

机器学习课程设计实验报告

Kaggle Titanic Survival Prediction

小组成员：

甘润东 学号：1511417

公岩松 学号：1511419

一.实验目的

本实验旨在利用特征工程内容训练出预测出泰塔尼克号上给定 id 人员的存活情况的模型。

二.实验环境

Windows10 操作系统,
jupyter notebook(python3.6)

三.实验内容与结果

1、数据分析与处理

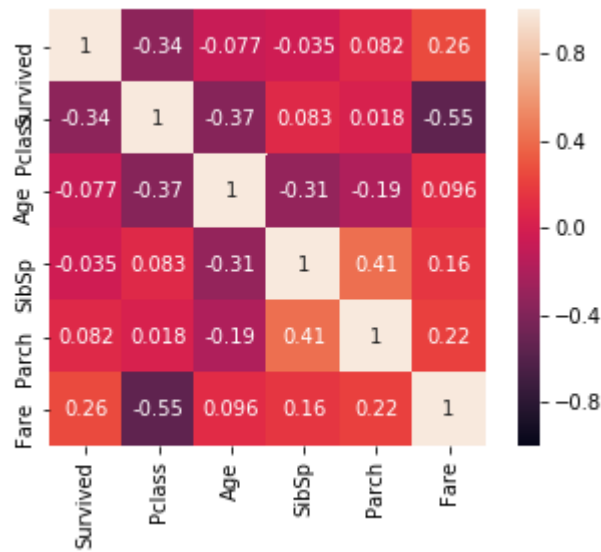
✓ 探索性数据分析与可视化

📊 数据总览

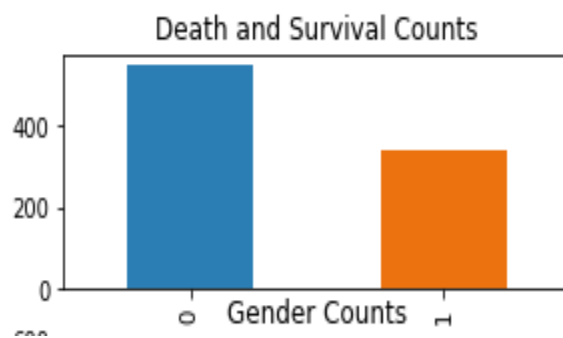
数据总览：各数据属性及非空个数：

