特征工程｜连续特征的常见处理方式（含实例）

原创 Thinkgamer 搜索与推荐Wiki 2020-08-20

收录于话题
#Thinkgamer 11 #特征工程 92 #精品小系列内容 25

连续特征离散化可以使模型更加稳健，比如当我们预测用户是否点击某个商品时，一个点击该商品所属类别下次数为100次和一个点击次数为105次的用户可能具有相似的点击行为，有时候特征精度过高也可能是噪声，这也是为什么在LightGBM中，模型采用直方图算法来防止过拟合。

连续特征经常是用户或者事物对应一些行为的统计值，常见的处理方法包括：

- 归一化
- 标准化
- 离散化
- 缺失值处理

这里要特别注意一下归一化和标准化的区别，在平常的使用中，很多同学都容易把这两者的概念混淆，因为两者的英文翻译是一样的（Feature scaling，特征缩放，参考维基百科Feature scaling - Wikipedia）。但是根据其本意和实际用途（或者称之为公式）来理解是不一样的。

归一化

归一化一般是将数据映射到指定的范围，用于去除不同维度数据的量纲以及量纲单位，最常见的归一化方法有：

1、Min-Max 归一化

Min-Max 归一化是指对原始数据进行线性变换，将值映射到[0,1]之间。Min-Max 归一化的计算公式为：

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

式中，x为原始数据中的一个数据，x_{min} 表示原始数据中的最小值，x_{max} 表示原始数据中的最大 值， x'为 Min-Max 归一化后的数据。

2、均值归一化

均值归一化是指通过原始数据中的均值、最大值和最小值来进行数据的标准化。均值归 一化法计算公式为：

$$x' = \frac{x - \mu}{x_{max} - x_{min}}$$

式中，x为原始数据中的一个数据，μ表示原始数据的均值，x_{max}表示原始数据中的最大值，x_{min}表示原始数据中的最小值， x' 为均值归一化后的数据。

3、小数定标归一化

小数定标归一化是指通过移动小数点的位置来进行数据的归一化。小数点移动的位数取决于原始数据中的最大绝对值。小数定标归一化的计算公式为：

$$x' = \frac{x}{10^j}$$

式中，x 为原始数据中的一个数据，x' 经过小数定标（Decimal scaling）归一化后的数据，j 表示满足条件的最小整数。

例如，一组数据为[-309, -10, -43,87,344,970]，其中绝对值最大的是 970。使用小数定标归一化，用 1000(即 j=3)除以每个值。这样，-309 被归一化为-0.309，970 被归一化为 0.97。

4、向量归一化

向量归一化是指通过用原始数据中的每个值除以所有数据之和来进行数据的归一化。向量归一化的计算公式为：

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

式中，x 为原始数据中的一个值，分母表示的是原始数据的所有数据之和，x' 为归一化后的数据。

5、指数归一化

指数转换是指通过对原始数据的值进行相应的指数函数变换来进行数据的标准化。进行指数转换常见的函数方法有lg 函数、$Softmax$ 函数和 $Sigmoid$ 函数。

a. lg 函数

lg 函数对应的标准化计算公式为：

$$x' = \frac{lg(x)}{lg(x_{max})}$$

式中，x 为原始数据中的一个数据，x_{max} 表示原始数据中的最大值，x' 为指数转换后的数据。

b. $Softmax$ 函数

$Softmax$ 函数对应的标准化计算公式为：

$$x' = \frac{e^x}{\sum_{i=1}^{n} e^{x_i}}$$

式中，x 为原始数据中的一个数据，e 为自然常数，分母表示的是原始数据中每个数据被 e 求指数后的和，分子表示的是原始数据中一个数据被 e 求指数，x' 为指数转换后的数据。

c. $Sigmoid$ 函数

Sigmoid 函数对应的标准化计算公式为：

$$x' = \frac{1}{1 + e^{-x}}$$

式中，x 为原始数据中的一个常数，e 为自然常数，x' 为指数转换后的数据。

标准化

标准化：对数据的分布的进行转换，使其符合某种分布（比如正态分布）的一种非线性特征变换。**需要注意的是标准化不改变数据分布的类型，比如原始数据服从正态分布，标准化处理之后依旧服从正态分布，只是数据的均值和方差发生了变化。可以结合PCA算法进行理解，如果分布改变了，那主成分分析就得不到原始数据的分布信息了。**

