

机器学习课程设计实验报告

June 12, 2018

1 问题描述

使用 Kaggle 上面给出的训练集（训练数据是一些乘客的个人信息及其存活情况），尝试建立合适的模型，并根据给出的测试数据预测其他人的存活情况。

2 数据分析

2.1 数据预处理

首先为了对数据进行更好的分析，我们使用 **ipython notebook** 导入训练数据。

In [3]:

train

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S

Figure 1: 使用 ipython 导入数据

2.1.1 异常数据去除

在对数据进行分析之前，我们首先使用 **Tukey 方法检测并去除数据中的异常点**。即针对 Age SibSp Parch Fare 四个数字类型的属性进行检测，若某一条数据的其中多于两个属性被判断为异常点，则将其去除。

下面是检测出的 10 个异常数据，其中 28, 89, 342 号乘客具有非常高的 Ticket Fare，其余 7 位乘客则具有非常高的 Sibsp 值：

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
27	28	0	1	Fortune, Mr. Charles Alexander	male	19.0	3	2	19950	263.00	C23 C25 C27	S
88	89	1	1	Fortune, Miss. Mabel Helen	female	23.0	3	2	19950	263.00	C23 C25 C27	S
159	160	0	3	Sage, Master. Thomas Henry	male	NaN	8	2	CA. 2343	69.55	NaN	S
180	181	0	3	Sage, Miss. Constance Gladys	female	NaN	8	2	CA. 2343	69.55	NaN	S
201	202	0	3	Sage, Mr. Frederick	male	NaN	8	2	CA. 2343	69.55	NaN	S
324	325	0	3	Sage, Mr. George John Jr	male	NaN	8	2	CA. 2343	69.55	NaN	S
341	342	1	1	Fortune, Miss. Alice Elizabeth	female	24.0	3	2	19950	263.00	C23 C25 C27	S
792	793	0	3	Sage, Miss. Stella Anna	female	NaN	8	2	CA. 2343	69.55	NaN	S
846	847	0	3	Sage, Mr. Douglas Bullen	male	NaN	8	2	CA. 2343	69.55	NaN	S
863	864	0	3	Sage, Miss. Dorothy Edith "Dolly"	female	NaN	8	2	CA. 2343	69.55	NaN	S

Figure 2: 数据中的异常点

2.1.2 数据初步统计

在去除了异常点的基础上，我们对训练数据中的空值进行统计，统计结果如 Figure 4 所示。
除了空值信息，对于数值型数据的一些分布我们也做一初步的统计。统计结果如 Figure 3 所示。

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	881.000000	881.000000	881.000000	711.000000	881.000000	881.000000	881.000000
mean	446.713961	0.385925	2.307605	29.731603	0.455165	0.363224	31.121566
std	256.617021	0.487090	0.835055	14.547835	0.871571	0.791839	47.996249
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	226.000000	0.000000	2.000000	20.250000	0.000000	0.000000	7.895800
50%	448.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.000000	1.000000	3.000000	38.000000	1.000000	0.000000	30.500000
max	891.000000	1.000000	3.000000	80.000000	5.000000	6.000000	512.329200

Figure 3: 数值型数据的分布信息

```

PassengerId    0
Survived        0
Pclass          0
Name            0
Sex             0
Age            170
SibSp           0
Parch           0
Ticket         170
Fare            0
Cabin          680
Embarked        2

```

Figure 4: 各属性空值个数统计



Figure 5: 数值型属性的相关性矩阵

2.2 数据初步分析

对于数值型的属性，我们使用热图观察其与生存情况的相关性，结果如 Figure 5 所示。

图中只有 Fare 和 Pclass 两个特征和生存情况有比较大的相关性，但这并不意味着其他的特征没有用处，或许这些特征数据的子群和生存情况有很大的相关性。

因此，我们需要进一步探究数据。

2.2.1 家庭成员人数

SibSP 和 Parch 属性分别代表了同龄亲属和非同龄亲属。根据常识进行初步判断，对于一个成年乘客，有非常多亲人与自己同行，意味着在灾难发生时需要用更多的精力照顾亲人，因而生存可能性会降低。

通过统计，正如 Figure 6 和 Figure 7 所示，SibSp 和 Parch 个数多的乘客果然具有相对少的生存可能性。

这两个属性共同反映了同行的家庭成员的人数大小，在后面的建模中，我们或许可以将其合成为同一特征。

2.2.2 女人和儿童

接着我们把关注点放在性别和年龄上面，在对数据进行观察之前，我们所做的假设是：女人比男性更有机会存活。年龄属性会对生存概率有很大的影响。

正如 Figure 8 所示，年龄的分布函数对于生存情况和遇难情况是不同的，两个整体呈现一个基本的高斯分布，但同时年龄较小的区间我们可以观察到一个峰值。在非常年幼的乘客（0 岁至五岁）有更多的机会存活，而年龄非常大的乘客则更有可能遇难。

