特征工程｜类别特征的常见处理方式（含代码）

原创 Thinkgamer 搜索与推荐Wiki 2020-08-12

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

特征工程的完整流程是：特征设计 -> 特征获取 -> 特征处理 -> 特征存储 -> 特征监控。本篇主要介绍如何对类别特征进行处理。

> 文章较长，建议【先收藏再细品】，点击文末【阅读原文】查看更多精彩内容

类别特征即特征的属性值是一个有限的集合，常见几种处理方法为：

- 序号编码
- One-Hot（独热）编码
- 哑变量（虚拟）编码
- 二进制编码
- 效应编码
- 哈希编码
- 统计学中常用编码

1、序号编码

序号编码（Label Encoding）即通过数字序号和值进行一一映射达到编码的目的。但它适用于处理具有大小关系或递进关系的特征，比如成绩的优良中差可以进行序列编码为：差->0，中->1，良->2，优->3，比如上学的不同阶段可以进行序列编码为：幼儿园->0，小学->1，初中->2，高中->3，大学->4，硕士->5，博士->6。但对于一些平级的特征则不适合用序号编码进行表示，比如性别（男、女），全国的省份，这些都是平级的，转变为序号数值之后会增加一层

大小关系，这对于数值大小敏感的模型（比如SVM、LR等）则是致命的，因为会影响损失函数之类的计算结果。

sklearn中包含了序号编码的处理模块（LabelEncoder），其使用方式如下：

```
# 序号编码
from sklearn.preprocessing import LabelEncoder
print("sklean版本为：" + sklearn.__version__)

score_level = ["优","良","中","差"]
le = LabelEncoder()
le.fit(score_level)
print("原始的类别数据为：%s" % score_level)
print("不重复的类别值有：%s" % le.classes_)
print("经过序号编码后的数据为：%s" % le.transform(score_level))
print("序号编码后的数据还原为：%s" % le.inverse_transform( le.transform(score_level) ))
```

输出结果为：

```
sklean版本为：0.22.1
原始的类别数据为：['优', '良', '中', '差']
不重复的类别值有：['中' '优' '差' '良']
经过序号编码后的数据为：[1 3 0 2]
序号编码后的数据还原为：['优' '良' '中' '差']
```

这里需要注意的是，经过转换后的数据格式为 numpy.ndarray。

2、One-Hot（独热）编码

One-Hot编码也叫独热编码，指的是将原始特征变量转换为原始特征值分类的多维度变量，在每个维度上使用0/1来进行是/否、有/无的量化。通俗来说就是，把每个取值作为一个新特征，一行数据中，取到了这个值就是1，没取到这个值就是0。

比如上文描述的成绩的优良中差，我们认为产出的4维特征数组表示的含义为：[是否是优,是否是良,是否是中,是否是差]，是为1，否为0，那么优可以表示为：[1,0,0,0]，良可以表示为：[0,1,0,0]，中可以表示为：[0,0,1,0]，差可以表示为：[0,0,0,1]。

sklearn中包含了序号编码的处理模块（OneHotEncoder），其使用方式如下：

```
# 序号编码
from sklearn.preprocessing import OneHotEncoder
print("sklean版本为：" + sklearn.__version__)

score_level = [["优"],["良"],["中"],["差"]]
enc = OneHotEncoder(handle_unknown='ignore')
enc.fit(score_level)
print("原始的类别数据为：%s" % score_level)
print("不重复的类别值有：%s" % enc.categories_)
print("经过One-Hot编码后的数据为：\n %s" % enc.transform(score_level).toarray())
print("One-Hot编码复原后的数据为：\n %s" % enc.inverse_transform([[0., 1., 0., 0.], [0., 0., 0.,
print("打印输出特征的名字：\n %s" % enc.get_feature_names(['score_level']))
```

输出结果为：

```
sklean版本为：0.22.1
原始的类别数据为：[['优'], ['良'], ['中'], ['差']]
不重复的类别值有：[array(['中', '优', '差', '良'], dtype=object)]
经过One-Hot编码后的数据为：
 [[0. 1. 0. 0.]
 [0. 0. 0. 1.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]]
One-Hot编码复原后的数据为：
 [['优']
 ['良']
 ['中']
 ['差']]
打印输出特征的名字：
 ['score_level_中' 'score_level_优' 'score_level_差' 'score_level_良']
```

