

作者: [寒小阳](#)

时间: 2015 年 11 月。

出处: http://blog.csdn.net/han_xiaoyang/article/details/49797143

声明: 版权所有, 转载请注明出处, 谢谢。

1. 引言

先说一句, 年末双十一什么的一来, 真是非(mang)常(cheng)欢(gou)乐(le)! 然后 push 自己抽出时间来写这篇 blog 的原因也非常简单:

- 写完前两篇逻辑回归的介绍和各个角度理解之后, 我们讨论群([戳我入群](#))的小伙伴们纷纷表示『好像很高级的样纸, but 然并卵 啊! 你们倒是拿点实际数据来给我们看看, 这玩意儿 有! 什! 么! 用! 啊! 』
- talk is cheap, show me the code!
- no example say a jb!

OK, OK, 这就来了咯, 同学们别着急, 我们先找个简单的实际例子, 来看看, 所谓的[数据挖掘](#)或者[机器学习](#)实际应用到底是怎么样一个过程。

『喂, 那几个说要看大数据上机器学习应用的, 对, 就是说你们! 别着急好么, 我们之后拉点大一点实际数据用 [liblinear](#) 或者 [spark, MLlib](#) 跑给你们看, 行不行? 咱们先拿个实例入入门嘛』

好了, 我是一个严肃的技术研究和分享者, 咳咳, 不能废话了, 各位同学继续往下看吧!

2. 背景

2.1 关于 Kaggle

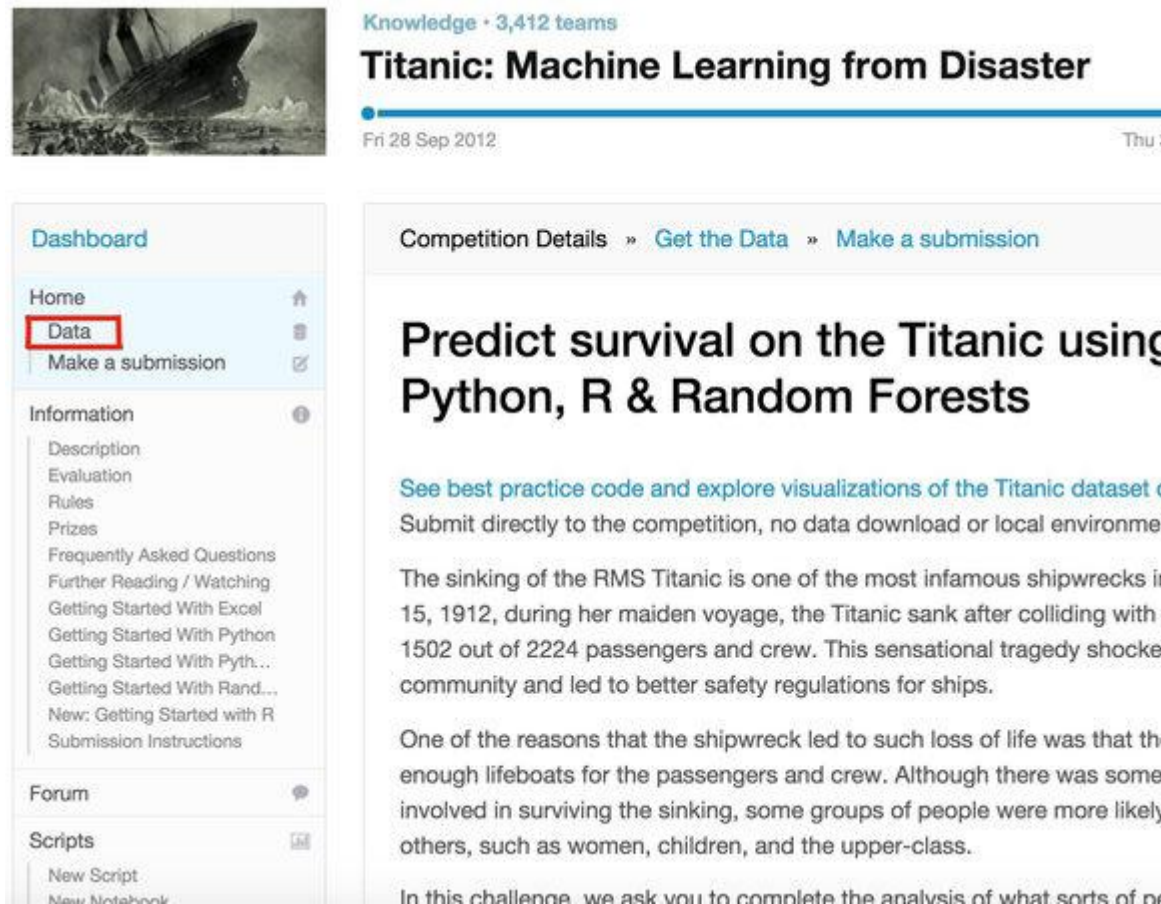
- [我是 Kaggle 地址, 翻我牌子](#)
- 亲, 逼格这么高的地方, 你一定听过对不对? 是! 这就是那个无数『数据挖掘先驱』们, 在回答“枪我有了, 哪能找到靶子练练手啊?”时候的答案!
- 这是一个要数据有数据, 要实际应用场景有场景, 要一起在数据挖掘领域 high 得不要不要的小伙伴就有小伙伴的地方啊!!!

艾玛, 逗逼模式开太猛了。恩, 不闹, 不闹, 说正事, Kaggle 是一个数据分析建模的应用竞赛平台, 有点类似 [KDD-CUP](#) (国际知识发现和数据挖掘竞赛), 企业或者研究者可以将问题背景、数据、期望指标等发布到 Kaggle 上, 以竞赛的形式向广大的数据科学家征集解决方案。而热爱数(dong)据(shou)挖(zhe)掘(teng)的小伙伴们可以[下载/分析数据](#), 使用统计/机器学习/数据挖掘等知识, 建立算法模型, 得出结果并提交, 排名 top 的可能会有奖金哦!

2.2 关于泰坦尼克号之灾

- 带大家去[该问题页面](#)溜达一圈吧

- - 下面是问题背景页



The screenshot displays the 'Titanic: Machine Learning from Disaster' competition page on Kaggle. At the top, it shows 'Knowledge • 3,412 teams' and the title 'Titanic: Machine Learning from Disaster'. Below the title, the date 'Fri 28 Sep 2012' is visible. The page is divided into two main sections. On the left is a 'Dashboard' sidebar with a navigation menu. The 'Data' option in the menu is highlighted with a red rectangle. The main content area on the right features the title 'Predict survival on the Titanic using Python, R & Random Forests' and a brief description of the competition. The description mentions that the sinking of the RMS Titanic is one of the most infamous shipwrecks in history, and the goal is to predict survival based on the dataset provided.

Knowledge • 3,412 teams

Titanic: Machine Learning from Disaster

Fri 28 Sep 2012

Competition Details » [Get the Data](#) » [Make a submission](#)

Predict survival on the Titanic using Python, R & Random Forests

[See best practice code and explore visualizations of the Titanic dataset](#)

Submit directly to the competition, no data download or local environment required.

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, resulting in the loss of 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of randomness involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

In this challenge, we ask you to complete the analysis of what sorts of people

- 下面是可下载 Data 的页面



Knowledge • 3,411 teams

Titanic: Machine Learning from Disaster

Fri 28 Sep 2012

Dashboard

- Home
- Data**
- Make a submission

Information

- Description
- Evaluation
- Rules
- Prizes
- Frequently Asked Questions
- Further Reading / Watching
- Getting Started With Excel
- Getting Started With Python
- Getting Started With Pyth...
- Getting Started With Rand...
- New: Getting Started with R
- Submission Instructions

Forum

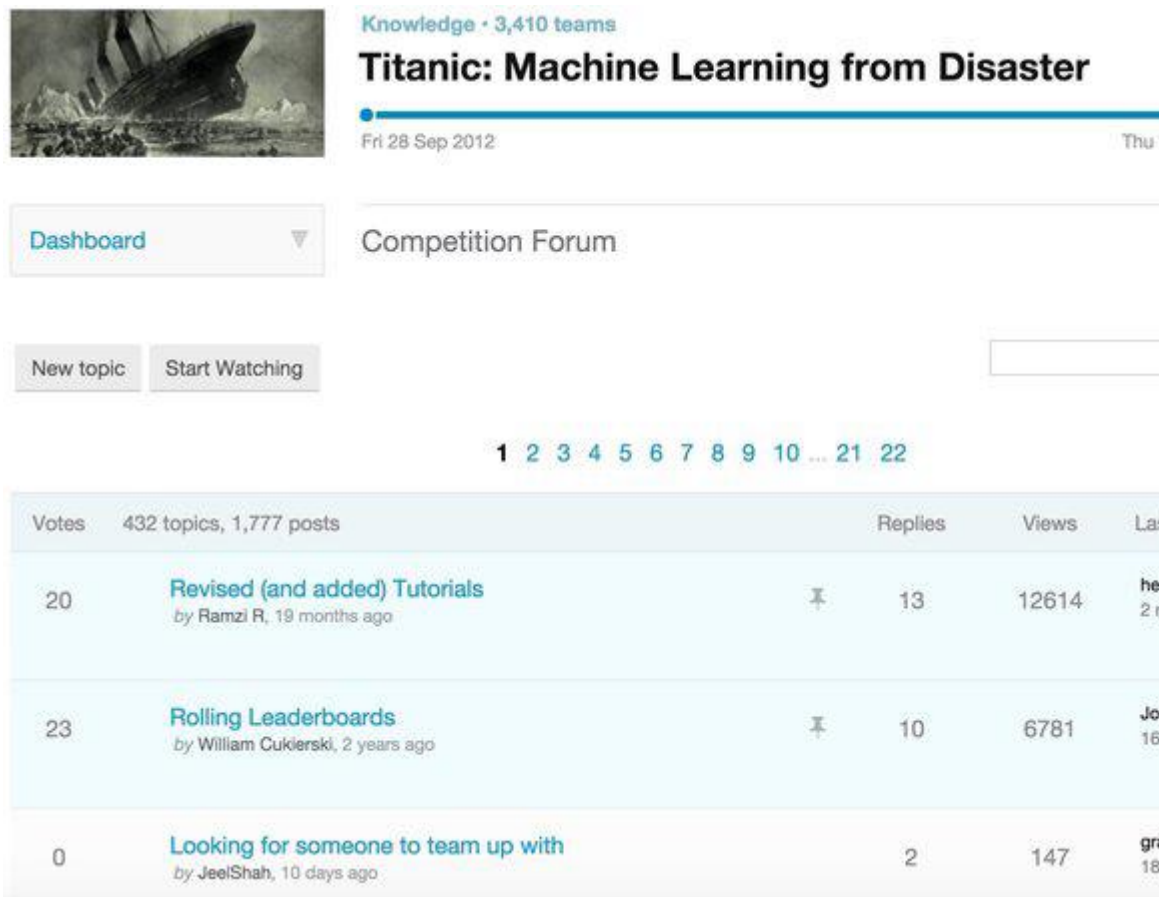
Scripts

[Competition Details](#) » [Get the Data](#) » [Make a submission](#)

