

第 2 章 离散变量与分类

本章学习目标

- 熟练掌握常用的分类算法
- 了解各类分类器的优劣

本章向读者介绍学术界及业界中常用的一系列机器学习分类器算法,同时介绍了几种分类算法的差异及优劣所在。分类器是对样本进行分类的方法的统称,包含 k 近邻分类器、决策树、对率回归、支持向量机、神经网络等算法。

2.1 K 近邻(KNN)分类器

一种懒惰的学习算法

KNN 是一种常用的监督学习算法,也是所有机器学习算法中最简单的一种分类和回归算法。KNN 不是从训练数据集中学习概率分布,而是靠记忆训练过的数据集来完成任务。在训练阶段,仅仅将样本保存起来,训练时间开销为 0。

2.1.1 KNN 算法简介

俗话说,物以类聚,人以群分。判别一个人是一个什么样的人,常常可以从他身边的朋友入手,所谓观其友,而识其人。KNN 算法是物以类聚、人以群分思想的体现之一,判断待分类样本的类别,则从该样本的邻居出发。

KNN 的全称是 K-Nearest Neighbors,即 k 个最近的邻居。从字面意思可以猜测到,未知样本 \mathbf{x} 的类别和 k 个最近的邻居有关。显然, k 的取值是一个关键因素。那么,什么是 k 个最近的邻居呢?未知样本 \mathbf{x} 的类别是如何由 k 个最近的邻居的类别来确定呢?

用图形直观表示,如图 2.1.1 所示,样本共有三种已知类别,分别为五角星类别、三角形类别及圆圈类别。用方框表示的样本为需要预测类别的样本 \mathbf{x} 。

假设 $k=7$,那么 KNN 就会寻找与样本 \mathbf{x} 最近的 7 个样本,看这 7 个样本中属于哪个类别的最多。显然,属于五角星的样本最多,因此,将样本 \mathbf{x} 的类别判别为五角星。因此,在本例中,对于未知样本 \mathbf{x} ,如果其 k 个 ($k=7$) 最近的邻居中大多数属于类别 i ,则样本 \mathbf{x} 也属于类别 i 。

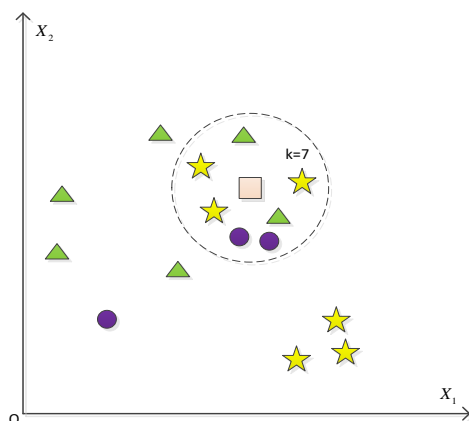


图 2.1.1 k 为 7 时样本类别的预测

然而，样本 \mathbf{x} 的类别和 k 的取值也有关系。假设 $k=9$ ，情形又会如何呢？如图 2.1.2 所示，KNN 会寻找与样本 \mathbf{x} 最近的 9 个样本，看这 9 个样本中属于哪个类别的最多。显然，属于三角形的样本最多，因此，将样本 \mathbf{x} 的类别判别为三角形。从本例可看出， k 的取值是个重要因素，影响到了样本 \mathbf{x} 的分类结果。

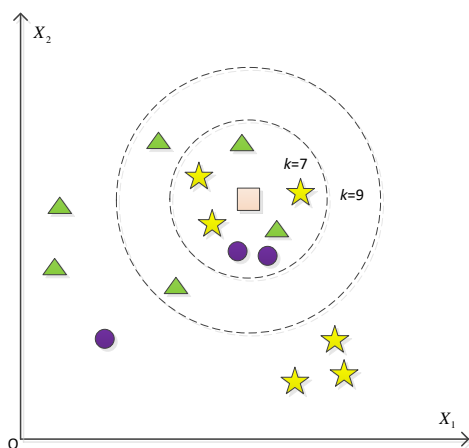


图 2.1.2 k 为 9 时样本类别的预测

综合以上两例，分析可得，待预测样本 \mathbf{x} 的类别取决于三个因素，分别为 k 的取值、距离的计算及决策规则的制定。

2.1.2 KNN 算法的距离计算

对于待预测样本 \mathbf{x} ，KNN 算法的基本思路就是将其与其他所有样本点进行距离计算，并从中选出 k 个“最近”的样本。实际上，空间中点到点的距离度量有多种方式，如欧氏距离、曼哈顿距离、Minkowski 距离、马氏距离、余弦相似度等。Minkowski 距离是欧氏距离和曼哈顿距离的推广，两个样本 $\mathbf{x}^{(i)}$ 和 $\mathbf{x}^{(j)}$ 的 Minkowski 距离为：

$$d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \left[\sum_k |\mathbf{x}_k^{(i)} - \mathbf{x}_k^{(j)}|^p \right]^{\frac{1}{p}} \quad (2.1.1)$$