同时OneHotEncoder也支持多个特征的转化，使用案例如下：

```
score_level_gender = [["优", "男"],["良", "女"],["中", "男"],["差", "女"]]
enc_more = OneHotEncoder(handle_unknown='ignore')
enc_more.fit(score_level_gender)
print("原始的类别数据为：%s" % score_level_gender)
```

```
print("不重复的类别值有：%s" % enc_more.categories_)
print("经过One-Hot编码后的数据为：\n %s" % enc_more.transform(score_level_gender).toarray())
print("独热One-Hot复原后的数据为：\n %s" % enc_more.inverse_transform([[0., 1., 0., 0., 0., 1.], [
print("打印输出特征的名字：\n %s" % enc_more.get_feature_names(['score_level', "gender"]))
```

输出结果为：

```
原始的类别数据为：[['优', '男'], ['良', '女'], ['中', '男'], ['差', '女']]
不重复的类别值有：[array(['中', '优', '差', '良'], dtype=object), array(['女', '男'], dtype=object)]
经过One-Hot编码后的数据为：
 [[0. 1. 0. 0. 0. 1.]
 [0. 0. 0. 1. 1. 0.]
 [1. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 1. 0.]]
One-Hot编码复原后的数据为：
 [['优' '男']
 ['良' '女']
 ['中' '男']
 ['差' '男']]
打印输出特征的名字：
 ['score_level_中' 'score_level_优' 'score_level_差' 'score_level_良'
 'gender_女' 'gender_男']
```

Pandas中也提供了相应的函数来进行特征的One-Hot编码，使用举例如下：

```
import pandas as pd
print("pandas版本为：%s" % pd.__version__)

df = pd.DataFrame([["优", "男"],["良", "女"],["中", "男"],["差", "女"]])
df.columns = ['score_level', "gender"]
print("经过One-Hot编码后的数据为：")
pd.get_dummies(df).values
```

输出结果为：

```
pandas版本为：1.0.1
经过One-Hot编码后的数据为：
array([[0, 1, 0, 0, 0, 1],
       [0, 0, 0, 1, 1, 0],
       [1, 0, 0, 0, 0, 1],
       [0, 0, 1, 0, 1, 0]], dtype=uint8)
```

采用One-Hot编码的特征，特征时间没有大小和级别关系，对于一些线性模型比较友好，同时扩充了特征，在一定程度上降低了过拟合的风险。同样如果类别值过多，则会导致One-Hot编码后的特征变得维度骤增，增加模型训练的成本和数据存储的空间。而且One-Hot编码之后的特征对于树模型并不友好，如果特征很多，取这个值的数量又很少，那么一些特征变量很可能因为树模型的参数配置而无法向下分裂，导致这些信息白白丢失掉了。

一种减少数据存储空间的有效办法是使用稀疏向量保存，比如成绩中的“优”对应的One-Hot编码为：[0,1,0,0]，可以表示为：[4,(2,1)]。但这样并不能降低特征的维度。

3、哑变量（虚拟）编码

哑变量（Dummy Variable），也叫虚拟变量，引入哑变量的目的是，将不能够定量处理的变量量化。哑变量编码（Dummy Encoding）的直观的解释就是任意的将一个状态位去除。整体使用0、1表示。比如上文中提到的成绩用优良中差表示，如果一个人的成绩既不是优良中，那么他的成绩就是差了，用哑变量编码优可以表示为：[1,0,0,0]，良可以表示为：[0,1,0,0]，中可以表示为：[0,0,1,0]，差可以表示为：[0,0,0,0]。

通过对比成绩优良中差的例子，可以看出它们的“思想路线”是相同的，只是哑变量编码觉得One-Hot编码太罗嗦了（一些很明显的事实还说的这么清楚），所以它就很那么很明显的东西省去了。这种简化不能说到底好不好，这要看使用的场景。但通常情况下还是建议使用One-Hot编码，哑变量编码也可以使用，不过最好选择前者。虽然哑变量编码可以去除One-Hot编码的冗余信息，但是因为每个离散型特征各个取值的地位都是对等的，随意取舍未免来的太随意。

4、二进制编码

二进制编码（Binary Encoding）即采用二进制来表示类别，其编码过程分为两步：

- 使用ID对每个类别值进行顺序编码，该过程类似于序号编码
- 将类别编码对应的ID用二进制表示

比如上文提到的成绩对应优良中差四个档次，其经过第一、第二步之后的二进制编码表示为：

成绩	类别ID	二进制编码
优	1	0 0 1

成绩	类别ID	二进制编码
良	2	0 1 0
中	3	0 1 1
差	4	1 0 0

可以看出，二进制编码本 质上是利用二进制对ID进行哈希映射，最终得到0/1特征向量，对比One-Hot编码或者序号编码维数较少，节省了存储空间。但因其直观性不强，在实际场景中使用很少。

