棕榈学院

7 天 Python 进阶训练营讲义

第三讲

2019 年 1 月 14 日-2019 年 1 月 20 日

目录

在前两节课我们已经对已有数据进行了很多处理，对缺失的值进行创建，还加了一些新的 feature，并对船票价格、年龄的数据进行划分，创建了新的 feature。在经历了这些 feature engineering 之后，接下来进入下一步。

一、更多的数据清理——独热编码

在上节课我们采用了 auto encoder 对 feature 进行编码。这节课首先学习一种新的编码——独热编码。

1. 导入

```
In [308]: # 从scikit-learn中引入OneHotEncoder
          from sklearn.preprocessing import OneHotEncoder
```

sklearn.preprocessing，即预处理中还有很多其他包裹我们可以使用，比如说一些数据如果比较分散我们可以对其进行标准化。

2. 创建对象

```
In [309]: # 创建一个名为enc的OneHotEncoder对象
          enc = OneHotEncoder()
```

3. 需要确定可以进行 OneHotEncoder 的变量:Title、Sex、Embarked、AgeBin、FareBin。

```
In [310]: # 创建一个名为onehot_features的list,并存放要进行编码的属性
          onehot_features = ['Title','Sex_Code','Embarked_Code','AgeBin_Code','FareBin_Code']
```

4. 进行独热编码

```
          # 对这些属性进行独热编码并且求得train_df[onehot_features]的均值等属性
          enc.fit(train_df[onehot_features])
Out[310]: OneHotEncoder(categorical_features=None, categories=None,
                dtype=<class 'numpy.float64'>, handle_unknown='error',
                n_values=None, sparse=True)
```

5. 查看 enc 对象里各个分类编码过后的值

```
In [311]: # 查看enc对象里各个分类编码过后的值
          enc.categories_
Out[311]: [array([1., 2., 3., 4., 5.]),
           array([0., 1.]),
           array([0., 1., 2.]),
           array([0., 1., 2., 3., 4.]),
           array([0., 1., 2., 3.])]
```

变量中有多少个不同的值，就会分成多少份，这就是 OneHotEncoder 所做的事情。

6. 创建一个名为 enc_res 的对象存放经过标准化处理后的独热编码的值

```
In [312]: # 创建一个名为enc_res的对象存放经过标准化处理后的独热编码的值
          enc_res = enc.transform(train_df[onehot_features])
```

一般对象创建后，不仅是这个，将来训练模型也会是一样，先对其进行 fit，相当于对已有数据进行了创建了 OneHotEncoder，接下来 transform 得到的就是结果。

7. 查看结果

```
In [313]: # 查看enc_res的值的情况
          print(enc_res.toarray())
```

```
[[1. 0. 0. ... 0. 0. 0.]
 [0. 0. 1. ... 0. 0. 1.]
 [0. 1. 0. ... 1. 0. 0.]
 ...
 [0. 1. 0. ... 0. 1. 0.]
 [1. 0. 0. ... 0. 1. 0.]
 [1. 0. 0. ... 0. 0. 0.]]
```

因为数据分布稀疏，所以一般会转化为 array，这样看起来更加清楚。

8. 查看数据维度

```
In [314]: # 查看enc_res的数据维度
          print(enc_res.toarray().shape)
```

```
(891, 19)
```

