第九周学习笔记—KNN 算法

2022-06-11

第一章　KNN 算法

1.1　KNN 算法原理

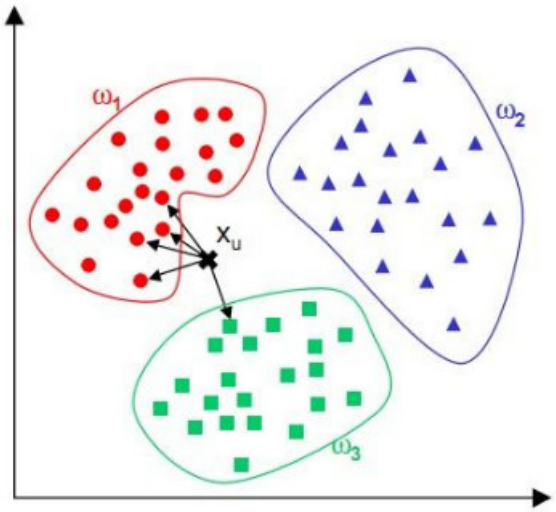
KNN 算法是选择与输入样本在特征空间内最近邻的 k 个训练样本并根据一定的决策规则，给出输出结果。

决策规则：

分类任务：输出结果为 k 个训练样本中占大多数的类。

回归任务：输出结果为 k 个训练样本值的平均值。

如下图的分类任务，输出结果为 w_1 类。



1.2　KNN 算法三要素

K 值的选择、距离度量和分类决策规则是 K 近邻算法的三个基本要素。当三个要素确定后，对于任何一个新的输入实例，它所属的 Y 值也确定了，本节介绍了三要素的含义。

1.2.1　分类决策规则

KNN 算法一般是用多数表决方法，即由输入实例的 K 个邻近的多数类决定输入实例的类。这种思想也是经验风险最小化的结果。

训练样本为 (x_i, y_i)。当输入实例为 x，标记为 c，$N_k(x)$ 是输入实例 x 的 k 近邻训练样本集。

我们定义训练误差率是 K 近邻训练样本标记与输入标记不一致的比例，误差率表示为：

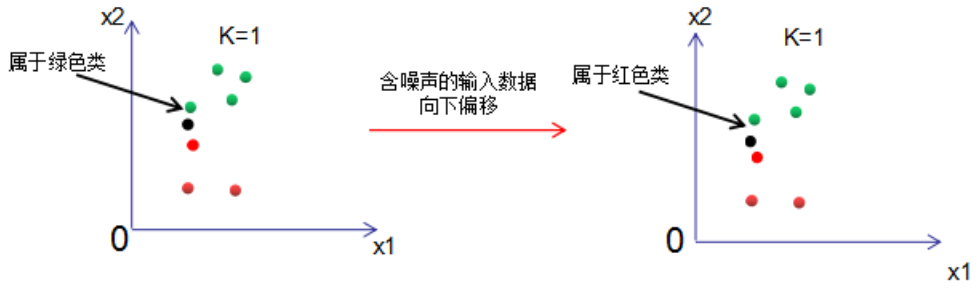
$$\frac{1}{k}\sum_{x_i \in N_k(x)} I(y_i \neq c_j) = 1 - \frac{1}{k}\sum_{x_i \in N_k(x)} I(y_i = c_j)$$

因此，要使误差率最小化即经验风险最小，就要使上式右端的 $\frac{1}{k}\sum_{x_i \in N_k(x)} I(y_i = c_j)$ 最大，即 K 近邻的标记值尽可能的与输入标记一致，所以多数表决规则等价于经验风险最小化。

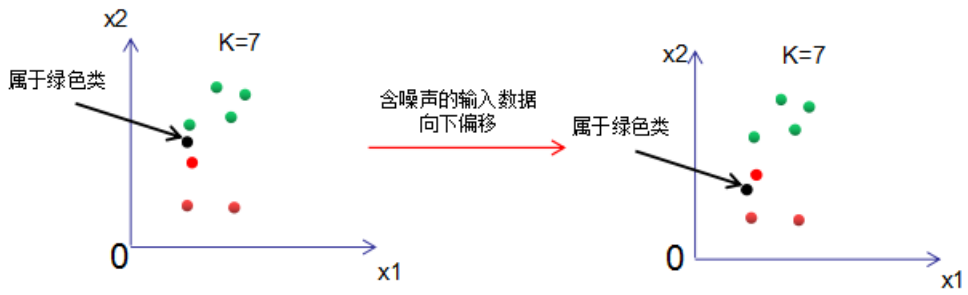
1.2.2　K 值的选择

K 取值较小时，模型复杂度高，训练误差会减小，泛化能力减弱；K 取值较大时，模型复杂度低，训练误差会增大，泛化能力有一定的提高。

KNN 模型的复杂度可以通过对噪声的容忍度来理解，若模型对噪声很敏感，则模型的复杂度高；反之，模型的复杂度低。为了更好理解模型复杂度的含义，我们取一个极端，分析 K=1 和 K=" 样本数" 的模型复杂度。



由上图可知，K=1 时，模型输出的结果受噪声的影响很大。



由上图可知，样本数等于 7，当 K=7 时，不管输入数据的噪声有多大，输出结果都是绿色类，模型对噪声极不敏感，但是模型太过简单，包含的信息太少，也是不可取的。

通过上面两种极端的 K 选取结果可知，K 值选择应适中，K 值一般小于 20，建议采用交叉验证的方法选取合适的 K 值。

1.2.3　距离度量

KNN 算法用距离来度量两个样本间的相似度，常用的距离表示方法：

欧式距离

$$D(x,y)=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+...+(x_n-y_n)^2}$$
$$=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

曼哈顿距离

$$D(x,y)=|x_1-y_1|+|x_2-y_2|+...+|x_n-y_n|$$
$$=\sum_{i=1}^{n}|x_i-y_i|$$

闵可夫斯基距离

$$D(x,y)=\sqrt[p]{|x_1-y_1|^p+|x_2-y_2|^p+...+|x_n-y_n|^p}$$
$$=\sqrt[p]{\sum_{i=1}^{n}|x_i-y_i|^p}$$

