# Kaggle101 titanic report

之前看了不少项目和代码，其中看的理论方面的东西较多，动手较少，导致实际操作能力不够，因此以后需要多写代码，多写项目报告，这样才能提升工程能力。

该项目的目的是通过ML对Titanic数据进行建模，找到哪一类人或者说是那些特征可以使遇难者具有较高的生存率，即输入是船上游客的各种特征（已有特征或构造特征），输出是这些人的生存概率，并且使用一个sigmoid函数将最后的输出映射到0或1（即二分类）。

根据<https://www.kaggle.com/startupsci/titanic-data-science-solutions> ，首先需要了解一些project的背景知识，这点有点类似咨询行业中的分析框架建构，比如“在1912年Titanic在处女航中就遇到冰山撞击事件，导致2224乘客和船员中1502人死亡——》也就是32%的生存率（类似于咨询里面强调的不仅需要知道绝对值，还需要知道比例和trend）”，导致如此严重的死亡率的原因之一是“没有足够的lifeboatd”给乘客和船员使用，但是女人、小孩和上层人士还是相对的具有更高的生存率。

# 数据

数据集分为train和test，其中train用来训练模型，所以这部分数据是已标注数据，即有正确答案lable的数据。

数据字段：1）survival：是否生存，0=NO，1=Yes；2）pclass：船票的等级，分为1、2、3等船票（类似于机票中的头等舱、经济舱的概念）；3）sex：性别；4）Age：年龄；5）sibsp：在船上的兄弟和配偶数量，即siblings+spoused；6）parch：在船上的父母和子女的数量，即parents+children；7）ticket：船票号码；8）fare：船票价格；9）cabin：船舱号；10）embarked：上船的港口，取值分别为C（cherbourg）、Q（Queenstown）、S（Southampton）

以上是原始数据形式，可以看出其中y为survival，因此理论上x包括9种，其中变量分为categorical和numerical两种，numerical可分为离散和连续，（PS.不过这里我对离散和连续的看法感觉和kernel中有些不一样）categorical又可以分为ordinal和category，也就是序数变量和种类变量，区分这些变量对后面进行数据编码和选择可视化方案有帮助。在数据预处理的时候可以进行空缺值处理、离散变量哑编码（dummy）、feature engineering等。

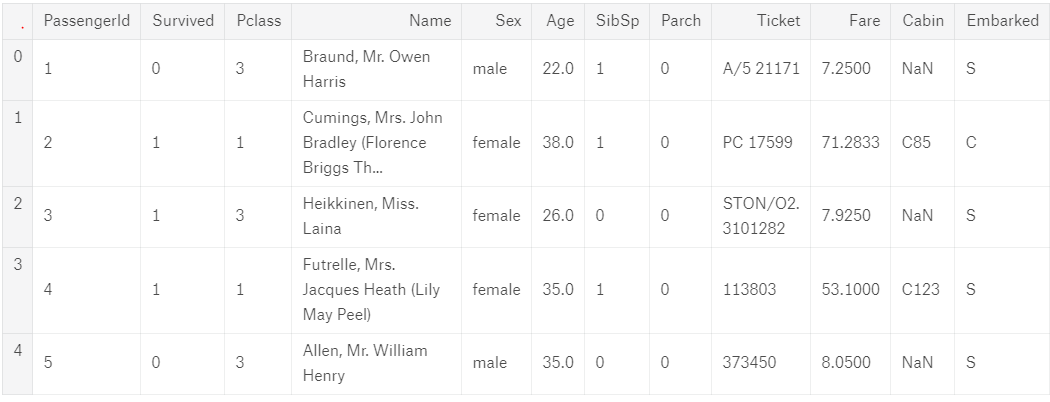
 [[1]](#footnote-1)

# 数据预处理&数据探索（EDA）

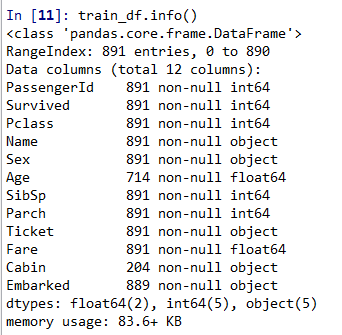
这部分可以包括处理缺失值、查看feature的相关系数、特征工程等，但其实这部分和EDA部分有时重合起来，可视化在所有部分都可能使用。

