第117天：机器学习算法之 K 近邻

原创 轩辕御龙 Python技术 1月17日

所谓“K 近邻（K-nearest neighbor，K-NN）”，顾名思义，指的是“K 个最近的邻居”，属于一种监督学习的方法。

1. 工作原理

简单地介绍一下 K 近邻算法的工作机制：首先给定一组训练集，作为算法的参照；然后给出特定的测试对象，也就是不带标签的测试数据，算法会在训练集中找到某种意义上与之最接近的 K 个训练数据，并根据这 K 个训练数据的标签来判定测试数据的类型（分类问题）或数值（回归问题）。

从 K 近邻算法的原理可以看出，得到训练数据之后其实并不存在所谓的“训练过程”，我们只需要“守株待兔”，等待外部输入一个测试数据，再与训练数据进行比较即可。这也是 K 近邻算法被称作“懒惰学习（lazy learning）”算法的原因。

1.1 距离度量

这里我们说的“某种意义上”，其实指的就是某种距离度量的方法。所谓“距离”，可以简单地理解为两个数据之间的差别，我们使用距离度量的方法可以定量地求出两个数据之间的差别到底有多大。其中我们最熟悉的一种距离度量方法就是“欧氏距离”，也就是我们最熟悉的“直线距离”。

在二维平面上，欧氏距离就表现为勾股定理。对于任意 N 维数据 $x=(x_1,x_2,\ldots,x_n)$ 和 $y=(y_1,y_2,\ldots,y_n)$，欧氏距离的通用计算公式为：

使用这个公式，我们可以用数值精确地衡量任意给定数据之间的差异程度。

此外还有诸如曼哈顿距离、（国际象棋）棋盘距离等各种距离度量方式。曼哈顿距离其实就是两点间各维度距离之差的和，就像在一个规划整齐的街区开车一样。

1.2 直观解释

通俗地讲，K 近邻算法的中心思想就是一种我们人人都明白、大家都认可的生活经验：人以类聚，物以群分。

对于特定的某个人，要尽快地对他有一个全面的认识，我们通常习惯观察他的朋友圈（咳，注意不是指微信朋友圈），与他交往最多的几个人基本上就能够反映他本人的大部分特质。

对于现实世界的其他对象也一样，这些对象的抽象——数据——当然也不例外。对于给定的测试数据，我们可以考察与

之相似度最高的几个数据，这些数据普遍具有的特征，我们自然而然地认为也会是这个测试数据的特征。

1.3 关于 K 值

K 值的选取看起来好像是比较随便的，但其实也是一门学问。也可以认为是 K 近邻算法的一个超参数，也就是由研究者手动设置、不为算法自动调整的参数。

K 值选得太大了固然不好。最极端的情况，如果不管不顾直接设置 K = N，也就是训练数据总的样本数，那么此时不论给定什么样的测试数据，都会得出相同的结果——算法会直接取训练数据样本中存在最多的标签作为测试数据的标签。这样一来，数据之间的差异完全被抹杀，体现不出任何的分类或是回归的价值。“千人一面”并不是我们追求的社会主义。

那 K 选得太小又有没有问题呢？还是有的。同样考虑最极端的情况——K = 1，这也就是 K 近邻算法的一个特例，所谓的“最近邻算法”。当测试数据的标签仅仅取决于与之最接近的一个训练数据时，我们都知道事情肯定遭了糕了——某个数据的偏差原本只是个例，但却好巧不巧地影响到了我们对测试数据的判断。

从以上分析我们可以得出一个社会意义上的教训：独裁是不好滴，盲目扩大的民主同样是遗祸无穷滴~ 集中民主是个好东西。

书归正传，认真地说，通过以上分析，我们可以得出一个结论：**K 的值不适宜太大，但也不应当过小**。这就比较麻烦了，因此一方面，K 值的设置是一个经验性的工作；另一方面，我们可以利用手头的训练数据，随机留出一部分作为测试数据来进行反复测试，从而选取一个较为适宜的 K 值。并且通常来说，这个“适宜的 K 值”不会太大。

1.4 决策方式

K 近邻算法的决策方式很多，我们简单介绍两种适用于不同问题的方式。

1）分类问题

对于分类问题，一般采用**多数表决制**，也就是说，K 个近邻中，占比最大的标签即为测试数据的标签。

2）回归问题

对于回归问题，可以采用**加权平均法**，也就是说，K 个近邻中，根据与测试数据的距离远近分别赋予不同的权值，然后求平均。

2. 算法实现

> 为减小阅读负担，本文仅给出分类问题的 K 近邻算法实现。回归问题的实现留待读者探索。
>
> 另外由于作者水平所限，代码如有问题还望读者不吝指正。

严格地讲，根据李航老师《统计学习方法》一书的内容，正式应用 K 近邻算法的时候还应当构建 kd 树，以方便检索近邻，提高算法效率。毕竟，在真正的工业应用上，算法面对的都是百万量级及以上的数据，如果不使用精心设计的数据结构来提升检索效率，将会带来成本的大幅提高和竞争劣势。

