

華東理工大學

《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	背景	1
2	泰坦尼克号数据集	1
2.1	数据集变量	1
2.2	特征分析	3
2.2.1	性别	4
2.2.2	乘客等级	4
2.2.3	年龄	5
2.2.4	同船的兄弟姊妹及配偶个数	5
2.2.5	同船的父母与子女个数	6
2.2.6	家庭规模	7
2.2.7	登船港口	8
2.2.8	姓名	9
2.2.9	船票编号	11
2.2.10	船票价格	12
3	数据处理	13
3.1	缺值填充	14
3.1.1	年龄	14
3.1.2	船票价格	14
3.1.3	登船港口	15
3.1.4	船舱号	15
3.2	数值化	16
3.2.1	姓名	16
3.2.2	性别	16
3.2.3	船票编号	17
3.2.4	船舱号码	17
3.2.5	登船港口	18
3.3	One-Hot 编码	19
4	特征提取	22
4.1	所有（包括扩展）特征	23
4.2	随机森林特征重要性	23
4.3	One-Hot 相关系数法	24
4.4	核主成分分析	26
4.5	调合	28
5	模型训练	28
6	总结	30
7	小组分工	32

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)、同船的兄弟姐妹及配偶个数 (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。

train											
PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25		S
2	1	1	Cummings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925		S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S
6	0	3	Moran, Mr. James	male		0	0	330877	8.4583		Q
7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	male	2	3	1	349909	21.075		S
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27	0	2	347742	11.1333		S
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14	1	0	237736	30.0708		C
11	1	3	Sandstrom, Miss. Marguerite Rut	female	4	1	1	PP 9549	16.7	G6	S
12	1	1	Bonnell, Miss. Elizabeth	female	58	0	0	113783	26.55	C103	S
13	0	3	Saunderscock, Mr. William Henry	male	20	0	0	A/5. 2151	8.05		S

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()
<class 'pandas.core.frame.DataFrame'>      PassengerId      0
RangeIndex: 891 entries, 0 to 890          Survived      0
Data columns (total 12 columns):           Pclass        0
#   Column          Non-Null Count  Dtype      Name           0
---  ---          -
0   PassengerId      891 non-null    int64      Sex            0
1   Survived         891 non-null    int64      Age            177
2   Pclass           891 non-null    int64      SibSp          0
3   Name             891 non-null    object     Parch          0
4   Sex              891 non-null    object     Ticket         0
5   Age              714 non-null    float64     Fare           0
6   SibSp            891 non-null    int64      Cabin          687
7   Parch            891 non-null    int64      Embarked       2
8   Ticket           891 non-null    object     dtype: int64
9   Fare             891 non-null    float64
10  Cabin            204 non-null    object
11  Embarked         889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

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())
```

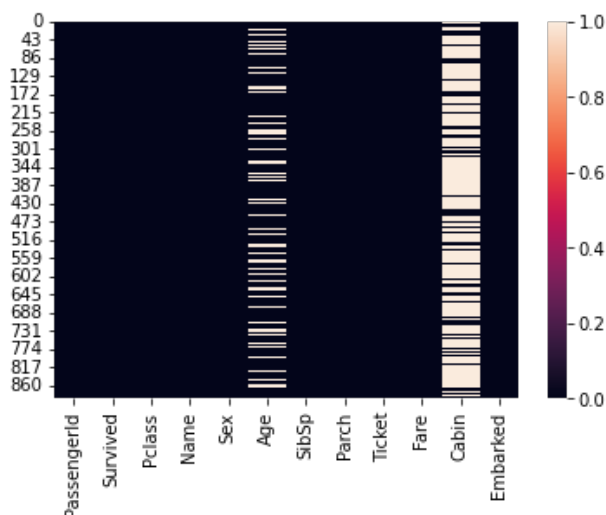


图 2-2 特征缺失值展现图

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)

0    0.616162
1    0.383838
Name: Survived, dtype: float64
```