使用最广泛的标准化方法为：Z-Score标准化。Z-Score（也叫 Standard Score，标准分数）标准化是指基于原始数据的均值（mean）和 标准差（standard deviation）来进行数据的标准化。Z-Score 标准化的计算公式为：

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

式中，x 为原始数据中的一个数据，μ 表示原始数据的均值，σ 表示原始数据的标准差，x' 为 Z-Score 标准化后的数据。

特征经过归一化或者标准化处理之后对于模型训练的好处有：

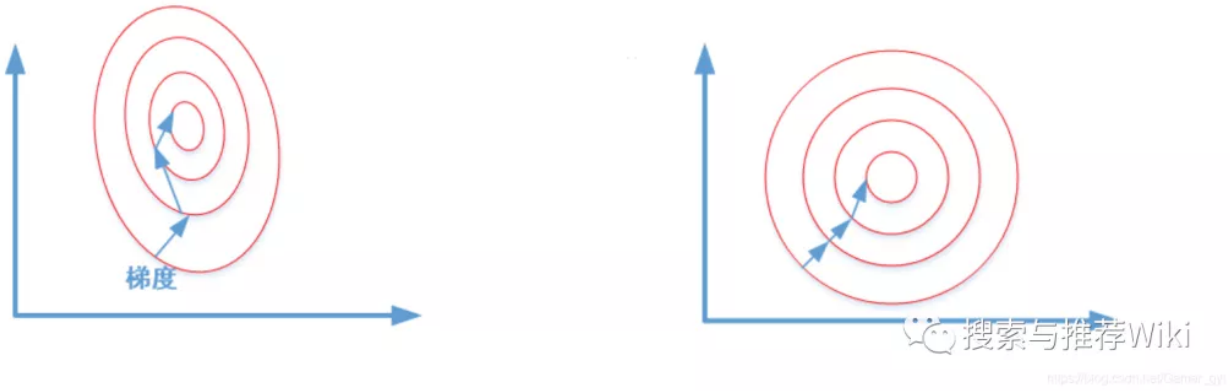
- 提升模型精度。

因为使不同量纲的特征处于同一数值量级，减少方差大的特征的影响。在KNN中，我们需要计算待分类点与所有实例点的距离。假设每个实例点（instance）由n个features构成。如果我们选用的距离度量为欧式距离，如果数据预先没有经过归一化，那么那些绝对值大的features在欧式距离计算的时候起了决定性作用。

从经验上说，归一化是让不同维度之间的特征在数值上有一定比较性，可以大大提高分类器的准确性。

- 提升收敛速度。

对于线性model来说，数据归一化后，最优解的寻优过程明显会变得平缓，更容易正确的收敛到最优解。



提升模型收敛速度

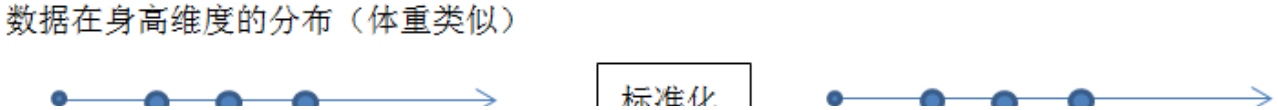
在对特征进行具体的处理时，需要结合具体的业务场景和算法模型评估特征是否需要进行归一化或者标准化。