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

热力相关图：展现出各数值属性之间的相关关系：



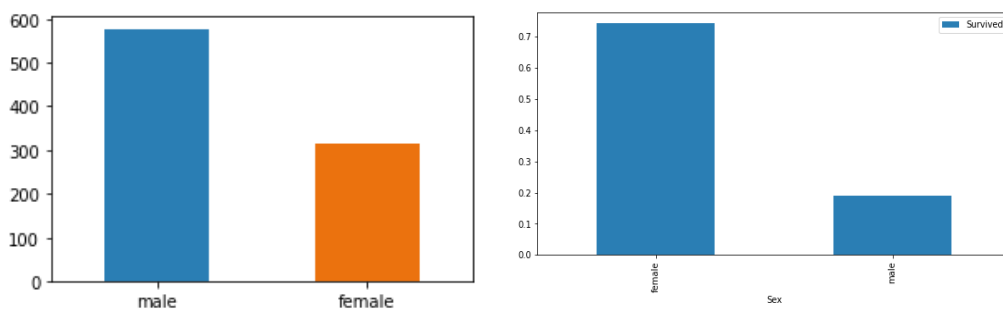
存活死亡人数对比：



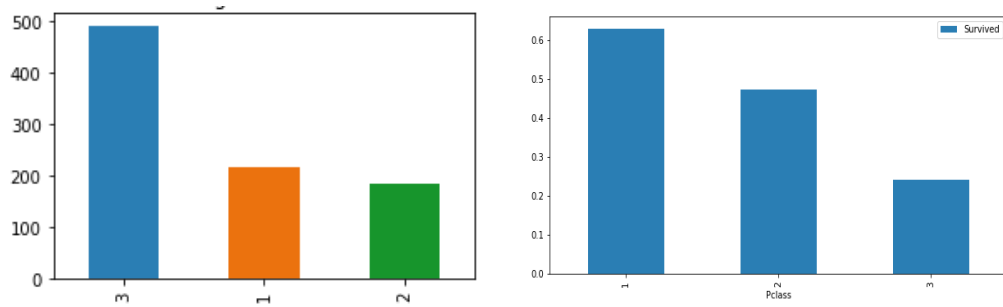
初步分析简单特征和生存率的关系

通过根据不同基本特征将人员划分成不同群体，并判断不同群体的存活状况，由此甄选出重要的人员特征作为后续分析的重点。

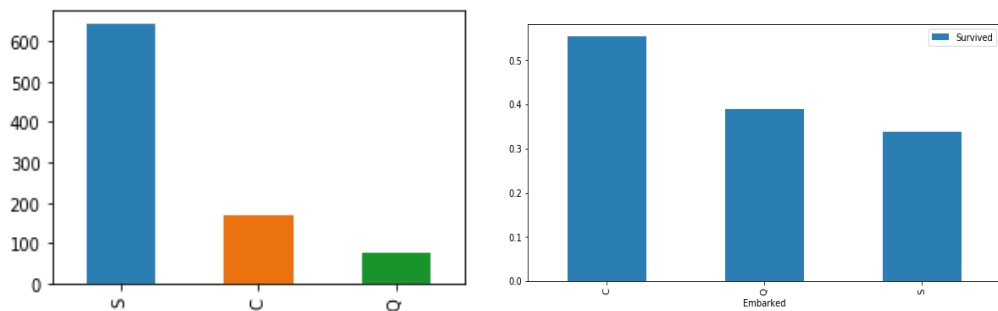
Sex：显然女性比男性拥有更高的生存概率。（左图为各属性人数，右图为各属性存活率，下面所有属性均是如此。）