若 $p = 1$ ，式 (2.1.1) 为曼哈顿距离；若 $p = 2$ ，式 (2.1.1) 为欧氏距离；若 $p \rightarrow \infty$ ，式 (2.1.1) 为切比雪夫距离。

马氏距离定义为两个服从同一分布并且其协方差矩阵为 Σ 的随机变量之间的差异程度。向量 \mathbf{x} 到向量 \mathbf{y} 的马氏距离为：

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \Sigma^{-1} (\mathbf{x} - \mathbf{y})} \quad (2.1.2)$$

式中, \mathbf{x} 和 \mathbf{y} 为服从同一分布的 n 维向量, $\mathbf{\Sigma}$ 是协方差矩阵, 其大小为 $n \times n$, $\mathbf{\Sigma}^{-1}$ 为 $\mathbf{\Sigma}$ 的逆矩阵。

向量 \mathbf{x} 到向量 \mathbf{y} 的余弦相似度为:

$$\text{cossim}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \cdot \sqrt{\sum_{i=1}^n y_i^2}} = \cos \theta \quad (2.1.3)$$

式中, \mathbf{x} 和 \mathbf{y} 都是 n 维向量, x_i 与 y_i 分别为 \mathbf{x} 和 \mathbf{y} 的第 i 个分量, θ 为向量 \mathbf{x} 与向量 \mathbf{y} 的夹角。

余弦相似度, 通过计算两个向量的夹角余弦值 $[-1, 1]$ 来度量它们之间的相似性, 值越大表明两个向量越相似。当两个样本 (例如两条新闻文本) 夹角的余弦值越接近 1, 两者就越相似; 相反, 越接近 -1 则越不相似。比较起来, 余弦相似度更注重两个向量在方向上的差异, 而欧氏距离注重的则是空间各点的绝对距离, 因此跟样本点的位置坐标直接相关。余弦相似度在表达两个特征向量之间的关系时用处较大, 可广泛用于人脸识别、推荐系统等应用。

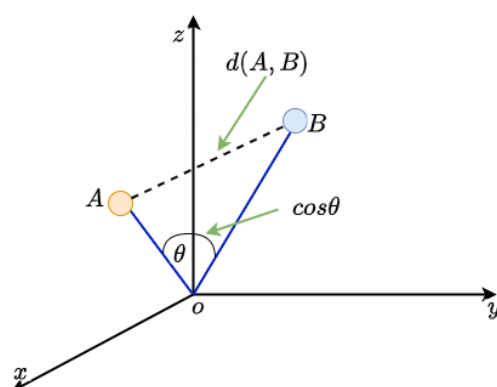


图 2.1.3 欧氏距离与余弦相似度的区别

如图 2.1.3 所示, $d(A, B)$ 表示空间点 A 和 B 之间的欧氏距离, $\cos \theta$ 表示空间点 A 和 B 之间的余弦相似度。从图 2.1.3 可看出, 欧氏距离跟各个点的位置坐标直接相关, 而余弦相似度衡量的是空间向量的夹角, 体现为方向上的差异, 而不是位置。如果保持 A 点位置不变, B 点朝原方向远离坐标轴原点, 此时由于夹角 θ 不变, 则余弦相似度 $\cos \theta$ 保持不变, 而 A 、 B 两点的欧氏距离显然在变大, 这就是欧氏距离和余弦相似度的不同之处。

除了对距离度量本身的选择, 一般还需要对数据进行归一化或标准化, 以确保每个特征都能对距离计算起同等的作用。例如, 鸢尾花数据集的花样本, 虽然其 4 个特征都是以厘米为单位, 但由于“花瓣的长度和宽度”相比“花萼的长度和宽度”取值范围更大, 为了消除这种取值范围不同 (尺度不同) 造成的影响, 需要对特征进行归一化或标准化。

2.1.3 KNN 算法的 k 值选择

正确选择 k 值对在过拟合与欠拟合之间找到恰当的平衡至关重要。简单而言, 当模型对训练数据拟合得太好时, 会发生过拟合。也就是说, 模型在训练集中的准确率很高, 但在验证集中的准确率偏低, 就预示着发生了过拟合的现象。当模型无法很好地拟合数据时, 就会发生欠拟合。具体而言, 如果模型在训练集中的准确率很低, 则预示着出现了欠拟合的现象。过拟合和欠拟合都会导致模型对新数据集的预测效果不佳, 因此在实践中要避免其发生。

从 2.1.1 节的例子中也可看出, k 的取值非常重要。那么该如何确定 k 的取值呢?

