



输出结果如下：

```
array([1])
```

8.5.3 KNN 算法

KNN 算法的思想很简单：计算待分类的样本与训练集中的所有样本的距离，取出与待分类样本距离最近的 k 个样本，统计这 k 个样本的类别数量，根据多数表决原则，取样本数量最多的那一类作为待分类样本的类别。距离度量可采用欧几里得距离、曼哈顿距离和余弦值等。

算法的实现代码如例 8.12 所示。

【例 8.12】 用 Python 实现 KNN 算法，并对 datingTestSet.txt 数据集的数据进行分析，训练分类器。根据“玩视频游戏所消耗时间百分比”“每年获得的飞行常客里程数”和“每周消费的冰淇淋升数”这 3 个特征来确定你对这个人的印象：“讨厌”“有些喜欢”和“非常喜欢”。

```
# -*- coding: utf-8 -*-
from matplotlib.font_manager import FontProperties
import matplotlib.lines as mlines
import matplotlib.pyplot as plt
import time
import numpy as np
import operator

def classify0(inX, dataSet, labels, k): # KNN 算法, 分类器
    dataSetSize = dataSet.shape[0] # shape[0] 返回 dataSet 的行数
    diffMat = np.tile(inX, (dataSetSize, 1)) - dataSet # 将 inX 重复 dataSetSize 次并排成一列
    sqDiffMat = diffMat**2 # 二维特征相减后平方
    sqDistances = sqDiffMat.sum(axis=1)
    # sum 将所有元素相加, sum(0) 将所有列相加, sum(1) 将所有行相加
    distances = sqDistances**0.5 # 开方, 计算距离
    # argsort 函数返回的是距离值从小到大的索引
    sortedDistIndicies = distances.argsort()
    classCount = {} # 定义一个记录同一类别中的样本个数的字典
    for i in range(k): # 选择距离最小的 k 个点
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1
    sortedClassCount = sorted(classCount.items(),
                              key = operator.itemgetter(1), reverse = True)
    return sortedClassCount[0][0] # 返回样本个数最多的类别, 即待分类样本的类别

def file2matrix(filename): # 打开解析文件, 对数据进行分类
    fr = open(filename)
    array0lines = fr.readlines()
```

```

numberOfLines = len(arrayOlines)
returnMat = np.zeros((numberOfLines, 3))
# 返回矩阵中的 numberOfLines 行、3 列
classLabelVector = [] # 创建分类标签向量
index = 0
for line in arrayOlines: # 读取每一行
    line = line.strip()
    # 去掉每一行首尾的空白符, 例如 '\n', '\r', '\t', ' '
    listFromLine = line.split('\t') # 将每一行内容根据 '\t' 进行切片
    returnMat[index, :] = listFromLine[0:3]
    # 提取数据的前 3 列, 保存在特征矩阵中
    # 根据文本内容进行分类: 1 为讨厌, 2 为喜欢, 3 为非常喜欢
    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 # 返回特征矩阵以及分类标签向量

def showdatas(datingDataMat, datingLabels): # 可视化数据
# 设置汉字为 14 号简体字
font = FontProperties(fname=r"C:\Windows\Fonts\simsun.ttc", size=14)
fig, axs = plt.subplots(nrows=2, ncols=2, sharex=False, sharey=False,
figsize=(13, 8))
LabelsColors = [] # label 的颜色配置矩阵
for i in datingLabels:
    if i == 1:
        LabelsColors.append('black')
    if i == 2:
        LabelsColors.append('orange')
    if i == 3:
        LabelsColors.append('red')
# 绘制散点图, 以 datingDataMat 矩阵第一列为 x, 第二列为 y, 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)
# 设置坐标轴的标目
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
所消耗时间百分比', 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百分比',
FontProperties=font)

```



```

plt.setp(axes0_title_text, size=9, weight='bold', color='red')
plt.setp(axes0_xlabel_text, size=7, weight='bold', color='black')
plt.setp(axes0_ylabel_text, size=7, weight='bold', color='black')
# 绘制散点图,以 datingDataMat 矩阵第一列为 x,第三列为 y,散点大小为 15,透明度为 0.5
axes[0][1].scatter(x=datingDataMat[:,0], y=datingDataMat[:,2],
color=LabelsColors, s=15, alpha=.5)
axes1_title_text=axes[0][1].set_title(u'每年获得的飞行常客里程数与每周消费的
冰淇淋升数', FontProperties=font)
axes1_xlabel_text=axes[0][1].set_xlabel(u'每年获得的飞行常客里程数',
FontProperties=font)
axes1_ylabel_text=axes[0][1].set_ylabel(u'每周消费的冰淇淋升数',
FontProperties=font)
plt.setp(axes1_title_text, size=9, weight='bold', color='red')
plt.setp(axes1_xlabel_text, size=7, weight='bold', color='black')
plt.setp(axes1_ylabel_text, size=7, weight='bold', color='black')
# 绘制散点图,以 datingDataMat 矩阵第二列为 x,第三列为 y,散点大小为 15,透明度为 0.5
axes[1][0].scatter(x=datingDataMat[:,1], y=datingDataMat[:,2], color=
LabelsColors, s=15, alpha=.5)
axes2_title_text=axes[1][0].set_title(u'玩视频游戏所消耗时间百分比与每周消费
的冰淇淋升数', FontProperties=font)
axes2_xlabel_text=axes[1][0].set_xlabel(u'玩视频游戏所消耗时间百分比',
FontProperties=font)
axes2_ylabel_text=axes[1][0].set_ylabel(u'每周消费的冰淇淋升数',
FontProperties=font)
plt.setp(axes2_title_text, size=9, weight='bold', color='red')
plt.setp(axes2_xlabel_text, size=7, weight='bold', color='black')
plt.setp(axes2_ylabel_text, size=7, weight='bold', color='black')
# 设置图例
didn'tLike = mlines.Line2D([], [], color='black', marker='.', markersize=6,
label='didn'tLike')
smallDoses = mlines.Line2D([], [], color='orange', marker='.', markersize=6,
label='smallDoses')
largeDoses = mlines.Line2D([], [], color='red', marker='.', markersize=6,
label='largeDoses')
# 添加图例
axes[0][0].legend(handles=[didn'tLike, smallDoses, largeDoses])
axes[0][1].legend(handles=[didn'tLike, smallDoses, largeDoses])
axes[1][0].legend(handles=[didn'tLike, smallDoses, largeDoses])
# 显示图片
plt.show()

def autoNorm(dataSet):
    # 对数据进行归一化
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    ranges = maxVals-minVals