对于性别这一特征，结论（Figure 9）和我们的假设一致，女性比男性具有更高的获救机会。

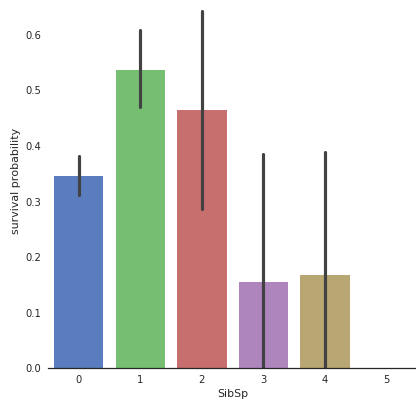


Figure 6: Sibsp 个数与生存可能性关系

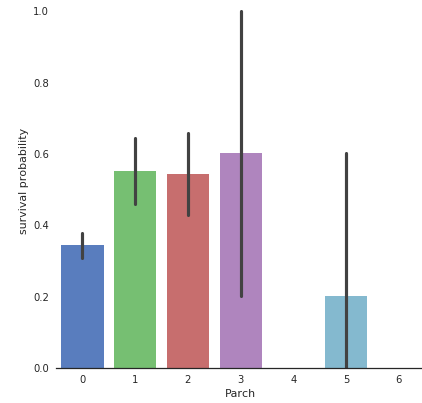


Figure 7: Parch 个数与生存可能性关系

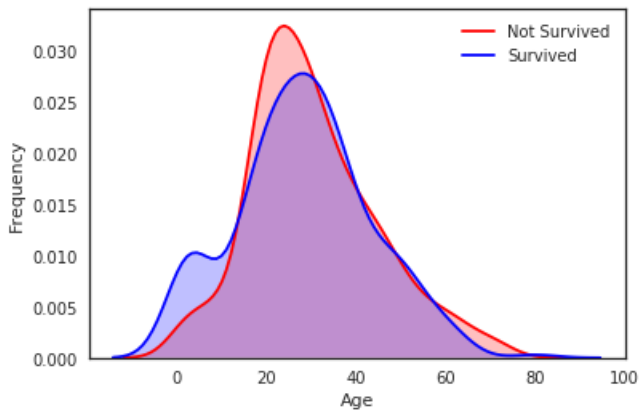


Figure 8: 存活率与遇难率和年龄的分布关系

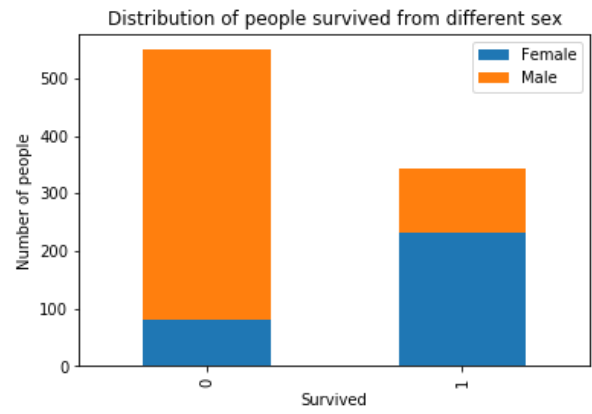


Figure 9: 存活率与遇难率和性别的关系

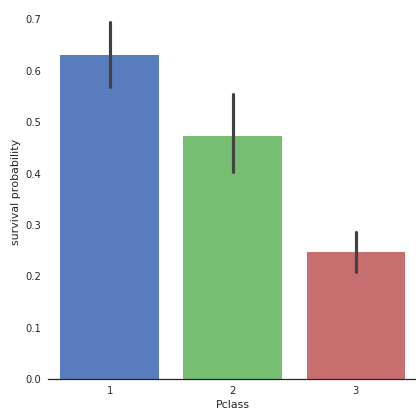


Figure 10: Pclass 与生存情况关系

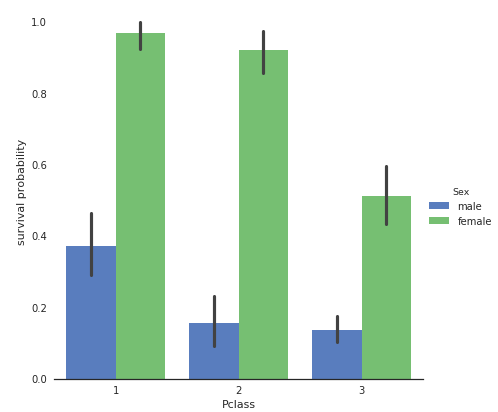


Figure 11: 不同 Pclass 下的性别与生存情况关系

2.2.3 船舱等级

通过 Figure10 我们可以看出，**头等舱乘客比三等舱乘客更有机会获救**，这有可能和客舱所处位置有关，当然也有可能和获救的优先顺序有关。当将乘客分为男乘客和女乘客后，这一特征仍被保存。同时女性比男性更有机会获救这一结论也被进一步证实。