如果 k 值较小, 相当于在一个较小的邻域内选择训练样本进行类别预测, 只有距待预测样本 \mathbf{x} 较近的训练样本才会对预测结果起作用。其缺点是预测结果对距离较近的训练样本非常敏感, 如果距离近的样本点含有噪声, 则预测结果易出错。换句话说, k 值的减小意味着整体模型变得复杂, 容易发生过拟合。

如果 k 值较大, 相当于在一个较大的邻域内选择训练样本进行类别预测, 距待预测样本 x 较远的训练样本也会对预测结果起作用。这时可对临近样本点的噪声起到一定的“缓冲”作用, 但较远的训练样本易给预测类别带来偏差。 k 值增大意味着整体的模型变得简单。原因在于, 假设 k 取最大值, 即训练集的大小, 那么无论输入的待预测样本是什么, 都会将其类别预测为训练集中样本个数最多的类, 这显然不合理。此时的模型过于简单, 完全忽略了训练集中大量的有用信息, 忽略了训练样本的具体概率分布。

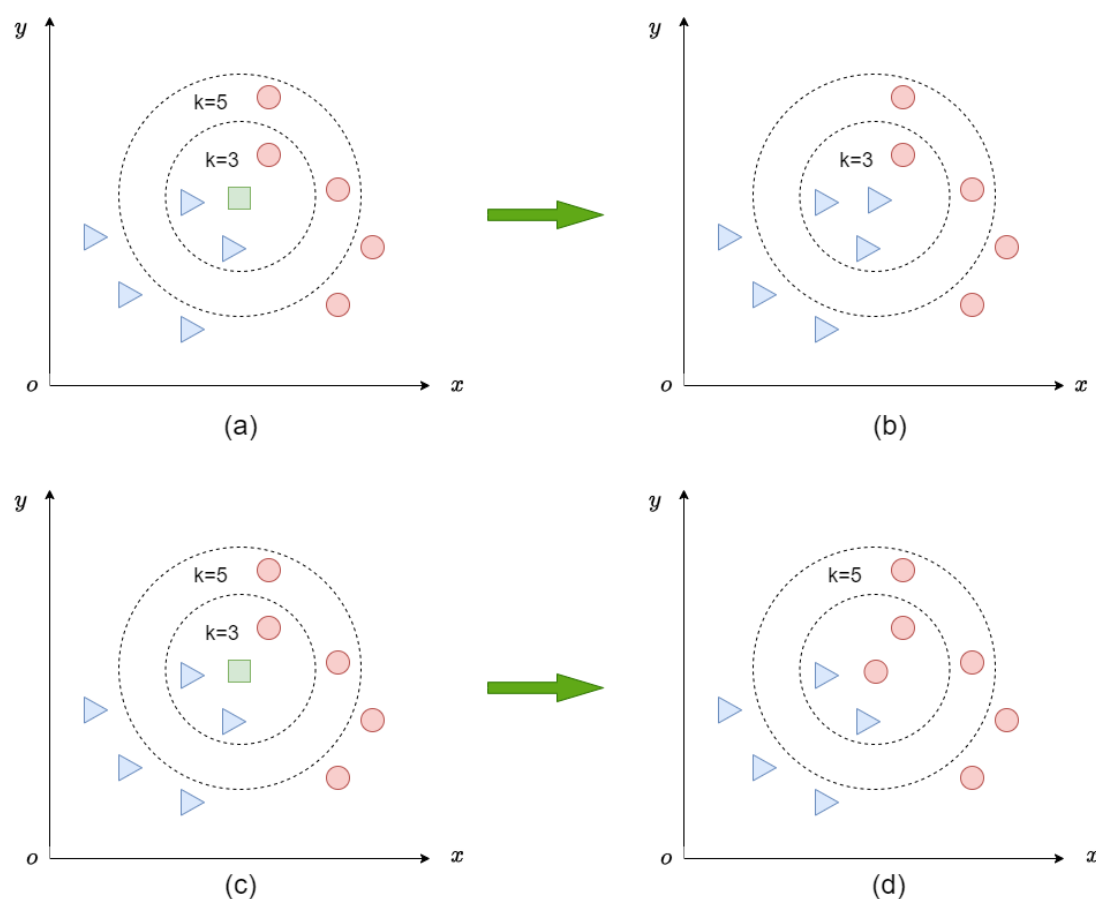


图 2.1.4 不同 k 值对分类结果的影响

图 2.1.4 展示了不同 k 值对 KNN 分类结果的影响。图 2.1.4 (a) 中的方块样本为待分类样本, 从 2.1.4 (b) 可看出, 当 $k = 3$ 时, 判断离方块样本最近的 3 个样本中哪一类图形最多, 则方块样本就被分为哪一类。显然, 方块样本被分类为三角类别。当 k 值改变时, 分类结果也会发生改变。从 2.1.4 (d) 可看出, 当 $k = 5$ 时, 判断离方块样本最近的 5 个样本中哪一类图形最多, 则方块样本就被分为哪一类。显然, 方块样本被分类为圆圈类别。

