

第二章 k-近邻算法

2.1 k-近邻法简介

2.1.1 k-近邻的基本原理

k-近邻的工作原理如下:

- 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.
- 一般来说, 我们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.
- 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.

2.1.2 k-近邻的距离度量

k-近邻的距离度量用的是欧氏距离.

设特征空间 \mathcal{X} 是 n 维实数向量空间 \mathbf{R}^n , $x_i, x_j \in \mathcal{X}$, $x_i = (x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(n)})^T$
 $x_j = (x_j^{(1)}, x_j^{(2)}, \dots, x_j^{(n)})^T$. 欧氏距离的计算公式如下:

$$L_2(x_i, x_j) = \left(\sum_{l=1}^n |x_i^{(l)} - x_j^{(l)}|^2 \right)^{\frac{1}{2}}$$

最简单的例子, 两点的距离计算, 如 $A(x_1, x_2), B(y_1, y_2)$, 那么距离 $|AB|$ 为:

$$|AB| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

注1: 欧氏距离的一般形式称为L_p距离, 相关内容参考我前面的<统计学习方法>第三章 k-近邻笔记

2.1.3 k-近邻算法的一般流程

k-近邻算法的一般流程如下:

(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.

那更为确切的关于k-近邻算法的实现步骤如下:

- 计算已知类别数据集中的点与当前点之间的距离;

- 按照距离递增次序排序;
- 选取与当前点距离最小的k个点;
- 确定前k个点所在类别的出现频率;
- 返回前k个点所出现频率最高的类别作为当前点的预测分类。

2.2 简单的k-近邻算法

在先讲具体代码之前, 先说明两点:

- 第一个是, 关于当前代码需要注意的比较大的点
- 第二个是, 关于代码本身, 基本上每行代码都会有注释, 便于理解.

2.2.1 准备简单的数据集

```
import numpy as np
"""
函数说明: 创建数据集
Parameters:
    无
Returns:
    group- 数据集
    labels-分类标签
"""
def createDataSet():
    #创建4x2数据集, 第一列是打斗镜头, 第二列是接吻镜头
    group = np.array([[3,104], [2,100], [101,10], [99,5]])
    #创建标签, 第一个是[3,104]对应的标签-爱情片, 以此类推
    labels = ['爱情片', '爱情片', '动作片', '动作片']
    return group, labels

if __name__ == '__main__':
    #创建数据集, 用group和labels接收函数createDataSet返回的两个变量的值
    group, labels = createDataSet()
    #查看数据集的结果
    print(group)
    print(labels)
```

运行结果如下:

```
In [9]: runfile('C:/Users/HeatonHsu/.spyder-py3/temp.py', wdir='C:/Users/
HeatonHsu/.spyder-py3')
[[ 3 104]
 [ 2 100]
 [101  10]
 [ 99   5]]
['爱情片', '爱情片', '动作片', '动作片']

In [10]:
```

这段代码需要注意的点有:

1. 关于 `if __name__ == '__main__':` 的理解

最简单的理解如下:

当.py文件本身被直接运行时, `if __name__ == '__main__':` 之下的代码块将被运行; 当.py文件以模块形式被导入时, `if __name__ == '__main__':` 之下的代码块不被运行.

2.2.2 k-近邻算法

根据两点距离公式, 计算距离, 选择距离最小的前k个点, 并返回分类结果. 详细见前面的实现步骤.

```
import numpy as np
import operator
"""
函数说明: 创建数据集
Parameters:
    无
Returns:
    group- 数据集
    labels-分类标签
"""
def createDataSet():
    #创建4x2数据集, 第一列是打斗镜头,第二列是接吻镜头
    group = np.array([[3,104], [2,100], [101,10], [99,5]])
    #创建标签, 第一个是[3,104]对应的标签-爱情片,以此类推
    labels = ['爱情片', '爱情片', '动作片', '动作片']
    return group,labels

"""
函数说明: KNN算法, 分类器

Parameters:
    inX - 用于分类数据(即测试集)
    dataSet - 用于训练的数据(即训练集)
    labels - 分类的标签(即什么类型的电影)
    k - KNN算法参数, 选择距离最小的k个点
Returns:
    sortedClassCount[0][0] - 分类结果
"""

def classify0(inX, dataSet, labels, k):
    #shape[0]返回的是dataSet的行数,比如前面的group是4x2, 那么shape[0]就是4
    dataSetSize = dataSet.shape[0]
    #np.tile(a, (b,c))函数含义是在列向量方向上重复a共b次(横向), 在行向量方向上重复a共c次(纵向)
    #np.tile(inX, (dataSetSize,1))的含义就是构建dataSetSize行inX,接着减去DataSet对应的.
    diffMat = np.tile(inX, (dataSetSize,1)) - dataSet
    #求得每一行差额的平方
    sqDiffMat = diffMat**2
    #然后求和, axis=1是按行求和,axis=0是按列求和
    sqDistances = sqDiffMat.sum(axis=1)
    #开平方,求得距离
    distances = sqDistances**0.5
    #numpy中的argsort()方法(也是numpy的顶级函数),返回distances中元素从小到大排列后的索引值.
```

```

#这两者是等价的:a.argsort()==np.argsort(a)
#sortedDistIndices返回的是类似于(3,0,1,2)
sortedDistIndices = distances.argsort()
#构建一个空的字典,用于记录类别次数
classCount = {}
for i in range(k):
    #提取出前k个元素的类别
    voteLabel = labels[sortedDistIndices[i]]
    #字典的get方法,返回指定键的值,如果指定键不存在,那么返回默认值,这里是0.
    #比如键是"爱情片",最开始字典是空的,那么返回0,后面加了1,那么就是{"爱情片":1}
    #接着如果还是"爱情片",那么返回1,再加1,就是2,那么字典变成了{"爱情片":2}
    classCount[voteLabel] = classCount.get(voteLabel, 0) + 1
#sorted函数里参数key,是根据一定的方式排序.
#这里的key=operator.itemgetter(1)表示的是根据字典的值进行排序
#参数reverse表示是否进行降序排序,True表示是,默认否,即false
sortedClassCount =
sorted(classCount.items(),key=operator.itemgetter(1),reverse=True)
#则第一个字典的键就是[0][0],也就是返回次数最多的类别,就是我们要的最终类别
return sortedClassCount[0][0]

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

