

《模式识别与统计学习》

**案例教学**

**泰坦尼克号生还与否预测**

|  |  |
| --- | --- |
| 组 别： | 第四组 |
| 组 长： | 牟容塬 |
| 组 员： | 吕 阳 |
|  | 宋之琦 |
| 指导老师： | 赵海涛 |

信息科学与工程学院

2022年5月

目 录

[1](#_Toc100928148)

[《模式识别与统计学习》 1](#_Toc100928149)

[泰坦尼克号生还与否预测 1](#_Toc100928150)

[1 背景 1](#_Toc100928151)

[2 泰坦尼克号数据集 1](#_Toc100928152)

[2.1 数据集变量 1](#_Toc100928153)

[2.2 特征分析 3](#_Toc100928154)

[2.2.1 性别 4](#_Toc100928155)

[2.2.2 乘客等级 4](#_Toc100928156)

[2.2.3 年龄 5](#_Toc100928157)

[2.2.4 同船的兄弟姊妹及配偶个数 5](#_Toc100928158)

[2.2.5 同船的父母与子女个数 6](#_Toc100928159)

[2.2.6 家庭规模 7](#_Toc100928160)

[2.2.7 登船港口 8](#_Toc100928161)

[2.2.8 姓名 9](#_Toc100928162)

[2.2.9 船票编号 11](#_Toc100928163)

[2.2.10 船票价格 12](#_Toc100928164)

[3 数据处理 13](#_Toc100928165)

[3.1 缺值填充 14](#_Toc100928166)

[3.1.1 年龄 14](#_Toc100928167)

[3.1.2 船票价格 14](#_Toc100928168)

[3.1.3 登船港口 15](#_Toc100928169)

[3.1.4 船舱号 15](#_Toc100928170)

[3.2 数值化 16](#_Toc100928171)

[3.2.1 姓名 16](#_Toc100928172)

[3.2.2 性别 16](#_Toc100928173)

[3.2.3 船票编号 17](#_Toc100928174)

[3.2.4 船舱号码 17](#_Toc100928175)

[3.2.5 登船港口 18](#_Toc100928176)

[3.3 One-Hot编码 19](#_Toc100928177)

[4 特征提取 22](#_Toc100928178)

[4.1 所有（包括扩展）特征 23](#_Toc100928179)

[4.2 随机森林特征重要性 23](#_Toc100928180)

[4.3 One-Hot相关系数法 24](#_Toc100928181)

[4.4 调合 26](#_Toc100928182)

[5 模型训练 27](#_Toc100928183)

泰坦尼克号生还与否预测

# 1 背景

在白星航运公司的第二艘奥林匹克级远洋班轮 RMS Titanic的处女航中，共有2208人从英国南安普敦前往纽约市。在航行途中，这艘船在1912年4月15日凌晨撞上冰山并沉没，造成1503人死亡。

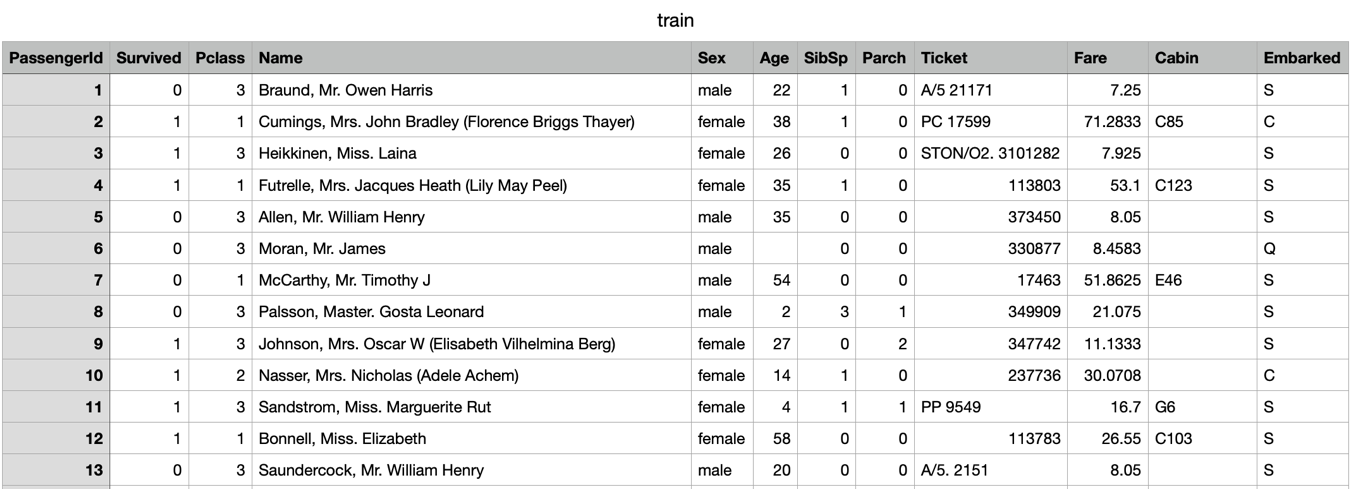
船上的乘客根据票价分为三个不同的等级：乘坐头等舱的乘客，其中大多数是船上最富有的乘客，包括上流社会的杰出成员、商人、政治家、高级军事人员、实业家、银行家、艺人、社会名流和职业运动员；二等舱旅客主要是中产阶级旅客，包括教授、作家、神职人员和游客；三等舱或统舱乘客主要是移居美国和加拿大的移民。