该部分通过可视化和简单的统计指标查看数据的特点，并进一步针对数据的特点构建后文的模型。

一般可以先使用train\_df.columns.values查看列名（也就是特征名），或者直接train\_df.head()查看前5行数据，具体看一下数据内容，这时可以看一下有没有缺失值、数据格式有没有统一等问题，其实就是统一编码。比如train\_df.head()之后如下所示，可以发现Ticket有字母和数字混合（是否处理？）、Cabin有NaN（怎么处理？删除还是填补？）、Name有Mr等体现一定身份背景的标识（是否需要文本提取？怎么提取？提取家族信息？或者其他的文本处理方法）、Fare的小数部分不一致（是否直接切分区域或者是重新确定近似值？）



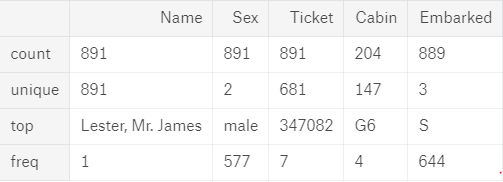
使用train\_df.info()可以查看train\_df的类型等信息，见下图



经过整体的查看之后，一般会进行简单的统计，可以使用train\_df.describe()函数，见下图。这时候需要将数字进行一些翻译，比如“Most passengers (> 75%) did not travel with parents or children.”，比如survived rate为38%。值得注意的是，直接使用dataframe的describe（）函数只对数值型数据进行统计。



查看category类型的统计信息需要使用train\_df.describe(include=['O'])，可以发现Name没有重复的，Sex只取两种值，top一行代表对应特征频次最高的取值，比如male的值有577条，在Sex中最高，这时可以简单计算male的占比为577/891



## 特征初步判断及假设

在进行info（）、describe（）等操作查看大概的数据信息之后，可以进行一些简单的特征选择操作，比如计算特征与survival的相关系数；Age有NaN，但因为根据常识，Age与survival有关（比如小孩优先？），因此需要填补Age的NaN；Embarked也存在NaN，也需要填充（登岸地点与survival有关，也许是因为不同的登岸地点会统一分配在船的某一位置，而这个位置可能会与危险相关）；Ticket可以删掉，因为存在很多重复的值（unique的值有681个），而且与survival不太相关；Cabin删掉，因为在train和test中都存在很多NaN；PassengerID就是一个认为的乘客编号，与survival无关，删掉；Name因为高度的不规则，而且也许与survival不相关，可以删掉（这是<https://www.kaggle.com/startupsci/titanic-data-science-solutions> 的观点）。

除了数据集中给出的特征，还可以对特征进行一些归并或者构造，比如构造特征Family，即船上家庭成员总数，Family=ParCh+SibSp；对Age和Fare进行范围切分，变为ordinal类型变量；从Name中抽取title（比如Mr）。

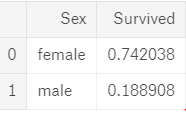
## 数据透视

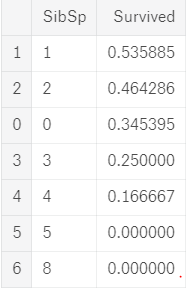
对特征和survival进行简单的数据透视分析，也就是判断特征对survival是否存在足够的区分度，之前的是一个直观的判断，这里就是使用统计数据来确认之前的判断是否正确。

train\_df[['Pclass','Survived']].groupby(['Pclass'],as\_index=False).mean().sort\_values(by='Survived',ascending=False)，该造作类似sql，结果如下，可以认为Pclass对survived的区分还是好的，可以作为feature。（PS.如果想慢慢查看特定feature X与survived的关系，可以使用train\_df[[‘X’,’Suvived’]].groupby([‘X’],as\_index=False).mean().sort\_values(by=’Survived’,ascending=False)，所以说coding就是一个机械的套路，多多训练即可）



下面几个结果使用同上操作







## 数据可视化

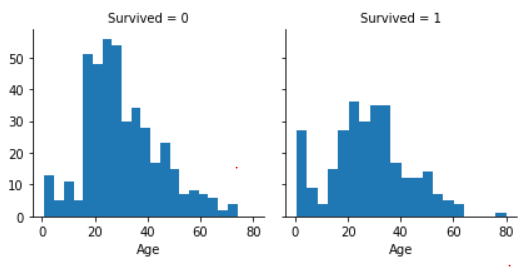
数据可视化的目的可以认为有两种：一是检验之前对feature的假设，二是可能发现新的feature。

