

[学习笔记] [机器学习] 3. KNN (K - 近邻算法、距离度量：曼哈顿距离、切比雪夫距离、K值选择、kd树、鸢尾花种类预测、特征工程、交叉验证、网格搜索、...)

1. [视频链接](#)

2. 数据集下载地址：[《3. KNN 及练习案例》配套数据集](#)

1. K - 近邻算法 (KNN) 概念

学习目标：

- 掌握 K - 近邻算法实现过程
- 知道 K - 近邻算法的距离公式
- 知道 K - 近邻算法的超参数 K 值以及取值问题
- 知道 kd 树实现搜索的过程
- 应用 KNeighborsClassifier 实现分类
- 知道 K - 近邻算法的优缺点
- 知道交叉验证实现过程
- 知道超参数搜索过程
- 应用 GridSearchCV 实现算法参数的调优

K Nearest Neighbor 算法又叫 KNN 算法，这个算法是机器学习里面一个比较经典的算法，总体来说 KNN 算法是相对比较容易理解的算法。

KNN 算法是一种基于实例的机器学习算法，用于分类和回归问题。在分类问题中，KNN 算法使用训练数据集中最接近目标样本的 K 个最近邻居的标签来预测目标样本的标签。在回归问题中，KNN 算法使用训练数据集中最接近目标样本的 K 个最近邻居的数值来预测目标样本的数值。

KNN 算法的核心思想：基于距离度量来判断样本之间的相似性。常用的距离度量包括欧几里得距离、曼哈顿距离等。

- KNN 算法的优点：简单易懂、易于实现，并且适用于各种类型的数据。
- KNN 算法的缺点：需要保存所有的训练数据集，计算复杂度高，对异常值敏感，以及对于高维数据的处理比较困难。

1.1 KNN 的定义

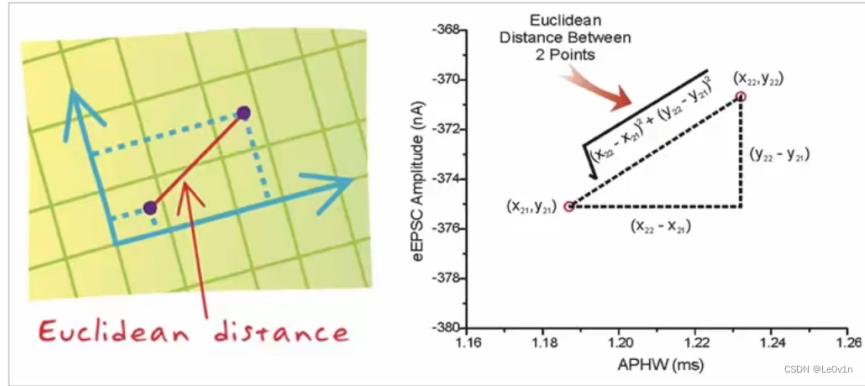
如果一个样本在特征空间中的 k 个最相似 (即特征空间中最邻近) 的样本中的大多数属于某一个类别，则该样本也属于这个类别。

不是说最近的是哪个类别就是哪个类别了，而是最近的 k 个哪个类别多就属于哪个类别。

来源：KNN 算法最早是由 Cover 和 Hart 提出的一种分类算法

1.2 距离公式

两个样本的距离可以通过如下公式计算，又叫欧式距离、欧几里得距离或欧几里得度量，欧式距离是欧几里得空间中两点间“普通”（即直线）距离。使用这个距离，欧式空间成为度量空间。相关联的范数称为欧几里得范数。较早的文献称之为毕达哥拉斯度量。



对于二维平面上的两个点 (x_1, y_1) 和 (x_2, y_2) , 它们之间的欧式距离为:

$$d_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

对于三维空间中的两个点 (x_1, y_1, z_1) 和 (x_2, y_2, z_2) , 它们之间的欧式距离为:

$$d_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

对于 n 维空间中的两个点 $p = (p_1, p_2, \dots, p_n)$ 和 $q = (q_1, q_2, \dots, q_n)$, 它们之间的欧式距离为:

$$d_{12} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

求的是两个点之间的距离, 不能是多个点之间的距离。

1.3 例子：电影类型分析

假设我们现在有几部电影:

序号	电影名称	搞笑镜头	拥抱镜头	打斗镜头	电影类型
1	功夫熊猫	39	0	31	喜剧片
2	叶问3	3	2	65	动作片
3	二次曝光	2	3	55	爱情片
4	代理情人	9	38	2	爱情片
5	新步步惊心	8	34	17	爱情片
6	谍影重重	5	2	57	动作片
7	美人鱼	21	17	5	喜剧片
8	宝贝当家	45	2	9	喜剧片
9	唐人街探案	23	3	17	?

其中 ? 表示电影不知道类别, 如何去预测?

我们可以利用 K 近邻 (KNN) 算法的思想。

序号	电影名称	搞笑镜头	拥抱镜头	打斗镜头	电影类型
1	功夫熊猫	39	0	31	喜剧片
2	叶问3	3	2	65	动作片
3	二次曝光	2	3	55	爱情片
4	代理情人	9	38	2	爱情片
5	新步步惊心	8	34	17	爱情片
6	谍影重重	5	2	57	动作片
7	美人鱼	21	17	5	喜剧片
8	宝贝当家	45	2	9	喜剧片
9	唐人街探案	23	3	17	?

$d = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$
 $d = \sqrt{(23-2)^2 + (3-3)^2 + (17-55)^2}$
 $= 43.42$

分别计算每个电影和被预测电影的距离, 然后求解。

序号	电影名称	搞笑镜头	拥抱镜头	打斗镜头	电影类型	距离	K=5时
1	功夫熊猫	39	0	31	喜剧片	21.47	✓
2	叶问3	3	2	65	动作片	52.01	
3	二次曝光	2	3	55	爱情片	43.42	
4	代理情人	9	38	2	爱情片	40.57	✓
5	新步步惊心	8	34	17	爱情片	34.44	✓
6	谍影重重	5	2	57	动作片	43.87	
7	美人鱼	21	17	5	喜剧片	18.55	✓
8	宝贝当家	45	2	9	喜剧片	23.43	✓
9	唐人街探案	23	3	17	?	—	?

根据上面的定义，我们知道《唐人街探案》与《美人鱼》的距离最近。 $K = 5$ 表明是否为最近的 5 部电影。最近的 5 部电影中，“喜剧片”有 3 个，“爱情片”有 2 个，因此《唐人街探案》的电影类型应该是“喜剧片”。

1.4 KNN 算法流程总结

1. 计算已知类别数据集中的点与当前点之间的距离
2. 按距离升序排序
3. 选取与当前点距离最小的 k 个点
4. 统计前 k 个点所在类别出现的频率
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

小结：

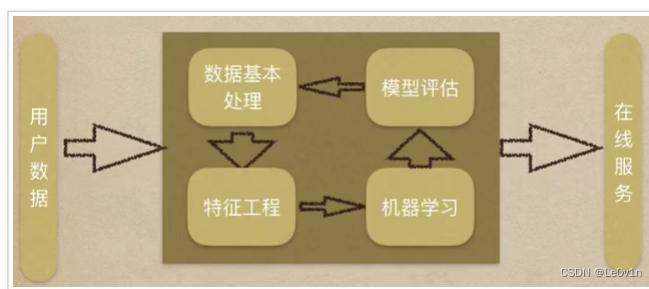
1. K - 近邻算法简介【了解】
2. 定义：就是通过你的“邻居”来判断你属于哪个类别
3. 如何计算你到你的“邻居”的距离：一般时候，都是使用欧氏距离

2. KNN 算法 API 的初步使用

学习目标：

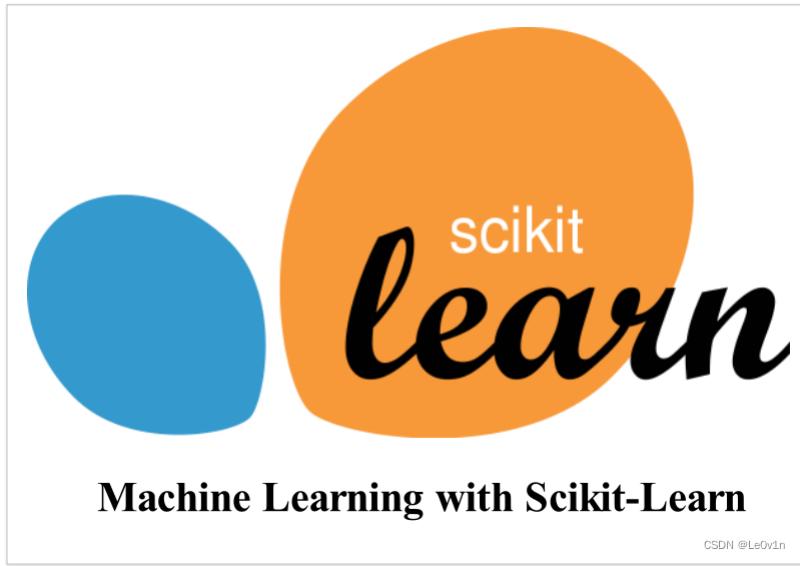
- 了解 sklearn 工具的优点和包含内容
- 应用 sklearn 中的 API 实现 KNN 算法的简单使用

机器学习流程：



1. 获取数据集
2. 数据基本处理
3. 特征工程
4. 机器学习
5. 模型评估

2.1 Scikit-learn 工具介绍



sklearn 是一个开源的 Python 机器学习库，提供了许多用于建立和分析机器学习模型的工具和算法。主要功能包括：

1. 数据预处理：提供了数据标准化、特征缩放、特征选择等预处理工具，能够帮助用户更好地处理数据。
2. 监督学习：提供了许多经典的监督学习算法，如线性回归、逻辑回归、决策树、支持向量机、随机森林等，并提供了交叉验证（Cross Validation）、网格搜索（Grid Search）等模型调参工具。
3. 无监督学习：提供了无监督学习算法，如聚类、降维、异常检测等，其中最常用的算法可能是 K-Means 聚类和主成分分析（PCA）等。
4. 模型评估：提供了常见的模型评估指标，如混淆矩阵、精度、召回率、F1 值等，并且可以使用交叉验证等方法来评估模型的性能。
5. 数据可视化：提供了一些数据可视化工具，如 t-SNE、PCA 等，以帮助用户更好地理解数据。

总之，sklearn 是一个强大的机器学习库，可用于解决各种机器学习问题，并且易于使用，适合各种程度的用户。

- Python 语言的机器学习工具
- Scikit-learn 包括许多知名的机器学习算法的实现
- Scikit-learn 文档完善，容易上手，丰富的 API

sklearn 的全称是 scikit-learn。scikit 表示 SciPy Toolkit，因为它依赖于 SciPy 库（Science Python）。而 learn 则表示机器学习。

安装：

```
pip install scikit-learn
```

注：安装 scikit-learn 需要 Numpy、Scipy 等库

2.2 KNN 算法 API

```
sklearn.neighbors.KNeighborsClassifier(n_neighbors=5)
```

其中 `n_neighbors`：查询使用的邻居数，`int`，默认为 5。

2.3 练习案例

2.3.1 步骤分析

1. 获取数据集
2. 数据基本处理（该案例中省略）
3. 特征工程（该案例中省略）
4. 机器学习
5. 模型评估（该案例中省略）

2.3.2 代码过程

1. 导入模块

```
from sklearn.neighbors import KNeighborsClassifier
```

2. 构造数据集

```
x = [[0], [1], [2], [3]]  
y = [0, 0, 1, 1]
```

3. 机器学习：训练模型

```
# 实例化API  
estimator = KNeighborsClassifier(n_neighbors=2)  
  
# 使用fit方法进行训练  
estimator.fit(x, y)  
  
# 预测  
estimator.predict([[1]]) # array([0])
```

小结：

- sklearn 的优势：
 - 文档多，且规范
 - 包含的算法多
 - 实现起来容易

- KNN 中的 API：
 - `sklearn.neighbors.KNeighborsClassifier(n_neighbors=5)`

问题：

1. 距离公式，除了欧式距离，还有哪些距离公式可以使用？
2. 如何选取 K 值的大小？
3. API 中其他参数的具体含义？

3. 距离度量

学习目标：

1. 了解距离公式的基本性质
2. 知道机器学习中常见的距离计算公式

3.1 距离公式的基本性质

在机器学习过程中，对于函数 $\text{dist}(*, *)$ ，若它是“距离度量”(Distance Measure)，则需满足一些基本性质：

1. 非负性： $\text{dist}(x_i, x_j) \geq 0$ —— 两点之间的距离必须 ≥ 0
2. 同一性： $\text{dist}(x_i, x_j) = 0$ (当且仅当 $x_i = x_j$) —— 同一个位置距离为 0
3. 对称性： $\text{dist}(x_i, x_j) = \text{dist}(x_j, x_i)$
4. 直递性： $\text{dist}(x_i, x_j) \leq \text{dist}(x_i, x_k) + \text{dist}(x_k, x_j)$ —— 三角不等式

直递性常被直接称为“三角不等式”

3.2 常见的距离公式

3.2.1 欧式距离 (Euclidean Distance)

欧式距离是最容易直观理解的距离度量方法，我们小学、初中和高中接触到的两个点在空间中的距离一般都是指欧式距离。

对于二维平面上的两个点 (x_1, y_1) 和 (x_2, y_2) ，它们之间的欧式距离为：

$$d_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

对于三维空间中的两个点 (x_1, y_1, z_1) 和 (x_2, y_2, z_2) ，它们之间的欧式距离为：

$$d_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

对于 n 维空间中的两个点 $p = (p_1, p_2, \dots, p_n)$ 和 $q = (q_1, q_2, \dots, q_n)$ ，它们之间的欧式距离为：

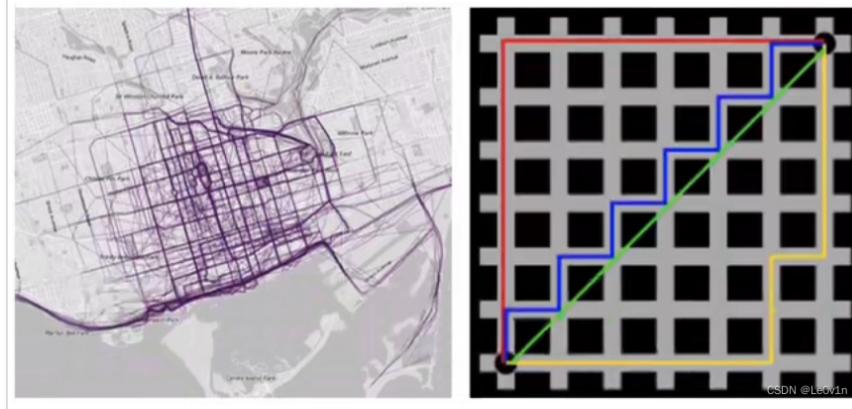
$$d_{12} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

举例：

```
x = [[1, 1], [2, 2], [3, 3], [4, 4]];
经计算得:
d = 1.4142 2.8284 4.4246 1.4142 2.8284 1.4142
```

3.2.2 曼哈顿距离 (Manhattan Distance)

在曼哈顿街区要从一个十字路口开车到另一个十字路口，驾驶距离显然不是两点间的直线距离。这个实际驾驶距离就是“曼哈顿距离”。曼哈顿距离也称为“城市街区距离”(City Block Distance)。



曼哈顿距离是指在一个二维平面上，从点 A 到点 B 沿着网格线走的最短距离。也就是说，曼哈顿距离是指两个点横坐标差的绝对值与纵坐标差的绝对值之和。例如，如果点 A 的坐标为 (x_1, y_1) ，点 B 的坐标为 (x_2, y_2) ，则它们之间的曼哈顿距离为 $|x_1 - x_2| + |y_1 - y_2|$ 。曼哈顿距离得名于美国纽约曼哈顿区的城市规划，因为这个地区的街道呈网格状布局，人们往往需要沿着网格线行走。曼哈顿距离广泛应用于计算机科学、数据分析以及城市规划等领域。

- 二维平面两点 $a(x_1, y_1)$ 与 $b(x_2, y_2)$ 间的曼哈顿距离：

$$d_{12} = |x_1 - x_2| + |y_1 - y_2|$$

- n 维空间点 $a(p_1, p_2, p_3, \dots, p_n)$ 与 $b(q_1, q_2, q_3, \dots, q_n)$ 的曼哈顿距离：

$$d_{12} = \sum_{k=1}^n |p_k - q_k|$$

举例：

```
x = [[1, 1], [2, 2], [3, 3], [4, 4]];
经计算得:
d = 2 4 6 2 4 2
```

3.2.3 切比雪夫距离 (Chebyshev Distance)

切比雪夫距离是指在数学中，两个 n 维向量之间的度量方式。它定义为两个向量各个维度差值的绝对值的最大值。

具体而言，对于两个 n 维向量 $x = (x_1, x_2, \dots, x_n)$ 和 $y = (y_1, y_2, \dots, y_n)$ ，它们之间的切比雪夫距离 $d(x, y)$ 是：