5、效应编码

效应编码（Effect Coding）是一种和哑变量编码（Dummy Encoding）类似的编码方法，对于有 n 个分类属性的自变量，通常需要选取1个分类作为参照，因此可以产生 $n-1$ 个虚拟变量。效应编码是使用1、0和-1来编码，来反映某个变量的不同属性，与哑变量编码相比，区别在于作为参照那一类的赋值。效应编码是将参照的那一类全部赋值为-1，而哑变量编码则是把参照的那一类全部赋值为0。

还是拿成绩的优良中差为例，用效应编码优可以表示为：[1,0,0,0]，良可以表示为：[0,1,0,0]，中可以表示为：[0,0,1,0]，差可以表示为：[-1,-1,-1,-1]。

6、哈希编码

首先看一下什么是哈希（Hash）算法，哈希算法并不是一个特定的算法而是一类算法的统称。哈希算法也叫散列算法，一般来说满足这样的关系：$f(data)=key$，输入任意长度的data数据，经过哈希算法处理后输出一个定长的数据key。同时这个过程是不可逆的，无法由key逆推出data。

因此在面对高基数类别变量时，就可以用特征哈希法编码的方式将原始的高维特征向量压缩成较低维特征向量，且尽量不损失原始特征的表达能力，这就是哈希编码（Hash Encoding）。

哈希编码有其自己的优劣：

- 优点
  - 降低特征数量，加速算法训练和预测过程，降低内存消耗
  - 可以添加新的任务（如新用户），或者新的原始特征而保持哈希转换后的特征长度不变，很适合任务数频繁变化的问题（如个性化推荐里新用户，新item的出现）
  - 可以保持原始特征的稀疏性，既然哈希转换时只有非0原始特征才起作用
  - 可以只哈希转换其中的一部分原始特征，而保留另一部分原始特征（如那些出现collision就会很影响精度的重要特征）

- 缺点
  - 通过哈希编码转换学习到的模型变得很难校验，很难对训练处的模型参数作出合理解释
  - 哈希编码会把多个原始特征哈希到相同的位置上，出现哈希中的collision现象，但实际实验表明这种collision对算法的精度影响很小（具体可以参考论文：Collaborative Email-Spam Filtering with the Hashing-Trick）

7、统计学中常用编码

在计量统计学中常用的几种类别特征处理方式包括：

- Indicator Contrast（指示对比）
- Simple Contrast（简单对比）
- Difference Contrast（差异对比）
  - Forward Difference Coding（前向差异对比）
  - Backward Difference Coding（后向差异对比）
- Helmert Contrast（赫尔默特对比）
- Repeated Contrast（重复对比）
- Polynomial Contrast（多项式对比）
- Deviation Contrast（偏差对比，也叫Sum Contrast）

这些编码实现，在R语言中实现比较方便，有相应的基础组件可以直接使用，但在Python中实现起来就没有那么方便了。感兴趣的可以自行调研其具体的表达含义和实现方法，这里不做介绍。

8、Categorical Encoders介绍

Github上有一个开源的类别特征处理包（category_encoders，https://github.com/scikit-learn-contrib/category_encoders），其包含了上文提到的一些和没有提到的一些类别特征处理方法，主要有：

无监督类方法：

- Backward Difference Contrast
- BaseN
- Binary
- Count
- Hashing
- Helmert Contrast
- Ordinal

- One-Hot
- Polynomial Contrast
- Sum Contrast

有监督类方法：

- CatBoost
- Generalized Linear Mixed Model
- James-Stein Estimator
- LeaveOneOut
- M-estimator
- Target Encoding
- Weight of Evidence

下面通过几个案例演示一下其使用方式（category encoders的安装可以参考github官方文档）。

无监督之序号编码

```
# https://github.com/scikit-learn-contrib/category_encoders  部分代码示例

import numpy as np
import pandas as pd
import category_encoders as ce

print("numpy 版本为：%s" %  np.__version__)
print("pandas 版本为：%s" % pd.__version__)
print("category_encoders 版本为：%s" % category_encoders.__version__)

# 无监督-序号编码
data = pd.DataFrame(np.array([["优", "男"],["良","女"],["中","男"],["差","女"]]), columns=["score",
print("\n 原始数据为：\n %s" % data)
encoder = ce.OrdinalEncoder(cols=['score', "sex"]).fit(data)
print("\n 序号编码信息为：\n %s" % encoder)
result = encoder.transform(data)
print("\n 序号编码后的数据为：\n %s" % result)
```