# 2 泰坦尼克号数据集

在kaggle网站上有一个名为“Titanic - Machine Learning from Disaster”的竞赛项目，在这个项目中可以下载到泰坦尼克号数据集，并且分为了训练集（train.csv）和测试集（test.csv）两组。

## 2.1 数据集变量

其中，训练集共891组，测试集共418组，每组数据含有12个特征变量：游客编号（PassengerId）、幸存与否（Survived）、乘客等级（Pclass）、姓名（Name）、性别（Sex）、年龄（Age）、同船的兄弟姊妹及配偶个数（SibSp）、同船的父母与子女个数（Parch）、船票编号（Ticket）、船票价格（Fare）、船舱号（Cabin）、登船港口（Embarked）。训练数据集部分数据如图2-1所示。

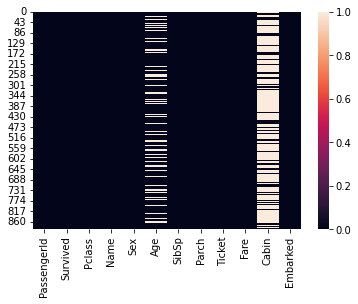
图2-1 训练数据集部分截图

使用程序对数据集完整性进行评估，可以发现在测试集中，年龄（Age）、船舱号（Cabin）、登船港口（Embarked）分别缺失177、687、2个特征值。

|  |
| --- |
| import pandas as pd train = pd.read\_csv(**"train.csv"**)  train.info() train.isnull().sum() |

当有数据缺失的记录在整个数据中只占一个很小比例时，可以直接删除缺失记录，对余下的完全数据进行处理。但是在实际数据中，往往缺失数据占有相当的比重，这样做不仅会产生偏差，甚至会得出有误导性的结论，同时丢失大量信息，造成浪费。

|  |
| --- |
| import seaborn as sns sns.heatmap(train.isnull()) |

图2-2 特征缺失值展现图

## 2.2 特征分析

首先，我们看一下训练集中的生还与否的概率，其中有38%的人生还，62%的人死去。

|  |
| --- |
| *# Overview* all = train[**'Survived'**].value\_counts() rate\_all = all / len(train) print(rate\_all) |

查尔斯·莱特勒是泰坦尼克号的二副，他是最后一个从冰冷的海水中被拖上救生船、职位最高的生还者。他写下了17页的回忆录，讲述了沉船灾难的细节。面对沉船灾难，船长命令先让妇女和儿童上救生艇，许多乘客显得十分平静，一些人则拒绝与家人分开。结合一些资料，对泰坦尼克号有了更深入的了解。接下来，对不同的特征进行进一步的分析。

### 2.2.1 性别

|  |
| --- |
| *# Gender* sns.barplot(x=**'Sex'**, y=**'Survived'**, data=train) |

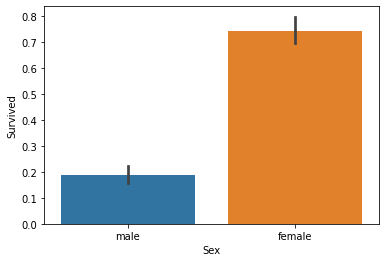


图2-3 性别特征存活概率

由于优先救援了妇女儿童，可以明显看出女性的存活率是高于男性的，所以性别（Sex）在预测中是一个很重要的特征。

### 2.2.2 乘客等级

|  |
| --- |
| *# Pclass* sns.barplot(x=**'Pclass'**, y=**'Survived'**, data=train) |

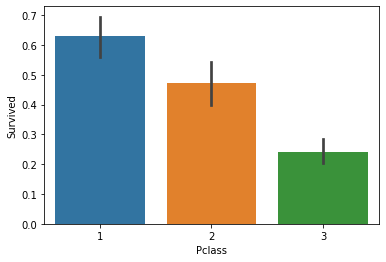


图2-4 乘客等级特征存活概率

乘客的等级可能与其社会地位、阶级势力相关，在图2-4可以看出，乘客等级越高，存活的概率也越高，在预测中乘客等级（Pclass）也是一个重要的特征。

### 2.2.3 年龄

|  |
| --- |
| *# Age* facet = sns.FacetGrid(train, hue=**'Survived'**, aspect=2) facet.map(sns.kdeplot, **'Age'**, shade=True) facet.set(xlim=(0, train[**'Age'**].max())) facet.add\_legend() plt.xlabel(**'Age'**) plt.ylabel(**'Probability'**) |

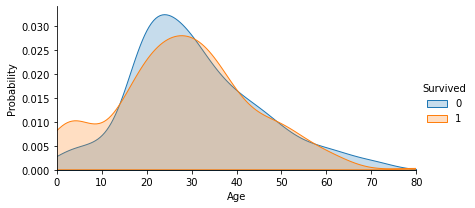


图2-5 年龄特征存活概率

从不同年龄生还与否的概率密度图可以看出，在年龄15岁的左侧、15岁至23岁之间，生还与否有明显差异，密度图非交叉区域面积非常大，但在其他年龄段，则差别不是很明显，可以近似看成一致，因此可以考虑将年龄偏小的区域作为重要特征提取。

