机器学习（L09）：K 最近邻算法

授课教师：王贝伦 / 助教：张嘉琦，黄旭，谈笑，徐浩卿

1 基于实例的学习（Instance-based Learning）

基于实例的学习属于懒惰学习（Lazy Learning）的一种，与急切学习（eager learning）相对，其主要思想是，先把训练样例存储起来，而泛化的工作被推迟到必须分类新的实例时。每当学习器遇到一个新的查询实例，它分析这个新实例与之前存储的实例的关系，并据此把一个目标函数值赋给新实例。

其优点很明显，它无需像其他算法一样在对新实例进行预测之前需要训练数据，而是直接通过存储的数据集进行分类或回归学习得到结果。同时缺点也很明显，它过于依赖已有数据，分类新实例时计算量可能非常大。

基于实例的学习方法包括 K 最近邻算法、加权回归、基于案例的推理，在本节内容中我们主要讨论 K 最近邻算法。

2 K 最近邻算法在分类问题中的应用

K 最近邻算法（K-NN）是一种基本分类与回归方法。既可用于分类问题，也可用于回归问题，在这一部分我们只讨论其在分类问题中的应用。

由于 K 最近邻算法是一种懒惰学习方法，其基于已有数据进行分类，对于分类器，我们首要需确定的五样信息为：

(1) 之前储存的训练数据集：

$$T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\} \tag{1}$$

其中，$x_i \in \mathcal{X} \subseteq \boldsymbol{R}^p$ 为实例的特征向量，$y_i \in \mathcal{Y} = \{c_1, c_2, ..., c_L\}$ 为实例的类别，$i = 1, 2, ..., N$；

(2) 采用的距离度量 $D(\cdot)$；

(3) 涵盖邻近点的个数 K；

(4) 待分类的新实例点特征向量 x；

(5) 邻近算法的分类决策机制；

2.1 K 最近邻算法的过程及其复杂度分析

而算法的主要过程可以分成三个步骤：

(1) 计算实例点到所有训练实例点的距离：

$$\mathcal{D} = \{D(x, x_1), D(x, x_2), ..., D(x, x_N)\} \tag{2}$$

在这个过程中，实例点的维度为 p，计算一次 $D(x, x_i)$ 的复杂度是 $O(p)$，共计要计算 N 次 $D(x, x_i)$，所以这一步的复杂度为 $O(Np)$。

(2) 选取 K 个离实例点最近的邻近点，也就是 $\mathcal{D}$ 中 K 个最小的值对应的 T 中的点，构成集合 $N_K(x)$：

$$N_K(x) = \{x_{n_1}, x_{n_2}, ..., x_{n_K}\} \tag{3}$$

这一步中要想选取 K 个最小的值，必须先对 $\mathcal{D}$ 进行从小到大的排序，然后选择前 K 个，此时的时间复杂度主要取决于排序方法的选择，一般来说，现在选用的排序方法的时间复杂度通常为 $O(n \log n)$。

(3) 根据集合 $N_K(x)$ 中点对应的标签，根据输入提供的分类决策机制进行判断投票，决定未知实例点的类别；

最后一步的时间复杂度决定于决策机制的选用，如果是多数表决法的话，其时间复杂度不高，当然这还区别于 K 个邻近点的类别数量。

分类决策机制中最简单最常用的方法就是**多数表决法（Majority voting rule）**，当然这种方法存在着很多缺陷，改进的方法将在该讲义的后续部分提到。