但是此处我们仅仅是为了了解 K 近邻算法，对该算法有一个感性的体会，因此就不涉及 kd 树的实现了（其实是作者嫌这部分有点麻烦哈哈哈哈——当然也是为了减轻读者阅读负担，否则还需要再介绍一下 kd 树的原理，篇幅就有点冗长了）。

2.1 划分数据

我们使用一个经常被作为机器学习算法基准测试的数据集——鸢尾花数据集，作为本文的示例数据集。

读者可从 https://www.kaggle.com/uciml/iris 下载，也可从本文配套代码获取。

简单介绍一下这个鸢尾花数据集，总共有 150 份鸢尾花的各项尺寸数据，均匀分属三种不同的鸢尾花，即 setosa、versicolor、virginica 三种鸢尾花各 50 份数据（作者水平所限，对于具体的种名就不做翻译了，以免造成误解，大家知道是三种不同的鸢尾花就好）。

其中的尺寸数据包括萼长、萼宽、瓣长、瓣宽四项，单位均为厘米，对应于每份数据，都有一个相应的类别标签以及数据本身在数据集中的 id。部分数据示例如下：

```
Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species
1,5.1,3.5,1.4,0.2,Iris-setosa
2,4.9,3.0,1.4,0.2,Iris-setosa
...
51,7.0,3.2,4.7,1.4,Iris-versicolor
52,6.4,3.2,4.5,1.5,Iris-versicolor
...
101,6.3,3.3,6.0,2.5,Iris-virginica
102,5.8,2.7,5.1,1.9,Iris-virginica
...
```

我们要做的就是根据给定的四项长度数据，通过 K 近邻算法来判断某个实例属于哪一种鸢尾花。

而现在的一个问题是，我们只有一个数据集，无法另外找到一个真实的鸢尾花数据。因此我们需要对原始数据集进行划分，用其中的一部分作为训练集，另一小部分作为测试集。这个比例由读者自行抉择，我们选择 4:1 的比例，即每中鸢尾花选择 40 份数据作为训练集，留出 10 份作为测试集。

注意，此处训练集和测试集的选择要具有随机性，以此尽量避免过多引入不必要的人为误差。

首先，读取数据：

```
import pandas as pd
iris = pd.read_csv('iris.csv').to_numpy()
```

`pd.read_csv()` 读取 CSV 文件后返回的是一个 DataFrame 类型的数据，使用 `to_numpy()` 方法可以将其转换为 Numpy 数组。

然后初始化各个数据的容器：

```
train_data = [] # 训练数据，与测试数据按 4:1 比例划分
test_data = [] # 测试数据
train_target = [] # 训练数据对应标签
test_target = [] # 测试数据对应标签
```

由于总共三个类别，分别是 0~49,50~99,100~149，因此使用循环，并设置一个偏移量 `offset`，在此基础上对数组进行切片：

```
import numpy as np

for i in range(3):
    offset = 50 * i
    data = iris[offset+0:offset+50, :]

    np.random.shuffle(data) # 就地随机打乱

    train_data.extend(data[0:40, 1:5].tolist())
    train_target.extend(data[0:40, 5].tolist())
    test_data.extend(data[40:, 1:5].tolist())
    test_target.extend(data[40:, 5].tolist())
```

为了方便将数据装入准备好的容器，我们使用 `tolist()` 方法，将切片后的数据转换为列表。

然后为了便于利用 Numpy 的性能（这里影响其实不大）及其各种函数（主要原因），我们再将这些准备好的数据转换为 Numpy 数组：

```
train_data = np.array(train_data)
test_data = np.array(test_data)
train_target = np.array(train_target)
test_target = np.array(test_target)
```

到这里，我们需要的数据就准备好了。

2.2 计算距离

这一步我们来计算测试数据与训练数据的距离。

首先准备好“距离”的容器：

```
distance = []
```

随后遍历测试集中的每个数据；对于这每个测试数据，又都需要遍历训练集中的每个数据，距离度量我们选择欧氏距离：

```
for i in range(len(test_data)):
    sub_dist = []
    for j in range(len(train_data)):
        dif_array = test_data[i] - train_data[j]
        dist = np.sqrt(np.sum(dif_array * dif_array))
        sub_dist.append(dist)

    distance.append(sub_dist)

distance = np.array(distance)
```

其中，`sub_dist`用于存储对于特定的测试数据，对应每个训练数据的“距离”，保存为一维列表。`distance`则是由若干个`sub_dist`组成的二维列表。最后把`distance`也转换为 Numpy 数组。

距离计算这一步也就完成了。

2.3 求解结果

同样的，首先我们初始化结果的容器：

```
results = []
```

这一步就需要用到“K 近邻”中的这个 K 了。为使程序更灵活，我们选择在运行时由用户指定 K 值：

```
K = int(input("请输入 K 值："))
```