Data Files

File Name	Available Form
train	.csv (59.76 kb)
gendermodel	.csv (3.18 kb)
genderclassmodel	.csv (3.18 kb)
test	.csv (27.96 kb)
gendermodel	.py (3.58 kb)
genderclassmodel	.py (5.63 kb)
myfirstforest	.py (3.99 kb)

- 下面是小伙伴们最爱的 forum 页面，你会看到各种神级人物厉(qi)害(pa)的
数据处理/建模想法，你会直视『世界真奇妙』。



Knowledge • 3,410 teams

Titanic: Machine Learning from Disaster

Fri 28 Sep 2012

Dashboard ▼

Competition Forum

New topic Start Watching

1 2 3 4 5 6 7 8 9 10 ... 21 22

Votes	Topics	Posts	Replies	Views	La
20	Revised (and added) Tutorials	by Ramzi R, 19 months ago	13	12614	he 21
23	Rolling Leaderboards	by William Cukierski, 2 years ago	10	6781	Jo 16
0	Looking for someone to team up with	by JeelShah, 10 days ago	2	147	gr 18

•

泰坦尼克号问题之背景

•

○

就是那个大家都熟悉的『Jack and Rose』的故事，豪华游艇倒了，大家都惊恐逃生，可是救生艇的数量有限，无法人人都有，副船长发话了『lady and kid first!』，所以是否获救其实并非随机，而是基于一些背景有 **rank** 先后的。

○

○

训练和测试数据是一些乘客的个人信息以及存活状况，要尝试根据它生成合适的模型并预测其他人的存活状况。

○

○

对，这是一个二分类问题，是我们之前讨论的 **logistic regression** 所能处理的范畴。

○

3. 说明

接触过 Kaggle 的同学们可能知道这个问题，也可能知道 RandomForest 和 SVM 等算法，甚至还对多个模型做过融合，取得过非常好的结果，那 maybe 这篇文章并不是针对你的，你可以自行略过。

我们因为之前只介绍了 Logistic Regression 这一种分类算法。所以本次的问题解决过程和优化思路，都集中在这种算法上。其余的方法可能我们之后的文章里会提到。

说点个人的观点。不一定正确。

【解决一个问题的方法和思路不止一种】

【没有所谓的机器学习算法优劣，也没有绝对高性能的机器学习算法，只有在特定的场景、数据和特征下更合适的机器学习算法。】

4. 怎么做？

手把手教程马上就来，先来两条我看到的，觉得很重要的经验。

1.

印象中 Andrew Ng 老师似乎在 coursera 上说过，应用机器学习，千万不要一上来就试图做到完美，先撸一个 **baseline** 的 **model** 出来，再进行后续的分析步骤，一步步提高，所谓后续步骤可能包括『分析 model 现在的状态(欠/过拟合)，分析我们使用的 **feature** 的作用大小，进行 **feature selection**，以及我们模型下的 **bad case** 和产生的原因』等等。

2.

3.

Kaggle 上的大神们，也分享过一些 **experience**，说几条我记得的哈：

4.

○ 【对数据的认识太重要了！】

○ 【数据中的特殊点/离群点的分析和处理太重要了！】

- 【特征工程(feature engineering)太重要了！在很多 Kaggle 的场景下，甚至比 model 本身还要重要】
- 【要做模型融合(model ensemble)啊啊啊！】

更多的经验分享请加讨论群，具体方式请联系作者，或者参见[《“ML 学分计划”说明书》](#)

5. 初探数据

先看看我们的数据，长什么样吧。在 Data 下我们 train.csv 和 test.csv 两个文件，分别存着官方给的训练和测试数据。

```
import pandas as pd #数据分析 import numpy as np #科学计算 from pandas
import Series,DataFrame
```

```
data_train = pd.read_csv("/Users/Hanxiaoyang/Titanic_data/Train.csv")
data_train
```

- 1
- 2
- 3
- 4
- 5
- 6

pandas 是常用的 python 数据处理包，把 csv 文件读入成 dataframe 各式，我们在 ipython notebook 中，看到 data_train 如下所示：

1):

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14	1	0	237736	30.0708	NaN	C

这就是典型的 **dataframe** 格式，如果你没接触过这种格式，完全没有关系，你就把它想象成 Excel 里面的列好了。

我们看到，总共有 12 列，其中 **Survived** 字段表示的是该乘客是否获救，其余都是乘客的个人信息，包括：

- PassengerId => 乘客 ID
- Pclass => 乘客等级(1/2/3 等舱位)
- Name => 乘客姓名
- Sex => 性别
- Age => 年龄
- SibSp => 堂兄弟/妹个数
- Parch => 父母与小孩个数
- Ticket => 船票信息
- Fare => 票价
- Cabin => 客舱
- Embarked => 登船港口

逐条往下看，要看完这么多条，眼睛都有一种要瞎的赶脚。好吧，我们让 **dataframe** 自己告诉我们一些信息，如下所示：

```
data_train.info()
```

- 1

看到了如下的信息：

```
1]: data_train.info()
#我们发现有一些列，比如说Cabin，有非常多的缺失值
#另外一些我们感觉在此场景中会有影响的属性，比如Age，也有一些缺失值

<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 90.5+ KB
```

上面的数据说啥了？它告诉我们，训练数据中总共有 891 名乘客，但是很不幸，我们有些属性的数据不全，比如说：

- Age（年龄）属性只有 714 名乘客有记录
- Cabin（客舱）更是只有 204 名乘客是已知的

似乎信息略少啊，想再瞄一眼具体数据数值情况呢？恩，我们用下列的方法，得到数值型数据的一些分布(因为有些属性，比如姓名，是文本型；而另外一些属性，比如登船港口，是类目型。这些我们用下面的函数是看不到的)：

data_train.describe()							
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

我们从上面看到更进一步的什么信息呢？

mean 字段告诉我们，大概 0.383838 的人最后获救了，2/3 等舱的人数比 1 等舱要多，平均乘客年龄大概是 29.7 岁(计算这个时候会略掉无记录的)等等...

6. 数据初步分析

每个乘客都这么多属性，那我们咋知道哪些属性更有用，而又应该怎么用它们啊？说实话这会儿我也不知道，但我们记得前面提到过

- 【对数据的认识太重要了！】
- 【对数据的认识太重要了！】
- 【对数据的认识太重要了！】

重要的事情说三遍，恩，说完了。仅仅最上面的对数据了解，依旧无法给我们提供想法和思路。我们再深入一点来看看我们的数据，看看每个/多个 属性和最后的 **Survived** 之间有着什么样的关系呢。

6.1 乘客各属性分布

脑容量太有限了...数值看花眼了。我们还是统计统计，画些图来看看属性和结果之间的关系好了，代码如下：

```
import matplotlib.pyplot as plt
fig = plt.figure()
fig.set(alpha=0.2) # 设定图表颜色 alpha 参数

plt.subplot2grid((2,3),(0,0)) # 在一张大图里分列几个小图
data_train.Survived.value_counts().plot(kind='bar') # 柱状图
plt.title(u"获救情况 (1 为获救)") # 标题
plt.ylabel(u"人数")

plt.subplot2grid((2,3),(0,1))
data_train.Pclass.value_counts().plot(kind="bar")
plt.ylabel(u"人数")
plt.title(u"乘客等级分布")

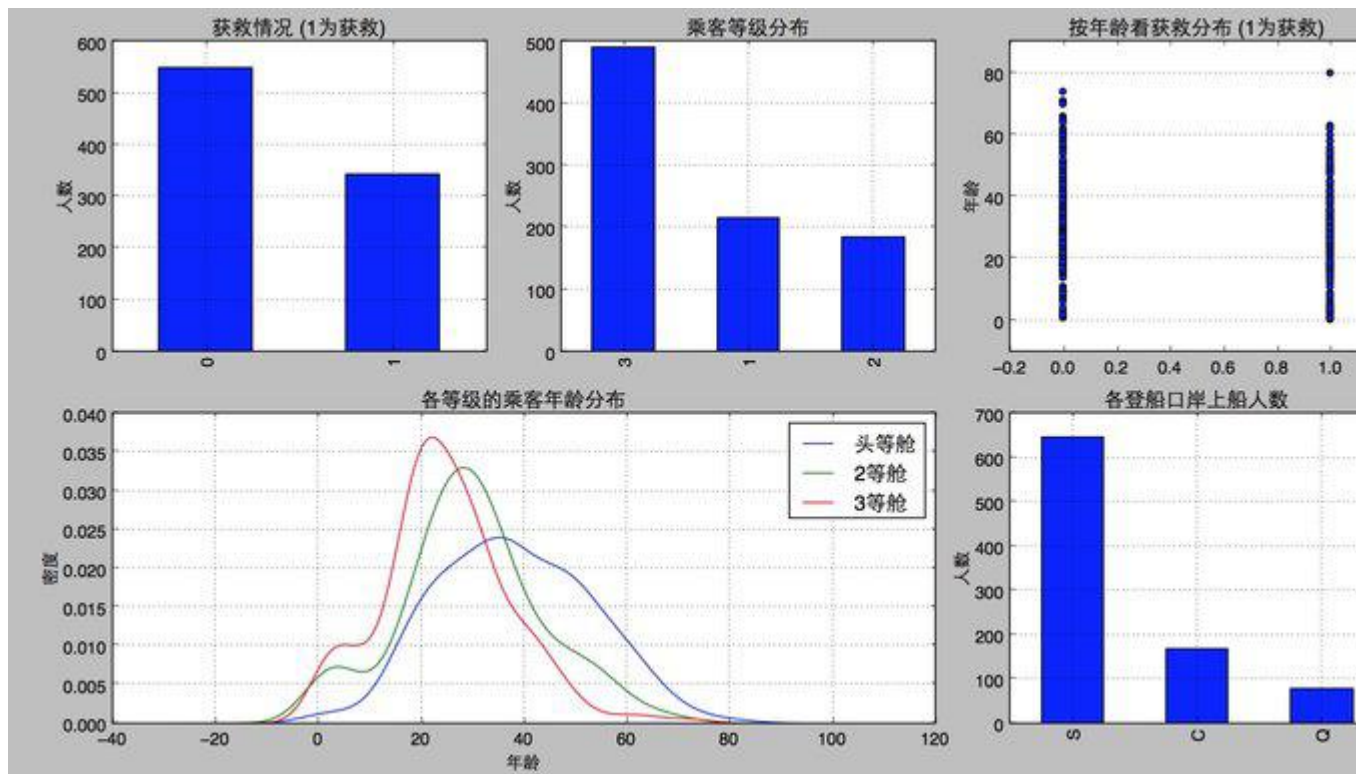
plt.subplot2grid((2,3),(0,2))
plt.scatter(data_train.Survived, data_train.Age)
plt.ylabel(u"年龄") # 设定纵坐标名称
plt.grid(b=True, which='major', axis='y')
plt.title(u"按年龄看获救分布 (1 为获救)")

plt.subplot2grid((2,3),(1,0), colspan=2)
data_train.Age[data_train.Pclass == 1].plot(kind='kde')
data_train.Age[data_train.Pclass == 2].plot(kind='kde')
data_train.Age[data_train.Pclass == 3].plot(kind='kde')
plt.xlabel(u"年龄") # plots an axis lable
plt.ylabel(u"密度")
plt.title(u"各等级的乘客年龄分布")
plt.legend((u'头等舱', u'2等舱', u'3等舱'), loc='best') # sets our legend
for our graph.

plt.subplot2grid((2,3),(1,2))
data_train.Embarked.value_counts().plot(kind='bar')
plt.title(u"各登船口岸上船人数")
plt.ylabel(u"人数")
plt.show()
```

