**Kaggle titanic 机器学习流程分析top30%**

题目：<https://www.kaggle.com/c/titanic>

Github地址：<https://github.com/cqychen/mykaggle/tree/master/titanic>

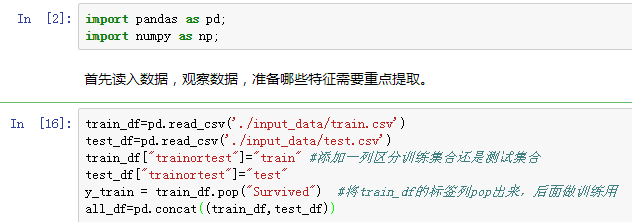
# 数据探探

## 数据总体分析

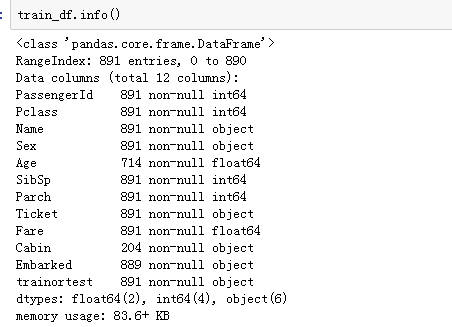
首先看看网页上的介绍：

<https://www.kaggle.com/c/titanic/data>

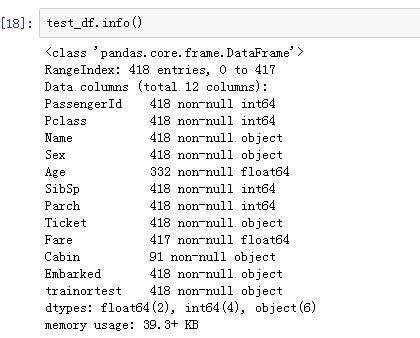
|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Definition | Key | 中文 |
| survival | Survival | 0 = No, 1 = Yes | 是否生还(0:没有生还，1：生还) |
| pclass | Ticket class | 1 = 1st, 2 = 2nd, 3 = 3rd | 票类别分三等，1等、2等、3等 |
| sex | Sex |  | 性别 |
| Age | Age in years |  | 年龄 |
| sibsp | # of siblings / spouses aboard the Titanic |  | 在船上的兄弟姐妹配偶数 |
| parch | # of parents / children aboard the Titanic |  | 在船上的父母子女数 |
| ticket | Ticket number |  | 票编号 |
| fare | Passenger fare |  | 票价格 |
| cabin | Cabin number |  | 客舱号码 |
| embarked | Port of Embarkation | C = Cherbourg, Q = Queenstown, S = Southampton | 登船港口 |
|  |  |  |  |
| PassengerId |  |  | 乘客id |
| Name |  |  | 乘客姓名 |



查看训练集数据情况：



查看测试集数据情况：



可以看到提供的特征有12列，其中age 和 cabin确实严重。

将训练集合测试集合并之后，查看缺失数据：

PassengerId 1309 non-null int64

Pclass 1309 non-null int64

Name 1309 non-null object

Sex 1309 non-null object

Age 1046 non-null float64

SibSp 1309 non-null int64

Parch 1309 non-null int64

Ticket 1309 non-null object

Fare 1308 non-null float64

Cabin 295 non-null object

Embarked 1307 non-null object

trainortest 1309 non-null object

代码：all\_df.columns[pd.isnull(all\_df).sum()>0]

Index(['Age', 'Fare', 'Cabin', 'Embarked'], dtype='object')

发现 有四列是有null的

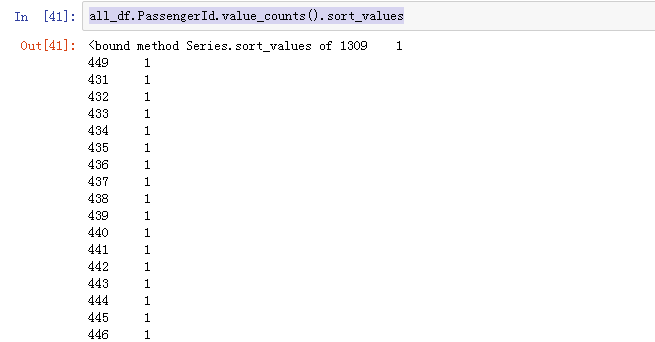
每列进行分析：

## PassengerId

PassengerId 用户的id号，好像这个没啥用

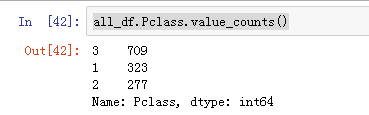
all\_df.PassengerId.value\_counts().sort\_values

得到结果：

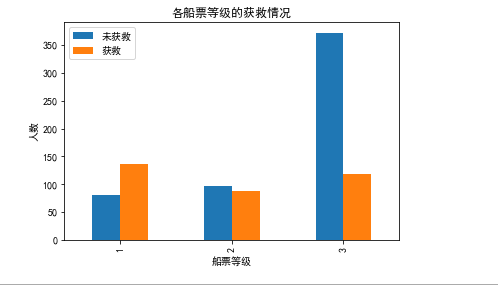


这是一个唯一的主键，现在看来没啥用。有的情况下我们可以通过聚合用户id得到一些分布，然后进行独热编码，比如同一个用户的行为，同类型的用户的行为。这里先抛弃吧，或者可以采用聚类的方式试试？

