

# Final Project: Titanic - Machine Learning from Disaster

---

作者: 付大为 学号: 2201110122

## 背景

---

泰坦尼克号的沉没是历史上最臭名昭著的海难之一。

1912 年 4 月 15 日，在处女航期间，被广泛认为“永不沉没”的皇家邮轮泰坦尼克号在与冰山相撞后沉没。不幸的是，船上的每个人都没有足够的救生艇，导致 2224 名乘客和船员中的 1502 人死亡。

虽然幸存有一些运气因素，但似乎有些人比其他入更有可能活下来。

在这个挑战中，我们要建立一个预测模型来回答这个问题：“什么样的人更有可能生存？”使用乘客数据（即姓名、年龄、性别、社会经济阶层等）。

## 数据集

---

### 下载地址

本次project用到的数据集来自kaggle的[Titanic](#)竞赛, 下载地址[见此](#). 我已经把整个数据放在附件 `data.zip` 文件中

# 描述

在本次比赛中，我们有两个相似的数据集，其中包括姓名、年龄、性别、社会经济阶层等乘客信息。一个数据集名为 `train.csv`，另一个数据集名为 `test.csv`。

`train.csv` 将包含机上部分乘客（准确地说是 891 人）的详细信息，重要的是，将揭示他们是否幸存，也称为“基本事实”。

`test.csv` 数据集包含类似的信息，但没有透露每位乘客的“基本事实”。预测这些结果是我们接下来的工作。

使用您在数据中找到的训练集 `train.csv`，预测机上其他 418 名乘客（在 `test.csv` 中找到）是否幸存。

## 变量解释

变量	定义	KEY
PassengerId	乘客ID	
Survived	生存	0=否, 1=是
Pclass	机票等级	1 = 第一, 2 = 第二, 3 = 第三
Name	姓名	
Sex	性别	
Age	以岁数为单位的年龄	
SibSp	泰坦尼克号上的兄弟姐妹/配偶数量	
Parch	登上泰坦尼克号的父母/孩子	
Ticket	票号	
Fare	旅客票价	
Cabin	客舱号	
Embarked	登船港口	C = Cherbourg, Q = Queenstown, S = Southampton

**Pclass:** 社会经济地位 (SES) 的代称

1st = 上层

2nd = 中产

3rd = 底层

**Age:** 如果年龄小于 1，则年龄为小数。如果年龄是估计的，则采用 xx.5 的形式

**sibsp:** 数据集以这种方式定义家庭关系...

Sibling = 兄弟，姐妹，继兄，继妹

Spouse = 丈夫，妻子（情妇和未婚夫被忽略）

**Parch:** 数据集以这种方式定义家庭关系...

Parent = 母亲、父亲

Child = 女儿、儿子、继女、继子

有些孩子只和保姆一起旅行，因此对他们来说 Parch=0。

## 算法选择

---

我们将选择并比较以下三种算法的结果

1. Logistic Regression
2. Random Forest
3. Xgboost

## 代码大纲

---

### 0. 加载数据集-训练集和测试集

导入第三方库并读取样本数据：

```

1 import pandas as pd
2 df_train = pd.read_csv('./data/train.csv')
3 df_test = pd.read_csv('./data/test.csv')

```

## 1. 数据清洗——填写“Age”、“Cabin”和“Embarked”中的缺失值

1.1 对于Age，我们知道上救生艇时“女士和孩子优先”的基本原则，所以我们相信年龄、性别和生存有着内在的联系。

在训练集中，我们按 (Sex, Survived) 对训练数据进行分组，并通过从相应的 (Sex, Survived) 组中使用bootstrap重采样方法来填充年龄的缺失值

```

1 import random
2
3 age = {
4     'male': {
5         1: df_train['Age'][(df_train['Survived']==1) &
6 (df_train['Sex']=='male') & ~df_train['Age'].isnull()].values,
7         0: df_train['Age'][(df_train['Survived']==0) &
8 (df_train['Sex']=='male') & ~df_train['Age'].isnull()].values
9     },
10    'female': {
11        1: df_train['Age'][(df_train['Survived']==1) &
12 (df_train['Sex']=='female') & ~df_train['Age'].isnull()].values,
13        0: df_train['Age'][(df_train['Survived']==0) &
14 (df_train['Sex']=='female') & ~df_train['Age'].isnull()].values
15    }
16 }
17
18 for i in df_train['Age'][df_train['Age'].isnull()].index:
19     df_train['Age'][i] = random.choice(age[df_train['Sex'][i]]
20 [df_train['Survived'][i]])