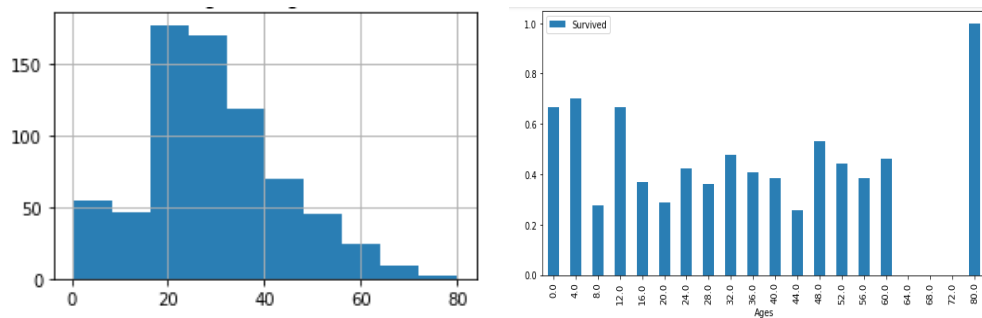
Pclass：阶级越高生存概率越大（1 阶级是最高阶级）。



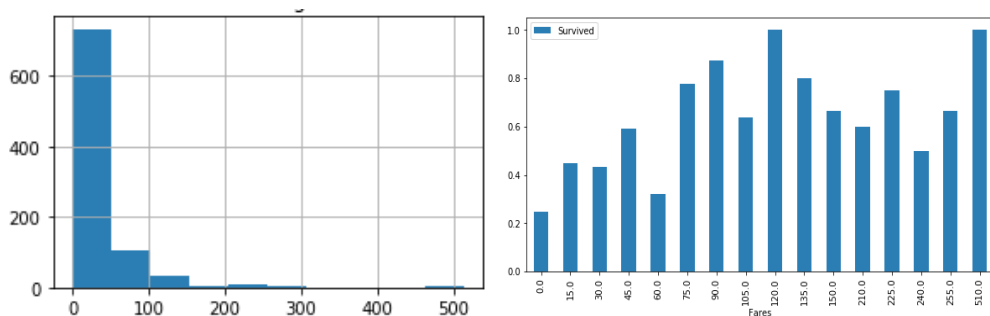
Embarked：S 港口人最多却存活率最低，C 港口存活率最高。



Age：未成年人拥有更高的生存概率（80 岁是个例，该分组只有一个人，二图中横轴 0.0 表示 0~4 岁，后面同理。）



Fare：支付较高价格船票的人拥有更高的生存概率（横轴意义与年龄相同）

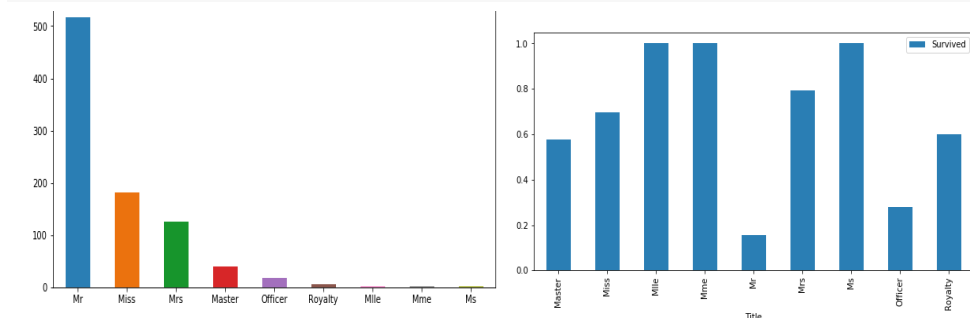


进一步的数据整合可视化

部分特征无法通过直接的群体划分得出结论，而且存在相关性的特征也无法通过单独分析而得出结论。对于这种特征需要进一步的分析论证。

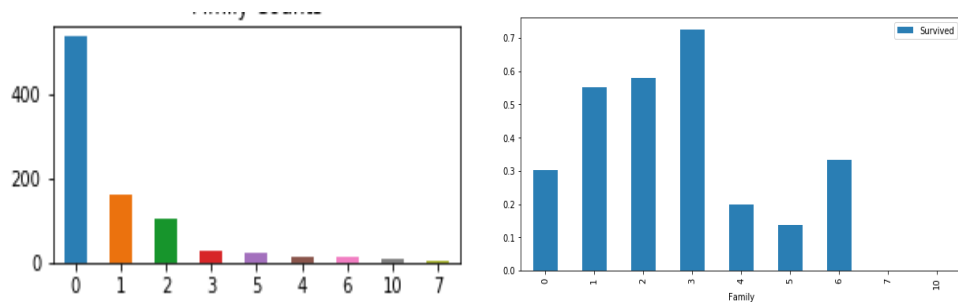
Name：姓名的处理，通过对数据的进一步分析，我们发现姓名当中蕴含着称谓，这在一方面反映出人员的阶级，年龄和职业。通过甄别分类，大致将人群分为：官员，贵族，已婚女性，未婚女性，成年男性，小孩。其代码片段如下：

```
# 替换称谓
df['Title'] = df['Name'].str.extract('^(.+)', expand=False).str.extract('^(.+?)\.', expand=False).str.strip()
df['Title'].replace(['Capt', 'Col', 'Major', 'Dr', 'Rev'], 'Officer', inplace = True)
df['Title'].replace(['Jonkheer', 'Don', 'Sir', 'the Countess', 'Dona', 'Lady'], 'Royalty', inplace = True)
df['Title'].replace(['Mme', 'Ms', 'Mrs'], 'Mrs', inplace = True)
df['Title'].replace(['Mlle', 'Miss'], 'Miss', inplace = True)
df['Title'].replace(['Mr'], 'Mr', inplace = True)
df['Title'].replace(['Master'], 'Master', inplace = True)
```