如图 1，在这种情况下，在邻近点全部为 $+$ 的情况下，该点也会被判定为 $+$；

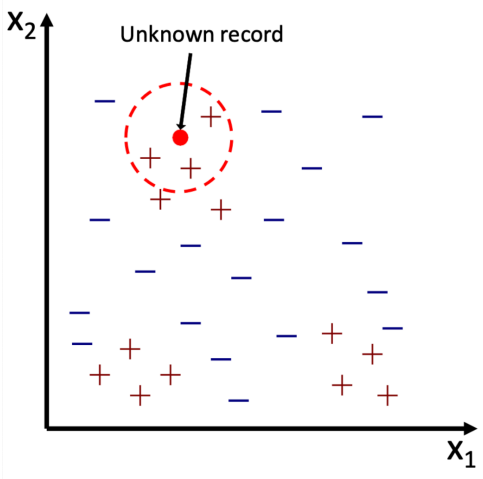

Figure 1: K 最近邻算法

2.2 $K = 1$ 的最近邻算法

当 $K = 1$ 时，此时的模型最为简单，最好理解，这里我们介绍一个新的概念，叫做**泰森多边形（Voronoi diagram）**，又可称为“冯洛诺伊图”，是一组由连接两邻点线段的垂直平分线组成的连续多边形组成。一个泰森多边形内的任一点到构成该多边形的控制点的距离小于到其他多边形控制点的距离，如图 2 所示。如图所示，待分类的实例点会落在不同色块中，其类型由该色块内的原实例点决定。泰森多边形的出现就是为了帮我们更好地理解 $K = 1$ 时的模型。

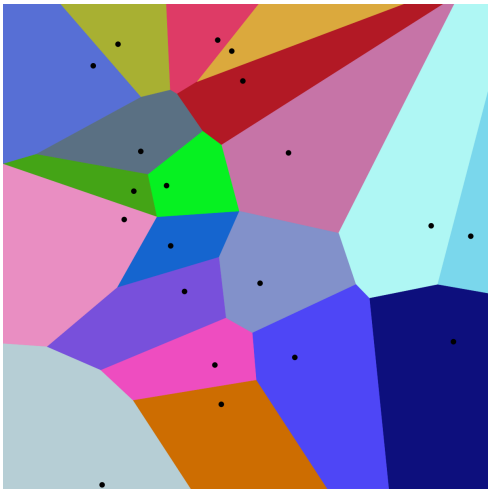

Figure 2: 泰森多边形

模型虽然简单，但是如果点恰好落在边界线上怎么办？当然不仅是 $K=1$ 会存在这个问题，如果我们使用多数表决法，不管 K 的值是多少，在满足一定条件下，都可能存在着这种多数表决法失灵的尴尬场面，所以我们一般会选择奇数尽量减少这种情况的发生。

Algorithm 1: K 邻近算法

1 **输入：训练数据集 T，待测实例点的特征向量 x，涵盖邻近点个数 K，采用的距离度量函数 $D(\cdot)$，分类决策机制函数;**
2 **for** $i \leftarrow 1$ *to* N **do**
3 　**计算距离** $D(x, x_i)$;
4 **end**
5 **对所有距离进行排序，选择最小的 K 个;**
6 **根据决策机制和 K 个邻近点的信息，确定最后待测实例点的标签;**
7 **输出：实例 x 所属的类 y。**

3 距离度量（Distance Metric）

特征空间中两个实例点的距离是两个实例点相似程度的反应。在 K 最近邻算法中，我们采用标准欧式距离（Euclidean Distance），这也是人们最常用的距离度量标准，令 $x=(x_1,x_2,...,x_n), x'=(x'_1,x'_2,...,x'_n)$，有

$$D(x,x')=\sqrt{\sum_{i=1}^{n}(x_i-x'_i)^2}=\sqrt{(x-x')^T(x-x')} \tag{4}$$

当然我们也可以使用其他距离，不仅限于欧式距离，比如

- 曼哈顿距离（Manhattan Distance）

$$L_1(x,x')=\sum_{i=1}^{n}|x_i-x'_i| \tag{5}$$

- L_∞ norm distance

$$L_\infty(x,x')=\max_i|x_i-x'_i| \tag{6}$$

我们在实际处理计算实例点之间欧式距离的时候，如果某一个特征数值过大，会导致两个实例点之间的相似性几乎只由这个特征决定，所以为了避免这种情况，我们需要对其先进行特征缩放。

