

人工智能之机器学习案例实践与思考

泰坦尼克号乘客生死预测

人工智能（AI）和机器学习（ML）现在是两个非常热门的流行语，当大数据、数据分析，以及更广泛的技术变革浪潮席卷全球时，这两个术语会频繁出现。

人工智能是一种机器能够以人们认为“聪明”的方式执行任务的更广泛的概念，而机器学习是解决人工智能问题中的一个重要方法，它基于这样一个想法：让机器访问数据，并让他们自适应学习，不断优化，达到最终解决问题的目的。

机器学习算法是把人类决策思考的过程抽象成一个模型，用数学方法给这个模型找到最优化的解，用代码把这个解变成机器可以执行的命令，最终完成一个机器大脑的构建。算法就是人对于一个事情的特定理解被转化成机器可以执行的模型和代码，依靠海量数据逐步优化，从而接近理想状态。

算法处理数据，数据反过来帮助优化算法。就像谷歌的算法和数据的交互通过人的搜索点击完成，即时反馈的记录增加了训练数据，每次新增的测试数据都是最新的培训数据。

完整的机器学习实现过程：

1. 提出问题

一切机器学习都是围绕着解决实际工作生活中的问题，如谷歌搜索推荐，豆瓣电影推荐等。明确的问题为后续的行动提供明确的方向。

2. 理解数据

根据问题采集数据，再将原数据导入 python。

3. 数据清洗

数据预处理，如处理缺失数据、转化数据类型等，然后提取特征工程。特征工程是本文的重点。

4. 构建模型

用训练数据带入合适的机器学习算法中，构建模型。

5. 模型评估

用测试数据评估模型，优化模型。

6. 方案实施

预测结果，并将分析结果形成报告。

工具：

语言工具：python,

数据处理模块：pandas, numpy, random

可视化处理模块：matplotlib, seaborn

模型算法：sklearn, xgboost

数据来源：kaggle 官方数据集（泰坦尼克号乘客情况的数据）

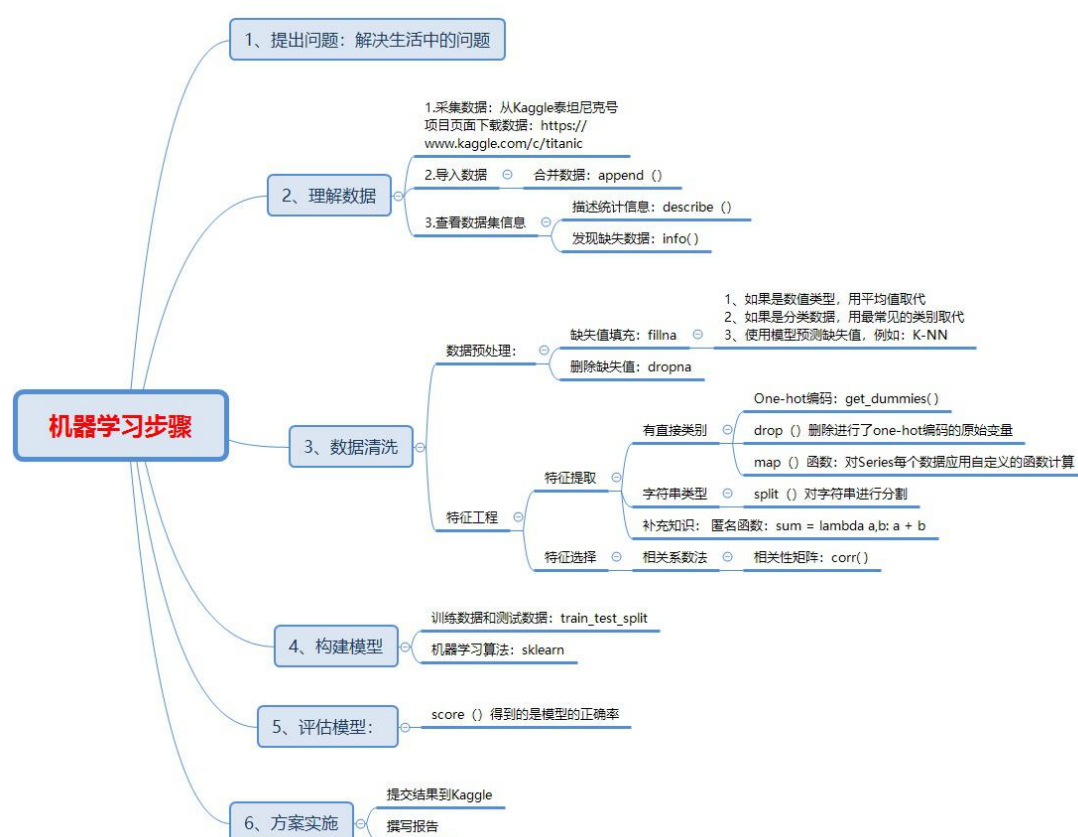
案例背景：

1912 年 4 月 14 日晚，英国皇家邮轮泰坦尼克号（Titanic）从英国南安普顿出发，途径法

国，爱尔兰在美国大西洋碰触冰山沉没，船上载人总数 2224 人，遇难人数 1517 人，震惊全世界。电影《泰坦尼克号》根据当时的史实改编，成为爱情片和灾难片的经典。大面积伤亡的原因之一是船上没有足够的救生艇供乘客和船员使用。尽管在沉船事故中幸存下来的人有一些运气成分，但有些人会比其他人更容易存活。

Titanic 案例是 Kaggle 竞赛里的经典比赛之一，要求参赛者根据乘客的属性来预测是否幸存，是典型的二分类（Binary Classifier）问题。解决二分类问题的算法有很多：决策树、随机森林、GBM，而 XGBoost 是 GBM 的优化实现。因此本文以 Titanic 幸存者预测竞赛为例，重点介绍机器学习算法实现过程以及机器学习算法大杀器 XGBoost 的使用及调优方法。

针对 Titanic 案例，实现过程如下图：



下面做详细介绍：

一、提出问题：

本文主要是对泰坦尼克号沉船事件进行预测：船上总共有 1309 名乘客，由于船上的救生艇很少，只有一部分乘客可以生还。需要通过一些数据特征，如性别，年龄，社会阶层，家庭大小等来判断乘客是否可以生还。

二、理解数据：

首先从 Kaggle 泰坦尼克号项目页面下载数据，此数据集主要包含两部分，一部分是 **Train** 训练集，另一部分是 **Test** 测试集，**Test** 测试集中没有 **Survived** 列（是否生存），其余部分和 **Train** 一致，需要我们根据机器学习算法训练模型，最终应用到 **Test** 测试集中预测乘客生还结果。

首先读取训练数据集和测试数据集，并把两个数据集合并在一个数据集 **full**，方便进行数据清洗。

```
#导入数据
import numpy as np
import pandas as pd

#训练数据集
train = pd.read_csv("train.csv")
#测试数据集
test = pd.read_csv("test.csv")

print('训练数据集:',train.shape,'测试数据集:',test.shape)
```

使用 **head** 查看训练集数据格式，如下图：

	Age	Cabin	Embarked	Fare	Name	Parch	PassengerId	Pclass	Sex	SibSp	Survived	Ticket
0	22.0	NaN	S	7.2500	Braund, Mr. Owen Harris	0	1	3	male	1	0.0	A/5 21171
1	38.0	C85	C	71.2833	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	2	1	female	1	1.0	PC 17599
2	26.0	NaN	S	7.9250	Heikkinen, Miss. Laina	0	3	3	female	0	1.0	STON/O2. 3101282
3	35.0	C123	S	53.1000	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	4	1	female	1	1.0	113803
4	35.0	NaN	S	8.0500	Allen, Mr. William Henry	0	5	3	male	0	0.0	373450