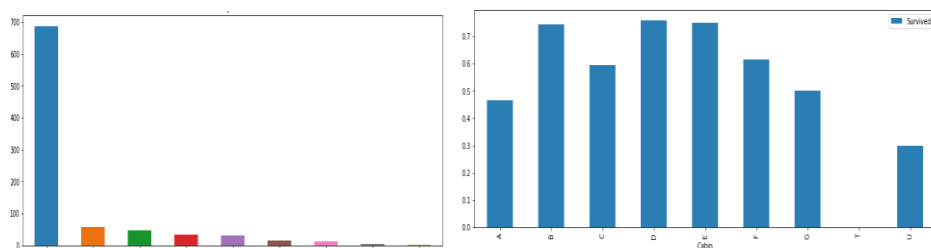
由图可以看出不同称谓对存活率影响比较明显，尤其是 Mr 在这个属性，人数最多，但存活率非常低，对结果有着非常大的影响，因此决定对数据新增“Title”属性加入到特征属性之中。

Family：通过原始数据我们可以得到两个有关家庭人数的数据，SibSp 和 Parch，分别表示兄弟姐妹伴侣数和父母孩子数。单独的属性对存活率的影响并不算突出，我们决定将两者组合起来 FamilySize = Parch + SibSp 获得家庭规模数，对该数据特征进行分析。



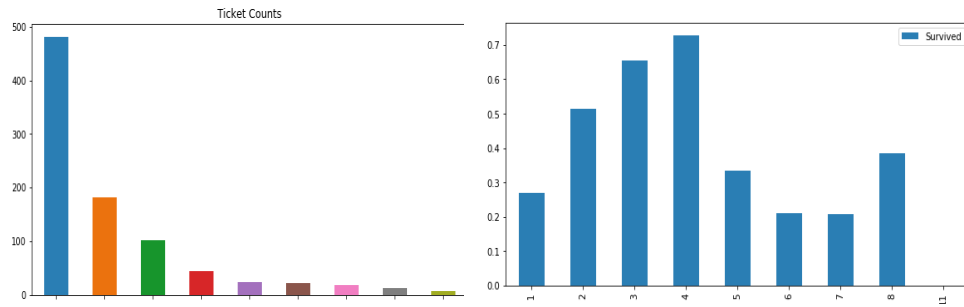
可以看到，家庭规模的大小与存活率有着不小的关系，我们也考虑将这个属性加入到最终的特征属性中。

Cabin：这个属性的处理参考了中期答辩时同学提到的泰坦尼克号舱位与出口之间的关系，首先因为 Cabin 数据有缺失，对该数据进行填充 "Unkown",抽取 Cabin 属性的首字母作为新属性，这个属性表现了乘客的舱位大致在何处。



可以看到，舱位与存活率也有一定的关系，也可以将其作为最后的特征属性中。

Ticket：这个属性种类繁多且不易找到与存活率之间的关系，我们经查阅大量资料，发现将训练集与测试集结合起来，计算票号相同的人数，如果与自己票号相同的人大多数都活了下来，那么测试集中的人也更容易存活，反之同理。关系图示如下：



至此，数据的初步可视化基本完成，之后的特征工程参考图示进行特征的选择。

✓ 数据清洗

🚩 缺失值

经过对数据的观察和统计得出，不符合要求，或者有缺失值的实行共 4 个，分别是：Age, Fare, Embarked, Cabin。

a, Age

对于年龄数据，其缺失值数量较大，实验最初选用 Sex,Pclass,Title 将人群划分为多个群体，然后选取每个群体人员年龄的中位数进行数据填充，但是这样的填充方式在一定程度上违背了客观规律，在查阅资料之后，小组决定使用 Sex,Pclass,Title 进行随机森林建模，进行缺失年龄的预测。

选择 Sex,Pclass,Title 这三个属性的原因是，首先 Title 作为称谓，其中就隐含了年龄的因素，举例来说如果一个人的称谓是 Miss，那么可以初步判定出她是一位年龄在 30 岁以下的女性。而 Pclass 这个属性，在前文的热力相关图中可以明显的看出与年龄有着负相关的关系，我们也选作用于预测的属性之一，至于年龄，我们认为这是重要属性之一，也对年龄有着重要影响，所以也选作用于预测的属性。