4 参数选择

在 K 近邻算法中，我们重点需要设置的参数就是 K。K 值的大小反映了在 K 近邻算法中我们检索相似数据的范围。图3展示了不同 K 值对于算法产生的影响。因为 K 的值并不能直接从数据中获得，我们需要通过使用不同 K 值来训练 K 最近邻算法，并且根据分类准确度来选择出最好的 K 值。

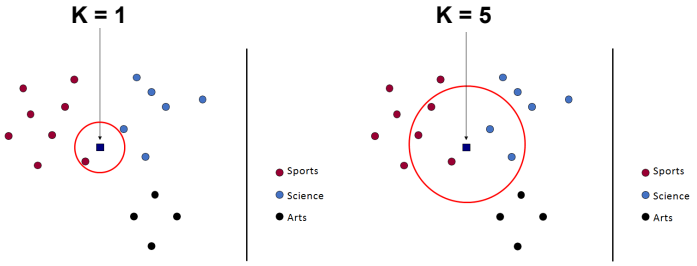

Figure 3: 不同 K 值对 K 近邻算法的影响。

本质上，K 的选择会影响模型的泛化性。从图4我们对比了线性回归和 K 近邻算法的预测结果。大部分的分类算法会针对数据预测出一条“边界”，这个边界一般被称为决策边界（decision boundary）[1]。分布在决策边界两边的数据会被划分到不同的类别中。我们先考虑二分类问题，决策边界就是一条曲线。特别地，对于线性回归，决策边界是一条直线。K 最近邻算法预测得到的决策边界取决于 K 指的选择。可以看到，当 $K=1$ 时，我们获得了近乎完美分类结果，但是决策边界却非常扭曲。这样的模型显然不具有泛化。而当我们设置 $K=15$ 时，虽然有很多数据被误分类了，但是决策边界的泛化性会更好。

所以我们会发现，当 K 很小时，模型会对数据很敏感。然而因为实际收集的数据中存在各种各样的噪声，模型会“学习”到噪声的信息，从而影响分类决策。而当 K 较大时，最近的数据可能会包含一些其他类别的数据，从而影响分类结果。因此，在实际使用 K 近邻算法时，为了在分类准确度和模型的泛化性能之间获得平衡，我们需要很谨慎地选择参数 K 的值。

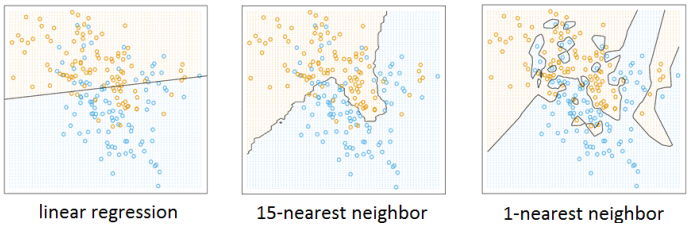

Figure 4: 线性回归以及 K 近邻算法使用不同 K 值的预测结果。

5 加权 K 最近邻算法

最后我们来考虑一下，当我们找到 K 个最近的数据点之后，怎么根据这些数据计算获得新数据的类别标签。

一个最简单也最常用的方法就是**多数表决法**。这个方法与投票过程类似，在 K 个最近邻中找出样本数量最多的那个类别作为新数据的类别。比如，假设我们找到了一个数据点的 3 个最近邻，其中两个属于类别 A 而另一个数据类别 B，那么我们就可以认为这个待分类的数据点属于类别 A。使用这种方法时，K 值需要有一

[1] 这里的例外是生成式模型（generative model）。我们在之前的讲义中简单解释过，对于生成式分类模型，我们并不会获得明确的决策边界，而是会的到每个类别的概率分布。