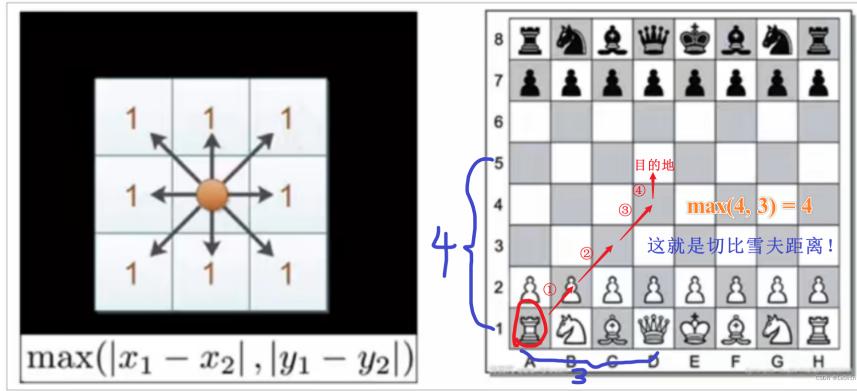
$$d(x, y) = \max(|x_1 - y_1|, |x_2 - y_2|, \dots, |x_n - y_n|)$$

也就是说，切比雪夫距离是各个维度差值的绝对值的最大值。例如，对于二维平面上的点 $x = (x_1, x_2)$ 和 $y = (y_1, y_2)$ ，它们之间的切比雪夫距离是：

$$d(x, y) = \max(|x_1 - y_1|, |x_2 - y_2|)$$

切比雪夫距离常用于衡量两个向量之间的相似性或差异性，特别是在聚类、异常检测和分类等机器学习领域中经常被使用。

国际象棋中，国王可以直行、横行、斜行，所以国王走一步可以移动到相邻 8 个方格中的任意一个。国王从格子 (x_1, y_1) 走到格子 (x_2, y_2) 最少需要多少步？这个距离就叫切比雪夫距离。



3.2.4 闵可夫斯基距离 (Minkowski Distance)

闵氏距离不是一种距离，而是一组距离的定义，是对多个距离度量公式的概括性的表述。闵可夫斯基距离是一种用于度量两个 n 维向量之间相似性或差异性的距离度量方式，它可以看做是将欧几里得距离和切比雪夫距离作为特例的一般化形式。

其定义为：对于两个 n 维向量 $x = (x_1, x_2, \dots, x_n)$ 和 $y = (y_1, y_2, \dots, y_n)$ ，它们之间的闵可夫斯基距离 $d(x, y)$ 为：

$$d(x, y) = \sqrt[p]{\sum_{i=1}^n |x_i - y_i|^p}$$

其中， p 为正实数，表示闵可夫斯基距离的阶数。

- 当 $p = 1$ 时，就是曼哈顿距离；
- 当 $p = 2$ 时，就是欧氏距离；
- 当 $p \rightarrow \infty$ 时，就是切比雪夫距离。

根据 p 的不同，闵氏距离可以表示某一类 / 种的距离。在机器学习领域，闵可夫斯基距离同样被广泛应用于聚类、异常检测和分类等任务中。

小结：

闵氏距离，包括曼哈顿距离、欧氏距离和切比雪夫距离，都存在明显的缺点。

- 欧氏距离的主要缺点是对异常值敏感，即如果某个特征明显偏离其他特征的分布，那么该特征对欧氏距离的计算结果会产生较大的影响。此外，在高维数据中，欧氏距离会出现所谓的“维数灾难”，即随着维度的增加，样本之间的距离变得越来越相似，这会导致聚类效果不佳。
- 曼哈顿距离的缺点是在处理高维数据时，由于数据点之间的距离计算只考虑每个维度上的差异，因此随着维度的增加，其计算量会快速增加，而且容易受到噪声数据的影响，使得聚类效果变差。
- 切比雪夫距离的主要缺点是它只考虑维度之间的最大差异，而忽略了各个维度之间的协同作用。因此，当数据点在多个维度上都存在轻微差异时，切比雪夫距离可能会产生不合理的结果。此外，切比雪夫距离对于对称分布的数据集可能会失效。

因此，在实际应用中，需要根据具体的问题和数据特征选择适当的距离度量方式，或者采用多种距离度量方式综合考虑。

举例：二维样本（身高 [单位：cm]，体重 [单位：kg]），现有三个样本： $a(80, 50)$ ， $b(190, 50)$ ， $c(180, 60)$ 。

a 与 b 的闵氏距离（无论是曼哈顿距离、欧氏距离或切比雪夫距离）等于 a 与 c 的闵氏距离。但实际上身高的 10cm 并不能和体重的 10kg 划等号。

闵氏距离的缺点：

1. 将各个分量的量纲 (scale)，也就是将不同分量的“单位”相同地看待了；
2. 未考虑各个分量的分布（期望，方差等）可能是不同的。

3.3 【拓展】其他距离公式

3.3.1 标准化欧氏距离 (Standardized Euclidean Distance)

标准化欧氏距离是一种基于欧氏距离的距离度量方式，用于在处理数据时将不同维度上的特征值进行标准化处理。其计算方式是对每个特征值减去该特征值的均值并除以该特征值的标准差，从而将特征值转换为标准正态分布，然后再计算欧氏距离。

具体而言，对于两个 n 维向量 $x = (x_1, x_2, \dots, x_n)$ 和 $y = (y_1, y_2, \dots, y_n)$ ，它们之间的标准化欧氏距离 $d(x, y)$ 可以表示为：

$$d(x, y) = \sqrt{\sum_{i=1}^n \left(\frac{x_i - \bar{x}_i}{s_i} - \frac{y_i - \bar{y}_i}{s_i} \right)^2}$$

其中, \bar{x}_i 和 \bar{y}_i 分别表示向量 x 和 y 在第 i 个维度上的均值, s_i 表示向量在第 i 个维度上的标准差。

标准化欧氏距离的优点是可以消除由于不同特征量纲和取值范围带来的影响, 使得各个特征在距离度量中起到相同的作用。此外, 标准化欧氏距离还可以减小异常值的影响。

在机器学习和数据挖掘领域, 标准化欧氏距离常常被用于聚类、分类和异常检测等任务中, 特别是在需要处理多个具有不同量纲和取值范围的特征时, 标准化欧氏距离能够提高模型的鲁棒性和泛化能力。

举例:

```
X=[[1, 1], [2, 2], [3, 3], [4, 4]]; (假设两个分量的标准差分别为0.5和1)
```

经计算得:

```
d = 2.2361 4.4721 6.7082 2.2361 4.4721 2.2361
```

3.3.2 余弦距离 (Cosine Distance)

几何中, 夹角余弦可用来衡量两个向量方向的差异; 机器学习中, 借用这一概念来衡量样本向量之间的差异。余弦距离是一种用于度量两个向量之间相似性的距离度量方式。其定义为两个向量之间的夹角的余弦值, 即:

$$d(x, y) = 1 - \frac{x \cdot y}{\|x\| \|y\|}$$

其中, x 和 y 分别表示两个向量, \cdot 表示向量的内积运算, $\|x\|$ 和 $\|y\|$ 分别表示两个向量的模长。余弦距离的取值范围在 0 到 1 之间:

- 当两个向量方向相同时, 余弦距离等于 1;
- 当两个向量方向完全相反时, 余弦距离等于 0;
- 当两个向量之间不存在线性关系时, 余弦距离接近于 0。

在机器学习领域, 余弦距离常被用于文本分类、推荐系统和图像处理等任务中, 特别是在需要考虑向量之间的方向而不仅仅是大小时, 余弦距离能够提供更好的距离度量方式。

举例:

```
X=[[1, 1], [1, 2], [2, 5], [1, -4]]
```

经计算得:

```
d = 0.9487 8.9191 -0.5145 0.9965 -0.7593 -0.8107
```

3.3.3 汉明距离 (Hamming Distance)

汉明距离是衡量两个等长字符串在相同位置上不同字符的个数, 即通过对比这两个字符串对应位置上的字符是否相同来计算它们之间的“距离”。

设有两个长度为 n 的字符串 A 和 B , 则它们的汉明距离 $HD(A, B)$ 定义为将 A 转换成 B 所需的最小替换次数。也就是说, 需要将 A 中的某些字符替换成另外的字符才能得到 B , 而替换的次数就是它们的汉明距离。

计算公式如下:

$$D_H(A, B) = \sum_{i=1}^n (A[i] \neq B[i])$$

其中, $A[i]$ 表示 A 中第 i 个字符, $B[i]$ 表示 B 中第 i 个字符。符号 \neq 代表不等于。该公式表示将 A 和 B 每个位置上的字符逐一比较, 若不同则累加 1。

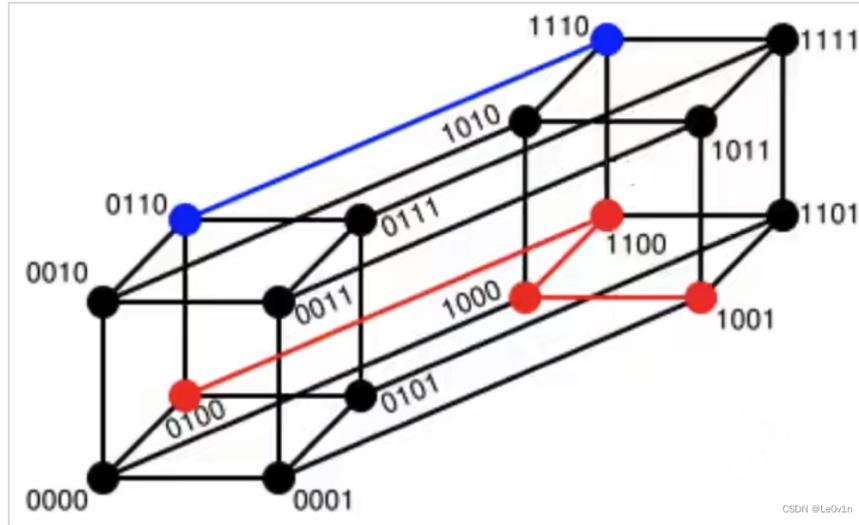
举个例子, 假设两个字符串 $A = "101110"$ 和 $B = "100101"$, 它们的汉明距离为 3, 因为 $A[2]$ 、 $A[4]$ 和 $A[5]$ 三个字符需要被替换成 “0” 才能得到 B 。

```
The Hamming distance between "1011101" and "1081881" is 2.  
The Hamming distance between "2143896" and "2233796" is 3.  
The Hamming distance between "toned" and " roses" is 3.
```

“1011101” 和 “1081881”的汉明距离是 2, 是因为它们在二进制下只有两个位置上的数字不同。

具体来说, 在这两个字符串中, 第 3 个和第 7 个字符不同。对应到二进制位上, “1011101” 中第 3 个字符是 1, 而 “1081881” 中是 0; “1011101” 中第 7 个字符是 0, 而 “1081881” 中是 8 (转换成二进制后是 “1000”)。

因此, 这两个字符串的汉明距离是 2, 即它们只需要进行 2 次单个字符的修改 (例如替换、插入、删除等操作), 就可以互相转换。



汉明距离是用于比较两个等长字符串之间的差异程度的度量方式，通常被应用在以下场景中：

1. 错误校验与纠错码：汉明距离可以用来检测并纠正传输或存储过程中的数据错误。例如，在网络传输中，发送方将信息编码成特定的纠错码，接收方通过计算汉明距离进行错误检测和自动纠错。
2. 基因组学：在基因序列比对中，汉明距离被广泛应用于比较不同物种之间的基因组序列。这有助于了解生物进化和变异的原因以及疾病的的相关性。
3. 数据库索引：在数据库中，可以使用汉明距离作为索引键来实现模糊搜索。例如，可以使用哈希函数计算所有文本字符串的汉明距离，并将其用作索引键，从而实现更快的模糊查询。
4. 图像处理：在图像处理中，汉明距离可以用于比较不同图片之间的相似度。例如，可以将图像分割成多个区域，并计算每个区域内像素值的汉明距离，从而确定两张图片之间的相似度。
5. 机器学习：在机器学习中，汉明距离可以用于计算特征向量之间的相似性或差异性，从而确定不同数据点之间的分类或聚类关系。

3.3.4 杰卡德距离 (Jaccard Distance)

杰卡德距离 (Jaccard distance) 是衡量两个集合之间差异性的一种度量方法，它定义为两个集合中不同元素的比例。

- 如果两个集合完全相同，则杰卡德距离为 0；
- 如果两个集合没有共同元素，则杰卡德距离为 1。

假设有两个集合 A 和 B ，它们的交集大小为 $|A \cap B|$ ，并集大小为 $|A \cup B|$ ，则它们的杰卡德距离可以表示为：

$$d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

其中， $|\cdot|$ 表示集合的元素个数。

举例：

```
X=[[1, 1, 0], [1, -1, 0], [-1, 1, 0]]
```

经计算得（注：以下计算中，把杰卡德距离定义为不同的维度的个数占“非全零维度”的比例）：

```
d = 0.5000 0.5000 1.0000
```

3.3.5 马氏距离 (Mahalanobis Distance)

马氏距离 (Mahalanobis distance) 是一种常用的度量两个样本点之间距离的方法，它考虑了不同特征之间的相关性。该距离度量方法通常用于聚类、分类和异常检测等机器学习任务中。

马氏距离可以看作是欧几里得距离的一种推广，它在计算时除了考虑各个特征之间的差异，还考虑了各个特征之间的相关性。具体来说，对于两个样本点 x 和 y ，它们之间的 Mahalanobis 距离 $D_M(x, y)$ 可以通过以下公式进行计算：

$$D_M(x, y) = \sqrt{(x - y)^T S^{-1} (x - y)}$$

其中， S 为协方差矩阵。当样本集合的协方差矩阵 S 已知时，可以利用上述公式计算任意两个样本点之间的马氏距离。需要注意的是，当协方差矩阵是单位矩阵时，马氏距离退化为欧几里得距离。

3.4 “连续属性” 和 “离散属性”的距离计算

我们常将属性划分为“连续属性”(continuous attribute) 和“离散属性”(categorical attribute)，前者在定义域上有无穷多个可能的取值，后者在定义域上是有限个取值。

- 若属性值之间存在序关系，则可以将其转化为连续值。例如：身高属性“高”“中等”“矮”，可转化为 $\{1, 0.5, 0\}$ 。

• 阁可夫斯基距离可以用于有序属性。

- 若属性值之间不存在序关系，则通常将其转化为向量的形式，例如：性别属性“男”“女”，可转化为 $\{(1, 0), (0, 1)\}$ 。

在距离中，序关系通常指的是点之间的大小（或者称为“比较”）关系。具体来说，在欧几里得空间中，两个点之间的距离是由它们在空间中的位置所决定的。因此，我们可以比较两个点的距离，即确定哪一个点更接近另一个点。

在序关系中，有三种可能的情况：等于、小于和大于。如果两个点之间的距离相同，则它们之间的序关系是等于。如果一个点比另一个点距离更近，则它们之间的序关系是小于。反之，如果一个点比另一个点距离更远，则它们之间的序关系是大于。

需要注意的是，序关系只能应用于可比较的对象。在距离中，任意两个点都是可比较的，因为它们之间的距离可以被确定。

小结：

1. 距离公式的基本性质：

1. 非负性
2. 统一性
3. 对称性
4. 直递性

2. 常见距离公式：

1. 欧式距离 (Euclidean Distance)：通过距离平方值进行计算
2. 曼哈顿距离 (Manhattan Distance)：通过距离的绝对值进行计算
3. 切比雪夫距离 (Chebyshev Distance)：维度的最大值进行计算
4. 阁可夫斯基距离 (Minkowski Distance)：
 1. 当 $p = 1$ 时，就是曼哈顿距离
 2. 当 $p = 2$ 时，就是欧式距离
 3. 当 $p \rightarrow \infty$ 时，就是切比雪夫距离

3. 属性

1. 连续属性
2. 离散属性：
 1. 存有序关系，可以将其转化为连续值
 2. 不存在序关系，通常将其转化为向量的形式

4. KNN 算法中 K 值的选择

学习目标：

- 知道 KNN 中 K 值大小选择对模型的影响

4.1 K 值选择说明

序号	电影名称	搞笑镜头	拥抱镜头	打斗镜头	电影类型	距离	K=5时
1	功夫熊猫	39	0	31	喜剧片	21.47	✓
2	叶问3	3	2	65	动作片	52.01	
3	二次曝光	2	3	55	爱情片	43.42	
4	代理情人	9	38	2	爱情片	40.57	✓
5	新步步惊心	8	34	17	爱情片	34.44	✓
6	谍影重重	5	2	57	动作片	43.87	
7	美人鱼	21	17	5	喜剧片	18.55	✓
8	宝贝当家	45	2	9	喜剧片	23.43	✓
9	唐人街探案	23	3	17	?	—	?