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)
```

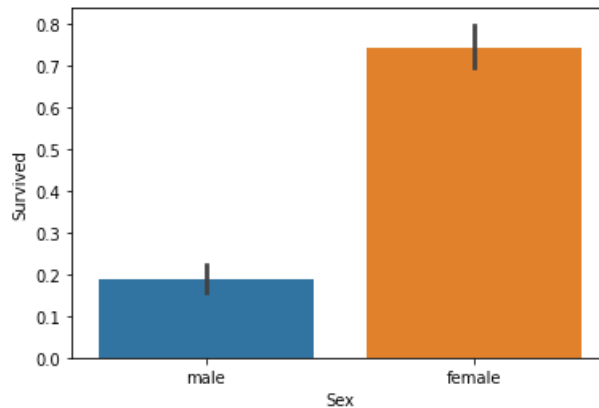


图 2-3 性别特征存活概率

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)
```

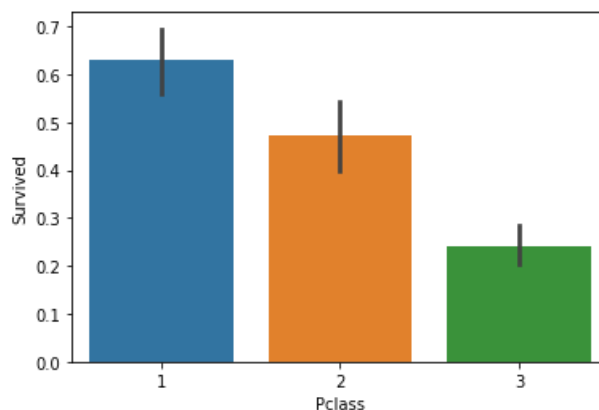


图 2-4 乘客等级特征存活概率

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')
```

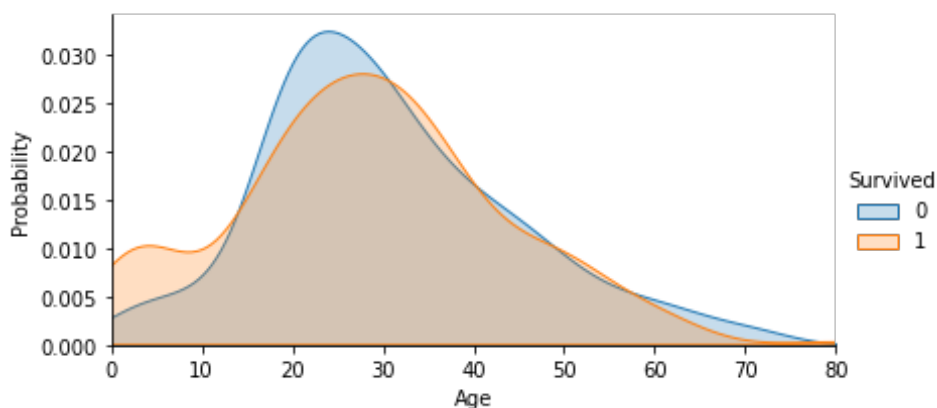


图 2-5 年龄特征存活概率

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 同船的兄弟姐妹及配偶个数

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

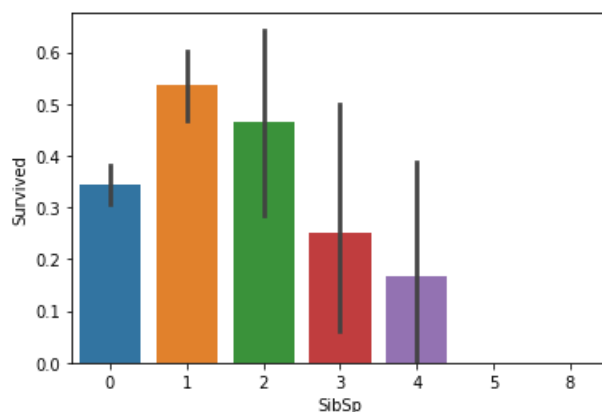


图 2-6 同船的兄弟姊妹及配偶个数特征存活概率

同船的兄弟姊妹及配偶适中的乘客生存率更高，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生存率最高。

2.2.5 同船的父母与子女个数

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

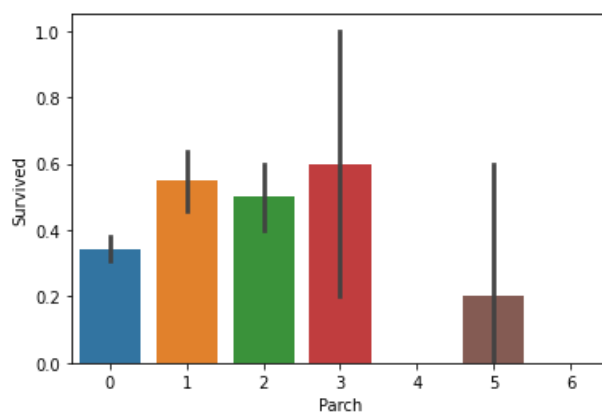


图 2-7 同船的父母与子女个数特征存活概率

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)
```