在实际应用中, k 一般取一个较小的数值, 通常采用“交叉验证法”来选取最优的 k 值 (见 1.2.3 节)。交叉验证将训练数据按照一定方式分成训练集和验证集, 然后利用验证集去评估最好的 k 值。其核心思想就是把一些可能的 k 逐个尝试, 然后选出效果最好的 k 值。

2.1.4 KNN 算法的决策规则

决策规则即以什么样的方式判定最终的类别预测结果。在分类预测时, 一般采用“多数表决议”或“加权多数表决议”。在 2.1.1 节的例子中采用的是通过“多数表决议”对结果进行分类。所谓多数表决议, 即使用待测样本的最近的 k 个邻居中出现次数最多的类别作为预测结果。

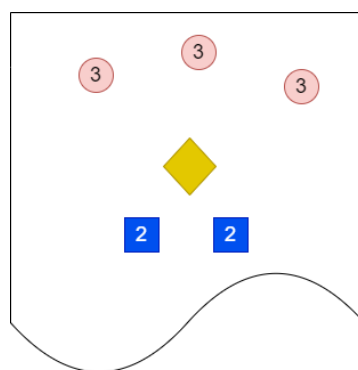


图 2.1.5 不同决策规则下样本类别的预测

假设图 2.1.5 中菱形样本表示待预测样本, 圆表示一类, 方块表示一类, 2 和 3 表示每个样本到待预测样本的距离。对于多数表决法而言, 每个邻近样本的权重是一样的, 也就是说最终预测的结果为出现类别最多的那个类。因此, 菱形样本被预测为圆类别。

对于“加权多数表决法”而言, 每个邻近样本的权重是不一样的, 一般情况下采用权重和距离成反比的方式来计算, 即 $w = \frac{1}{d}$, 也就是说最终预测结果为权重最大的那个类别。如图 2.1.5, 圆样本到待预测样本的距离为 3, 方块样本到待预测样本的距离为 2, 权重与距离成反比, 所以方块样本的权重比较大, 则待预测样本被预测为方块类别。更一般的“加权多数表决法”会将同一类别样本的权重加起来, 作为最终的得票数。比如图 2.1.5 中, 方块的总票数为 $1/2+1/2=1$, 而圆的总票数为 $1/3+1/3+1/3=1$, 两个类别票数相同, 从而将菱形样本预测为方块或圆都可以。

2.1.5 KNN 算法小结

KNN 的工作机制非常简单: 给定测试样本, 基于某种距离度量找出训练集中与其最靠近的 k 个训练样本, 然后基于这 k 个邻居的信息进行预测。在分类任务中, 一般使用这 k 个邻居出现次数最多的类别作为预测结果。

KNN 分类算法可以总结为以下几个步骤:

- (1) 选择 k 值和一个距离度量。
- (2) 找到待分类样本的 k 近邻。
- (3) 以投票机制确定分类标签。

KNN 算法是一种非参, 惰性的算法模型。其中, “非参”并不意味着算法不需要参数 (显然, 算法涉及超参数 k 的选取), 而体现在算法不会对数据做出任何的假设。与之相对应的是线性回归模型 (见第 3 章), 该模型总是假设数据的分布是一条直线。也就是说, KNN 建立的模型是由实际的数据决定的, 这比较符合实际, 毕竟实际的数据分布与理论的分布假设常常并不严格相符。“惰性”在于, KNN 分类算法的训练过程就是简单的将训练样本存下来。相比较而言, 本章后面几节将要依次介绍的决策树、对率回归这些分类算法都需要先基于训练集进行大量的训练, 才能得到一个可用于预测的算法模型。

KNN 算法的优点为简单, 在编程上易于实现; 对数据没有假设, 无需估计参数; 无需训练, 对异常值不敏感。其中, 对异常值不敏感体现为, 异常样本一般数目较少, 而 KNN 采用了投票机制, 因此, 异常样本的类别一般不至于对分类结果产生严重的影响。

KNN 算法在实际使用中也会存在一些问题。它对规模超大的数据集计算量大、内存开销大。假设样本数量为 N , 单个样本特征的维数为 p , 则对一个待测样本进行 KNN 分类, 所需的时间复杂度为 $O(Np)$ 。原因在于, 对任何一个待测样本进行分类时, 需要循环所有的训练样本, 复杂度为 $O(N)$ 。另外, 当计算两个样本之间距离的时候, 此处的复杂度依赖于样本的