查看维度会有更明确的感受：（891，19），即为 891 行，19 列。

【思考】891 行和 19 列分别该怎么解释呢？
891 行和数据是一致的，原本有 891 行。
19 列是什么意思呢？
本身具有 5 列数据，但是每种分类加起来为 19，所以为 5 列数据中进行变化，变为了 19 列数据。这样变化的好处是什么呢？
最终生存决定如果是函数：survived=ax+by+cz（abc 为常量，xyz 为变量），因为机器学习的模型很多也是用类似的数学表达式创建出来的,所以可以进行这样的表达。这里可以看到。假设 x 是 Title，如果直接用 Title 来计算，就会产生倍数关系的问题，比如 Mr. 为 1，Miss. 为 2，那么 Miss. 会变为 Mr. 的 2 倍，但实际中生存率并不一定是这种倍数关系。所以如果我们进行 OneHotEncoder 的话，x 即变为 x，x2，x3，x4，x5，原函数即变为 survived=by+cz+a1x+a2x2+a3x3+a4x4+a5x5，相当于这几个变量都变成了单独的一个，也有自己单独的常量在其前面，变成了相对独立的 feature，而不是互相有倍数关系。
一般这种独热编码多运用在这种变量种类不是特别多，比如四五个的情况。如果变量种类太多则不好运用，比如一百个不同的选择，得到的结果多且分布稀疏，会对

之后的模型训练造成过度拟合的情况，所以不适宜采用独热编码。
所以在这里对这五个变量进行了 OneHotEncoder。
接下来也会比较一下，进行 OneHotEncoder 和不进行的情况下会不会对最终的模型有所影响。

二、特征选择&数据划分

在进行最后一步——模型训练前，要对所有变量进行最后的选择。

1. **选取不同变量，创建两个 List 来存放 feature（包括已有 feature 和之后创建的新 feature）**

In [315]:
```
# 选取不同变量，创建两个List（分别为原先的特征List和新的特征List）来分别存放特征
original_features = ['PassengerId','Pclass', 'Name', 'Sex', 'Age' ,'SibSp', 'Parch','Ticket','Fare', 'Cabin', 'Embarked']
new_features = ['Title','FamilySize','IsAlone','Sex_Code','Embarked_Code','AgeBin_Code','FareBin_Code']
```

2. **创造 final_features，看看接下来在模型中需要用到哪些 feature，排除不需要的和重复的 feature**

In [316]:
```
final_features = ['Pclass','Age','SibSp','Parch','Fare','Title','FamilySize',
                  'IsAlone','Sex_Code','Embarked_Code','AgeBin_Code','FareBin_Code']
```

因为模型相当于用多个 x 来预测 y 的值，因此将预测定义为 all_x

y 就是最终生存与死亡的情况，1 为存活，0 为死亡。

注意！Title 中大小写很重要，一定不可以有一点点差错。

```
# 删去难以运用的变量&重复的变量
onehot_final = list(set(final_features) - set(onehot_features))
```

In [317]:
```
#打印出onehot_final变量的名称
onehot_final
```

Out[317]: ['Pclass', 'SibSp', 'Age', 'Parch', 'Fare', 'IsAlone', 'FamilySize']

把进行过独热编码的排除掉，这样不会重复。

onehot_features 和 final_features 都是 list 的形式，不能进行相加减，所以在这里转化为一个 set，之后可以进行加减。结果为余下来的新的变量。

减掉之后还有很多新加入的变量。

In [318]:
```
# 将原来数据里的final_features这些列存放到all_x里方便后面使用
all_x = train_df[final_features]
# 引入原先数据'survive'并且赋值y
y = train_df['Survived']
# 查看all_x前几行的信息
all_x.head()
```

Out[318]:

	Pclass	Age	SibSp	Parch	Fare	Title	FamilySize	IsAlone	Sex_Code	Embarked_Code	AgeBin_Code	FareBin_Code
0	3	22.0	1	0	7.2500	1	2	0	1	2	1	0
1	1	38.0	1	0	71.2833	3	2	0	0	0	2	3
2	3	26.0	0	0	7.9250	2	1	1	0	2	1	1
3	1	35.0	1	0	53.1000	3	2	0	0	2	2	3
4	3	35.0	0	0	8.0500	1	1	1	1	2	2	1