```

在测试集中，我们将测试数据按性别分组，并用训练集中同性别组的平均值填充年龄的缺失值

```

1 mean_age = {
2     sex: df_train['Age'][df_train['Sex']==sex].mean() for sex in
   df_train['Sex'].unique()
3 }
4 for i, v in df_test['Age'][df_test['Age'].isnull()].items():
5     df_test['Age'][i] = mean_age[df_test['Sex'][i]]

```

## 1.2 对于Cabin，因为大多数值都被遗漏了，我们决定使用-1来填充训练集和测试集的所有遗漏值

在训练集中

```

1 df_train.fillna({'Cabin': -1}, inplace=True)

```

在测试集中

```

1 df_test.fillna({'Cabin': -1}, inplace=True)

```

## 1.3 对于 Embarked，只遗漏了两个值，所以使用 bootstrap 非常直接有效

在训练集中，我们通过对训练集使用bootstrap重采样来填充缺失值

```

1 embarked = df_train['Embarked']
   [~df_train['Embarked'].isnull()].values
2 for i in df_train['Embarked'][df_train['Embarked'].isnull()].index:
3     df_train['Embarked'][i] = random.choice(embarked)

```

在测试集中，我们仍然通过bootstrap重采样训练集来填充缺失值

```

1 for i in df_test['Embarked'][df_test['Embarked'].isnull()].index:
2     df_test['Embarked'][i] = random.choice(embarked)

```

## 1.4 额外填充测试集中缺失的Fare值

我们使用Pclass相同组的训练集的Fare平均值进行填充

```
1 for i in df_test['Fare'][df_test['Fare'].isnull()].index:
2     df_test['Fare'][i] = df_test['Fare']
    [df_test['Pclass']==df_test['Pclass'][i]].mean()
```

## 2. 分离特征和目标

从训练集和测试集中提取features和targets如下

```
1 df_train_features, y_train = df_train.drop(['PassengerId',
    'Survived', 'Name', 'Ticket'], axis=1), df_train['Survived'].values
2 df_test_features = df_test.drop(['PassengerId', 'Name', 'Ticket'],
    axis=1)
```

