棕榈学院

7 天 Python 进阶训练营讲义

第二讲

2019 年 1 月 14 日-2019 年 1 月 20 日

目录

通过上节课的学习我们对泰坦尼克号的数据有了一定的了解，也对其中一些数据进行了一定的可视化，比如年龄、不同客舱生存情况等。这节课我们要对已有数据进行更多的清理，并且创建一些新的变量供接下来做模型使用。

一、数据导入

首先一开始需要导入在 python 中最常见的包 numpy。之后简化名为 np。

```
In [21]: #载入numpy数据包，以np来简化命名；numpy系统是Python的一种开源的数值计算扩展,这种工具可用来存储和处理大型矩阵.
         import numpy as np

In [22]: #查看训练集的末尾几行，当然Python中，默认打印是5行
         train_df.tail()
```

Out[22]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q

【Title 内容回顾】

Passagerid：乘客的 ID，每个乘客都是不同的。

Survived：乘客最后能否存活下来。

Pclass：客舱的等级，分为 123。1 为最高等级。

Name：乘客名字，其中可以看到会有称呼，小姐、先生等等，接下来会用到名字上的称呼。

Sex：性别，分为男性、女性。

Age：年龄，可以看到有 Nan，表示这个数据是缺失的，缺失数据在机器学习中是经常需要应对的问题，即数据经常不是完整的，所以要对缺失的数据进行填充等等。

SibSp：每一位乘客兄弟姐妹和伴侣的数量，可以看到一般都是 0 和 1 比较多。

Parch：每一位乘客孩子或父母的数量。

Tichket：船票的号码，没有具体的意义在其中，只是像 ID 一样。

Fare：票的价格，在不同地方登上泰坦尼克号、舱位不同，价位也会不同。

Cabin：不同舱位，也可以看到有缺失数据。

Embarked：乘客从哪几个港口登上泰坦尼克号，有三个不同港口，之后也会对这一数据进行利用。

二、数据清理

得到数据后，一般先去看一下每一列缺失多少数据：

```
In [23]: #判断数据集里每一个数据变量的缺失值有多少个，我们可以看到Cabin这个变量的缺失值是最多的；
         train_df.isnull().sum()

Out[23]: PassengerId      0
         Survived         0
         Pclass           0
         Name             0
         Sex              0
         Age            177
         SibSp            0
         Parch            0
         Ticket           0
         Fare             0
         Cabin          687
         Embarked         2
         dtype: int64
```

因为数据的大小一共有 891 行，cabin 缺失了接近 700 个，接近总数值的 80%，在这种情况下基本可以确定不能使用这种数据，因为数据缺失太多，进行人工填充很容易产生很大误差或其他情况。

```
In [29]: # 去掉 Ticket Cabin两列
         train_df = train_df.drop(['Ticket','Cabin'], axis = 1)
```

1. Embarked

因为 embarked 只缺失了两个数据，所以可以用最简单的方法填充进去：

```
In [26]: # 填充Embarked中的missing value
         train_df['Embarked'] = train_df['Embarked'].fillna(method = 'ffill')
```

. embarked 和[‘embarked’]实质上一样，就是指 embarked 这一列数据。

fillna 是指对缺失数据进行填充。填充使用的方法是 ffill，即 forward fill，简单来说是用前一个的数据来去填充这个数据。这只是一个随机的方法，因为这里只缺失了两个数据，所以不再做更深的研究，当然除了这个方法，也可以后面一个数据或前面第二第三个数据来填充，都可以在 Python 中填充。

2. Age

年龄因为缺失的比较多,所以用更高级的方法来填充,这样得到的数据也更加精准。所以这里先看一下每一个年龄层次对应着多少的人：

```
         # 计算Age中各值出现的次数，normalize=Treue表示将结果归一化到[0,1]上
         s = train_df['Age'].value_counts(normalize = True)

In [27]: #train_df['Age'].interpolate(method='linear', inplace=True)
```

使用的是 value_counts （），如果在括号内加上 normalize = true，就不再是原来的数字而是百分比,相当于是相应年龄段的数量除以总数。在接下来会用到百分比。因为这里有 177 个缺失的数据，我们就以每个年龄段具体比例来进行填充。这样得到的年龄的结构的分布就会和原来的几乎是一样的。

有了这个比例后先要找到 missing age 的位置，使用 isnull function：

```
In [28]: # 判断每一个人的年龄数据是否为空
         missing_age = train_df['Age'].isnull()
```