输出结果为：

```
numpy 版本为：1.18.1
pandas 版本为：1.0.1
category_encoders 版本为：2.2.2

 原始数据为：
   score sex
0      优   男
1      良   女
2      中   男
3      差   女

 序号编码信息为：
 OrdinalEncoder(cols=['score', 'sex'], drop_invariant=False,
                handle_missing='value', handle_unknown='value',
                mapping=[{'col': 'score', 'data_type': dtype('O'),
                          'mapping': 优       1
良        2
中        3
差        4
NaN    -2
dtype: int64},
                         {'col': 'sex', 'data_type': dtype('O'),
                          'mapping': 男       1
女        2
NaN    -2
dtype: int64}],
                return_df=True, verbose=0)

 序号编码后的数据为：
    score  sex
0      1    1
1      2    2
2      3    1
3      4    2
```

无监督之Helmert Contrast编码

```
# 无监督-Helmert Contrast
encoder1 = ce.HelmertEncoder(cols=['score', "sex"]).fit(data)
print("\n Helmert Contrast编码信息为：\n %s" % encoder1)
print("\n Helmert Contrast编码后的数据为：\n %s" % encoder1.transform(data))
```

输出结果为：

```
Helmert Contrast编码信息为:
 HelmertEncoder(cols=['score', 'sex'], drop_invariant=False,
                handle_missing='value', handle_unknown='value',
                mapping=[{'col': 'score',
                          'mapping':      score_0   score_1   score_2
 1       -1.0       -1.0       -1.0
 2        1.0       -1.0       -1.0
 3        0.0        2.0       -1.0
 4        0.0        0.0        3.0
-1        0.0        0.0        0.0
-2        0.0        0.0        0.0},
                          {'col': 'sex',
                           'mapping':      sex_0
 1    -1.0
 2     1.0
-1     0.0
-2     0.0}],
                return_df=True, verbose=0)

 Helmert Contrast编码后的数据为:
    intercept   score_0   score_1   score_2   sex_0
0           1      -1.0      -1.0      -1.0    -1.0
1           1       1.0      -1.0      -1.0     1.0
2           1       0.0       2.0      -1.0    -1.0
3           1       0.0       0.0       3.0     1.0
```

有监督之目标编码

目标编码是一种不仅基于特征值本身，还基于相应因变量的类别变量编码方法。

对于分类问题：将类别特征替换为给定某一特定类别值的因变量后验概率与所有训练数据上因变量的先验概率的组合。对于连续目标：将类别特征替换为给定某一特定类别值的因变量目标期望值与所有训练数据上因变量的目标期望值的组合。

该方法严重依赖于因变量的分布，但这大大减少了生成编码后特征的数量。

```
# 训练集
x_train = pd.DataFrame(np.array([["男", 12],["男",15],["男",13],["女",56],["女",69],["女",59]]), c
y_train = np.array([0, 0, 0, 1, 1, 1])
# 测试集
x_test = pd.DataFrame(np.array([["男", 14],["女", 61]]), columns=["sex", "num"])
x_test.loc[2,'num'] = np.nan
# 在训练集上进行训练
encoder2 = ce.TargetEncoder(cols=["sex", "num"], handle_unknown='value',  handle_missing='value')
print("\n 目标编码信息为：\n %s" % encoder2)

encoder_train = encoder2.transform(x_train)
print("\n 经过目标编码后的训练集数据为：\n %s" % encoder_train)

encoder_test = encoder2.transform(x_test)
print("\n 经过目标编码后的测试集数据为：\n %s" % encoder_test)

# 验证一下计算的结果，在测试集中，男性类别的编码值为 0.059601
# 先验概率
prior = y_train.mean()
# 默认为1.0
min_samples_leaf = 1.0
# 默认为1.0
smoothing = 1.0
# 训练集中，三个样本包含‘male’这个标签
n = 3
# 在训练集中，这三个包含‘male’标签的样本中仅有零个有正的因变量标签
n_positive = 0
smoove = 1 / (1 + np.exp(-(n - min_samples_leaf) / smoothing))
male_encode = prior * (1-smoove) + smoove * n_positive/n
# return 0.05960146101105884，与要验证的值吻合
print("\n 校验数据，男性编码值为：\n %s" % male_encode)
```

输出结果为：

```
目标编码信息为：
 TargetEncoder(cols=['sex', 'num'], drop_invariant=False, handle_missing='value',
               handle_unknown='value', min_samples_leaf=1, return_df=True,
               smoothing=1.0, verbose=0)
```

经过目标编码后的训练集数据为：

```
        sex   num
0  0.059601   0.5
1  0.059601   0.5
2  0.059601   0.5
3  0.940399   0.5
4  0.940399   0.5
5  0.940399   0.5
```

经过目标编码后的测试集数据为：

```
        sex   num
0  0.059601   0.5
1  0.940399   0.5
2  0.500000   0.5
```

校验数据，男性编码值为：

```
0.059601461011058 84
```

The end