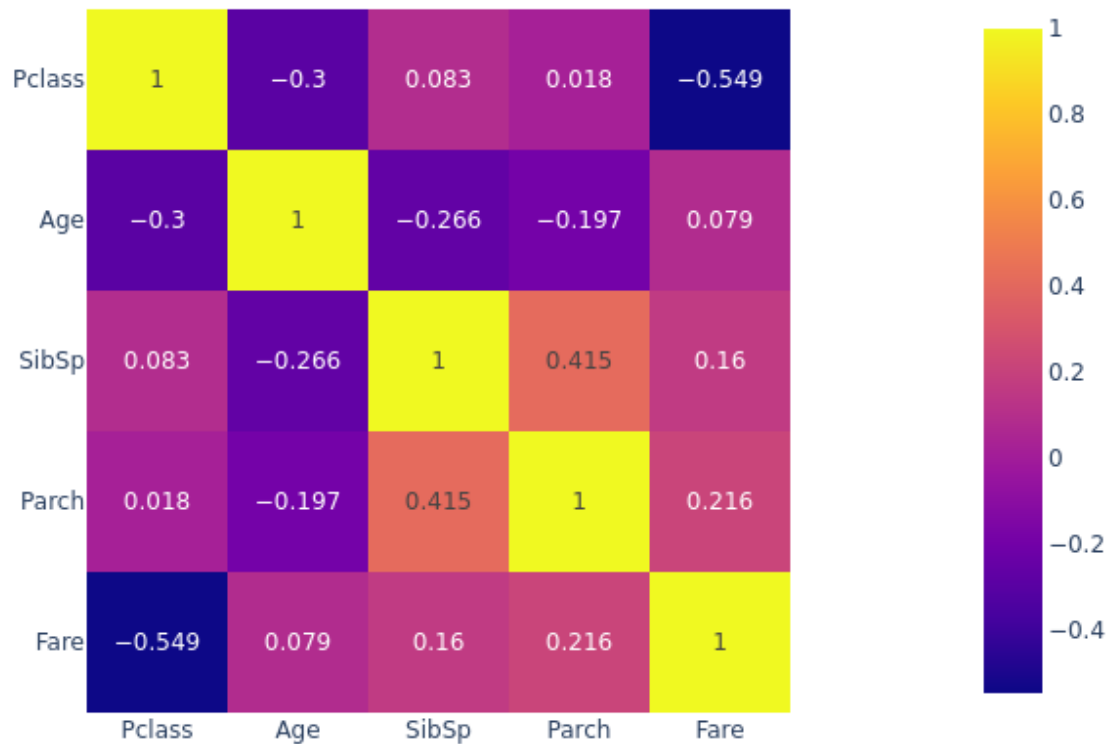
## 3. 数据预处理——数值型特征缩放和分类型特征编码

### 3.1 数值型特征缩放：标准化（也称为“Z-Score Normalization”）

训练集中, 首先可视化数值型特征的关联矩阵

```
1 import plotly.express as px
2 import os
3
4 fig = px.imshow(df_train_features.corr().round(3), text_auto=True)
5 fig.show()
6 if not os.path.exists('./plots'):
7     os.mkdir('./plots')
8 fig.write_image('./plots/train_numerical_corr.pdf')
```

得到的可视化结果如下



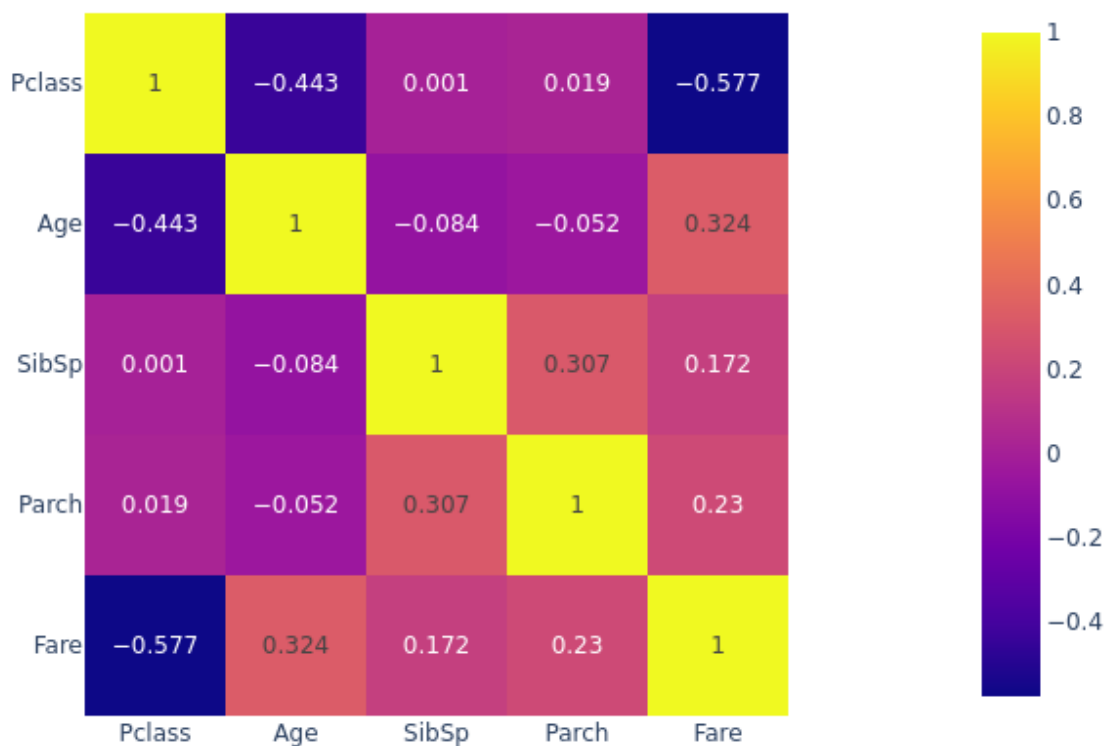
然后对数值型变量使用z-normalization操作:  $z = \frac{x - \bar{x}}{\sigma_x}$

```
1 from sklearn.preprocessing import StandardScaler
2
3 z_scaler = StandardScaler()
4 numerical_train = df_train[[f for f in
   df_train_features.corr().columns]].values
5 numerical_train = z_scaler.fit_transform(numerical_train)
```