```

这段代码需要注意的点有:

1. np.tile(a,(b,c))含义, 如果想深入了解np.tile的含义, 可以参考: <https://www.jianshu.com/p/4b74a367833c>
2. sqDiffMat.sum(axis=1), axis=1是按行求和,axis=0是按列求和
3. argsort函数: numpy中的argsort()方法(也是numpy的顶级函数),返回distances中元素从小到大排列后的索引值.这两者是等价的:a.argsort()==np.argsort(a)
4. 字典的get方法: 字典的get方法,返回指定键的值,如果指定键不存在,那么返回默认值,这里是0.
5. sorted函数: sorted(iterable, cmp=None, key=None, reverse=False)

2.3 k-近邻算法之约会网站配对

海伦一直使用在线约会网站寻找适合自己的约会对象. 尽管约会网站会推荐不同的人选, 但她并不是喜欢每一个人. 经过一番总结, 她发现曾交往过三种类型的人:

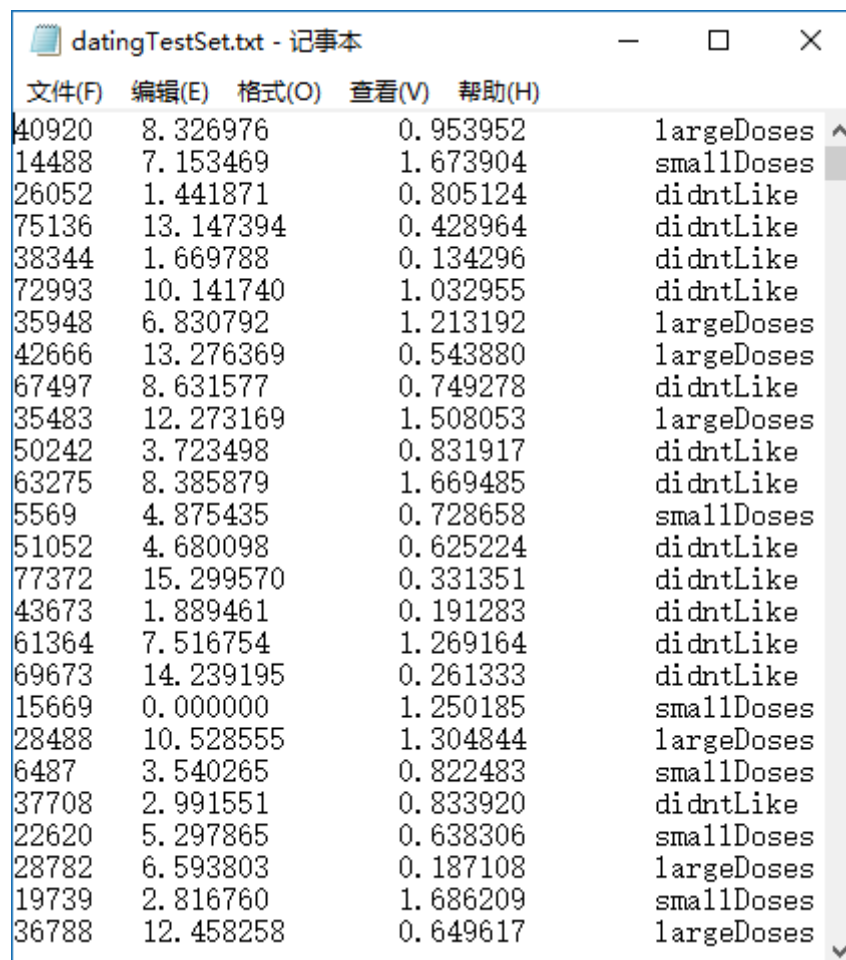
- 不喜欢的人
- 魅力一般的人

- 极具魅力的人

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](#)(点击下载)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

预览一下datingTestSet.txt数据集, 如下:



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
19739	2.816760	1.686209	smallDoses
36788	12.458258	0.649617	largeDoses

共四列数据, 分别对应: 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和三种类型的人.