3. **用 concat function 将数据拼接起来。这里采用以列的方式进行拼接。**

In [319]:
```
# 将转换为数组的独热编码的值存放到onehot_added里
onehot_added = pd.DataFrame(enc_res.toarray())
# 使用pandas的concat函数将原数据中onehot_final这些列和onehot_added合并起来，concat函数专门用于连接两个或多个数组
# axis指定了合并的轴，此处axis=1意为逐列合并，若axis=0则为逐行合并；合并后的函数赋值为新的数据集all_x_2
all_x_2 = pd.concat([train_df[onehot_final],onehot_added],axis = 1)
```

4. 查看信息

In [320]:
```
# 查看all_x_2前几行的信息
all_x_2.head()
```

Out[320]:

	Pclass	SibSp	Age	Parch	Fare	IsAlone	FamilySize	0	1	2	...	9	10	11	12	13	14	15	16	17	18
0	3	1	22.0	0	7.2500	0	2	1.0	0.0	0.0	...	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1	1	1	38.0	0	71.2833	0	2	0.0	0.0	1.0	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0
2	3	0	26.0	0	7.9250	1	1	0.0	1.0	0.0	...	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
3	1	1	35.0	0	53.1000	0	2	0.0	0.0	1.0	...	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0
4	3	0	35.0	0	8.0500	1	1	1.0	0.0	0.0	...	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0

5 rows × 26 columns

5. 确认是否有缺失值。

In [321]:
```
#确认一下没有缺失值
all_x.isnull().sum()
```

Out[321]:
```
Pclass          0
Age             0
SibSp           0
Parch           0
Fare            0
Title           0
FamilySize      0
IsAlone         0
Sex_Code        0
Embarked_Code   0
AgeBin_Code     0
FareBin_Code    0
dtype: int64
```

这里相当于数据所有都准备完毕。

6. 对数据进行训练集和测试集的划分（非常重要!!!）

【疑问】在最开始不是导入了训练集和测试集吗，为什么还要进行划分呢？

这是因为测试集中并没有最终的结果，所以并不知道如果用训练集对数据进行训练后，无法对测试集进行 fit model，得到结果，很难知道 model 好坏，只能测试到已有的训练集的最终精确结果，所以要对自己的数据进行划分。比如把 20%训练集拿出来当作测试集，当作测试集，当作对这部分数据完全不知道，这样训练自己的模型后再去测试假设完全不知道的结果，才知道模型最终好坏。如果直接看测试集最终精确度的话，模型很容易过度拟合。而且在现实生活中，很多机器学习，都是要去预测未知的东西，要用已有数据来得到这个模型预测未知东西，所以要分出一部分来作为未知的一部分。

（1）从 sklearn.model_selection 中导入 train_test_split。

```
In [322]: # 从scikit-learn中引入train_test_split
          from sklearn.model_selection import train_test_split
```

（2）对 all_x 进行数据集划分

```
In [323]: # 对all_x进行数据集划分为训练集和测试集：xTrain为训练集数据，xTest为测试集数据
          # y为数据集的标签（即该乘客是否存活），yTrain对应了训练集的标签，yTest对应了测试集的标签
          # test_size=0.2表示测试集占总数据集的20%
          xTrain, xTest, yTrain, yTest = train_test_split(all_x, y, test_size = 0.2, random_state = 0)
```

最简单的划分方法是把前 80%数据当作训练集，后 20%当作测试集，但这样导致的结果是数据不一定会均等，所以这里用到 train_test_split，其中的 random_state 是说如果没有 random_state 每次结果会不同，因为是随机分配 20%，如果定义了 random_state 每次结果会保持一致，只要保证每次用同样的数字，得到的结果都会是一样的。

（3）查看训练集和测试集的数量。

```
In [324]: # 查看训练集和测试集的数据量
          xTrain.shape, xTest.shape
Out[324]: ((712, 12), (179, 12))
```