## Pclass



用户的分布情况如下，对比下和标签的关系：

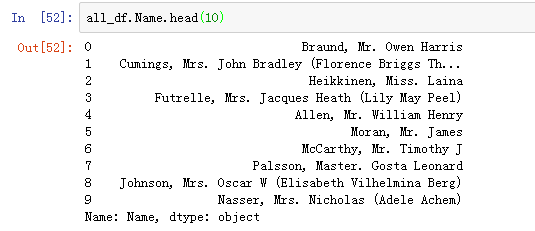


可以看到船票等级高的获救概率越大哈。

这个可以作为一个good feature。

## Name

名字这个特征，按理说名字这个东东肯定没啥用，不过我们看看数据：



这个名字不是单纯的名字哈，包含了称呼：Mr,Mrs,Master等等

代码：

import re

import numpy as np

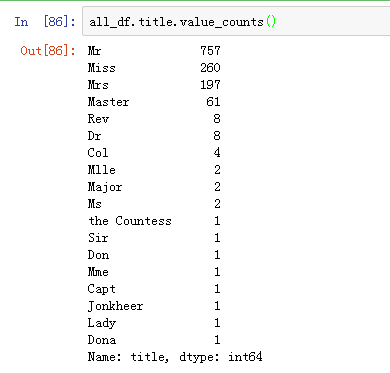
regx = re.compile('(.\*, )|(\\..\*)')

title=[]

for name in all\_df.Name.values:

title.append(re.sub(regx,'',name))

all\_df['title']=title



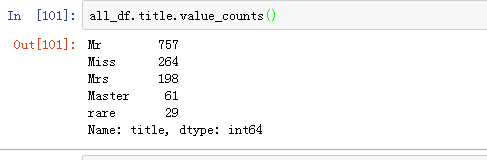
直接这样用呢，肯定是不行的啦，因为你独热编码之后，那种频率很低的，咋整？

Lady /Mile /Ms 这个可以放入到Mrs 嘛，就是称呼的别称嘛。

这样处理下。

像Capt 稀有动物就单独组合喽。

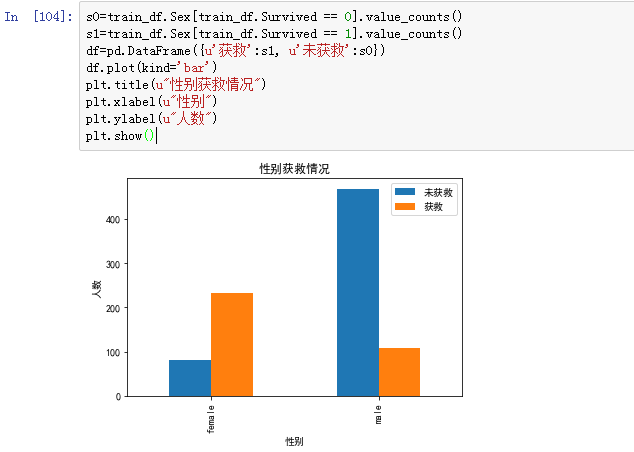
整理之后的结果如下：



还可以提取姓氏，后面可以优化的时候做。

## Sex

查看sex和获救的情况



可以明显看出，女性获救概率远远超过男性。一个好的特征。

## Age

Age 1046 non-null float64

年龄这个字段缺失的比较严重，1309条记录，只有1046条记录有值。

对于缺失值，我们有如下三种处理方法：

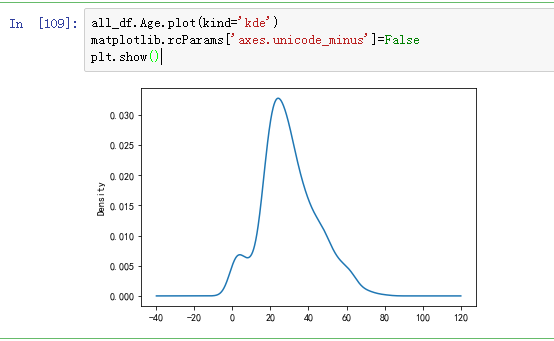
1. 中值、平均值、众数等进行填充
2. 通过模型将已有的一些因素进行预测。
3. 将缺失值作为一种类型编码