然后根据`distance`的大小，可以知道总共有多少个测试数据，使用`np.argsort()`获取升序排序后的前 K 个距离的索引。再用这个索引组成的数组筛选训练数据的标签。最后使用`Counter`类中的`most_common()`方法获取出现频率最高的元素，加入准备好的容器：

```
for i in range(len(distance)):
    index = np.argsort(distance[i])[:K]
    result = train_target[index]

    species = Counter(result).most_common(1)[0][0]

    results.append(species)
```

此时，`results`中即保存了 K 近邻算法全部的结果。

2.4 评估结果

先初始化一个正误计数器：

```
right = 0
```

分别取出测试数据的预期类别和实际类别，并打印；判定预期类别和实际类别是否相符，视相符情况决定是否增加计算器的值：

```
for i in range(len(results)):
    print('Right Species = ', test_target[i], \
        ', \tReal Species = ', results[i])
    if results[i] == test_target[i]:
        right += 1
```

这样，还可以求得 K 近邻算法的正确率，并打印：

```
right_rate = right / len(results)
print("Right Rate: ", right_rate)
```

到这一步，整个算法的实现也就完成了。

接下来我们可以运行看看。

2.5 运行示例

以下是一个运行结果的示例：

```
PS D:\justdopython\KNN> python KNN.py
--------------------------------------------------------------------------------
请输入 K 值：5
--------------------------------------------------------------------------------
Right Species =  Iris-setosa ,  Real Species =  Iris-setosa
Right Species =  Iris-setosa ,  Real Species =  Iris-setosa
Right Species =  Iris-setosa ,  Real Species =  Iris-setosa
Right Species =  Iris-setosa ,  Real Species =  Iris-setosa
Right Species =  Iris-setosa ,  Real Species =  Iris-setosa
Right Species =  Iris-setosa ,  Real Species =  Iris-setosa
Right Species =  Iris-setosa ,  Real Species =  Iris-setosa
Right Species =  Iris-setosa ,  Real Species =  Iris-setosa
Right Species =  Iris-setosa ,  Real Species =  Iris-setosa
```

```
Right Species =  Iris-setosa ,  Real Species =  Iris-setosa
Right Species =  Iris-versicolor ,      Real Species =  Iris-versicolor
Right Species =  Iris-versicolor ,      Real Species =  Iris-versicolor
Right Species =  Iris-versicolor ,      Real Species =  Iris-versicolor
Right Species =  Iris-versicolor ,      Real Species =  Iris-virginica
Right Species =  Iris-versicolor ,      Real Species =  Iris-versicolor
Right Species =  Iris-versicolor ,      Real Species =  Iris-versicolor
Right Species =  Iris-versicolor ,      Real Species =  Iris-versicolor
Right Species =  Iris-versicolor ,      Real Species =  Iris-virginica
Right Species =  Iris-versicolor ,      Real Species =  Iris-versicolor
Right Species =  Iris-versicolor ,      Real Species =  Iris-versicolor
Right Species =  Iris-virginica ,       Real Species =  Iris-virginica
Right Species =  Iris-virginica ,       Real Species =  Iris-virginica
Right Species =  Iris-virginica ,       Real Species =  Iris-virginica
Right Species =  Iris-virginica ,       Real Species =  Iris-virginica
Right Species =  Iris-virginica ,       Real Species =  Iris-virginica
Right Species =  Iris-virginica ,       Real Species =  Iris-virginica
Right Species =  Iris-virginica ,       Real Species =  Iris-virginica
Right Species =  Iris-virginica ,       Real Species =  Iris-virginica
Right Species =  Iris-virginica ,       Real Species =  Iris-virginica
Right Species =  Iris-virginica ,       Real Species =  Iris-virginica
--------------------------------------------------------------------------------
Right Rate:  0.9333333333333333
--------------------------------------------------------------------------------
```

3. 总结

本文带大家初步认识了所谓的 K 近邻算法，并一步步给出了具体的算法实现。如果有同学感觉慢慢堆砌代码有点枯燥，本文在“示例代码”中也给出了上述代码的完整实现，可以前往“示例代码”处直接下载对应代码运行看看结果，这样或许会更有兴趣一点。

激发出兴趣之后，希望读者能够重新实现以下本文描述的代码。

另外，考虑到萼片和花瓣的尺寸范围可能存在一定的差异，其实在算法实现中需要有一个“归一化”的步骤，本文省略了这个步骤。读者可以自行实现再看看结果。

> 示例代码：https://github.com/JustDoPython/python-100-day/tree/master/

参考资料

https://book.douban.com/subject/26708119/

https://book.douban.com/subject/10590856/

https://www.kaggle.com/uciml/iris

https://blog.csdn.net/pipisorry/article/details/51822775

系列文章

第 116 天：机器学习算法之朴素贝叶斯理论
第 115 天：Python 到底是值传递还是引用传递
第 114 天：三木板模型算法项目实战
第 113 天：Python XGBoost 算法项目实战
第 112 天：机器学习算法之蒙特卡洛
第 111 天：Python 垃圾回收机制
从 0 学习 Python 0 - 110 大合集总结

PS： 公号内回复：Python，即可进入Python 新手学习交流群，一起**100天计划！**

-END-

Python 技术

关于 Python 都在这里