可以看出，欧式距离是闵可夫斯基距离在 p=2 时的特例，而曼哈顿距离是 p=1 时的特例。

1.3　KNN 算法之暴力实现方法

暴力搜索（brute-force search）是线性扫描输入实例与每一个训练实例的距离并选择前 k 个最近邻的样本来多数表决，算法简单，但是当训练集或特征维度很大时，计算非常耗时，故这种暴力实现原理是不可行的。

1.4　KNN 算法之 kd 树实现方法

kd 树是一种对 k 维空间中的实例点进行存储以便对其进行快速检索的树形数据结构，构造 kd 树相当于不断用垂直于坐标轴的超平面将 k 维空间进行划分，构成一系列的 K 维超矩形区域，kd 树省去了对大部分数据的搜索，大大的较少了计算量。

kd 树的 KNN 算法实现包括三部分：kd 树的构建，kd 树的搜索和 kd 树的分类。

1.4.1　构建 kd 树

kd 树实质是二叉树，其划分思想与 cart 树一致，即切分使样本复杂度降低最多的特征。kd 树认为特征方差越大，则该特征的复杂度亦越大，优先对该特征进行切分，切分点是所有实例在该特征的中位数。重复该切分步骤，直到切分后无样本则终止切分，终止时的样本为叶节点。

【例】给定一个二维空间的数据集：

$$T=(2,3)^T,(5,4)^T,(9,6)^T,(4,7)^T,(8,1)^T,(7,2)^T$$

构造 kd 树的步骤：

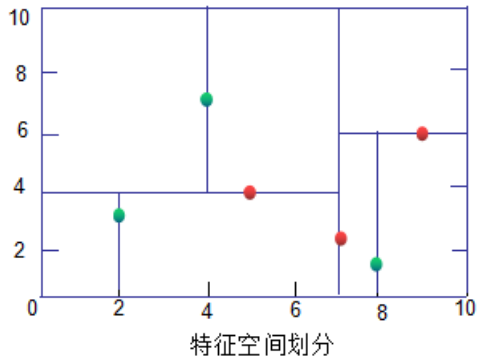
(1) 数据集在维度 $x^{(1)}$ 和 $x^{(2)}$ 的方差分别为 6.9 和 5.3，因此首先从 $x^{(1)}$ 维度进行切分。

(2) 数据集在 $x^{(1)}$ 维度的中位数是 7，以平面 $x^{(1)}=7$ 将空间分为左右两个矩形。

(3) 分别对左右两个矩形的样本在 $x^{(2)}$ 维度的中位数进行切分。

(4) 重复步骤（2）（3），直到无样本，该节点为叶子节点。

如下图，绿色为叶子节点，红色为节点和根节点。



1.4.2　KD 树搜索

(1) 搜索路径从根节点到叶节点，在 KD 树里面找到包含目标点的叶子节点。

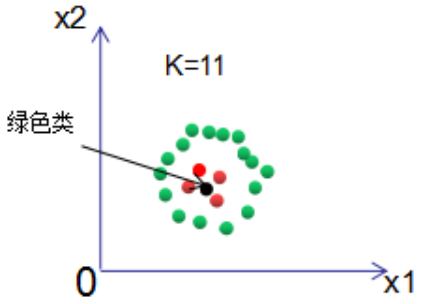
(2) 搜索路径从叶节点到根节点，找到距离目标点最近的样本实例点。

1.4.3　KD 树预测

每一次搜寻与输入样本最近的样本节点，然后忽略该节点，重复同样步骤 K 次，找到与输入样本最近邻的 K 个样本，投票法确定输出结果。

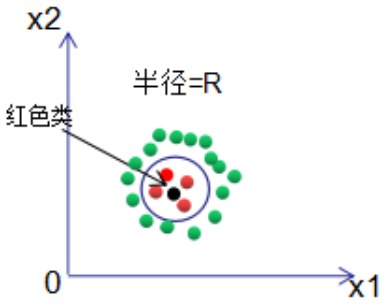
1.5　KNN 算法之训练样本不平衡情况

若正负样本处于不平衡状态，运用投票决策的 KNN 算法判断输入样本的所属类别：



结果显示输入样本为绿色类。原因是红色类的个数远远小于绿色样本，导致出现的分类错误。

（1）若分类决策选择限定半径最近邻法，即以输入样本为圆心，最大半径 R 的圆内选择出现次数最多的类做为输入样本的类。如下图，黑色样本的分类结果正确。



（2）投票法是默认每个样本的权重相等，我们假定权重与距离成反比，即距离越大，对结果的影响越小，那么该样本的权重也越小，反之，权重则越大，根据权重对输入样本进行分类。