这三种方法一种比一种精确。

这这里我们采用2）进行预测填充。

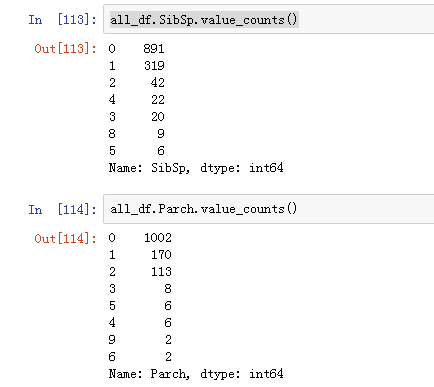
先要分析其他变量，这个放在特征工程中进行处理。

不过可以看看年龄和获救情况的密度图



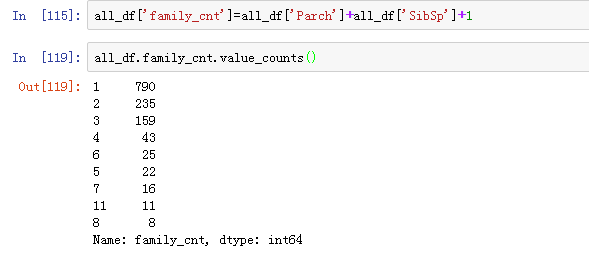
## SibSp 和Parch

兄弟姐妹和父母孩子就是一个大家庭。



这两个字段可以构成大家族和小家庭

大家族人数= SibSp+Parch+1



这个我们还是没法直接用的，和那个称呼一样，占比小的我们需要进行归类。

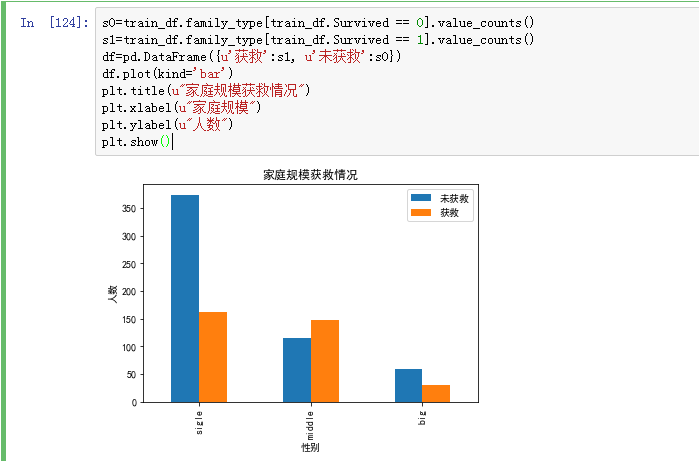
可以分成

一个人：1

小众 ：2~3

大众： 4以上

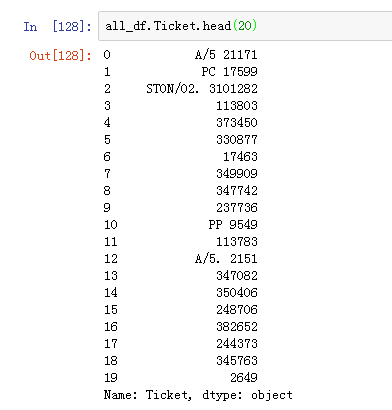
同时我们查看家庭规模和获救情况对比：



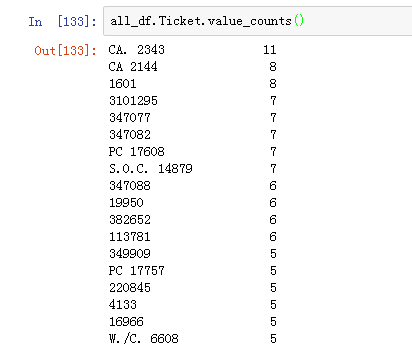
发现家庭规模确实是一个比较很有用的特征。

## Ticket

船票号码：



这个貌似没啥用啊，除非知道其他信息，可以将船票进行转换，比如根据船票关联。

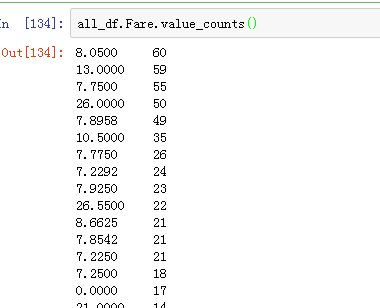


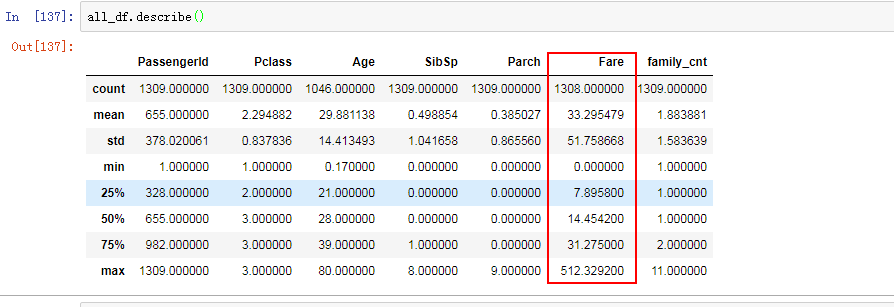
船票的统计情况，发现可能用处不大，即使进行独热编码会造成维度爆炸。

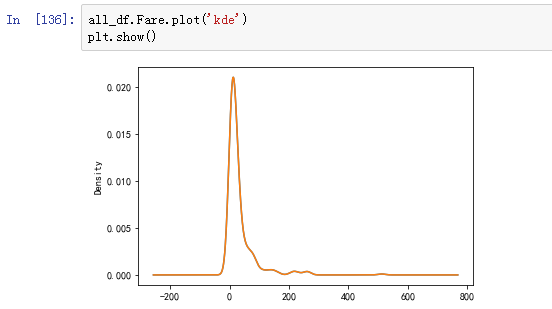
## Fare

船票费用。

查看样例数据







可以看到这个数字最大最小相差很大，均值33，方差有51。比较离散。

票价还缺了一个。

这个样子的数据直接用中位数填充，因为比较离散，中位数填充比较好。就像北京的平均工资都9000多了，中位数呢？

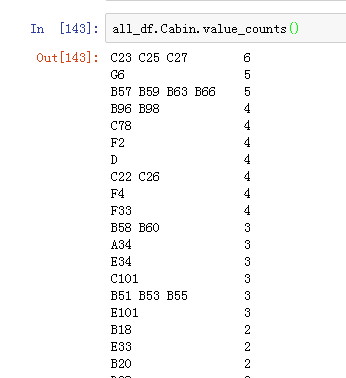
all\_df.loc[all\_df.Fare.isnull(),'Fare']=14.4542

