
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    #最大值和最小值的范围
    ranges = maxVals - minVals
    #shape(dataSet)返回dataSet的矩阵行列数
    normDataSet = np.zeros(np.shape(dataSet))
    #返回dataSet的行数
    m = dataSet.shape[0]
    #原始值减去最小值
    normDataSet = dataSet - np.tile(minVals, (m, 1))
    #除以最大和最小值的差,得到归一化数据
    normDataSet = normDataSet / np.tile(ranges, (m, 1))
    #返回归一化数据结果,数据范围,最小值
    return normDataSet, ranges, minVals

# 函数说明:通过输入一个人的三维特征,进行分类输出
def classifyPerson():
    #输出结果
    resultList = ['讨厌','有些喜欢','非常喜欢']
    #三维特征用户输入
    precentTats = float(input("玩视频游戏所耗时间百分比:"))
    ffMiles = float(input("每年获得的飞行常客里程数:"))
    iceCream = float(input("每周消费的冰激淋公升数:"))
    #打开的文件名
    filename = "datingTestSet.txt"
    #打开并处理数据
    datingDataMat, datingLabels = file2matrix(filename)
    #训练集归一化
    normMat, ranges, minVals = autoNorm(datingDataMat)
    #生成NumPy数组,测试集
    inArr = np.array([precentTats, ffMiles, iceCream])
    #测试集归一化
    norminArr = (inArr - minVals) / ranges
    #返回分类结果
    classifierResult = classify0(norminArr, normMat, datingLabels, 3)
    #打印结果
    print("你可能%s这个人" % (resultList[classifierResult-1]))