1.5.1　分类过程

(1) 选择与输入样本距离 X_0 最近的 K 个训练样本 X_i（i = 1,2,...,K），$d(X_0, X_i)$ 表示输入样本和训练样本的距离。

(2) 根据距离与样本成反比的性质将距离转化成权重 W_i，W_i 表示输入样本 X_0 与训练样本 X_i 的权重。

(3) 我们累加每一类的样本权重，并认为该权重占所有权重和的比例是该类的生成概率，概率最大的类就是输入样本的分类结果。

假设目标是二分类 C1，C2，表达式：

$$P_{C_h} = \frac{\sum_{j \in C_h} W_j}{\sum_{i=1}^{K} W_i} \quad ,(h = 1,2)$$

若 $P_{C_1} \geq P_{C_2}$，则分类结果为 C1 类，反之 C2 类。

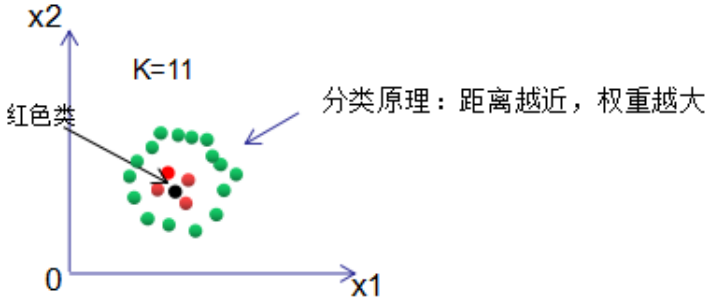
1.5.2　回归过程

（1）（2）步骤与分类过程一直，第（3）步使用如下表达式得到回归值：

$$y = \frac{\sum_{i=1}^{K} W_i f(x_i)}{\sum_{i=1}^{K} W_i}$$

其中，y 为输出结果，$f(x_i)$ 为最近邻样本的值。若权重相同的话，则输出结果为 K 个训练样本的平均值。

用权重思想重新对上例进行分类，可得输入样本为红色类。



1.6　KNN 算法优缺点

1.6.1　优点

1）算法简单，理论成熟，可用于分类和回归。

2）对异常值不敏感。

3）可用于非线性分类。

4）比较适用于容量较大的训练数据，容量较小的训练数据则很容易出现误分类情况。

5）KNN 算法原理是根据邻域的 K 个样本来确定输出类别，因此对于不同类的样本集有交叉或重叠较多的待分样本集来说，KNN 方法较其他方法更为合适。

1.6.2　缺点

1）时间复杂度和空间复杂度高。

2）训练样本不平衡，对稀有类别的预测准确率低。

3）相比决策树模型，KNN 模型可解释性不强。

第二章　KNN 算法的实现

K-近邻算法的伪代码如下：

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。

2.1　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：

(1) 每年获得的飞行常客里程数

(2) 玩视频游戏所耗时间百分比

(3) 每周消费的冰淇淋公升数

我们简单看一下数据的构成：

```
file_name="./2022-06-08data/datingTestSet.txt"
path=file_name
df=pd.read_csv(path)
print(df)
```

结果展示如下所示:

```
      40920\t8.326976\t0.953952\t largeDoses
0     14488\t7.153469\t1.673904\t smallDoses
1      26052\t1.441871\t0.805124\t didntLike
2     75136\t13.147394\t0.428964\t didntLike
3      38344\t1.669788\t0.134296\t didntLike
```

```
4      72993\t10.141740\t1.032955\t didntLike
..                                        ...
994    11145\t3.410627\t0.631838\t smallDoses
995     68846\t9.974715\t0.669787\ tdidntLike
996   26575\t10.650102\t0.866627\t largeDoses
997    48111\t9.134528\t0.728045\t largeDoses
998    43757\t7.882601\t1.332446\t largeDoses
[999 rows x 1 columns]
```

didntLike 表示不喜欢，smallDoses 表示魅力一般，largeDoses 表示极具魅力。

从输出结果可以看出，我们的数据有四列，以分隔符隔开。在将上述特征数据输入到分类器之前，我们必须将待处理数据的格式改变为分类器可以接受的格式。将字符串，训练为输出样本矩阵和类标签向量。

将文本记录转换为 Numpy 矩阵的代码实现如下：

```
"""
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
:parameter
file_name:文件名
returns:
character_matrix:特征矩阵
class_label_vector:分类label向量
"""
def file_matrix(file_name):
    #打开文件
    file=open(file_name)
    #读取文件所有内容
    array_lines=file.readlines()
    #得到文件的行数
    number_lines=len(array_lines)
    #返回numpy矩阵，解析完成的数据:number_lines行，3列
    return_character_matrix=np.zeros((number_lines,3))
    #返回的分类标签向量
    class_label_vector=[]
    #行的索引值
    index=0
    for line in array_lines:
        #s.strip(rm),当rm为空时，默认删除空白符（包括'\n','\t','\r',''）
```

```
        line=line.strip()
        #使用s.strip(str='',num=string,count(str))将字符串根据'\t'分隔符进
                                                行切片
        list_from_line=line.split('\t')
        #将数据前三列提取出来，存放到character_matrix的numpy矩阵中
        return_character_matrix[index,:]=list_from_line[0:3]
        #根据文本中标记的喜欢程度进行分类，1代表不喜欢，2代表魅力一般，3代
                                                表极具魅力
        if list_from_line[-1]=='didntLike':
            class_label_vector.append(1)
        elif list_from_line[-1]=='smallDoses':
            class_label_vector.append(2)
        elif list_from_line[-1]=='largeDoses':
            class_label_vector.append(3)
        index+=1
    return return_character_matrix,class_label_vector
```

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

首先我们需要知道文本文件包含多少行。打开文件，得到文件的行数。然后创建以零填充的矩阵 NumPy（实际上，NumPy 是一个二维数组）。为了简化处理，我们将该矩阵的另一维度设置为固定值 3。循环处理文件中的每行数据，首先使用函数 line.strip() 截取掉所有的回车字符，然后使用 tab 字符 t 将上一步得到的整行数据分割成一个元素列表。接着，我们选取前 3 个元素，将它们存储到特征矩阵中。Python 语言可以使用索引值-1 表示列表中的最后一列元素，利用这种负索引，我们可以很方便地将列表的最后一列存储到向量 classlabelvector 中。需要注意的是，我们必须明确地通知解释器，告诉它列表中存储的元素值为整型，否则 Python 语言会将这些元素当作字符串处理。

我们写出数据的结果:

```
dating_mat,dating_labels=KNN_practice.file_matrix(file_name)
```

结果如下所示:

```
==========================================================================
dating_mat= [[4.0920000e+04 8.3269760e+00 9.5395200e-01]
 [1.4488000e+04 7.1534690e+00 1.6739040e+00]
 [2.6052000e+04 1.4418710e+00 8.0512400e-01]
 ...
 [2.6575000e+04 1.0650102e+01 8.6662700e-01]
```

```
 [4.8111000e+04 9.1345280e+00 7.2804500e-01]
 [4.3757000e+04 7.8826010e+00 1.3324460e+00]]
==========================================================================
dating_labels= [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,....2, 1, 3, 3, 3]
==========================================================================
```