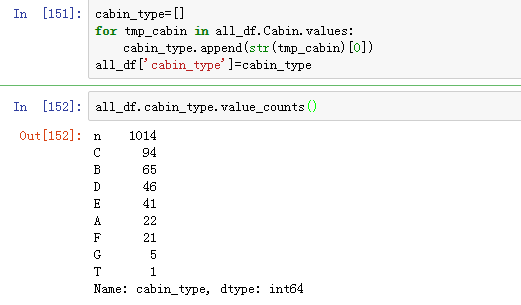
## Cabin

这个特征明显的感觉到缺失值严重哇

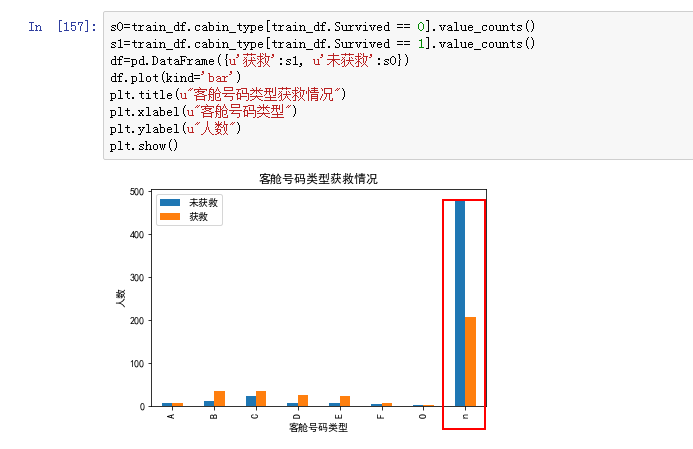
Cabin 295 non-null object

这个还是有些规律的，比如貌似可以将第一个大写字母一起的归一类。





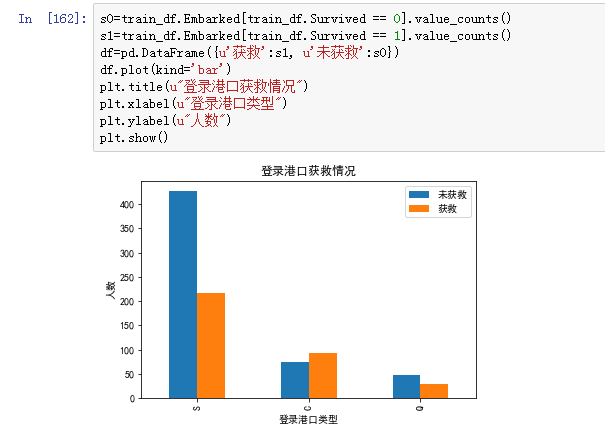
这里将G和T划到其他吧。只有几条数据，模型基本无法学到任何东西。



发现这个也是个不错的特征也。啦啦啦啦

## Embarked

这个特征也有缺失，不过缺失的很少，直接用众数填充就好了。



这个特征也不错，留着。

# 特征工程

根据数据探探我们可以得到以下结论：

1. PassengerId ：用户id不需要啦，或者后面可以做做聚类瞅瞅
2. Pclass： 就先独热编码
3. Name：名字提取了称呼，后面特征挖掘也可以挖掘姓氏。
4. Sex：性别独热编码
5. Age：年龄需要采用模型插入
6. SibSp和Parch:家庭成员可以划分大家庭和小家庭，可以看出获救情况和该特征不是线性关系，后面或许可以做做kernel版本。
7. Ticket:船票这个和用户id一样 ，没啥用啦
8. Fare:费用，这个船票还算集中，但是最大和最小差异比较大，采用中位数进行填充。后面可以将该值进行类别化，改成：屌丝、中产、大富豪。
9. Cabin:这个缺失值严重，但是本身也有些规律，提取出来。一个乘客貌似也有多个房间，可以进一步挖掘

10）Embarked：这个缺失了值，因为是类别型的，同时缺失值少，用众数进行填充。

Ok我们现在对上面的点一个个进行特征工作：

## 归一化

### 连续值

数值型进行标准变换。

import sklearn.preprocessing as preprocessing

scaler = preprocessing.StandardScaler()

age\_scale\_param = scaler.fit(all\_df[['Fare']])

all\_df['Fare\_scaled'] = scaler.fit\_transform(all\_df[['Fare']], age\_scale\_param)

age\_scale\_param = scaler.fit(all\_df[['family\_cnt']])

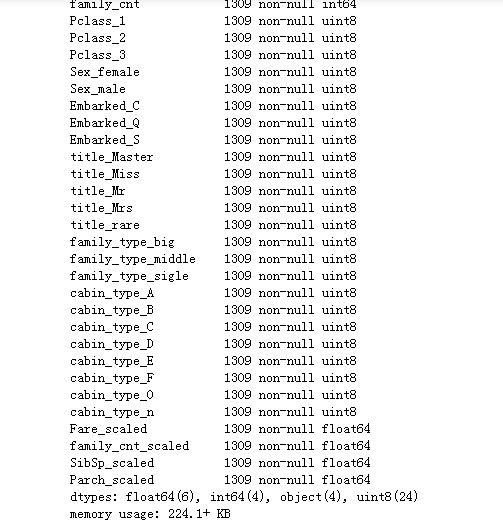
all\_df['family\_cnt\_scaled'] = scaler.fit\_transform(all\_df[['family\_cnt']], age\_scale\_param)

age\_scale\_param = scaler.fit(all\_df[['SibSp']])

all\_df['SibSp\_scaled'] = scaler.fit\_transform(all\_df[['SibSp']], age\_scale\_param)

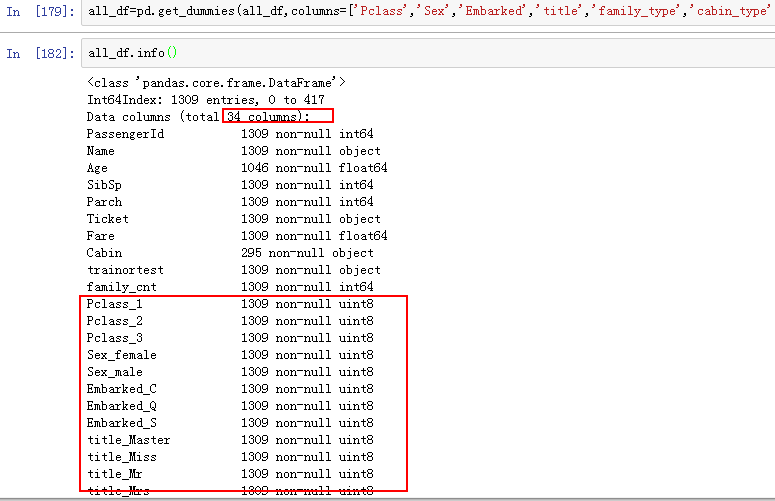
age\_scale\_param = scaler.fit(all\_df[['Parch']])

all\_df['Parch\_scaled'] = scaler.fit\_transform(all\_df[['Parch']], age\_scale\_param)



### 类别值

all\_df=pd.get\_dummies(all\_df,columns=['Pclass','Sex','Embarked','title','family\_type','cabin\_type'])



这个变成了一个稀疏的矩阵。

## 随机森林填充年龄

代码：

age\_factor=[ 'Pclass\_1', 'Pclass\_2',

'Pclass\_3', 'Sex\_female', 'Sex\_male', 'Embarked\_C', 'Embarked\_Q',

'Embarked\_S', 'title\_Master', 'title\_Miss', 'title\_Mr', 'title\_Mrs',

'title\_rare', 'family\_type\_big', 'family\_type\_middle',

'family\_type\_sigle', 'cabin\_type\_A', 'cabin\_type\_B', 'cabin\_type\_C',

'cabin\_type\_D', 'cabin\_type\_E', 'cabin\_type\_F', 'cabin\_type\_O',

'cabin\_type\_n', 'Fare\_scaled', 'family\_cnt\_scaled', 'SibSp\_scaled',

'Parch\_scaled']

agenull\_test=all\_df.loc[all\_df.Age.isnull(),age\_factor]

agenotnull\_train=all\_df.loc[~all\_df.Age.isnull(),age\_factor]

agenotnull\_label=all\_df.loc[~all\_df.Age.isnull(),'Age']