2.3.1 数据解析

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. 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_Hsu02.py中创建名为file2matrix的函数. 该函数就是将文件转换成矩阵的形式.

```
import numpy as np
```

```
.....
```

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

Parameters:

filename - 文件名

Returns:

returnMat - 特征矩阵

classLabelVector - 分类label向量

"""

```
def file2matrix(filename):
```

```
    #打开文件
```

```
    with open(filename) as filename_small:
```

```
        #读取文件的所有内容.其中,readlines是从文件中读取每一行, 并将其存储在一个列表
    中
```

```
        array0lines = filename_small.readlines()
```

```
    #得到文件的行数
```

```
    number0Lines = len(array0lines)
```

```
    #生成一个numpy array,其中,number0Lines行(一行是一个训练数据),3列(因为是3个特征)
```

```
    returnMat = np.zeros((number0Lines,3))
```

```
    #返回的分类标签向量
```

```
    classLabelVector = []
```

```
    #行的索引
```

```
    index = 0
```

```
    for line in array0lines:
```

```
        #strip()是删除前后空白
```

```
        line = line.strip()
```

```
        #split('example')方法是根据example进行分割,最后存储在列表中
```

```
        listFromline = line.split('\t')
```

```
        #将数据的前三列提取处理,存放在returnMatd的array中,也就是特征array
```

```
        returnMat[index,:] = listFromline[0:3]
```

```
        #根据文本标记中标记的喜欢的成都进行分类,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)
```

```
        #索引每次加1,一直到index=999,最后加一就是1000,共1000行
```

```
        index +=1
```

```
    return returnMat, classLabelVector
```

```
if __name__ == '__main__':
```

```
    #打开的文件名
```

```
    filename = "datingTestSet.txt"
```

```
    #打开并处理数据集
```

```
    datingDataMat, datingLabels = file2matrix(filename)
```

```
    print(datingDataMat)
```

```
    print(datingLabels)
```

运行结果如下:


```

with open(filename) as filename_small:
    #读取文件的所有内容.其中,readline方法是从小文件中读取每一行, 并将其存储在一个列表中
    array0lines = filename_small.readlines()
#得到文件的行数
number0Lines = len(array0lines)
#返回得到numpy array,其中,number0Lines行,3列(因为是3个特征)
returnMat = np.zeros((number0Lines,3))
#返回的分类标签向量
classLabelVector = []
#行的索引
index = 0
for line in array0lines:
    #strip()是删除前后空白
    line = line.strip()
    #split('example')方法是根据example进行分割,最后存储在列表中
    listFromline = line.split('\t')
    #将数据的前三列提取处理啊,存放在returnMatd的array中,也就是特征array
    returnMat[index,:] = listFromline[0:3]
    #根据文本标记中标记的喜欢的成都进行分类,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)
    #索引每次加1
    index +=1
return returnMat, classLabelVector

"""
函数说明:数据可视化
Parameters:
    datingDataMat - 特征矩阵
    datingLabels - 分类标签
Returns:
    无
"""

def showdatas(datingDataMat, datingLabels):
    #plt.subplot()和fig = plt.figure(), axes = fig.subplot()是等价的,具体请参考<利用
    Python进行数据分析>
    fig, axs = plt.subplots(nrows=2, ncols=2, sharex=False, sharey=False,
figsize=(13,8))
    LabelsColors = []
    #把1,2,3分别赋予黑,黄和红,并对应存储在Labels Colors列表中,其实就是把datingLabels中的
    1,2,3转换成颜色存储在LabelsColors中
    for i in datingLabels:
        if i == 1:
            LabelsColors.append('black')
        elif i == 2:
            LabelsColors.append('orange')
        elif i == 3:
            LabelsColors.append('red')
    #第一块画布,画出以非常客路程为x轴(datingDataMat第一列数据), 以玩游戏为y轴
    (datingDataMat第二列数据),颜色以对应数字代表的颜色.
    axs[0,0].scatter(x=datingDataMat[:,0],y=datingDataMat[:,1],
color=LabelsColors, s=15)
    #设置x轴和y轴的标题
    axs[0,0].set_title("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")