数据特征含义为：

特征名	含义
Age	乘客年龄
Cabin	客舱号
Embarked	登船港口
Fare	船票价格
Name	乘客姓名
Parch	船上父母数/子女数（不同代直系亲属数）
PassengerId	乘客编号
Pclass	客舱等级（1=一等舱，2=二等舱，3=三等舱）
Sex	性别
SibSp	船上兄弟姐妹数/配偶数
Survived	生存情况（1=存活，0=死亡）
Ticket	船票编号

数据类型一般划分为数值类型和分类数据两类，其中分类数据也分为有类别和字符串两

种，下面是本文中出现的变量的数据类型划分结果：

数据类型	列名 (变量)	含义
标签	Survived	生存情况 (1=存活, 0=死亡)
数值类型	PassengerId	乘客编号
	Age	年龄
	Fare	船票价格
	SibSp	船上兄弟姐妹数/配偶数 (同代直系亲属数)
	Parch	船上父母数/子女数 (不同代直系亲属数)
分类数据 (有类别)	Sex	性别 (男, 女)
	Embarked	登船港口 出发地点: S=英国南安普顿 Southampton 途径地点1: C=法国 瑟堡市 Cherbourg 出发地点2: Q=爱尔兰 昆士 敦Queenstown
	Pclass	客舱等级 (1=1等舱, 2=2等舱, 3=3等舱)
	Name	姓名
分类数据 (字符串)	Cabin	客舱号
	Ticket	船票编号

使用 describe 查看数据集描述统计信息，对整体数据有一个了解及分析：

	Age	Fare	Parch	PassengerId	Pclass	SibSp	Survived
count	1046.000000	1308.000000	1309.000000	1309.000000	1309.000000	1309.000000	891.000000
mean	29.881138	33.295479	0.385027	655.000000	2.294882	0.498854	0.383838
std	14.413493	51.758668	0.865560	378.020061	0.837836	1.041658	0.486592
min	0.170000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000
25%	21.000000	7.895800	0.000000	328.000000	2.000000	0.000000	0.000000
50%	28.000000	14.454200	0.000000	655.000000	3.000000	0.000000	0.000000
75%	39.000000	31.275000	0.000000	982.000000	3.000000	1.000000	1.000000
max	80.000000	512.329200	9.000000	1309.000000	3.000000	8.000000	1.000000

使用 info 查看每一列的数据及数据类型，确定这些数据是否存在缺失值：

```
# 查看每一列的数据类型，和数据总数
full.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 12 columns):
Age          1046 non-null float64
Cabin        295 non-null object
Embarked     1307 non-null object
Fare         1308 non-null float64
Name         1309 non-null object
Parch        1309 non-null int64
PassengerId  1309 non-null int64
Pclass       1309 non-null int64
Sex          1309 non-null object
SibSp        1309 non-null int64
Survived     891 non-null float64
Ticket       1309 non-null object
dtypes: float64(3), int64(4), object(5)
memory usage: 122.8+ KB
```

通过上面的结果发现数据总共有 1390 行，其中：

数值类型列：年龄（Age）、船舱号（Cabin）里面有较多缺失数据：

① 年龄（Age）里面数据总数是 1046 条，缺失了 $1309-1046=263$ ，缺失率 $263/1309=20\%$ ；

② 船票价格（Fare）里面数据总数是 1308 条，缺失了 1 条数据；

字符串列：

③ 登船港口（Embarked）里面数据总数是 1307，只缺失了 2 条数据，缺失比较少；

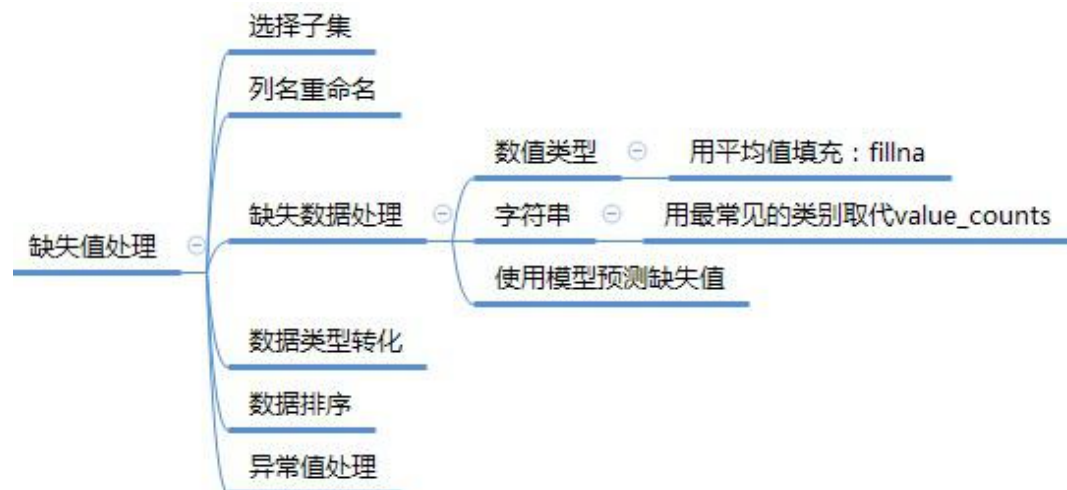
④ 客舱号（Cabin）里面数据总数是 295，缺失了 $1309-295=1014$ ，缺失率 $=1014/1309=77.5\%$ ，缺失比较大。

⑤ 生存情况（Survived）里面只有 891 条数据，这是因为只有训练数据集中才有 Survived，通过训练数据集中 Survived 作为标签构建机器学习的模型，最终预测出测试数据集中的 Survived 情况，这也是本次项目的目的，所以不需要处理这一列的数据。

以上的结果为我们下一步数据清洗指明了方向，只有知道哪些数据缺失数据，我们才能有针对性的处理。

三、数据清洗：

（1）处理缺失值



1.数值型缺失值处理

年龄（Age）和船票价格（Fare）都存在缺失值，且数据类型都为浮点型，可以使用平均值或中位数填充的方法填充缺失值，其中 `fillna` 表示数据填充的方法，`mean` 表示使用平均值填充。

2.字符串类型缺失值处理

客舱号（Cabin）的数据类型为字符串，通过查看客舱号（Cabin）这一列的数据发现缺失数值较多，缺失率达到 77.5%，且无规律可循，这里可以直接填充为 U，表示未知（Unknown）。

3.有类别数据缺失值处理

登船港口（Embarked）属于有类别数据，共分为三类，从结果中可以看出这一列只有两个缺失值，先统计在出发的地点中出现最多的地点是哪个，然后使用最频繁出现的地点填充。

```

#数据类型为 float 的用平均值填充。
full['Age'] = full['Age'].fillna(full['Age'].mean())
full['Fare'] = full['Fare'].fillna(full['Fare'].mean())
#只有 2 条缺失数据的 Embarked 列用出现最多的登船港口填充。
Embarked_Rank =
full.groupby('Embarked').count().sort_values(by='Name',
ascending=False).index.values.tolist()
full['Embarked'] = full['Embarked'].fillna(Embarked_Rank[0])
#Cabin 列缺失数据太多，填充为 U，表示 Unknown
full['Cabin'] = full['Cabin'].fillna('U')
  