其代码片段如下：

```
#用随机森林预测年龄
age_df = df[['Age', 'Pclass', 'Sex', 'Title']]
age_df = pd.get_dummies(age_df)
known_age = age_df[age_df.Age.notnull()].as_matrix()
unknown_age = age_df[age_df.Age.isnull()].as_matrix()
y = known_age[:, 0]
X = known_age[:, 1:]
rfr = RandomForestRegressor(random_state=0, n_estimators=100, n_jobs=-1)
rfr.fit(X, y)
predictedAges = rfr.predict(unknown_age[:, 1::])
df.loc[ (df.Age.isnull()), 'Age' ] = predictedAges
```

b, Fare

对于票价属性, Fare 缺失量为 1, 缺失 Fare 信息的乘客的 Embarked 为 S, Pclass 为 3, 我们决定用 Embarked = 'S', Pclass = 3 这个分组平均数填充。选取这两个属性的原因也自然是从感性上这两个属性对 Fare 的影响最大。举例来说, 阶级越高越容易买贵的票 (这点在上面的相关图得以证实)。

其代码片段如下：

```
#Fare缺失量为1, 缺失Fare信息的乘客的Embarked为S, Pclass为3 用这个分组平均数填充
fare_mean = df[(df['Embarked'] == "S") & (df['Pclass'] == 3)].Fare.mean()
df['Fare'] = df['Fare'].fillna(fare_mean)
```

c, Embarked

对于港口属性, 有两个缺失值, 我们直接用众数'S'填充。这样做的原因很简单, 通常来说, 缺失值为'S'的概率最大。

d, Cabin

我们暂且把空值 Cabin 设置为'Unknown', 这样做的原因前文中已经说明

连续变量离散化

对于 Age 和 Fare 这两个属性, 由于不太好选取离散化间隔范围, 很

难去加强特征对结果的影响，稍微选取失误就非常容易造成结果的下降，而且会增大特征的个数，模型筛选特征时也容易造成损失，所以这两个连续变量不做离散化处理。

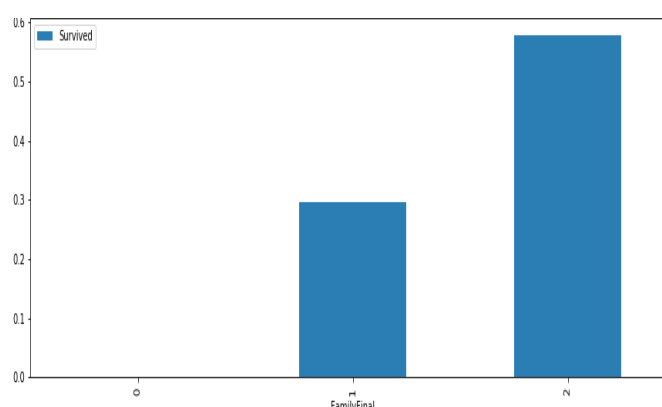
其余，我们把前文数据探索时得到的 TicketGroup 和 FamilySize 也看作连续变量，虽然这两个属性的值均为整数，但其范围比较大，而且从意义上难以确定其具体上界，所以我们按照连续变量进行处理。

然而前文图示所看到的结果，这两个变量与结果没有明显的正负相关关系，但是可以看到 FamilySize 这个属性，1-3 之间存活率非常高，0 或者 4-6 之间，存活率较低，但有存活例子，然而当大于 6 时，存活率为 0。所以我们将这个属性离散成这三个范围。最后构造出一个正相关的新属性，便于最后模型的适应性。