对Age进行histogram可视化，可以查看在对应类别中Age的分布，其中bins是直方图中柱子的个数，可以认为是柱子的细化程度，比如数据范围为0-80，取bins=20，则一个bin代表的数据跨度为（80-0）/20。结果见下图，横轴是Age，纵轴是样本个数（#passenger）。从直方图可以发现至少一下信息：1）80岁的游客都活下来了（对比两图最右端）；2）infant有较高的存活率（Age<4，也就是图的最左端第一个柱子）；3）大部分游客年龄在15-35之间，有较多15-25部分的人死亡——》其实可以认为看直方图可以看大部分的区域、极端少部分区域，两极才可以发现区分度强的信息

根据上面对hist图的分析，可以判断：1）应该使用Age特征，并且需要填补空缺；2）可以对Age进行分段

g=sns.FacetGrid(train\_df,col='Survived')

g.map(plt.hist,'Age',bins=20)



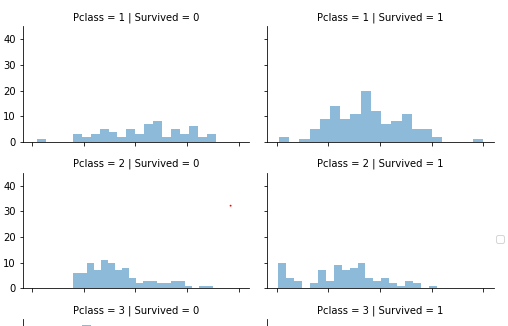
还可以更加组合的进行可视化，比如使用categorical类型的变量来组合分类，对numerical进行统计可视化，如下所示为使用Pclass和Survived两个种类变量进行分组，之后对数值变量Age进行统计。

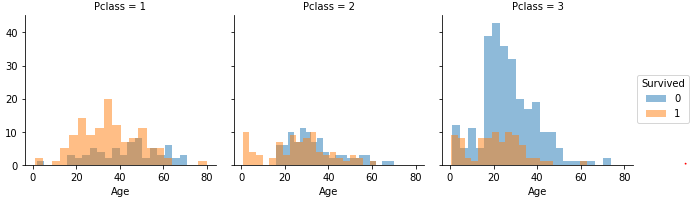
*# grid = sns.FacetGrid(train\_df, col='Pclass', hue='Survived')*

grid = sns.FacetGrid(train\_df, col='Survived', row='Pclass', size=2.2, aspect=1.6)

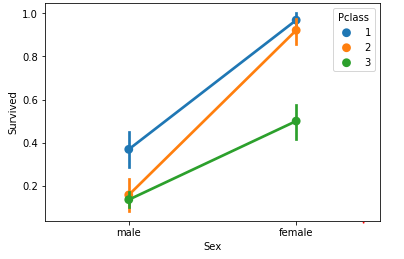
grid.map(plt.hist, 'Age', alpha=.5, bins=20)

grid.add\_legend();

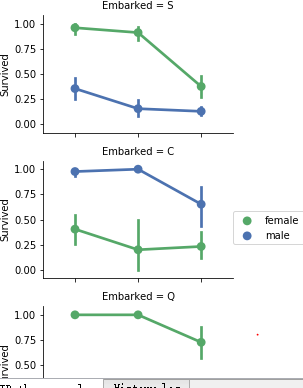




如果需要都对categorical变量进行统计分析，则可以使用pointplot图，即分面的点图[[2]](#footnote-2)，主要就是看不同类别的趋势变化。例如sns.pointplot(x="Sex", y="Survived", hue="Pclass", data=train\_df)结果如下图所示：



下图是例子中完整的可视化结果，关键是解读可视化，可以发现除了C port的结果，其他都是female生存率高于male。其实就是从总体分析和从每个变量具体讨论分析。