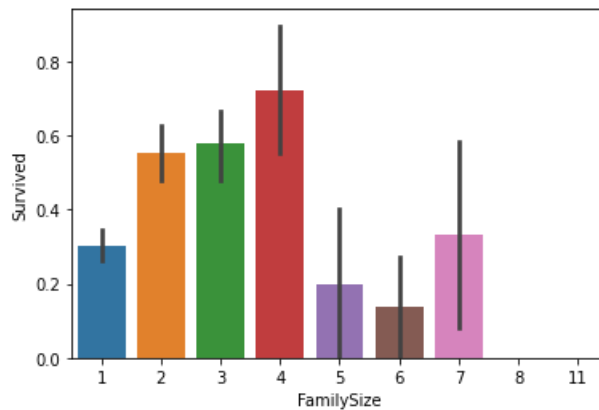


图 2-8 同船的家庭人数特征存活概率

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)
```

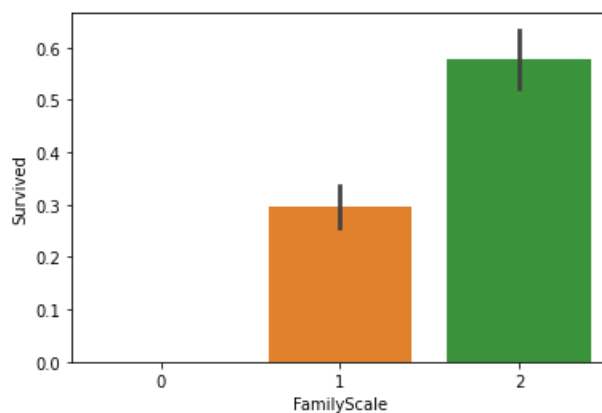


图 2-9 家庭规模特征存活概率

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)
```

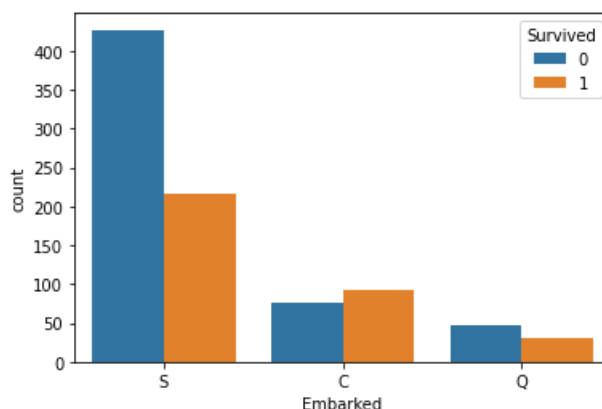


图 2-10 登船港口特征存活概率

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 的比比称呼为 Mr 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()

Mr          757
Miss        260
Mrs         197
Master       61
Rev          8
Dr           8
Col          4
Mlle         2
Major        2
Ms           2
Lady         1
Sir          1
Mme          1
Don          1
Capt        1
the Countess 1
Jonkheer     1
Dona         1
Name: Title, dtype: int64
```

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)
```