```

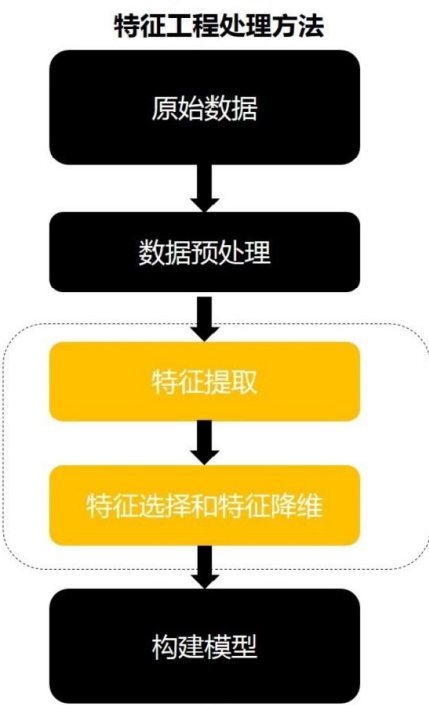
（2）特征提取

什么是特征工程？

特征工程就是最大限度地从原始数据中提取能表征原始数据的特征，以供机器学习算法和

模型使用。若训练的特征相关性高，则能事半功倍，故而**特征工程是整个项目的核心所在**。

特征工程处理方法：

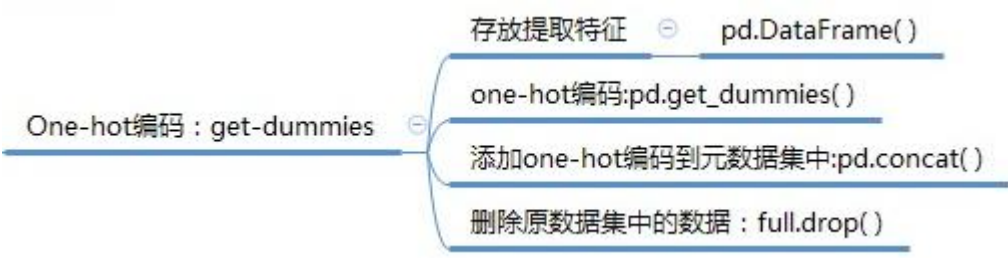


数据可基本分为**分类数据**、**数值数据**和**时间序列**，如下图所示：



分类数据分为两种情况：

- 1、变量只有两种类别的可以 `map()` 函数做值映射为数值，如性别；
- 2、变量有两以上类别的，进行 `one-hot` 编码，用 `get_dummies()` 函数，产生虚拟变量。



1.性别

取值只有“男”“女”两种，通过 `map()` 函数将性别的值映射为数值“0”和“1”。

2.登船港口(Embarked)

通过 `get_dummies()` 进行 one-hot 编码。

3.客舱等级 (Pclass)

通过 `get_dummies()` 进行 one-hot 编码。

4、船舱号 Cabin

客舱号的类别值是首字母，例如：C85 类别映射为首字母 C。

通过 `get_dummies()` 进行 one-hot 编码。

5、Parch, SibSp 数据

SibSp: 表示船上兄弟姐妹数和配偶数量，理解为同代直系亲属数量，

Parch: 表示船上父母数和子女数，理解为不同代直系亲属数量。

家庭人数 (familySize) = 同代直系亲属数 (Parch) + 不同代直系亲属数 (SibSp) + 乘客自己
(因为乘客自己也是家庭成员的一个，所以这里加 1)，则有：

`familydf['FamilySize'] = full['Parch'] + full['SibSp'] + 1`

由于 FamilySize 的类别比较多，要进行降维，因此根据 familySize 的大小将家庭数量分为大、中、小三种，如下：

家庭类别：

小家庭 Family_Single: 家庭人数=1

中等家庭 Family_Small: 2<=家庭人数<=4

大家庭 Family_Large: 家庭人数>=5

	FamilySize	Family_Single	Family_Small	Family_Large
0	2	0	1	0
1	2	0	1	0
2	1	1	0	0
3	2	0	1	0
4	1	1	0	0

6.字符串类型特征提取

仔细观察发现，每一个 Name 中都含有称谓，如 Mr, Mrs, Miss 等，可用 `split` 分割字符串提取出称谓。经查西方人对头衔的分类，可将头衔分为以下 6 类：

Officer	政府官员
Royalty	王室 (皇室)
Mr	已婚男士
Mrs	已婚妇女
Miss	年轻未婚女子
Master	有技能的人/教师

这样将大大的简化乘客身份的分类，并用 map 函数完成转换，用 get_dummies 方法实现 one-hot 编码。

```
#特征提取
#将性别映射为数值类型
sex_mapDict = {'male':1, 'female':0}
full['Sex'] = full['Sex'].map(sex_mapDict)
full.head()

#将登船港口、客舱等级进行 One-hot 编码
#存放提取后的登船港口特征
embarkedDf = pd.DataFrame()
#使用 get_dummies 进行 one-hot 编码，产生虚拟变量（dummy variables），列名前缀是 Embarked
embarkedDf = pd.get_dummies( full['Embarked'], prefix='Embarked' )
embarkedDf.head()

#存放提取后的客舱等级特征
pclassDf = pd.DataFrame()
#使用 get_dummies 进行 one-hot 编码，列名前缀是 Pclass
pclassDf = pd.get_dummies(full['Pclass'], prefix='Pclass')
pclassDf.head()

#姓名列包括三部分：名、头衔、姓，头衔是表明身份地位的特征，可能对生存结果有影响，需要提取出来
#存放提取后的头衔特征
titleDf = pd.DataFrame()
import re

titleDf['Title'] = full['Name'].apply(lambda x: re.search(r'(?=<=, )[\w]+', x).group())
titleDf.head()

'''
定义以下几种头衔类别：
Officer 政府官员
```

Royalty 王室（皇室）

Mr 已婚男士

Mrs 已婚妇女

Miss 年轻未婚女子

Master 有技能的人/教师

...

#姓名中头衔字符串与定义头衔类别的映射关系

```
title_mapDict = {  
    "Capt":      "Officer",  
    "Col":        "Officer",  
    "Major":      "Officer",  
    "Jonkheer":   "Royalty",  
    "Don":        "Royalty",  
    "Sir" :       "Royalty",  
    "Dr":         "Officer",  
    "Rev":        "Officer",  
    "the Countess": "Royalty",  
    "Dona":       "Royalty",  
    "Mme":        "Mrs",  
    "Mlle":       "Miss",  
    "Ms":         "Mrs",  
    "Mr" :        "Mr",  
    "Mrs" :       "Mrs",  
    "Miss" :      "Miss",  
    "Master" :    "Master",  
    "Lady" :      "Royalty"  
}
```

```
titleDf['Title'] = titleDf['Title'].map(title_mapDict)
```

#使用 get_dummies 进行 one-hot 编码

```
titleDf = pd.get_dummies(titleDf['Title'])
```

```
titleDf.head()
```

#提取客舱号首字母

#存放客舱号信息 b

```
cabinDf = pd.DataFrame()
```

```
full[ 'Cabin' ] = full[ 'Cabin' ].map(lambda x: x[0])
```