```



```

axs[0,0].set_xlabel("每年获得的飞行常客里程数")
axs[0,0].set_ylabel("玩视频游戏所消耗时间占比")

#第二块画布,画出以飞行常客路程为x轴(datingDataMat第一列数据),以冰激凌为y轴
(datingDataMat第三列数据)
axs[0, 1].scatter(x=datingDataMat[:, 0], y=datingDataMat[:, 2],
color=LabelsColors, s=15)
# 设置x轴和y轴的标题
axs[0, 1].set_title("每年获得的飞行常客里程数与每周冰激凌消费量")
axs[0, 1].set_xlabel("每年获得的飞行常客里程数")
axs[0, 1].set_ylabel("每周消费冰激凌量")

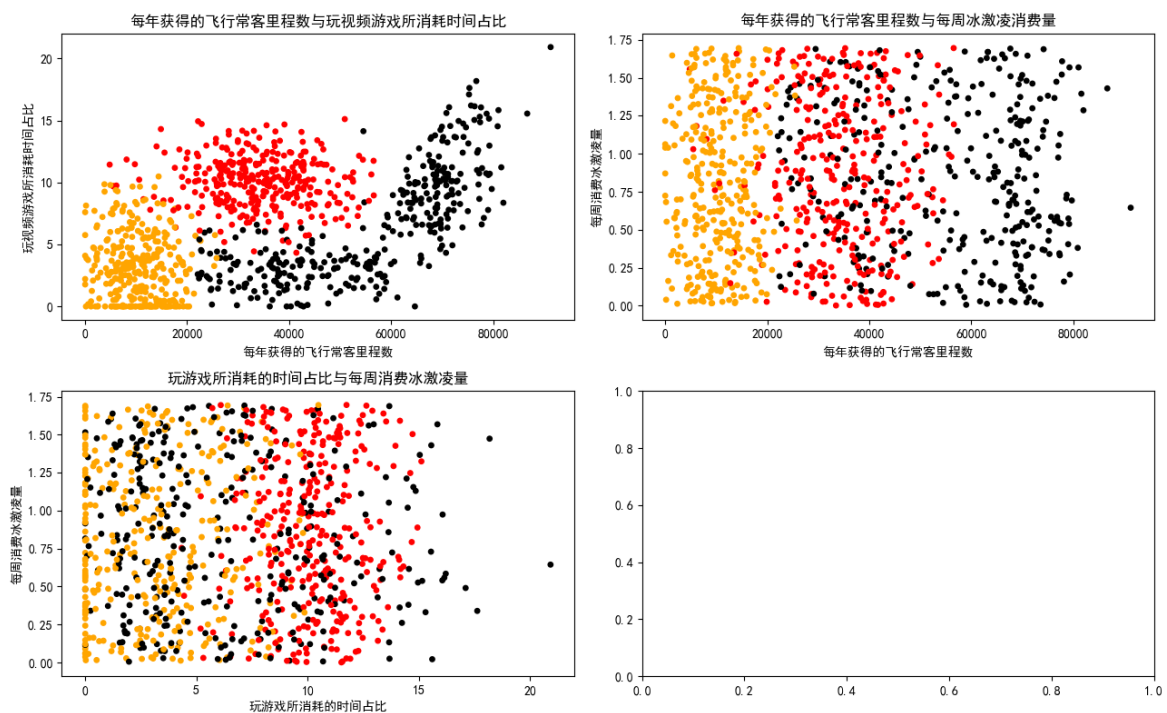
#第三块画布, 画出以玩游戏为x轴(datingDataMat第二列数据), 以冰激凌为y轴(datingDataMat
第三列数据)
axs[1, 0].scatter(x=datingDataMat[:, 1], y=datingDataMat[:, 2],
color=LabelsColors, s=15)
# 设置x轴和y轴的标题
axs[1, 0].set_title("玩游戏所消耗的时间占比与每周消费冰激凌量")
axs[1, 0].set_xlabel("玩游戏所消耗的时间占比")
axs[1, 0].set_ylabel("每周消费冰激凌量")

plt.show()

if __name__ == '__main__':
    #打开文件名
    filename = 'datingTestSet.txt'
    datingDataMat, datingLabels = file2matrix(filename)
    showdatas(datingDataMat,datingLabels)

```

效果如下:



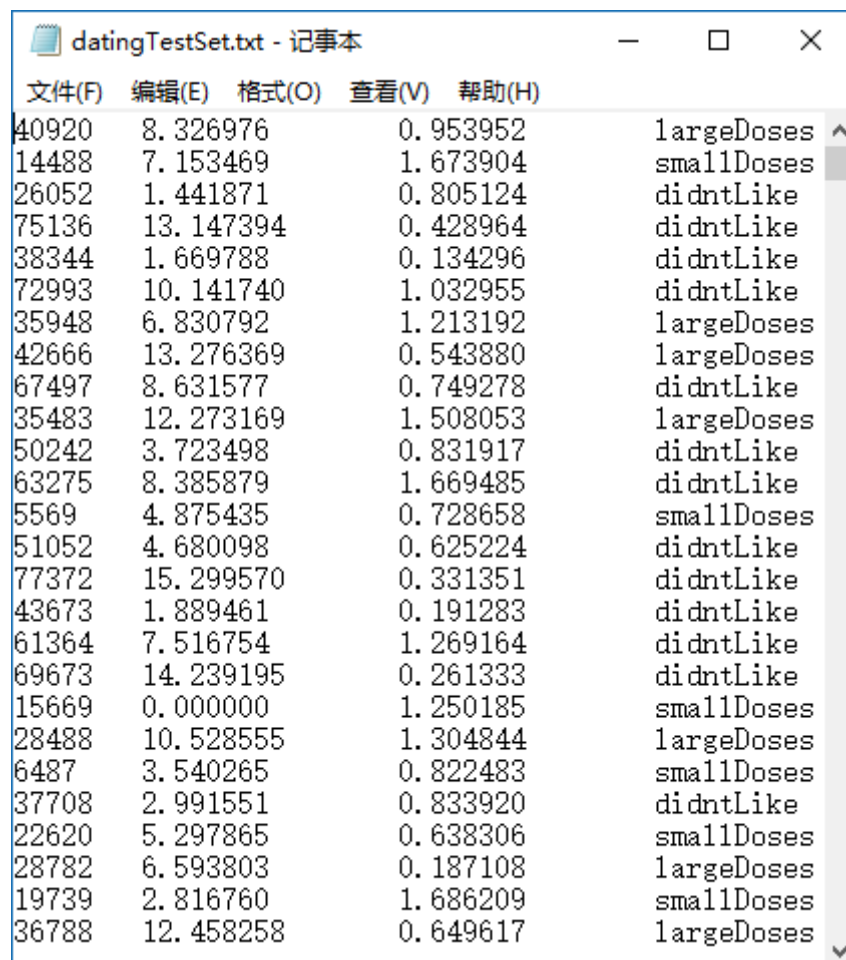
可以从图中, 直观的看出散点的分布情况

这段代码需要注意的点有:

1. plt.subplot()和fig = plt.figure(), axes = fig.subplot()是等价的,具体请参考<利用Python进行数据分析>
2. 其他的画图命令和参数详细可参考相关书籍, 如<利用Python进行数据分析>

2.3.3 数据归一化

还是上面的datingTestSet.txt数据集



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
19739	2.816760	1.686209	smallDoses
36788	12.458258	0.649617	largeDoses

如果我没要计算样本1(第一行)和样本2(第二行)的距离, 那么根据欧氏距离计算公式, 我没可以得到距离为:

$$\sqrt{(40920 - 14488)^2 + (8.326976 - 7.153469)^2 + (0.953952 - 1.673904)^2}$$

易发现, 上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.

在处理这种不同取值范围的特征值时, 我们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:

$$\text{newValue} = (\text{oldValue} - \text{min}) / (\text{max} - \text{min})$$

其中min和max分别是数据集中的最小特征值和最大特征值.

在kNN_Hsu02.py中编写一个名为autoNorm的函数, 用该函数将数据归一化.

具体代码如下:

```
import numpy as np

"""
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

Parameters:
    filename - 文件名
Returns:
    returnMat - 特征举证
    classLabelVector - 分类label向量
"""

def file2matrix(filename):
    #打开文件
    with open(filename) as filename_small:
        #读取文件的所有内容.其中,readline是方法是从文件中读取每一行, 并将其存储在一个列表中
        array0lines = filename_small.readlines()
    #得到文件的行数
    number0Lines = len(array0lines)
    #返回得到numpy array,其中,number0Lines行,3列(因为是3个特征)
    returnMat = np.zeros((number0Lines,3))
    #返回的分类标签向量
    classLabelVector = []
    #行的索引
    index = 0
    for line in array0lines:
        #strip()是删除前后空白
        line = line.strip()
        #split('example')方法是根据example进行分割,最后存储在列表中
        listFromline = line.split('\t')
        #将数据的前三列提取处理啊,存放在returnMatd的array中,也就是特征array
        returnMat[index,:] = listFromline[0:3]
        #根据文本标记中标记的喜欢的成都进行分类,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)
        #索引每次加1
        index +=1
    return returnMat, classLabelVector

"""
函数说明:对数据进行归一化

Parameters:
    dataSet - 特征矩阵
Returns:
    normDataSet - 归一化后的特征矩阵
    ranges - 数据的范围
    minVals - 数据的最小值
"""
```

```

"""
def autoNorm(dataSet):
    #获取每列数据的最小值和最大值(也就是每列的属性)
    minVals = dataSet.min(axis=0)
    maxvals = dataSet.max(axis=0)
    #最大值和最小值的范围,也就是差值
    ranges = maxvals - minVals
    #np.shape(dataSet)返回dataSet的行列数,也可以这样写:dataSet.shape,效果一样.
    #然后构建具有dataSet行列数的0矩阵,用于后续的填充
    normDataSet = np.zeros(np.shape(dataSet))
    #返回dataSet的行数
    m = dataSet.shape[0]
    #用原始值减去最小值
    normDataSet = dataSet - np.tile(minVals, (m,1))
    #归一化,也就是除以最大值和最小值的差
    normDataSet = normDataSet / np.tile(ranges, (m,1))
    #最后返回归一化数据的结果,数据范围和数据的最小值
    return normDataSet, ranges, minVals

if __name__ == '__main__':
    #打开的文件名
    filename = "datingTestSet.txt"
    #打开文件并处理数据
    datingDataMat, datingLabels = file2matrix(filename)
    normDataSet, ranges, minVals = autoNorm(datingDataMat)
    print(normDataSet)
    print(ranges)
    print(minVals)

```

结果如下:

Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

```

In [1]: runfile('C:/Users/HeatonHsu/.spyder-py3/temp.py', wdir='C:/Users/HeatonHsu/.spyder-py3')
[[0.44832535 0.39805139 0.56233353]
 [0.15873259 0.34195467 0.98724416]
 [0.28542943 0.06892523 0.47449629]
 ...
 [0.29115949 0.50910294 0.51079493]
 [0.52711097 0.43665451 0.4290048 ]
 [0.47940793 0.3768091 0.78571804]]
[9.1273000e+04 2.0919349e+01 1.6943610e+00]
[0.      0.      0.001156]

```

In [2]: |

这段代码需要注意的点有:

本段代码没有特别的地方,主要是注意整个代码的逻辑.