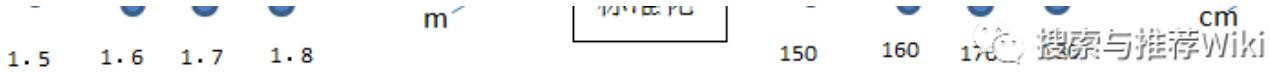
归一化改变数据分布，标准化不会改变数据分布，如下例：

根据人的身高和体重预测人的健康指数，假设有如下原始样本数据是四维的（当然一般不会有这么无聊的数据）

样本|特征	身高（m）	体重（kg）	身高（cm）	体重（g）
样本 1	1.7	70	170	70000
样本 2	1.8	80	180	80000
样本 3	1.6	60	160	60000

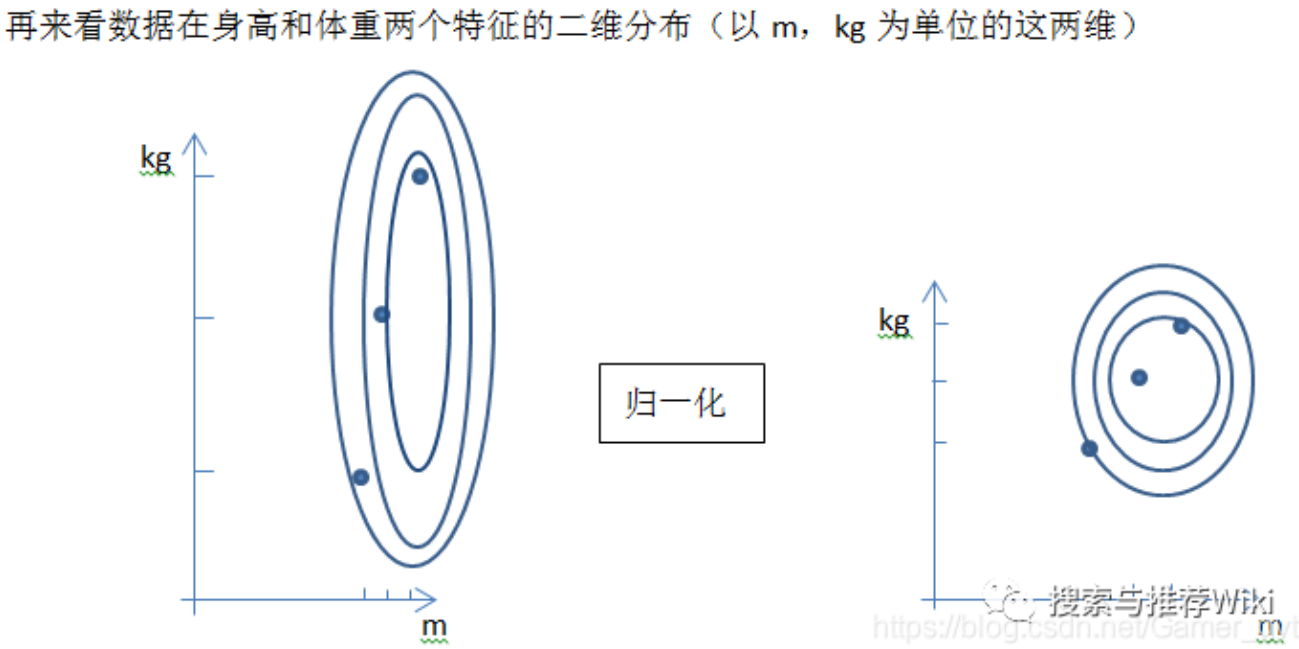
在这里插入图片描述





在这里插入图片描述

从上面两个坐标图可以看出，样本在数据值上的分布差距是不一样的，但是其几何距离是一致的。而标准化就是一种对样本数据在不同维度上进行一个伸缩变化（**而不改变数据的几何距离**），也就是不改变原始数据的信息（分布）。这样的好处就是在进行特征提取时，忽略掉不同特征之间的一个度量，而保留样本在各个维度上的信息（分布）。