- 1
- 2
- 3
- 4

- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36



bingo，图还是比数字好看多了。所以我们在图上可以看出来，被救的人 300 多点，不到半数；3 等舱乘客非常多；遇难和获救的人年龄似乎跨度都很广；3 个不同的舱年龄总体趋势似乎也一致，2/3 等舱乘客 20 岁多点的人最多，1 等舱 40 岁左右的最多(→_→似乎符合财富和年龄的分配哈，咳咳，别理我，我瞎扯的)；登船港口人数按照 S、C、Q 递减，而且 S 远多于另外俩港口。

这个时候我们可能会有一些想法了：

- 不同舱位/乘客等级可能和财富/地位有关系，最后获救概率可能会不一样
- 年龄对获救概率也一定是有影响的，毕竟前面说了，副船长还说『小孩和女士先走』呢
- 和登船港口是不是有关系呢？也许登船港口不同，人的出身地位不同？

口说无凭，空想无益。老老实实再来统计统计，看看这些属性值的统计分布吧。

6.2 属性与获救结果的关联统计

#看看各乘客等级的获救情况

```
fig = plt.figure()
```

```
fig.set(alpha=0.2) # 设定图表颜色 alpha 参数
```

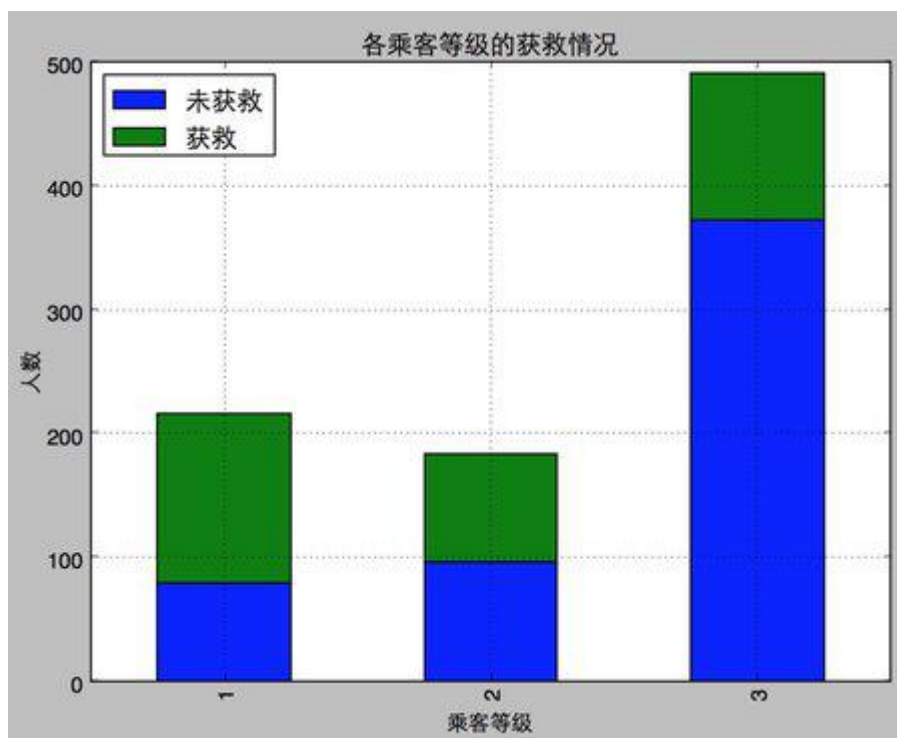
```
Survived_0 = data_train.Pclass[data_train.Survived == 0].value_counts()
```

```
Survived_1 = data_train.Pclass[data_train.Survived == 1].value_counts()
```

```
df=pd.DataFrame({u' 获救':Survived_1, u' 未获救':Survived_0})
```

```
df.plot(kind='bar', stacked=True)
plt.title(u"各乘客等级的获救情况")
plt.xlabel(u"乘客等级")
plt.ylabel(u"人数")
plt.show()
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12



啧啧，果然，钱和地位对舱位有影响，进而对获救的可能性也有影响啊←_←
 咳咳，跑题了，我想说的是，明显等级为 1 的乘客，获救的概率高很多。恩，这个一定是影响最后获救结果的一个特征。

#看看各性别的获救情况

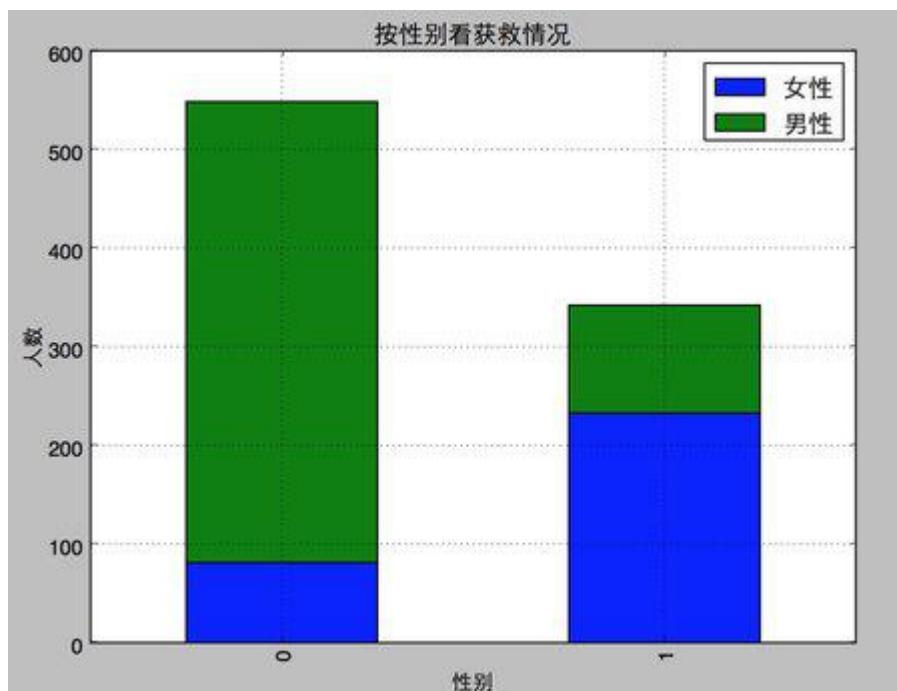
```
fig = plt.figure()
fig.set(alpha=0.2) # 设定图表颜色 alpha 参数
```

```

Survived_m = data_train.Survived[data_train.Sex ==
'male'].value_counts()
Survived_f = data_train.Survived[data_train.Sex ==
'female'].value_counts()
df=pd.DataFrame({u'男性':Survived_m, u'女性':Survived_f})
df.plot(kind='bar', stacked=True)
plt.title(u"按性别看获救情况")
plt.xlabel(u"性别")
plt.ylabel(u"人数")
plt.show()

```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12



歪果盆友果然很尊重 lady, lady first 践行得不错。性别无疑也要作为重要特征加入最后的模型之中。

再来个详细版的好了。

```
#然后我们再来看看各种舱级别情况下各性别的获救情况
fig=plt.figure()
fig.set(alpha=0.65) # 设置图像透明度, 无所谓
plt.title(u"根据舱等级和性别的获救情况")

ax1=fig.add_subplot(141)
data_train.Survived[data_train.Sex == 'female'][data_train.Pclass !=
3].value_counts().plot(kind='bar', label="female highclass",
color='#FA2479')
ax1.set_xticklabels([u"获救", u"未获救"], rotation=0)
ax1.legend([u"女性/高级舱"], loc='best')

ax2=fig.add_subplot(142, sharey=ax1)
data_train.Survived[data_train.Sex == 'female'][data_train.Pclass ==
3].value_counts().plot(kind='bar', label='female, low class',
color='pink')
ax2.set_xticklabels([u"未获救", u"获救"], rotation=0)
plt.legend([u"女性/低级舱"], loc='best')

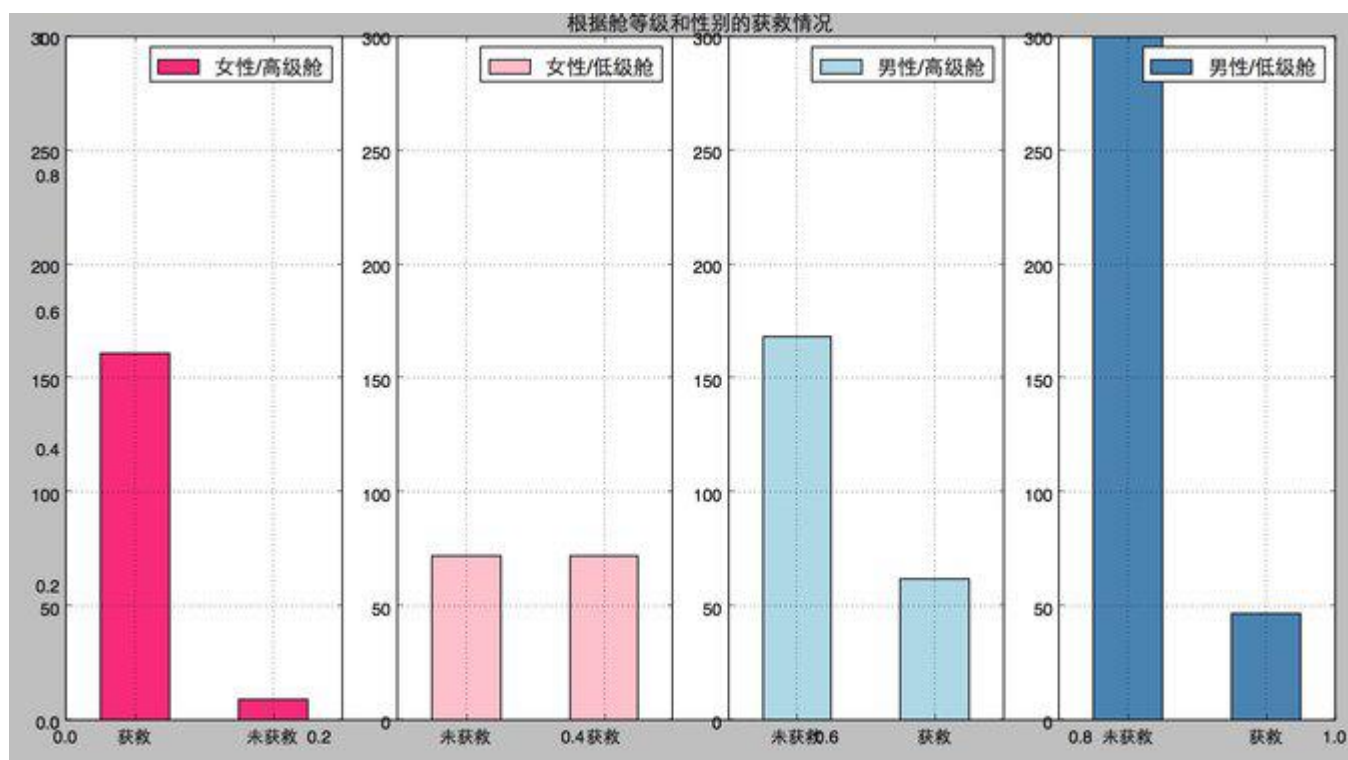
ax3=fig.add_subplot(143, sharey=ax1)
data_train.Survived[data_train.Sex == 'male'][data_train.Pclass !=
3].value_counts().plot(kind='bar', label='male, high
class',color='lightblue')
ax3.set_xticklabels([u"未获救", u"获救"], rotation=0)
plt.legend([u"男性/高级舱"], loc='best')