2.3.4 测试算法: 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. 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, 所以我们可以随意选择10%数据而不影响其随机性.

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.

下面的代码中, 定义了一个计数器变量, 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.

为了测试分类器的效果, 在kNN_Hsu02.py中创建函数datingClassTest, 具体代码如下:

```
import numpy as np
import operator

"""
函数说明: KNN算法, 分类器

Parameters:
    inX - 用于分类数据(即测试集)
    dataSet - 用于训练的数据(即训练集)
    labels - 分类的标签(即什么类型的电影)
    k - KNN算法参数, 选择距离最小的k个点
Returns:
    sortedClassCount[0][0] - 分类结果
"""

def classify0(inX, dataSet, labels, k):
    #shape[0]返回的是dataSet的函数
    dataSetSize = dataSet.shape[0]
    #np.tile(a,(b,c))函数含义是在列向量方向上重复a共b次(横向), 在行向量方向上重复a共c次(纵向)
    #np.tile(inX, (dataSetSize,1))的含义就是构建dataSetSize行inX,接着减去DataSet对应的.
    diffMat = np.tile(inX, (dataSetSize,1)) - dataSet
    #求得每一行差额的平方
    sqDiffMat = diffMat**2
    #然后求和, axis=1是按行求和,axis=0是按列求和
    sqDistances = sqDiffMat.sum(axis=1)
    #开平方,求得距离
    distances = sqDistances**0.5
    #numpy中的argsort()方法(也是numpy的顶级函数),返回distances中元素从小到大排列后的索引值.
    #这两者是等价的:a.argsort()==np.argsort(a)
    sortedDistIndices = distances.argsort()
    #构建一个空的字典,用于记录类别次数
    classCount = {}
    for i in range(k):
```

```

        #提取出前k个元素的类别
        voteIlabel = labels[sortedDistIndices[i]]
        #字典的get方法,返回指定键的值,如果指定键不存在,那么返回默认值,这里是0.
        #比如键是"爱情片",最开始字典是空的,那么返回0,后面加了1,那么就是{"爱情片":1}
        #接着如果还是"爱情片",那么返回1,再加1,就是2,那么字典变成了{"爱情片":2}
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1
    #sorted函数,里面有参数key,根据一定的方式排序.
    #这里的key=operator.itemgetter(1)表示的是根据字典的值进行排序
    #参数reverse表示是否进行降序排序,True表示是,默认否,即false
    sortedClassCount = sorted(classCount.items(),
key=operator.itemgetter(1),reverse=True)
    #则第一个字典的键就是[0][0],也就是返回次数最多的类别,就是我们要的最终类别
    return sortedClassCount[0][0]

"""

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

Parameters:
    filename - 文件名
Returns:
    returnMat - 特征举证
    classLabelVector - 分类label向量
"""

def file2matrix(filename):
    #打开文件
    with open(filename) as filename_small:
        #读取文件的所有内容.其中,readline是方法是从文件中读取每一行,并将其存储在一个列表中
        array0lines = filename_small.readlines()
    #得到文件的行数
    number0Lines = len(array0lines)
    #返回得到numpy array,其中,number0Lines行,3列(因为是3个特征)
    returnMat = np.zeros((number0Lines,3))
    #返回的分类标签向量
    classLabelVector = []
    #行的索引
    index = 0
    for line in array0lines:
        #strip()是删除前后空白
        line = line.strip()
        #split('example')方法是根据example进行分割,最后存储在列表中
        listFromline = line.split('\t')
        #将数据的前三列提取处理啊,存放在returnMatd的array中,也就是特征array
        returnMat[index,:] = listFromline[0:3]
        #根据文本标记中标记的喜欢的成都进行分类,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)
        #索引每次加1
        index +=1
    return returnMat, classLabelVector

"""

函数说明:对数据进行归一化

