你好，从这一课时开始，我们将进入“模块三：分类问题”的学习。在算法部分，我会介绍一个跟算法思想相关的小例子，然后介绍算法的优缺点和适用场景，对于部分算法我将给出算法模块的调用方法，此外一些扩展的内容我会放在最后讲解。在每一个类型的算法最后，我都尽量安排一节小小的实践课，一起来看看数据挖掘是如何做的。

今天我要讲的这个算法是**最近邻算法**（K-NearestNeighbor），简称 KNN 算法。

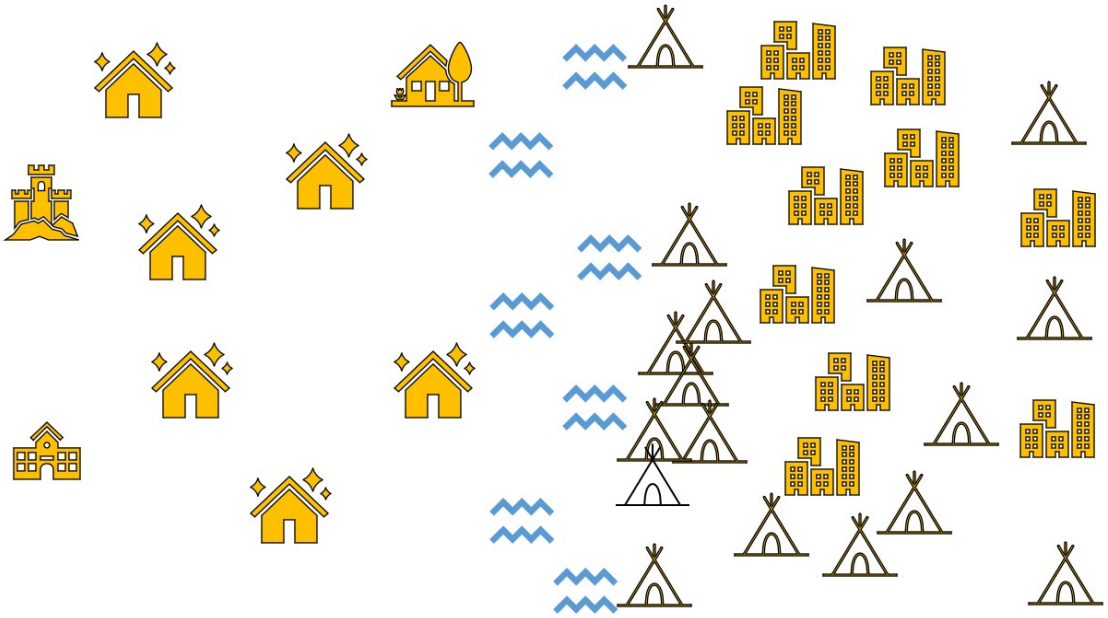
一个例子

有一句老话叫作“物以类聚、人以群分”。想象我们在一个特别的社区里，一条清澈的小河从社区中心流过，小河左侧环境优美，住着一群有钱人，家家户户都是别墅；而小河的另一侧，住着大片贫民，用茅草和纸板搭建的临时住所密密麻麻的。这时有一个新的住户从外面搬进了这个社区，他住在了小河的左侧，此时社区里就传开了消息：“我们这又搬来了一户有钱人家。”可是谁都不认识他，也看不到他的银行账户，为什么就认定他是有钱人呢？那是因为他跟有钱人住在一起了。故事到了这里，也就说明了最近邻算法的思路：“你跟谁住得近，你就跟谁是同一类”。

算法原理

有了思路，我们再来看看原理，KNN 算法是如何处理的。用一句话来解释 KNN 算法原理，那就是找到 K 个与新数据最近的样本，取样本中最多的一个类别作为新数据的类别。在前面的例子中，找到和新搬进来的一户人家住的**距离最近的 K 户人家**，看看 K 户人家中是有钱人多还是穷人多，取多的那个类别作为新搬来这户的类别。所以，显然他住在富人区，那附近就会有更多的富人。

这里面我们提到了一个**距离最近**，关于距离该怎么计算呢？最常见的一个计算方法就是**欧式距离**，即两点之间的连线，如果放在地图上就是两个房子的直线距离。当然除了欧式距离，还有很多距离计算的方式，比如曼哈顿距离、切比雪夫距离等。



算法的优缺点

如此简单的算法都有哪些优缺点呢？下面我结合使用场景进行分析。

优点

简单易实现： 刚把 KNN 算法介绍完了，是不是很简单？从上面的内容可以看出来，KNN 算法最后实际上并没有抽象出任何模型，而是把全部的数据集直接当作模型本身，当一条新数据来了之后跟数据集里面的每一条数据进行对比。

所以可以看到 KNN 算法的一些优点，首当其冲的就是这个算法简单，简单到都不需要进行什么训练了，只要把样本数据整理好了，就结束了，来一条新数据就可以进行预测了。

对于边界不规则的数据效果较好： 可以想到，我们最终的预测是把未知数据作为中心点，然后画一个圈，使得圈里有 K 个数据，所以对于边界不规则的数据，要比线性的分类器效果更好。因为线性分类器可以理解成画一条线来分类，不规则的数据则很难找到一条线将其分成左右两边。