```



```

normDataSet = np.zeros(np.shape(dataSet))
# shape(dataSet) 返回 dataSet 的矩阵行列数
m = dataSet.shape[0] # shape[0] 返回 dataSet 的行数
normDataSet = dataSet - np.tile(minVals, (m, 1)) # 原始值减去最小值 (x - xmin)
normDataSet = normDataSet / np.tile(ranges, (m, 1))
# 上面的差值除以最大值和最小值之差

return normDataSet, ranges, minVals

def datingClassTest(): # 分类器测试函数
    filename = "E:\数据挖掘 & Python\第 8 章\data\datingTestSet.txt"
    # 将返回的特征矩阵和分类标签向量分别存储到 datingDataMat 和 datingLabels 中
    datingDataMat, datingLabels = file2matrix(filename)
    hoRatio = 0.10 # 取所有数据的 10%。hoRatio 越小, 错误率越低
    # 数据归一化, 返回归一化数据结果、数据范围和最小值
    normMat, ranges, minVals = autoNorm(datingDataMat)
    m = normMat.shape[0]
    numTestVecs = int(m * hoRatio) # 10% 的测试数据的个数
    errorCount = 0.0 # 分类错误计数
    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
        # k 选择标签数 + 1 (结果比较好)
        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))

def classifyPerson(): # 输入一个人的 3 个特征, 分类输出
    resultList = ['讨厌', '有些喜欢', '非常喜欢']
    percentTats = float(input("玩视频游戏所消耗时间百分比:")) # 三维特征用户输入
    ffMiles = float(input("每年获得的飞行常客里程数:"))
    iceCream = float(input("每周消费的冰淇淋升数:"))
    filename = "E:\数据挖掘 & Python\第 8 章\data\datingTestSet.txt"
    datingDataMat, datingLabels = file2matrix(filename)
    normMat, ranges, minVals = autoNorm(datingDataMat) # 训练集归一化
    inArr = np.array([percentTats, ffMiles, iceCream])
    norminArr = (inArr - minVals) / ranges # 测试集归一化
    classifierResult = classify0(norminArr, normMat, datingLabels, 4)
    print("你可能%s 这个人" % (resultList[classifierResult - 1]))

def main():
    start = time.clock() # 获取程序运行时间
    filename = "E:\数据挖掘 & Python\第 8 章\data\datingTestSet.txt"
    datingDataMat, datingLabels = file2matrix(filename)
    normDataSet, ranges, minVals = autoNorm(datingDataMat)
    datingClassTest()

```



```

        # showdatas(datingDataMat, datingLabels)          # 绘制散点图
    classifyPerson()
    end = time.clock()
    print('Running time: %f seconds'% (end-start))        # 打印程序运行时间
if __name__ == '__main__':
    main()

```

输出结果如下:

```

分类结果:1    真实类别:1
分类结果:1    真实类别:1
分类结果:3    真实类别:3
分类结果:2    真实类别:3
分类结果:1    真实类别:1
分类结果:2    真实类别:2
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%
玩视频游戏所消耗时间百分比:11
每年获得的飞行常客里程数:1
每周消费的冰淇淋升数:1
你可能有些喜欢这个人
Running time: 8.539824 seconds

```

以上只给出了部分测试数据的预测结果。本例对给出的数据集中的特征进行预测分类,同时利用输入的3个特征预测自己是否喜欢这个人。也可以针对每个特征绘制散点图,实现方法已经写在代码中,感兴趣的读者可以通过调用函数绘制散点图。

例 8.12 中的主要函数如表 8.11 所示。

表 8.11 例 8.12 中的主要函数

函 数	描 述	参 数	返 回
classify0(inX, dataSet, labels, k)	KNN 算法, 分类器	inX: 用于分类的数据(测试集) dataSet: 用于训练的数据(训练集, n 维列向量) labels: 分类标准(n 维列向量) k: KNN 算法参数, 选择距离最小的 k 个点	分类结果
file2matrix(filename)	打开解析文件, 对数据进行分类	filename: 文件名	特征矩阵和分类标签向量