这里就随便用几千个树啦，随机森林不会过拟合，那就用啦。

from sklearn.ensemble import RandomForestRegressor

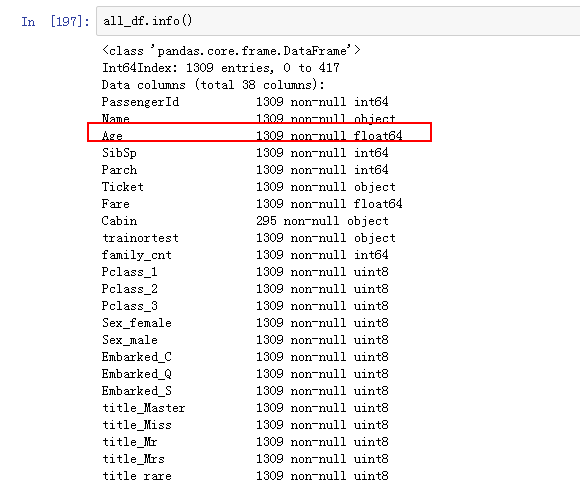
rfr = RandomForestRegressor(random\_state=0, n\_estimators=2000, n\_jobs=-1)

rfr.fit(agenotnull\_train.values,agenotnull\_label.values)

最后填充：

all\_df.loc[all\_df.Age.isnull(),'Age']=age\_pre

结果如下；



年龄已经搞定了。Ok

根据年龄我们可以进行离散化，比如划分为儿童、中年、老年。

儿童：0~18

中年：19~40

老年：40以上

因为体力问题，所以这里进行了这样的划分。

all\_df.loc[all\_df.Age<=12,'Age\_type']='child'

all\_df.loc[all\_df.Age.between(13,40),'Age\_type']='adult'

all\_df.loc[all\_df.Age>40,'Age\_type']='old'

all\_df=pd.get\_dummies(all\_df,columns=['Age\_type'])

## 特征融合

筛选出重要特征，通过逻辑回归的贪婪搜索

代码如下：

import numpy as np

import sklearn.linear\_model as lm

from sklearn import metrics, preprocessing

class greedyFeatureSelection(object):

def \_\_init\_\_(self, data, labels, scale=1, verbose=0):

if scale == 1:

self.\_data = preprocessing.scale(np.array(data))

else:

self.\_data = np.array(data)

self.\_labels = labels

self.\_verbose = verbose

def evaluateScore(self, X, y):

model = lm.LogisticRegression()

model.fit(X, y)

predictions = model.predict\_proba(X)[:, 1]

auc = metrics.roc\_auc\_score(y, predictions)

return auc

def selectionLoop(self, X, y):

score\_history = []

good\_features = set([])

num\_features = X.shape[1]

while len(score\_history) < 2 or score\_history[-1][0] > score\_history[-2][0]:

scores = []

for feature in range(num\_features):

if feature not in good\_features:

selected\_features = list(good\_features) + [feature]

Xts = np.column\_stack(X[:, j] for j in selected\_features)

score = self.evaluateScore(Xts, y)

scores.append((score, feature))

if self.\_verbose:

print ("Current AUC:",np.mean(score))

good\_features.add(sorted(scores)[-1][1])

score\_history.append(sorted(scores)[-1])

if self.\_verbose:

print ("Current Features : ", sorted(list(good\_features)))

# Remove last added feature

good\_features.remove(score\_history[-1][1])

good\_features = sorted(list(good\_features))

if self.\_verbose:

print ("Selected Features : ", good\_features)

return good\_features

def transform(self, X):

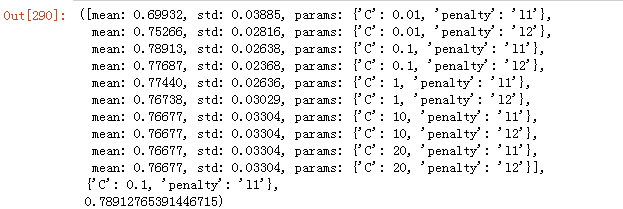
X = self.\_data

y = self.\_labels

good\_features = self.selectionLoop(X, y)

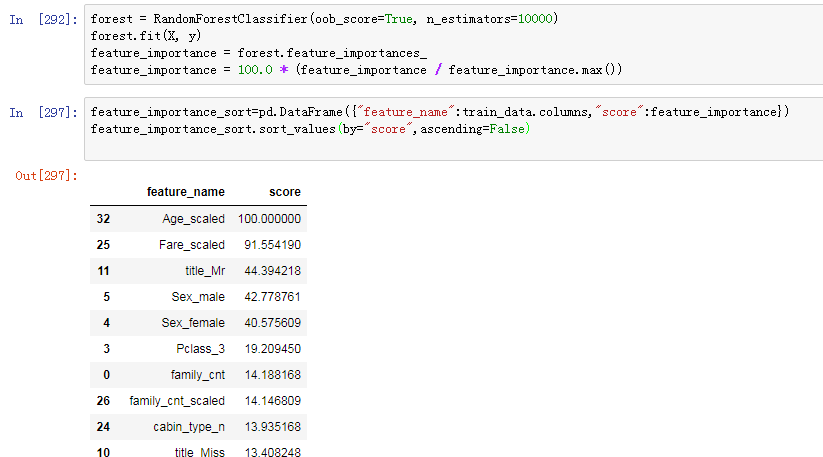
return X[:, good\_features]