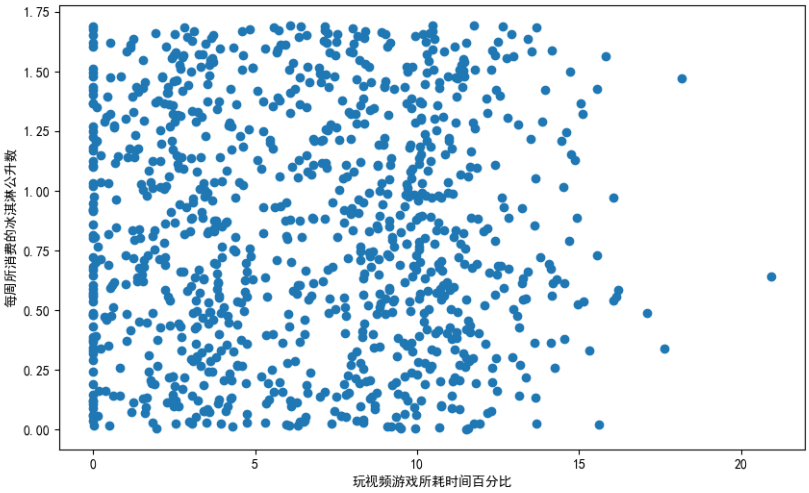
现在已经从文本文件中导入了数据，并将其格式化为想要的格式，接着我们需要了解数据的真实含义。当然我们可以直接浏览文本文件，但是这种方法非常不友好，一般来说，我们会采用图形化的方式直观地展示数据。

2.2　分析数据：使用 Matplotlib 创建散点图

首先我们使用 Matplotlib 制作原始数据的散点图。

```
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib.font_manager import FontProperties  # 导入FontProperties
# matplotlib其实是不支持显示中文的 显示中文需要一行代码设置字体
mpl.rcParams['font.family'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False
fig,ax=plt.subplots(figsize=(10,6))
ax.scatter(dating_mat[:,1],dating_mat[:,2])
plt.xlabel('玩视频游戏所耗时间百分比')
plt.ylabel('每周所消费的冰淇淋公升数')
plt.show()
```

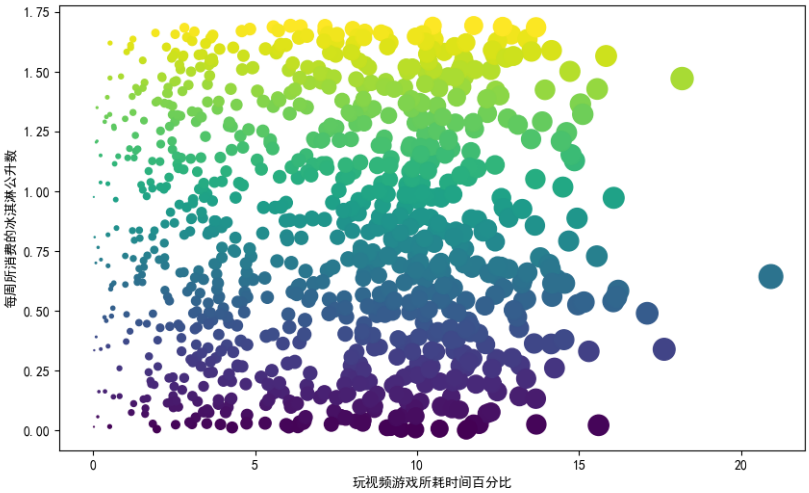
散点图使用 datingmat 矩阵的第二、第三列数据，分别表示特征值“玩视频游戏所耗时间百分比”和“每周所消费的冰淇淋公升数”。



由于没有使用样本分类的特征值，我们很难从图 2-3 中看到任何有用的数据模式信息。一般来说，我们会采用色彩或其他的记号来标记不同样本分类，以便更好地理解数据信息。Matplotlib 库提供的 scatter 函数支持个性化标记散点图上的点。

重新输入上面的代码，调用 scatter 函数时使用下列参数：

结果如下图所示:



上述代码利用变量 datinglabels 存储的类标签属性，在散点图上绘制了色彩不等、尺寸不同的点。我们可以看到一个与上图类似的散点图。从上图中，我们很难看到任何有用的信息，然而由于利用颜色及尺寸标识了数据点的属性类别，因而我们基本上可以从图中看到数据点所属三个样本分类的区域轮廓。但是依旧很难得出结论性信息。

2.3　准备数据：归一化数值

根据上面提取的四组数据，我们计算第一组和第二组数据之间的距离：

$$\sqrt{(40920-14488)^2+(8.32697-7.143569)^2+(0.953952-1.673904)^2}$$

我们很容易发现，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。但我们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：

```
new_value=(old_value-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。

```
"""
函数说明:对数据进行归一化
:parameter
character_matrix:特征矩阵
returns:
norm_dataset:归一化后的特征矩阵
ranges:数据范围
min_vals:数据的最小值
"""
def auto_norm(dataset):
    #获取数据的最小值
    min_vals=dataset.min(0)
    max_vals=dataset.max(0)
    #最大值和最小值的范围
    ranges=max_vals-min_vals
    norm_dataset=np.zeros(np.shape(dataset))
    # 返回特征矩阵的行数
    m = dataset.shape[0]
    norm_dataset=dataset-np.tile(min_vals,(m,1))
    #除以最大和最小值的差，得到归一化数据
```

```
    norm_dataset=norm_dataset/np.tile(ranges,(m,1))
    #返回归一化数据结果,数据范围, 最小值
    return norm_dataset,ranges,min_vals
```

在函数 autonorm() 中，我们将每列的最小值放在变量 minvals 中，将最大值放在变量 maxvals 中，其中 dataset.min(0) 中的参数 0 使得函数可以从列中选取最小值，而不是选取当前行的最小值。然后，函数计算可能的取值范围，并创建新的返回矩阵。正如前面给出的公式，为了归一化特征值，我们必须使用当前值减去最小值，然后除以取值范围。需要注意的是，特征值矩阵 1000×3 个值，而 minvals 和 ranges 的值都为 1×3。为了解决这个问题，我们使用 Numpy 库中 tile() 函数将变量内容复制成输入矩阵同样大小的矩阵，注意这是具体特征值相除，而对于某些数值处理软件包，/可能意味着矩阵除法，但在 NumPy 库中，矩阵除法需要使用函数 linalg.solve(matA,matB)。

我们返回的结果如下：

```
==============================================================
norm_mat= [[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]]
==============================================================
ranges= [9.1273000e+04 2.0919349e+01 1.6943610e+00]
==============================================================
min_vals= [0.       0.       0.001156]
==============================================================
```

2.4　KNN 算法

```
"""
parameters:
X:用于分类的数据, 即测试集
dataset:用于训练的数据集, 即训练集
labels:分类标签
k:kNN算法参数, 选择距离最小的k个点
return:
```