### 2.2.4 同船的兄弟姊妹及配偶个数

|  |
| --- |
| *# Siblings and Spouse* sns.barplot(x=**'SibSp'**, y=**'Survived'**, data=train) |

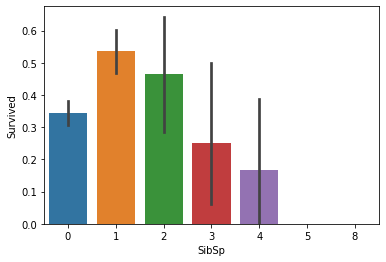


图2-6 同船的兄弟姊妹及配偶个数特征存活概率

同船的兄弟姊妹及配偶适中的乘客幸存率更高，可以看出，同船一位兄弟姊妹或配偶的存活率最高，可以猜测是带个配偶幸存率最高。

### 2.2.5 同船的父母与子女个数

|  |
| --- |
| *# Parents and Children* sns.barplot(x=**'Parch'**, y=**'Survived'**, data=train) |

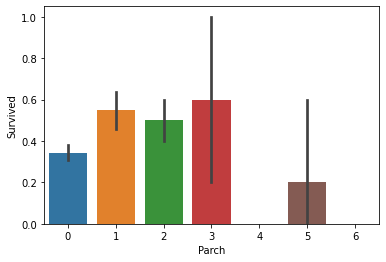


图2-7 同船的父母与子女个数特征存活概率

与上一特征相似，适中的存活率高，可以考虑将两个特征整合在一起，组合成为家庭成员人数特征，进行进一步的分析，考虑将总体作为重要特征来预测。

### 2.2.6 家庭规模

根据2.2.4和2.2.5两个特征的分析，因为分布比较接近，所以我们认为可以将两个特征进行合并，组成一个名为家庭规模的特征，同船的家庭人数等于同船的兄弟姊妹及配偶个数加上同船的父母与子女个数。

|  |
| --- |
| *#Family Scale* all\_data[**'FamilySize'**] = all\_data[**'Parch'**] + all\_data[**'SibSp'**] + 1 sns.barplot(x=**"FamilySize"**, y=**"Survived"**, data=all\_data) |

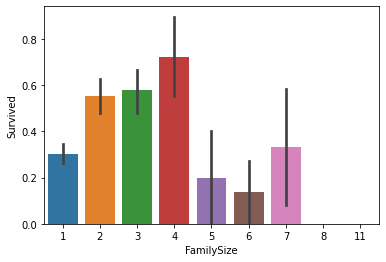


图2-8 同船的家庭人数特征存活概率

根据图表可以发现适中的存活概率较高，两边偏低，人数过多出现“团灭”现象。将家庭规模分成三类：中等家庭（2～4人）、单身（1人）和大家庭（5～7人）、超大家庭（大于7人）。

|  |
| --- |
| *# Divided into 3 type of Family* def FamilyScale(s):  if (s >= 2) & (s <= 4):  return 2  elif ((s > 4) & (s <= 7)) | (s == 1):  return 1  elif (s > 7):  return 0 all\_data[**'FamilyScale'**] = all\_data[**'FamilySize'**].apply(FamilyScale) sns.barplot(x=**"FamilyScale"**, y=**"Survived"**, data=all\_data) |

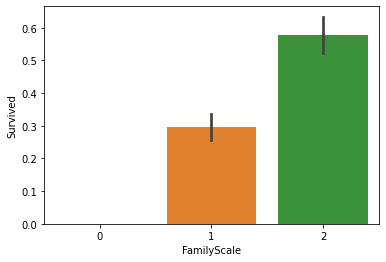


图2-9 家庭规模特征存活概率

最后，选用家庭规模这一特征来预测。

### 2.2.7 登船港口

出发地点S：英国南安普顿（Southampton），途径地点C ：法国瑟堡市（Cherbourg），途径地点Q：爱尔兰昆士敦（Queenstown），先按照登港港口与幸存率的关系画图，代码如下：

|  |
| --- |
| *# Embarked* sns.countplot(**'Embarked'**, hue=**'Survived'**, data=train) |

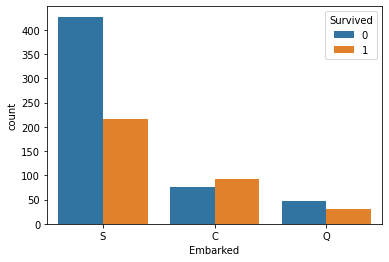


图2-10 登船港口特征存活概率

可以发现C地的生还率高，其余两地死亡率高，应该可以作为特征来预测。

### 2.2.8 姓名

使用名字来预测看上去仿佛是一件非常不科学的事情，因为每个人都有属于自己的名字，如何利用这个特征来将名字和幸存关联起来呢？经过分析，我们发现特征隐藏在称呼当中，比如上面我们提到过女生优先，所以称呼为 Miss 或 Mrs的就比称呼为 Mr 的的更可能幸存。所以利用姓名中包含的称谓、头衔等特征可以作为非常有用的一个新预测变量，可以帮助我们进行预测。于是我们从姓名中提取称谓、头衔并建立新的特征列Title。

|  |
| --- |
| *# Combine datasets* all\_data = pd.concat([train, test], ignore\_index=True) *# Get Title* all\_data[**'Title'**] = all\_data.Name.apply(lambda name: name.split(**','**)[1].split(**'.'**)[0].strip()) all\_data.Title.value\_counts() |