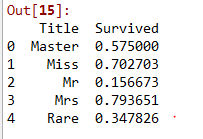
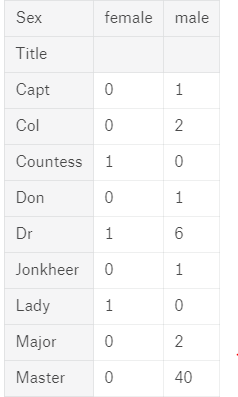


类似这样的可视化还可以做很多，具体参见kernel[[3]](#footnote-3)。

## Wrangle Data

这一步会删除一下原始特征并构造新特征，首先删除的是Cabin和Ticket，之后删除Name和PassengerID，但因为可以从Name中提取出Title，因此需要从Name中构造出title的特征。

Kernel中使用了一个交叉表，类似于多次的分组之后进行统计，是数据透视表的一个更加简单的形式，具体参见[[4]](#footnote-4)[[5]](#footnote-5)。在使用交叉表统计频次之后，发现一些title频次很少，因此将这些频次低的title合并为rare，之后再分组查看结果。这里使用了replace和groupby函数



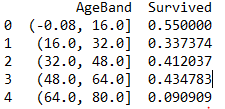
## 数值型变量NAN填充

这里主要是对Age变量进行填充，对数值型变量填充一般有三种思路：1.利用整体的mean和std选取随机数；2.分组之后选取组内的median值；3.结合其他信息分组之后再利用组内的mean和std选取随机数。因为方案2是能够保证确定值，所以kernel中使用了方案2。

对Age进行填充之后需要对Age进行一个范围的切片，使用pd.cut()函数。

train\_df['AgeBand']=pd.cut(train\_df['Age'],5)

train\_df[['AgeBand','Survived']].groupby(['AgeBand'],as\_index=False).mean().sort\_values(by='AgeBand',ascending=True)



此时AgeBand变量的类型是pandas.\_libs.interval.Interval，需要将对应的Band转换为序数型。使用的是列表递推式或者说是布尔表达式。

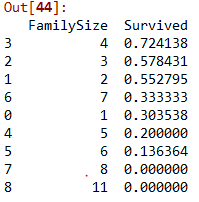
## 构造新特征FamilySize

FamilySize=SibSp+Parch。

For dataset in combine:

Dataset[‘Family’]=Dataset[‘SibSp’]+dataset[‘Parch’]+1#因为自己也是family中的一个

Train\_df[[‘Falimy’,’Survived’]].groupby([‘Family’],as\_index=False).mean().sort\_values(by=‘Survived’, ascending=True)

结果如下：  


还可以根据FamilySize构造一个IsAlone特征：

For dataset in combine:

Dataset[‘IsAlone’]=0

Dataset.loc[dataset[‘FamilySize’]==1,’IsAlone’]=1

结果如下，可以认为平均而言，非Alone的游客生存率比Alone的游客高0.2



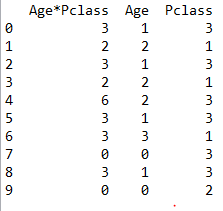
Kernel中最后也将FamilySize删去，只保留了IsAlone，值得注意的是drop的时候需要从train\_df和test\_df中删，不能从combine中删，原因未知，具体见代码备注。

此外，还可以构造一个“Age\*Pclass”的特征，也就是Age和Pclass的值相乘，注意，在往combine或者说是train\_df和test\_df中添加变量的时候可以使用for x in combine操作，自动影响到train\_df和test\_df，但之前的drop不行。

for dataset in combine:

dataset['Age\*Pclass']=dataset['Age']\*dataset['Pclass']

train\_df.loc[:,['Age\*Pclass','Age','Pclass']].head(10)



## 填补categorical变量

对Embarked进行填补，使用的是频次最高的Embarked值，这里有一个新使用的函数：pandas的mode（）函数[[6]](#footnote-6)，也就是给出dataset的众数，mode（）[0]表示如果存在很多相同频次的众数，则选取第一个众数。

但是因为ML model输入都输需要是数值型，因此对于文本型或者说是category类型的变量都需要将其转换为numerical，即自己定义映射dict。

So，coding这种死东西就是套路，多多traininig，不断training！

KTA&TETG！

## 处理剩下的numerical feature

首先填充直接使用median值

然后可以将Fare也切分为FareBand，但值得注意的是这里使用的是qcut而不是cut，之前切分Age的时候使用的是cut，二者的区别在于根据不同的标准确定间隔，cut根据值进行切分，qcut根据频次进行切分。但是两个函数的形式都是（q）cut（dataset，#band），具体参见[[7]](#footnote-7)。