维数, 为 $O(p)$ 。将循环样本的过程看做外层循环, 计算样本之间距离看作内层循环, 所以总的复杂度为二者的乘积, 即 $O(Np)$ 。

KNN 对高维数据集、稀疏数据集拟合欠佳, 容易引起“维数灾难”(见 4.1 节的讨论)。原因在于, 在高维空间, 数据变得异常稀疏, 使得即使是最近的邻居数据点, 计算出的数据点之间的距离也变得很远。这导致了随着变量维度的增加, 训练集所要求的数据量呈指数级的增长, 计算量也随之变大。这时可先对数据进行降维, 如采用 PCA 等降维方法。

此外, KNN 在样本类别不平衡的情形下, 预测准确度将降低。如图 2.1.6, 对于样本 X , 通过 KNN 算法, 显然可以得到 X 应属于红点。但对于样本 Y , 通过 KNN 算法似乎可判别 Y 应属于蓝点, 而这个结论从图中看来并没有说服力。此例属于样本类别不平衡的一个表现, 即一个类的样本数量很大, 而其他类样本数量很小时, 很可能导致当输入一个待测样本时, 该样本的 k 个邻居中大数量类的样本占多数。但是, 待测样本更靠近小数量类的样本。这种情况下, 更倾向于认为该待测样本属于数量小的样本所属的类, 因此可采用“加权多数表决策法”来进行判别: 与该待测样本距离小的邻居设置相对较大权值, 与该样本距离大的邻居则设置相对较小权值, 由此, 将距离远近的因素也考虑在内, 就可以避免因样本类别不平衡而导致误判的情况。

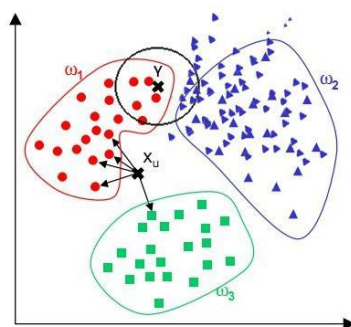


图 2.1.6 KNN 在样本不平衡情形下的分类效果

KNN 是一种基于实例的学习, 训练数据并没有形成一个“模型”, 仅将待预测样本和训练数据逐一比较。因此, 在一些常见的应用场合中, KNN 使用得并不多。然而, KNN 算法仍然具有相当的理论价值, 是一种非常经典、原理十分易于理解的算法, 其涉及的距离计算、决策规则可为更复杂模型提供参考。例如, 距离计算的“余弦相似度”可在文本分类的“词袋”模型中使用。例如一篇文章一共出现了 5000 个词, 因此用一个 5000 维度的向量 X 表示这篇文章, 每个维度代表词语出现的数目。另外一篇文章也恰好共出现了这 5000 个词, 并用向量 Y 表示该文章, 则这两篇文章的相似度可以用余弦相似度来度量。

2.1.6 KNN 核心代码

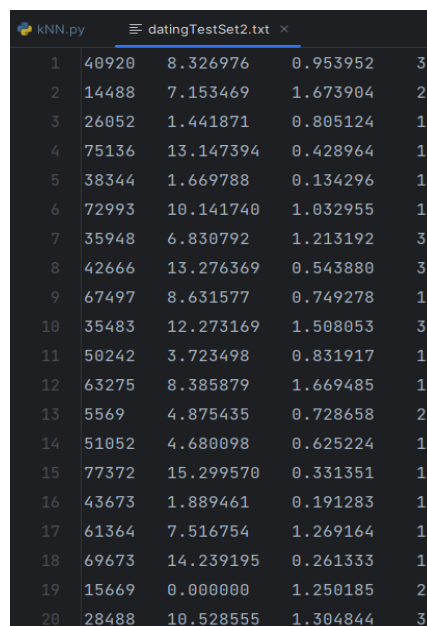
以线上交友应用为例, 解决流程及 Python3 代码实现如下:

- (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 画二维散点图。
- (4) 训练算法: 此步骤不适用于 k-近邻算法。
- (5) 测试算法: 使用线上交友应用的数据作为测试样本。
- (6) 使用算法: 编写简单的命令程序, 然后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

线上交友数据集存放在文件 `datingTestSet2.txt` 中, 如图 2.1.7 所示, 数据集中共有 1000

条样本，每条样本占据一行，图中仅显示前 20 条样本。每个样本有 3 个特征，即图中的前三列，分别表示“每年获得的飞行常客里程数”，“每周消费的冰淇淋公升数”，及“玩视频游戏所耗时间百分比”。图中第四列表示样本标签，取值分别为 1, 2, 3。其中，标签“1”表示不喜欢，标签“2”表示有较小可能喜欢，标签“3”表示有较大可能喜欢。



