**汽车评估案例**

# 1简介

数据编码是数据预处理中的一个关键环节。实际遇到的数据中，很多特征的取值使用字符串来编码。例如将性别分别用“男”和“女”来表示。数据分析和模型却只能处理数字型的特征，例如分别使用0和1来表示“男”和“女”。常见的特征编码方法包括数字编码和one-hot编码等。

# 2数据及环境准备

## 2.1数据和环境说明

本案例基于真实的汽车评估数据集进行。该数据集包含1728个汽车样本，每个样本包括7个特征（其中acceptance特征为预测特征）。特征代表的含义及其取值如下：

buying：购买价格，取值为vhigh、high、med、low。

maint：维护价格，同样取值为vhigh、high、med、low。

doors：车门数量，取值为2,3,4,5more。

persons：核载人数，取值为2,4，more。

lug\_boot：后备箱尺寸，取值为small、med、big。

safety：安全性，取值为low、med、high。

acceptance：可接受程度，取值为unacc、acc、good、vgood。

请大家将案例文件夹中的数据集“car.csv”拷贝到我们创建的input文件夹中。

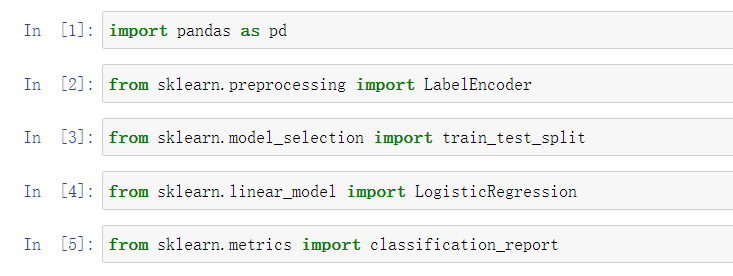
我们接下来首先演示如何对数据进行特征编码。然后通过建立逻辑回归分类模型来评估不同的编码方法对模型效果的影响。

我们将使用Anaconda结合Jupyter Notebook环境进行实验，所用到的工具集有pandas和sklearn（scikit-learn）。

## 2.2引入依赖

可以逐步调试和运行是notebook的主要优势之一，大家在导入依赖时，尽量逐个导入，这样在导入失败时，方便确认问题所在。

import pandas as pd  
from sklearn.preprocessing import LabelEncoder  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import classification\_report



## 2.3读入数据

借助pandas文件读写函数read\_csv()，我们将原数据集读入。

*# 读入数据*car = pd.read\_csv(**"D:/input/car.csv"**)  
*# 打印数据集的前五行*car.head(5)

查看数据集的大小

*# 查看数据集大小*car.shape

# 3数据编码

## 3.1查看数据

从上面的特征介绍可以看到，本数据集中几乎所有的特征都是用字符串表示的离散型特征。我们可以用pandas的value\_counts函数查看某一特征各个取值的分布情况。

这里用客户的接受程度“acceptance”为例，查看此特征各个取值的分布。

*# 查看各个取值的数量分布*print(car[**"acceptance"**].value\_counts())

*# 查看各个取值的比例分布*print((car[**"acceptance"**].value\_counts() + 0.0) / car.shape[0])

可接受程度为“不可接受”的汽车占比超过70%，“可接受”的汽车占比约22%，而接受程度“好”和“非常好”的汽车占比不到8%，不同类别的样本分布不均衡。

## 3.2数字编码

在我们的汽车评估数据集中，离散型特征为词序型。例如汽车价格为“低”、“中等”、“高”、“非常高”。数字编码时需要注意保持这种次序关系，例如我们使用0-3分别表示这四个值：

|  |  |
| --- | --- |
| 特征取值 | 编码 |
| low | 0 |
| med | 1 |
| high | 2 |
| vhigh | 3 |

我们可以借助一个map函数完成特征转换，此函数会根据原数据取值在字典中查找编码结果，完成转换。因为此后要对比数字编码和one-hot编码对模型性能的影响，我们需要新建一个数据集进行编码。

*# 创建要进行数字编码的数据集*car\_num\_encoded = car.copy()  
*# 根据编码规则创建编码字典*buying\_dict = {**"low"**: 0, **"med"**: 1, **"high"**: 2, **"vhigh"**: 3}  
*# 进行编码*car\_num\_encoded[**"buying"**] = car\_num\_encoded[**"buying"**].map(buying\_dict)

输出此列的前五行，查看编码结果。

*# 查看编码列的前五行*car\_num\_encoded[**"buying"**].head(5)

对其他几个全字符串属性maint、lug\_boot、safety采用同样的方法完成特征编码。

*# maint特征数字编码*maint\_dict = {**"low"**: 0, **"med"**: 1, **"high"**: 2, **"vhigh"**: 3}  
car\_num\_encoded[**"maint"**] = car\_num\_encoded[**"maint"**].map(maint\_dict)  
  