学院君小提示⏰：

在训练模型的时候绝对不能使用 Test 的部分，因为这一部分当成未知的部分，是用来最终检验模型的好坏，所以不可以用在模型中，否则就相当于“作弊”啦～

（4）查看训练集标签数和测试集标签数

```
In [325]: # 查看训练集标签数和测试集标签数
          yTrain.shape,yTest.shape
Out[325]: ((712,), (179,))
```

（5）计算训练集中乘客存活率平均值和测试集中乘客存活率平均值

如前所问，为什么用这样一个模型而不是简单用 80%、20%进行划分呢？

因为希望训练集和测试集中的各种分布都是大致一样的，并不希望训练集中的人可能都存活，测试集中的人可能都死亡，这样的话这个模型会很难进行预测。在进行机器学习的时候，也是预想现在已经得到的数据和未来要预测的数据大致是一样的，才可以进行预测，如果出现很大偏差则无法进行预测。

所以可以查看存活率的平均值。

```
In [326]: # 计算训练集中乘客存活率平均值和测试集中乘客存活率平均值
          yTrain.mean(),yTest.mean()
Out[326]: (0.38342696629213485, 0.3854748603351955)
```

根据结果看到二者平均值相当接近，训练集和测试集的情况比较类似，可以继续进行操作。

（6）同样的对 all_进行数据集划分

```
In [327]: # 同样的对all_进行数据集划分
          x2Train, x2Test, y2Train, y2Test = train_test_split(all_x_2, y, test_size = 0.2, random_state = 0)
```

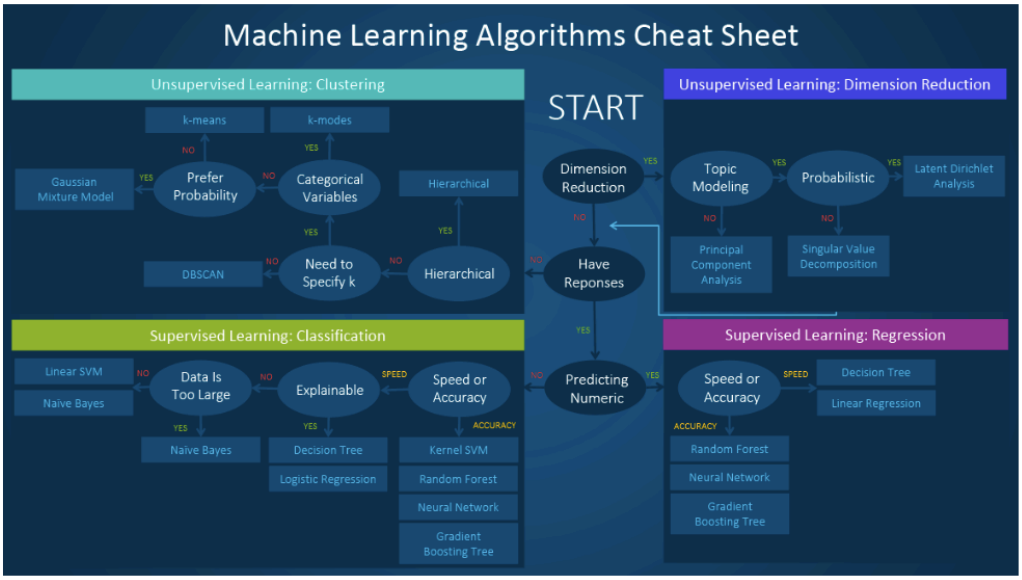
在这里几乎已经得到所有想要的数据，做好了准备。接下来进入模型部分。

三、理解&选择模型

在模型部分，首先要对模型进行选择。

In [328]:
```
# 如何选择适合的机器学习模型，这里的函数目的是给大家引入一个模型图的概括表！这个对于各位有志于从事机器学习和数据分析行业的同学来讲是个大大的
# 在这张表里你可以全面地看到非监督学习和监督学习两种模式下最常用的Model们，收藏好哦！
Image(filename='models.png')
```

Out[328]:



上图即为比较全面的进行模型选择的图片。图片可以分为上下两部分来看，上半部分叫做非监督学习，下半部分为监督学习。非监督学习和监督学习的差别在于是否可以得到明确结果。非监督学习一般不去预测结果，无法得到明确结果。而监督学习则相反。我们要做的是得到最终结果，预测乘客能否幸存下来，所以是监督性学习。

如图，监督学习一般也分为两部分：Classification——分类，Regression——预测，在这个案例中因为只有存活或死亡两类结果，所以使用 Classification。

Classification 又分模型的数据大小等问题，根据不同情况有三类可以进行选择：

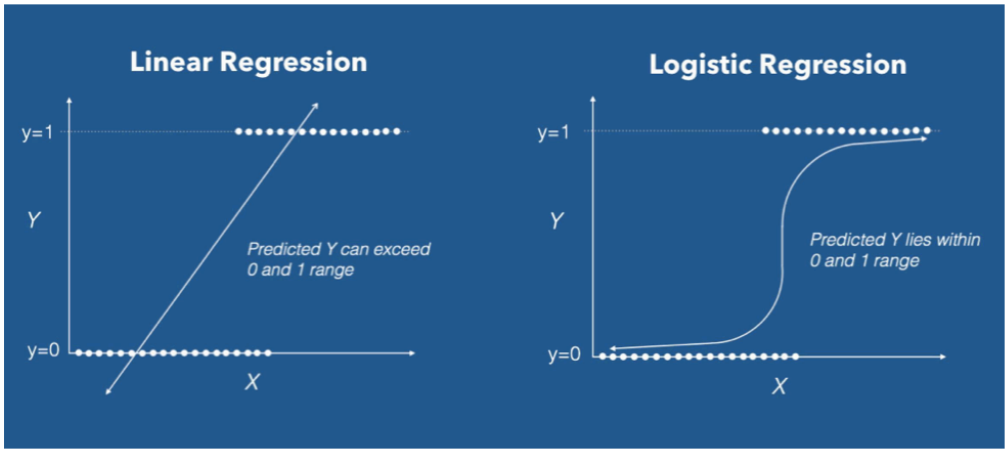
- 如果想要追求精确度可以选用：Kernel SVM, Random Forest, Neural Network, Gradient Boosting Tree.
- 如果想要追求速度、结果又好理解的话，可以选用：Decision Tree, Logistic Regression.
- 其他的话就用：Naive Bayes.

泰坦尼克号这一案例数据量不是很大，不用担心速度问题，但是同时想要结果能够更好的展示，所以在这里用到三个比较主流的模型：

1. Logistic Regression

```
In [329]: # logistic回归的例子，左边是线性回归的模型可视化呈现，右边是逻辑回归的模型可视化呈现
          Image(filename='logr.jpg')
```

Out[329]:

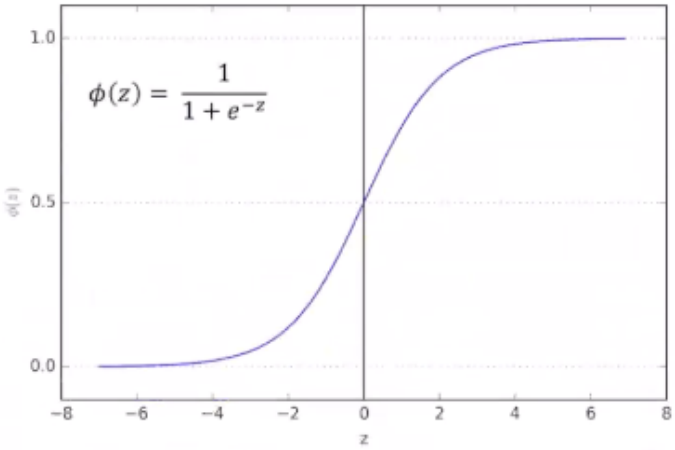


Logistic Regression 是从 Linear Regression 演化而来。

大部分结果都是在 0 或 1，只有很少的一部分在中间，是很好的一种进行二元分类的模型。因为如果数据非常大直接归类为 1，特别小直接归类为 0，非常容易理解。

```
In [123]: 1 Image(filename='logist regres.png')
```