可以看到 False 指并不是空的，True 指是空的。

得到空的 Age 的坐标后，可以使用 numpy function 进行填充：

```
# 此处使用numpy.random的choice方法，以各年龄出现的频次为概率从所有年龄中选取缺失的年龄数目个年龄值
# 将这些随机选中的年龄值赋给缺失的年龄值
train_df.loc[missing_age,'Age'] = np.random.choice(s.index, size = len(train_df[missing_age]),p=s.values)
```

首先找到位置：

loc 指 location；

missing_age 指从横向来看，‘Age’指列；

如果打印出来，即对 177 个数据进行填充。

其次看填充的方法：

填充的方法用到了 np.random.choice：

random.choice 指要进行一些随机的选择。

在进行随机选择的时候，有三个东西要进行定义：

第一个就是 index，就是指上方 s 的 index。

其次 len 就是指长度，即为 177 对，也就是要填充 177 个数据，len 及括号内内容也可以改为 177，效果相同。

最后 p 是指概率，每个里面要填充的概率，而 s.value 是刚刚有的一些 value。

这是一种填充方法，在这里找的是年龄，因为已经存在年龄的分布，所以使用相对简单的方法来填充,如果想做的更细致一些,可以提取出性别和家长孩子数目等等，一同运用进来。比如 Parch 数字大于 2，则说明必定有孩子，那么年龄应该在 20 或 30 岁以上,这样进行填充会更加细致,但在这里先学习运用最简单的方法进行填充。

3. Name

对比数据可以看到，名字这一列有很多称呼，比如先生、女士、小姐等。所以需要对里面的数据进行挖掘。

挖掘数据采用的是正则表达式的方式。

```
In [27]: # 使用正则表达式将Title中的称呼提取出来，如Mr.
         train_df['Title'] = train_df.Name.str.extract(' ([A-Za-z]+)\.',expand = False)
```

加了一列数据 Title。

人名的构成为：名字，称呼. 名字，而正则表达式在这里的作用为提取称呼。

可以用 unique 看究竟有多少个分类。

```
In [28]: # 有的称呼可能出现多次，如 Mr Mrs，此处只显示一次
         train_df.Title.unique()

Out[28]: array(['Mr', 'Mrs', 'Miss', 'Master', 'Don', 'Rev', 'Dr', 'Mme', 'Ms',
                'Major', 'Lady', 'Sir', 'Mlle', 'Col', 'Capt', 'Countess',
                'Jonkheer'], dtype=object)
```

可以看到 Mr. 是男士，Mrs. Miss. Ms. Lady. 是女士，Master 是指少爷公子之类，及其他少见称呼等。

也可以看一下 crosstab。

```
In [29]: # 依据乘客的称呼和最后的生存情况创建交叉表
         pd.crosstab(train_df['Title'], train_df['Survived'])
```

Out[29]:

Survived	0	1
Title		
Capt	1	0
Col	1	1
Countess	0	1
Don	1	0
Dr	4	3
Jonkheer	1	0
Lady	0	1
Major	1	1
Master	17	23
Miss	55	127
Mlle	0	2
Mme	0	1
Mr	436	81
Mrs	26	99
Ms	0	1
Rev	6	0
Sir	0	1

可以看到：少见的称呼数据比较松散，所以对其进行归类，这样模型运用起来会更加简单。

```
In [30]: # 替换称呼来重新赋值Title,如将Lady Countess等出现频次较低的重新命名为Rare
         train_df['Title'] = train_df['Title'].replace(['Lady', 'Countess','Capt', 'Col',\
          'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')
         # 以下作用同上
         train_df['Title'] = train_df['Title'].replace('Mlle', 'Miss')
         train_df['Title'] = train_df['Title'].replace('Ms', 'Miss')
         train_df['Title'] = train_df['Title'].replace('Mme', 'Mrs')
```

先把少见的女士等称呼都换算成 Rare，再把其他一些比较常见的都指一类的称呼归入同一类。

之后可以看到现在只剩下五类，这样训练起来不会过度拟合。

学院君小提示⏰：

请大家注意一个小细节，\是指 Python 中一行代码过长想在第二行继续写用\表明

两行是连着的，在第二行继续写，不是笔误哦～

之后可以看看在分类之后五个不同 Title 的生存率是多少。用到 groupby function。

```
In [31]: # 通过groupby进行分组，选取Title 和 Survived两列，然后使用mean()计算每一种Title的乘客的生还率
train_df[['Title', 'Survived']].groupby(['Title'], as_index=False).mean()
```

Out[31]:

	Title	Survived
0	Master	0.575000
1	Miss	0.702703
2	Mr	0.156673
3	Mrs	0.793651
4	Rare	0.347826