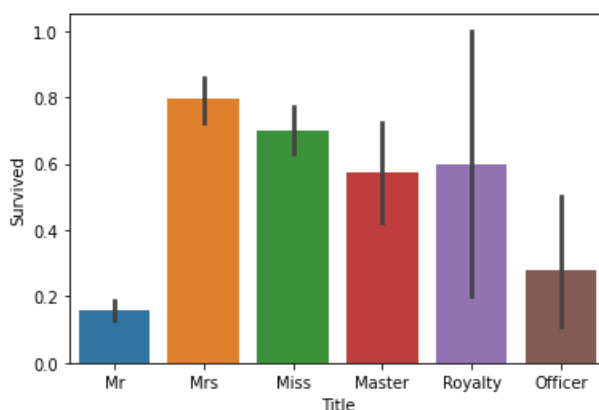


图 2-11 头衔特征存活概率

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)
```

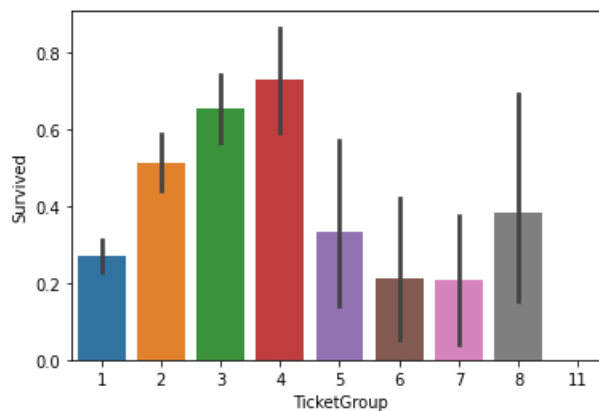


图 2-12 船票编号相同乘客个数存活概率

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)
```

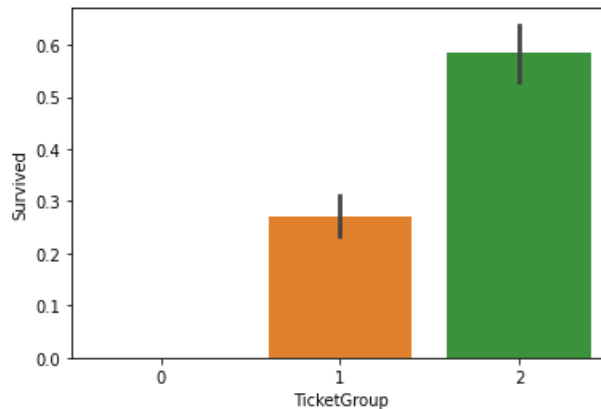


图 2-13 船票编号相同规模存活概率

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')

```

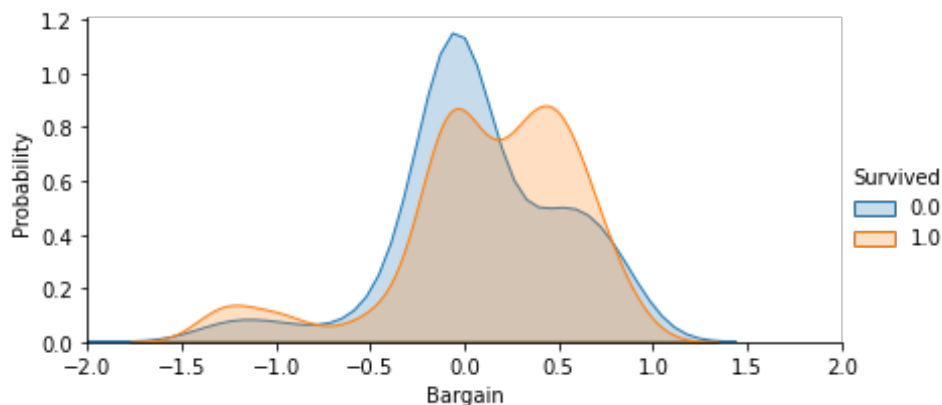


图 2-14 还价能力存活概率

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)
```