在这里插入图片描述

从采用大单位的身高和体重这两个特征来看，如果采用标准化，不改变样本在这两个维度上的分布，则左图还是会保持二维分布的一个扁平性；而采用归一化则会在不同维度上对数据进行不同的伸缩变化（**归一区间，会改变数据的原始距离，分布，信息**），使得其呈类圆形。虽然这样样本会失去原始的信息，但这防止了归一化前直接对原始数据进行梯度下降类似的优化算法时最终解被数值大的特征所主导。归一化之后，各个特征对目标函数的影响权重是一致的。这样的好处是在提高迭代求解的精度。

当然，数据归一化并不是万能的。在实际应用中，通过梯度下降法求解的模型通常是需要归一化的，包括线性回归、逻辑回归、支持向量机、神经网络等模型。但对于决策树模型则并不适用，以C4.5为例，决策树在进行节点分裂时主要依据数据集*D*关于特征*x*的信息增益比，而信息增益比跟特征是否经过归一化是无关的，因为归一化并不会改变样本在特征 x 上的信息增益。

离散化

数据离散化（也叫数据分组）是指将连续的数据进行分组，使其变为一段段离散化的区间，离散化后的特征根据其所在的组进行One-Hot编码。

根据离散化过程中是否考虑类别属性，可以将离散化算法分为有监督算法和无监督算法两 类。由于有监督算法（如基于熵进行数据的离散化）充分利用了类别属性的信息，所以在分类中能获得较高的正确率。

常见的数据离散化方法有以下几种（以下介绍的数据分组方法均需要对数据进行排序，且假设待离散化的数据按照升序排列）：

- 二值化分组
- 等宽分组
- 等频分组
- 单变量分组
- 基于信息熵分组

1、二值化分组

二值化分组比较好理解，即将离散特征根据某个值划分成两个字段。

例如，一组变量(1,7,12,12,22,30,34,38,46)，可以将分割值设置为22，即小于等于22的为一组，大于22的为一组，划分后为：(1,7,12,12,22)，(30,34,38,46)。

2、等宽分组

等宽分组的原理是，根据分组的个数得出固定的宽度，分到每个组中的变量的宽度是相等的。

例如，将一组变量(1,7,12,12,22,30,34,38,46)分成三组。

- 宽度为 15，即用变量中的最大值(46)减去变量中的最小值(1)，然后用差除以组数(3)。
- 每组的范围为:(1,16]、(16,31]、(31,46]
- 分组后的结果为:(1,7,12,12)、(22,30)、(34,38,46)

> (1,16]表示半开半闭区间，即大于 1 且小于或等于 16。另外，采用半开半闭区间时，最小值不能进行有效分组，这里默认将其归为第一组。

3、等频分组

等频分组也叫分位数分组，即分组后每组的变量个数相同。

例如，将一组变量(1,7,12,12,22,30,34,38,46)分成三组。

变量的总个数为 9，所以每组的变量为 3 个。分组后的结果为:(1,7,12)、(12,22,30)、(34,38,46)。

> 等宽分组和等频分组实现起来比较简单，但都需要人为地指定分组个数。
>
> 等宽分组的缺点是：对离群值比较敏感，将属性值不均匀地分布到各个区间。有些区间包含的变量较多，有些区间包含的变量较少。
>
> 等频分组虽然能避免等宽分组的缺点，但是会将相同的变量值分到不同的组（以满足每个组内的变量个数相同）。