之前说到过这个表，当时选择的 K 是 5。 $K = 5$ 意味着我们需要选取距离最近的 5 部电影作为参考，然后选择数量最多的类别。

如果我们将 K 设置为 6，此时结果如下表：

序号	电影名称	搞笑镜头	拥抱镜头	打斗镜头	电影类型	距离	K=6时
1	功夫熊猫	39	0	31	① 喜剧片	21.47	✓
2	叶问3	3	2	65	动作片	52.01	
3	二次曝光	2	3	55	③ 爱情片	43.42	✓
4	代理情人	9	38	2	② 爱情片	40.57	✓
5	新步步惊心	8	34	17	① 爱情片	34.44	✓
6	谍影重重	5	2	57	动作片	43.87	
7	美人鱼	21	17	5	② 喜剧片	18.55	✓
8	宝贝当家	45	2	9	③ 喜剧片	23.43	✓
9	唐人街探案	23	3	17	?	—	?

CSDN @LeGuYin

此时我们发现，爱情片和喜剧片的数量都是 3，数量是相等的，那么《唐人街探案》属于爱情片还是喜剧片呢？此时模型也不知道了 😅 —— K 值选择过大面临的问题

假设我们将 K 值设置为 1，那么直接将距离最近的《美人鱼》所属的喜剧片作为《唐人街探案》的电影类型即可。但是这样就太单一了，过分地依赖一个数据了，结果并不可靠。假如《美人鱼》这部电影的类型在人工统计的时候错了，那么此时《唐人街探案》的电影类型也会跟着错，抗风险能力太弱了！—— K 值选择过小面临的问题

- K 值过小：容易受到异常点的影响
- K 值过大：受到样本均衡的问题

关于 K 值选择问题，李航博士的一书《统计学习方法》上所说：

1. 选择较小的 K 值，就相当于用较小的领域中的训练实例进行预测，“学习”近似误差会减小，只有与输入实例较近或相似的训练实例才会对预测结果起作用；与此同时带来的问题是“学习”的估计误差会增大。换句话说， K 值的减小就意味着整体模型变得复杂，容易发生过拟合。
2. 选择较大的 K 值，就相当于用较大领域中的训练实例进行预测，其优点是可以减少学习的估计误差，但缺点是学习的近似误差会增大。这时候，与输入实例较远（不相似的）训练实例也会对预测器起作用，使预测发生错误，且 K 值的增大就意味着整体的模型变得简单。
3. $K = N$ (N 为训练样本个数)，则完全不可靠。因为此时无论输入实例是什么，都只是简单的预测它属于在训练实例中最多的类，模型过于简单，忽略了训练实例中大量有用信息。

在实际应用中， K 值一般取一个比较小的数值，例如采用交叉验证法（简单来说，就是把训练数据分成两组：训练集和测试集）来选择最优的 K 值。

4.2 近似误差和估计误差

- 近似误差：
 - 对现有训练集的训练误差，主要关注训练集
 - 如果近似误差过小可能会出现过拟合的现象。虽然对现有的训练集能有很好的预测效果，但是对未知的测试样本将会出现较大偏差的预测。
 - 模型本身不是最接近最佳模型。
- 估计误差：
 - 可以理解为对测试集的测试误差，主要关注测试集
 - 估计误差小说明对未知数据的预测能力好
 - 模型本身最接近最佳模型
- 近似误差：训练集 Loss；
- 估计误差：测试集 Loss。
- 测试集 Loss 低才是好模型，说明模型的泛化能力出色；
- 训练集 Loss 低只能说明模型对数据集拟合得较好，但不能说明它的泛化能力强。

小结：

- KNN 中 K 值大小选择对模型的影响【知道】
 - K 值过小：
 - 容易受到异常点的影响
 - 容易过拟合
 - K 值过大：

- 受到样本均衡的问题容易欠拟合

5. kd 树 (kd Tree)

学习目标：

- 知道 kd 树构建和搜索过程

问题导入：实现 K 近邻算法时，主要考虑的问题是如何对训练数据进行快速 K 近邻搜索。这在特征空间的维数大及训练数据容量大时尤其必要。

K 近邻法最简单的实现是线性扫描（穷举搜索），即计算输入实例与每一个训练实例的距离并存储好，之后再查找 K 近邻。当训练集很大时，计算非常耗时。

为了提高 KNN 搜索的效率，可以考虑使用特殊的结构存储训练数据，以减小计算距离的次数。

5.1 kd 树简介

5.1.1 什么是 kd 树

根据 KNN 每次需要预测一个点时，我们都需要计算训练数据集里每个点到这个点的距离，然后选出距离最近的 K 个点进行投票。当数据集很大时，这个计算成本非常高。

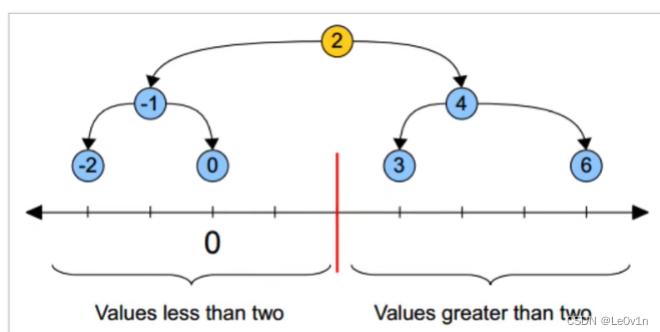
kd 树：为了避免每次都重新计算一遍距离，算法会把距离信息保存在一棵树里，这样在计算之前从树里查询距离信息，尽量避免重新计算。其基本原理是：如果 A 和 B 距离很远，B 和 C 距离很近，那么 A 和 C 的距离也很远。有了这个信息，就可以在合适的时候跳过距离远的点。

这样优化后的算法复杂度可降低到 $O(D \times N \log N)$ 。

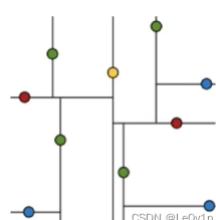
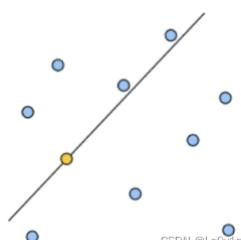
K 近邻搜索如果使用线性扫描（穷举搜索）时的算法复杂度为 $O(D \times N^2)$

kd 树的英文全称是：K-dimensional Tree (K-dimensional Tree)。

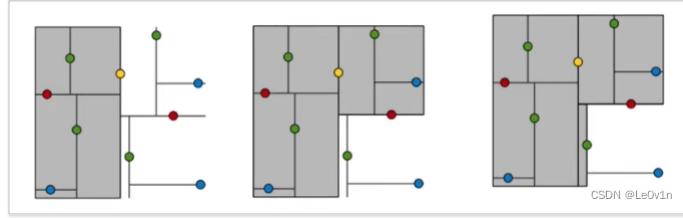
5.1.2 kd 树的原理



黄色的点作为根节点，小于 root 的在左边，大于 root 的在右边，接下来再不断地划分。分割的那条线叫做分割超平面 (splitting hyperplane)，在一维中是一个点，二维中是线，三维的是面。



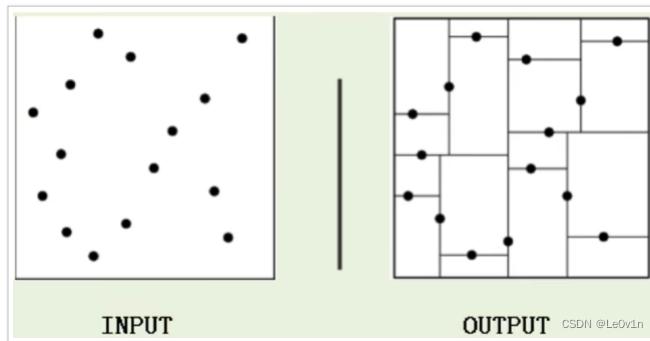
黄色节点就是 Root 节点，下一层是红色，再下一层是绿色，再下一层是蓝色。



要想实现划分和搜索，分为两个过程：

1. 树的建立
2. 最近邻域搜索 (Nearest-Neighbor Lookup)

kd 树 (K-dimension tree) 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。kd 树是一种二叉树，表示对 K 维空间的一个划分，构造 kd 树相当于不断地用垂直于坐标轴的超平面将 K 维空间切分，构成一系列的 K 维超矩形区域。kd 树的每个结点对应于一个 K 维超矩形区域。利用 kd 树可以省去对大部分数据点的搜索，从而减少搜索的计算量。



类比 “二分查找”：给出一组数据 [9 1 4 7 2 5 0 3 8]，要查找 8。

- 如果挨个查找（线性扫描），那么将会把数据集都遍历一遍
- 而如果排一下序那数据集就变成了：[0 1 2 3 4 5 6 7 8 9]，按前一种方式我们进行了很多没有必要的查找。

现在如果我们以 5 为分界点，那么数据集就被划分为了左右两个 “簇 (Cluster)” [0 1 2 3 4] 和 [6 7 8 9]。

因此，根本就没有必要进入第一个簇，可以直接进入第二个簇进行查找。把二分查找中的数据点换成 k 维数据点，这样的划分就变成了用超平面对 K 维空间的划分。空间划分就是对数据点进行分类，“挨得近”的数据点就在一个空间里面。

二叉树中簇的英文是 cluster，它的含义是：簇是一种特殊的树形数据结构，它由若干个二叉树组成，每个二叉树都满足以下条件：

1. 它的左子树的所有节点的值都小于它的根节点的值；
2. 它的右子树的所有节点的值都大于它的根节点的值；
3. 左右子树也分别是二叉树。

在二叉树中，簇的作用是将相似的节点放在同一个簇中，以便更高效地查找、插入和删除节点。这样做可以减少树的高度，从而提高查找、插入和删除操作的效率。

例如，在一个公司的员工档案系统中，可以将员工按照他们的工资、工作年限等因素分成不同的簇，然后使用搜索引擎等工具来快速查找、浏览和更新这些员工的信息。

二叉树中簇还可以用于构建广度优先搜索 (BFS) 算法。在 BFS 算法中，我们从根节点开始遍历整个树，每次选择一个起始节点，然后遍历它的所有邻居节点，并将它们加入到一个队列中。当我们遍历完整个树时，我们将队列中的所有节点依次出队，即可得到从根节点到达该节点的路径。这个过程可以用图形化的方式表示为一个树形结构，其中节点的度数表示其与其父节点的关系，节点的高度表示它所在的簇的深度。

5.2 kd 树的构造方法

1. 构造根结点，使根结点对应于 K 维空间中包含所有实例点的超矩形区域。
2. 通过递归的方法，不断地对 K 维空间进行切分，生成子结点。在超矩形区域上选择一个坐标轴和在此坐标轴上的一个切分点，确定一个超平面，这个超平面通过选定的切分点并垂直于选定的坐标轴，将当前超矩形区域切分为左右两个子区域（子结点）；这时，实例被分到两个子区域。

3. 上述过程直到子区域内没有实例时终止（终止时的结点为叶结点）。在此过程中，将实例保存在相应的结点上。

4. 通常，循环的选择坐标轴对空间切分，选择训练实例点在坐标轴上的中位数为切分点，这样得到的 kd 树是平衡的（平衡二叉树：它是一棵空树，或其左子树和右子树的深度之差的绝对值不超过 1，且它的左子树和右子树都是平衡二叉树）。

kd 树中每个节点是一个向量，和二叉树按照数的大小划分不同的是，kd 树每层需要选定向量中的某一维，然后根据这一维按左小右大的方式划分数据。在构建 kd 树时，关键需要解决两个问题：

1. 选择向量的哪一维进行划分

2. 如何划分数据

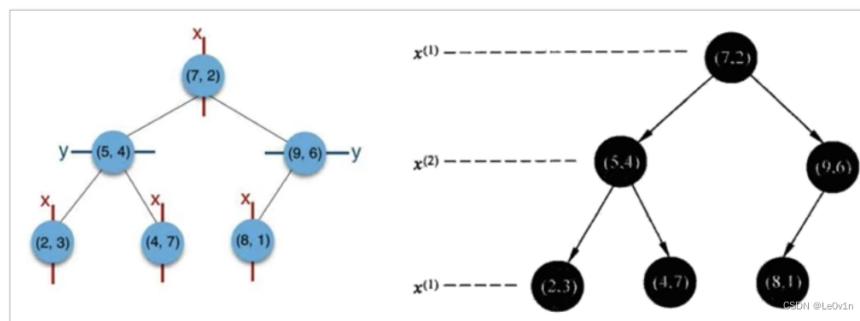
第一个问题简单的解决方法可以是随机选择某一维或按顺序选择，但是更好的方法应该是在数据比较分散的那一维进行划分（分散的程度可以根据方差来衡量）。

第二个问题中，好的划分方法可以使构建的树比较平衡，可以每次选择中位数来进行划分。

5.3 kd 树的练习案例

5.3.1 树的建立

给定一个二维空间数据集： $T = \{(2, 3), (5, 4), (9, 6), (4, 7), (8, 1), (7, 2)\}$ ，构造一个平衡 kd 树。



第一维度 (x 轴) : 2, 5, 9, 4, 8, 7

排序: 2, 4, 5, 7, 8, 9

中位数: 6

1. 【选择根结点】所以选择 $x = 5$ 或 $x = 7$ 都可以，这里选取 $x = 7$ 作为 root，即 $(7, 2)$ 为 root 根节点。

2. 【通过递归的方法，不断地对 K 维空间进行切分，生成子结点】既然 root 已经指定，那么就可以将其他节点放在 root 的左右，即左小右大—— $x = 2, 4, 5$ 在 root 的左边； $x = 8, 9$ 在 root 的右边。

1. 对 $x = 2, 4, 5$ 再进行划分，此时不再使用 x 轴，而使用 y 轴进行划分。 $x = 2, 4, 5$ 对应的数据 y 轴数据是 $y = 3, 4, 7$ 。

1. 中位数为 4，将 4 作为 root

2. root 确定，左小右大，所以 $(2, 3)$ 放在左边， $(4, 7)$ 放在右边。

2. 对 $x = 8, 9$ 再进行划分，此时不再使用 x 轴，而使用 y 轴进行划分。 $x = 8, 9$ 对应的数据 y 轴数据是

$y = 1, 6$ 。

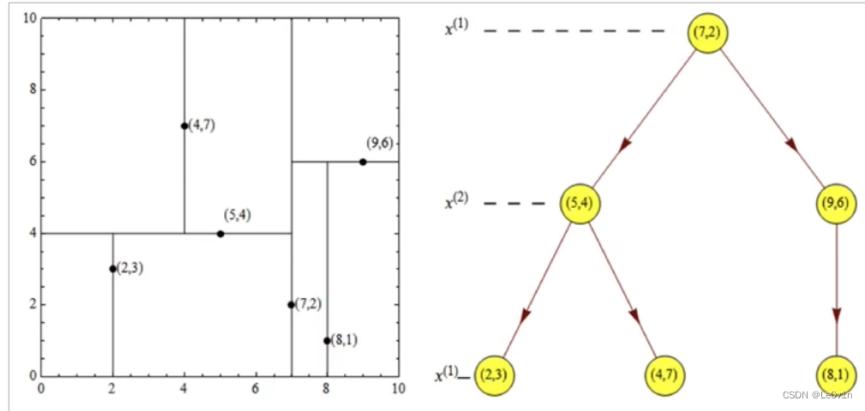
1. 因为只有两个数，所以中位数选择哪个都可以，这里选择 $x = 6$ 作为 root

2. 只剩下两个数据了，且 y 轴数据比 root 小，所以放在左边

因为要垂直划分，所以对于二维平面，先 x 轴再 y 轴再 x 轴再 y 轴，以此类推。

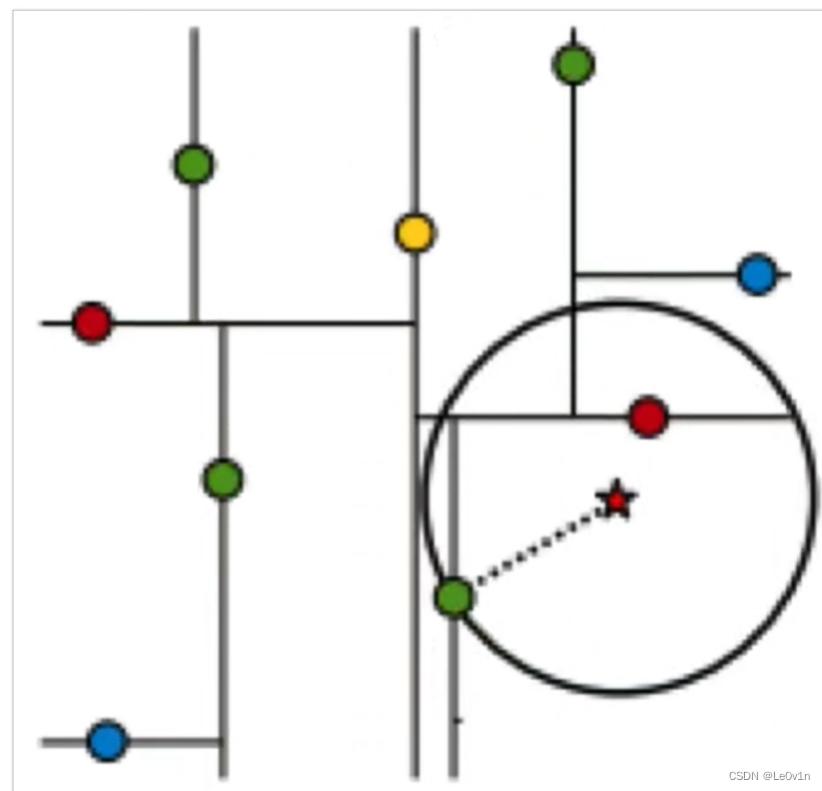
思路引导

根结点对应包含数据集 T 的矩形，选择 $x^{(1)}$ 轴，6 个数据点的 $x^{(1)}$ 坐标中位数是 6，这里选最接近的 $(7, 2)$ 点，以平面 $x^{(1)} = 7$ 将空间分为左、右两个子矩形（子结点）；接着左矩形以 $x^{(2)} = 4$ 分为两个子矩形（左矩形中 $\{(2, 3), (5, 4), (4, 7)\}$ 点的 $x^{(2)}$ 坐标中位数正好为 4），右矩形以 $x^{(2)} = 6$ 分为两个子矩形，如此递归，最后得到如下图所示的特征空间划分和 kd 树。



