数据科学 6 机器学习：k-近邻算法

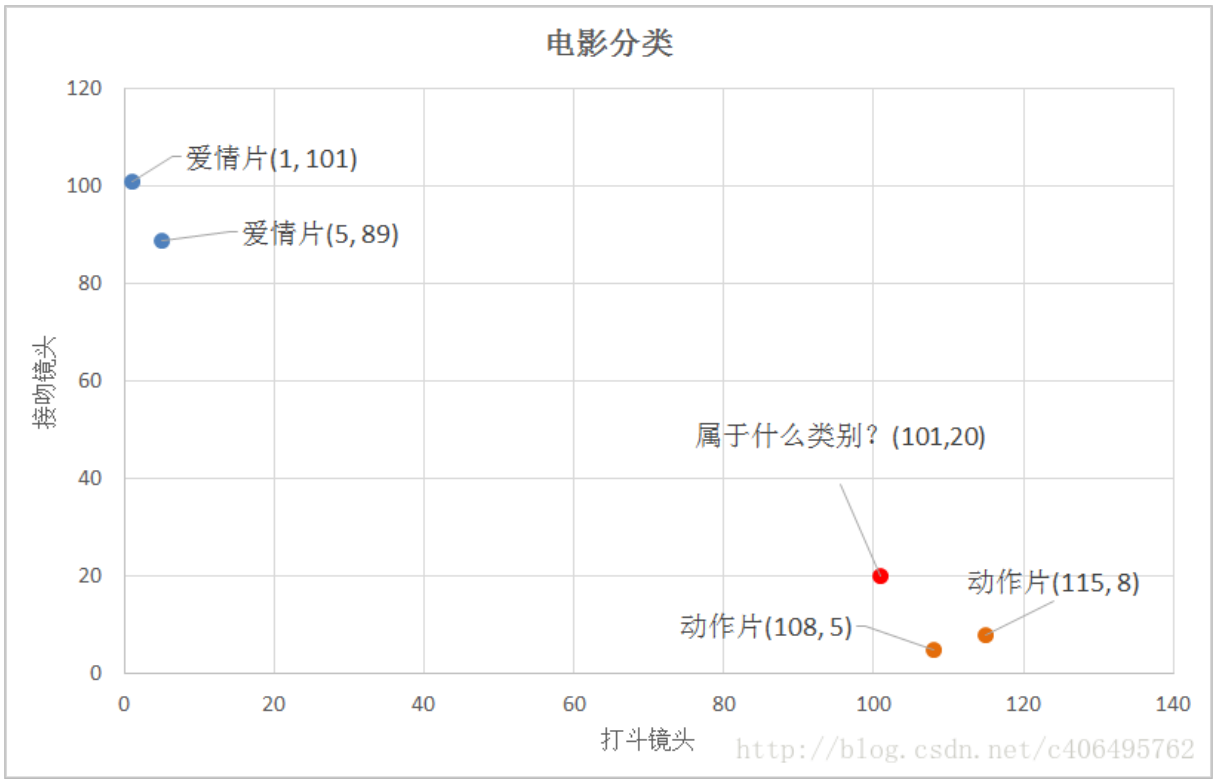
k-近邻法简介

k近邻法(k-nearest neighbor, k-NN)是1967年由Cover T和Hart P提出的一种基本分类与回归方法。它的工作原理是：存在一个样本数据集合，也称作为训练样本集，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个数据与所属分类的对应关系。输入没有标签的新数据后，将新的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。

电影名称	打斗镜头	接吻镜头	电影类型
电影1	1	101	爱情片
电影2	5	89	爱情片
电影3	108	5	动作片
电影4	115	8	动作片

表1.1 每部电影的打斗镜头数、接吻镜头数以及电影类型

电影打斗镜头数为49，接吻镜头数为51
人的判断　k-近邻算法的判断



k-近邻算法用距离进行度量

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

图1.2 两点距离公式

- (101,20)->动作片(108,5)的距离约为16.55
- (101,20)->动作片(115,8)的距离约为18.44
- (101,20)->爱情片(5,89)的距离约为118.22
- (101,20)->爱情片(1,101)的距离约为128.69

红色圆点标记的电影到动作片 (108,5)的距离最近，为16.55。如果算法直接根据这个结果，判断该红色圆点标记的电影为动作片，这个算法就是最近邻算法，而非k-近邻算法

k-近邻算法步骤如下：

- 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。

比如，现在我这个k值取3，那么在电影例子中，按距离依次排序的三个点分别是动作片(108,5)、动作片(115,8)、爱情片(5,89)。在这三个点中，动作片出现的频率为三分之二，爱情片出现的频率为三分之一，所以该红色圆点标记的电影为动作片。这个判别过程就是k-近邻算法。

准备数据集

```
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():
    #四组二维特征
    group = np.array([[1,101],[5,89],[108,5],[115,8]])
    #四组特征的标签
    labels = ['爱情片','爱情片','动作片','动作片']
    return group, labels
```

```
import numpy as np
import operator

"""
函数说明:kNN算法,分类器
```

```
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
"""
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
        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]
```