*# lug\_boot特征数字编码*lug\_boot\_dict = {**"small"**: 0, **"med"**: 1, **"big"**: 2}  
car\_num\_encoded[**"lug\_boot"**] = car\_num\_encoded[**"lug\_boot"**].map(lug\_boot\_dict)  
  
*# safety特征数字编码*safety\_dict = {**"low"**: 0, **"med"**: 1, **"high"**: 2}  
car\_num\_encoded[**"safety"**] = car\_num\_encoded[**"safety"**].map(safety\_dict)

查看编码后的前五行。

car\_num\_encoded.head(5)

需要注意的是doors和persons两个特征取值并不是数字型，详情请看一下案例开头的数据说明，我们也需要将其进行编码。doors取值范围为2,3,4,5more，我们直接将5more编码为6，其余取值不变。Persons特征的取值范围为2,4，more，我们直接将more编码成6，其余取值不变。

*# 将车门数量一列中，5more替换为6*car\_num\_encoded.doors.replace(**"5more"**, **"6"**, inplace=True)  
*# 将核载人数一列中，more替换为6*car\_num\_encoded.persons.replace(**"more"**, **"6"**, inplace=True)

查看两个属性取值的分布情况。

print(car\_num\_encoded[**"doors"**].value\_counts())

print(car\_num\_encoded[**"persons"**].value\_counts())

可以看到这两个特征的取值分布较为均匀，那么，我们完成数字编码了吗？

答案是：没有！

我们可以查看一下doors和persons特征的数据类型：

print(car\_num\_encoded.doors.dtype, car\_num\_encoded.persons.dtype)

可以看到，这两个特征的数据类型还是字符串类型！我们还需要将它们转化为整型数据。

*# 将车门数量属性类型变更为整型*car\_num\_encoded[**"doors"**] = car\_num\_encoded[**"doors"**].astype(**"int"**)  
*# 将核载人数属性类型变更为整型*car\_num\_encoded[**"persons"**] = car\_num\_encoded[**"persons"**].astype(**"int"**)

再次查看两个特征的数据类型：

print(car\_num\_encoded.doors.dtype, car\_num\_encoded.persons.dtype)

到这里，我们已经完成了除目标特征“acceptance”外，所有特征的数字编码。查看编码后的数据：

car\_num\_encoded.head(5)

有的老师可能就要问了：Python提供了很多工具函数，难道其中没有数字编码的函数吗？这个问题问得好，Python确实提供了数字编码的函数，但这个案例中，我们不能使用该函数。

造成这种情况的原因是本数据集中的离散特征为词序型，编码时需要尽量保持次序关系，以免丢失信息影响模型性能。我们用车的价格“buying”来看看工具函数的编码结果：

*# 创建编码器*buying\_encoder = LabelEncoder()  
*# 适配数据集并完成编码*encoded\_buying = buying\_encoder.fit\_transform(car[**"buying"**])

查看原数据和编码的对应关系：

buying\_encoder.inverse\_transform([0, 1, 2, 3])

可见原本的次序已被打乱，编码过程中丢失了次序信息，这将影响模型的性能。

## 3.3 one-hot编码

One-hot编码将一个k取值的特征转换为k个二元特征。我们可以借助pandas的get\_dummies函数很方便地完成。get\_dummies函数默认使用特征取值来命名新生成的二元特征。因为同一取值可能有多个特征存在，例如buying和maint特征均有取值“med”等。为了区别，我们给新生成的列名加上前缀进行区分。

pd.get\_dummies(car[**"buying"**], prefix=**"buying\_"**).head(5)

为了保持代码简洁，我们用一个循环来完成多个特征的编码。

*# 新建dataframe存储编码后的数据*car\_onehot\_encoded = pd.DataFrame()  
*# 对各个特征依次编码*for col in [**"buying"**, **"maint"**, **"doors"**, **"persons"**, **"lug\_boot"**, **"safety"**]:  
 col\_encoded = pd.get\_dummies(car[col], prefix=col + **"\_"**)  
 car\_onehot\_encoded = pd.concat([car\_onehot\_encoded, col\_encoded], axis=1)

在数据后，加上目标特征列

car\_onehot\_encoded = pd.concat([car\_onehot\_encoded, car[**"acceptance"**]], axis=1)

查看编码后的数据集

car\_onehot\_encoded.head(5)

# 5数据保存

就将one-hot编码后的数据集保存下来，留给以后使用。

car\_onehot\_encoded.to\_csv(**"D:/output/one-hot.csv"**)



之后要使用数据集时，即可直接使用编码后的数据集。

当然，大家同样可以将数据集保存为Excel文件、TXT文件等。