4、单变量分组

单变量分组也叫秩分组。其原理是：将所有变量按照降序或升序排序，排序名次即为排序结果，即将值相同的变量划分到同一组。

例如，将一组变量(1,7,12,12,22,30,34,38,46)分成三组，去重后，变量个数为 8，所以该组变量的分组数目为 8。

结果为：(1)、(7)、(12,12)、(22)、(30)、(34)、(38)、(46)。

5、基于信息熵分组

在学习信息熵分组之前，先了解一下信息量和熵的概念。

信息量

香农（Shannon）被称为“信息论之父”，他认为“信息是用来消除随机不确定性的东西”。即，衡量信息量大小就看这个信息消除不确定性的程度。

信息量是对事件发生概率的度量，一个事件发生的概率越低，则这个事件包含的信息量越大，这跟我们直观上的认知也是吻合的，越稀奇的新闻包含的信息量越大，因为这种新闻出现的概率低，即信息量的大小和事件发生的概率成反比。香农提出了一个定量衡量信息量的公式：

$$l(x) = -log_2 p(x)$$

式中，$p(x)$表示 x 事件发生的概率。信息量度量的是“一个具体事件发生”所带来的信息。

熵

熵是在结果出来之前对可能产生的信息量的期望——考虑该随机变量的所有可能取值，即所有可能发生事件所带来的信息量的期望。熵越大表示结果的不确定性越大，熵为0表示没有任何不确定性。熵的计算公式为：

$$E(x) = -\sum_{i=1}^{n} p(x_i) log_2 p(x_i)$$

式中，$p(x_i)$ 表示 x_i 事件发生的概率，n 为 x 中所有类别的个数。

按照随机变量的所有可能取值划分数据的总熵 E 是所有事件的熵的加权平均：

$$E = \sum_{i=1}^{k} w_i E_i$$

式中，$w_i = \frac{m_i}{m}$ 是第 x 个事件出现的比例，m_i 是第 i 个可能取值出现的次数，m 是所有取值出现的总次数。

所以，基于信息熵进行数据分组的具体做法是：

- 1、对属性 A 的所有取值从小到大排序

- 2、遍历属性 A 的每个值 V_i，将属性 A 的值分为两个区间 S_1、S_2，使得将其作为分隔点划分数据集后的熵 S 最小，熵 S 的计算方式见式上边的两个公式
- 3、当划分后的熵大于设置的阈值且小于指定的数据分组个数时，递归对 S_1、S_2 执行步骤2中的划分

6、基于用户是否点击和信息熵对商品价格进行离散化

接下来通过一个实例看一下如何基于信息熵对数据进行离散化。基于信息熵的数据离散化算法是有监督学习算法。在使用该方法对数据进行离散化时，需 要数据有对应的标签。假设现在有一份下面这样的数据，其表示的是用户是否点击商品和商品价格之间的关系（当然这是一份假设的数据）。

价格
标签
价格
标签
价格
标签

56	1	453	1	764	0
87	1	10	1	121	1
129	0	9	0	28	0
23	0	88	1	49	1
342	1	222	0	361	1
641	1	97	0	164	0
63	0	2398	1	1210	1
2764	1	592	1		
2323	0	561	1		