```

```

Parameters:
    dataSet - 特征矩阵
Returns:
    normDataSet - 归一化后的特征矩阵
    ranges - 数据的范围
    minVals - 数据的最小值
"""
def autoNorm(dataSet):
    #获取每列数据的最小值和最大值(也就是每列的属性)
    minVals = dataSet.min(axis=0)
    maxVals = dataSet.max(axis=0)
    #最大值和最小值的范围,也就是差值
    ranges = maxVals - minVals
    #np.shape(dataSet)返回dataSet的行列数,也可以这样写: dataSet.shape,效果一样.
    #然后构建具有dataSet行列数的0矩阵,用于后续的填充
    normDataSet = np.zeros(np.shape(dataSet))
    #返回dataSet的行数
    m = dataSet.shape[0]
    #用原始值减去最小值
    normDataSet = dataSet - np.tile(minVals, (m,1))
    #归一化,也就是除以最大值和最小值的差
    normDataSet = normDataSet / np.tile(ranges, (m,1))
    #最后返回归一化数据的结果,数据范围和数据的最小值
    return normDataSet, ranges, minVals

"""
函数说明:分类器的测试函数
Parameters:
    无
Returns:
    noumDataSet - 归一化后的特征矩阵
    ranges - 数据范围
    minVals - 数据最小值
"""

def datingClassTest(normDataSet):
    #取所有数据的百分之十用于测试
    hoRatio = 0.1
    #获取normDataSet的行数
    m = normDataSet.shape[0]
    #百分之十的测试数据的个数
    numTestVecs = int(m * hoRatio)
    #分类错误计数
    erroCount = 0
    #分类错误的计算
    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
        #同时这里注意, k-近邻没有显式的训练过程,不需要训练,所以此处只需要选择前10%直接验证即可
        #normDataSet[i,:]表示的就是第i行数据(用于测试)
        #normDataSet[numTestVecs:m]就是第numTestVecs行到第m行的数据
        classifierResult = classify0(normDataSet[i,:],
normDataSet[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]:
            erroCount += 1

```

```
print("错误率:%f%%" % (erroCount/float(numTestVecs)*100))
```

```
if __name__ == '__main__':  
    #打开文件名  
    filename = 'datingTestSet.txt'  
    #打开文件并处理数据  
    datingDataMat, datingLabels = file2matrix(filename)  
    normDataSet, ranges, minVals = autoNorm(datingDataMat)  
    datingClassTest(normDataSet)
```

结果如下:

```
kNN_Hsu02 x  
分类结果: 1 真实类别:1  
分类结果: 3 真实类别:3  
分类结果: 3 真实类别:3  
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%  
  
Process finished with exit code 0
```

这段代码需要注意的点有:

本段代码没有特别的地方,主要是增加了一个datingClassTest函数,然后再把前面的代码整合在一起,.

2.3.5 使用算法: 构建完整可用系统

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.接着程序会给出她对对方喜欢程度的预测值.也就是上面的三种标签.

在kNN_Hsu02.py中创建函数classifyPerson,具体代码如下:

```
import numpy as np  
import operator
```

```
"""
```

函数说明: KNN算法, 分类器

Parameters:

inX - 用于分类数据(即测试集)


```

    dataSet - 用于训练的数据(即训练集)
    labels - 分类的标签(即什么类型的电影)
    k - KNN算法参数, 选择距离最小的k个点
Returns:
    sortedClassCount[0][0] - 分类结果
"""

def classify0(inX, dataSet, labels, k):
    #shape[0]返回的是dataSet的函数
    dataSetSize = dataSet.shape[0]
    #np.tile(a,(b,c))函数含义是在列向量方向上重复a共b次(横向), 在行向量方向上重复a共c次(纵向)
    #np.tile(inX, (dataSetSize,1))的含义就是构建dataSetSize行inX,接着减去DataSet对应的.
    diffMat = np.tile(inX, (dataSetSize,1)) - dataSet
    #求得每一行差额的平方
    sqDiffMat = diffMat**2
    #然后求和, axis=1是按行求和,axis=0是按列求和
    sqDistances = sqDiffMat.sum(axis=1)
    #开平方,求得距离
    distances = sqDistances**0.5
    #numpy中的argsort()方法(也是numpy的顶级函数),返回distances中元素从小到大排列后的索引值.等价:a.argsort()==np.argsort(a)
    sortedDistIndices = distances.argsort()
    #构建一个空的字典,用于记录类别次数
    classCount = {}
    for i in range(k):
        #提取出前k个元素的类别
        voteIlabel = labels[sortedDistIndices[i]]
        #字典的get方法,返回指定键的值,如果指定键不存在,那么返回默认值,这里是0.
        #比如键是"爱情片",最开始字典是空的,那么返回0,后面加了1,那么就是{"爱情片":1}
        #接着如果还是"爱情片",那么返回1,再加1,就是2,那么字典变成了{"爱情片":2}
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1
    #sorted函数,里面有参数key,根据一定的方式排序.这里的key=operator.itemgetter(1)表示的是根据字典的值进行排序
    #参数reverse表示是否进行降序排序,True表示是,默认否,即false
    sortedClassCount = sorted(classCount.items(),
key=operator.itemgetter(1),reverse=True)
    #则第一个字典的键就是[0][0],也就是返回次数最多的类别,就是我们要的最终类别
    return sortedClassCount[0][0]

"""
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

Parameters:
    filename - 文件名
Returns:
    returnMat - 特征举证
    classLabelVector - 分类label向量
"""

def file2matrix(filename):
    #打开文件
    with open(filename) as filename_small:
        #读取文件的所有内容.其中,readline是方法是从文件中读取每一行,并将其存储在一个列表中
        array0lines = filename_small.readlines()
    #得到文件的行数
    number0Lines = len(array0lines)