到目前为止都是处理数据和处理特征，因此这是数据分析最麻烦的一步，没处理一次都使用DataFrame.head()来查看一下数据结果，目的就是总要知道数据处理怎么样了。

## 数据预处理和特征工程总结

使用的方法一般是可视化、统计等，首先先观察数据，将数据区分为数值型和种类型，然后数值型里面又区分为离散型和连续型等，观察一下有没有什么明显的数值错误（结合常识和直觉）、观察一下有没有NaN，Nan导致的问题严重不严重（这部分就需要考虑将那些feature保留下来，可以只用一个简单的分组统计或者是计算相关系数等），如果NaN导致的问题很严重，则需要想办法进行填充。

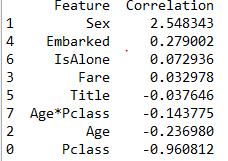
简单来说就是考虑将那些feature保留，然后怎么填充存在NaN的特征，根据特征的类型选择填充方法，比如数值型可以选择median、mode等，而种类型可以选择mode。

构造新特征，本文尝试了将SibSp+Parch=FamilySize；从Name中使用extract（）和【（A-Za-z）+.】提取title，并且又对Title进行统一格式和归并为Rare；还有一些其他构造的特征，比如Age\*Pclass。

# 建模分析Modeling

经过数据预处理和特征的构造，之后进行分类。

首先使用logisticregression，值得注意的是kernel中取出了logistic回归方程中的每个feature对应的系数，如下所示，结构与kernel给出的结果不一样。因为目前已知答案只有在train\_df中，因此计算accuracy的时候仍然使用train\_df计算，logistic准确率大概在80

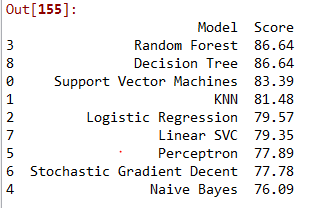


之后用SVM，准确率在83左右；k=3的knn准确率在81；gaussian NB准确率在76perceptron准确率在77；Linear SVC准确率为79；SGD分类器有意思的是每次run的结果都可能不一样，最差一次run的结果准确率只有66，最好的可以达到77；决策树准确率达到86.64，而且似乎结果很稳定，每次运行accuracy都是同样的；随机森林准确率也在86.64。

从准确率来看，虽然DecisionTreeClassifier和RandomForest结果一样，但是因为随机森林还可以减少过拟合问题，因此选择随机森林作为最后的分类器。

# 评价

使用accuracy，即true predict/total number of passengers，在kaggle中评测部分，其中50%的predict在public上进行，剩下50%在private上进行，可以认为这种操作目的是减少过拟合问题。但是在自己评测的时候，因为只有train有答案，所以计算的score仍然是在train中进行。



# Project总结

CS的东西其实就是一个多练的问题，多做项目，多写代码，多写报告即可。

该项目较好的进行了问题分析、数据预处理、特征构造、分类器选择等步骤的展示，但对于自己个人的项目可能还需要设计数据爬取等更加前期的工作，所以完整部分应该集成到论文中。

记住使命！

1. <https://www.kaggle.com/startupsci/titanic-data-science-solutions> [↑](#footnote-ref-1)
2. <https://zhuanlan.zhihu.com/p/27683042> <https://zhuanlan.zhihu.com/p/27816821> [↑](#footnote-ref-2)
3. <https://www.kaggle.com/startupsci/titanic-data-science-solutions> [↑](#footnote-ref-3)
4. <https://blog.csdn.net/hustqb/article/details/78086394#%E4%BA%A4%E5%8F%89%E8%A1%A8crosstab> [↑](#footnote-ref-4)
5. <https://blog.csdn.net/AlanGuoo/article/details/52330404> [↑](#footnote-ref-5)
6. <https://segmentfault.com/q/1010000008252098> [↑](#footnote-ref-6)
7. <https://blog.csdn.net/a787264137/article/details/78573436> <https://blog.csdn.net/cc_jjj/article/details/78878878> [↑](#footnote-ref-7)