对该份数据进行离散化，其实现的代码如下：

```
# Author: Thinkgamer
# Desc  : 基于信息熵的数据离散化
```

```
import numpy as np
import math

class DataSplitByEntropy:
    def __init__(self, group, threshold):
        # 最大分组数
        self.max_group = group
        # 停止划分的最小熵
        self.min_threshold = threshold
        # 保存最终结果变量
        self.result = dict()

    # 加载数据
    def load_data(self):
        data = np.array(
            [
                [56, 1],   [87, 1],  [129, 0],  [23, 0],   [342, 1],
                [641, 1],  [63, 0],  [2764, 1], [2323, 0], [453, 1],
                [10, 1],   [9, 0],   [88, 1],   [222, 0],  [97, 0],
                [2398, 1], [592, 1], [561, 1],  [764, 0],  [121, 1],
                [28, 0],   [49, 1],  [361, 1],  [164, 0],  [1210, 1]
            ]
        )
        return data

    # 计算按照数据指定数据分组后的香农熵
    def cal_entropy(self, data):
        num_data = len(data)
        label_counts = {}
        for feature in data:
            # 获得标签
            one_label = feature[-1]
            # 如果标签不在新定义的字典里则创建该标签
            label_counts.setdefault(one_label, 0)
            # 该类标签下含有数据的个数
            label_counts[one_label] += 1
        entropy = 0.0
        for key in label_counts:
            # 同类标签出现的概率
            prob = float(label_counts[key]) / num_data
            # 以2为底求对数
            entropy -= prob * math.log(prob, 2)
        return entropy
```

```
# 按照调和信息熵最小化原则分割数据集
def split(self, data):
    # inf为正无穷大
    min_entropy = np.inf
    # 记录最终分割索引
    index = -1
    # 按照第一列对数据进行排序
    sort_data = data[np.argsort(data[:, 0])]
    # 初始化最终分割数据后的熵
    last_e1, last_e2 = -1, -1
    # 返回的数据结构，包含数据和对应的熵
    S1 = dict()
    S2 = dict()
    for i in range(len(sort_data)):
        # 分割数据集
        split_data_1, split_data_2 = sort_data[: i + 1], sort_data[i + 1 :]
        entropy1, entropy2 = ( self.cal_entropy(split_data_1), self.cal_entropy(split_data
        entropy = entropy1 * len(split_data_1) / len(sort_data) +  entropy2 * len( split_d
        # 如果调和平均熵小于最小值
        if entropy < min_entropy:
            min_entropy = entropy
            index = i
            last_e1 = entropy1
            last_e2 = entropy2
    S1["entropy"] = last_e1
    S1["data"] = sort_data[: index + 1]
    S2["entropy"] = last_e2
    S2["data"] = sort_data[index + 1 :]
    return S1, S2, entropy

# 对数据进行分组
def train(self, data):
    # 需要遍历的 key
    need_split_key = [0]
    # 将整个数据作为一组
    self.result.setdefault(0, {})
    self.result[0]["entropy"] = np.inf
    self.result[0]["data"] = data
    group = 1
    for key in need_split_key:
        S1, S2, entropy = self.split(self.result[key]["data"])
```

```
            # 如果满足条件
            if entropy > self.min_threshold or group < self.max_group:
                self.result[key] = S1
                newKey = max(self.result.keys()) + 1
                self.result[newKey] = S2
                need_split_key.extend([key])
                need_split_key.extend([newKey])
                group += 1
            else:
                break

if __name__ == "__main__":
    dse = DataSplitByEntropy(group=6,threshold=0.3)
    data = dse.load_data()
    dse.train(data)
    print("result is {}".format(dse.result))
```

输出结果为：

```
result is {0: {'entropy': 0.8112781244591328, 'data': array([[ 9,  0],
       [10,  1],
       [23,  0],
       [28,  0]])}, 1: {'entropy': 0.0, 'data': array([[342,   1],
       [361,   1],
       [453,   1],
       [561,   1],
       [592,   1],
       [641,   1]])}, 2: {'entropy': 0.0, 'data': array([[129,   0]])}, 3: {'entropy': 0.97095
       [1210,    1],
       [2323,    0],
       [2398,    1],
       [2764,    1]])}, 4: {'entropy': 0.863120568566631, 'data': array([[ 49,   1],
       [ 56,   1],
       [ 63,   0],
       [ 87,   1],
       [ 88,   1],
       [ 97,   0],
       [121,   1]])}, 5: {'entropy': 0.0, 'data': array([[164,   0],
       [222,   0]])}}
```

分析结果可以看出，熵越大的簇类下数据对应的标签越凌乱，熵为0的簇类下数据对应的标签是一致的，这里可以自行调节group、threshold参数观察结果。

The end

收录于话题 #特征工程·92个

上一篇
特征工程｜特征交叉（Feature Crosses）

下一篇
特征工程｜类别特征的常见处理方式（含代码）

阅读原文

喜欢此内容的人还喜欢