不过发现如果采用筛选出来的特征拿去做模型，发现准确率降低了：

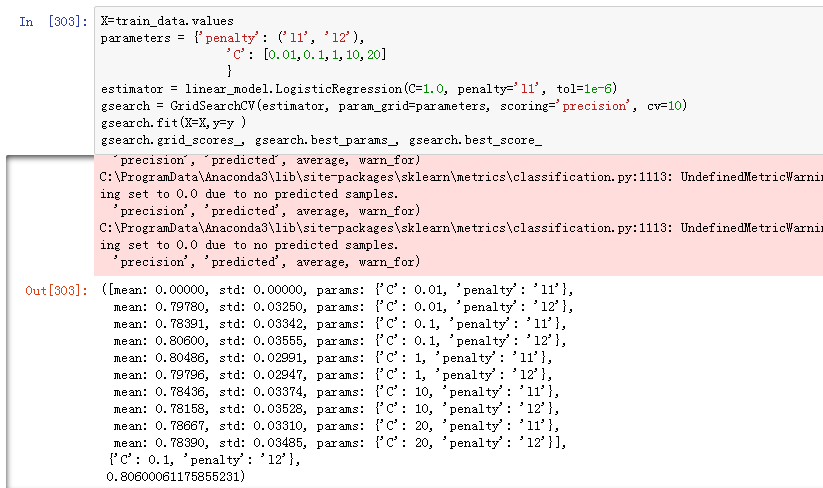


所以这里先将重要特征做融合。

采用随机森林得到特征重要度，然后将很重要的特征做一次融合。



貌似并没有什么用啊。



并没有什么太大提升。但是对比以前的情况整体来说大约有1个点的提升。

# 特征提取

特征选择啦，后面根据模型不断的进行迭代。

<https://github.com/abhishekkrthakur/greedyFeatureSelection>

特征的选取可以采用贪婪算法不断地迭代，才看上面的代码。

经过特征工程，我们选取到的特征如下：

Data columns (total 42 columns):

family\_cnt 891 non-null int64

Pclass\_1 891 non-null uint8

Pclass\_2 891 non-null uint8

Pclass\_3 891 non-null uint8

Sex\_female 891 non-null uint8

Sex\_male 891 non-null uint8

Embarked\_C 891 non-null uint8

Embarked\_Q 891 non-null uint8

Embarked\_S 891 non-null uint8

title\_Master 891 non-null uint8

title\_Miss 891 non-null uint8

title\_Mr 891 non-null uint8

title\_Mrs 891 non-null uint8

title\_rare 891 non-null uint8

family\_type\_big 891 non-null uint8

family\_type\_middle 891 non-null uint8

family\_type\_sigle 891 non-null uint8

cabin\_type\_A 891 non-null uint8

cabin\_type\_B 891 non-null uint8

cabin\_type\_C 891 non-null uint8

cabin\_type\_D 891 non-null uint8

cabin\_type\_E 891 non-null uint8

cabin\_type\_F 891 non-null uint8

cabin\_type\_O 891 non-null uint8

cabin\_type\_n 891 non-null uint8

Fare\_scaled 891 non-null float64

family\_cnt\_scaled 891 non-null float64

SibSp\_scaled 891 non-null float64

Parch\_scaled 891 non-null float64

Age\_type\_adult 891 non-null uint8

Age\_type\_child 891 non-null uint8

Age\_type\_old 891 non-null uint8

Age\_scaled 891 non-null float64

Age\_Fare 891 non-null float64

Age\_title 891 non-null float64

Age\_male 891 non-null float64

Age\_female 891 non-null float64

Fare\_title 891 non-null float64

Fare\_male 891 non-null float64

Fare\_female 891 non-null float64

title\_male 891 non-null uint8

title\_female 891 non-null uint8

# 模型调优

## LR 模型+bagging

采用LR 模型作为baseline

from sklearn import linear\_model

from sklearn.model\_selection import GridSearchCV

from sklearn import linear\_model

parameters = {'penalty': ('l1', 'l2'),

'C': [0.01,0.1,1,10,20]

}

estimator = linear\_model.LogisticRegression(C=1.0, penalty='l1', tol=1e-6)

gsearch = GridSearchCV(estimator, param\_grid=parameters, scoring='roc\_auc', cv=10)

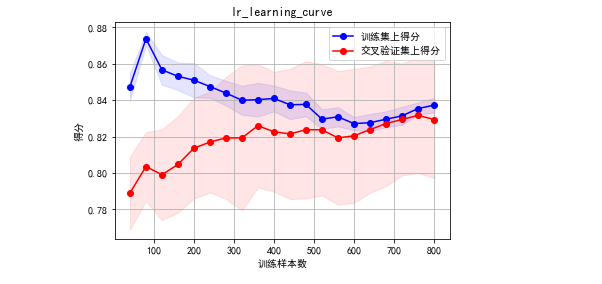
gsearch.fit(X=X,y=y )

gsearch.grid\_scores\_, gsearch.best\_params\_, gsearch.best\_score\_

由于系统判定是采用的准确率，这里的判定也采用了准确率



判定模型状态：



貌似还不错，没有过拟合。不过我们可以看到验证集合中。方差还是蛮大的。

采用LR作为基模型，然后使用bagging方式减小variance。

代码如下：

from sklearn.ensemble import BaggingClassifier

bagging\_clf = BaggingClassifier(clf\_lr, n\_estimators=20,

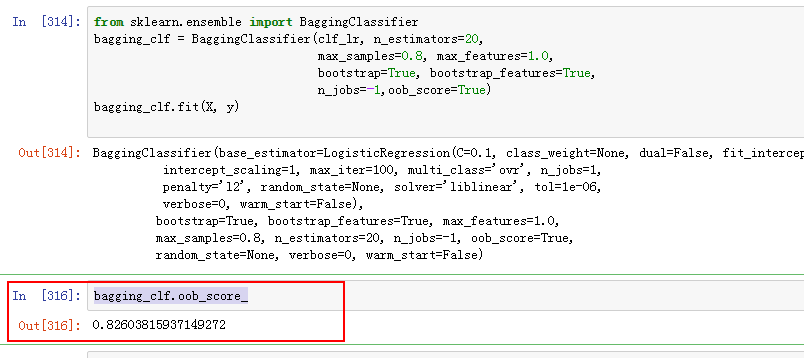
max\_samples=0.8, max\_features=1.0,

bootstrap=True, bootstrap\_features=True,

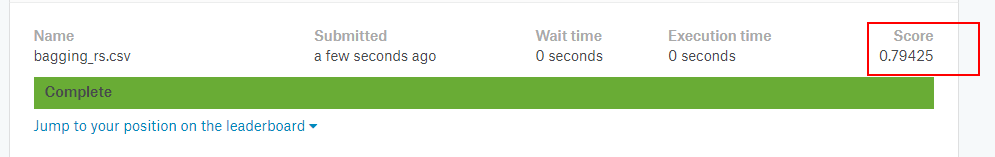
n\_jobs=-1,oob\_score=True)

bagging\_clf.fit(X, y)