注意！在 Python 中 Ture、False 首字母都要大写，只有变为绿色才表明是正确的。得到新的 feature，在接下来的模型进行应用。

现在有的 feature 都是数据而非数字。但更多情况下要转化成数字，这样模型训练起来更加方便。因为模型背后的理念即为数学计算。
在这里对这一 Title 用一个最简单的 mapping，map 为不同的数字。然后进行对换。其中如果有任何 missing value 直接用 0 来填充。

```
In [33]: # 定义一个字典title_mapping，将原来Title中的值一次映射为字典里对应的值
title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}
train_df['Title'] = train_df['Title'].map(title_mapping)
# 用0来填充缺失值
train_df['Title'] = train_df['Title'].fillna(0)
```

```
In [34]: # 查看训练集的前几行，当然Python中，默认打印是5行
train_df.head()
```

其中产生的问题在于：转换为数字后，计算时候会产生不同数字间的倍数换算，会产生误解，所以并不是最佳方式，但如果数字类型比较少的时候，也是可以运用这一方法。后面会提到一种更加合理的方法进行 mapping。

4. SibSp & Parch

SibSp 表示兄弟姐妹和伴侣数量，Parch 表示父母孩子数量。合并起来就相当于是整个家庭的数量，所以在这里会创建新的变量：FamilySize，直接等同于每一列数字相加。

```
In [35]: # 利用SibSp和Parch创建FamilySize和IsAlone
# SibSp: 兄弟姐妹/配偶的数量
# Parch：父母/孩子的数量
#IsAlone=1表明是alone的状态
#0表示不是
train_df['FamilySize'] = train_df['SibSp'] + train_df['Parch'] + 1
train_df['IsAlone'] = 1
train_df['IsAlone'].loc[train_df['FamilySize'] > 1] = 0
```

因为均为数字且是同样格式，所以可以直接相加。即指每一位乘客登录时整个家庭

的数量。

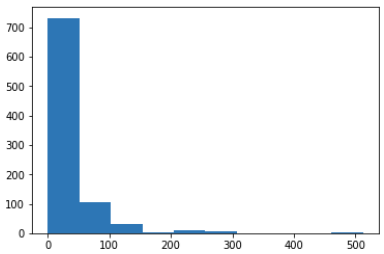
既然是家庭的数量还可以增加一个变量：考虑是否为一个人（Alone）。因为要考虑到一个人牵挂较少是否更容易逃生？所以在这里加了两个变量。

5. Fare

Fare 为票价，先用直方图看其分布：

```
In [38]: # 依据票价绘制直方图
         plt.hist(train_df['Fare'])
```

```
Out[38]: (array([732., 106.,  31.,   2.,  11.,   6.,   0.,   0.,   0.,   3.]),
          array([  0.    ,  51.23292, 102.46584, 153.69876, 204.93168, 256.1646 ,
                 307.39752, 358.63044, 409.86336, 461.09628, 512.3292 ]),
          <a list of 10 Patch objects>)
```



可以看到：大多数票都在 50 元以下，以上很少，极少为更贵的票价。用百分比划分。

特点为：分布比较集中。

所以将其切割为不同的部分，这样可以增加一个变量：用票价已有的数据创建新的变量：FareBin。

```
In [39]: #利用cut把数据用25%，50%，75%进行划分
         train_df['FareBin'] = pd.qcut(train_df['Fare'], 4)
```

这一行具体意义就是指把其分为 4 个区间，相当于对票的等级进行了划分。除了简单数字外又多了一个多的变量：票的等级为一等、两等、三等、四等。

6. Age

用划分 Fare 的方法，可以对 Age 也进行划分。

特点为：分布相对来说更加均匀。

所以将其用数字均等分为 5 份。

```
In [40]: #利用cut把数据分成均等的五份
         train_df['AgeBin'] = pd.cut(train_df['Age'], 5)
```

由于年龄和票价的分布特点不同，所以采用不同方法进行划分。

7. sklearn 和 encode

mapping 的方法很容易在数学中计算的时候公式可能理解为倍数，这是这一方法存

在的问题。解决这一问题常见的方法就是对其从 sklearn 中进行 encode。sklearn 是 python 中最重要的库，接下来所有模型都是在 sklearn 中，preprocessing 就是其中的一部分分支，即预处理，导入的 LabelEncoder 就是对数据进行预处理的方法。和 mapping 相似，但 LabelEncoder 是更加智能的方式，可以进行自动化操作。

```
In [41]: # 调用机器学习库scikit-learn中的LabelEncoder
         from sklearn.preprocessing import LabelEncoder
```