ax4=fig.add_subplot(144, sharey=ax1)
data_train.Survived[data_train.Sex == 'male'][data_train.Pclass ==
3].value_counts().plot(kind='bar', label='male low class',
color='steelblue')
ax4.set_xticklabels([u"未获救", u"获救"], rotation=0)
plt.legend([u"男性/低级舱"], loc='best')

plt.show()
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7

- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28



恩，坚定了之前的判断。

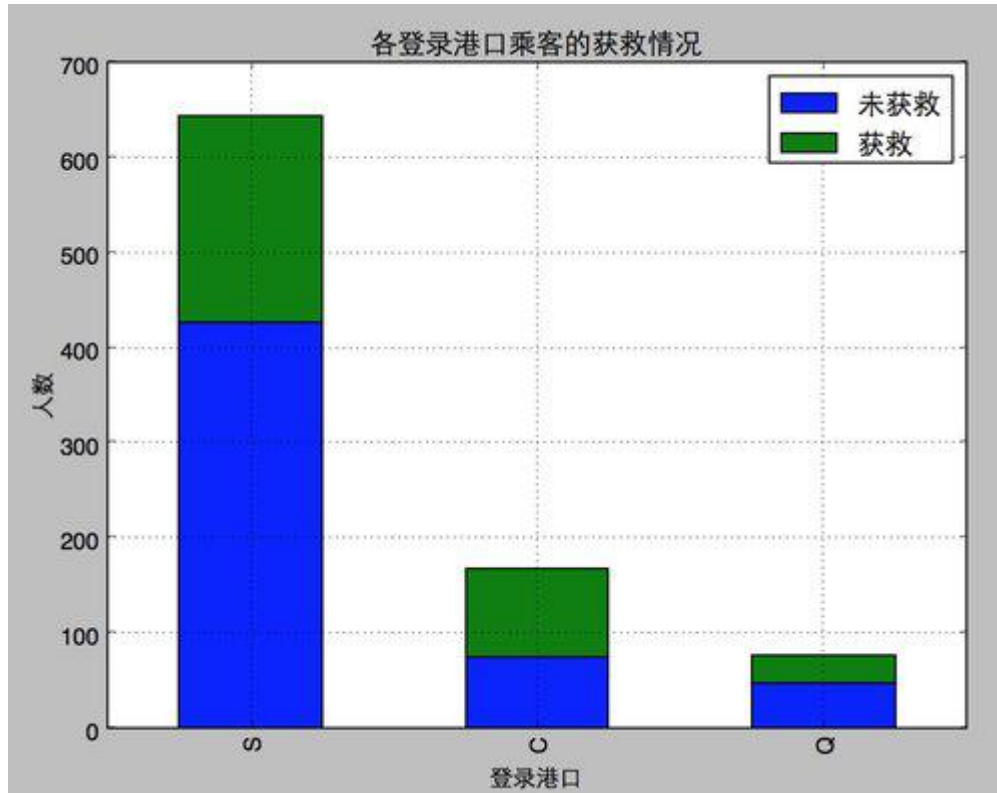
我们看看各登船港口的获救情况。


```
fig = plt.figure()
fig.set(alpha=0.2) # 设定图表颜色 alpha 参数

Survived_0 = data_train.Embarked[data_train.Survived ==
0].value_counts()
Survived_1 = data_train.Embarked[data_train.Survived ==
1].value_counts()
df=pd.DataFrame({u'获救':Survived_1, u'未获救':Survived_0})
df.plot(kind='bar', stacked=True)
plt.title(u"各登录港口乘客的获救情况")
plt.xlabel(u"登录港口")
plt.ylabel(u"人数")

plt.show()
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12



下面我们来看看 堂兄弟/妹，孩子/父母有几人，对是否获救的影响。

```
g = data_train.groupby(['SibSp', 'Survived'])
df = pd.DataFrame(g.count()['PassengerId'])
print df
```

```
g = data_train.groupby(['SibSp', 'Survived'])
df = pd.DataFrame(g.count()['PassengerId'])
print df
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

		PassengerId
SibSp	Survived	
0	0	398
	1	210
1	0	97
	1	112
2	0	15
	1	13
3	0	12
	1	4
4	0	15
	1	3
5	0	5
8	0	7

		PassengerId
Parch	Survived	
0	0	445
	1	233
1	0	53
	1	65
2	0	40
	1	40
3	0	2
	1	3
4	0	4
5	0	4
	1	1
6	0	1

好吧，没看出特别特别明显的规律(为自己的智商感到捉急...)，先作为备选特征，放一放。

#ticket 是船票编号，应该是 unique 的，和最后的结果没有太大的关系，先不纳入考虑的特征范畴把#cabin 只有 204 个乘客有值，我们先看看它的一个分布
`data_train.Cabin.value_counts()`

- 1
- 2
- 3
- 4
- 5

部分结果如下：

C23 C25 C27	4
G6	4
B96 B98	4
D	3
C22 C26	3
E101	3
F2	3
F33	3
B57 B59 B63 B66	2
C68	2
B58 B60	2
E121	2
D20	2
E8	2
E44	2
B77	2
C65	2
D26	2
E24	2
E25	2
B20	2
C93	2
D33	2
E67	2
D35	2
D36	2
C52	2
F4	2
C125	2
C124	2

这三三两两的...如此不集中...我们猜一下，也许，前面的 **ABCDE** 是指的甲板位置、然后编号是房间号？...好吧，我瞎说的，别当真...

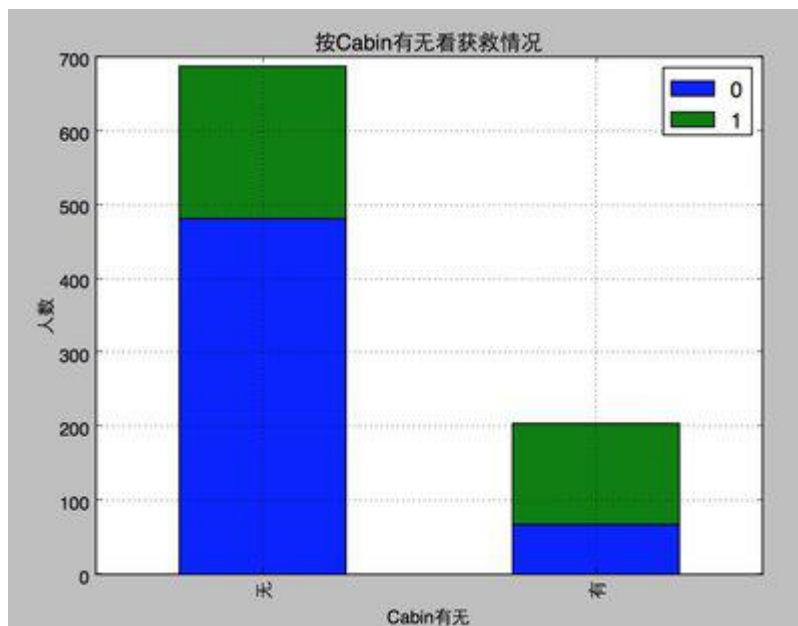
关键是 **Cabin** 这鬼属性，应该算作类目型的，本来缺失值就多，还如此不集中，注定是个棘手货...第一感觉，这玩意儿如果直接按照类目特征处理的话，太散了，估计每个因子化后的特征都拿不到什么权重。加上有那么多缺失值，要不我们先把 **Cabin** 缺失与否作为条件(虽然这部分信息缺失可能并非未登记，**maybe** 只是丢失了而已，所以这样做未必妥当)，先在有无 **Cabin** 信息这个粗粒度上看看 **Survived** 的情况好了。

```
fig = plt.figure()
```

```
fig.set(alpha=0.2) # 设定图表颜色 alpha 参数

Survived_cabin =
data_train.Survived[pd.notnull(data_train.Cabin)].value_counts()
Survived_nocabin =
data_train.Survived[pd.isnull(data_train.Cabin)].value_counts()
df=pd.DataFrame({u'有':Survived_cabin, u'无':Survived_nocabin}).transpose()
df.plot(kind='bar', stacked=True)
plt.title(u"按 Cabin 有无看获救情况")
plt.xlabel(u"Cabin 有无")
plt.ylabel(u"人数")
plt.show()
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13



咳咳，有 Cabin 记录的似乎获救概率稍高一些，先这么着放一放吧。

7. 简单数据预处理

大体数据的情况看了一遍，对感兴趣的属性也有个大概的了解了。
下一步干啥？咱们该处理处理这些数据，为机器学习建模做点准备了。

对了，我这里说的数据预处理，其实就包括了很多 Kaggler 津津乐道的 feature engineering 过程，灰常灰常有必要！

【特征工程(feature engineering)太重要了！】

【特征工程(feature engineering)太重要了！】

【特征工程(feature engineering)太重要了！】

恩，重要的事情说三遍。

先从最突出的数据属性开始吧，对，Cabin 和 Age，有丢失数据实在是对下一步工作影响太大。

先说 Cabin，暂时我们就按照刚才说的，按 Cabin 有无数据，将这个属性处理成 Yes 和 No 两种类型吧。

再说 Age：

通常遇到缺值的情况，我们会有几种常见的处理方式

- 如果缺值的样本占总数比例极高，我们可能就直接舍弃了，作为特征加入的话，可能反倒带入 noise，影响最后的结果了
- 如果缺值的样本适中，而该属性非连续值特征属性(比如说类目属性)，那就把 NaN 作为一个新类别，加到类别特征中
- 如果缺值的样本适中，而该属性为连续值特征属性，有时候我们会考虑给定一个 step(比如这里的 age，我们可以考虑每隔 2/3 岁为一个步长)，然后把它离散化，之后把 NaN 作为一个 type 加到属性类目中。
- 有些情况下，缺失的值个数并不是特别多，那我们也可以试着根据已有的值，拟合一下数据，补充上。

本例中，后两种处理方式应该都是可行的，我们先试试拟合补全吧(虽然说没有特别多的背景可供我们拟合，这不一定是一个多么好的选择)

我们这里用 scikit-learn 中的 RandomForest 来拟合一下缺失的年龄数据(注：RandomForest 是一个用在原始数据中做不同采样，建立多颗 DecisionTree，再进行 average 等等来降低过拟合现象，提高结果的机器学习算法，我们之后会介绍到)