if __name__ == '__main__':
    classifyPerson()
```

通过交互式的输入相应的数据，就可以得到想要的结论。

```
>>>
玩视频游戏所耗时间百分比:10
每年获得的飞行常客里程数:10000
每周消费的冰激淋公升数:0.5
你可能讨厌这个人
```

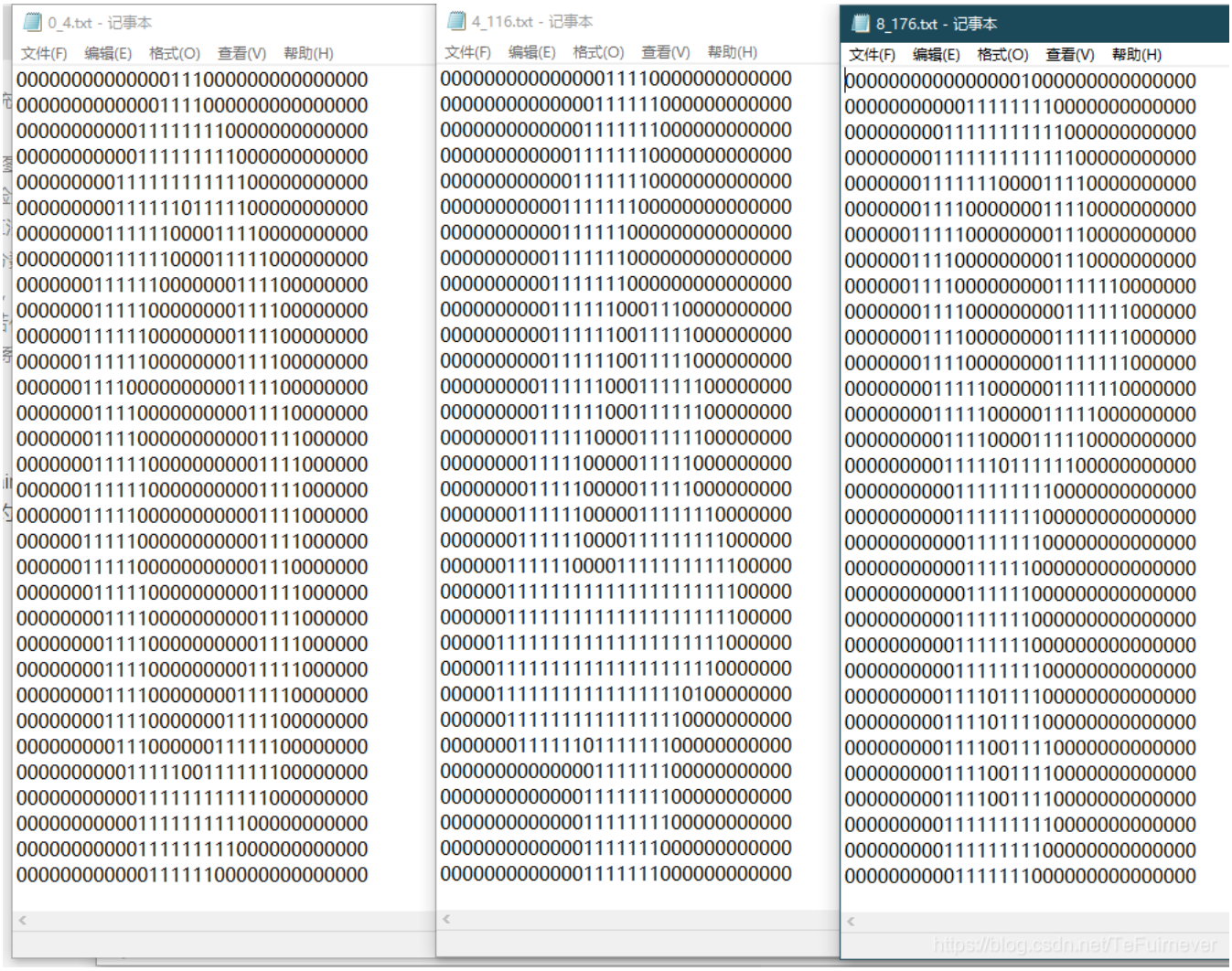
3、Sklearn构建k-近邻分类器用于手写数字识别

> 示例：使用k-近邻算法的手写识别系统
> (1) 收集数据：提供文本文件。
> (2) 准备数据：编写函数classify0()，将图像格式转换为分类器使用的list格式。
> (3) 分析数据：在Python命令提示符中检查数据，确保它符合要求。
> (4) 训练算法：此步骤不适用于k-近邻算法。
> (5) 测试算法：编写函数使用提供的部分数据集作为测试样本，测试样本与非测试样本的区别在于测试样本是已经完成分类的数据，如果预测分类与实际类别不同，则标记为一个错误。
> (6) 使用算法：本例没有完成此步骤，若你感兴趣可以构建完整的应用程序，从图像中提取数字，并完成数字识别，美国的邮件分拣系统就是一个实际运行的类似系统。

1. 准备数据：将图像转换为测试向量

构造的系统只能识别数字0到9，目录trainingDigits中包含了大约2000个例子，每个数字大约有200个样本；目录testDigits中包含了大约900个测试数据。尽管采用文本格式存储图像不能有效地利用内存空间，但是为了方便理解，我们将图片转换为文本格式，数字的文本

格式如图所示。



文件的名字也是有特殊含义的，格式为：数字的值_该数字的样本序号。

2. 测试算法：使用k-近邻算法识别手写数字

把一个32×32的二进制图像矩阵转换为1×1024的向量，这样前两节使用的分类器就可以处理数字图像信息了。

```
import numpy as np
import operator
from os import listdir
from sklearn.neighbors import KNeighborsClassifier as kNN

"""
Parameters:
    filename - 文件名
Returns:
    returnVect - 返回的二进制图像的1x1024向量
"""
# 函数说明:将32x32的二进制图像转换为1x1024向量。
def img2vector(filename):
    #创建1x1024零向量
    returnVect = np.zeros((1, 1024))
    #打开文件
    fr = open(filename)
    #按行读取
    for i in range(32):
        #读一行数据
        lineStr = fr.readline()
        #每一行的前32个元素依次添加到returnVect中
        for j in range(32):
            returnVect[0, 32*i+j] = int(lineStr[j])
    #返回转换后的1x1024向量
```