```
    sorted_class_count[0][0]:分类结果
    """
    def classify(X,dataset,labels,k):
        dataset_size=dataset.shape[0] #numpy函数shape[0]返回dataset的行数
        #tile具有重复功能，dataset_size是重复四遍，后面的1保证重复完了是四行，
                                                而不是一行里有四个是一样的
        diff_mat=np.tile(X,(dataset_size,1))-dataset
        #二次特征相减后平方
        sqr_diff_mat=diff_mat**2
        #sum()所有元素相加，sum(0)列相加，sum(1)行相加
        sq_distance=sqr_diff_mat.sum(axis=1)
        #开方，计算出距离
        distance=sq_distance**0.5
        #返回distance中元素从小到大排列后的索引值
        sorted_distance=distance.argsort()
        #定一个记录类别次数的字典
        class_count={}
        for i in range(k):
             #取出前k个元素的类别
            label=labels[sorted_distance[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
            class_count[label]=class_count.get(label,0)+1
            #python3中用items()替换python2中的iteritems()
            #key=operator.iteritems(1)根据字典的值进行排序
            #key=operator.iteritems(0)根据字典的键进行排序
            #reverse降序排列字典
        sorted_class_count=sorted(class_count.items(),key=operator.itemgetter(1
                                                ),reverse=True)
        #返回次数最多的类别
        return sorted_class_count[0][0]
```

函数有 4 个输入参数：用于分类的输入向量是 X，输入的训练样本集为 dataset，标签向量为 labels，最后的参数 k 表示用于选择最近邻居的数目，其中标签向量的元素数目和矩阵 dataset 的行数相同。程序使用欧氏距离公式，计算两个向量点 xA 和 xB 之间的距离:

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

例如，点 (0, 0) 与 (1, 2) 之间的距离计算为：

$$\sqrt{(0-1)^2+(0-2)^2}$$

计算完所有点之间的距离后，可以对数据按照从小到大的次序排序。然后，确定前 k 个距离最小元素所在的主要分类，输入 k 总是正整数；最后，将 classcount 字典分解为元组列表，然后使用程序第二行导入运算符模块的 items 方法，按照第二个元素的次序对元组进行排序。此处的排序为逆序，即按照从最大到最小次序排序，最后返回发生频率最高的元素标签。

2.5　测试算法：作为完整程序验证分类器

上节我们已经将数据按照需求做了处理，本节我们将测试分类器的效果，如果分类器的正确率满足要求，海伦就可以使用这个软件来处理约会网站提供的约会名单了。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，所以我们可以随意选择 10% 数据而不影响其随机性。

前面我们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。

我们的测试代码如下：

```
"""
函数说明：分类器测试函数
Parameters:
无
returns:
norm_dataset:归一化后的特征矩阵
ranges:数据范围
min_vals:数据最小值
"""
def dating_class_test():
    #打开文件名
```

```
    file_name="./2022-06-08data/datingTestSet.txt"
    #将返回的特征矩阵和分类向量分别存储到dating_mat和dating_labels中
    dating_mat,dating_labels=file_matrix(file_name)
    #读取所有数据的10%
    data_rate=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
    norm_mat,ranges,min_vals=auto_norm(dating_mat)
    #获取norm_matrix的行数
    m=norm_mat.shape[0]
    #10%的测试数据的数量
    number_test_data=int(m*data_rate)
    #分类错误计数
    error_number=0.0
    for i in range(number_test_data):
        #前number_test_data个数据作为测试集，后m-number_test_data个数据作为
                                                训练集
        classify_result=classify(norm_mat[i,:],norm_mat[number_test_data:m,
                                                :],dating_labels[
                                                number_test_data:m],3)
        print('分类结果:%d\t真实类别:%d'%(classify_result,dating_labels[i])
                                                )
        if classify_result!=dating_labels[i]:
            error_number+=1.0
    print('错误分类:%f%%'%(error_number/float(number_test_data)*100))
```

函数 datingclasstest 如程序所示，它首先使用了 filematrix 和 autonorm() 函数从文件中读取数据并将其转换为归一化特征值。接着计算测试向量的数量，此步决定了 normmat 向量中哪些数据用于测试，哪些数据用于分类器的训练样本；然后将这两部分数据输入到原始 kNN 分类器函数 classify。最后，函数计算错误率并输出结果。

我们将得到下面的输出结果：

```
==================================================
分类结果:3   真实类别:3
分类结果:2   真实类别:2
分类结果:1   真实类别:1
分类结果:1   真实类别:1
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: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:1   真实类别:1
分类结果:3   真实类别:1
错误分类:5.000000%
None
==================================================
```

分类器处理约会数据集的错误率是 5%，这是一个相当不错的结果。我们可以改变函数 datingclasstest 内变量 rate 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。

2.6　使用算法：构建完整可用系统

上面我们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来进行分类。

实现的代码如下：

```
def classify_person():
    file_name='./2022-06-08data/datingTestSet.txt'
    result_list = ['didntLike', 'smallDoses', 'largeDoses']
    fly_miles = float(input('每周飞行的时长为:'))
    game_hours = float(input('每天玩游戏的时间为:'))
    ice_cream = float(input('每年吃的冰淇淋为:'))
    dating_mat, dating_labels = file_matrix(file_name)
    norm_mat, ranges, min_vals = auto_norm(dating_mat)
    input_array = np.array([fly_miles, game_hours, ice_cream])
    class_result = classify((input_array - min_vals) / ranges, norm_mat,
                                          dating_labels, 3)
    print('可能喜欢这个人:', result_list[class_result - 1])
```

输出的结果如下：