查看袋外得分。因为做bagging的时候，这个oob 可以作为验证集上的分数。



好的，我们的第一个组合模型的得分如下：



看起来效果还不错。在top27% 范围。下面采用其他模型，然后通过blending的过程，进一步提升模型的能力。

## 随机森林

from sklearn.ensemble import RandomForestClassifier

parameters = {

"n\_estimators": [100,200,500,800,1500,3000],

"max\_depth":[5,8,15,25,30,None],

"min\_samples\_leaf": [1,2, 5, 7,10],

"min\_samples\_split": [1.0,2,5, 10,15,100],

"max\_features":["log2","sqrt",None]

}

estimator = RandomForestClassifier(random\_state=0, n\_estimators=2000, n\_jobs=-1)

gsearch = GridSearchCV(estimator, param\_grid=parameters, scoring='roc\_auc', cv=10)

gsearch.fit(X=X,y=y )

gsearch.grid\_scores\_, gsearch.best\_params\_, gsearch.best\_score\_

这个跑起来很慢的，像哇牛，最好是一个个的选择，但是这样就不一定是全局最优的了，所以嘛，真麻烦。

可以大致算一算跑多少轮：6\*6\*5\*6\*3大约等于2700次

加上cv=10 ，要算27000次。

一次跑一分钟也要跑一个小时哇。

这个我跑了一天都没有跑出来，只能采用逐个击破的办法了，局部最优就局部最优吧。

最后搜索到的结果如下：

parameters = {

"n\_estimators": [100],

"max\_depth":[5],

"min\_samples\_leaf": [1],

"min\_samples\_split": [5],

"max\_features":["sqrt"]

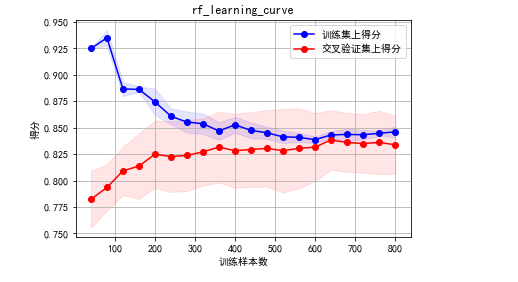
}

使用该模型，查看学习曲线，看看模型是否过拟合了。随机森林是不会过拟合的啦。啦啦啦啦

clf\_rf=RandomForestClassifier(random\_state=0, n\_estimators=100,max\_depth=5, min\_samples\_leaf=1,min\_samples\_split=5,max\_features='sqrt',n\_jobs=-1)

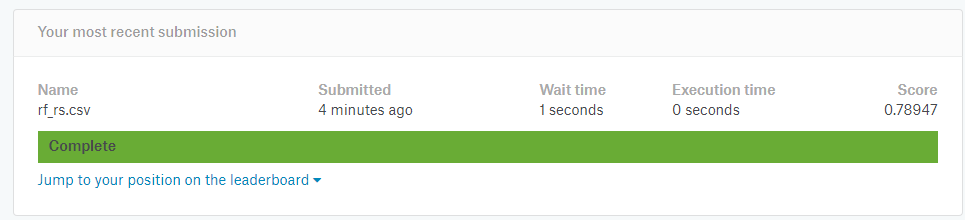
clf\_rf.fit(X=X,y=y)

plot\_learning\_curve(estimator=clf\_rf,title="lr\_learning\_curve",X=X,y=y)



看样子模型还是可以的嘛，没有过拟合，稍微欠拟合了点。

我们看看模型提交后的结果吧：



比LR + bagging 要来的低啊。

## SVM

这里采用SVC进行

这里我们采用验证曲线进行调参。

如下是调节C的大小，就是正则化的情况。

from sklearn.learning\_curve import validation\_curve

from sklearn.svm import SVC

param\_range = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]

train\_scores, test\_scores = validation\_curve(estimator=SVC(), X=X, y=y, param\_name='C', param\_range=param\_range, cv=10)

train\_mean = np.mean(train\_scores, axis=1)

train\_std = np.std(train\_scores, axis=1)

test\_mean = np.mean(test\_scores, axis=1)

test\_std = np.std(test\_scores, axis=1)

plt.plot(param\_range, train\_mean, color='blue', marker='o', markersize=5, label='training accuracy')

plt.fill\_between(param\_range, train\_mean + train\_std, train\_mean - train\_std, alpha=0.15, color='blue')

plt.plot(param\_range, test\_mean, color='green', linestyle='--', marker='s', markersize=5, label='validation accuracy')

plt.fill\_between(param\_range, test\_mean + test\_std, test\_mean - test\_std, alpha=0.15, color='green')

plt.grid()

plt.xscale('log')

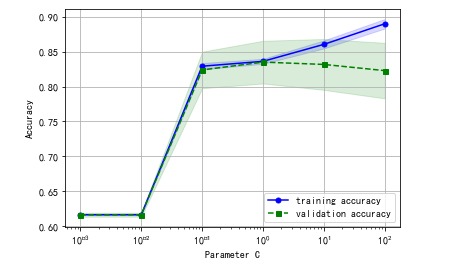
plt.legend(loc='lower right')

plt.xlabel('Parameter C')

plt.ylabel('Accuracy')