Index	Feature 1	Feature 2	Feature 3	Label
1	40920	8.326976	0.953952	3
2	14488	7.153469	1.673904	2
3	26052	1.441871	0.805124	1
4	75136	13.147394	0.428964	1
5	38344	1.669788	0.134296	1
6	72993	10.141740	1.032955	1
7	35948	6.830792	1.213192	3
8	42666	13.276369	0.543880	3
9	67497	8.631577	0.749278	1
10	35483	12.273169	1.508053	3
11	50242	3.723498	0.831917	1
12	63275	8.385879	1.669485	1
13	5569	4.875435	0.728658	2
14	51052	4.680098	0.625224	1
15	77372	15.299570	0.331351	1
16	43673	1.889461	0.191283	1
17	61364	7.516754	1.269164	1
18	69673	14.239195	0.261333	1
19	15669	0.000000	1.250185	2
20	28488	10.528555	1.304844	3

图 2.1.7 线上交友数据集

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。代码如下：

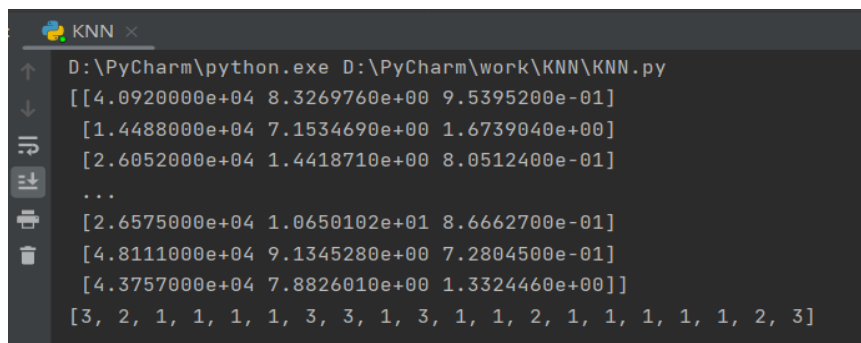
```
def file2matrix(filename):
    fr = open(filename)                # 打开文件
    arrayOLines = fr.readlines()       # 读取文件内容
    numberOflines = len(arrayOLines)   # 得到文件行数
    returnMat = np.zeros((numberOflines, 3))  # 返回解析后的数据，创建返回的
    NumPy 矩阵
    classLabelVector = []              # 定义类标签向量
    index = 0
    for line in arrayOLines:           # 解析文件数据到列表
        line = line.strip()            # 截取掉所有的回车字符
        listFromLine = line.split('\t') # 用'\t'将行数据分割成元素列表
        returnMat[index, :] = listFromLine[0 : 3] # 选取前三个数据将他们存储到
        特征矩阵中
        # 把该样本对应的标签放置标签向量，顺序与样本集对应
        classLabelVector.append(int(listFromLine[-1]))
        index += 1
    return returnMat, classLabelVector
```

(1) 使用 `file2matrix` 读取文件数据。

使用函数 `file2matrix` 读取文件数据，必须确保文件 `datingTestSet2.txt` 存储在工作目录中。在 Python 命令提示符下输入下面命令：

```
>>> import kNN
>>> datingDataMat, datingLabels = kNN.file2matrix('datingTestSet2.txt')
>>> print(datingDataMat)
>>> print(datingLabels[0:20])
```

(2) 程序运行结果如图 2.1.8 所示。



```
KNN x
D:\PyCharm\python.exe D:\PyCharm\work\KNN\KNN.py
[[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]]
[3, 2, 1, 1, 1, 1, 3, 3, 1, 3, 1, 1, 2, 1, 1, 1, 1, 2, 3]
```

图 2.1.8 文件数据读取的运行结果

说明：成功导入 datingTestSet2.txt。现已从文本文件中导入了数据，并将其格式化为想要的格式。

2. 分析数据：使用 Matplotlib 创建样本散点图

下面这段代码用于数据的可视化，数据分析及散点图的创建。首先，使用 Matplotlib 制作原始数据的散点图，并采用色彩或其他记号来标记不同的样本分类，以更好地理解数据信息。其次，Matplotlib 库提供的 scatter 函数支持个性化标记散点图上的点，调用 scatter 函数时使用相应参数进行配置（对应如下代码）。此外，散点图使用 datingDataMat 矩阵的第 0、1 列数据，分别表示特征值“每年获得的飞行常客里程数”和“每周消费的冰淇淋公升数”。

```
import matplotlib.pyplot as plt

import kNN

import numpy as np

import matplotlib.font_manager as fm # 导入字体管理器

# 设置中文字体，SimHei 是黑体，您也可以使用其他中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode_minus'] = False # 解决负号 '-' 显示为方块的问题

fig = plt.figure()
ax = fig.add_subplot(111)

datingDataMat, datingLabels = kNN.file2matrix('datingTestSet2.txt')
```



```
datingLabels = np.array(datingLabels)

idx_1 = np.where(datingLabels == 1)

# 设置各散点颜色、大小和代表含义

p1 = ax.scatter(datingDataMat[idx_1, 0], datingDataMat[idx_1, 1],

                marker='o', color='r', label='不喜欢', s=30)

idx_2 = np.where(datingLabels == 2)

p2 = ax.scatter(datingDataMat[idx_2, 0], datingDataMat[idx_2, 1],

                marker='o', color='g', label='魅力一般', s=45)

idx_3 = np.where(datingLabels == 3)

p3 = ax.scatter(datingDataMat[idx_3, 0], datingDataMat[idx_3, 1],

                marker='o', color='b', label='极具魅力', s=60)

plt.legend(loc='upper left')

plt.xlabel("每年获取的飞行常客里程数") # 设置 x,y 轴所代表内容（显示的文字）

plt.ylabel("每周消费的冰淇淋公升数")

plt.show() # 显示散点图图像程序运行结果如图 2.1.9 所示。
```