```
from sklearn.ensemble import RandomForestRegressor
```



```

#### 使用 RandomForestClassifier 填补缺失的年龄属性 def
set_missing_ages(df):

    # 把已有的数值型特征取出来丢进 Random Forest Regressor 中
    age_df = df[['Age', 'Fare', 'Parch', 'SibSp', 'Pclass']]

    # 乘客分成已知年龄和未知年龄两部分
    known_age = age_df[age_df.Age.notnull()].as_matrix()
    unknown_age = age_df[age_df.Age.isnull()].as_matrix()

    # y 即目标年龄
    y = known_age[:, 0]

    # X 即特征属性值
    X = known_age[:, 1:]

    # fit 到 RandomForestRegressor 之中
    rfr = RandomForestRegressor(random_state=0, n_estimators=2000,
n_jobs=-1)
    rfr.fit(X, y)

    # 用得到的模型进行未知年龄结果预测
    predictedAges = rfr.predict(unknown_age[:, 1::])

    # 用得到的预测结果填补原缺失数据
    df.loc[ (df.Age.isnull()), 'Age' ] = predictedAges

    return df, rfr
def set_Cabin_type(df):
    df.loc[ (df.Cabin.notnull()), 'Cabin' ] = "Yes"
    df.loc[ (df.Cabin.isnull()), 'Cabin' ] = "No"
    return df

data_train, rfr = set_missing_ages(data_train)
data_train = set_Cabin_type(data_train)

```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8

- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171	7.2500	No	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599	71.2833	Yes	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282	7.9250	No	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803	53.1000	Yes	S
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0	373450	8.0500	No	S
5	6	0	3	Moran, Mr. James	male	23.828953	0	0	330877	8.4583	No	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.000000	0	0	17463	51.8625	Yes	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.000000	3	1	349909	21.0750	No	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.000000	0	2	347742	11.1333	No	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.000000	1	0	237736	30.0708	No	C
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.000000	1	1	PP 9549	16.7000	Yes	S

恩。目的达到，OK了。

因为逻辑回归建模时，需要输入的特征都是数值型特征，我们通常会先对类目型的特征因子化。

什么叫做因子化呢？举个例子：

以 Cabin 为例，原本一个属性维度，因为其取值可以是['yes','no']，而将其平展开为'Cabin_yes','Cabin_no'两个属性

- 原本 Cabin 取值为 yes 的，在此处的"Cabin_yes"下取值为 1，在"Cabin_no"下取值为 0
- 原本 Cabin 取值为 no 的，在此处的"Cabin_yes"下取值为 0，在"Cabin_no"下取值为 1

我们使用 pandas 的"get_dummies"来完成这个工作，并拼接在原来的"data_train"之上，如下所示。

```
dummies_Cabin = pd.get_dummies(data_train['Cabin'], prefix= 'Cabin')
```

```
dummies_Embarked = pd.get_dummies(data_train['Embarked'], prefix=
'Embarked')
```

```
dummies_Sex = pd.get_dummies(data_train['Sex'], prefix= 'Sex')
```

```
dummies_Pclass = pd.get_dummies(data_train['Pclass'], prefix= 'Pclass')
```

```
df=pd.concat([data_train, dummies_Cabin, dummies_Embarked, dummies_Sex,
dummies_Pclass], axis=1)
df.drop(['Pclass', 'Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], axis=1,
inplace=True)
df
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13

SibSp	Parch	Fare	Cabin_No	Cabin_Yes	Embarked_C	Embarked_Q	Embarked_S	Sex_female	Sex_male	Pclass_1	Pclass_2	Pclass_3
1	0	7.2500	1	0	0	0	1	0	1	0	0	1
1	0	71.2833	0	1	1	0	0	1	0	1	0	0
0	0	7.9250	1	0	0	0	1	1	0	0	0	1
1	0	53.1000	0	1	0	0	1	1	0	1	0	0
0	0	8.0500	1	0	0	0	1	0	1	0	0	1
0	0	8.4583	1	0	0	1	0	0	1	0	0	1
0	0	51.8625	0	1	0	0	1	0	1	1	0	0
3	1	21.0750	1	0	0	0	1	0	1	0	0	1
0	2	11.1333	1	0	0	0	1	1	0	0	0	1
1	0	30.0708	1	0	1	0	0	1	0	0	1	0

bingo，我们很成功地把这些类目属性全都转成 0，1 的数值属性了。

这样，看起来，是不是我们需要的属性值都有了，且它们都是数值型属性呢。

有一种临近结果的宠宠欲动感吧，莫急莫急，我们还得做一些处理，仔细看看 Age 和 Fare 两个属性，乘客的数值幅度变化，也太大了吧！！如果大家了解逻辑回归与梯度下降的话，会知道，各属性值之间 scale 差距太大，**将对收敛速度造成几万点伤害值！甚至不收敛！(╯‿╰)**...所以我们先用 scikit-learn 里面的 preprocessing 模块对这俩货做一个 scaling，所谓 scaling，其实就是将一些变化幅度较大的特征化到[-1,1]之内。

```
import sklearn.preprocessing as preprocessing
```

```

scaler = preprocessing.StandardScaler()
age_scale_param = scaler.fit(df['Age'])
df['Age_scaled'] = scaler.fit_transform(df['Age'], age_scale_param)
fare_scale_param = scaler.fit(df['Fare'])
df['Fare_scaled'] = scaler.fit_transform(df['Fare'], fare_scale_param)
df

```

- 1
- 2
- 3
- 4
- 5
- 6
- 7

1:

	Cabin_No	Cabin_Yes	Embarked_C	Embarked_Q	Embarked_S	Sex_female	Sex_male	Pclass_1	Pclass_2	Pclass_3	Age_scaled	Fare_scaled
		0	0	0	1	0	1	0	0	1	-0.561417	-0.5024
)		1	1	0	0	1	0	1	0	0	0.613177	0.78684
		0	0	0	1	1	0	0	0	1	-0.267768	-0.4888
)		1	0	0	1	1	0	1	0	0	0.392941	0.42073
		0	0	0	1	0	1	0	0	1	0.392941	-0.4863
		0	0	1	0	0	1	0	0	1	-0.427149	-0.4781
)		1	0	0	1	0	1	1	0	0	1.787771	0.39581
		0	0	0	1	0	1	0	0	1	-2.029659	-0.2240

恩，好看多了，万事俱备，只欠建模。马上就要看到成效了，哈哈。我们把需要的属性值抽出来，转成 `scikit-learn` 里面 `LogisticRegression` 可以处理的格式。

8. 逻辑回归建模

我们把需要的 `feature` 字段取出来，转成 `numpy` 格式，使用 `scikit-learn` 中的 `LogisticRegression` 建模。

```

from sklearn import linear_model
# 用正则取出我们要的属性值
train_df =
df.filter(regex='Survived|Age_.*|SibSp|Parch|Fare_.*|Cabin_.*|Embarke
d_.*|Sex_.*|Pclass_.*')
train_np = train_df.as_matrix()
# y 即 Survival 结果
y = train_np[:, 0]
# X 即特征属性值
x = train_np[:, 1:]
# fit 到 RandomForestRegressor 之中

```

```
clf = linear_model.LogisticRegression(C=1.0, penalty='l1', tol=1e-6)
clf.fit(X, y)
```

```
clf
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17

good, 很顺利, 我们得到了一个 model, 如下:

```
3]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
    intercept_scaling=1, max_iter=100, multi_class='ovr',
    penalty='l1', random_state=None, solver='liblinear', tol=1e-06,
    verbose=0)
```

先淡定! 淡定! 你以为把 test.csv 直接丢进 model 里就能拿到结果啊...骚年, 图样图森破啊! 我们的"test_data"也要做和"train_data"一样的预处理啊!!

```
data_test = pd.read_csv("/Users/Hanxiaoyang/Titanic_data/test.csv")
data_test.loc[ (data_test.Fare.isnull()), 'Fare' ] = 0# 接着我们对
test_data 做和 train_data 中一致的特征变换# 首先用同样的
RandomForestRegressor 模型填上丢失的年龄
tmp_df = data_test[['Age', 'Fare', 'Parch', 'SibSp', 'Pclass']]
null_age = tmp_df[data_test.Age.isnull()].as_matrix()# 根据特征属性 X
预测年龄并补上
X = null_age[:, 1:]
predictedAges = rfr.predict(X)
data_test.loc[ (data_test.Age.isnull()), 'Age' ] = predictedAges

data_test = set_Cabin_type(data_test)
```

```

dummies_Cabin = pd.get_dummies(data_test['Cabin'], prefix= 'Cabin')
dummies_Embarked = pd.get_dummies(data_test['Embarked'], prefix=
'Embarked')
dummies_Sex = pd.get_dummies(data_test['Sex'], prefix= 'Sex')
dummies_Pclass = pd.get_dummies(data_test['Pclass'], prefix= 'Pclass')

```

```

df_test = pd.concat([data_test, dummies_Cabin, dummies_Embarked,
dummies_Sex, dummies_Pclass], axis=1)
df_test.drop(['Pclass', 'Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'],
axis=1, inplace=True)
df_test['Age_scaled'] = scaler.fit_transform(df_test['Age'],
age_scale_param)
df_test['Fare_scaled'] = scaler.fit_transform(df_test['Fare'],
fare_scale_param)
df_test

```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25

Cabin_No	Cabin_Yes	Embarked_C	Embarked_Q	Embarked_S	Sex_female	Sex_male	Pclass_1	Pclass_2	Pclass_3	Age_scaled	Fare_scaled
	0	0	1	0	0	1	0	0	1	0.307495	-0.496637
	0	0	0	1	1	0	0	0	1	1.256225	-0.511497
	0	0	1	0	0	1	0	1	0	2.394702	-0.463335
	0	0	0	1	0	1	0	0	1	-0.261743	-0.481704
	0	0	0	1	1	0	0	0	1	-0.641235	-0.416740
	0	0	0	1	0	1	0	0	1	-1.248423	-0.471623
	0	0	1	0	1	0	0	0	1	-0.034048	-0.500221
	0	0	0	1	0	1	0	1	0	-0.337642	-0.117238
	0	1	0	0	1	0	0	0	1	-0.944829	-0.507390
	0	0	0	1	0	1	0	0	1	-0.717134	-0.204154
	0	0	0	1	0	1	0	0	1	-0.189852	-0.495444
	0	0	0	1	0	1	1	0	0	1.180327	-0.171000
	1	0	0	1	1	0	1	0	0	-0.565337	0.837349

不错不错，数据很 OK，差最后一步了。
下面就做预测取结果吧！！