Out[123]:



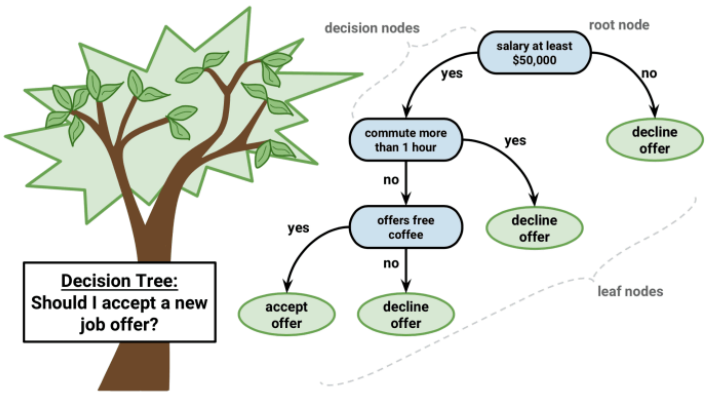
e 是自然常数，z 是变量所涵盖的东西，是前面所提到过的 by+cz+a1x+a2x2+a3x3+a4x4+a5x5。如果 z 特别大，结果接近于 1，如果 z 特别小，结果接近于 0，这是本图像的由来。

2. Decision Tree

用简单的图来展示一下决策树的逻辑：

In [330]:
```
# 决策树的例子，决策树也是用于预测领域非常普遍的一个模型
Image(filename='decision.png')
```

Out[330]:



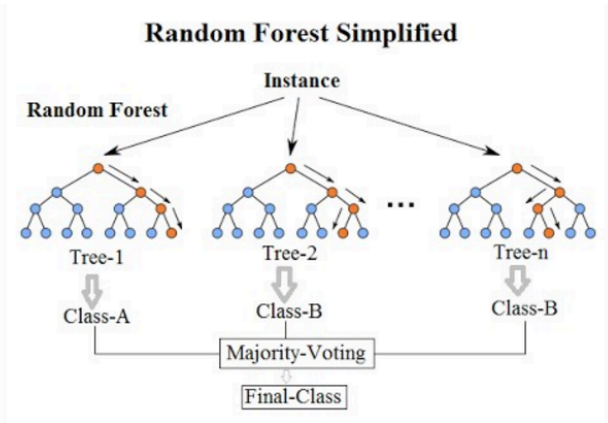
将其运用到我们的模型中，每一个变量都会到树中，逐级进行划分，最终得到不同结果。

决策树的优点：可以直接进行训练，不需要转化为数字。而逻辑回归模型中因为是数学公式，所以所有变量都要转化为数字才能进行训练。

3. Random Forest

In [331]:
```
# 随机森林的例子
Image(filename='random.png')
```

Out[331]:



随即森林与决策树很像，但随机森林相当于每次从所有数据中选取一部分，是比较随机的选取一部分数据，进行小的决策树。每次都创建很多个小的决策树，每个小的决策树的变量都会不一样,然后进行单独训练,训练后将所有结果拼接整合起来，得到最终结果。在很多情况下决策森林是可以得到更好的结果的方式，是一种训练比较快、结果比较好的模型。

在下节课会运用到这三种模型对数据进行预测，让我们一起来看看在做了这么多数据预处理后，结果到底会怎样，敬请期待哦～

四、作业

我们在 train/test split 之后比较了 yTrain 和 yTest，知道了他们的分布是一样的，同学回去可以自己去检查了一下 xTrain 和 xTest 中的变量是否分布大致相同，看看有没有哪个变量偏差比较大？