根据对上一代码段输出，可以将数据集中的姓名分成以下六类：

* 男性（Mr）：Mr；
* 女性（Miss）：Miss，Mlle；
* 夫人（Mrs）：Ms，Mrs，Mme；
* 男婴（Master）：Master；
* 官员（Officer）：Rev，Dr，Col，Major，Capt；
* 权贵（Royalty）：Lady，Sir，Don，the Countess，Jonkheer，Dona。

|  |
| --- |
| *# Create new column Title* all\_data[**'Title'**] = all\_data[**'Name'**].apply(lambda x:x.split(**','**)[1].split(**'.'**)[0].strip()) Title\_Dict = {} Title\_Dict.update(dict.fromkeys([**'Mr'**], **'Mr'**)) Title\_Dict.update(dict.fromkeys([**'Mlle'**, **'Miss'**], **'Miss'**)) Title\_Dict.update(dict.fromkeys([**'Mme'**, **'Ms'**, **'Mrs'**], **'Mrs'**)) Title\_Dict.update(dict.fromkeys([**'Master'**], **'Master'**)) Title\_Dict.update(dict.fromkeys([**'Capt'**, **'Col'**, **'Major'**, **'Dr'**, **'Rev'**], **'Officer'**)) Title\_Dict.update(dict.fromkeys([**'Don'**, **'Sir'**, **'the Countess'**, **'Dona'**, **'Lady'**,**'Jonkheer'**], **'Royalty'**)) all\_data[**'Title'**] = all\_data[**'Title'**].map(Title\_Dict) sns.barplot(x=**"Title"**, y=**"Survived"**, data=all\_data) |

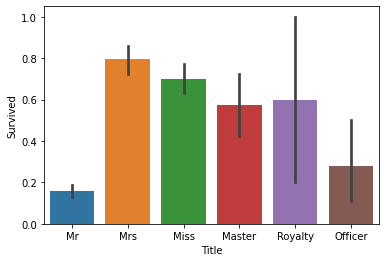


图2-11 头衔特征存活概率

### 2.2.9 船票编号

从数据集中可以发现，有些人的船票号码是一样的，所以可以统计船票编号相同的乘客个数。

|  |
| --- |
| Ticket\_Count = dict(all\_data[**'Ticket'**].value\_counts()) all\_data[**'TicketGroup'**] = all\_data[**'Ticket'**].apply(lambda x: Ticket\_Count[x]) sns.barplot(x=**'TicketGroup'**, y=**'Survived'**, data=all\_data) |

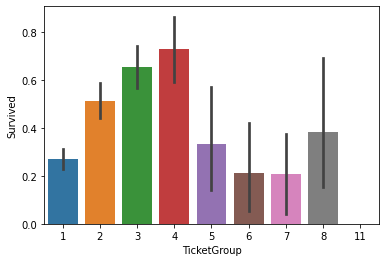


图2-12 船票编号相同乘客个数存活概率

据此，可以将乘客分成三类：适中（2～4人）、单票（1人）和多人（5～8人）、超多（大于8人）。

|  |
| --- |
| *# Divided into 3 groups* def Ticket\_Label(s):  if (s >= 2) & (s <= 4):  return 2  elif ((s > 4) & (s <= 8)) | (s == 1):  return 1  elif (s > 8):  return 0 all\_data[**'TicketGroup'**] = all\_data[**'TicketGroup'**].apply(Ticket\_Label) sns.barplot(x=**'TicketGroup'**, y=**'Survived'**, data=all\_data) |

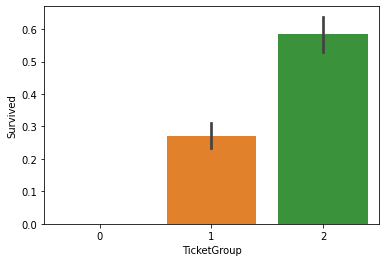


图2-13 船票编号相同规模存活概率

### 2.2.10 船票价格

资料显示，泰坦尼克号头等舱单独铺位的票价是30英镑（合今日的2515英镑），二等舱的票价是13英镑（相当于今天的1090英镑），三等舱的票价在7英镑到9英镑之间（相当于今天的587到754英镑），儿童票价是3英镑。然而在数据集中，同一座舱的票价差距很大。我们通过还价能力来看看存活的概率。

|  |
| --- |
| *# Fare # Get Medium Fare of Each Pclass* P1med = all\_data[**'Fare'**].where(all\_data[**'Pclass'**] == 1).median() P2med = all\_data[**'Fare'**].where(all\_data[**'Pclass'**] == 2).median() P3med = all\_data[**'Fare'**].where(all\_data[**'Pclass'**] == 3).median() *# Compute the ability of bargain of each class* all\_data[**'Bargain1'**] = (all\_data[**'Fare'**].where(all\_data[**'Pclass'**] == 1) - P1med) / all\_data[**'Fare'**].where(all\_data[**'Pclass'**] == 1) all\_data[**'Bargain2'**] = (all\_data[**'Fare'**].where(all\_data[**'Pclass'**] == 2) - P2med) / all\_data[**'Fare'**].where(all\_data[**'Pclass'**] == 2) all\_data[**'Bargain3'**] = (all\_data[**'Fare'**].where(all\_data[**'Pclass'**] == 3) - P3med) / all\_data[**'Fare'**].where(all\_data[**'Pclass'**] == 3) *# Put them together* all\_data[**'Bargain'**] = all\_data[**'Bargain1'**] all\_data[**'Bargain'**] = all\_data[**'Bargain'**].fillna(all\_data[**'Bargain2'**]) all\_data[**'Bargain'**] = all\_data[**'Bargain'**].fillna(all\_data[**'Bargain3'**]) all\_data[**'Bargain'**] *# Plot* facet = sns.FacetGrid(all\_data, hue=**'Survived'**, aspect=2) facet.map(sns.kdeplot, **'Bargain'**, shade=True) facet.set(xlim=(-2, 2)) facet.add\_legend() plt.xlabel(**'Bargain'**) plt.ylabel(**'Probability'**) |