代码片段如下：

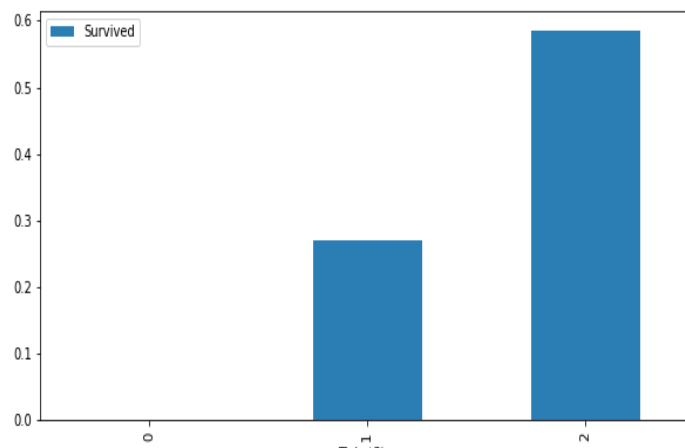
```
#在1到3范围内存活率最高赋值2 大于6存活率最低赋值0  
df['Family'] = df['FamilySize'].apply(lambda x: 0 if x > 6 else (2 if x >=1 and x <=3 else 1))
```

构造出的属性与存活率之间的关系如下：



同样，对于 TicketGroup 属性，我们也将其划分成 3 各部分，分别是 2-4、1 或 5-8，8 以上。其代码片段和分组与存活率之间的关系如下图所示：

```
#同Family
df['TicketGroup'] = df['TicketGroup'].apply(lambda x: 0 if x > 8 else (2 if x >= 2 and x <= 4 else 1))
```



🚦 连续变量标准化

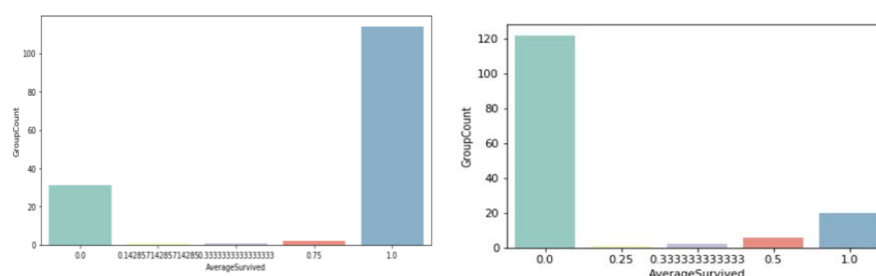
对于 Fare 和 Age 这两个属性，由于范围非常的大，我们认为直接经过模型处理并不十分妥当，所以我们对其进行了标准化，将数值都缩小到 $(-1, 1)$ 这个范围中，这样做有好处是：使某些需要递归的模型收敛更加迅速，当然某种程度上也增加了数据的精确性。其处理代码如下：

```
#对连续数据进行标准化处理
scaler = preprocessing.StandardScaler()
df['Fare'] = scaler.fit_transform(df.filter(['Fare']))
df['Age'] = scaler.fit_transform(df.filter(['Age']))
```

🚦 惩罚性修改

将姓氏相同的乘客分为一组，从人数大于 1 的群组中获取将成年男性和女性，孩童分离开。

从这些数据中，我们发现不同群组孩童，女性的生存率呈现非常极端的分化。



可以看出不同姓氏群体，同组儿童和成年女性的生存率或者 0 或者 1。并且大多数存活。同样的道理，同组的成年男性的也是居于两级分布，但是其存活率为 0 的分布更高。我们认为少数的男性存活和女性（儿童）死亡是不符合规律的，是数据中的噪音，所以我们对这些反常的数据进行惩罚性修改，其具体代码如下：

```
#因为普遍规律是女性和儿童幸存率高，成年男性幸存率低，所以我把不符合普遍规律的反常组及
Female_Child_Group = Female_Child_Group.groupby('Surname')['Survived'].mean()
Dead_List=set(Female_Child_Group[Female_Child_Group.apply(lambda x:x==0)].index)
Male_Adult_List=Male_Adult_Group.groupby('Surname')['Survived'].mean()
Survived_List=set(Male_Adult_List[Male_Adult_List.apply(lambda x:x==1)].index)

#为了使处于这两种反常组中的样本能够被正确分类，对测试集中处于反常组中的样本的Age, Title
train=train.loc[train['Survived'].notnull()]
test=test.loc[test['Survived'].isnull()]
test.loc[(test['Surname'].apply(lambda x:x in Dead_List)), 'Sex'] = 'male'
test.loc[(test['Surname'].apply(lambda x:x in Dead_List)), 'Age'] = 60
test.loc[(test['Surname'].apply(lambda x:x in Dead_List)), 'Title'] = 'Mr'
test.loc[(test['Surname'].apply(lambda x:x in Survived_List)), 'Sex'] = 'female'
test.loc[(test['Surname'].apply(lambda x:x in Survived_List)), 'Age'] = 5
test.loc[(test['Surname'].apply(lambda x:x in Survived_List)), 'Title'] = 'Miss'
```