Title	
Master	4.0
Miss	22.0
Mr	29.0
Mrs	35.0
Officer	49.5
Royalty	39.5

Name: Age, dtype: float64

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()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 13 columns):
#   Column          Non-Null Count  Dtype
---  -
0   PassengerId     1309 non-null  int64
1   Survived        891 non-null   float64
2   Pclass          1309 non-null  int64
3   Name            1309 non-null  object
4   Sex             1309 non-null  object
5   Age             1309 non-null  float64
6   SibSp           1309 non-null  int64
7   Parch           1309 non-null  int64
8   Ticket          1309 non-null  object
9   Fare            1309 non-null  float64
10  Cabin           1309 non-null  object
11  Embarked        1309 non-null  object
12  Title           1309 non-null  object
dtypes: float64(3), int64(4), object(6)
memory usage: 133.1+ KB
```

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()。
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 15 columns):
#   Column      Non-Null Count  Dtype
---  -
0   PassengerId  1309 non-null   int64
1   Survived     891 non-null    float64
2   Pclass       1309 non-null   int64
3   Name         1309 non-null   object
4   Sex          1309 non-null   int64
5   Age          1309 non-null   float64
6   SibSp        1309 non-null   int64
7   Parch        1309 non-null   int64
8   Ticket       1309 non-null   object
9   Fare         1309 non-null   float64
10  Cabin        1309 non-null   object
11  Embarked     1309 non-null   int64
12  Title        1309 non-null   int64
13  TicketGroup  1309 non-null   int64
14  Deck         1309 non-null   int64
dtypes: float64(3), int64(9), object(3)
memory usage: 153.5+ KB
(891, 15) (418, 15) (1309, 15)
(891, 14) (891,)
```

```

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()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   Pclass          891 non-null    int64
1   Sex             891 non-null    int64
2   Age            891 non-null    float64
3   SibSp          891 non-null    int64
4   Parch          891 non-null    int64
5   Fare           891 non-null    float64
6   Embarked       891 non-null    int64
7   Title          891 non-null    int64
8   TicketGroup    891 non-null    int64
9   Deck           891 non-null    int64
10  FamilySize     891 non-null    int64
11  FamilyScale    891 non-null    int64
dtypes: float64(2), int64(10)
memory usage: 83.7 KB
-----

```

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()

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 33 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Sex                                    891 non-null    int64
1   Age                                    891 non-null    float64
2   SibSp                                  891 non-null    int64
3   Parch                                  891 non-null    int64
4   Fare                                    891 non-null    float64
5   FamilySize                             891 non-null    int64
6   Pclass_1                              891 non-null    uint8
7   Pclass_2                              891 non-null    uint8
8   Pclass_3                              891 non-null    uint8
9   Title_Master                          891 non-null    uint8
10  Title_Miss                            891 non-null    uint8
11  Title_Mr                              891 non-null    uint8
12  Title_Mrs                             891 non-null    uint8
13  Title_Officer                         891 non-null    uint8
14  Title_Royalty                         891 non-null    uint8
15  Deck_A                                891 non-null    uint8
16  Deck_B                                891 non-null    uint8
17  Deck_C                                891 non-null    uint8
18  Deck_D                                891 non-null    uint8
19  Deck_E                                891 non-null    uint8
20  Deck_F                                891 non-null    uint8
21  Deck_G                                891 non-null    uint8
22  Deck_T                                891 non-null    uint8
23  Deck_U                                891 non-null    uint8
24  Embarked_C                            891 non-null    uint8
25  Embarked_Q                            891 non-null    uint8
26  Embarked_S                            891 non-null    uint8
27  FamilyScale_Medium                    891 non-null    uint8
28  FamilyScale_Single & Large            891 non-null    uint8
29  FamilyScale_Super                     891 non-null    uint8
30  TicketGroup_Fine                      891 non-null    uint8
31  TicketGroup_Single & Multi            891 non-null    uint8
32  TicketGroup_Super                     891 non-null    uint8
dtypes: float64(2), int64(4), uint8(27)
memory usage: 65.4 KB

```

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'])
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Pclass      891 non-null    int64
1   Sex         891 non-null    int64
2   Age         891 non-null    float64
3   SibSp       891 non-null    int64
4   Parch       891 non-null    int64
5   Fare        891 non-null    float64
6   Embarked    891 non-null    int64
7   Title       891 non-null    int64
8   TicketGroup 891 non-null    int64
9   Deck        891 non-null    int64
dtypes: float64(2), int64(8)
memory usage: 69.7 KB
Accuracy: 78.73134328358209%.
```