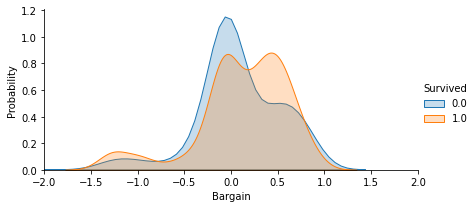


图2-14 还价能力存活概率

可以发现，还价的乘客死亡概率高，加价的乘客生还概率高，在两侧也存在一些反常现象。

# 3 数据处理

综上文所述，除去2个对预测无关的变量：游客编号（PassengerId）、幸存与否（Survived），1个缺失过多的变量：船舱号（Cabin），我们已经对其余9个重要特征变量进行了可视化、有机组合和深入分析，对数据的处理具有重要的意义。在建模之前，需要对数据集中的数据进行缺值填充、数值化操作。

## 3.1 缺值填充

因为船舱号缺失数据过多，本节不对船舱号（Cabin）进行分析填充，针对缺失177个数据的年龄（Age）、缺失1个数据的船票价格（Fare）和2个数据的登船港口（Embarked）进行缺值填充。

### 3.1.1 年龄

采用整体中位数或均值直接填充是最简单的方式，但是可能会造成误差。根据2.2.8节分析，在姓名中可以提取到一些称谓、头衔的特征，据此，可以更好地对不同人群的年龄进行填充。

|  |
| --- |
| grouped = all\_data.groupby([**'Title'**]) median = grouped.Age.median() print(median) |

可以发现不同称谓、头衔之间的年龄中位数差距较大，所以根据称谓、头衔来对年龄进行缺值填充。

|  |
| --- |
| *# Fill nan* for i in range(len(all\_data[**'Age'**])):  if pd.isnull(all\_data[**'Age'**][i]):  all\_data[**'Age'**][i] = median[all\_data[**'Title'**][i]] |

### 3.1.2 船票价格

由于只缺少1个数据，可知是一位三等舱的乘客的船票价格缺失，所以用三等舱票价的中位数进行填充。

|  |
| --- |
| *# Get P3 Medium Fare to Fill nan* P3med = all\_data[**'Fare'**].where(all\_data[**'Pclass'**] == 3).median() for i in range(len(all\_data[**'Fare'**])):  if pd.isnull(all\_data[**'Fare'**][i]):  all\_data[**'Fare'**][i] = P3med |

### 3.1.3 登船港口

由于只缺少2个数据，根据先验概率最大原则，S地的登船人数最多，所以我们将缺失值填充为最频繁出现的值S，即使用众数进行填充。

|  |
| --- |
| all\_data[**'Embarked'**] = all\_data[**'Embarked'**].fillna(**'S'**) |

### 3.1.4 船舱号

由于缺失数值过多，所以我们把缺失值填充定义为U，表示未知（Unknown）。

|  |
| --- |
| all\_data[**'Cabin'**] = all\_data[**'Cabin'**].fillna(**'U'**) |

经过填充，所有数据已经非空，但是发现数据中存在整形、浮点型、对象型三种数据类型。接下来就要对这些不同类型的数据进行数值化处理。

|  |
| --- |
| all\_data.info() |

## 3.2 数值化

需要将数据类型为对象型的数据进行数值化，涉及姓名（Name）、性别（Sex）、船票编号（Ticket）、船舱号（Cabin）、登船港口（Embarked）5个特征。

### 3.2.1 姓名

因为姓名已经使用中间提取到的称谓、头衔作为特征了，所以就对称谓、头衔进行数值化。共分6种称谓、头衔，使用整形0～5表示：

* 男性：使用0表示；
* 女性：使用1表示；
* 夫人：使用2表示；
* 男婴：使用3表示；
* 官员：使用4表示；
* 权贵：使用5表示。

|  |
| --- |
| *# 1. Title* all\_data.loc[all\_data[**'Title'**] == **'Mr'**, **'Title'**] = 0 all\_data.loc[all\_data[**'Title'**] == **'Miss'**, **'Title'**] = 1 all\_data.loc[all\_data[**'Title'**] == **'Mrs'**, **'Title'**] = 2 all\_data.loc[all\_data[**'Title'**] == **'Master'**, **'Title'**] = 3 all\_data.loc[all\_data[**'Title'**] == **'Officer'**, **'Title'**] = 4 all\_data.loc[all\_data[**'Title'**] == **'Royalty'**, **'Title'**] = 5 all\_data[**'Title'**] = all\_data[**'Title'**].astype(**'int64'**) |