#使用 get_dummies 进行 one-hot 编码，列名前缀是 Cabin

```
cabinDf = pd.get_dummies( full['Cabin'], prefix = 'Cabin' )
```

```
cabinDf.head()
```

#建立家庭人数和家庭类别

#家庭人数 = 同代直系亲属数（Parch）+ 不同代直系亲属数（SibSp）+ 乘客自己

#存放客舱号信息

```
familyDf = pd.DataFrame()
```

```

familyDf[ 'FamilySize' ] = full[ 'Parch' ] + full[ 'SibSp' ] + 1
'''
家庭类别：
小家庭 Family_Single: 家庭人数=1
中等家庭 Family_Small: 2<=家庭人数<=4
大家庭 Family_Large: 家庭人数>=5
'''

familyDf['Family_Single'] = familyDf['FamilySize'].map( lambda x: 1 if x == 1 else 0 )
familyDf['Family_Small']   = familyDf['FamilySize'].map( lambda x: 1 if 2 <= x <= 4 else 0 )
familyDf['Family_Large']   = familyDf['FamilySize'].map( lambda x: 1 if 5 <= x else 0 )
familyDf.head()

#合并数据集
full = pd.concat([full, embarkedDf, pclassDf, titleDf, cabinDf, familyDf], axis=1)
full.drop(['Embarked', 'Pclass', 'Name', 'Cabin'], axis=1, inplace=True)
print(full.shape)
full.head()

```

补充知识：匿名函数

python 使用 lambda 来创建匿名函数。

所谓匿名，意即不再使用 def 语句这样标准的形式定义一个函数，预防如下：

lambda 参数 1，参数 2：函数体或者表达式

定义匿名函数：对两个数相加

sum = lambda a,b: a + b

至此完成了所有数据特征的提取。

（3）特征选择

选取原则：根据所有变量的相关系数矩阵，筛选出与预测标签 **Survived** 最相关的特征变量。

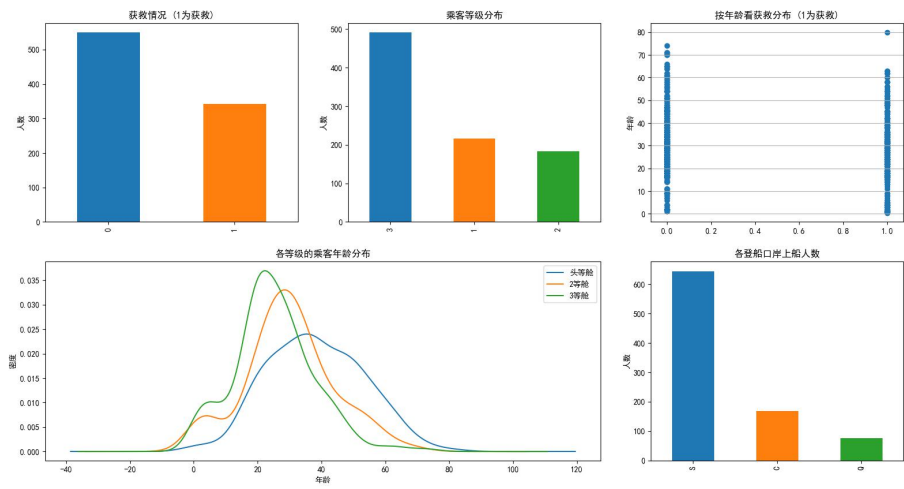
	Age	Fare	Parch	PassengerId	Sex	SibSp	Survived	Embarked_C	Embarked_Q	Embarked_S	...	Cabin_D	Cabin_E
Age	1.000000	0.171521	-0.130872	0.025731	0.057397	-0.190747	-0.070323	0.076179	-0.012718	-0.059153	...	0.132886	0.106600
Fare	0.171521	1.000000	0.221522	0.031416	-0.185484	0.160224	0.257307	0.286241	-0.130054	-0.169894	...	0.072737	0.073949
Parch	-0.130872	0.221522	1.000000	0.008942	-0.213125	0.373587	0.081629	-0.008635	-0.100943	0.071881	...	-0.027385	0.001084
PassengerId	0.025731	0.031416	0.008942	1.000000	0.013406	-0.055224	-0.005007	0.048101	0.011585	-0.049836	...	0.000549	-0.008136
Sex	0.057397	-0.185484	-0.213125	0.013406	1.000000	-0.109609	-0.543351	-0.066564	-0.088651	0.115193	...	-0.057396	-0.040340
SibSp	-0.190747	0.160224	0.373587	-0.055224	-0.109609	1.000000	-0.035322	-0.048396	-0.048678	0.073709	...	-0.015727	-0.027180
Survived	-0.070323	0.257307	0.081629	-0.005007	-0.543351	-0.035322	1.000000	0.168240	0.003650	-0.149683	...	0.150716	0.145321
Embarked_C	0.076179	0.286241	-0.008635	0.048101	-0.066564	-0.048396	0.168240	1.000000	-0.164166	-0.778262	...	0.107782	0.027566
Embarked_Q	-0.012718	-0.130054	-0.100943	0.011585	-0.088651	-0.048678	0.003650	-0.164166	1.000000	-0.491656	...	-0.061459	-0.042877
Embarked_S	-0.059153	-0.169894	0.071881	-0.049836	0.115193	0.073709	-0.149683	-0.778262	-0.491656	1.000000	...	-0.056023	0.002960
Pclass_1	0.362587	0.599956	-0.013033	0.026495	-0.107371	-0.034256	0.285904	0.325722	-0.166101	-0.181800	...	0.275698	0.242963
Pclass_2	-0.014193	-0.121372	-0.010057	0.022714	-0.028862	-0.052419	0.093349	-0.134675	-0.121973	0.196532	...	-0.037929	-0.050210
Pclass_3	-0.302093	-0.419616	0.019521	-0.041544	0.116562	0.072610	-0.322308	-0.171430	0.243706	-0.003805	...	-0.207455	-0.169063
Master	-0.363923	0.011596	0.253482	0.002254	0.164375	0.329171	0.085221	-0.014172	-0.009091	0.018297	...	-0.042192	0.001860
Miss	-0.254146	0.092051	0.066473	-0.050027	-0.672819	0.077564	0.332795	-0.014351	0.198804	-0.113886	...	-0.012516	0.008700
Mr	0.165476	-0.192192	-0.304780	0.014116	0.870678	-0.243104	-0.549199	-0.065538	-0.080224	0.108924	...	-0.030261	-0.032953
Mrs	0.198091	0.139235	0.213491	0.033299	-0.571176	0.061643	0.344935	0.098379	-0.100374	-0.022950	...	0.080393	0.045538
Officer	0.162818	0.028696	-0.032631	0.002231	0.087288	-0.013813	-0.031316	0.003678	-0.003212	-0.001202	...	0.006055	-0.024048
Royalty	0.059466	0.026214	-0.030197	0.004400	-0.020408	-0.010787	0.033391	0.077213	-0.021853	-0.054250	...	-0.012950	-0.012202
Cabin_A	0.125177	0.020094	-0.030707	-0.002831	0.047561	-0.039808	0.022287	0.094914	-0.042105	-0.056984	...	-0.024952	-0.023510
Cabin_B	0.113458	0.393743	0.073051	0.015895	-0.094453	-0.011569	0.175095	0.161595	-0.073613	-0.095790	...	-0.043624	-0.041103
Cabin_C	0.167993	0.401370	0.009601	0.006092	-0.077473	0.048616	0.114652	0.158043	-0.059151	-0.101861	...	-0.053083	-0.050016
Cabin_D	0.132886	0.072737	-0.027385	0.000549	-0.057396	-0.015727	0.150716	0.107782	-0.061459	-0.056023	...	1.000000	-0.034317