```
    return returnvect

# 函数说明:手写数字分类测试
def handwritingClassTest():
    #测试集的Labels
    hwLabels = []
    #返回trainingDigits目录下的文件名
    trainingFileList = listdir('trainingDigits')
    #返回文件夹下文件的个数
    m = len(trainingFileList)
    #初始化训练的Mat矩阵,测试集
    trainingMat = np.zeros((m, 1024))
    #从文件名中解析出训练集的类别
    for i in range(m):
        #获得文件的名字
        fileNameStr = trainingFileList[i]
        #获得分类的数字
        classNumber = int(fileNameStr.split('_')[0])
        #将获得的类别添加到hwLabels中
        hwLabels.append(classNumber)
        #将每一个文件的1x1024数据存储到trainingMat矩阵中
        trainingMat[i,:] = img2vector('trainingDigits/%s' % (fileNameStr))
    #构建kNN分类器
    neigh = kNN(n_neighbors = 3, algorithm = 'auto')
    #拟合模型, trainingMat为测试矩阵,hwLabels为对应的标签
    neigh.fit(trainingMat, hwLabels)
    #返回testDigits目录下的文件列表
    testFileList = listdir('testDigits')
    #错误检测计数
    errorCount = 0.0
    #测试数据的数量
    mTest = len(testFileList)
    #从文件中解析出测试集的类别并进行分类测试
    for i in range(mTest):
        #获得文件的名字
        fileNameStr = testFileList[i]
        #获得分类的数字
        classNumber = int(fileNameStr.split('_')[0])
        #获得测试集的1x1024向量,用于训练
        vectorUnderTest = img2vector('testDigits/%s' % (fileNameStr))
        #获得预测结果
        # classifierResult = classify0(vectorUnderTest, trainingMat, hwLabels, 3)
        classifierResult = neigh.predict(vectorUnderTest)
        print("分类返回结果为%d\t真实结果为%d" % (classifierResult, classNumber))
        if(classifierResult != classNumber):
            errorCount += 1.0
    print("总共错了%d个数据\n错误率为%f%%" % (errorCount, errorCount/mTest * 100))


if __name__ == '__main__':
    handwritingClassTest()
```

```
>>>
分类返回结果为9   真实结果为9
分类返回结果为9   真实结果为9
分类返回结果为9   真实结果为9
分类返回结果为9   真实结果为9
总共错了12个数据
错误率为1.268499%
```

上述代码使用的algorithm参数是auto，更改algorithm参数为brute，使用暴力搜索，你会发现，运行时间变长了，变为10s+。更改n_neighbors参数，你会发现，不同的值，检测精度也是不同的。可以尝试更改这些参数的设置，加深对其函数的理解。

4、sklearn.neighbors.KNeighborsClassifier

sklearn.neighbors.KNeighborsClassifier是一个很好的模型，k-近邻算法就是通过它实现的，详细的看这个博客——sklearn.neighbors.KNeighborsClassifier()函数解析

5、总结

k-近邻算法是分类数据最简单最有效的算法，通过两个例子（约会网站和手写数字识别）讲述了如何使用k-近邻算法构造分类器。。k-近邻算法是基于实例的学习，使用算法时我们必须有接近实际数据的训练样本数据。k-近邻算法必须保存全部数据集，如果训练数据集的很大，必须使用大量的存储空间。此外，由于必须对数据集中的每个数据计算距离值，实际使用时可能非常耗时。

k-近邻算法的另一个缺陷是它无法给出任何数据的基础结构信息，因此也无法知晓平均实例样本和典型实例样本具有什么特征。下一章我们将使用概率测量方法处理分类问题，该算法可以解决这个问题。