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'])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   Pclass          891 non-null   int64
1   Sex             891 non-null   int64
2   Age            891 non-null   float64
3   SibSp          891 non-null   int64
4   Parch          891 non-null   int64
5   Fare           891 non-null   float64
6   Embarked       891 non-null   int64
7   Title          891 non-null   int64
8   TicketGroup    891 non-null   int64
9   Deck           891 non-null   int64
10  FamilySize     891 non-null   int64
11  FamilyScale    891 non-null   int64
dtypes: float64(2), int64(10)
memory usage: 83.7 KB
Accuracy: 77.98507462686567%.
```

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 0.24783240980216295
Sex 0.22085662840251638
Fare 0.10848789055203537
Pclass 0.09826473523861358
Deck 0.0811458119450691
Age 0.0743591248937088
FamilySize 0.05168208275425809
TicketGroup 0.04053647877036051
SibSp 0.022376317035377555
Embarked 0.020455586651696476
FamilyScale 0.01857175146279034
Parch 0.01543118249141089
```

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'])
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 2 columns):
#   Column  Non-Null Count  Dtype
---  ---
0    Sex      891 non-null    int64
1   Title     891 non-null    int64
dtypes: int64(2)
memory usage: 14.0 KB
Accuracy: 77.61194029850746%.
```

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'])
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 6 columns):
#   Column  Non-Null Count  Dtype
---  ---
0   Pclass   891 non-null    int64
1    Sex     891 non-null    int64
2    Age     891 non-null    float64
3   Fare     891 non-null    float64
4   Title    891 non-null    int64
5   Deck     891 non-null    int64
dtypes: float64(2), int64(4)
memory usage: 41.9 KB
Accuracy: 80.22388059701493%.
```

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)
```

Survived	1.000000
Sex	0.543351
Title_Mrs	0.344935
Title_Miss	0.332795
TicketGroup_Fine	0.315591
Pclass_1	0.285904
FamilyScale_Medium	0.279855
Fare	0.257307
Deck_B	0.175095
Embarked_C	0.168240
Deck_D	0.150716
Deck_E	0.145321
Deck_C	0.114652
Pclass_2	0.093349
Title_Master	0.085221
Parch	0.081629
Deck_F	0.057935
Title_Royalty	0.033391
Deck_A	0.022287
FamilySize	0.016639
Deck_G	0.016040
Embarked_Q	0.003650
PassengerId	-0.005007
Deck_T	-0.026456
Title_Officer	-0.031316
SibSp	-0.035322
TicketGroup_Super	-0.070234
Age	-0.072174
FamilyScale_Super	-0.096040
Embarked_S	-0.149683
FamilyScale_Single & Large	-0.252576
TicketGroup_Single & Multi	-0.301392
Deck_U	-0.316912
Pclass_3	-0.322308
Title_Mr	-0.549199

Name: Survived, dtype: float64

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'])
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   Sex              891 non-null    int64
1   Title            891 non-null    int64
2   TicketGroup      891 non-null    int64
3   FamilyScale      891 non-null    int64
4   Fare             891 non-null    float64
5   Pclass           891 non-null    int64
6   Deck             891 non-null    int64
dtypes: float64(1), int64(6)
memory usage: 48.9 KB
Accuracy: 79.47761194029852%.
```