```

```

#返回得到numpy array,其中,number0Lines行,3列(因为是3个特征)
returnMat = np.zeros((number0Lines,3))
#返回的分类标签向量
classLabelVector = []
#行的索引
index = 0
for line in array0lines:
    #strip()是删除前后空白
    line = line.strip()
    #split('example')方法是根据example进行分割,最后存储在列表中
    listFromline = line.split('\t')
    #将数据的前三列提取处理啊,存放在returnMatd的array中,也就是特征array
    returnMat[index,:] = listFromline[0:3]
    #根据文本标记中标记的喜欢的成都进行分类,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)
    #索引每次加1
    index +=1
return returnMat, classLabelVector

```

"""

函数说明:对数据进行归一化

Parameters:

dataSet - 特征矩阵

Returns:

normDataSet - 归一化后的特征矩阵

ranges - 数据的范围

minVals - 数据的最小值

"""

def autoNorm(dataSet):

#获取每列数据的最小值和最大值(也就是每列的属性)

minVals = dataSet.min(axis=0)

maxvals = dataSet.max(axis=0)

#最大值和最小值的范围,也就是差值

ranges = maxvals - minVals

#np.shape(dataSet)返回dataSet的行列数,也可以这样写:dataSet.shape,效果一样.

#然后构建具有dataSet行列数的0矩阵,用于后续的填充

normDataSet = np.zeros(np.shape(dataSet))

#返回dataSet的行数

m = dataSet.shape[0]

#用原始值减去最小值

normDataSet = dataSet - np.tile(minVals, (m,1))

#归一化,也就是除以最大值和最小值的差

normDataSet = normDataSet / np.tile(ranges, (m,1))

#最后返回归一化数据的结果,数据范围和数据的最小值

return normDataSet, ranges, minVals

"""

函数说明:通过输入一个人的数据,进行分类输出

Parameters:

无

Returns:

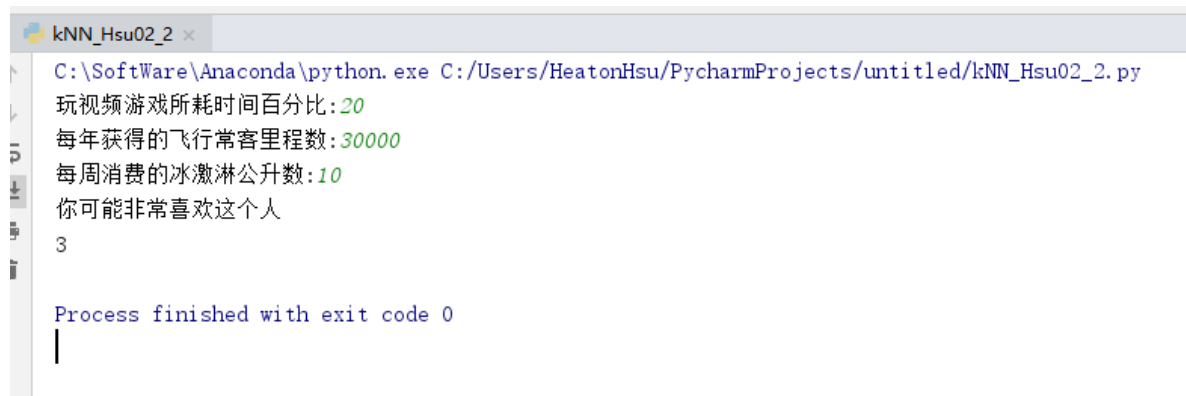
```

无
"""
def classifyPerson():
    #输出结果
    resultList = ['讨厌', '有些喜欢', '非常喜欢']
    #输入一个人的特征信息
    precentTat = 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:'))
    # 打开文件名
    filename = 'datingTestSet.txt'
    # 打开文件并处理数据
    datingDataMat, datingLabels = file2matrix(filename)
    normDataSet, ranges, minVals = autoNorm(datingDataMat)
    #生成数组, 测试集
    inArr = np.array([ffMiles, precentTat, iceCream])
    #对测试集进行归一化
    norminArr = (inArr-minVals) / ranges
    #返回结果分类
    classifierResult = classify0(norminArr, normDataSet, datingLabels,3)
    #打印结果
    print('你可能%s这个人' % (resultList[classifierResult-1]))

if __name__ == '__main__':
    classifyPerson()

```

运行结果:



```

kNN_Hsu02_2 x
C:\SoftWare\Anaconda\python.exe C:/Users/HeatonHsu/PycharmProjects/untitled/kNN_Hsu02_2.py
玩视频游戏所耗时间百分比:20
每年获得的飞行常客里程数:30000
每周消费的冰激淋公升数:10
你可能非常喜欢这个人
3

Process finished with exit code 0

```

这个笔记先就写到这里, 下一节写一下关于手写识别的东西, 并且一并学习下强大的sklearn库中关于k-近邻模块的使用.