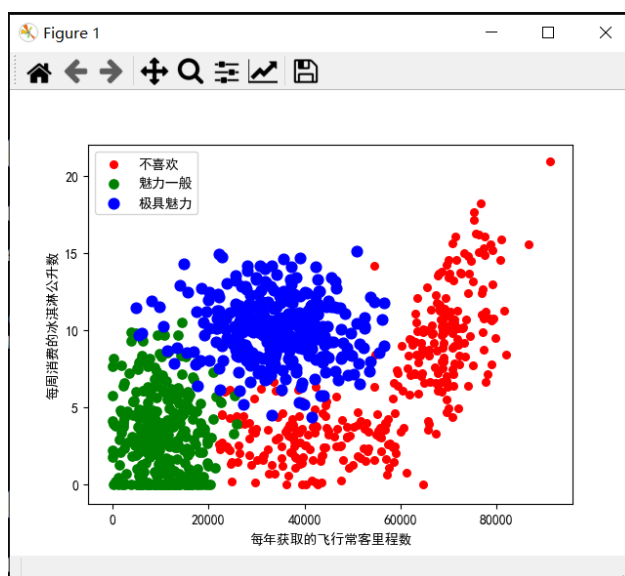


图 2.1.9 带有分类标签的样本散点图

由图 2.1.9 可见，采用合适的属性值可以得到较好的可视化效果。该图中清晰地标识了三个不同的样本类别区域，具有不同爱好的人其类别区域也不同。

3. 准备数据，归一化数值

表 2-1 为从交友数据集中随机采样的 4 条样本数据，可看出 3 个特征的取值范围差异较大。由于交友数据中“每年获得的飞行常客里程数”远大于特征值“每周消费的冰淇淋公升数”

和“玩视频游戏所耗时间百分比”，如果直接使用数据，会造成“飞行常客里程数”对分类结果的影响最大。但通常认为这 3 个特征是同等重要的，因此作为 3 个等权重的特征之一，“每年获得的飞行常客里程数”不应该严重地影响到分类结果。

表 2-1 交友数据集的 4 条样本数据

样本 序号	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	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，通常采用的方法是将数值归一化，如将取值范围处理为 0~1 或者 -1~1 之间。公式（2.1.4）可将任意取值范围的特征值转化为 0 到 1 之间的值：

$$newValue = \frac{oldValue - \min Value}{\max Value - \min Value} \quad (2.1.4)$$

其中， $\min Value$ 和 $\max Value$ 分别为数据集中的最小特征值和最大特征值。下面编写归一化函数 `autoNorm()` 将特征值转换为 0 到 1 的区间。

在函数 `autoNorm()` 中，将每列的最小值放在变量 `minVals` 中，将每列的最大值放在变量 `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。为了归一化特征值，必须使用当前值减去最小值，然后除以取值范围。

```
def autoNorm(dataSet):          # 归一化特征值，将数字特征值转化为 0~1 的区间
    minVals = dataSet.min(0)    # 存放每列最小值
    maxVals = dataSet.max(0)    # 存放每列最大值
    ranges = maxVals - minVals  # 最大值与最小值的差值
    normDataSet = (dataSet - minVals)/ranges #Numpy 广播机制
    return normDataSet, ranges, minVals
```

4. 构建 KNN 分类器

`Classify()` 函数有 4 个输入参数，即待分类的输入向量 `inX`，训练样本集 `dataSet`，训练样本标签 `labels`，选择最近邻居的数目 `k`。标签向量的元素数目和样本矩阵 `dataSet` 的行数相同，即数据集的样本个数。程序中距离的计算方式为欧式距离，对应公式（2.1.1）。

该函数计算完待分类样本和所有训练样本的距离后，将距离按照从小到大的顺序排序，然后确定对应前 `k` 个最小距离的训练样本的类别标签。代码“`sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)`”将字典 `classCount` 分解为元组列表，使用运算符模块的 `itemgetter` 方法，按照第二个元素（多数表决的票数）的次序对元组进行从大到小的排序，最后返回发生频率最高的元素标签。

```
def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    diffMat = inX - dataSet #Numpy 广播机制
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances**0.5
```

```

sortedDistIndicies = distances.argsort()
classCount={}
for i in range(k):
    voteLabel = labels[sortedDistIndicies[i]]
    classCount[voteLabel] = classCount.get(voteLabel, 0) + 1
sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)
return sortedClassCount[0][0]