```
print(KNN_practice.classify_person())
```

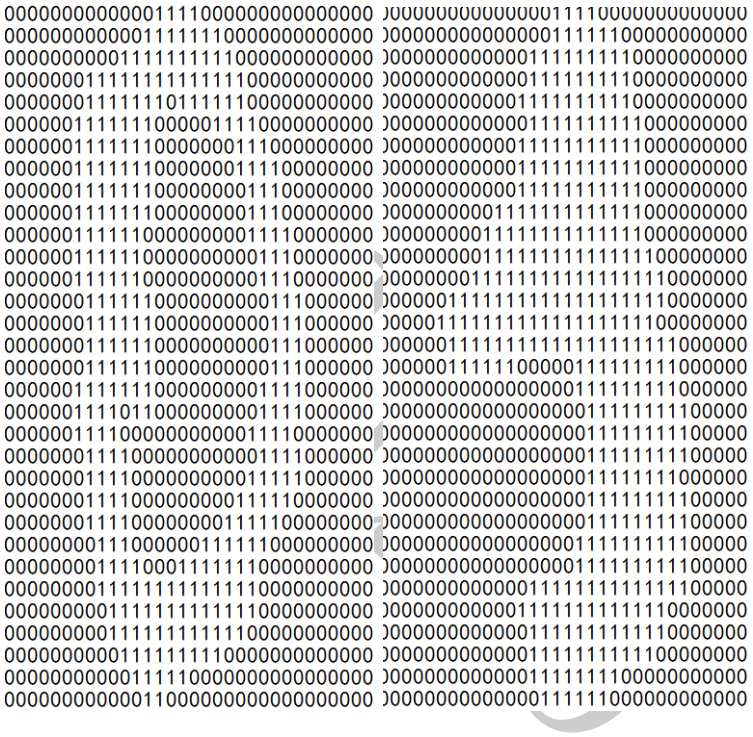
```
============================================================
每周飞行的时长为:10000
每天玩游戏的时间为:10
```

```
每年吃的冰淇淋为:0.5
可能喜欢这个人: smallDoses
```

目前为止，我们已经看到如何在数据上构建分类器。这里所有的数据让人看起来都很容易，但是如何在人不太容易看懂的数据上使用分类器呢？

第三章　手写识别系统

本节我们一步步的构造使用 K—近邻分类器的手写识别系统。为了简单起见，这里构造的系统只能识别数字 0 到 9。如下图所示：



需要识别的数字已经使用了图像处理软件，处理成具有相同的色彩和大小：宽度是 32 像素 ×32 像素的黑白图像。尽管采用文本格式存储图像不能有效的利用内存空间，但是为了方便理解，我们还是将图像转换成文本格式。

（1）收集数据：提供文本文件。

（2）准备数据：编写函数 classify()，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。

3.1　准备数据：将图像转换为测试向量

实际图像存储在两个子目录内: 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。我们使用目录 *trainingDigits* 中的数据训练分类器，使用目录 *testDigits* 中的数据测试分类器的效果。

为了使用前面两个例子的分类器，我们必须将图像格式化处理为一个向量。我们将把一个 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。

我们首先编写一段函数 imagevector 函数，将图像转换为向量：该函数创建 1×1024 的 NumPy 数组，然后打开给定的文件，循环读出文件的前 32 行，并将每行的头 32 个字符值存储在 NumPy 数组中，最后返回数组。

我们定义函数如下：

```
def image_vector(file_name):
    return_vector=np.zeros((1,1024)) #返回一个1*1024的向量
    file=open(file_name)
    for i in range(32):
        line_string=file.readline() #每次读一行
        for j in range(32):
            return_vector[0,32*i+j]=int(line_string[j]) #这个向量本身也完全
                                                        是由0,1构成，相当于将原来
                                                        的矩阵每一行首尾相连
    return return_vector
```

我们测试函数的结果：

```
test_vector=KNN_practice.image_vector('./2022-06-08data/testDigits/0_13.txt
                                      ')
print(test_vector[0,0:31])
print(test_vector[0,32:63])
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0.]
```

3.2　测试算法：使用 k-近邻算法识别手写数字

上节我们已经将数据处理成分类器可以识别的格式，本节我们将这些数据输入到分类器，检测分类器的执行效果。函数 handwritingClassTest() 是测试分类器的代码。在写入这些代码之前，我们必须确保将 from os import listdir 写入文件的起始部分，这段代码的主要功能是从 os 模块中导入函数 listdir，它可以列出给定目录的文件名。

手写数字识别系统的测试代码如下所示：

```
def handwritingClassTest():
    hwLabels = []
    training_file_list = os.listdir('./2022-06-08data/trainingDigits') #导
                                            入训练集，‘trainingDigits’是一
                                            个文件夹
    m = len(training_file_list) #计算训练样本个数
    training_mat = np.zeros((m,1024)) #初始化数据集，将所有训练数据用一个m
                                            行，1024列的矩阵表示
    for i in range(m):
        file_name_str = training_file_list[i]   #获得所有文件名，文件名格式
                                                ‘x_y.txt’,x表示这个手写数字
                                                实际表示的数字（label）
        file_str = file_name_str.split('.')[0]  #去除 .txt
        class_num_str = int(file_str.split('_')[0])   #classnumber为每个样
                                                本的分类，用‘_’分割，取得
                                                label
        hwLabels.append(class_num_str)  #将所有标签都存进hwLables[]
        training_mat[i,:] = image_vector('./2022-06-08data/trainingDigits/%
                                                s' % file_name_str) #将文件转
                                                化为向量后存入trainingMat[],
                                                这里展现了灵活的文件操作
    test_file_list = os.listdir('./2022-06-08data/testDigits')  #迭代测试集
    error_count = 0.0
    m_test = len(test_file_list)
    for i in range(m_test):
        file_name_str = test_file_list[i]
```

```
        file_str = file_name_str.split('.')[0]  #去除 .txt
        class_num_str = int(file_str.split('_')[0])
        vector_under_test = image_vector('./2022-06-08data/testDigits/%s' %
                                         file_name_str)  #这部分针对
                                         测试集的预处理和前面基本相同
        classify_result = classify(vector_under_test, training_mat,
                                   hwLabels, 3)    #使用算法预测
                                   样本所属类别，调用了前面写的
                                   classify0()函数
        print ("分类结果:%d,真实类别:%d" % (classify_result, class_num_str)
                                   )
        if (classify_result != class_num_str): error_count += 1.0       #算
                                   法结果与样本的实际分类做对比
    print ("\nthe total number of errors is: %d" % error_count)
    print ("\nthe total error rate is: %f" % (error_count/float(m_test)))
```

在上述程序中，将 *trainingDigits* 目录中的文件内容存储在列表中，然后可以得到目录中有多少文件，并将其存储在变量 m 中。接着，代码创建一个 m 行 1024 列的训练矩阵，该矩阵的每行数据存储一个图像。我们可以从文件名中解析出分类数字。该目录下的文件按照规则命名，如文件 9.45.txt 的分类是 9，它是数字 9 的第 45 个实例。然后我们可以将类代码存储在 hwLabels 向量中，使用前面讨论的 imgagevector 函数载入图像。在下一步中，我们对 testDigits 目录中的文件执行相似的操作，不同之处是我们并不将这个目录下的文件载入矩中，而是使用 classify() 函数测试该目录下的每个文件。由于文件中的值已经在 0 和 1 之间，本节并不需要使用 autonorm() 函数。

输出结果如下：