单独选择 **Survived** 列，按列降序排列，就能看到哪些特征最正相关，哪些特征最负相关，将其筛选出来作为模型的特征输入。

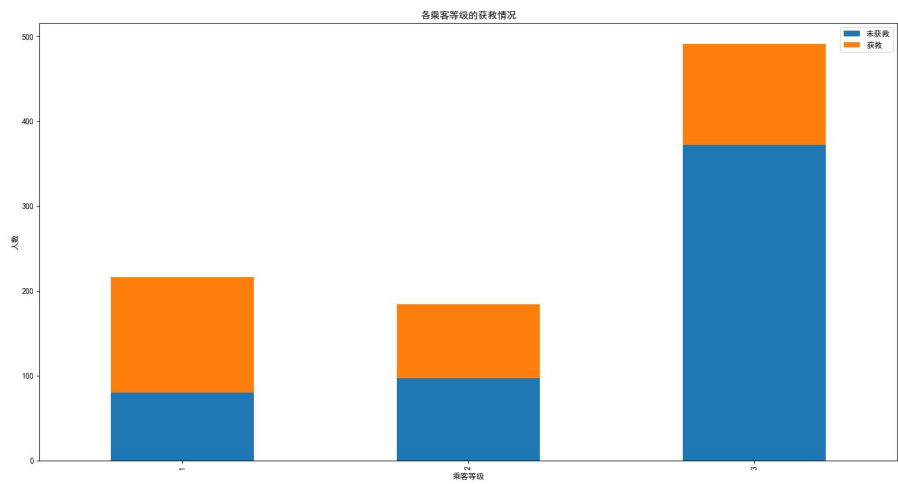
```
'''
查看各个特征与生成情况（Survived）的相关系数，
ascending=False表示按降序排列
'''
corrDf['Survived'].sort_values(ascending=False)
```

```
Survived      1.000000
Mrs           0.344935
Miss          0.332795
Pclass_1      0.285904
Family_Small  0.279855
Fare          0.257307
Cabin_B       0.175095
Embarked_C    0.168240
Cabin_D       0.150716
Cabin_E       0.145321
Cabin_C       0.114652
Pclass_2      0.093349
Master        0.085221
Parch         0.081629
Cabin_F       0.057935
Royalty       0.033391
Cabin_A       0.022287
FamilySize    0.016639
Cabin_G       0.016040
Embarked_Q    0.003650
PassengerId   -0.005007
Cabin_T       -0.026456
Officer       -0.031316
SibSp         -0.035322
Age           -0.070323
Family_Large  -0.125147
Embarked_S    -0.149683
Family_Single -0.203367
Cabin_U       -0.316912
Pclass_3      -0.322308
```