### 3.2.2 性别

共分2种性别，使用整形0～1表示：男性使用0表示；女性使用1表示。

|  |
| --- |
| *# 2. Sex* all\_data.loc[all\_data[**'Sex'**] == **'male'**, **'Sex'**] = 0 all\_data.loc[all\_data[**'Sex'**] == **'female'**, **'Sex'**] = 1 all\_data[**'Sex'**] = all\_data[**'Sex'**].astype(**'int64'**) |

### 3.2.3 船票编号

根据2.2.9节分析，船票编号特征提取为共票人数，共分为3类，使用整形0～2表示：

* 超多（大于8人）：使用0表示；
* 单票（1人）和多人（5～8人）：使用1表示；
* 适中（2～4人）：使用2表示。

|  |
| --- |
| *# 3. Ticket* Ticket\_Count = dict(all\_data[**'Ticket'**].value\_counts()) all\_data[**'TicketGroup'**] = all\_data[**'Ticket'**].apply(lambda x: Ticket\_Count[x]) sns.barplot(x=**'TicketGroup'**, y=**'Survived'**, data=all\_data) def Ticket\_Label(s):  if (s >= 2) & (s <= 4):  return 2  elif ((s > 4) & (s <= 8)) | (s == 1):  return 1  elif (s > 8):  return 0 all\_data[**'TicketGroup'**] = all\_data[**'TicketGroup'**].apply(Ticket\_Label) all\_data[**'TicketGroup'**] = all\_data[**'TicketGroup'**].astype(**'int64'**) |

### 3.2.4 船舱号码

由于一个甲板设有多个船舱号码，所以提取第一个字母作为甲板号，其中U表示未知。共分9种船舱，A、B、C、D、E、F、G、T、U分别使用整形0～8表示。

|  |
| --- |
| *# 4. Cabin* all\_data[**'Deck'**] = all\_data[**'Cabin'**].str.get(0) all\_data.loc[all\_data[**'Deck'**] == **'A'**, **'Deck'**] = 0 all\_data.loc[all\_data[**'Deck'**] == **'B'**, **'Deck'**] = 1 all\_data.loc[all\_data[**'Deck'**] == **'C'**, **'Deck'**] = 2 all\_data.loc[all\_data[**'Deck'**] == **'D'**, **'Deck'**] = 3 all\_data.loc[all\_data[**'Deck'**] == **'E'**, **'Deck'**] = 4 all\_data.loc[all\_data[**'Deck'**] == **'F'**, **'Deck'**] = 5 all\_data.loc[all\_data[**'Deck'**] == **'G'**, **'Deck'**] = 6 all\_data.loc[all\_data[**'Deck'**] == **'T'**, **'Deck'**] = 7 all\_data.loc[all\_data[**'Deck'**] == **'U'**, **'Deck'**] = 8 all\_data[**'Deck'**] = all\_data[**'Deck'**].astype(**'int64'**) |

### 3.2.5 登船港口

共分3种登船港口，使用整形0～2表示：

* 英国南安普顿（S）：使用0表示；
* 法国瑟堡市（C）：使用1表示；
* 爱尔兰昆士敦（Q）：使用2表示；

|  |
| --- |
| *# 5. Embarked* all\_data.loc[all\_data[**'Embarked'**] == **'S'**, **'Embarked'**] = 0 all\_data.loc[all\_data[**'Embarked'**] == **'C'**, **'Embarked'**] = 1 all\_data.loc[all\_data[**'Embarked'**] == **'Q'**, **'Embarked'**] = 2 all\_data[**'Embarked'**] = all\_data[**'Embarked'**].astype(**'int64'**) |

经过对数据的处理，现在我们来检查一下各个变量的完整性和数据类型的统一性，并将训练集和测试集根据原始情况进行分割，除去无用的变量。

|  |
| --- |
| *# all\_data.info()。*  train, test = all\_data[:891], all\_data[891:] train\_data, train\_target = train.drop(**'Survived'**, axis=1), train[**'Survived'**] train\_data.pop(**'Name'**) train\_data.pop(**'Ticket'**) train\_data.pop(**'Cabin'**) train\_data.pop(**'PassengerId'**) *train\_data.info()* |

## 3.3 One-Hot编码

在统计机器学习算法中的回归，分类这些问题中，特征之间距离的计算或相似度计算非常重要。

本节我们通过One-Hot编码方式对数据的特征进行处理，我们可以将离散特征的取值扩展到欧式空间，在机器学习中，我们的研究范围就是在欧式空间中，对于One-Hot处理的离散的特征的某个取值也就对应了欧式空间的某个点！