```
分类结果:0,真实类别:0
分类结果:0,真实类别:0
分类结果:0,真实类别:0
分类结果:0,真实类别:0
分类结果:0,真实类别:0
分类结果:0,真实类别:0
分类结果:0,真实类别:0
.....
分类结果:1,真实类别:1
分类结果:1,真实类别:1
分类结果:1,真实类别:1
分类结果:1,真实类别:1
```

```
分类结果:1,真实类别:1
分类结果:1,真实类别:1
分类结果:1,真实类别:1
......
分类结果:2,真实类别:2
分类结果:2,真实类别:2
分类结果:2,真实类别:2
分类结果:2,真实类别:2
分类结果:2,真实类别:2
分类结果:2,真实类别:2
分类结果:2,真实类别:2
......
分类结果:9,真实类别:9
分类结果:9,真实类别:9
分类结果:9,真实类别:9
分类结果:9,真实类别:9
分类结果:9,真实类别:9
分类结果:9,真实类别:9
分类结果:9,真实类别:9
```

最终的汇总结果如下：

```
the total number of errors is: 10
the total error rate is: 0.010571
```

k-近邻算法识别手写数字数据集，错误率为 1.0571%。改变变量 k 的值、修改函数 handwriting-ClassTest 随机选取训练样本、改变训练样本的数目，都会对 k-近邻算法的错误率产生影响，感兴趣的话可以改变这些变量值，观察错误率的变化。

实际使用这个算法时，算法的执行效率并不高。因为算法需要为每个测试向量做 2000 次距离计算，每个距离计算包括了 1024 个维度浮点运算，总计要执行 900 次。

3.3　本章小结

k-近邻算法是分类数据最简单最有效的算法，本章通过两个例子讲述了如何使用 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，因此我们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。

第四章　源程序

4.1　函数定义

```
import numpy as np
import operator
import os
"""
parameters:
X:用于分类的数据，即测试集
dataset:用于训练的数据集，即训练集
labels:分类标签
k:kNN算法参数，选择距离最小的k个点
return:
sorted_class_count[0][0]:分类结果
"""
def classify(X,dataset,labels,k):
    dataset_size=dataset.shape[0] #numpy函数shape[0]返回dataset的行数
    #tile具有重复功能，dataset_size是重复四遍，后面的1保证重复完了是四行，
                                        而不是一行里有四个是一样的
    diff_mat=np.tile(X,(dataset_size,1))-dataset
    #二次特征相减后平方
    sqr_diff_mat=diff_mat**2
    #sum()所有元素相加，sum(0)列相加，sum(1)行相加
    sq_distance=sqr_diff_mat.sum(axis=1)
    #开方，计算出距离
    distance=sq_distance**0.5
    #返回distance中元素从小到大排列后的索引值
    sorted_distance=distance.argsort()
    #定一个记录类别次数的字典
    class_count={}
    for i in range(k):
```

```
            #取出前k个元素的类别
            label=labels[sorted_distance[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
            class_count[label]=class_count.get(label,0)+1
            #python3中用items()替换python2中的iteritems()
            #key=operator.iteritems(1)根据字典的值进行排序
            #key=operator.iteritems(0)根据字典的键进行排序
            #reverse降序排列字典
        sorted_class_count=sorted(class_count.items(),key=operator.itemgetter(1
                                                    ),reverse=True)
        #返回次数最多的类别
        return sorted_class_count[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
:parameter
file_name:文件名
returns:
character_matrix:特征矩阵
class_label_vector:分类label向量
"""
def file_matrix(file_name):
    #打开文件
    file=open(file_name)
    #读取文件所有内容
    array_lines=file.readlines()
    #得到文件的行数
    number_lines=len(array_lines)
    #返回numpy矩阵，解析完成的数据:number_lines行，3列
    return_character_matrix=np.zeros((number_lines,3))
    #返回的分类标签向量
    class_label_vector=[]
    #行的索引值
    index=0
    for line in array_lines:
        #s.strip(rm),当rm为空时，默认删除空白符（包括'\n','\t','\r',''）
        line=line.strip()
```

```
        #使用s.strip(str='',num=string,count(str))将字符串根据'\t'分隔符进
                                                行切片
        list_from_line=line.split('\t')
        #将数据前三列提取出来，存放到character_matrix的numpy矩阵中
        return_character_matrix[index,:]=list_from_line[0:3]
        #根据文本中标记的喜欢程度进行分类，1代表不喜欢，2代表魅力一般，3代
                                                表极具魅力
        if list_from_line[-1]=='didntLike':
            class_label_vector.append(1)
        elif list_from_line[-1]=='smallDoses':
            class_label_vector.append(2)
        elif list_from_line[-1]=='largeDoses':
            class_label_vector.append(3)
        index+=1
    return return_character_matrix,class_label_vector
"""
函数说明:对数据进行归一化
:parameter
character_matrix:特征矩阵
returns:
norm_dataset:归一化后的特征矩阵
ranges:数据范围
min_vals:数据的最小值
"""
def auto_norm(dataset):
    #获取数据的最小值
    min_vals=dataset.min(0)
    max_vals=dataset.max(0)
    #最大值和最小值的范围
    ranges=max_vals-min_vals
    norm_dataset=np.zeros(np.shape(dataset))
    # 返回特征矩阵的行数
    m = dataset.shape[0]
    norm_dataset=dataset-np.tile(min_vals,(m,1))
    #除以最大和最小值的差，得到归一化数据
    norm_dataset=norm_dataset/np.tile(ranges,(m,1))
    #返回归一化数据结果,数据范围，最小值
    return norm_dataset,ranges,min_vals
"""
```