```
test =
df_test.filter(regex='Age_*|SibSp|Parch|Fare_*|Cabin_*|Embarked_*
|Sex_*|Pclass_*')
predictions = clf.predict(test)
result =
pd.DataFrame({'PassengerId':data_test['PassengerId'].as_matrix(),
'Survived':predictions.astype(np.int32)})
result.to_csv("/Users/Hanxiaoyang/Titanic_data/logistic_regression_pr
edictions.csv", index=False)
```

- 1
- 2
- 3
- 4

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	1
5	897	0
6	898	1
7	899	0
8	900	1
9	901	0
10	902	0
11	903	0

啧啧，挺好，格式正确，去 make a submission 啦啦啦！

在 Kaggle 的 Make a submission 页面，提交上结果。如下：

2726	new	jeremyyeo	0.76555	1	Mon, 09 Nov 2015 22:45:26
2727	new	JustinL	0.76555	2	Mon, 09 Nov 2015 23:29:01 (-0.3h)
2728	new	NSK	0.76555	10	Tue, 10 Nov 2015 02:05:39 (-1.9h)
2729	new	MichelleNgan	0.76555	2	Tue, 10 Nov 2015 03:18:05
2730	new	Aakash Rana	0.76555	1	Tue, 10 Nov 2015 07:03:38
2731	new	hanxiaoyang	0.76555	3	Tue, 10 Nov 2015 09:39:25
Your Best Entry ↑ You improved on your best score by 0.00957. You just moved up 148 positions on the leaderboard. Tweet this!					
2732	↓245	sd tin	0.76077	4	Sun, 13 Sep 2015 08:48:27 (-44.4h)
2733	↓245	ChristopherKeune	0.76077	1	Fri, 11 Sep 2015 13:38:48
2734	↓245	🏠 🏠 🏠	0.76077	10	Wed, 16 Sep 2015 01:14:37 (-4.1d)
2735	↓245	MariaShen	0.76077	1	Mon, 14 Sep 2015 03:39:21
2736	↓245	Mary Nichol	0.76077	4	Wed, 16 Sep 2015 04:18:10 (-18.1h)

0.76555, 恩, 结果还不错。毕竟, 这只是我们简单分析处理过后出的一个 **baseline** 模型嘛。

9. 逻辑回归系统优化

9.1 模型系数关联分析

亲, 你以为结果提交上了, 就完事了?

我不会告诉你, 这只是万里长征第一步啊(泪牛满面)!!! 这才刚撸完 **baseline model** 啊!!! 还得优化啊!!!

看过 **Andrew Ng** 老师的 **machine Learning** 课程的同学们, 知道, 我们应该分析分析模型现在的状态了, 是过/欠拟合?, 以确定我们需要更多的特征还是更多数据, 或者其他操作。我们有一条很著名的 **learning curves** 对吧。

不过在现在的场景下, 先不着急做这个事情, 我们这个 **baseline** 系统还有些粗糙, 先再挖掘挖掘。

-

首先, **Name** 和 **Ticket** 两个属性被我们完整舍弃了(好吧, 其实是因为这俩属性, 几乎每一条记录都是一个完全不同的值, 我们并没有找到很直接的处理方式)。

-

-

然后, 我们想想, 年龄的拟合本身也未必是一件非常靠谱的事情, 我们依据其余属性, 其实并不能很好地拟合预测出未知的年龄。再一个, 以我们的日常经验, 小盆友和老人可能得到的照顾会多一些, 这样看的话, 年龄作为一个连续值, 给一个固定的系数, 应该和年龄是一个正相关或者负相关, 似乎体现不出两头受照顾的实际情况, 所以, 说不定我们把年龄离散化, 按区段分作类别属性会更合适一些。

-

上面只是我瞎想的, **who knows** 是不是这么回事呢, 老老实实先把得到的 **model** 系数和 **feature** 关联起来看看。

```
pd.DataFrame({"columns":list(train_df.columns)[1:],  
             "coef":list(clf.coef_.T)})
```

- 1

	coef	columns
0	[-0.344189431858]	SibSp
1	[-0.104924350555]	Parch
2	[0.0]	Cabin_No
3	[0.902071479485]	Cabin_Yes
4	[0.0]	Embarked_C
5	[0.0]	Embarked_Q
6	[-0.417226462259]	Embarked_S
7	[1.95649520339]	Sex_female
8	[-0.677484871046]	Sex_male
9	[0.341226064445]	Pclass_1
10	[0.0]	Pclass_2
11	[-1.19410912948]	Pclass_3
12	[-0.523774279397]	Age_scaled
13	[0.0844279740271]	Fare_scaled

首先，大家回去[前两篇文章](#)里瞄一眼公式就知道，这些系数为正的特征，和最后结果是一个正相关，反之为负相关。

我们先看看那些权重绝对值非常大的 feature，在我们的模型上：

- Sex 属性，如果是 female 会极大提高最后获救的概率，而 male 会很大程度拉低这个概率。
- Pclass 属性，1 等舱乘客最后获救的概率会上升，而乘客等级为 3 会极大地拉低这个概率。
- 有 Cabin 值会很大程度拉升最后获救概率(这里似乎能看到了一点端倪，事实上从最上面的有无 Cabin 记录的 Survived 分布图上看，即使有 Cabin 记录的乘客也有一部分遇难了，估计这个属性上我们挖掘还不够)
- Age 是一个负相关，意味着在我们的模型里，年龄越小，越有获救的优先权(还得回原数据看看这个是否合理)
- 有一个登船港口 S 会很大程度拉低获救的概率，另外俩港口压根就没啥作用(这个实际上非常奇怪，因为我们从之前的统计图上并没有看到 S 港口的获救率非常低，所以也许可以考虑把登船港口这个 feature 去掉试试)。
- 船票 Fare 有小幅度的正相关(并不意味着这个 feature 作用不大，有可能是我们细化的程度还不够，举个例子，说不定我们得对它离散化，再分至各个乘客等级上?)

噢啦，观察完了，我们现在有一些想法了，但是怎么样才知道，哪些优化的方法是 promising 的呢？

因为 test.csv 里面并没有 Survived 这个字段(好吧，这是废话，这明明就是我们要预测的结果)，我们无法在这份数据上评定我们算法在该场景下的效果...

而『每做一次调整就 make a submission，然后根据结果来判定这次调整的好坏』其实是行不通的...

9.2 交叉验证

重点又来了：

『要做交叉验证(cross validation)!』

『要做交叉验证(cross validation)!』

『要做交叉验证(cross validation)!』

恩，重要的事情说三遍。我们通常情况下，这么做 cross validation：把 train.csv 分成两部分，一部分用于训练我们需要的模型，另外一部分数据上看我们预测算法的效果。

我们用 scikit-learn 的 cross_validation 来帮我们完成小数据集上的这个工作。

先简单看看 cross validation 情况下的打分

```
from sklearn import cross_validation

#简单看看打分情况
clf = linear_model.LogisticRegression(C=1.0, penalty='l1', tol=1e-6)
all_data =
df.filter(regex='Survived|Age_*|SibSp|Parch|Fare_*|Cabin_*|Embarke
d_*|Sex_*|Pclass_*')
X = all_data.as_matrix()[1:]
y = all_data.as_matrix()[0]
print
cross_validation.cross_val_score(clf, X, y, cv=5)
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8

结果是下面酱紫的：

[0.81564246 0.81005587 0.78651685 0.78651685 0.81355932]

似乎比 Kaggle 上的结果略高哈，毕竟用的是不是同一份数据集评估的。

等等，既然我们要做交叉验证，那我们干脆先把交叉验证里面的 **bad case** 拿出来看看，看看人眼审核，是否能发现什么蛛丝马迹，是我们忽略了哪些信息，使得这些乘客被判定错了。再把 **bad case** 上得到的想法和前头系数分析的合在一起，然后逐个试试。

下面我们做数据分割，并且在原始数据集上瞄一眼 **bad case**：

```
# 分割数据，按照 训练数据:cv 数据 = 7:3 的比例
split_train, split_cv = cross_validation.train_test_split(df,
test_size=0.3, random_state=0)
train_df =
split_train.filter(regex='Survived|Age_|SibSp|Parch|Fare_|Cabin_|
*|Embarked_|Sex_|Pclass.*')# 生成模型
clf = linear_model.LogisticRegression(C=1.0, penalty='l1', tol=1e-6)
clf.fit(train_df.as_matrix()[:,1:], train_df.as_matrix()[:,0])
# 对 cross validation 数据进行预测

cv_df =
split_cv.filter(regex='Survived|Age_|SibSp|Parch|Fare_|Cabin_|E
mbarked_|Sex_|Pclass.*')
predictions = clf.predict(cv_df.as_matrix()[:,1:])

origin_data_train =
pd.read_csv("/Users/HanXiaoyang/Titanic_data/Train.csv")
bad_cases =
origin_data_train.loc[origin_data_train['PassengerId'].isin(split_cv[
predictions != cv_df.as_matrix()[:,0]]['PassengerId'].values)]
bad_cases

• 1
• 2
• 3
• 4
• 5
• 6
• 7
• 8
• 9
• 10
• 11
• 12
• 13
• 14
• 15
```

我们判定错误的 bad case 中部分数据如下：

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
14	15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	female	14.00	0	0	350406	7.8542	NaN	S
49	50	0	3	Arnold-Franchi, Mrs. Josef (Josefine Franchi)	female	18.00	1	0	349237	17.8000	NaN	S
55	56	1	1	Woolner, Mr. Hugh	male	NaN	0	0	19947	35.5000	C52	S
65	66	1	3	Moubarek, Master. Gerios	male	NaN	1	1	2661	15.2458	NaN	C
68	69	1	3	Andersson, Miss. Erna Alexandra	female	17.00	4	2	3101281	7.9250	NaN	S
85	86	1	3	Backstrom, Mrs. Karl Alfred (Maria Mathilda Gu...	female	33.00	3	0	3101278	15.8500	NaN	S
113	114	0	3	Jussila, Miss. Katriina	female	20.00	1	0	4136	9.8250	NaN	S
140	141	0	3	Boulos, Mrs. Joseph (Sultana)	female	NaN	0	2	2678	15.2458	NaN	C
204	205	1	3	Cohen, Mr. Gurshon "Gus"	male	18.00	0	0	A/5 3540	8.0500	NaN	S
240	241	0	3	Zabour, Miss. Thamine	female	NaN	1	0	2665	14.4542	NaN	C
251	252	0	3	Strom, Mrs. Wilhelm (Elna Matilda Persson)	female	29.00	1	1	347054	10.4625	G6	S

大家可以自己跑一遍试试，拿到 bad cases 之后，仔细看看。也会有一些猜测和想法。其中会有一部分可能会印证在系数分析部分的猜测，那这些优化的想法优先级可以放高一些。