```

5. 测试算法：作为完整程序验证分类器

机器学习算法一个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。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。

```

def datingClassTest():          # 测试算法：作为完整程序验证分类器
    hoRatio = 0.10
    datingDataMat, datingLabels = file2matrix('datingTestSet2.txt')
    normMat, ranges, minVals = autoNorm(datingDataMat)
    m = normMat.shape[0]
    numTestVecs = int(m * hoRatio)
    errorCount = 0.0
    for i in range(numTestVecs):
        classifierResult = classify0(normMat[i, :], normMat[numTestVecs:m, :],
                                     datingLabels[numTestVecs:m], 3)
        print ("the classifier came back with: {}, the real answer is {}".format(classifierResult, datingLabels[i]))
        if (classifierResult != datingLabels[i]): errorCount += 1.0
    print("the total error rate is: {}".format(errorCount / float(numTestVecs)))

```

(1) 测试函数，计算错误率：

```

>>> import kNN
>>> kNN.datingClassTest()

```

(2) 程序运行结果如图 2.1.10 所示。

```

the classifier came back with: 1, the real answer is 1
the classifier came back with: 2, the real answer is 2
the classifier came back with: 3, the real answer is 3
the classifier came back with: 3, the real answer is 1
the classifier came back with: 3, the real answer is 3
the classifier came back with: 1, the real answer is 1
the classifier came back with: 2, the real answer is 2
the classifier came back with: 2, the real answer is 2

```

图 2.1.10 分类器的测试结果

说明：分类器处理交友数据集的错误率是 5%，表明分类器可以通过输入未知对象的属性信息，来帮助判定某一对象的可交往程度：讨厌、一般喜欢、非常喜欢。

6. 使用算法构建完整可用系统

```
def classifyPerson():    #使用算法：构建完整可用的系统
    resultList = ['讨厌', '一般喜欢', '非常喜欢']    # 类标签列表
    # 用户输入不同特征值
    precentTats = float(input("玩视频游戏所占时间百分比?"))
    ffMiles = float(input("每年获得的飞行常客里程数?"))
    iceCream = float(input("每周消费的冰淇淋公升数?"))
    # 打开文件并处理数据
    datingDataMat, datingLabels = file2matrix('datingTestSet2.txt')
    normMat, ranges, minVals = autoNorm(datingDataMat)    # 归一化训练集
    inArr = np.array([precentTats, ffMiles, iceCream])
    # 创建测试集数组
    norm_in_arr = (inArr - minVals) / ranges    # 归一化测试集
    classifierResult = classify0(norm_in_arr, normMat, datingLabels, 3)
    # 返回分类结果
    print("你对这个人的感觉可能是: ", resultList[classifierResult - 1])    # 输出结果
```

(1) 加载 KNN 模块，实现算法可用：

```
>>> import kNN
```

```
>>> kNN.classifyPerson()
```

(2) 程序运行结果如图 2.1.11 所示。

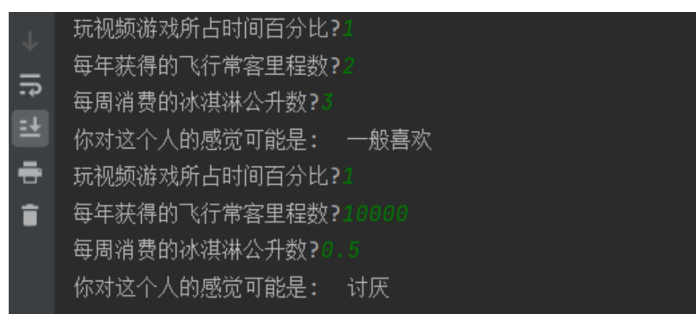


图 2.1.11 系统的分类结果

习题 2.1

1. 为什么要对数据做归一化？
2. 简述马氏距离与欧氏距离的区别和联系。
3. 在线上交友应用中将欧氏距离度量改变为马氏距离，比较实验结果的异同。
4. 基于线上交友数据集 `datingTestSet2.txt` 验证数据归一化对 kNN 分类精度的影响，根据你的验证结果你能得出什么结论。
5. 基于线上交友数据集 `datingTestSet2.txt`，尝试搜索 kNN 模型最佳的 k 值。
6. 在 2.1.5 节谈到了 kNN 模型的计算复杂度，请查阅文献了解常用的一些加速算法，比如 [KNeighborsClassifier — scikit-learn 1.5.1 documentation](#)。