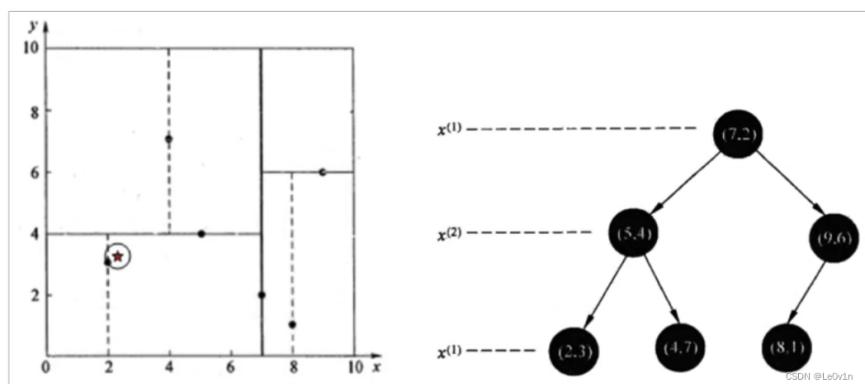
5.3.2 最近邻域的搜索

假设标记为星星的点是 test point \star , 绿色的点是找到的近似点。在回溯过程中, 需要用到一个队列, 存储需要回溯的点。在判断其他子节点空间中是否有可能有距离查询点更近的数据点时, 做法是以查询点为圆心, 以当前的最近距离为半径画圆, 这个圆称为候选超球 (candidate hypersphere), 如果圆与回溯点的轴相交, 则需要将轴另一边的节点都放到回溯队列里面来。



样本集: $\{(2, 3), (5, 4), (9, 6), (4, 7), (8, 1), (7, 2)\}$

一、查找点为 $(2.1, 3.1)$



Step.1 构造队列

从根结点开始划分。在当初构建树的时候, root 是根据 x 轴划分出来的, 即 $x^{(1)}$ 为 x 轴, $x^{(2)}$ 为 y 轴。当初树是怎么建

立而来的，那么在构造队列的时候也遵循当时的规则。

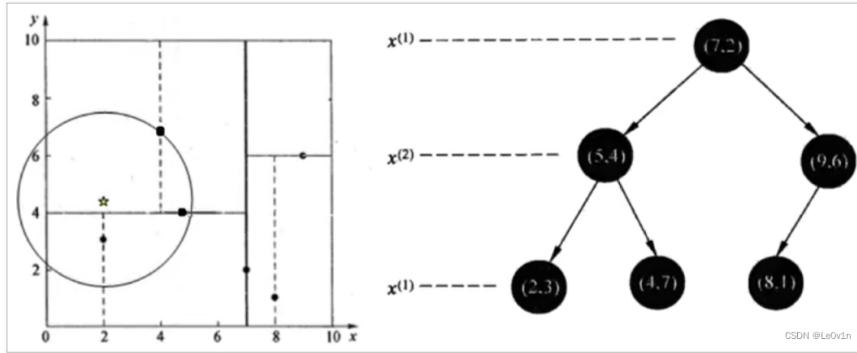
1. (2.1, 3.1) 与 root 在 x 轴对比，在 root 的左边。
2. (2.1, 3.1) 与 (5, 4) 在 y 轴对比，在 (5, 4) 的左边。
3. 此时已经到了最后一层，因此停止。此时最近的点是 (2, 3)。

Step.2 回溯搜索

走完上面的过程就得到一个队列 $\langle(7, 2), (5, 4), (2, 3)\rangle$ 。找到该队列后，我们需要回溯，目的是找最近点。

1. 从队列中取出 (2, 3) 作为当前最佳结点 nearest，此时 (2.1, 3.1) 与 (2, 3) 的欧式距离为 0.141。
2. 然后回溯至 (5, 4)。又因为以 (2.1, 3.1) 为圆心， $dist = 0.141$ 为半径画一个圆，并不和超平面 $y = 4$ 相交，如上图，所以不必跳到结点 (5, 4) 的右子空间去搜索，因为右子空间中不可能有更近样本点了。
3. 再回溯至 (7, 2)。同理，以 (2.1, 3.1) 为圆心， $dist = 0.141$ 为半径画一个圆并不和超平面 $x = 7$ 相交，所以也不用跳到结点 (7, 2) 的右子空间去搜索。
4. 至此，search_path 队列为空，结束整个搜索，返回 nearest(2, 3) 作为 (2.1, 3.1) 的最近邻点，最近距离为 0.141。

二、查找点为 (2, 4.5)



Step.1 构造队列

在 (7, 2) 处测试到达 (5, 4)，在 (5, 4) 处测试到达 (4, 7)【优先选择在本域搜索】，然后 search_path 中的结点为 $\langle(7, 2), (5, 4), (4, 7)\rangle$ ，从 search_path 中取出 (4, 7) 作为当前最佳结点 nearest， $dist$ 为 3.202。

Step.2 回溯搜索

1. 回溯至 (5, 4)。以 (2, 4.5) 为圆心， $dist = 3.202$ 为半径画一个圆与超平面 $y = 4$ 相交，所以需要跳到 (5, 4) 的左子空间去搜索。所以要将 (2, 3) 加入到 search_path 中，现在 search_path 中的结点为 $\langle(7, 2), (2, 3)\rangle$ ；另外，(5, 4) 与 (2, 4.5) 的距离为 $3.04 < dist = 3.202$ ，所以将 (5, 4) 赋给 nearest，并且 $dist = 3.04$ 。
2. 回溯至 (2, 3)。(2, 3) 是叶子节点，直接判断 (2, 3) 是否离 (2, 4.5) 更近，计算得到距离为 1.5，所以 nearest 更新为 (2, 3)， $dist$ 更新为 1.5。
3. 回溯至 (7, 2)。同理，以 (2, 4.5) 为圆心， $dist = 1.5$ 为半径画一个圆并不和超平面 $x = 7$ 相交，所以不用跳到结点 (7, 2) 的右子空间去搜索。
4. 至此，search_path 为空，结束整个搜索，返回 nearest(2, 3) 作为 (2, 4.5) 的最近邻点，最近距离为 1.5。

总结：

- kd 树的构建过程【知道】
 1. 构造根节点
 2. 通过递归的方法，不断地对 k 维空间进行划分，生成子节点
 3. 重复第二步骤，直到子区域中没有示例时终止
 4. 需要关注细节：
 1. 选择向量的哪一维进行划分
 2. 如何划分数据
- kd 树的搜索过程【知道】
 1. 二叉树搜索比较待查询节点和分裂节点的分裂维的值（小于等于就进入左子树分支，大于就进入右子树分支直到叶子结点）
 2. 顺着“搜索路径”找到最近邻的近似点
 3. 回溯搜索路径，并判断搜索路径上的结点的其他子结点空间中是否可能有距离查询点更近的数据点。如果有可能，则需要跳到其他子结点空间中去搜索
 4. 重复这个过程直到搜索路径为空

6. 练习案例：鸢尾花种类预测

学习目标：

- 知道 sklearn 中获取数据集的方法
- 知道 sklearn 中对数据集的划分方法

本实验介绍了使用 Python 进行机器学习的一些基本概念。在本案例中，将使用 K-NearestNeighbor (KNN) 算法对鸢尾花的种类进行分类，并测量花的特征。

本案例目的：

1. 遵循并理解完整的机器学习过程
2. 对机器学习原理和相关术语有基本的了解
3. 了解评估机器学习模型的基本过程

6.1 鸢尾花数据集介绍

Iris 数据集是常用的分类实验数据集，由 Fisher 于 1936 收集整理。Iris 也称鸢尾花卉数据集，是一类多重变量分析的数据集。关于数据集的具体介绍：

- 特征值 4 个：花瓣的长度、花瓣的宽度、花萼的长度、花萼的宽度
- 目标值 3 个：setosa, vericolor, virginica

6.2 scikit-learn 中的数据集介绍

一、scikit-learn 数据集 API 介绍

```
# 获取本地小规模数据集（无需下载）
sklearn.datasets.load_data集名称()

# 获取大规模数据集，需要从网络中下载。参数data_home表示数据集的下载目录，默认是~/scikit_learn_data
sklearn.datasets.fetch_数据集名称(data_home=None)
```

二、sklearn 小数据集

```
# 加载并返回鸢尾花数据集
sklearn.datasets.load_iris()
```

名称	数量
类别	3
特征	4
样本总数	150
每个类别数量	50

三、sklearn 大数据集

```
sklearn.datasets.fetch_20newsgroups(data_home=None, subset="train")
```

- subset : "train" , "test" , "all" 可选，目的是要加载的数据集部分
 - "train" : 加载训练数据
 - "test" : 加载测试数据
 - "all" : 加载训练 + 测试数据

6.3 sklearn 数据集返回值介绍

load 和 fetch 返回的数据类型为 `datasets.base.Bunch`(字典格式)，其中包含：

- data : 特征数据数组，是 `[n_samples * n_features]` 的二维 `numpy.ndarray` 数组
- target : 标签数组，是 `n_samples` 的一维 `numpy.ndarray` 数组
- DESCR : 数据描述
- feature_names : 特征名。如新闻数据、手写数字。回归数据集则没有该键值对。
- target_names : 标签名

```
from sklearn.datasets import load_iris, fetch_20newsgroups

# 获取鸢尾花数据集（返回值是一个继承自字典的Bunch）
iris = load_iris()
```

```

# 查看字典的所有键key
print(iris.keys())
"""
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR',
          'feature_names', 'filename', 'data_module'])
"""

```

- `data` : 这个键值表示鸢尾花数据集中所有样本的特征值 (特征矩阵), 它是一个二维数组, 每行代表一个样本, 每列代表一个特征。
- `target` : 这个键值表示鸢尾花数据集中所有样本的真实标签 (目标向量), 它是一个一维数组, 每个元素表示对应样本的类别编号。
- `frame` : 这个键值表示数据集的 DataFrame 对象, 其中包含了 `data` 和 `target` 以及其他元数据。
- `target_names` : 这个键值表示类别名称, 它是一个字符串数组, 每个元素对应一个类别名称。
- `DESCR` : 这个键值表示对数据集的描述, 包括数据集来源、数据集大小等详细信息。
- `feature_names` : 这个键值表示特征名称, 它是一个字符串数组, 每个元素对应一个特征名称。
- `filename` : 这个键值表示数据集文件所在的路径和文件名。
- `data_module` : 这是一个自定义键值, 表示数据集模块的版本信息。

这个包含多个键值的字典提供了数据集的各种元数据信息, 使得我们能够更好地理解和使用该数据集。

6.4 查看鸢尾花数据集的数据分布

通过创建一些图, 以形象地查看不同类别是如何通过特征来区分的。在理想情况下, 标签类将由一个或多个特征对完美分隔。在现实世界中, 这种理想情况很少会发生。

`sns.lmplot(data=None, x=None, y=None, hue=None, col=None, row=None, fit_reg=True)` 是一个非常有用的方法, 它会在绘制二维散点图时, 自动完成回归拟合。其中:

- `data` 是关联到的数据集
- `x`, `y` 分别代表横纵坐标的列名
- `hue` 代表按照 `target` 即花的类别分类显示
- `fit_reg=True` 是否进行线性拟合

```

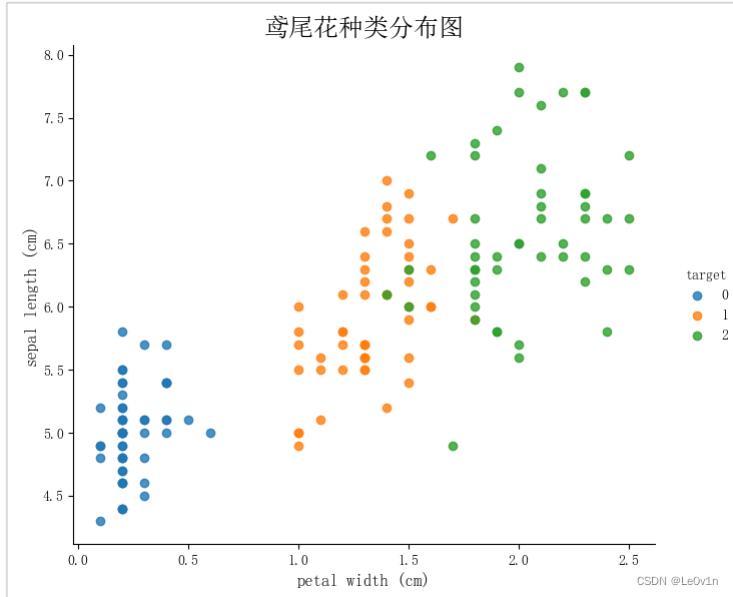
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_iris
from matplotlib.font_manager import FontProperties
import matplotlib
myfont = FontProperties(fname=r'C:\Windows\Fonts\simsun.ttc', size=14)
matplotlib.rcParams['font.sans-serif'] = ['SimSun']
matplotlib.rcParams['font.family'] = 'sans-serif'
matplotlib.rcParams['axes.unicode_minus'] = False

# 画图函数
def plot_iris(iris, col1, col2):
    """
    sns.lmplot()是Seaborn库中的一个函数, 用于绘制数据集中两个变量之间关系的回归图。
    它可以创建一个带有线性回归拟合的散点图, 并且可以通过'hue'、'col'和'row'参数在多个类别之间进行分面。
    """
    sns.lmplot(x=col1, y=col2, data=iris, hue="target", fit_reg=False)
    plt.xlabel(col1, fontproperties=myfont, size=12)
    plt.ylabel(col2, fontproperties=myfont, size=12)
    plt.title(label="鸢尾花种类分布图", fontproperties=myfont, size=18)
    plt.show()

if __name__ == "__main__":
    # 直接加载的时候让其变为DF格式
    iris = load_iris(as_frame=True)["frame"]
    print(iris)
    """
        sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  target
    0            5.1           3.5            1.4           0.2      0
    1            4.9           3.0            1.4           0.2      0
    2            4.7           3.2            1.3           0.2      0
    3            4.6           3.1            1.5           0.2      0
    4            5.0           3.6            1.4           0.2      0
    ..
    ...
    ...
    145           6.7           3.0            5.2           2.3      2
    146           6.3           2.5            5.0           1.9      2
    147           6.5           3.0            5.2           2.0      2
    148           6.2           3.4            5.4           2.3      2
    149           5.9           3.0            5.1           1.8      2
    """
    [150 rows x 5 columns]
    """
    print(iris.keys()) # Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)', 'target'], dtype='object')

    # 调用画图函数绘图
    plot_iris(iris=iris, col1="petal width (cm)", col2="sepal length (cm)")

```



6.5 数据集的划分

机器学习一般的数据集会划分为两个部分：

1. 训练数据：用于训练，调整模型
2. 测试数据：在模型检验时使用，用于评估模型是否有效

划分比例：

- 训练集：70% / 80% / 75%
- 测试集：30% / 20% / 25%

数据集划分 API：

`sklearn.model_selection.train_test_split` 是 scikit-learn 库中的一个函数，用于将数组或矩阵拆分为随机的训练和测试子集。它是一个快速实用的工具，可以将输入验证、下一个 (`ShuffleSplit().split(X, y)`) 和应用于输入数据的拆分（以及可选的子采样）数据封装到一个调用中，以便一次性拆分数据。

```
sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None,
                                         random_state=None, shuffle=True, stratify=None)
-> X_train, X_test, y_train, y_test
```