现在有了“train_df”和“vc_df”两个数据部分，前者用于训练 model，后者用于评定和选择模型。可以开始可劲折腾了。

我们随便列一些可能可以做的优化操作：

- Age 属性不使用现在的拟合方式，而是根据名称中的『Mr』『Mrs』『Miss』等的平均值进行填充。
- Age 不做成一个连续值属性，而是使用一个步长进行离散化，变成离散的类目 feature。
- Cabin 再细化一些，对于有记录的 Cabin 属性，我们将其分为前面的字母部分(我猜是位置和船层之类的信息)和后面的数字部分(应该是房间号，有意思的事情是，如果你仔细看看原始数据，你会发现，这个值大的情况下，似乎获救的可能性高一些)。
- Pclass 和 Sex 俩太重要了，我们试着用它们去组出一个组合属性来试试，这也是另外一种程度的细化。
- 单加一个 Child 字段，Age<=12 的，设为 1，其余为 0(你去看看数据，确实小盆友优先程度很高啊)
- 如果名字里面有『Mrs』，而 Parch>1 的，我们猜测她可能是一个母亲，应该获救的概率也会提高，因此可以多加一个 Mother 字段，此种情况下设为 1，其余情况下设为 0
- 登船港口可以考虑先去掉试试(Q 和 C 本来就没权重，S 有点诡异)
- 把堂兄弟/兄妹 和 Parch 还有自己个数加在一起组一个 Family_size 字段(考虑到大家族可能对最后的结果有影响)
- Name 是一个我们一直没有触碰的属性，我们可以做一些简单的处理，比如说男性中带某些字眼的('Capt', 'Don', 'Major', 'Sir')可以统一到一个 Title，女性也一样。

大家接着往下挖掘，可能还可以想到更多可以细挖的部分。我这里先列这些了，然后我们可以使用手头上的“train_df”和“cv_df”开始试验这些 feature engineering 的 tricks 是否有效了。

试验的过程比较漫长，也需要有耐心，而且我们经常会面临很尴尬的状况，就是我们灵光一闪，想到一个 feature，然后坚信它一定有效，结果试验下来，效果还不如试验之前的结果。恩，需要坚持和耐心，以及不断的挖掘。

我最好的结果是在

『Survived~C(Pclass)+C(Title)+C(Sex)+C(Age_bucket)+C(Cabin_num_bucket)Mother+Fare+Family_Size』下取得的，结果如下(抱歉，博主君 commit 的时候手抖把页面关了，于是没截着图，下面这张图是在我得到最高分之后，用这次的结果重新 make commission 的，截了个图，得分是 0.79426，不是目前我的最高分哈，因此排名木有变...):

691	new	Tamas S.	0.80383	4	Tue, 10 Nov 2015 20:45:41 (-0.1h)
692	new	Jack Collins	0.80383	8	Wed, 11 Nov 2015 21:58:38 (-22.7h)
693	↑1353	BillyFung	0.80383	3	Wed, 11 Nov 2015 00:17:25
694	↑829	JyothiPriyan	0.80383	4	Wed, 11 Nov 2015 03:10:16
695	new	hanxiaoyang	0.80383	9	Thu, 12 Nov 2015 03:27:44 (-13.9h)
Your Best Entry ↑ Your submission scored 0.79426, which is not an improvement of your best score. Keep trying!					
696	new	dgt01	0.80383	3	Wed, 11 Nov 2015 16:41:58 (-0.9h)
697	new	Roman Minko	0.80383	43	Wed, 11 Nov 2015 16:26:47
698	↓30	yoTTaBo55	0.80383	16	Wed, 11 Nov 2015 17:37:35
699	↑1228	rehlezen	0.80383	14	Thu, 12 Nov 2015 01:06:46 (-6.5h)

9.3 learning curves

有一个很可能发生的问题是，我们不断地做 feature engineering，产生的特征越来越多，用这些特征去训练模型，会对我们的训练集拟合得越来越好，同时也可能在逐步丧失泛化能力，从而在待预测的数据上，表现不佳，也就是发生过拟合问题。

从另一个角度上说，如果模型在待预测的数据上表现不佳，除掉上面说的过拟合问题，也有可能是欠拟合问题，也就是说在训练集上，其实拟合的也不是那么好。

额，这个欠拟合和过拟合怎么解释呢。这么说吧：

- 过拟合就像是你班那个学数学比较刻板的同学，老师讲过的题目，一字不漏全记下来了，于是老师再出一样的题目，分分钟精确出结果。but 数学考试，因为总是碰到新题目，所以成绩不咋地。
- 欠拟合就像是，咳咳，和博主 level 差不多的差生。连老师讲的练习题也记不住，于是连老师出一样题目复习的周测都做不好，考试更是可想而知了。

而在机器学习的问题上，对于**过拟合**和**欠拟合**两种情形。我们优化的方式是不同的。

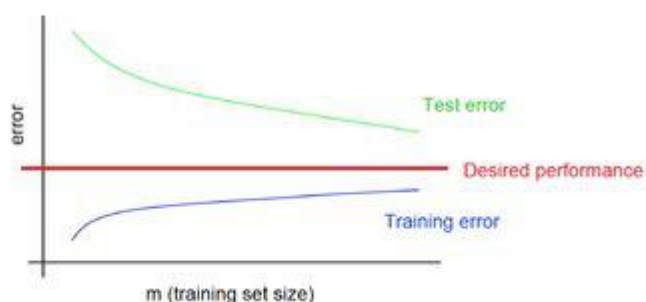
对过拟合而言，通常以下策略对结果优化是有用的：

- 做一下 **feature selection**，挑出较好的 **feature** 的 **subset** 来做 **training**
- 提供更多的数据，从而弥补原始数据的 **bias** 问题，学习到的 **model** 也会更准确

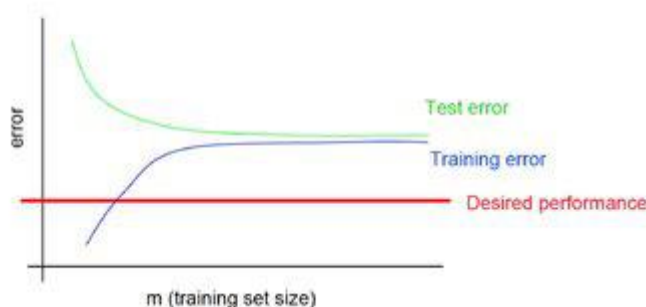
而对于欠拟合而言，我们通常需要更多的 **feature**，更复杂的模型来提高准确度。

著名的 **learning curve** 可以帮我们判定我们的模型现在所处的状态。我们以样本数为横坐标，训练和交叉验证集上的错误率作为纵坐标，两种状态分别如下两张图所示：过拟合(overfitting/high variance)，欠拟合(underfitting/high bias)

Typical learning curve for high variance:



Typical learning curve for high bias:



我们也可以把错误率替换成准确率(得分)，得到另一种形式的 **learning curve**(sklearn 里面是这么做的)。

回到我们的问题，我们用 `scikit-learn` 里面的 `learning_curve` 来帮我们分辨我们模型的状态。举个例子，这里我们一起画一下我们最先得到的 `baseline model` 的 `learning curve`。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.learning_curve import learning_curve
# 用 sklearn 的 learning_curve 得到 training_score 和 cv_score，使用
# matplotlib 画出 learning curve
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None, n_jobs=1,
                        train_sizes=np.linspace(.05, 1., 20),
                        verbose=0, plot=True):
    """
    画出 data 在某模型上的 learning curve.
    参数解释
    -----
    estimator : 你用的分类器。
    title : 表格的标题。
    X : 输入的 feature, numpy 类型
    y : 输入的 target vector
    ylim : tuple 格式的(ymin, ymax), 设定图像中纵坐标的最低点和最高点
    cv : 做 cross-validation 的时候，数据分成的份数，其中一份作为 cv 集，
    其余 n-1 份作为 training(默认为 3 份)
    n_jobs : 并行的任务数(默认 1)
    """

    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes,
        verbose=verbose)

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    if plot:
        plt.figure()
        plt.title(title)
        if ylim is not None:
            plt.ylim(*ylim)
        plt.xlabel(u"训练样本数")
        plt.ylabel(u"得分")
        plt.gca().invert_yaxis()
        plt.grid()
```

```

plt.fill_between(train_sizes, train_scores_mean -
train_scores_std, train_scores_mean + train_scores_std,
                  alpha=0.1, color="b")
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
test_scores_mean + test_scores_std,
                  alpha=0.1, color="r")
plt.plot(train_sizes, train_scores_mean, 'o-', color="b",
label=u"训练集上得分")
plt.plot(train_sizes, test_scores_mean, 'o-', color="r",
label=u"交叉验证集上得分")

plt.legend(loc="best")

plt.draw()
plt.show()
plt.gca().invert_yaxis()

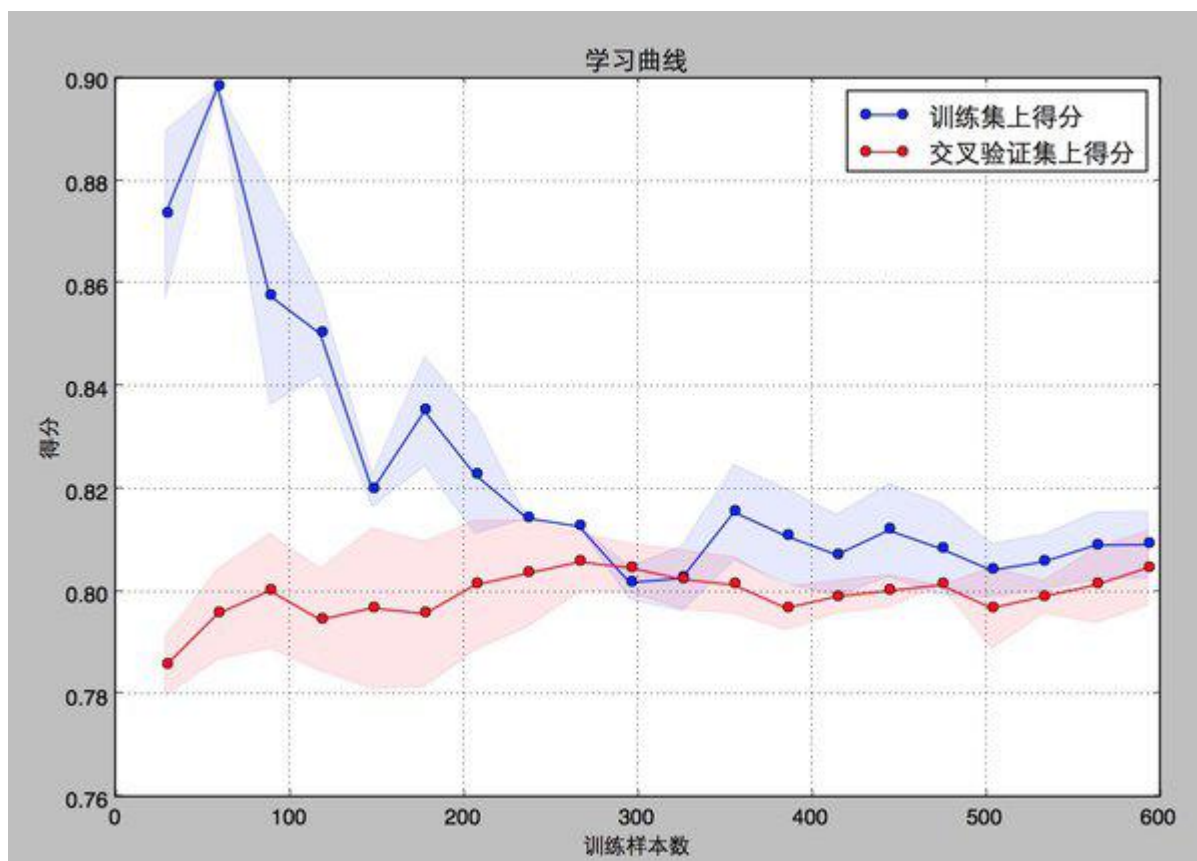
midpoint = ((train_scores_mean[-1] + train_scores_std[-1]) +
(test_scores_mean[-1] - test_scores_std[-1])) / 2
diff = (train_scores_mean[-1] + train_scores_std[-1]) -
(test_scores_mean[-1] - test_scores_std[-1])
return midpoint, diff