|  |
| --- |
| *# One-hot Encode # 0.Pclass* pclassDf = pd.DataFrame() pclassDf = pd.get\_dummies(all\_data[**'Pclass'**], prefix=**'Pclass'**) all\_data = pd.concat([all\_data, pclassDf], axis=1) all\_data.pop(**'Pclass'**) *# 1.Title* titleDf = pd.DataFrame() titleDf = pd.get\_dummies(all\_data[**'Title'**], prefix=**'Title'**) all\_data = pd.concat([all\_data, titleDf], axis=1) all\_data.pop(**'Title'**) *# 2.Cabin* deckDf = pd.DataFrame() deckDf = pd.get\_dummies(all\_data[**'Deck'**], prefix=**'Deck'**) all\_data = pd.concat([all\_data, deckDf], axis=1) all\_data.pop(**'Deck'**) *# 3.Embarked* embarkedDf = pd.DataFrame() embarkedDf = pd.get\_dummies(all\_data[**'Embarked'**], prefix=**'Embarked'**) all\_data = pd.concat([all\_data, embarkedDf], axis=1) all\_data.pop(**'Embarked'**) *# 4.Family Scale* fsDf = pd.DataFrame() fsDf = pd.get\_dummies(all\_data[**'FamilyScale'**], prefix=**'FamilyScale'**) all\_data = pd.concat([all\_data, fsDf], axis=1) all\_data.pop(**'FamilyScale'**) *# 5.Ticket Group* tgDf = pd.DataFrame() tgDf = pd.get\_dummies(all\_data[**'TicketGroup'**], prefix=**'TicketGroup'**) all\_data = pd.concat([all\_data, tgDf], axis=1) all\_data.pop(**'TicketGroup'**) *# III. Initial # all\_data.info()* train, test = all\_data[:891], all\_data[891:] train\_data, train\_target = train.drop(**'Survived'**, axis=1), train[**'Survived'**] train\_data.pop(**'Name'**) train\_data.pop(**'Ticket'**) train\_data.pop(**'Cabin'**) train\_data.pop(**'PassengerId'**)  train\_data.info() |

# 4 特征提取

特征提取的方法主要是通过属性间的关系，如组合不同的属性得到新的属性，这样就改变了原来的特征空间。特征选择的方法是从原始特征数据集中选择出子集，是一种包含的关系，没有更改原始的特征空间。

用于评估特征提取的好坏，本节使用随机森林在验证集（测试集30%的数据）上的正确率进行评估。

|  |
| --- |
| def rf\_classifier(data, target, test, predictors):  *# predictors = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked', 'Title', 'TicketGroup', 'Deck', 'FamilySize', 'FamilyScale']* data, test = data[predictors], test[predictors]  data.info()  data\_train, data\_validation, target\_train, target\_validation = train\_test\_split(data, target, test\_size=0.3, random\_state=12345)  clf = RandomForestClassifier(random\_state=10, n\_estimators=26, max\_depth=6)  clf.fit(data\_train, target\_train)  test\_accuracy = clf.score(data\_validation, target\_validation) \* 100  print(**"Accuracy: %s%%."** % test\_accuracy) |

首先，我们使用所有原始的特征作为基准。

|  |
| --- |
| rf\_classifier(train\_data, train\_target, test, [**'Pclass'**, **'Sex'**, **'Age'**, **'SibSp'**, **'Parch'**, **'Fare'**, **'Embarked'**, **'Title'**, **'TicketGroup'**, **'Deck'**]) |

## 4.1 所有（包括扩展）特征

|  |
| --- |
| rf\_classifier(train\_data, train\_target, test, [**'Pclass'**, **'Sex'**, **'Age'**, **'SibSp'**, **'Parch'**, **'Fare'**, **'Embarked'**, **'Title'**, **'TicketGroup'**, **'Deck'**, **'FamilySize'**, **'FamilyScale'**]) |

可以发现，加上扩展特征后，验证集上的准确率降低了，说明其中含有一些无关干扰特征，接下来，采用多种方法对特征的进行评估。

## 4.2 随机森林特征重要性

|  |
| --- |
| *# 1. Random Forest Feature Importance* forest = RandomForestClassifier(random\_state=10, n\_estimators=26, max\_depth=6) forest.fit(train\_data, train\_target) importance = forest.feature\_importances\_ indices = np.argsort(importance)[::-1] feature\_name = train\_data.columns.values for i in range(len(indices)):  print(feature\_name[indices[i]], importance[indices[i]]) |

Title、Sex两个特征的重要性明显大于其他所有的特征，我们先选取这两个特征进行验证。

|  |
| --- |
| rf\_classifier(train\_data, train\_target, test, [**'Sex'**, **'Title'**]) |

准确性显著下降，我们增大阈值到0.07，添加更多的特征继续尝试。

|  |
| --- |
| rf\_classifier(train\_data, train\_target, test, [**'Pclass'**, **'Sex'**, **'Age'**, **'Fare'**, **'Title'**, **'Deck'**]) |

扩大阈值后，准确率超过了使用所有原始的特征近2个百分点。

## 4.3 One-Hot相关系数法

通过计算特征与特征之间的相关系数的大小，可判定两两特征之间的相关程度。相关系数取值区间在[-1, 1]之间，取值关系如下：

1. corr(x1,x2)相关系数值小于0表示负相关（这个变量下降，那个就会上升），即x1与x2是互补特征；

2. corr(x1,x2)相关系数值等于0表示无相关；

3. corr(x1,x2)相关系数值大于0表示正相关，即x1与x2是替代特征。

|  |
| --- |
| corrDf = all\_data.corr() res = corrDf[**'Survived'**].sort\_values(ascending=False) print(res) |

选择阈值为0.25，使用Sex、Title、TicketGroup、Pclass、Deck、FamilyScale、Fare七个特征。

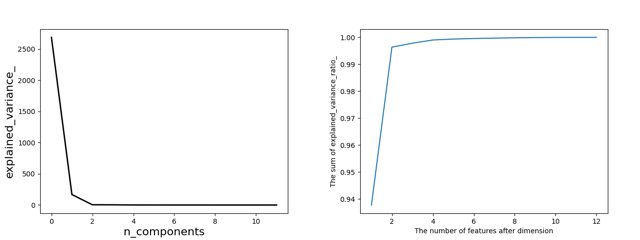
|  |
| --- |
| rf\_classifier(train\_data, train\_target, test, [**'Sex'**, **'Title'**, **'TicketGroup'**, **'FamilyScale'**, **'Fare'**, **'Pclass'**, **'Deck'**]) |