✓ 特征工程

✚ 去除无用属性：

经过前两步的处理，我们不仅有了原始属性，还构造出了很多新的属性，这使得有些属性有着重复的意义，我们必须去除一些无用属性。其代码如下：

```
#drop掉无用属性
df = df.drop(['PassengerId'], axis=1)
df = df.drop(['Ticket'], axis=1)
df = df.drop(['FamilySize'], axis=1)
df = df.drop(['Parch'], axis=1)
df = df.drop(['SibSp'], axis=1)
df = df.drop(['Name'], axis=1)
df = df.drop(['Surname'], axis=1)
df = df.drop(['FamilyGroup'], axis=1)
```

最终我们留下的属性有： Age Fare Pclass Sex Embarked Title Family

Cabin(首字母) TicketGroup Title

独热编码：

目前，Sex, Embarked, Cabin, Title 这几个属性存放的都是非数值值，我们对其进行独热编码，其方法是使用 N 位状态寄存器来对 N 个状态进行编码，每个状态都有它独立的寄存器位，并且在任意时候，其中只有一位有效。其代码片段如下：

```
#独热编码  
df = pd.get_dummies(df)
```

最终我们得到了 26 个特征属性。

2、实验方法与分析

✓ 模型选择

由于查阅了很多相关资料，借鉴了很多其他人的成果，这些都显示随机森林这个模型是对于这个问题最有效的模型，由此我们并没有尝试很多模型，我们分别尝试了逻辑回归，SVM，和随机森林，经过初步测试，在都没有具体的调整参数时，随机森林比其余两个训练出的模型效果要好 1~4 个百分点。

之后我们尝试过模型融合，用投票的方式得到最终的预测结果，结果也并不理想，最终预测结果低 2~3 个百分点。所以我们也放弃了这种做法。

最终我们决定用随机森林作为我们训练模型的方法。

✓ 模型原理

2001 年 Breiman 把分类树组合成随机森林 (Breiman 2001a)，即在变量（列）的使用和数据（行）的使用上进行随机化，生成很多分类树，再汇总分类树的结果。随机森林在运算量没有显著提高的前提下提高了

预测精度。随机森林对多元共线性不敏感，结果对缺失数据和非平衡的数据比较稳健，可以很好地预测多达几千个解释变量的作用（Breiman 2001b），被誉为当前最好的算法之一（Iverson et al. 2008）。

随机森林顾名思义，是用随机的方式建立一个森林，森林里面有很多的决策树组成，随机森林的每一棵决策树之间是没有关联的。在得到森林之后，当有一个新的输入样本进入的时候，就让森林中的每一棵决策树分别进行一下判断，看看这个样本应该属于哪一类（对于分类算法），然后看看哪一类被选择最多，就预测这个样本为那一类。

✓ 参数调整

我们在参数调整时尝试过 Grid Search 方法进行调优，但是对于这个问题，最终得出的参数并不是我们最终结果最好的参数。其代码片段如下：

```
# pipe=Pipeline([('select', SelectKBest(k=20)),
#                ('classify', RandomForestClassifier(random_state = 10, max_features = 'sqrt'))])

# param_test = {'classify__n_estimators':list(range(20, 50, 2)),
#               'classify__max_depth':list(range(3, 60, 3))}
# gsearch = GridSearchCV(estimator = pipe, param_grid = param_test, scoring='roc_auc', cv=10)
# gsearch.fit(train_data_X, train_data_Y)
```