```
函数说明：分类器测试函数
Parameters:
无
returns:
norm_dataset:归一化后的特征矩阵
ranges:数据范围
min_vals:数据最小值
"""
def dating_class_test():
    #打开文件名
    file_name="./2022-06-08data/datingTestSet.txt"
    #将返回的特征矩阵和分类向量分别存储到dating_mat和dating_labels中
    dating_mat,dating_labels=file_matrix(file_name)
    #读取所有数据的10%
    data_rate=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
    norm_mat,ranges,min_vals=auto_norm(dating_mat)
    #获取norm_matrix的行数
    m=norm_mat.shape[0]
    #10%的测试数据的数量
    number_test_data=int(m*data_rate)
    #分类错误计数
    error_number=0.0
    for i in range(number_test_data):
        #前number_test_data个数据作为测试集，后m-number_test_data个数据作为
                                                训练集
        classify_result=classify(norm_mat[i,:],norm_mat[number_test_data:m,
                                                :],dating_labels[
                                                number_test_data:m],3)
        print('分类结果:%d\t真实类别:%d'%(classify_result,dating_labels[i])
                                                )
        if classify_result!=dating_labels[i]:
            error_number+=1.0
    print('错误分类:%f%%'%(error_number/float(number_test_data)*100))

def classify_person():
    file_name='./2022-06-08data/datingTestSet.txt'
    result_list = ['didntLike', 'smallDoses', 'largeDoses']
    fly_miles = float(input('每周飞行的时长为:'))
```

```
    game_hours = float(input('每天玩游戏的时间为:'))
    ice_cream = float(input('每年吃的冰淇淋为:'))
    dating_mat, dating_labels = file_matrix(file_name)
    norm_mat, ranges, min_vals = auto_norm(dating_mat)
    input_array = np.array([fly_miles, game_hours, ice_cream])
    class_result = classify((input_array - min_vals) / ranges, norm_mat,
                                          dating_labels, 3)
    print('可能喜欢这个人:', result_list[class_result - 1])

"""
将图像转换为测试向量
"""
def image_vector(file_name):
    return_vector=np.zeros((1,1024)) #返回一个1*1024的向量
    file=open(file_name)
    for i in range(32):
        line_string=file.readline() #每次读一行
        for j in range(32):
            return_vector[0,32*i+j]=int(line_string[j]) #这个向量本身也完全
                                                    是由0,1构成，相当于将原来
                                                    的矩阵每一行首尾相连
    return return_vector

"""
识别手写数字
"""
#识别手写数字
def handwritingClassTest():
    hwLabels = []
    training_file_list = os.listdir('./2022-06-08data/trainingDigits') #导
                                            入训练集，‘trainingDigits’是一
                                            个文件夹
    m = len(training_file_list) #计算训练样本个数
    training_mat = np.zeros((m,1024)) #初始化数据集，将所有训练数据用一个m
                                            行，1024列的矩阵表示
    for i in range(m):
        file_name_str = training_file_list[i]   #获得所有文件名，文件名格式
                                            ‘x_y.txt’,x表示这个手写数字
                                            实际表示的数字（label）
```

```
        file_str = file_name_str.split('.')[0]  #去除 .txt
        class_num_str = int(file_str.split('_')[0])   #classnumber为每个样
                                                本的分类，用‘_’分割，取得
                                                label
        hwLabels.append(class_num_str)  #将所有标签都存进hwLables[]
        training_mat[i,:] = image_vector('./2022-06-08data/trainingDigits/%
                                                s' % file_name_str) #将文件转
                                                化为向量后存入trainingMat[],
                                                这里展现了灵活的文件操作
    test_file_list = os.listdir('./2022-06-08data/testDigits')  #迭代测试集
    error_count = 0.0
    m_test = len(test_file_list)
    for i in range(m_test):
        file_name_str = test_file_list[i]
        file_str = file_name_str.split('.')[0]  #去除 .txt
        class_num_str = int(file_str.split('_')[0])
        vector_under_test = image_vector('./2022-06-08data/testDigits/%s' %
                                                 file_name_str)  #这部分针对
                                                测试集的预处理和前面基本相同
        classify_result = classify(vector_under_test, training_mat,
                                                hwLabels, 3)    #使用算法预测
                                                样本所属类别，调用了前面写的
                                                classify0()函数
        print ("分类结果:%d,真实类别:%d" % (classify_result, class_num_str)
                                                )
        if (classify_result != class_num_str): error_count += 1.0       #算
                                                法结果与样本的实际分类做对比
    print ("\nthe total number of errors is: %d" % error_count)
    print ("\nthe total error rate is: %f" % (error_count/float(m_test)))
```

4.2　主程序

```
import numpy as np
import pandas as pd
import KNN_practice
file_name="./2022-06-08data/datingTestSet.txt"
path=file_name
df=pd.read_csv(path)
```

```
print(df)
dating_mat,dating_labels=KNN_practice.file_matrix(file_name)
print('=='*30)
print('dating_mat=',dating_mat)
print('=='*30)
print('dating_labels=',dating_labels)
print('=='*30)
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib.font_manager import FontProperties  # 导入FontProperties
# matplotlib其实是不支持显示中文的 显示中文需要一行代码设置字体
mpl.rcParams['font.family'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False
fig,ax=plt.subplots(figsize=(10,6))
ax.scatter(dating_mat[:,1],dating_mat[:,2],15.0*np.array(dating_mat[:,1]),
                                        15.0*np.array(dating_mat[:,2]))
plt.xlabel('玩视频游戏所耗时间百分比')
plt.ylabel('每周所消费的冰淇淋公升数')
norm_mat,ranges,min_vals=KNN_practice.auto_norm(dating_mat)
print('=='*30)
print('norm_mat=',norm_mat)
print('=='*30)
print('ranges=',ranges)
print('=='*30)
print('min_vals=',min_vals)
print('=='*30)
m=4
ranges=np.array([0,1,2,3,4,5,6])
print('=='*50)
print('tile(ranges,(3,1))=',np.tile(ranges,(3,1)))
print('=='*50)
print('tile(ranges,3)',np.tile(ranges,3))
print('=='*50)
print(KNN_practice.dating_class_test())
print('=='*30)
print(KNN_practice.classify_person())
test_vector=KNN_practice.image_vector('./2022-06-08data/testDigits/0_13.txt
                                        ')
print(test_vector[0,0:31])
```

```
print(test_vector[0,32:63])
print(KNN_practice.handwritingClassTest())
```