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 核主成分分析

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()
```

```
[9.37754652e-01 5.86131572e-02 1.46791012e-03 1.19226536e-03
 3.50277912e-04 1.91078957e-04 1.53510495e-04 1.22472619e-04
 7.95740227e-05 5.21396603e-05 2.29614829e-05 1.65877505e-34]
[0.93775465 0.99636781 0.99783572 0.99902798 0.99937826 0.99956934
 0.99972285 0.99984532 0.9999249 0.99997704 1. 1. ]
```

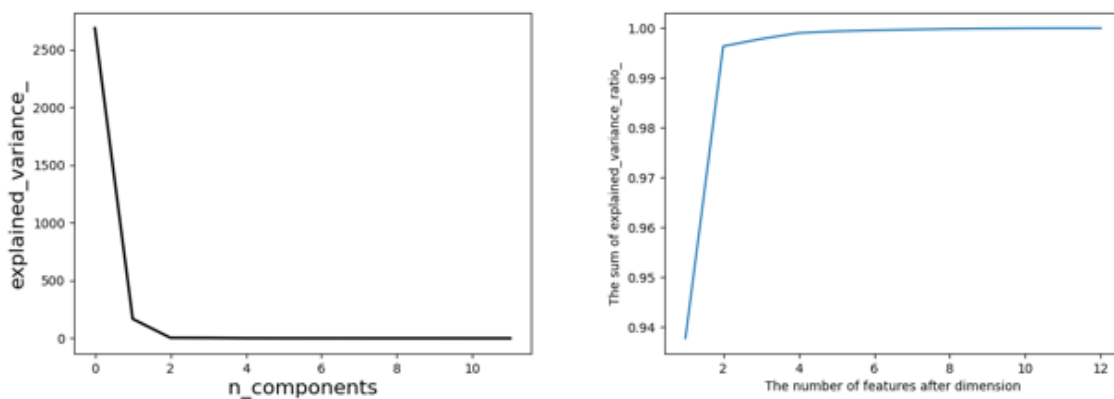


图 4-1 主成分分析

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。

```
scaler = StandardScaler()
X_train = scaler.fit_transform(data_train)
X_test = scaler.transform(data_test)
transformer = KernelPCA(n_components=7, kernel='rbf')
X_re_train = transformer.fit_transform(X_train)
X_re_test = transformer.transform(X_test)
print(X_re_train.shape, X_re_test.shape)
print("Fitting the classifier to the training set")
param_grid = {
    "C": loguniform(1e3, 1e5),
    "gamma": loguniform(1e-4, 1e-1),
}
clf = RandomizedSearchCV(
    SVC(kernel="rbf", class_weight="balanced"), param_grid, n_iter=10)
clf = clf.fit(X_re_train, train_target)
print("Best estimator found by grid search:")
```

```
print(clf.best_estimator_)
acc = clf.score(X_re_test, target_test)
print("The Classification Accuracy is %.2f%%." %(acc*100))
```

Fitting the classifier to the training set

Best estimator found by grid search:

SVC(C=8575.276620050307, class_weight='balanced', gamma=0.08458639408880911)

The Classification Accuracy is 88.28%.

经过高斯核主成分分析，选取前 7 个特征，相比于原始所有特征，以及挑选出的 7 个特征，结果都要好出不少。

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'])
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 891 entries, 0 to 890
```

```
Data columns (total 7 columns):
```

#	Column	Non-Null Count	Dtype
0	Pclass	891 non-null	int64
1	Sex	891 non-null	int64
2	Age	891 non-null	float64
3	Fare	891 non-null	float64
4	Title	891 non-null	int64
5	Deck	891 non-null	int64
6	FamilyScale	891 non-null	int64

```
dtypes: float64(2), int64(5)
```

```
memory usage: 48.9 KB
```

```
Accuracy: 80.59701492537313%.
```