缺点

只适合小数据集： 正是因为这个算法太简单，每次预测新数据都需要使用全部的数据集，所以如果数据集太大，就会消耗非常长的时间，占用非常大的存储空间。

数据不平衡效果不好： 如果数据集中的数据不平衡，有的类别数据特别多，有的类别数据特别少，那么这种方法就会失效了，因为特别多的数据最后在投票的时候会更有竞争优势。

必须要做数据标准化： 由于使用距离来进行计算，如果数据量纲不同，数值较大的字段影响就会变大，所以需要对数据进行标准化，比如都转换到 0-1 的区间。

不适合特征维度太多的数据： 由于我们只能处理小数据集，如果数据的维度太多，那么样本在每个维度上的分布就很少。比如我们只有三个样本，每个样本只有一个维度，这比每个样本有三个维度特征要明显很多。

关于 K 的选取

K 值的选取会影响到模型的效果。在极端情况下，如果 K 取 1，由于富人区人均面积都很大，家里可能是别墅加后花园，富人与富人房子的距离相对较远，那个恰好住在河边的人可能跟河对面的一户贫民家最近，那么这个新人就会被判定为贫民。

如果 K 取值与数据集的大小一样，那么也可想而知，由于贫民的人数户数都远远多于富人，那么所有新进来的人，不管他住哪里都会被判定为贫民。这种情况下，最终结果就是整个样本中占多数的分类的结果，这个模型也就没有什么作用了。

用我们前面学过的内容来看，当 **K 越小的时候容易过拟合**，因为结果的判断与某一个点强相关。而 **K 越大的时候容易欠拟合**，因为要考虑所有样本的情况，那就等于什么都不考虑。

对于 K 的取值，一种显而易见的办法就是从 1 开始不断地尝试，查看准确率。随着 K 的增加，一般情况下准确率会先变大后变小，然后选取效果最好的那个 K 值就好了。当然，关于 K 最好使用奇数，因为偶数在投票的时候就困难了，如果两个类别的投票数量是一样的，那就没办法抉择了，只能随机选一个。

所以选取一个合适的 K 值也是 KNN 算法在实现时候的一个难点，需要根据经验和效果去进行尝试。

尝试动手

接下来，我们尝试借助代码来使用 KNN 算法。今天的动手环节可能要多一点，因为还涉及一些周边的东西，所以我会把前后的代码都写上，包括数据集获取、数据的处理以及训练和预测等环节，在后面一些算法的动手环节就不需要再去重复了。

首先是导入我们所需要的依赖库：

```
from sklearn import datasets  #sklearn的数据集


from sklearn.neighbors import KNeighborsClassifier #sklearn模块的KNN类
```

```
import numpy as np #矩阵运算库numpy

np.random.seed(0)

#设置随机种子，不设置的话默认是按系统时间作为参数，设置后可以保证我们每次产生的随机数是一样的
```

在这里我们使用一个叫作鸢尾花数据集的数据，这个数据集里面有 150 条数据，共有 3 个类别，即 Setosa 鸢尾花、Versicolour 鸢尾花和 Virginica 鸢尾花，每个类别有 50 条数据，每条数据有 4 个维度，分别记录了鸢尾花的花萼长度、花萼宽度、花瓣长度和花瓣宽度。

```
iris=datasets.load_iris()

iris_x=iris.data

iris_y=iris.target

randomarr= np.random.permutation(len(iris_x))

iris_x_train = iris_x[randomarr[:-10]]

iris_y_train = iris_y[randomarr[:-10]]

iris_x_test  = iris_x[randomarr[-10:]]

iris_y_test  = iris_y[randomarr[-10:]]

knn = KNeighborsClassifier()

knn.fit(iris_x_train, iris_y_train)

iris_y_predict = knn.predict(iris_x_test)

probility=knn.predict_proba(iris_x_test)

neighborpoint=knn.kneighbors([iris_x_test[-1]],5)

score=knn.score(iris_x_test,iris_y_test,sample_weight=None)
```

```
print('iris_y_predict = ')

print(iris_y_predict)

#输出原始测试数据集的正确标签，以方便对比

print('iris_y_test = ')

print(iris_y_test)

#输出准确率计算结果

print('Accuracy:',score)
```

下面是输出的结果：

```
iris_y_predict =

[1 2 1 0 0 0 2 1 2 0]

iris_y_test =

[1 1 1 0 0 0 2 1 2 0]

Accuracy: 0.9
```

可以看到，该模型的准确率为 0.9，其中第二个数据预测错误了。

经过上面的一个动手尝试，我们已经成功地实践了 KNN 算法，并使用它对鸢尾花数据进行了分类计算，不知道你是不是有点小激动？当然，关于里面的很多细节这里都没有涉及，希望大家接下来能够更加深入地去探索。

总结

这一小节，我们开始真正走进了一个算法之中，去研究算法的奥秘。当然，我期望以一种简单易学的方式向你介绍算法的原理，并去掉了那些让人头疼的计算公式。在这一节里，我介绍了 KNN 分类算法，从一个例子开始，然后引入了它的原理，并希望你能了解它的优缺点，对于后面的算法，我也会沿用这种方式去介绍。最后，我还写出了一段简单的代码，如果你已经在电脑上安装了 Python，那你可以复制并直接运行它，当然我希望你能够自己去敲一遍代码，这样也能够加深你的印象。