该函数的主要参数包括：

- `*arrays` : 序列，允许长度大于等于 2 的元组。需要拆分的数组。这里的形参应该是 `x` 和 `y`，即
 - `x=None` 表示特征数据
 - `y=None` 表示目标数据
- `test_size` : 浮点数、整数或 `None` (默认为 `None`)。
 - 如果为浮点数，则应在 0.0 和 1.0 之间，并表示要包含在测试拆分中的数据比例。
 - 如果为整数，则表示测试样本的绝对数量。
 - 如果为 `None`，则将其设置为 0.25。
- `train_size` : 浮点数、整数或 `None` (默认为 `None`)。
 - 如果为浮点数，则应在 0.0 和 1.0 之间，并表示要包含在训练拆分中的数据比例。
 - 如果为整数，则表示训练样本的绝对数量。
 - 如果为 `None`，则将其自动设置为 `n_samples - test_size`。
- `random_state` : `int` 或 `RandomState` 实例或 `None` (默认为 `None`)。即 随机数种子，不同的种子会造成不同的随机采样结果。相同的种子采样结果相同。
- `shuffle` : 布尔值 (默认为 `True`)。是否在拆分之前打乱数据。
- `stratify` : 数组或 `None` (默认为 `None`)。如果不是 `None`，则数据按照指定类别进行分层抽样。

返回值是拆分的训练和测试数据。返回的数据类型与输入数据类型相同。例如，如果你传递了两个数组 `x` 和 `y` 作为输入数据，则该函数将返回四个数组：`X_train`、`X_test`、`y_train` 和 `y_test`。其中：

- `X_train` 和 `y_train` 是拆分后的训练数据
- `X_test` 和 `y_test` 是拆分后的测试数据

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

```

import numpy as np

# 1. 获取鸢尾花数据集
iris = load_iris()

# 2. 对鸢尾花数据集进行分割
x_train, x_test, y_train, y_test = train_test_split(iris.data,
                                                    iris.target,
                                                    test_size=0.20, # 写一个就行，另一个会自动算出来
                                                    shuffle=True)

print(f"x_train: {x_train.shape}") # (120, 4)
print(f"x_test: {x_test.shape}") # (30, 4)
print(f"y_train: {y_train.shape}") # (120,)
print(f"y_test: {y_test.shape}") # (30,)

# 3. 测试一下随机种子对划分的影响
x_train_1, x_test_1, y_train_1, y_test_1 = train_test_split(iris.data,
                                                            iris.target,
                                                            random_state=1)
x_train_2, x_test_2, y_train_2, y_test_2 = train_test_split(iris.data,
                                                            iris.target,
                                                            random_state=2)
x_train_3, x_test_3, y_train_3, y_test_3 = train_test_split(iris.data,
                                                            iris.target,
                                                            random_state=2)
print("如果随机种子不相同: ", np.all(x_train_1 == x_train_2)) # False
print("如果随机种子相同: ", np.all(x_train_2 == x_train_3)) # True

```

小结：

- 获取数据集【知道】
 - 小数据： sklearn.datasets.load_*
 - 大数据集： sklearn.datasets.fetch_*
- 数据集返回值介绍【知道】
 - 返回值类型是 bunch —— 是一个字典类型。
 - 返回值的属性：
 - data : 特征数据数组
 - target : 标签（目标）数组
 - DESCR : 数据描述
 - feature_names : 特征名
 - target_names : 标签（目标值）名
- 数据集的划分【掌握】
 - sklearn.model_selection.train_test_split(arrays, "options")
 - 参数：
 - x : 特征值
 - y : 目标值
 - test_size : 测试集大小
 - random_state : 随机数种子
 - 返回值： x_train , x_test , y_train , y_test

7. 特征工程之特征预处理

学习目标：

- 了解什么是特征预处理
- 知道归一化和标准化的原理及区别

7.1 特征预处理的定义

特征预处理就是对数据进行清洗和整理，以便更好地进行机器学习。它包括一些常见的操作，如缩放数据以使其处于相同的范围内、删除异常值、将分类变量转换为数值变量等。特征预处理的目的是使数据更容易被机器学习算法理解和使用。它可以帮助提高模型的准确性和性能。

特征1 特征2 特征3 特征4				特征1 特征2 特征3 特征4			
90	2	10	40	1.	0.	0.	0.
60	4	15	45	0.	1.	1.	0.83
75	3	13	46	0.5	0.5	0.6	1.

CSDN @Leovyin

Q：为什么我们要进行归一化 / 标准化？

A：特征的单位或者大小相差较大，或者某特征的方差相比其他的特征要大出几个数量级，容易影响（支配）目标结果，使得一些算法无法学习到其它的特征。

举例：约会对象数据

里程数	公升数	消耗时间比	评价
14488	7.153469	1.673904	smallDoses
26052	1.441871	0.805124	didn'tLike
75136	13.147394	0.428964	didn'tLike
38344	1.669788	0.134296	didn'tLike
72993	10.141740	1.032955	didn'tLike
35948	6.830792	1.213192	largeDoses
42666	13.276369	0.543880	largeDoses
67497	8.631577	0.749278	didn'tLike
35483	12.273169	1.508053	largeDoses
50242	3.723498	0.831917	didn'tLike

相亲约会对象数据，这个样本是男士的数据，有三个特征：

1. 玩游戏所消耗时间的百分比
2. 每年获得的飞行常客里程数
3. 每周消费的冰淇淋公升数

然后有一个所属类别，被女士评价的三个类别：

1. 不喜欢 didn'tLike
2. 魅力一般 smallDoses
3. 极具魅力 largeDoses

对于不同的女士，可能对“飞行里程数”更加看重，这就导致这个特征对所属类别造成不平衡的影响，但是统计的人觉得这三个特征同等重要。

因此我们需要用到一些方法进行无量纲化，使不同规格的数据转换到同一规格。

7.2 包含内容（数值型数据的无量纲化）

1. 归一化
2. 标准化

7.3 特征预处理 API

sklearn.preprocessing

7.4 归一化

7.4.1 归一化的定义

归一化是一种常用的数据预处理方法，它可以将数据缩放到一个特定的范围内。最常用的归一化方法之一是最小 - 最大缩放 (Min-Max Scaling)，它将数据缩放到 $[0, 1]$ 的范围内。其公式如下：

$$x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

其中， x 表示原始数据， x_{\min} 和 x_{\max} 分别表示数据中的最小值和最大值， x_{norm} 表示归一化后的数据。

特征1	特征2	特征3	特征4
90	2	10	40
60	4	15	45
75	3	13	46

特征1	特征2	特征3	特征4
90-60	2-2	10-10	40-40
60-60	4-2	15-10	45-40
75-60	3-2	13-10	46-40

如果你想指定归一化的范围，可以使用最小 - 最大缩放 (Min-Max Scaling) 方法，并对其进行修改。假设你想将数据缩放到 $[a, b]$ 的范围内，那么可以使用以下公式：

$$x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \times (b - a) + a$$

其中， x 表示原始数据， x_{\min} 和 x_{\max} 分别表示数据中的最小值和最大值， x_{norm} 表示归一化后的数据， a 和 b 表示你想将数据缩放到的范围。

例如，如果你想将数据缩放到 $[-1, 1]$ 的范围内，那么可以使用以下公式：

$$x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \times 2 - 1$$

归一化不一定非要将数据缩放到 $[0, 1]$ 的范围内。虽然最小 - 最大缩放 (Min-Max Scaling) 是一种常用的归一化方法，它将数据缩放到 $[0, 1]$ 的范围内，但也有其他归一化方法，它们将数据缩放到不同的范围内。

例如，Z 分数标准化 (Z-Score Normalization) 是一种常用的归一化方法，它将数据缩放为均值为 0 ($\mu = 0$)，标准差为 1 ($\sigma = 1$) 的分布。这种方法并不会将数据缩放到固定的范围内，而是根据数据的分布情况进行缩放。

此外，我们也可以使用其他归一化方法，如 Robust Scaling、Quantile Transformation 等，它们都有各自的优缺点和适用场景。

总之，归一化并不是固定将数据缩放到 [0, 1] 的范围内，而是根据实际情况选择合适的归一化方法来进行数据预处理。

7.4.2 归一化的 API

1. `sklearn.preprocessing.MinMaxScaler()`

2. `MinMaxScaler.fit_transform()`

一、`sklearn.preprocessing.MinMaxScaler()`

`sklearn.preprocessing.MinMaxScaler` 是 scikit-learn 库中的一个类，它提供了最小 - 最大缩放 (Min-Max Scaling) 的实现。

```
sklearn.preprocessing.MinMaxScaler(feature_range=(0, 1), *, copy=True, clip=False)
```

该类的主要参数包括：

- `feature_range`：元组（默认为 (0, 1)）。期望的缩放范围。所有特征都将被缩放到这个范围内。
- `copy`：布尔值（默认为 `True`）。如果为 `True`，则对数据进行复制，否则进行原地缩放。
- `clip`：布尔值（默认为 `False`）。如果为 `True`，则在转换过程中将数据裁剪到指定范围内。

`clip` 参数用于控制是否在转换过程中将数据裁剪到指定范围内。如果将 `clip` 设置为 `True`，则在使用 `transform` 方法转换数据时，所有超出指定范围的值都将被裁剪到范围的边界上。

例如，如果你将 `feature_range` 设置为 (0, 1)，并且将 `clip` 设置为 `True`，则所有小于 0 的值都将被裁剪为 0，所有大于 1 的值都将被裁剪为 1。

如果将 `clip` 设置为 `False`（默认值），则不会对数据进行裁剪，超出指定范围的值将保持不变。

请注意，`clip` 参数仅在使用 `transform` 方法转换数据时生效，在使用 `fit` 方法拟合数据时不会对数据进行裁剪。

二、`MinMaxScaler.fit_transform()`

`MinMaxScaler.fit_transform` 方法用于同时拟合和转换数据。

注意：`fit` 是拟合数据；而 `transform` 是转换数据

```
fit_transform(X, y=None, **fit_params)
```

它接受以下参数：

- `X`：数组，形状为 `(n_samples, n_features)`。表示要转换的数据。
- `y`：忽略此参数，仅用于与 Pipeline API 兼容。

`fit_transform` 方法返回一个新的数组，其中包含缩放后的数据。返回的数组与输入数组具有相同的形状。

例如，如果你传入一个形状为 `(n_samples, n_features)` 的数组，则该方法将返回一个形状为 `(n_samples, n_features)` 的数组。

7.4.3 归一化的数据计算

我们对以下数据进行运算，在 `dating.txt` 中。保存的就是之前的约会对象数据。

milage	Liters	Consumtime	target
14488	7.153469	1.673904	2
26052	1.441871	0.405124	1
75136	13.147394	0.428964	1
38344	1.669788	0.134296	1
72993	10.14174	1.032955	1
35948	6.830792	1.213192	3
42666	13.276369	0.543883	1
67497	8.631577	0.749278	1
35483	12.273169	1.508053	3

分析：

1. 实例化 `MinMaxScalar`
2. 通过 `fit_transform` 转换

我们不需要转换目标值（`target`），因为它是 `Ground Truth`，我们千万不要转换，不然会破坏标准答案。

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

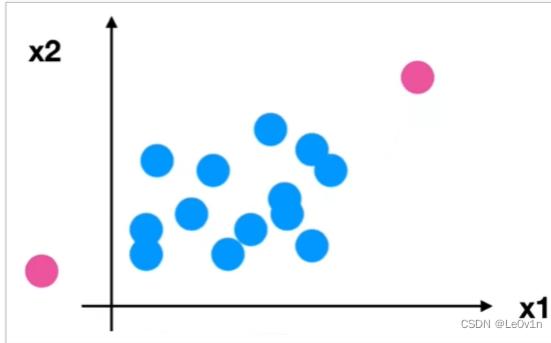
data = pd.read_csv("../data/dating.txt", delimiter='\t')
print(data)
"""
    milage      Liters Consumtime target
0     40920    8.326976   0.953952      3
1     14488    7.153469   1.673904      2
2     26052    1.441871   0.805124      1
3     75136   13.147394   0.428964      1
4     38344   1.669788   0.134296      1
...
995   11145    3.410627   0.631838      2
996   68846    9.974715   0.669787      1
997   26575   10.650102   0.866627      3
998   48111    9.134528   0.728045      3
999   43757    7.882601   1.332446      3
[1000 rows x 4 columns]
"""

# 1. 实例化一个转换器
transformer = MinMaxScaler(feature_range=(2, 3), copy=True)

# 2. 调用fit_transform方法
data = transformer.fit_transform(X=data.loc[:, ["milage", "Liters", "Consumtime"]])
# 或者data = transformer.fit_transform(X=data[["milage", "Liters", "Consumtime"]])
print("最小值-最大值归一化处理的结果为:\r\n", data)
"""
最小值-最大值归一化处理的结果为:
[[2.4483253 2.39805139 2.56233353]
 [2.15873259 2.34195467 2.98724416]
 [2.28542943 2.06892523 2.47449629]
 ...
 [2.29115949 2.50910294 2.51079493]
 [2.52711097 2.43665451 2.4290048 ]
 [2.47940793 2.3768091 2.78571804]]
[1000 rows x 3 columns]
print(data.shape) # (1000, 3)
```

所有的数据都被缩放到了 $[2, 3]$ 之间（没有动 `target`！）。

问题：如果数据中异常点较多，会有什么影响？



回答：归一化处理可以将数据转换到一个特定的范围内，这样可以减少异常点对模型的影响。但是，如果数据中存在异常点，那么归一化处理可能会受到影响。

例如，最大最小归一化（Min-Max Normalization）会将数据转换到 $[0, 1]$ 的范围内，但是如果数据中存在异常点，那么最大值和最小值可能会不稳定，导致归一化结果不稳定。

因此，在进行归一化处理之前，通常需要先对数据进行清洗，去除异常点。

注意：最大值最小值是变化的；另外，最大值与最小值非常容易受异常点影响，所以这种方法鲁棒性较差，只适合传统精确小数据场景。

为了解决这个问题，引入“标准化”。

7.5 标准化 (Standardization)

7.5.1 标准化的定义

标准化 (Standardization) 是一种数据预处理方法，它通过将数据转换为均值为 0 ($\mu = 0$)，标准差为 1 ($\sigma = 1$) 的正态分布来消除数据的量纲和尺度差异。标准化的公式为：

$$x' = \frac{x - \mu}{\sigma}$$

其中 x 表示原始数据， μ 表示数据的均值， σ 表示数据的标准差， x' 表示标准化后的数据。

标准差为 1 和方差为 1 是一回事。

【补充】均值、标准差以及方差的计算公式

均值 (Mean) 是一组数据的算术平均值，它表示数据的中心位置。均值的计算公式为：

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i$$

其中 n 表示数据的个数， x_i 表示第 i 个数据。

方差 (Variance) 是用来衡量一组数据的离散程度的统计量。它表示数据与均值之间的平均差异。方差的计算公式为：

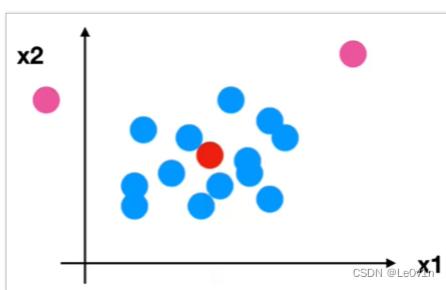
$$\sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2$$

其中 n 表示数据的个数， x_i 表示第 i 个数据， μ 表示数据的均值。

标准差 (Standard Deviation) 是方差的平方根，它也用来衡量一组数据的离散程度。标准差的计算公式为：

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2}$$

回到刚才异常点的地方，我们再来看看标准化：



- 对于归一化来说：如果出现异常点，影响了最大值和最小值，那么结果显然会发生改变
- 对于标准化来说：如果出现异常点，由于样本具有一定数据量（样本数量多），少量的异常点对于平均值的影响并不大，从而方差改变较小（方差和标准差是根据均值计算得到的）。

7.5.2 标准化的 API

```
sklearn.preprocessing.StandardScaler(copy=True, with_mean=True, with_std=True)
```

`sklearn.preprocessing.StandardScaler` 是一个类，它用来对数据进行标准化处理。它的构造函数接受以下参数：

- `copy`：布尔值，表示是否复制数据，默认为 `True`。
- `with_mean`：布尔值，表示是否去除均值 ($-\mu$)，默认为 `True`。
- `with_std`：布尔值，表示是否除以标准差 ($\frac{1}{\sigma}$)，默认为 `True`。

该类的主要方法有：

- `fit(X[, y])`：计算数据的均值和标准差。
- `fit_transform(X[, y])`：计算数据的均值和标准差，并对数据进行标准化处理。
- `transform(X)`：使用之前计算（`fit`）的均值和标准差对数据进行标准化处理。

这些方法都返回一个二维数组，表示标准化后的数据。

7.5.3 标准化的数据计算

同样对上面的数据进行处理

1. 实例化 `StandardScaler`

2. 通过 `fit_transform` 进行学习和转换

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
```

```

data = pd.read_csv(filepath_or_buffer="..../data/dating.txt", delimiter="\t")
print(data)
"""
    milage    Liters  Consumtime  target
0   40920    8.326976   0.953952      3
1   14488    7.153469   1.673904      2
2   26052    1.441871   0.805124      1
3   75136   13.147394   0.428964      1
4   38344    1.669788   0.134296      1
...
...
995  11145    3.410627   0.631838      2
996  68846    9.974715   0.669787      1
997  26575   10.650102   0.866627      3
998  48111    9.134528   0.728045      3
999  43757    7.882601   1.332446      3
"""
[1000 rows x 4 columns]

# 1. 实例化一个转换器类
transformer = StandardScaler()

# 2. 调用fit_transform方法
data = transformer.fit_transform(X=data.loc[:, ["milage", "Liters", "Consumtime"]])
print("标准化的结果: \r\n", data)
print("每一列特征的平均值: ", transformer.mean_)
print("每一列特征的方差: ", transformer.var_)

"""
标准化的结果:
[[ 0.33193158  0.41660188  0.24523407]
 [-0.87247784  0.13992897  1.69385734]
 [-0.34554872 -1.20667094 -0.05422437]
 ...
 [-0.32171752  0.96431572  0.06952649]
 [ 0.65959911  0.60699509 -0.20931587]
 [ 0.46120328  0.31183342  1.00680598]]
每一列特征的平均值: [ 3.36354210e+04  6.55996083e+00  8.32072997e-01]
每一列特征的方差: [ 4.81628039e+08  1.79902874e+01  2.46999554e-01]
"""