导入LabelEncoder 之后需要确认下哪些需要encode。其中Age、Embarked、Farebin、Agebin 因为是区间的形式，所以无法在训练时候进行运用。

```
In [47]: # 创建一个名为label_encode的LabelEncoder对象
         label_encode = LabelEncoder()
         # 有下列label
         labels = ['Sex','Embarked','AgeBin','FareBin']
Out[47]: list

In [43]: #train_df['Embarked'] = train_df['Embarked'].fillna(method = 'ffill')
```

使用 For 的循环，label 指四个列的名字。定义新 label 的名字，相当于一个新变量。新变量中有什么数字呢？原有的 label。

```
In [48]: # 遍历labels列表
         for label in labels:
             # 输出每一个label和它的数据类型
             print (label,type(label))
             # 新的标签名在原标签名的基础上加上_Code
             new_label = label + '_Code'
             # 调用LabelEncoder的fit_transform方法对原train_df[label]先拟合再标准化
             train_df[new_label] = label_encode.fit_transform(train_df[label])

         Sex <class 'str'>
         Embarked <class 'str'>
         AgeBin <class 'str'>
         FareBin <class 'str'>
```

在这里把很难再进行数学操作的变量转化为数字进行操作。如前所述，1 和 2 很容易让人觉得 2 是 1 的两倍，但比如港口这样的变量很显然不是这种情况。所以还有另一种 encode 的方法，将在下节课进行讲授：对于只有两个的情况结果会是一样的，但如果一个变量中有两个以上可能考虑用到新方法，结果可能会有些不同。现在使用的 labelcode 也是非常常见的把不能处理的变量转化为数字的方法。

操作进行到这里已经得到了大多数想要的数据，对其略做处理：Kaggle 给予的变量如下（已经删除很难或无法使用的变量）：

```
In [55]: original_feature = ['PassengerId','Pclass','Sex','Age','SibSp','Parch','Fare','Embarked']
         new_feature = ['Title','FamilySize','IsAlone','Sex_Code','Embarked_Code','AgeBin_Code','FareBin_Code']
```

包括新加入的 feature。接下来会对已有的 feature 进行训练。

三、数据可视化

所有的变量得到后可以用另一个可视化的包 seaborn。

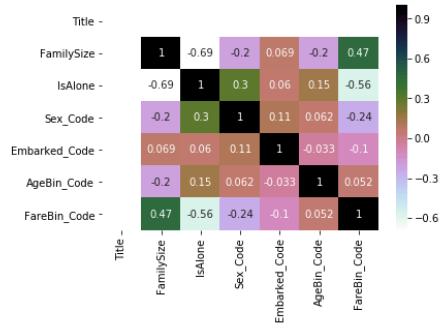
In [51]:
```
#载入seaborn包并以sns简化命名，当你使用数据科学中的Python时，你很有可能已经用了Matplotlib,一个供你创建高质量图像的2D库。
#另一个免费的可视化库就是Seaborn,他提供了一个绘制统计图形的高级接口。Seaborn是比Matplotlib更高级的免费库，特别地以数据可视化为目标，
#Matplotlib试着让简单的事情更加简单，困难的事情变得可能，而Seaborn就是让困难的东西更加简单。
#用Matplotlib最大的困难是其默认的各种参数，而Seaborn则完全避免了这一问题。
import seaborn as sns
```

学院君小提示⏰：Matplot 和 seaborn 可以混合运用，效果加倍哦～

特别介绍 seaborn 中的 Heatmap：

In [56]:
```
# 在上面，将众多属性名聚在一个list命名为new_feature
# 此处使用seaborn的heatmap方法，画出new_feature中各种特征两两之间的相似度热力图
sns.heatmap(train_df[new_feature].corr(),annot=True,cmap='cubehelix_r')
```

Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x1f82f418278>



Heatmap：为两个 feature 之间相关性，数字越大，两个变量相关性越强。
并未包括所有变量，如果感兴趣可以把其他变量放入表中看看哪些相关性更强。

四、作业

根据第二节课上的对训练集数据的处理和创建，请对测试机数据进行同样的处理。

学院君小提示⏰：

鉴于只有训练集和测试集二者变量保持一致，才能最终用来预测等操作。所以希望同学们根据老师的代码对测试集进行操作，看看其中有哪些不一样的结果。特别是要采用 seaborn 中的 Heatmap，可以加上更多变量去对比结果。

扫码关注棕榈学院，解锁更多精彩课程