准确率超过了使用所有原始的特征近1.5个百分点，但是效果不如4.2节中的特征选择。

## 4.4 主成分分析

主成分分析经常用于减少数据集的维数，同时保持数据集的对方差贡献最大的特征。这是通过保留低阶主成分，忽略高阶主成分做到的，这样低阶成分往往能够保留住数据的最重要方面。

|  |
| --- |
| *# 3. PCA* vara\_all = [**'Pclass'**, **'Sex'**, **'Age'**, **'SibSp'**, **'Parch'**, **'Fare'**, **'Embarked'**, **'Title'**, **'TicketGroup'**, **'Deck'**, **'FamilySize'**, **'FamilyScale'**] raw\_all = all\_data[vara\_all] pca = decomposition.PCA() pca.fit(raw\_all) pca\_info = pca.explained\_variance\_ratio\_ print(pca\_info) pca\_info\_sum = np.cumsum(pca\_info) print(pca\_info\_sum) plt.figure() plt.plot([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], pca\_info\_sum) plt.xlabel(**'The number of features after dimension'**) plt.ylabel(**'The sum of explained\_variance\_ratio\_'**) plt.show() |

图4-1 主成分分析

根据输出和图像，结合前面小结实验结果，可以适当控制维度，但也不宜取的太小，目测7个左右比较合适。

## 4.5 调合

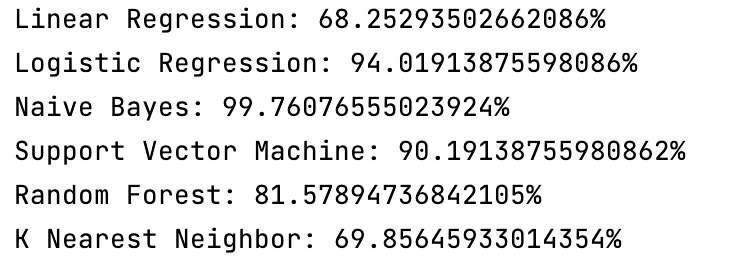
根据随机森林特征重要性和One-Hot相关系数法得到的结果与设置阈值选取出的特征，以及在随机森林中验证集验证的准确率，最终选择Pclass、Sex、Age、Fare、Title、Deck、FamilyScale共七个特征。

|  |
| --- |
| rf\_classifier(train\_data, train\_target, test, [**'Pclass'**, **'Sex'**, **'Age'**, **'Fare'**, **'Title'**, **'Deck'**, **'FamilyScale'**]) |

# 5 模型训练

最后，将数值化后的、挑选出来的特征分别使用线型回归模型、逻辑蒂斯回归模型、朴素贝叶斯方法、支持向量机、随机森林、K近邻6种方法对训练集进行训练，并在测试集上进行测试，使用准确率来衡量分类的好坏。

|  |
| --- |
| *# V. Train* target\_test = target\_test[**'Survived'**] print(target\_test) predictor = [**'Pclass'**, **'Sex'**, **'Age'**, **'Fare'**, **'Title'**, **'Deck'**, **'FamilyScale'**] data\_train, data\_test = train\_data[predictor], test[predictor] data\_train.info() data\_test.info() *# 1. Linear Regression* clf = LinearRegression() clf.fit(data\_train, train\_target) acc\_clf = clf.score(data\_test, target\_test) \* 100 print(**"Linear Regression: %s%%"** % acc\_clf) *# 2. Logistic Regression* lr = LogisticRegression() lr.fit(data\_train, train\_target) acc\_lr = lr.score(data\_test, target\_test) \* 100 print(**"Logistic Regression: %s%%"** % acc\_lr) *# 3. Naive Bayes* nb = BernoulliNB() nb.fit(data\_train, train\_target) acc\_nb = nb.score(data\_test, target\_test) \* 100 print(**"Naive Bayes: %s%%"** % acc\_nb) *# 4. Support Vector Machine* svm = LinearSVC() svm.fit(data\_train, train\_target) acc\_svm = svm.score(data\_test, target\_test) \* 100 print(**"Support Vector Machine: %s%%"** % acc\_svm) *# 5. Random Forest* rf = RandomForestClassifier() rf.fit(data\_train, train\_target) acc\_rf = rf.score(data\_test, target\_test) \* 100 print(**"Random Forest: %s%%"** % acc\_rf) *# 6. K Nearest Neighbor* knn = KNeighborsClassifier() knn.fit(data\_train, train\_target) acc\_knn = knn.score(data\_test, target\_test) \* 100 print(**"K Nearest Neighbor: %s%%"** % acc\_knn) |

图5 不同分类器准确率

可以发现，朴素贝叶斯方法在这6种分类器中表现最佳，可能是因为数据规模比较大，提供了足够的先验和似然知识；逻辑蒂斯回归其次，支持向量机的准确率摆动幅度较大，随机森林表现中等，线性回归模型和K近邻比盲猜强，可能是由于数据维度高、分布复杂，线型模型不能很好的拟合。