```

标准化在已有样本足够多的情况下比较稳定，适合现代嘈杂大数据场景。

小结：

- 什么是特征工程【知道】

- 定义：通过一些转化函数将特征数据转换成更加适合算法模型的特征数据过程。
- 包含内容：
 - 归一化
 - 标准化

- 归一化【知道】

- 定义：对原始数据进行变换把数据映射到指定区间（默认为 [0, 1]）。
- API：

- `sklearn.preprocessing.MinMaxScaler(feature_range=(0, 1), *, copy=True, clip=False)`
- 参数：
 - `feature_range`：元组（默认为 (0, 1)）。期望的缩放范围。所有特征都将被缩放到这个范围内。
 - `copy`：布尔值（默认为 `True`）。如果为 `True`，则对数据进行复制，否则进行原地缩放。
 - `clip`：布尔值（默认为 `False`）。如果为 `True`，则在转换过程中将数据裁剪到指定范围内。

- 总结：

- 鲁棒性比较差（容易受到异常点的影响）
- 只适合传统精确小数据场景（以后不会用你了）

- 标准化【掌握】

- 定义：对原始数据进行变换，把数据变换到均值为 0，标准差为 1 的范围（符合正态分布）
- api：

- `sklearn.preprocessing.StandardScaler(copy=True, with_mean=True, with_std=True)`
- 参数：
 - `copy`：布尔值，表示是否复制数据，默认为 `True`。
 - `with_mean`：布尔值，表示是否去除均值，默认为 `True`。
 - `with_std`：布尔值，表示是否除以标准差，默认为 `True`。

- 总结：

- 异常值对其影响小
- 适合现代嘈杂大数据场景（以后就是用你了）

归一化（Normalization）和标准化（Standardization）都是数据预处理方法，它们都可以消除数据的量纲和尺度差异。但是，它们并没有绝对的优劣之分，它们在不同的情况下都有各自的优点。

- 归一化通过将数据转换到一个特定的范围内来消除数据的量纲和尺度差异。它适用于数据分布比较均匀，且没有异常点的情况。

- 标准化通过将数据转换为均值为 0，标准差为 1 的正态分布来消除数据的量纲和尺度差异。它适用于数据分布不均匀，且存在异常点的情况。

在实际应用中，可以根据数据的特点选择合适的方法。

8. 练习案例：鸢尾花种类预测

学习目标：

- 知道 `KNeighborsClassifier` 的用法

8.1 再识 K - 近邻算法 API

```
sklearn.neighbors.KNeighborsClassifier(n_neighbors=5,
                                       weights='uniform',
                                       algorithm='auto', leaf_size=30,
                                       p=2, metric='minkowski',
                                       metric_params=None, n_jobs=None
                                       ) -> None
```

`sklearn.neighbors.KNeighborsClassifier()` 是一个实现了 K -近邻投票的分类器。它有许多参数，包括：

- `n_neighbors`：默认为 5，用于 K -近邻查询的邻居数量
- `weights`：默认为 ‘uniform’，用于预测的权重函数
- `algorithm`：默认为 ‘auto’，用于计算最近邻居的算法
 - “brute”：蛮力搜索，也就是线性扫描，当训练集很大时，计算非常耗时。
 - “kd_tree”：构造 kd 树存储数据以便对其进行快速检索的树形数据结构，kd 树也就是数据结构中的二叉树。以中值切分构造的树，每个结点是一个超矩形，在维数小于 20 时效率高。
 - “ball_tree”：为了克服 kd 树高维失效而发明的，其构造过程是以质心 C 和半径 r 分割样本空间，每个节点是一个超球体。
- `leaf_size`：默认为 30，传递给 BallTree 或 KDTree 的叶大小
- `p`：默认为 2，Minkowski 度量的幂参数
- `metric`：默认为‘ minkowski’，用于距离计算的度量等等。

它没有返回值，但是它有一些属性，如 `classes_`（已知分类器的类标签数组）和 `effective_metric_`（使用的距离度量）。

8.2 鸢尾花种类预测

8.2.1 步骤分析

1. 获取数据集
2. 数据基本处理
3. 特征工程
 1. 训练集：`transformer.fit_transform(x_train)`
 2. 测试集：`transformer.transform(x_test)` 千万不要 `fit_transform`
4. 机器学习（模型训练）
5. 模型评估

8.2.2 代码实现

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

# 1. 获取数据集
iris = load_iris()

# 2. 数据基本处理
x_train, x_test, y_train, y_test = train_test_split(iris.data,
                                                    iris.target,
                                                    test_size=0.2,
                                                    random_state=22)

# 3. 特征工程：标准化
transformer = StandardScaler()
x_train = transformer.fit_transform(x_train)
x_test = transformer.transform(x_test) # 测试数据处理时应该使用transform而不是fit_transform!

# 4. 机器学习（模型训练）
estimator = KNeighborsClassifier(n_neighbors=5)
estimator.fit(x=x_train, y=y_train)

# 5. 模型评估
# 方法1：对比真实值和预测值
y_predict = estimator.predict(x=x_test)
print("预测结果为:\r\n", y_predict)
print("对比真实值和预测值:\r\n", y_predict == y_test)

# 方法2：直接计算准确率（这里是预测+计算分数）
score = estimator.score(x=x_test, y=y_test)
print("准确率为: {:.2f}%".format(score * 100))
```

结果：

```
预测结果为:  
[0 2 1 2 1 1 1 0 2 1 2 2 0 2 1 1 1 1 0 2 0 1 2 0 2 2 2]  
对比真实值和预测值:  
[ True  True  True  True  True  True  True  False  True  True  True  True  
 True  True  True  True  True  False  True  True  True  True  True  True  
 True  True  True  True  True]  
准确率为: 93.33%
```

小结：

- KNeighborsClassifier 的使用【知道】
 - `sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, algorithm='auto')`
 - auto
 - brute
 - kd_tree
 - ball_tree

9. KNN 算法总结

9.1 优点

1. 简单有效：因为它基于一种直观的思想：如果一个样本在特征空间中的 K 个最近邻居中的大多数属于某一个类别，那么该样本也属于这个类别。这种方法简单易懂，且在许多实际应用中都取得了很好的效果。此外，KNN 算法不需要建立模型，只需要存储训练数据即可，这也使得它更加简单。
2. 重新训练的代价低：因为它不需要在训练阶段建立模型。它只是在分类时，根据测试样本的特征，从训练集中找到最近的 K 个邻居，然后根据这些邻居的类别来决定测试样本的类别。因此，当训练集发生变化时，KNN 算法不需要重新训练模型，只需要更新训练集即可。这就使得 KNN 算法重新训练的代价低。
3. 适合类域交叉样本：KNN 方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的。因此对于类域的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适合。
 1. 例如，如果一个样本位于两个类域的交叉区域，那么它周围的邻居可能属于不同的类别。KNN 算法会根据这些邻居的投票结果来确定该样本的类别，从而更好地处理类域交叉的情况。
4. 适合大样本自动分类：KNN 算法适合大样本自动分类，因为它能够处理大量的训练数据。当样本容量较大时，KNN 算法能够更好地捕捉数据的分布特征，从而更准确地进行分类。但是，当样本容量较小时，KNN 算法可能会产生误分。这是因为当样本容量较小时，邻居的数量也会减少，这会影响投票结果的准确性。因此，KNN 算法比较适用于样本容量比较大的类域的自动分类。

9.2 缺点

1. 惰性学习：KNN 算法是一种惰性学习方法（Lazy Learning，基本上不学习），相比一些积极学习的算法要慢很多。因为它不需要在训练阶段建立模型。它只是在分类时，根据测试样本的特征，从训练集中找到最近的 K 个邻居，然后根据这些邻居的类别来决定测试样本的类别。因此，KNN 算法不需要在训练阶段进行大量的计算，而是将计算推迟到分类阶段。这就使得 KNN 算法成为一种惰性学习算法。
2. 类别评分未规范化：KNN 算法的类别评分未规范化，是指它没有像一些通过概率评分的分类器那样，给出每个类别的概率估计。KNN 算法只是根据测试样本的 K 个最近邻居的类别来决定测试样本的类别，而没有给出每个类别的概率估计。这就使得 KNN 算法的类别评分未规范化。
3. 输出的可解释性不强：KNN 算法的输出可解释性不强，是指它没有像一些其他算法（如决策树）那样，给出清晰的分类规则。KNN 算法只是根据测试样本的 K 个最近邻居的类别来决定测试样本的类别，而没有给出清晰的分类规则。这就使得 KNN 算法的输出可解释性不强，而决策树的输出可解释性就较强。
4. 不擅长解决不均衡的样本：KNN 算法在处理不均衡样本时可能会遇到一些问题。当样本不平衡时，训练的模型有可能更偏向于其中样本数量较多的类别。为了解决这个问题，我们可以剔除一些数样本，也可以生成一些样本，让各种类别的样本保持平衡，而通过 KNN 剔除冗余样本的方式，也可以得到相对平衡的训练样本。
5. 计算量较大：KNN 算法的计算量较大，主要是因为它需要计算输入实例与所有训练数据的距离，然后再进行判断。当训练集或特征维度很大时，这种计算非常耗时。此外，KNN 的数据存储量也很大，因为每次对一个未标记样本进行分类时，都需要全部计算一遍距离。

10. 交叉验证、网格搜索

学习目标：

- 知道交叉验证、网格搜索的概念
- 会使用交叉验证、网格搜索优化训练模型

10.1 什么是交叉验证 (Cross Validation, CV)

交叉验证 (Cross Validation, CV) 是一种用来观察模型稳定性的方法。它通过将数据划分为 n 份，依次使用其中一份作为测试集，其他 $n - 1$ 份作为训练集，多次计算模型的精确性来评估模型的平均准确程度。

以下图为例：将数据分成 4 份，其中一份作为测试集。然后经过 4 次（组）的测试，每次都更换不同的测试集。即得到 4 组模型的结果，取平均值作为最终结果。又称 4 折交叉验证。

10.1.1 分析

我们之前知道数据分为训练集和测试集，但是为了让从训练得到模型结果更加准确。做以下处理：

组	1	2	3	4	准确率
第一组	测试集	训练集	训练集	训练集	80%
第二组	训练集	测试集	训练集	训练集	78%
第三组	训练集	训练集	测试集	训练集	75%
第四组	训练集	训练集	训练集	测试集	82%

那么模型的准确率（测试集准确率）为：

$$\begin{aligned} \text{Acc}_{\text{train}} &= \frac{1}{k} \sum_{k=1}^4 \text{acc}_k \\ &= \frac{1}{4} (80\% + 78\% + 75\% + 82\%) \\ &= 78.75\% \end{aligned}$$

$$\text{Acctrain} = k \sum_{k=1}^4 \text{acck} = 4(80\% + 78\% + 75\% + 82\%) = 78.75\%$$

10.1.2 为什么需要交叉验证

交叉验证目的：为了让被评估的模型更加准确可信。

Q: 交叉验证可以提高准确率吗?

A: 交叉验证可以提高模型的准确率。交叉验证是一种评估机器学习模型性能的方法，它将数据集划分为训练集和测试集，并多次重复这个过程，每次使用不同的子集作为测试集和训练集。通过这样做，我们可以更好地评估模型在不同数据集上的表现，并减少因数据集选择引起的偏差。

交叉验证的主要优点是可以提供更准确的模型评估，因为它可以对模型进行更全面的测试，包括评估模型的泛化能力。泛化能力是指模型在新数据上的性能，而不仅仅是适合于用于拟合训练数据的模型。

通过使用交叉验证，我们可以更好地评估模型对未见过数据的性能，从而提高模型的准确率。

总之，交叉验证是一种有用的技术，可以帮助我们在开发机器学习模型时更准确地了解其性能和泛化能力。

Q：交叉验证只是让被评估的模型更加准确可信，那么怎么选择或者调优参数呢？

A：利用网络搜索优化模型参数，从而提高模型的准确率

10.2 什么是网格搜索 (Grid Search)

网格搜索（Grid Search）是一种调参手段，它通过穷举搜索，在所有候选的参数选择中，通过循环遍历，尝试每一种可能性。表现好的参数就是你的结果。它的原理就像是在数组里找最大值。

例如，以有两个参数的模型为例，参数 a 有 3 种可能，参数 b 有 4 种可能，把所有可能性列出来，可以表示成一个 3×4 的表格。其中每个 cell 就是一个网格。循环过程就像是在每个网格里遍历、搜索，所以叫 Grid Search。

通常情况下，有很多参数是需要手动指定的（如 K - 近邻算法中的 K 值），这种叫超参数。但是手动过程繁杂，所以需要对模型训练出多种超参数组合，每组超参数都采用交叉验证来进行评估，最后选出最优参数组合建立模型。

在机器学习中，需要手动指定的参数都是超参数

网格搜索的目的是选择最优的超参数

K 值	$K = 3$	$K = 5$	$K = 7$
模型名称	模型 1	模型 2	模型 3

10.3 交叉验证、网格搜索（模型选择与调优）的 API

作用：对估计器的指定参数值进行详尽搜索。

参数：

- `estimator` : 估计器对象（模型对象）。
- `param_grid` : 字典或字典列表，用于指定要搜索的参数值。例如：`param_dict = {"n_neighbors": [1, 3, 5]}`
- `scoring` : 用于评估预测的字符串、可调用对象或无。
- `n_jobs` : 并行运行的 CPU 核心数。
- `cv` : 确定交叉验证拆分策略的整数、可调用对象或无。

`cv = cross validation`

方法：

- `fit` : 在指定参数值范围内对估计器进行穷举搜索。
- `score` : 返回给定测试数据和标签的平均精度。
- `score_samples` : 返回给定测试数据和标签的分数。
- `predict` : 预测给定测试数据的类别。
- `predict_proba` : 预测给定测试数据属于每个类别的概率。
- `decision_function` : 计算给定测试数据的决策函数值。
- `transform` : 在拟合数据上应用转换。
- `inverse_transform` : 撤消先前转换的所有转换。

这些方法是否实现取决于所使用的估计器。

属性（结果分析）：

- `best_score_` : 在交叉验证期间找到的最佳分数（最好的结果）。
- `best_estimator_` : 在交叉验证期间找到的最佳估计器（最好的参数模型）。
- `cv_results_` : 一个字典，其中包含交叉验证期间所有试验的结果（每次交叉验证后的测试集准确率结果和训练集准确率结果）。

10.4 鸢尾花案例增加 K 值调优

第一步：使用 GridSearchCV 构建估计器

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

# 1. 获取数据
iris = load_iris()

# 2. 数据基本处理：划分数据集
x_train, x_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=22)

# 3. 特征工程：标准化
transformer = StandardScaler()
x_train = transformer.fit_transform(X=x_train)
x_test = transformer.transform(X=x_test)

# 4. KNN预估器流程
## 4.1 实例化预估器
estimator = KNeighborsClassifier()

## 4.2 模型选择与调优：网格搜索和交叉验证
param_dict = {"n_neighbors": [1, 3, 5]} # 准备调整的超参数
estimator = GridSearchCV(estimator=estimator, param_grid=param_dict, cv=3)

## 4.3 fit数据进行训练
estimator.fit(X=x_train, y=y_train)

## 5. 评估模型效果
## 5.1 方法一：比对预测值和真实值
y_predict = estimator.predict(x_test)
print("比对预测值和真实值:\r\n", y_predict == y_test)

## 5.2 方法二：直接计算准确率
score = estimator.score(X=x_test, y=y_test)
print(f"KNN准确率为: {score*100:.2f}%")
```

第二步：进行评估查看最终选择的结果和交叉验证的结果

```
print(f"在交叉验证中，最好的结果是: {estimator.best_score_*100:.2f}%")
print("最好的参数模型是: ", estimator.best_estimator_.get_params())
print("每次交叉验证后的准确率为: \r\n", estimator.cv_results_)
```