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)

```

Linear Regression: 68.25293502662086%
Logistic Regression: 94.01913875598086%
Naive Bayes: 99.76076555023924%
Support Vector Machine: 90.19138755980862%
Random Forest: 81.57894736842105%
K Nearest Neighbor: 69.85645933014354%

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。

6 总结

通过本次实验我们得出以下结论：

1. 船上男性乘客人数大约为女性乘客的 2 倍，然而男性乘客的存活率仅约为 20%，女性乘客的存活率达到 70%以上，很可能泰坦尼克号当时采取女性优先逃离的原则。
2. 大部分乘客是从泰坦尼克号起始地点英国南安普敦港口出发的，而这部分乘客的存活率却最低，可能与其基数大、乘客身份复杂有关。在法国瑟堡市港口登船的乘客相对存活率最高，达到 50%以上。
3. 船票价格越高，乘客所在的客舱等级也越高，乘客存活率也越高。购买三等舱船票的乘客最多，存活率最低，这是由于客舱等级越低，所居住的位置就越靠近船舱的底部，灾难发生时逃生所需的时间越久，导致存活率越低。

4. 乘客中已婚男士数量最多，其次是未婚女士和已婚女士。然而已婚男士的存活率最低，已婚女士和未婚女士的存活率分别居第一和第二，进一步证明了泰坦尼克号当时采取女性优先逃离的原则。

5. 独自出行的乘客人数最多，其次是中等规模家庭（2~4人）的乘客人数，5人以上的大规模家庭较少。从存活率上看，由于大规模家庭在三等舱乘客占比最高，中等规模家庭在一等舱乘客占比最高，因此大规模家庭的乘客存活率最低，中等规模家庭的乘客存活率最高。

本次《泰坦尼克号生还与否预测》通过 6 种分类方法，准确率分别为：

Linear Regression	68.25%
Logistic Regression	94.02%
Naive Bayes	99.76%
Support Vector Machine	90.19%
Random Forest	81.58%
K Nearest Neighbor	69.86%

该实验具有 12 个特征变量 1300 组数据。在实验过程中，经过对泰坦尼克号沉船事件的背景调查，我们对数据及特征进行了深入了解。通过数据清洗，数据缺失补全，及单特征分析，我们除去了 2 个对预测无关的变量：游客编号 (PassengerId)、幸存与否 (Survived)。初步数据分析完毕，接下来，我们对其余 9 个重要特征变量进行可视化、有机组合和深入分析。

为了从人名中的爵位等信息中抽取身份信息，从家庭规模中抽取可能相关联的信息，我们对这些对象作数值化处理，并用 One-hot 编码方式将离散特征值映射至欧氏空间。同时，为了减弱特征与特征之间的强关联性，我们尽可能把关联特征化为弱关联特征。最后，根据在随机森林中验证集验证的准确率，我们选择 Pclass、Sex、Age、Fare、Title、Deck（即 Cabin）、FamilyScale（由同船的兄弟姊妹及配偶个 Sibsp 及同船的父母与子女个数 Parch 两个特征变量组成）共七个特征输入模型进行训练。



























如此，前期大量的数据处理工作使得数据具有更好的完整性，可解释性，并保留了特征空间的原始关联，为模型训练提供了良好的数据。

最终，朴素贝叶斯方法在这 6 种分类器中表现最佳，高达 99.76%，这也印证了朴素贝叶斯在特征尽可能独立的情况下更加贴合数据的概率模型，且在数据规模较大时呈现很好的稳定性。因此，在能提供足够先验和似然知识的前提下，朴素贝叶斯是高鲁棒性的选择。

7 小组分工

本次案例采用轮流交替的方式开展，保证每一位同学都沉浸式地参与到项目的设计、实践和总结中来，对每一个环节都使用腾讯会议在小组中进行交流：会议前同学在网络中搜集资料，会议上对方案进行横向评测，最后挑选出适合的方案在会议后使用 python 实现，实现后再次通过会议交流总结。

具体实施方案如下表：

	牟容源	吕 阳	宋之琦
背景研究	 		
数据分析	 		
数据处理			 
特征提取	 	 	 
模型训练	 	 	
总 结			

注：  表示材料收集；
 表示会议记录；
 表示代码实现；
 表示总结撰写。

表 7 小组分工矩阵