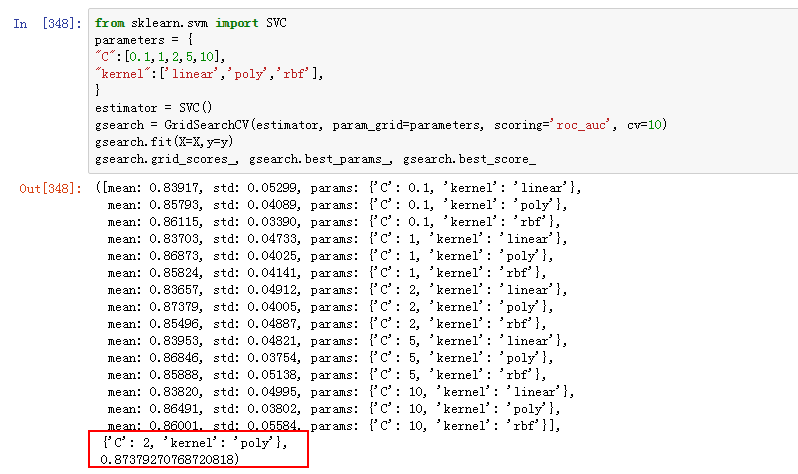
plt.show()

结果如下：

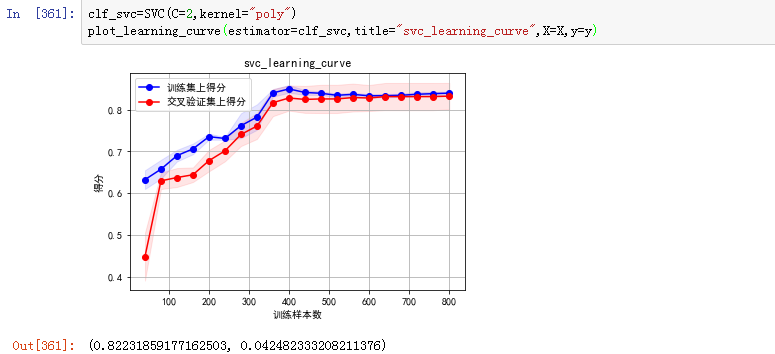


我们选择C=1左右。

在用网格搜索进行微调



得到参数：C=2，kernel =’poly’ 进行多项式的核函数映射，最后画出来的模型学习曲线如下：

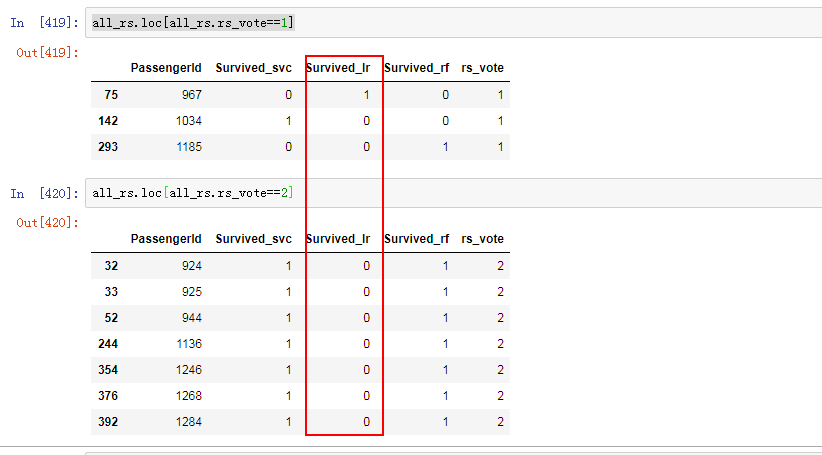


还好。

提交预测试试：

# 模型融合

首先拼接三个模型的结果，查看不一致的地方：



可以看到我们一共有10个地方预测不一致。

即使全部正确，大约只能提升2个点。 10/418 =0.239

两个点呢，哇咔咔如果正确就可以使得整个模型突破80%的预测率。因为lr已经79.45%啦。

如果不嫌弃麻烦，可以挨着修改这十个试试。反正也木有几个。

先用逻辑回归搞一把

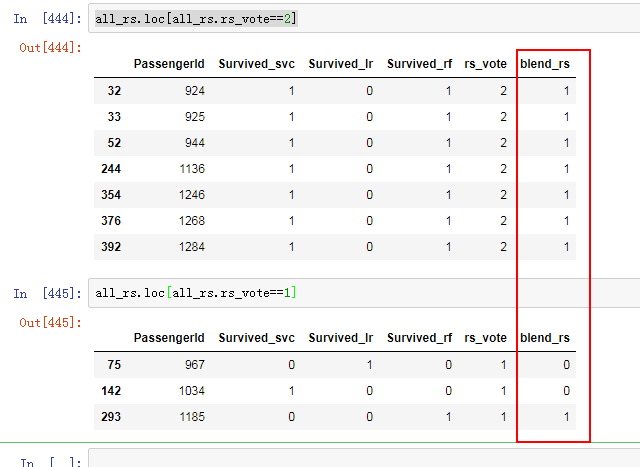
代码如下：

bagging\_clf\_rs=bagging\_clf.predict(X)

clf\_svc\_rs=clf\_svc.predict(X)

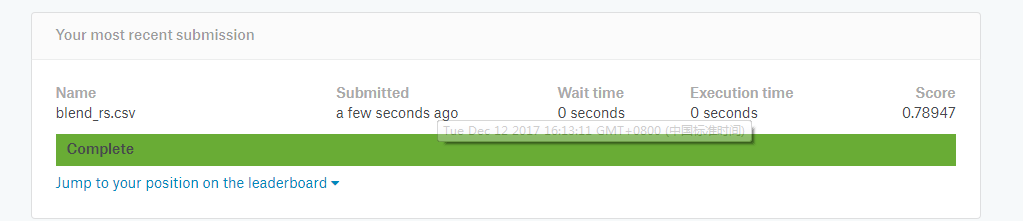
clf\_rf\_rs=clf\_rf.predict(X)

X\_blend=pd.DataFrame({"bagging\_clf\_rs":bagging\_clf\_rs,"clf\_svc\_rs":clf\_svc\_rs,"clf\_rf\_rs":clf\_rf\_rs})



# 提交

最后提交结果



忧伤，还没有LR+bagging的高

要想进一步提升效果，还是在特征上。模型调参数和融合带来的效果远不如特征来的快。

特征工程决定了模型的天花板，模型调参和融合只是不断地接近这个天花板！！！！

# 参考连接

<https://zhuanlan.zhihu.com/p/25185856>

<https://www.cnblogs.com/qqhfeng/p/5341840.html>

<https://www.cnblogs.com/north-north/p/4360121.html>