结果：

```
比对预测值和真实值：  
[ True True True True True True False True True True True  
True True True True True False True True True True True  
True True True True True True ]  
KNN准确率为：93.33%  
在交叉验证中，最好的结果是：97.50%  
最好的参数模型是：{'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 5, 'p':  
每次交叉验证后的准确率为：  
{'mean_fit_time': array([0.00033243, 0.00066495, 0.00066503]), 'std_fit_time': array([0.00047013, 0.00047019, 0.00047025]), 'mean_score_time':  
mask=[False, False, False],  
fill_value='?'},  
dtype=object), 'params': [{'n_neighbors': 1}, {'n_neighbors': 3}, {'n_neighbors': 5}], 'split0_test_score': array([1. , 0.975,
```

- “比对预测值和真实值” 显示了模型在测试集上的预测结果与真实结果的比较。
 - True 表示预测正确
 - False 表示预测错误。
- “KNN 准确率为: 93.33%” 显示了模型在测试集上的准确率。
- “在交叉验证中，最好的结果是：97.50%” 显示了在交叉验证过程中，最佳超参数组合下的平均准确率。
- “最好的参数模型是：...” 显示了最佳超参数组合。
- “每次交叉验证后的准确率为：...” 显示了交叉验证过程中每次迭代的详细信息，包括每个超参数组合下的平均拟合时间、平均评分时间、测试得分等。

小结：

- 交叉验证【知道】
 - 定义：
 - 将拿到的训练数据，分为训练集和测试集
 - k 折交叉验证（用不同的数据集训练几次模型）
 - 分割方式：
 - 训练集：训练集 + 验证集
 - 测试集：测试集
 - 为什么需要交叉验证：为了让被评估的模型更加准确可信
- 网格搜索【知道】
 - 超参数：在 sklearn 中，需要手动指定的参数，叫做超参数
 - 网格搜索就是把这些超参数的值，通过字典的形式传递进去，然后进行选择最优值
- API【知道】
 - sklearn.model_selection.GridSearchCV(estimator, param_grid=None, cv=None)
 - estimator：选择了哪个训练模型
 - param_grid：需要传递的超参数
 - cv：几折交叉验证（用不同的数据集训练几次模型）

11. 练习案例：预测 Facebook 签到位置

学习目标：

- 通过 Facebook 位置预测案例熟练掌握 KNN 的学习内容。

11.1 项目描述



本次比赛的目的是预测一个人将要签到的地方。为了本次比赛，Facebook 创建了一个虚拟世界，其中包括 10 公里 * 10 公里共 100 平方公里，约 10 万个地方。对于给定的坐标集，您的任务是：根据用户的位置，准确性和时间戳等预测用户下一次的签到位置。数据被制作成类似于来自移动设备的位置数据。请注意：只能使用提供的数据进行预测。

11.2 数据集介绍

数据集下载地址：[Facebook V: Predicting Check Ins](#)

row_id	x	y	accuracy	time	place_id
0	0	0.7941	9.0809	54	470702 8523065625
1	1	5.9567	4.7968	13	186555 1757726713
2	2	8.3078	7.0407	74	322648 1137537235
3	3	7.3665	2.5165	65	704587 6567393236
4	4	4.0961	1.1307	31	472130 7440663949

CSDN @Leovin

文件说明: train.csv, test.csv

row id: 签入事件的id
x y: 坐标
accuracy: 准确度, 定位精度
time: 时间戳
place_id: 签到的位置, 这也是你需要预测的内容

	row_id	x	y	accuracy	time	place_id
count	2.911802e+07	2.911802e+07	2.911802e+07	2.911802e+07	2.911802e+07	2.911802e+07
mean	1.455901e+07	4.999770e+00	5.001814e+00	8.284912e+01	4.170104e+05	5.493787e+09
std	8.405649e+06	2.857601e+00	2.887505e+00	1.147518e+02	2.311761e+05	2.611088e+09
min	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	1.000016e+09
25%	7.279505e+06	2.534700e+00	2.496700e+00	2.700000e+01	2.030570e+05	3.222911e+09
50%	1.455901e+07	5.009100e+00	4.988300e+00	6.200000e+01	4.339220e+05	5.518573e+09
75%	2.183852e+07	7.461400e+00	7.510300e+00	7.500000e+01	6.204910e+05	7.764307e+09
max	2.911802e+07	1.000000e+01	1.000000e+01	1.033000e+03	7.862390e+05	9.999932e+09

11.3 步骤分析

- 数据基本处理 (这里所做的一些处理不一定达到很好的效果, 我们只是简单尝试, 有些特征我们可以根据一些特征选择的方式去做处理)
- 缩小数据集范围 (目的是让训练速度变快, 实际中不需要这么做) —— DataFrame.query()
- 选取有用的时间特征
- 将签到位置少于 n 个用户的删除
- 分割数据集
- 标准化处理
- k - 近邻预测

具体步骤:

```
# 1. 获取数据集
# 2. 基本数据处理
## 2.1 缩小数据范围
## 2.2 选择时间特征
## 2.3 去掉签到较少的地方
## 2.4 确定特征值和目标值
## 2.5 分割数据集

# 3. 特征工程 —— 特征预处理 (标准化)
## 3.1 实例化一个转换器
## 3.2 调用fit_transform和transform

# 4. 机器学习 —— KNN + CV (交叉验证Cross Validation)
## 4.1 实例化一个估计器
## 4.2 调用GridSearchCV
## 4.3 训练模型

# 5. 模型评估
```

11.4 代码实现

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

# 1. 获取数据集
facebook = pd.read_csv(filepath_or_buffer='../data/FacebookLocation/train.csv')
print(facebook.head())
```
row_id x y accuracy time place_id
0 0 0.7941 9.0809 54 470702 8523065625
1 1 5.9567 4.7968 13 186555 1757726713
2 2 8.3078 7.0407 74 322648 1137537235
```
```

```

3      3  7.3665  2.5165      65 704587 6567393236
4      4  4.0961  1.1307      31 472130 7440663949
"""

# 2. 基本数据处理

## 2.1 缩小数据范围
partial_data = facebook.query("x > 2.0 & x < 2.5 & y > 2.0 & y < 2.5").copy()

## 2.2 选择时间特征
# 使用to_datetime函数将DF中的time列转换为日期时间格式。其中，unit='s'表示时间单位为秒
time = pd.to_datetime(partial_data["time"], unit="s")
print(time.head())
"""

163    1970-01-08 18:02:17
310    1970-01-03 17:11:59
658    1970-01-06 19:32:23
1368   1970-01-04 16:50:22
1627   1970-01-07 21:18:04
Name: time, dtype: datetime64[ns]
"""

# 将转换后的日期时间数据存储在一个新的数据框time中
time = pd.DataFrame(time)

# 从time数据框中提取出日期、小时和星期几的信息，并分别存储在facebook_data数据框的新列day、hour和weekday中
# .dt是一个访问器，它用于访问时间日期序列的属性和方法。它类似于.str访问器
partial_data.loc[:, "day"] = time['time'].dt.day
partial_data.loc[:, "hour"] = time['time'].dt.hour
partial_data.loc[:, "weekday"] = time['time'].dt.weekday

print(partial_data.head())
"""

   row_id     x     y  accuracy    time  place_id  day  hour  weekday
163    163  2.1663  2.3755      84 669737 3869813743    8   18       3
310    310  2.3695  2.2034      3 234719 2636621520    3   17       5
658    658  2.3236  2.1768     66 502343 7877745055    6   19       1
1368   1368  2.2613  2.3392     73 319822 9775192577    4   16       6
1627   1627  2.3331  2.0011     66 595084 6731326909    7   21       2
"""

## 2.3 去掉签到较少的地方
# 使用groupby和count函数统计facebook_data数据框中每个地点的出现次数，并将结果存储在place_count数据框中
place_count = partial_data.groupby(by="place_id").count()
print(place_count.head())
"""

   place_id  row_id  x     y  accuracy    time  day  hour  weekday
1006234733      1   1     1      1   1     1      1       1
1008823061      4   4     4      4   4     4      4       4
1012580558      3   3     3      3   3     3      3       3
1025585791     21  21    21     21  21    21     21      21
1026507711     220 220   220    220  220   220    220      220
"""

# 筛选出place_count数据框中出现次数大于3的地方
place_count = place_count[place_count["row_id"] > 3]
print(place_count["row_id"] > 3)
"""

place_id
1006234733    False
1008823061    True
1012580558    False
1025585791    True
1026507711    True

Name: row_id, Length: 2524, dtype: bool
"""

print(place_count.head())
"""

   place_id  row_id  x     y  accuracy    time  day  hour  weekday
1008823061      4   4     4      4   4     4      4       4
1025585791     21  21    21     21  21    21     21      21
1026507711     220 220   220    220  220   220    220      220
1032417180     10  10    10     10  10    10     10      10
1040557418     123 123   123    123  123   123    123      123
"""

# 使用isin函数只保留facebook_data数据框中属于筛选后的地点的数据
partial_data = partial_data[partial_data["place_id"].isin(place_count.index)]
print(place_count.index)
"""

Int64Index([1008823061, 1025585791, 1026507711, 1032417180, 1040557418,
           1067960232, 1068428112, 1068896566, 1104074781, 1113141722,
           ...,
           9929803766, 9934025626, 9944591314, 9951996370, 9953487159,
           9966115681, 9970566102, 9983648790, 9995108787, 9998968845],
           dtype='int64', name='place_id', length=929)
"""

print(partial_data["place_id"].isin(place_count.index))
"""

163      True
310      True
658      True
1368     True
1627     True

Name: place_id, Length: 71664, dtype: bool
"""

print(partial_data.head())
"""

   place_id  row_id  x     y  accuracy    time  day  hour  weekday
163    163  2.1663  2.3755      84 669737 3869813743    8   18       3
310    310  2.3695  2.2034      3 234719 2636621520    3   17       5
658    658  2.3236  2.1768     66 502343 7877745055    6   19       1
1368   1368  2.2613  2.3392     73 319822 9775192577    4   16       6
1627   1627  2.3331  2.0011     66 595084 6731326909    7   21       2
"""

## 2.4 确定特征值和目标值
x = partial_data[['x', 'y', 'accuracy', 'day', 'hour', 'weekday']] # 两个中括号，或者使用.loc
y = partial_data["place_id"]

```

```

## 2.5 分割数据集
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=2, test_size=0.25)

print(x_train.shape) # (51948, 6)
print(x_test.shape) # (17316, 6)
print(y_train.shape) # (51948,)
print(y_test.shape) # (17316,)

# 3. 特征工程 —— 特征预处理 (标准化)

## 3.1 实例化一个转换器
transformer = StandardScaler()

## 3.2 调用fit_transform和transform
x_train = transformer.fit_transform(x_train)
x_test = transformer.transform(x_test)

# 4. 机器学习 —— KNN + CV (交叉验证Cross Validation)

## 4.1 实例化一个估计器
estimator = KNeighborsClassifier()

## 4.2 调用GridSearchCV
param_grid = {"n_neighbors": [3, 5, 7, 9]}
estimator = GridSearchCV(estimator=estimator, param_grid=param_grid, cv=3, n_jobs=-1)
# 因为cv=3, 即3折交叉验证, 而超参数n_neighbors有4种, 所以需要跑12次
# n_jobs=是需要调用几个CPU来运行, -1表示全部的CPU

## 4.3 训练模型
estimator.fit(x=x_train, y=y_train)

# 5. 模型评估

## 5.1 基本评估方式

### 5.1.1 方法1
score = estimator.score(x=x_test, y=y_test)
print(f"最后预测的准确率为: {score*100:.2f}%")

### 5.1.2 方法2
y_predict = estimator.predict(x=x_test)
print("最后的预测值为: ", y_predict)
print("预测值和真实值的对比情况: \r\n", y_predict == y_test)

## 5.2 使用CV (交叉验证) 后的评估方式
print("在交叉验证中, 最好的准确率为: {estimator.best_score_*100:.2f}%")
print("在交叉验证中, 最好的模型参数为: {estimator.best_estimator_.get_params()}")
print("在交叉验证中, 记录所有的结果为: \r\n", estimator.cv_results_)

```

结果：

```

最后预测的准确率为: 37.08%
最后的预测值为: [8283681418 8980163153 1247398579 ... 1891783132 8169595806 3661555534]
预测值和真实值的对比情况:
21112721 False
15305275 False
2286447 False
2683519 False
276963 True
...
5377682 False
5778143 True
10652670 True
5835344 True
12327543 False
Name: place_id, Length: 17316, dtype: bool
在交叉验证中, 最好的准确率为: 33.69%
在交叉验证中, 最好的模型参数为: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 3, 'weights': 'uniform'}
在交叉验证中, 记录所有的结果为:
{'mean_fit_time': array([0.07546322, 0.06881396, 0.07014338, 0.07014489]), 'std_fit_time': array([0.00517131, 0.00141153, 0.00169545, 0.00049441]), 'mean_score_time': array([0.00016667, 0.00016667, 0.00016667, 0.00016667]), 'std_score_time': array([0.00016667, 0.00016667, 0.00016667, 0.00016667]), 'mean_test_score': array([0.3369, 0.3369, 0.3369, 0.3369]), 'std_test_score': array([0.0111, 0.0111, 0.0111, 0.0111]), 'params': [{n_neighbors: 3}, {n_neighbors: 5}, {n_neighbors: 7}, {n_neighbors: 9}], 'split0_test_score': array([0.3369, 0.3369, 0.3369, 0.3369])}

```

这里的效果不好是因为我们截取了很多数据，所以数据量不够多，模型欠拟合！

12. 【知识补充】数据分割

前面已经讲过，我们可通过实验测试来对学习器的泛化误差进行评估并进而做出选择。为此，需使用一个“测试集”(testing set)来测试学习器对新样本的判别能力，然后以测试集上的“测试误差”(testing error)作为泛化误差的近似。

通常我们假设测试样本也是从样本真实分布中独立同分布采样而得。但需注意的是，测试集应该尽可能与训练集互斥。

互斥：即测试样本尽量不在训练集中出现、未在训练过程中使用过。

测试样本为什么要尽可能不出现在训练集中呢？为理解这一点，不妨考虑这样一个场景：

老师出了 10 道习题供同学们练习，考试时老师又用同样的这 10 道题作为试题，这个考试成绩能否有效反映出同学们学得好不好呢？

答案是否定的，可能有的同学只会做这 10 道题却能得高分。

回到我们的问题上来，我们希望得到泛化性能强的模型，好比是希望同学们对课程学得很好、获得了对所学知识“举一反三”的能力；训练样本相当于给同学们练习的习题，测试过程则相当于考试。显然，若测试样本被用作训练了，则得到的将是过于“乐观”的估计结果。

可是，如果我们只有一个包含 m 个样例的数据集 $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ 。这个数据集既要训练，又要测试，怎样才能做到呢？

答案是：通过对 D 进行适当的处理，从中产生出训练集 S 和测试集 T （这个也是我们前面一直在做的事情）。

下面我们一起总结一下几种常见的做法：

1. 留出法
2. 交叉验证法
3. 自助法

12.1 留出法 (Hold-out Method)

12.1.1 留出法的原理

“留出法”（Hold-out）直接将数据集 D 划分为两个互斥的集合，其中一个集合作为训练集 S ，另一个作为测试集 T ，即 $D = S \cup T$, $S \cap T = \emptyset$ 。在 S 上训练出模型后，用 T 来评估其测试误差，作为对泛化误差的估计。

之前我们使用的划分方法就是留出法😊

大家在使用的过程中，需注意的是，训练 / 测试集的划分要尽可能保持数据分布的一致性，避免因数据划分过程引入额外的偏差而对最终结果产生影响，例如在分类任务中至少要保持样本的类别比例相似。

如果从采样（Sampling）的角度来看待数据集的划分过程，则保留类别比例的采样方式通常称为“分层采样”（Stratified Sampling）。

例如通过对 D 进行分层样而获得含 70% 样本的训练集 S 和含 30% 样本的测试集 T 。

若 D 包含 500 个正例、500 个反例，则分层采样得到的 S 应包含 350 个正例、350 个反例，而 T 则包含 150 个正例和 150 个反例。——训练集和测试集的正例：反例得到的比例应该相同！

$$\frac{\text{正例}}{\text{反例}}_{\text{训练集}} = \frac{\text{正例}}{\text{反例}}_{\text{测试集}}$$

若 S 、 T 中样本类别比例差别很大，则误差估计将由于训练 / 测试数据分布的差异而产生偏差。

另一个需注意的问题是，即便在给定训练测试集的样本比例后，仍存在多种划分方式对初始数据集 D 进行分割。

例如在上面的例子中，可以把 D 中的样本排序，然后把前 350 个正例放到训练集中，也可以把最后 350 个正例放到训练集中，这些不同的划分将导致不同的训练 / 测试集。相应的，模型评估的结果也会有差别。

因此，单次使用留出法得到的估计结果往往不够稳定可靠。在使用留出法时，一般要采用若干次随机划分、重复进行实验评估后取平均值作为留出法的评估结果。

例如进行 100 次随机划分，每次产生一个训练 / 测试集用于实验评估。100 次后就得到 100 个结果，而留出法返回的则是这 100 个结果的平均。

$$acc_{test} = \frac{1}{n} \sum_{i=1}^n acc_i$$

其中， i 是每次实验的 ID。

12.1.2 留出法的问题

此外，我们希望评估的是用 D 训练出的模型的性能，但留出法需划分训练 / 测试集，这就会导致一个窘境：

- 若令训练集 S 包含绝大多数样本，则训练出的模型可能更接近于用 D 训练出的模型，由于 T 比较小，评估结果可能不够稳定准确
- 若令测试集 T 多包含一些样本，则训练集 S 与 D 差别大了，被评估的模型与用 D 训练出的模型相比可能有较大差别，从而降低了评估结果的保真性（fidelity）。

fidelity：英 [fi'deləti] 美 [fi'deləti]

n. 忠诚；忠实；（对丈夫、妻子或性伴侣的）忠贞；准确性；精确性；

这个问题没有完美的解决方案，常见做法是将大约 $\frac{2}{3} \sim \frac{4}{5}$ 的样本用于训练，剩余样本用于测试。

12.1.3 留出法的实现

使用 Python 实现留出法：