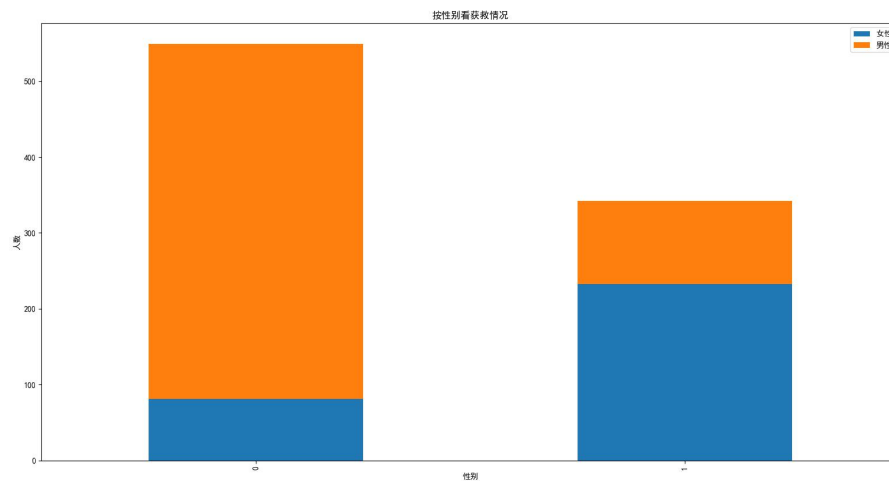
下面对数据集数据分布进行分析，如下图：



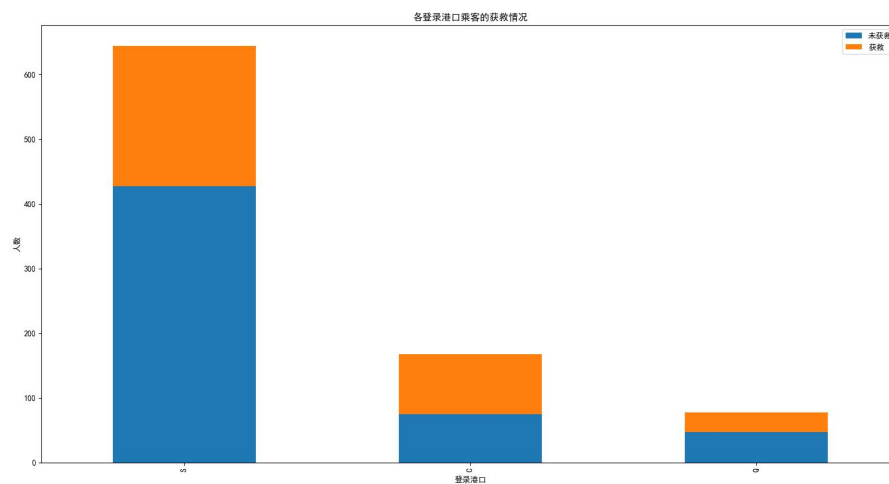
现在针对各特征，逐一分析此特征与生存结果之间的关联关系：



由上图可知，等级高的乘客有更高的获救概率，所以乘客等级是影响最终结果的重要特征。

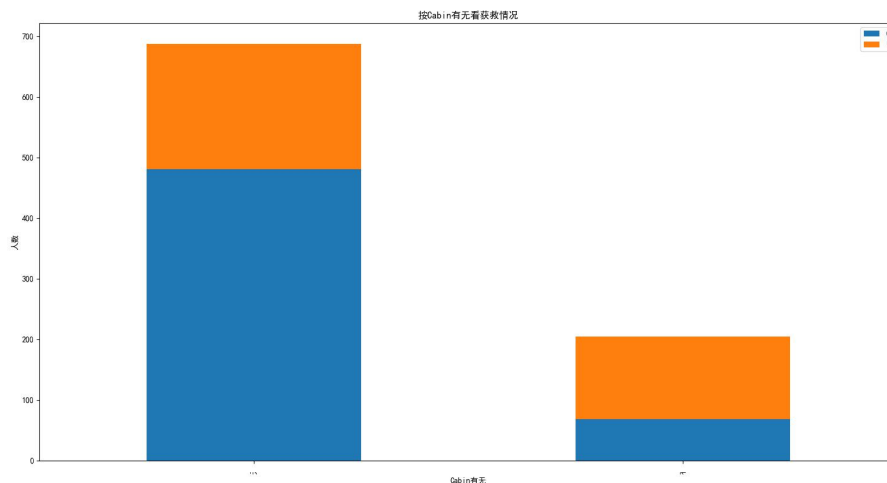


由上图可知，女性乘客有更高的获救概率，所以性别是影响最终结果的重要特征。



由上图可知，在不同港口登船的乘客有不同的获救概率，所以登船港口也是影响最终结果的重要特征。





由上图可知，乘客有无船舱号也会影响获救概率，所以船舱号也是影响最终结果的重要特征。

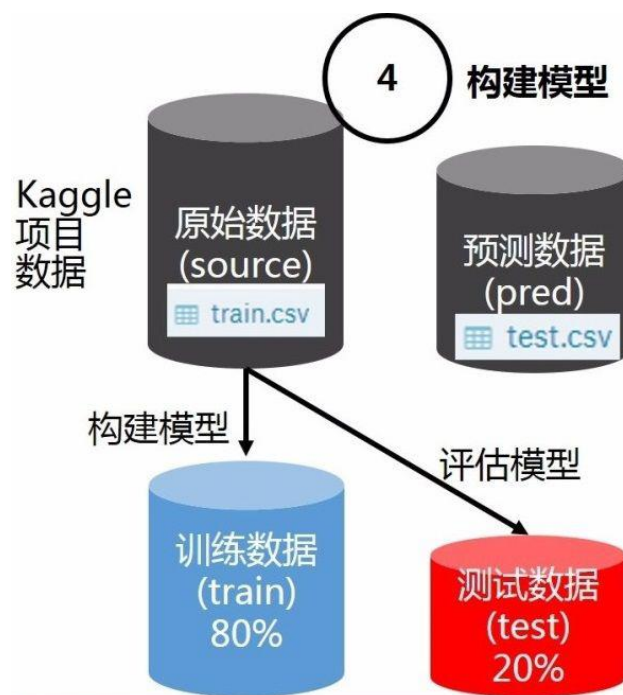
类似分析，头衔、家庭大小、船票价格也都是重要的特征。

综上所述，根据各个特征与生成情况（Survived）的关联关系，我们选择这几个特征作为模型的输入：

头衔（前面所在的数据集 titleDf）、客舱等级（pclassDf）、家庭大小（familyDf）、船票价格（Fare）、船舱号（cabinDf）、登船港口（embarkedDf）、性别（Sex）。并把这些特征合并成新的数据集。

四、构建模型：

上一步数据清洗中我们对缺失值进行了处理，并且进行特种工程的特征提取和选择，接下来就可以构建模型了。



我们将测试数据集称为**预测数据集（pred）**，用于生成预测结果提供给 kaggle。

我们将训练数据集称为**原始数据集（source）**

从原始数据中拆分为训练数据（train）和测试数据（test），训练数据用于构建模型，测试数据用于评估模型，得到的预测结果要提供给 kaggle。

序号	名称	变量名	行数	说明
1	整体数据集	full_df	1309	train.csv和test.csv的合并集
2	原始数据集	source_df	891	train.csv数据集
3	预测数据集	pred_df	481	test.csv数据集
4	整体集特征	full_x	1309	整体数据集的特征
5	原始集特征	source_x	891	原始数据集的特征
6	原始集标签	source_y	481	原始数据集的标签
7	训练集特征	train_x	712	原始数据集的特征的80%
8	训练集标签	train_y	712	原始数据集的特征的80%
9	测试集特征	test_x	179	原始数据集的特征的20%
10	测试集标签	test_y	179	原始数据集的特征的20%
11	预测集特征	pred_x	481	预测数据集的特征

五、训练模型

预测泰坦尼克号上的人们生存或死亡是二分类问题，能进行分类的算法很多，需要选择一个合适的算法进行训练模型。

```
model.fit(train_X, train_y)
```

六、评估模型

模型评估重要是对训练模型进行测试，检测其测试准确率。

调用 `score` 函数，获得 `score` 评分来最终测试模型的准确度。

```
model.score(test_X, test_y)
```

七、方案实施

主要是对测试数据集的数据进行预测，并将预测的结果上传到 kaggle 网站，通过网站最终排名，来检验预测的结果。

```
# 使用机器学习模型，对预测数据集中的生存情况进行预测
predictions = model.predict(test_data[features].values)
# 保存结果
result = pd.DataFrame({'PassengerId':test_data['PassengerId'].values,
                       'Survived':predictions.astype(np.int32)})
result.to_csv("Prediction.csv", index=False)
```

下面使用机器学习主流的六种算法分别做模型的训练和预测，并将结果直接上传 kaggle 验证。(下述训练数据集为全量 train 数据集，不做拆分)

逻辑回归算法:

```
# 第 1 步: 导入算法
from sklearn.linear_model import LogisticRegression
# 第 2 步: 创建模型: 逻辑回归 (logistic regression)
model = LogisticRegression()
# 第 3 步: 训练模型
model.fit(train_data[features].values, train_data["Survived"].values)
# 使用机器学习模型, 对预测数据集中的生存情况进行预测
predictions = model.predict(test_data[features].values)
# 保存结果
result =
pd.DataFrame({'PassengerId':test_data['PassengerId'].values,
'Survived':predictions.astype(np.int32)})
result.to_csv("LogisticRegressionPrediction.csv", index=False)
```

训练后的模型为:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
verbose=0, warm_start=False)
```

上传 Kaggle 预测准确率为:

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
LogisticRegressionPrediction.csv	just now	0 seconds	0 seconds	0.75598
Complete				

随机森林 Random Forests Model 算法:

```
# 第 1 步: 导入算法
from sklearn.ensemble import RandomForestClassifier
# 第 2 步: 创建模型: 随机森林 Random Forests Model
model = RandomForestClassifier(n_estimators=100)
# 第 3 步: 训练模型
model.fit(train_data[features].values, train_data["Survived"].values)
# 使用机器学习模型, 对预测数据集中的生存情况进行预测
predictions = model.predict(test_data[features].values)
# 保存结果
result =
```

```
pd.DataFrame({'PassengerId':test_data['PassengerId'].values,
'Survived':predictions.astype(np.int32)})
result.to_csv("RandomForestClassifierPrediction.csv", index=False)
```

训练后的模型为:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
max_depth=None, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
oob_score=False, random_state=None, verbose=0,
warm_start=False)
```

上传 Kaggle 预测准确率为:

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
RandomForestClassifierPrediction.csv	just now	0 seconds	0 seconds	0.77511
Complete				

支持向量机 Support Vector Machines 算法:

第 1 步: 导入算法

```
from sklearn.svm import SVC, LinearSVC
```

第 2 步: 创建模型: 支持向量机 Support Vector Machines

```
model = SVC()
```

第 3 步: 训练模型

```
model.fit(train_data[features].values, train_data["Survived"].values)
```

使用机器学习模型, 对预测数据集中的生存情况进行预测

```
predictions = model.predict(test_data[features].values)
```

保存结果

```
result =
```

```
pd.DataFrame({'PassengerId':test_data['PassengerId'].values,
```

```
'Survived':predictions.astype(np.int32)})
```

```
result.to_csv("SVCPrediction.csv", index=False)
```

训练后的模型为:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
max_iter=-1, probability=False, random_state=None, shrinking=True,
tol=0.001, verbose=False)
```

上传 Kaggle 预测准确率为:

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
SVCPrediction.csv	just now	0 seconds	0 seconds	0.72727
Complete				

Gradient Boosting Classifier 算法:

```
# 第 1 步: 导入算法
from sklearn.ensemble import GradientBoostingClassifier
# 第 2 步: 创建模型: Gradient Boosting Classifier
model = GradientBoostingClassifier()
# 第 3 步: 训练模型
model.fit(train_data[features].values, train_data["Survived"].values)
# 使用机器学习模型, 对预测数据集中的生存情况进行预测
predictions = model.predict(test_data[features].values)
# 保存结果
result =
pd.DataFrame({'PassengerId':test_data['PassengerId'].values,
'Survived':predictions.astype(np.int32)})
result.to_csv("GradientBoostingClassifierPrediction.csv", index=False)
```

训练后的模型为:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
learning_rate=0.1, loss='deviance', max_depth=3,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100,
presort='auto', random_state=None, subsample=1.0, verbose=0,
warm_start=False)
```

上传 Kaggle 预测准确率为:

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
GradientBoostingClassifierPrediction....	just now	0 seconds	0 seconds	0.79425
Complete				

K-nearest neighbors 算法:

```
# 第 1 步: 导入算法
from sklearn.neighbors import KNeighborsClassifier
# 第 2 步: 创建模型: K-nearest neighbors
```

```

model = KNeighborsClassifier(n_neighbors = 3)
# 第 3 步：训练模型
model.fit(train_data[features].values, train_data["Survived"].values)
# 使用机器学习模型，对预测数据集中的生存情况进行预测
predictions = model.predict(test_data[features].values)
# 保存结果
result = pd.DataFrame({'PassengerId':test_data['PassengerId'].values,
'Survived':predictions.astype(np.int32)})
result.to_csv("KNeighborsClassifierPrediction.csv", index=False)

```

训练后的模型为：

```

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=3, p=2,
weights='uniform')

```

上传 Kaggle 预测准确率为：

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
KNeighborsClassifierPrediction.csv	just now	0 seconds	0 seconds	0.70334
Complete				

朴素贝叶斯 Gaussian Naive Bayes 算法：

```

# 第 1 步：导入算法
from sklearn.naive_bayes import GaussianNB
# 第 2 步：创建模型：朴素贝叶斯 Gaussian Naive Bayes
model = GaussianNB()
# 第 3 步：训练模型
model.fit(train_data[features].values, train_data["Survived"].values)
# 使用机器学习模型，对预测数据集中的生存情况进行预测
predictions = model.predict(test_data[features].values)
# 保存结果
result =
pd.DataFrame({'PassengerId':test_data['PassengerId'].values,
'Survived':predictions.astype(np.int32)})
result.to_csv("GaussianNBPrediction.csv", index=False)

```

训练后的模型为：

```
GaussianNB(priors=None)
```

上传 Kaggle 预测准确率为：

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
GaussianNBPrediction.csv	just now	0 seconds	0 seconds	0.76076
Complete				

对比各算法预测准确率：

Submission and Description	Public Score	Use for Final Score
GaussianNBPrediction.csv a few seconds ago by Yang Jun add submission details	0.76076	<input type="checkbox"/>
KNeighborsClassifierPrediction.csv a minute ago by Yang Jun add submission details	0.70334	<input type="checkbox"/>
GradientBoostingClassifierPrediction.csv 2 minutes ago by Yang Jun add submission details	0.79425	<input type="checkbox"/>
SVCPrediction.csv 3 minutes ago by Yang Jun add submission details	0.72727	<input type="checkbox"/>
RandomForestClassifierPrediction.csv 4 minutes ago by Yang Jun add submission details	0.77511	<input type="checkbox"/>
LogisticRegressionPrediction.csv 21 minutes ago by Yang Jun add submission details	0.75598	<input type="checkbox"/>

可见，针对此数据集，GradientBoostingClassifier 算法准确率最高（79.43%），而 KNeighborsClassifier 准确率最低（70.33%）。

使用上述算法，在 kaggle 中预测准确率排名为（8321~4053）/11407，准确率并不理想，有待提升。

优化思路：

使用 Bagging 和 Boosting，这两个都是模型融合的方法，可以将弱分类器融合之后形成一个强分类器，而且融合之后的效果会比最好的弱分类器更好。

优化方案之一：

使用 BaggingRegressor。

Bagging 即套袋法，从原始数据中随机抽样得到多个同样大小的数据集，来训练多个基学习器，各学习器之间互不依赖。是一种并行的方法。各分类器的权重都是相等的。

其算法过程如下：

从原始样本集中抽取训练集。每轮从原始样本集中使用 Bootstrapping 的方法抽取 n 个训练

样本（在训练集中，有些样本可能被多次抽取到，而有些样本可能一次都没有被抽中）。共进行 k 轮抽取，得到 k 个训练集。（ k 个训练集之间是相互独立的）
每次使用一个训练集得到一个模型， k 个训练集共得到 k 个模型。（注：这里并没有具体的分类算法或回归方法，我们可以根据具体问题采用不同的分类或回归方法，如决策树、感知器等）
对分类问题：将上步得到的 k 个模型采用投票的方式得到分类结果；对回归问题，计算上述模型的均值作为最后的结果。（所有模型的重要性相同）

具体实现如下：

```
# 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)
bagging_clf.fit(train_data[features].values, train_data["Survived"].values)
predictions = bagging_clf.predict(test_data[features].values)
result =
pd.DataFrame({'PassengerId':test['PassengerId'].values,
              'Survived':predictions.astype(np.int32)})
result.to_csv("logistic_regression_bagging_predictions.csv", index=False)
```

训练后的模型为：

```
BaggingRegressor(base_estimator=LogisticRegression(C=1.0, class_weight=None,
dual=False, fit_intercept=True, intercept_scaling=1,
max_iter=100, multi_class='ovr', n_jobs=1,
penalty='l1', random_state=None, solver='liblinear',
tol=1e-06, verbose=0, warm_start=False), bootstrap=True,
bootstrap_features=False, max_features=1.0, max_samples=0.8,
n_estimators=20, n_jobs=1, oob_score=False, random_state=None,
verbose=0, warm_start=False)
```

上传 Kaggle 预测准确率为：

logistic_regression_bagging_predictions.csv	0.77990	<input type="checkbox"/>
5 days ago by Yang Jun		
add submission details		

将上述

```
clf = linear_model.LogisticRegression(C=1.0, penalty='l1', tol=1e-6)
修改为
clf = GradientBoostingClassifier()
```

上传 Kaggle 预测准确率为：

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
GradientBoostingClassifierBaggingPr...	just now	0 seconds	0 seconds	0.77033
Complete				

上述两种方法预测准确率相对有所提升，但仍不理想，参数需要继续优化。

优化思路之二：

使用分类器 XGBoost。

XGBoost 背景介绍：

自从 2014 年 9 月份在 Kaggle 的希格斯玻色子机器学习大赛中夺魁以来，XGBoost 与深度学习两个算法垄断了 Kaggle 大赛的大部分冠军。

现在 Kaggle 大赛的情况基本是这样的，凡是非结构化数据相关，比如语音、图像，基本都是深度学习获胜，凡是结构化数据上的竞赛，基本都是 XGBoost 获胜。要知道大部分的业务数据，都是以良好格式存储在关系数据库中的结构化数据，这也就是说，跟行业应用、业务优化这些真金白银息息相关的场景里，XGBoost 是目前最好用的大杀器之一。