些限定。比如对于二分类问题，K 应该是一个奇数，这样能避免不同类别样本数相同的情况。

另一个更加合理的方法就是**权重法**。这种方法会根据近邻点和待分类点之间的距离来为近邻点分配一个权重 w，然后计算每个类别的权重来进行分类。这种分类方法称作加权 K 最近邻算法。比如在图5中，我们找到了三个最近邻点，绿色表示待分类的数据点，红色表示类别 A，而蓝色表示类别 B。如果我们使用多数表决法，那么数据应该被分类为 B（蓝色）。然而，显然类别 A（红色的点）距离待分类点更近。在引入权重之后，假设我们为红色点分类权重 $w_{红}=1$，而为两个蓝色点分别分配权重 $w_{蓝\,1}=1/2$ 与 $w_{蓝\,2}=1/3$。那么待分类点被分类到 A 的分数为 1，而分类为 B 的分数为 $1/2+1/3=5/6$。显然，类别 A 的的分数更高，那么新数据应该被分到 A 类中。

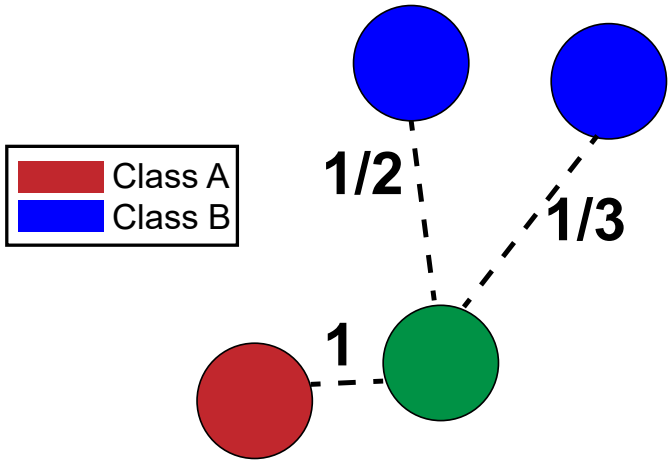

Figure 5: 加权 K 最近邻算法示例。

多数表决法其实是权重法的一个特例，它将每个数据样本的权重设置为 1，并不根据数据间距离进行区分。

5.1 权重计算方法

在加权 K 最近邻算法之中，最重要的就是权重计算的方法。不同的计算方法会影响到分类结果。其中一种常用的方法是利用两个数据点之间的距离来计算权重

$$w_j=\frac{1}{d(x_i,x_j)} \tag{7}$$

其中 x_i 是待分类数据点，并且 $x_j\in \mathrm{NN}(x_i)$ 是最近邻点。这样距离越远的数据，权重越小。一般距离计算的函数选择为 ℓ_1 或者 ℓ_2 范数，即

$$\begin{aligned} d_1(x_i,x_j)&=||x_i-x_j||_1\\ d_2(x_i,x_j)&=||x_i-x_j||_2 \end{aligned} \tag{8}$$

除了范数的倒数之外，还通常使用高斯函数来计算权重

$$w_j=\exp(-\lambda||x_i-x_j||_2^2) \tag{9}$$

高斯函数的曲线如图6所示，靠近 0 的地方函数值趋近于 1，而在远离 0 的地方，函数值无限接近于 0.