预测红色圆点标记的电影(101，20)的类别，K-NN的k值为3

```
#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)
```

多个特征点，可以用欧氏距离(也称欧几里德度量)

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

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

k-近邻算法实战之约会网站配对效果判定

k-近邻算法的一般流程：

- 收集数据：可以使用爬虫进行数据的收集，也可以使用第三方提供的免费或收费的数据。一般来讲，数据放在txt文本文件中，按照一定的格式进行存储，便于解析及处理。
- 准备数据：使用Python解析、预处理数据。
- 分析数据：可以使用很多方法对数据进行分析，例如使用Matplotlib将数据可视化。
- 测试算法：计算错误率。
- 使用算法：错误率在可接受范围内，就可以运行k-近邻算法进行分类。

海伦女士一直使用在线约会网站寻找适合自己的约会对象

她发现自己交往过的人可以进行如下分类：

- 不喜欢的人
- 魅力一般的人
- 极具魅力的人

海伦收集约会数据存放在文本文件datingTestSet.txt

样本数据主要包含以下3种特征：

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

```
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
    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
```

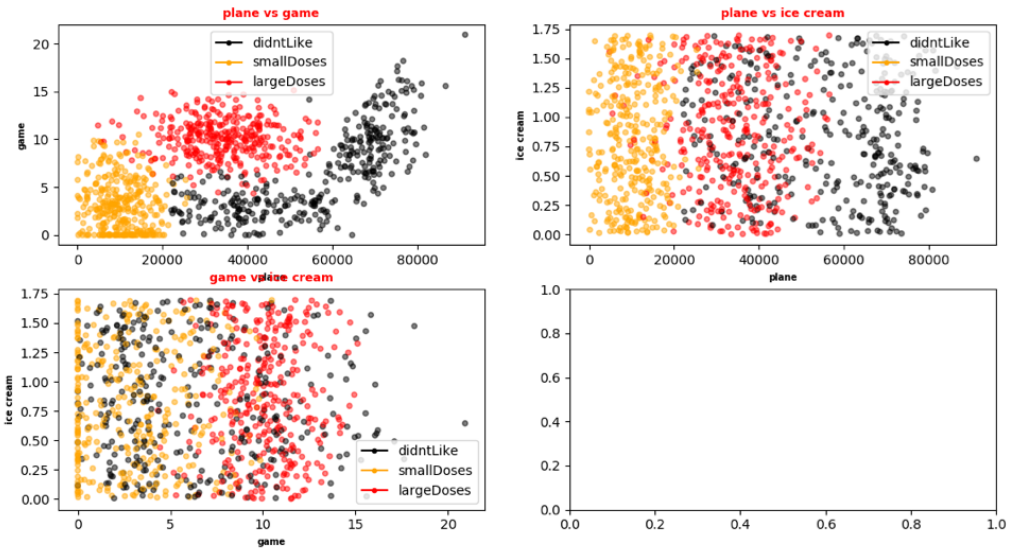
```
#打开的文件名
filename = 'examples/knn/datingTestSet.txt'
#打开并处理数据
datingDataMat, datingLabels = file2matrix(filename)
print(datingDataMat)
print(datingLabels)
```

分析数据：数据可视化

```
import matplotlib.lines as mlines
import matplotlib.pyplot as plt
import numpy as np

"""
函数说明:可视化数据

Parameters:
    datingDataMat - 特征矩阵
    datingLabels - 分类Label
Returns:
    无
"""
```

```
def showdatas(datingDataMat, datingLabels):
    #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('plane vs game')
    axs0_xlabel_text = axs[0][0].set_xlabel('plane')
    axs0_ylabel_text = axs[0][0].set_ylabel(u'game')
    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('plane vs ice cream')
    axs1_xlabel_text = axs[0][1].set_xlabel(u'plane')
    axs1_ylabel_text = axs[0][1].set_ylabel(u'ice cream')
    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'game vs ice cream')
    axs2_xlabel_text = axs[1][0].set_xlabel(u'game')
    axs2_ylabel_text = axs[1][0].set_ylabel(u'ice cream')
    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()
```



样本	玩游戏所耗时间百分比	每年获得的飞行常用里程数	每周消费的冰淇淋公升数	样本分类
1	0.8	400	0.5	1
2	12	134000	0.9	3
3	0	20000	1.1	2
4	67	32000	0.1	2

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

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

```
"""
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 - 数据范围
```

```
    minVals - 数据最小值
"""
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
```

```
#打开的文件名
filename = "examples/knn/datingTestSet.txt"
#打开并处理数据
datingDataMat, datingLabels = file2matrix(filename)
normDataSet, ranges, minVals = autoNorm(datingDataMat)
print(normDataSet)
print(ranges)
print(minVals)
```

测试算法：验证分类器

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

```
"""
函数说明:分类器测试函数
```

```
Parameters:
    无
Returns:
    normDataSet - 归一化后的特征矩阵
    ranges - 数据范围
    minVals - 数据最小值

Modify:
    2017-03-24
"""
def datingClassTest():
    #打开的文件名
    filename = "examples/knn/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))
```

```
datingClassTest()
```