XGBoost 的安装

XGBoost 的安装颇费周折，普通的第三方包使用 pip 安装都能成功，但 XGBoost 直接安装失败，所以上网查询安装方法，首先尝试网上流行的安装方法，即 Git+MinGW+python install 的方法，但经过各种尝试以失败告终；最后，查阅英文网站，使用了 Visual Studio 2013 编译+CMAKE+Git+python install 的方法终于安装成功。XGBoost 也成了本人所安装过的所有软件中难度之最了，不过终究还是安装成功了，很有成就感。

boosting 原理介绍：

用所有的数据去训练基学习器，个体学习器之间存在依赖关系，每一个学习器都是基于之前训练的学习器的结果，集中关注被错分的数据，来获得新的学习器，达到提升的效果。

（通俗来说，就是每次都只学习一点，然后一步步的接近最终要预测的值。）分类的结果是基于所有分类器的加权求和结果的，分类器的权重并不相等，每个权重代表的是其对应分类器在上一轮迭代中的成功度。

具体实现如下：

```
xgb_clf =
XGBClassifier(learning_rate=0.1,
max_depth=2, silent=True, objective='binary:logistic')
xgb_clf.fit(train_X.values, train_y.values)
predictions = xgb_clf.predict(pred_X.values)
result =
pd.DataFrame({'PassengerId':test['PassengerId'].values,
'Survived':predictions.astype(np.int32)})
```

```
result.to_csv("XGBClassifier_predictions.csv", index=False)
```

首次配置的 **XGBClassifier** 往往不是最优参数，需要调参优化。

调参过程类如：

#设置 boosting 迭代计算次数

```
param_test = {
    'n_estimators': range(30, 50, 2),
    'max_depth': range(2, 7, 1)
}
grid_search =
GridSearchCV(estimator = clf, param_grid = param_test, scoring='accuracy', cv=5)
grid_search.fit(train_data[features], train_data["Survived"])
print grid_search.grid_scores_, grid_search.best_params_,
grid_search.best_score_
```

输出：

```
[
mean: 0.81594, std: 0.00673, params: {'n_estimators': 30, 'max_depth': 2},
mean: 0.82267, std: 0.00978, params: {'n_estimators': 32, 'max_depth': 2},
.....
mean: 0.82828, std: 0.03210, params: {'n_estimators': 30, 'max_depth': 5},
mean: 0.82716, std: 0.03130, params: {'n_estimators': 32, 'max_depth': 5},
mean: 0.82941, std: 0.03304, params: {'n_estimators': 34, 'max_depth': 5},
.....
mean: 0.82716, std: 0.03524, params: {'n_estimators': 48, 'max_depth': 6}]
{'n_estimators': 32, 'max_depth': 6}
0.83164983165
```

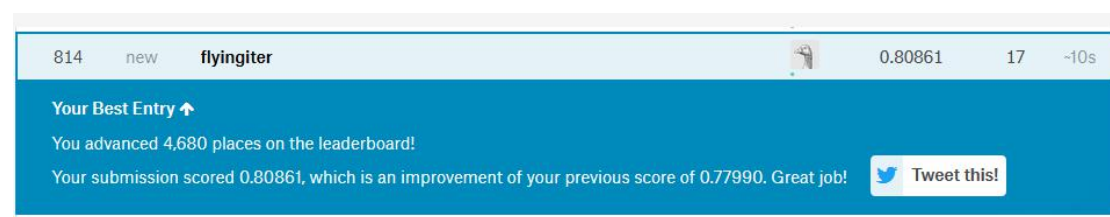
则可得参数优化取值：'n_estimators': 32, 'max_depth': 6

模型更新为：

```
clf =XGBClassifier(learning_rate=0.1, max_depth=6, n_estimators=32, silent=True,
objective='binary:logistic')
```

重复上述过程，最终确定优化后的模型，用于预测。

使用上述模型，预测结果上传 **kaggle**，排名为 **814/11407**，准确率为 **80.87%**。




814 new flyingiter 0.80861 17 -10s

Your Best Entry ↑

You advanced 4,680 places on the leaderboard!

Your submission scored 0.80861, which is an improvement of your previous score of 0.77990. Great job!

 Tweet this!

优化方案之三：

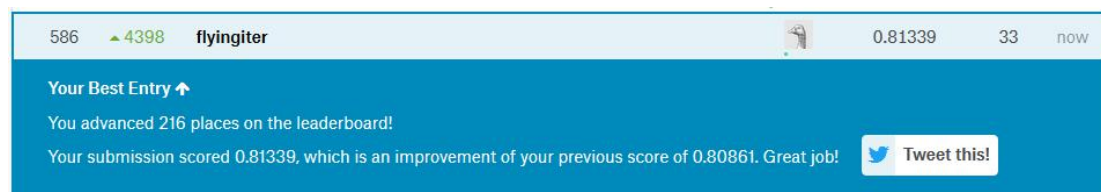
考虑到数据集中年龄特征缺失值较多，而此前解决方案都是使用中位数替换缺失值，没有考虑到年龄与其他特征值存在的潜在联系，因此可以就此着手继续优化。

对于年龄缺失值使用多重缺失补充方案，具体实现方案如下：

```
# 把已有的数值型特征取出来丢进 Random Forest Regressor 中
age_df = titanic[['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()].values
unknown_age = age_df[age_df.Age.isnull()].values
# print known_age
# print unknown_age
# 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
titanic.loc[(titanic.Age.isnull()), 'Age'] = predictedAges
```

其中，使用随机森林模型，建模分析'Age'与'Fare', 'Parch', 'SibSp', 'Pclass'之间的关系，并进行预测填充。

进行此项优化后，沿用上述方案二中的 XGBClassifier 进行预测，结果上传 kaggle，排名 586/11407（5%），准确率提升至 81.34%。



因训练数据集较少，准确率能达 80%以上，已说明模型基本达到要求了。为避免过拟合而非实质性的改进，此案例的尝试就此告一段落。

总结与思考

1. 本案例按照机器学习的步骤进行预测尝试，取得了理想的效果。现实应用中，训练数据集和预测数据集往往都是动态的、时常更新的，而机器学习便是基于动态更新的数据对模型不断进行优化、完善，随着数据量不断增大，模型也就越来越接近理想的状态。

态，进而满足实践应用的需要。

2. 机器学习算法中，特征工程是核心，数据是基石，完备的数据集和完善的特征工程将决定机器学习预测准确率的上限，而模型算法的选择和优化则是不断接近这个上限，由此可见数据采集和清洗的重要性。
3. 机器学习作为人工智能领域的核心算法，被更广泛应用的一个新时代即将到来，机器学习与人工智能也注定成为计算机发展的必然趋势。

问题	描述	应用案例
分类	基于输入确定每个输入所属的分类	垃圾邮件过滤、情感分析、欺诈检测、客户广告定位、流失预测、支持案例标记、内容个性化、制造缺陷检测、客户细分、事件发现、基因学、药效学
回归	基于输入预测每个输入的实际输出	股票市场预测、需求预测、价格估计、广告竞价优化、风险管理、资产管理、天气预报、优生预测
推荐	预测用户喜欢的方案	产品推荐、工作招聘、Netflix 奖金、在线约会、内容推荐
插补	对于缺失的数据推断其价值	不完整的医疗记录、客户数据缺失、人口数据普查