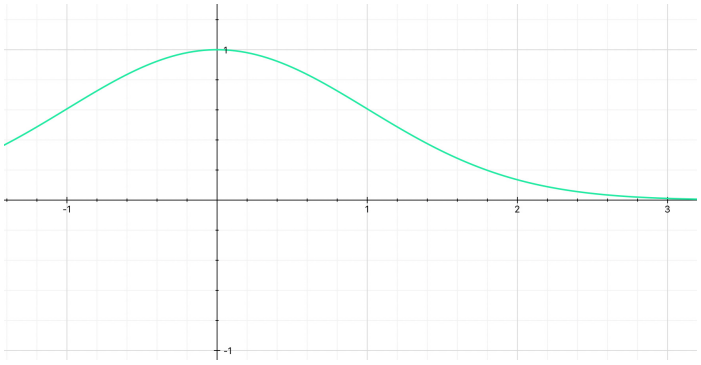

Figure 6: 高斯函数。x 轴表示 $||x_i-x_j||_2^2$。

6 K 最近邻算法复杂度分析

正如我们之前所说的，K 最近邻算法并不是真正地在学习，因此也就不存在训练过程。在预测时，K 最近算法的时间开销主要由两部分组成：计算数据点之间距离以及排序。令 p 表示变量个数，用 n 表示样本数量。在表格1中我们比较了线性回归（使用正规方程）与 K 最近邻算法之间的时间复杂度对比。其中对于 K 最近邻算法，我们把排序算法的时间复杂度看作 $O(n\log n)$。

Table 1: 线性回归与 K 最近邻算法时间复杂度对比。

	训练过程	预测过程
线性回归	$O(np^2+p^3)$	$O(p)$
K 最近邻算法	$O(1)$	$O(np+n\log n)$

7 K 近邻算法在回归问题中的应用

K 近邻算法并不是仅为了解决分类问题而提出的，它还可以被用来解决回归问题。与解决分类问题类似，K 最近邻算法在回归问题中唯一需要改变的就是怎么通过近邻点的 y 值来计算待预测点的 y。类比于分类问题，我们可以根据样本权重来计算

$$\widehat{y}_i=\sum_{x_j\in \mathrm{NN}(x_i)} w_j y_j \tag{10}$$

8 K 最近邻算法总结

K 最近邻算法是一种比较简单的**非参数模型**。我们之前介绍的都是参数模型，比如线性回归模型，我们将 x 与 y 之间关系建模为参数 θ。而非参数模型并不需要参数 θ 的存在。在之后的课程中，我们会详细介绍非参数模型背后的思想以及几种常用模型。

K 最近邻算法具有简单，容易实现的好处。然而算法需要大量空间来存储数据来作预测。除此，K 近邻算法的预测时间成本很高，对于大规模的数据，仅仅是找到近邻点就会花费大量时间。然而 K 最近邻算法仍然适用于很多应用中，并且实际性能并不差。

9 编程实现

这里我们简单实现一个 K 近邻分类算法。

```python
import numpy as np
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt

# 生成二分类的数据
N = 100
p = 2
x, y = make_classification(N, p, n_informative = p, n_redundant = 0, n_classes = 2)
# 将数据划分为训练集和测试集；对于K近邻算法，训练集并不会用来训练，只表示我们已经存储的带有类别标签的数据
train_x = x[:60,:]
train_y = y[:60]
testing_x = x[60:, :]
testing_y = y[60:]

# 定义K最近邻算法
def KNN(train_x,train_y, testing_x, K = 5):
    predicted_y = []
    for sample in testing_x:
        # 使用L1范数计算数据点距离
        distance = np.linalg.norm(train_x - sample, axis = 1, ord=1)
        # 找到距离最小的K个数据点
        sort_index = np.argsort(distance)
        neighbors = sort_index[:K]
        neighbors_y = train_y[neighbors]
        # 利用多数投票法对数据进行分类
        class1_vote = np.sum(neighbors_y == 0)
        class2_vote = np.sum(neighbors_y == 1)
        predicted_y.append(0) if class1_vote > class2_vote else predicted_y.append(1)
    return predicted_y


predicted_y = KNN(train_x, train_y, testing_x, K = 2)
# 计算分类准确度
correct_count = 0
for index in range(len(testing_y)):
    if testing_y[index] == predicted_y[index]:
        correct_count += 1
print('分类正确率为：', correct_count / len(testing_y))

# Output：
# 分类正确率为： 0.95
```

引用

[1] K-Nerest Neighbor Algorithm: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[2] Inverse Distance Weighting: https://en.wikipedia.org/wiki/Inverse_distance_weighting#Shepard's_method

[3] Hang Li "Statistical Learning"