```

from sklearn.model_selection import train_test_split

data = ["数据1", "数据2", "数据3", "数据4", "数据5", "数据6", "数据7", "数据8", "数据9", "数据10"]
target = ["标签1", "标签2", "标签3", "标签4", "标签5", "标签6", "标签7", "标签8", "标签9", "标签10"]

# 使用train_test_split划分训练集和测试集
x_train, x_test, y_train, y_test = train_test_split(data, target, test_size=0.3, random_state=0)

print("训练集: ", x_train, y_train)
print("测试集: ", x_test, y_test)

"""
训练集: ['数据1', '数据2', '数据3', '数据4', '数据5', '数据6', '数据7', '数据8', '数据9', '数据10']
测试集: ['数据3', '数据9', '数据5'] ['标签3', '标签9', '标签5']
"""

```

12.1.4 留出法的特例：留一法（Leave-One-Out, LOO）

在留出法中，有一个特例，叫做留一法（Leave-One-Out，简称 LOO）。即每次抽取一个样本作为测试集。显然，留一法不受随机样本划分方式的影响，因为 m 个样本只有唯一的方式划分为 m 个子集——每个子集包含一个样本。

简而言之，留一法就是每次训练时，都只留下一个样本作为测试集，其余的样本都用来训练模型。

使用 Python 实现留一法：

`LeaveOneOut()` 是 `sklearn.model_selection` 模块中的一个函数，它提供了训练 / 测试索引来将数据分成训练集和测试集。

注意：`LeaveOneOut` 必须实例化对象后才能使用 `.split()` 方法。

`LeaveOneOut.split()` 方法用于将数据分成训练集和测试集。它接受一个参数 `x`，表示要分割的数据。它返回一个生成器，可以用来迭代训练集和测试集的索引。

```

from sklearn.model_selection import LeaveOneOut

data = ["数据1", "数据2", "数据3", "数据4", "数据5", "数据6", "数据7", "数据8", "数据9", "数据10"]
target = ["标签1", "标签2", "标签3", "标签4", "标签5", "标签6", "标签7", "标签8", "标签9", "标签10"]

# 使用LeaveOneOut划分训练集，必须实例化LeaveOneOut对象
loo = LeaveOneOut()
res = loo.split(X=data, y=target) # y=target可写可不写

for train_idx, test_idx in res:
    print("训练集: ", train_idx)
    print("测试集: ", test_idx)

"""
结果
训练集: [1 2 3 4 5 6 7 8 9]
测试集: [0]
训练集: [0 2 3 4 5 6 7 8 9]
测试集: [1]
训练集: [0 1 3 4 5 6 7 8 9]
测试集: [2]
训练集: [0 1 2 4 5 6 7 8 9]
测试集: [3]
训练集: [0 1 2 3 5 6 7 8 9]
测试集: [4]
训练集: [0 1 2 3 4 6 7 8 9]
测试集: [5]
训练集: [0 1 2 3 4 5 7 8 9]
测试集: [6]
训练集: [0 1 2 3 4 5 6 8 9]
测试集: [7]
训练集: [0 1 2 3 4 5 6 7 9]
测试集: [8]
训练集: [0 1 2 3 4 5 6 7 8]
测试集: [9]
"""

```

Q：为什么 `LeaveOneOut.split()` 方法会返回好几组数据？

A：`LeaveOneOut.split()` 方法返回一个生成器，它会生成多组数据，每组数据都包含一组训练集索引和一组测试集索引。这是因为 `LeaveOneOut` 是一种特殊的交叉验证方法，它会将数据集分成多个子集，每次留下一个子集作为测试集，其余的子集作为训练集。因此，它会返回多组数据，每组数据都对应一种不同的训练集和测试集的划分方式。

留一法的优缺点：

- 优点：
 - 留一法使用的训练集与初始数据集相比只少了一个样本，这就使得在绝大多数情况下，被实际评估的模型与期望评估的用 D 训练出的模型很相似（数据尽可能用来训练而不是保留一部分用来测试）。因此，留一法的评估结果往往被认为比较准确。
- 缺点：
 - 在数据集比较大时，训练 m 个模型的计算开销可能是难以忍受的（例如数据集包含一百万个样本，则需训练一百万个模型，而这还是在未考虑算法调参的情况下）。

留一法适合小数据集的场景。

12.2 交叉验证法（Cross Validation, CV）

“交叉验证法”（Cross Validation, CV）先将数据集 D 划分为 k 个大小相似的互斥的子集，即：

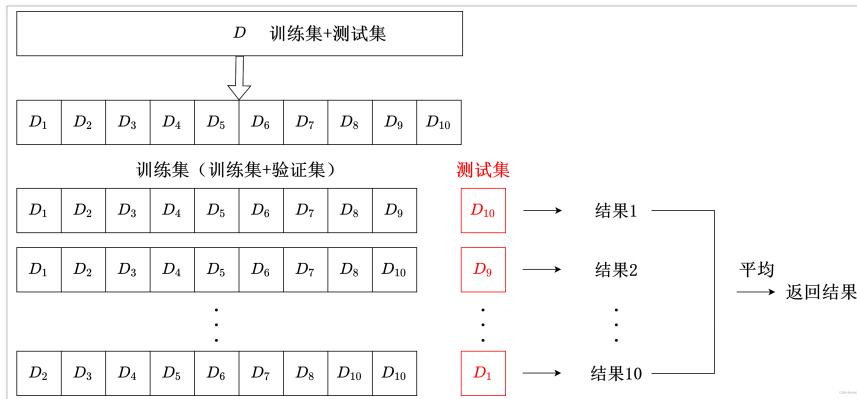
$$D = D_1 \cup D_2 \cup \dots \cup D_k$$

其中, $D_i \cap D_j = \emptyset (i \neq j)$

每个子集 D_i 都尽可能保持数据分布的一致性, 即从 D 中通过分层抽样得到。

然后, 每次用 $k - 1$ 个子集的并集作为训练集, 余下的那个子集作为测试集; 这样就可获得 k 组训练集 / 测试集, 从而可进行 k 次训练和测试, 最终返回的是这 k 个测试结果的均值。

显然, 交叉验证法评估结果的稳定性和保真性在很大程度上取决于 k 的取值, 为强调这一点, 通常把交叉验证法称为 “ k 折交叉验证” (k -fold cross validation, k -CV)。 k 最常用的取值是 10, 此时称为 10 折交叉验证; 其他常用的 k 值有 5、20 等。下图给出了 10 折交叉验证的示意图。



与留出法相似, 将数据集 D 划分为 k 个子集同样存在多种划分方式。为减小因样本划分不同而引入的差别, k -CV 通常要随机使用不同的划分重复 p 次, 最终的评估结果是这 p 次 K-CV 结果的均值, 例如常见的有 “10 次 10 折交叉验证” —— $K\text{-CV}(p=10)$ 。

与留出法相似, 并不是一次得出结果, 而是多次得出结果。

交叉验证实现方法, 除了咱们前面讲的 `GridSearchCV` 之外, 还有 `KFold`、`StratifiedKFold`。

`KFold` 和 `StratifiedKFold` 的代码实现

```
from sklearn.model_selection import KFold, StratifiedKFold
```

用法:

- 将训练 / 测试数据集划分 `n_splits` 个互斥的子集, 每次用其中一个子集当作测试集, 剩下的 `n_splits-1` 个作为训练集。进行 `n_splits` 次训练和测试, 得到 `n_splits` 个结果。
- `StratifiedKFold` 的用法和 `KFold` 的区别是: `StratifiedKFold` 是分层采样, 确保训练集 / 测试集中, 各类别样本的比例是和原始数据集中的一致。

Q: 什么是分层采样?

A: 分层抽样 (stratified sampling) 是一种概率抽样方法, 它将总体按照某些特征分成若干个互不相交的层, 然后从每一层中按照一定的比例随机抽取样本。这样可以保证每一层中的样本数量与总体中该层所占比例相同, 从而提高抽样的代表性。

Q: 举个例子来解释分层抽样。

A: 假设你想调查一个城市中人们对某个问题的看法, 你可以将这个城市的人口按照年龄分成若干个层, 比如 18-30 岁、31-40 岁、41-50 岁、51-60 岁和 60 岁以上。然后, 你可以根据每个层在总人口中所占的比例来确定每个层中应该抽取多少样本。例如, 如果 18-30 岁这个层占总人口的 20%, 那么在抽取 1000 个样本时, 就应该从这个层中抽取 200 个样本。这样, 你就能获得一个更具代表性的样本。

注意点:

- 对于不能均等分数据集, 其前 `n_samples % n_splits` 子集拥有 `n_samples // n_splits + 1` 个样本, 其余子集都只有 `n_samples // n_splits` 个样本。(前面几个多 1 个样本, 后面的少一个样本)

参数说明:

- `n_splits` : 表示划分几等份
- `shuffle` : 在每次划分时, 是否进行洗牌
 - 若为 `False` 时, 其效果等同于 `random_state` 等于整数, 每次划分的结果相同
 - 若为 `True` 时, 每次划分的结果都不一样, 表示经过洗牌, 随机取样的操作

属性:

- `split(X=None, y=None, groups=None)` : 将数据集划分成训练集和测试集, 返回索引生成器。

- 注意：KFold 和 StratifiedKFold 都必须实例化才能使用 .split() 方法！

之前我们使用的留出法 train_test_split() : 返回的不是索引生成器。它返回的是划分后的训练集和测试集数据。

举例：

```
import numpy as np
from sklearn.model_selection import KFold, StratifiedKFold

x = np.array([[1, 2, 3, 4],
              [11, 12, 13, 14],
              [21, 22, 23, 24],
              [31, 32, 33, 34],
              [41, 42, 43, 44],
              [51, 52, 53, 54],
              [61, 62, 63, 64],
              [71, 72, 73, 74]]))

y = np.array([1, 1, 0, 0, 1, 1, 0, 0])

# 实例化
kf = KFold(n_splits=4, random_state=None, shuffle=False)
skf = StratifiedKFold(n_splits=4, random_state=None, shuffle=False)

count = 1
for train_idx, test_idx in kf.split(X=x, y=y):
    print("第%d组: " % count, end="")
    print("训练集索引%s\t测试集索引%s" % (train_idx, test_idx))
    count += 1

print('-' * 50)
count = 1
for train_idx, test_idx in skf.split(X=x, y=y):
    print("第%d组: " % count, end="")
    print("训练集索引%s\t测试集索引%s" % (train_idx, test_idx))
    count += 1
```

结果：

```
第1组: 训练集索引[2 3 4 5 6 7] 测试集索引[0 1]
第2组: 训练集索引[0 1 4 5 6 7] 测试集索引[2 3]
第3组: 训练集索引[0 1 2 3 6 7] 测试集索引[4 5]
第4组: 训练集索引[0 1 2 3 4 5] 测试集索引[6 7]
-----
第1组: 训练集索引[1 3 4 5 6 7] 测试集索引[0 2]
第2组: 训练集索引[0 2 4 5 6 7] 测试集索引[1 3]
第3组: 训练集索引[0 1 2 3 5 7] 测试集索引[4 6]
第4组: 训练集索引[0 1 2 3 4 6] 测试集索引[5 7]
```

```
x = np.array([[1, 2, 3, 4],
              [11, 12, 13, 14],
              [21, 22, 23, 24],
              [31, 32, 33, 34],
              [41, 42, 43, 44],
              [51, 52, 53, 54],
              [61, 62, 63, 64],
              [71, 72, 73, 74]])
```

4个0, 2个1, 不太合理

所属类别: 0 0 1 1 0 0 0和1都是属于类别1的, 说明切分是很随意的

第1组: 训练集索引 [2 3 4 5 6 7] 测试集索引 [0 1] 0的类别是1, 1的类别是0, 很合理

第2组: 训练集索引 [0 1 4 5 6 7] 测试集索引 [2 3] 2和3的类别都是0, 分得很随意

第3组: 训练集索引 [0 1 2 3 6 7] 测试集索引 [4 5]

第4组: 训练集索引 [0 1 2 3 4 5] 测试集索引 [6 7]

----- 所属类别: -1-0-1-1-0-0 -> 3个0, 3个1, 很合理

第1组: 训练集索引 [1 3 4 5 6 7] 测试集索引 [0 2] 0的类别是1, 1的类别是0, 很合理

第2组: 训练集索引 [0 2 4 5 6 7] 测试集索引 [1 3] 1的类别是1, 3的类别是0, 很合理

第3组: 训练集索引 [0 1 2 3 5 7] 测试集索引 [4 6]

第4组: 训练集索引 [0 1 2 3 4 6] 测试集索引 [5 7]

CSDN @LeDvIn

可以看出，SKFold 进行 4 折计算时候，是平衡了测试集中，样本正负的分布的：但是 KFold 却没有。

12.3 自助法 (Bootstrap Method)

我们希望评估的是用 D 训练出的模型。但在留出法和交叉验证法中，由于保留了一部分样本用于测试，因此实际评估的模型所使用的训练集比 D 小，这必然会引起一些因训练样本规模不同而导致的估计偏差。留一法受训练样本规模变化的影响较小，但计算复杂度又太高了。

有没有什么办法可以减少训练样本规模不同造成的影响，同时还能比较高效地进行实验估计呢？

“自助法”(bootstrap)是一个比较好的解决方案，它直接以自助采样法 (bootstrap sampling, BS) 为基础。给定包含 m 个样本的数据集 D ，我们对它进行采样产生数据集 D' ：

- 每次随机从 D 中挑选一个样本，将其拷贝放入 D' ，然后再将该样本放回初始数据集 D 中，使得该样本在下次采样时仍有可能被到；
- 这个过程重复执行 m 次后，我们就得到了包含 m 个样本的数据集 D' ，这就是自助采样的结果。

显然, D 中有一部分样本会在 D' 中多次出现, 而另一部分样本不出现。可以做一个简单的估计, 样本在 m 次采样中始终不被采到的概率是 $(1 - \frac{1}{m})^m$, 取极限得到:

$$\lim_{m \rightarrow +\infty} (1 - \frac{1}{m})^m \rightarrow \frac{1}{e} \approx 36.8\%$$

即通过自助采样, 初始数据集 D 中约有 36.8% 的样本未出现在采样数据集 D' 中。

于是我们可将 D' 用作训练集, D/D' 用作测试集。这样, 实际评估的模型与期望评估的模型都使用 m 个训练样本, 而我们仍有数据总量约 $1/3$ 的、没在训练集中出现的样本用于测试。

这样的测试结果, 亦称“包外估计”(out-of-bag estimate, OOB)。

Q: bootstrap 这个单词应该怎么理解?

A: bootstrap 这个单词原意指靴子上的拉环, 用来帮助人们穿上靴子。后来, 这个词被引申为“自助”的意思, 表示通过自己的力量解决问题。在计算机科学中, bootstrap 指的是计算机启动时加载操作系统的过程; 在统计学中, bootstrap 指的是一种通过对原始数据集进行重采样来生成新数据集的方法, 用于估计总体参数或构建置信区间。

自助法优缺点:

- 优点:
 - 自助法在数据集较小、难以有效划分训练集 / 测试集时很有用
 - 自助法能从初始数据集中产生多个不同的训练集, 这对集成学习等方法有很大的好处。
- 缺点:
 - 自助法产生的数据集改变了初始数据集的分布, 这会引入估计偏差。因此, 在初始数据量足够时, 留出法和交叉验证法更常用一些。

小结:

- 当我们数据量足够时, 选择留出法简单省时, 在牺牲很小的准确度的情况下, 换取计算的简便
- 当我们的数据量较小时, 我们应该选择交叉验证法, 因为此时划分样本集将会使训练数据过少
- 当我们的数据量特别少的时候, 我们可以考虑留一法。

全文完

本文由简悦 SimpRead 优化, 用以提升阅读体验

使用了全新的简悦词法分析引擎 beta, 点击查看详细说明