plot_learning_curve(clf, u"学习曲线", x, y)

```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19

- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45
- 46
- 47
- 48
- 49
- 50
- 51
- 52
- 53
- 54
- 55



在实际数据上看，我们得到的 learning curve 没有理论推导的那么光滑哈，但是可以大致看出来，训练集和交叉验证集上的得分曲线走势还是符合预期的。

目前的曲线看来，我们的 model 并不处于 overfitting 的状态(overfitting 的表现一般是训练集上得分高，而交叉验证集上要低很多，中间的 gap 比较大)。因此我们可以再做些 feature engineering 的工作，添加一些新产出的特征或者组合特征到模型中。

10. 模型融合(model ensemble)

好了，终于到这一步了，我们要祭出机器学习/数据挖掘上通常最后会用到的大杀器了。恩，模型融合。

『强迫症患者』打算继续喊喊口号...

【模型融合(model ensemble)很重要！】

【模型融合(model ensemble)很重要！】

【模型融合(model ensemble)很重要！】

重要的事情说三遍，恩，噢啦。

先解释解释，一会儿再回到我们的问题上哈。

啥叫模型融合呢，我们还是举几个例子直观理解一下好了。

大家都看过知识问答的综艺节目中，求助现场观众时候，让观众投票，最高的答案作为自己的答案的形式吧，每个人都有一个判定结果，最后我们相信答案在大多数人手里。

再通俗一点举个例子。你和你班某数学大神关系好，每次作业都『模仿』他的，于是绝大多数情况下，他做对了，你也对了。突然某一天大神脑子犯糊涂，手一抖，写错了一个数，于是...恩，你也只能跟着错了。

我们再来看看另外一个场景，你和你班 5 个数学大神关系都很好，每次都把他们作业拿过来，对比一下，再『自己做』，那你想想，如果哪天某大神犯糊涂了，写错了，but 另外四个写对了啊，那你肯定相信另外 4 人的是正确答案吧？

最简单的模型融合大概就是这么个意思，比如分类问题，当我们手头上有一堆在同一份数据集上训练得到的分类器(比如 **logistic regression**, **SVM**, **KNN**, **random forest**, 神经网络)，那我们让他们都分别去做判定，然后对结果做投票统计，取票数最多的结果为最后结果。

bingo，问题就这么完美的解决了。

模型融合可以比较好地缓解，训练过程中产生的过拟合问题，从而对于结果的准确度提升有一定的帮助。

话说回来，回到我们现在的问题。你看，我们现在只讲了 **logistic regression**，如果我们还想用这个融合思想去提高我们的结果，我们该怎么做呢？

既然这个时候模型没得选，那咱们就在数据上动动手脚咯。大家想想，如果模型出现过拟合现在，一定是在我们的训练上出现拟合过度造成的对吧。

那我们干脆就不要用全部的训练集，每次取训练集的一个 **subset**，做训练，这样，我们虽然用的是同一个机器学习算法，但是得到的模型却是不一样的；同时，因为我们没有任何一份子数据集是全是，因此即使出现过拟合，也是在子训练集上出现过拟合，而不是全体数据上，这样做一个融合，可能对最后的结果有一定的帮助。对，这就是常用的 **Bagging**。

我们用 **scikit-learn** 里面的 **Bagging** 来完成上面的思路，过程非常简单。代码如下：

```
from sklearn.ensemble import BaggingRegressor

train_df =
df.filter(regex='Survived|Age_|SibSp|Parch|Fare_|Cabin_|Embarked_|Sex_|Pclass_|Mother|Child|Family|Title')
train_np = train_df.as_matrix()
# y 即 Survival 结果
y = train_np[:, 0]
# X 即特征属性值
x = train_np[:, 1:]
```

```



# fit 到 BaggingRegressor 之中
clf = linear_model.LogisticRegression(C=1.0, penalty='l1', tol=1e-6)
bagging_clf = BaggingRegressor(clf, n_estimators=20, max_samples=0.8,
max_features=1.0, bootstrap=True, bootstrap_features=False, n_jobs=-1)
bagging_clf.fit(X, y)

test =
df_test.filter(regex='Age_|SibSp|Parch|Fare_|Cabin_|Embarked_|
|Sex_|Pclass_|Mother|Child|Family|Title')
predictions = bagging_clf.predict(test)
result =
pd.DataFrame({'PassengerId':data_test['PassengerId'].as_matrix(),
'Survived':predictions.astype(np.int32)})
result.to_csv("/Users/HanXiaoyang/Titanic_data/logistic_regression_ba
gging_predictions.csv", index=False)

```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20

然后你再 Make a submission，恩，发现对结果还是有帮助的。

690	↑1466	JamesSeymour	0.80383	82	Wed, 11 Nov 2015 11:33:46 (-17.5h)
691	new	Tamas S.	0.80383	4	Tue, 10 Nov 2015 20:45:41 (-0.1h)
692	new	Jack Collins	0.80383	4	Tue, 10 Nov 2015 23:32:07 (-0.2h)
693	↑1341	BillyFung	0.80383	3	Wed, 11 Nov 2015 00:17:25
694	↑825	JyothiPriyan	0.80383	4	Wed, 11 Nov 2015 03:10:16
695	new	 hanxiaoyang	0.80380	6	Wed, 11 Nov 2015 13:35:38
Your Best Entry ↑ You improved on your best score by 0.03823. You just moved up 2,029 positions on the leaderboard. Tweet this!					
696	↓77	tom hacker	0.79904	2	Sat, 12 Sep 2015 17:07:20
697	↓77	mamitasu	0.79904	2	Sat, 12 Sep 2015 17:16:54
698	↓77	zduey	0.79904	6	Sat, 12 Sep 2015 17:42:46
699	↓77	 Aishi jiang	0.79904	10	Sun, 13 Sep 2015 01:09:09 (-0.3h)
700	↓77	Jon Marlow	0.79904	12	Thu, 17 Sep 2015 14:23:48 (-3.6d)

11. 总结

文章稍微有点长，非常感谢各位耐心看到这里。

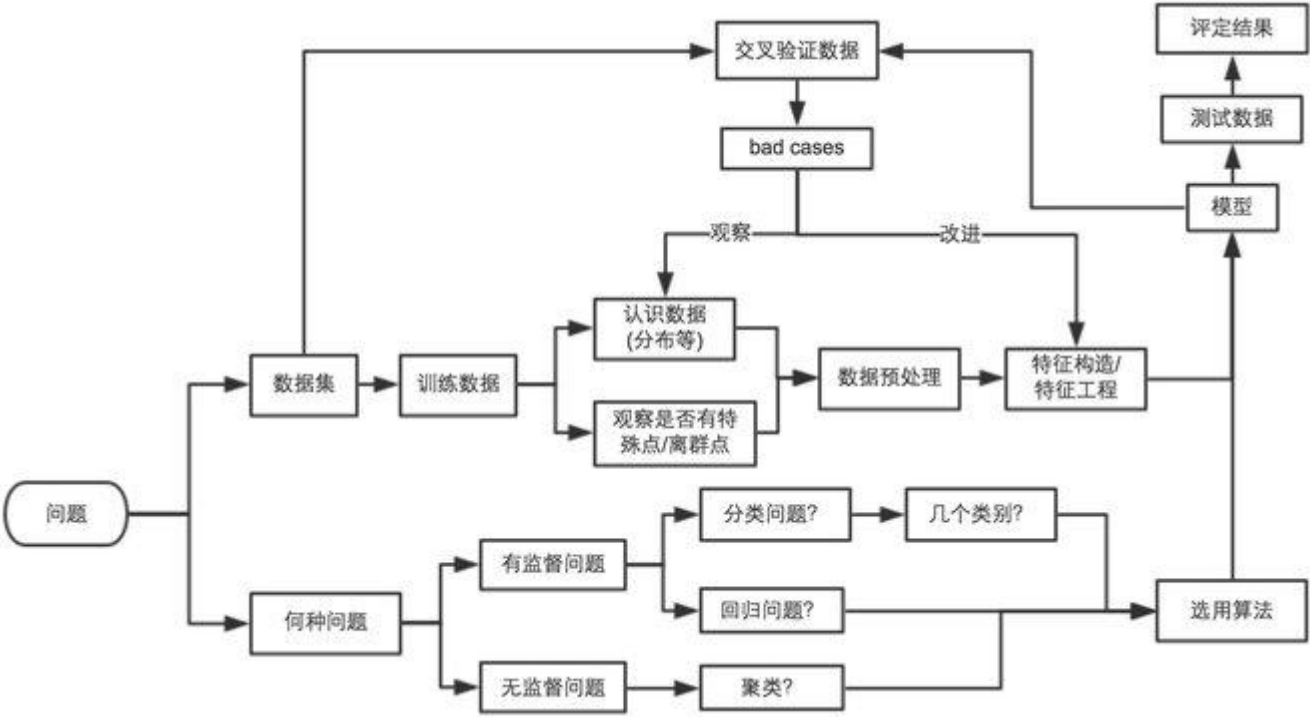
总结的部分，我就简短写几段，出现的话，很多在文中有对应的场景，大家有兴趣再回头看看。

对于任何的机器学习问题，不要一上来就追求尽善尽美，先用自己会的算法撸一个 **baseline** 的 **model** 出来，再进行后续的分析步骤，一步步提高。

在问题的结果过程中：

- 【对数据的认识太重要了！】
- 【数据中的特殊点/离群点的分析和处理太重要了！】
- 【特征工程(feature engineering)太重要了！】
- 【模型融合(model ensemble)太重要了！】

本文中用机器学习解决问题的过程大概如下图所示：



12. 关于数据和代码

本文中的数据和代码已经上传至 [github](#) 中，欢迎大家下载和自己尝试。