测试集中, 也首先可视化数值型特征的关联矩阵

```
1 fig = px.imshow(df_test_features.corr().round(3), text_auto=True)
2 fig.show()
3 if not os.path.exists('./plots'):
4     os.mkdir('./plots')
5 fig.write_image('./plots/test_numerical_corr.png')
```

得到的可视化结果如下



然后对数值型变量使用z-normalization操作:  $z = \frac{x - \bar{x}}{\sigma_x}$

```
1 z_scaler = StandardScaler()
2 numerical_test = df_test[[f for f in
    df_test_features.corr().columns]].values
3 numerical_test = z_scaler.fit_transform(numerical_test)
```

随后随着Tree Number上升进入饱和状态, 在Tree Number>60后甚至Accuracy略微下降, 可以认为是模型过大出现了过拟合现象.



## 3.2 分类特征编码——使用 OneHotEncoder

对于Cabin，我们只关心船舱的首字母

- 训练集中

```
1 from sklearn.preprocessing import OneHotEncoder
2
3 encoder = OneHotEncoder()
4 for i, v in df_train_features['Cabin'].items():
5     if isinstance(v, str):
6         df_train_features['Cabin'][i] = v[0]
7     else:
8         df_train_features['Cabin'][i] = str(v)
9 categorical_train = df_train_features[['Sex', 'Cabin',
10 'Embarked']].values
11 categorical_train_encoding =
12     encoder.fit_transform(categorical_train).toarray()
```

- 测试集中

```
1 for i, v in df_test_features['Cabin'].items():
2     if isinstance(v, str):
3         df_test_features['Cabin'][i] = v[0]
4     else:
5         df_test_features['Cabin'][i] = str(v)
6 categorical_test = df_test_features[['Sex', 'Cabin',
7 'Embarked']].values
8 categorical_test_encoding =
9     encoder.transform(categorical_test).toarray()
```

## 4. 构造和连接训练变量

```
1 import numpy as np
2
3 X_train = np.concatenate([numerical_train,
4 categorical_train_encoding], axis=1)
5
6 X_test = np.concatenate([numerical_test,
7 categorical_test_encoding], axis=1)
```

其中训练集分类结果`y_train`已在前面求出

## 5. 模型训练和预测输出

我们为三种算法的实现提供统一的函数负责训练和预测,如下

```
1 model = {}
2 y_predict = {}
3
4 def train_and_predict(model_name, classifier):
5     model[model_name] = classifier
6     model[model_name].fit(X_train, y_train)
7     y_predict[model_name] = model[model_name].predict(X_test)
8     df_test['Survived'] = y_predict[model_name]
9     output = df_test[['PassengerId', 'Survived']]
10    if not os.path.exists('./submission'):
11        os.mkdir('./submission')
12    output.to_csv(f'./submission/{model_name}.csv', index=False,
header=True)
```

## 5.1 Logistic Regression 模型

```
1 from sklearn.linear_model import LogisticRegression
2
3 train_and_predict(model_name='logistic',
  classifier=LogisticRegression())
```




## 5.2 Random Forest 模型

```
1 from sklearn.ensemble import RandomForestClassifier
2
3 train_and_predict(model_name='randomForest',
  classifier=RandomForestClassifier())
```

## 5.3 Xgboost 模型

```
1 import xgboost as xgb
2
3 train_and_predict(model_name='xgboost',
  classifier=xgb.XGBClassifier())
```

我们将三种模型的结果提交到kaggle进行检验, 得到测试集上准确率结果如下

Submissions	
All	Successful
Errors	Recent
Submission and Description	
Public Score	
 <b>xgboost.csv</b> Complete · 2h ago · xgboost	<b>0.73684</b>
 <b>randomForest.csv</b> Complete · 2h ago · randomForest	<b>0.77751</b>
 <b>logistic.csv</b> Complete · 2h ago · logistic	<b>0.76794</b>

## 结果分析

我们可以看到随机森林(Random Forest)的准确率比Logistic Regression模型准确率略高, 这也说明随机森林模型通过多个随机决策树投票的方式减小了模型variance, 即减小了过拟合度, 从而在测试集上达到了更好的效果.

而Xgboost模型比前两个模型在测试集上的效果都要差,可能是因为训练集的数据量还不够大,对于Xgboost算法来说是欠拟合状态,从而难以于前两个算法学习的结果相比较.

## 代码实现

---

代码具体实现见附件 `final.ipynb` 文件中.

## 提交结果

---

见附件 `submission.zip` 文件