我们最后在得到的最佳参数附近，选取了如下参数进行训练，得到了最终最高的结果：

```
Pipeline(memory=None,
      steps=[('selectkbest', SelectKBest(k=20, score_func=<function f_classif at 0x000001FF358EAD90>)),
            stClassifier(bootstrap=True, class_weight=None, criterion='gini',
                        max_depth=6, max_features='sqrt', max_leaf_nodes=None,
                        min_impurity_decreas...estimators=26, n_jobs=1,
                        oob_score=False, random_state=10, verbose=0, warm_start=True))])
```

3、结果分析与讨论

✓ 交叉验证集结果

我们将前 700 个数据当作训练集，之后的训练及作为测试集，我们

最终的交叉验证的代码和结果如下：

```
In [14]: #交叉验证
select = SelectKBest(k = 20)
clf = RandomForestClassifier(random_state = 10, warm_start = True,
                             n_estimators = 26,
                             max_depth = 6,
                             max_features = 'sqrt')

pipeline = make_pipeline(select, clf)
pipeline.fit(train_data_X[:700], train_data_Y[:700])
predictions = pipeline.predict(train_data_X[700:])
fail_values = train_data_Y[700:] - predictions
accuracy = fail_values.value_counts()[0]/len(fail_values)
accuracy

Out[14]: 0.8534031413612565
```

✓ 测试集结果

最终提交的正确率是 0.84210

4、难点与创新点

✓ 难点

我们认为实验的难点在于以下两个部分：

✚ 对数据的进阶处理

对于与结果之间有明显关系的属性，比如：Age，Sex 等，对于较难发现与结果之间关系的属性的处理，是这次实验的难点。我们很难用肉眼发现 Ticket 和 Name 与结果之间有什么关系，但是我们最终通过观察，查阅资料和借鉴他人的成果，找到了他们对结果的重要影响。我们经过测试，如果没有对 Ticket 共票数的挖掘，我们的结果将降低 3 个百分点。

✚ 参数的调整

参数对结果的影响非常大，在没有经过参数调整时我们的结果只能在 81 左右徘徊，之后我们通过中期答辩了解到了 Grid Search 这项技术，于是我们应用到了我们的实验中，但是我们发现，这样搜索出的参数并不是最优秀的，最后经过几次尝试，我们在搜索到的最佳参数附近，找

到了一个最终得分更高的参数。

✓ 创新点


这次实验的创新点就在于对数据的惩罚性修改，这并不是我们想到的，我们借鉴了其他人的成果，把它加入到了我们的实验之中，我们也从中学习到了对于某些不符合常理的，或者极端个别的案例，我们可以对其进行惩罚性的修改，这样可以加强训练的效果，最终也体现到了最终的预测结果之中，如果没有惩罚性修改。我们的结果将降低 2 个百分点。

5、注册帐户名以及最终排名

注册帐户名：wdyiwdwd

最终结果及排名截图：

Name	Submitted	Wait time	Execution time	Score
result.csv	a minute ago	0 seconds	0 seconds	0.84210
Complete				
Jump to your position on the leaderboard				

120	▲2092	wdyiwdwd		0.84210	33	2m
-----	-------	----------	---	---------	----	----

Your Best Entry ▲

Your submission scored 0.84210, which is not an improvement of your best score. Keep trying!

结果为：0.84210

排名为：120

四.提交文件说明

- 1，文件一：‘组员名单.pdf’，介绍了组员以及具体分工。
- 2，文件二：‘实验报告.pdf’，实验报告的具体内容
- 3，文件夹三：实验代码

其中：

‘plot.py’是数据可视化的代码，此报告中大部分用图可从中生成。

‘final.py’是最终训练和预测的代码，我们实验的结果就是由这段代码生成。

另外：还有对应的.ipynb 文件和.html 文件，这是因为我们的开发环境是在 jupyter notebook 中，代码文件的原格式为.ipynb，为了使检查时环境没有问题，我们用该文件生成了对应的.py 文件和.html 文件。.html 文件即为开发时我们看到的最终结果。

4，文件夹四：最后提交结果文件以及排名截图。

其中：result.csv 是最终提交文件。其他两个是图片格式的文件，是最终结果的截图。

5，文件五：中期答辩 ppt