2.2.4 Cabin 属性

对于 Cabin 属性，根据常识初步判断不同的 Cabin 值对应不同的船舱位置，因此我们可以对 Cabin 值中的甲板号进行提取，用于训练数据。

2.2.5 Embarked 属性

由于有两个数据的登陆港口属性是空值，因此在分析之前，我们用数据最多的 S 对其进行了填充。

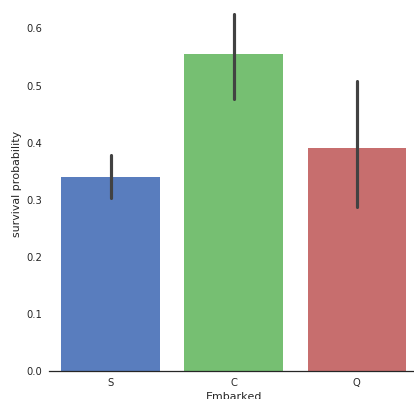


Figure 12: Embarked、Pclass 属性与生存情况关系

根据 Figure 12 反应的估计数据，我们似乎可以得出来自 **C** 的乘客具有**更高的存活概率**。为了分析其中原因，初步假设港口和乘客的富有程度有一定关系，并作了进一步分析。

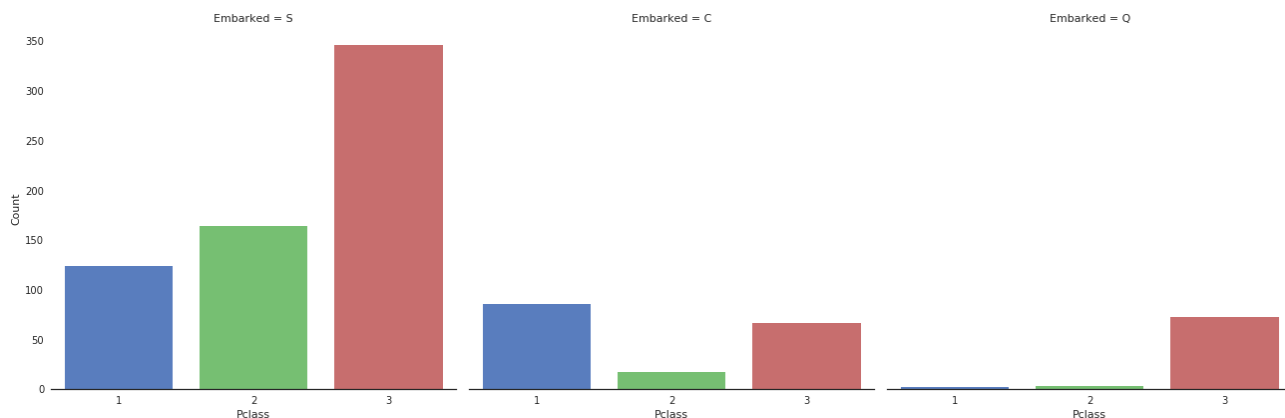


Figure 13: Embarked、Pclass 属性与生存情况关系

确实三等舱的乘客普遍来自 S 和 Q 港口，而头等舱乘客则更多来自 C 港。

2.3 结论

根据初步的数据分析，我们可以得出如下结论：

- Age 缺失值很多需要填补
- Fare 取值范围较大需要标准化
- 是否获救与 Sex、Pclass、Age、Fare 关系很大
- 年龄非常小的乘客有更高的获救概率
- 同行的家庭成员人数越多，生存可能性越小
- Cabin 所反映的甲板号和获救情况有一定关系
- 来自 C 港的乘客相比其他具有更高的获救概率

3 特征工程

3.1 填充缺失值

首先将 Embarked 这一属性的缺失值以出现最多的属性值 S 填充

```
full_data.Embarked.fillna('S', inplace=True)
```

再将票价属性的缺失值以未缺失的票价的中位数填充。

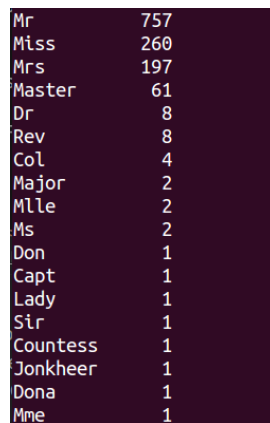
```
full_data.Fare.fillna(np.median(full_data.Fare[full_data.Fare.notnull()]), inplace=True)
```

3.2 提取特征

下面从数据中提取特征进行建模。

3.2.1 姓名

姓名的特征是字符串数据，从中提取特征比较困难，因此首先将姓名中的称谓信息提取出来，发现不同称谓的出现次数大致如下：



Mr	757
Miss	260
Mrs	197
Master	61
Dr	8
Rev	8
Col	4
Major	2
Mlle	2
Ms	2
Don	1
Capt	1
Lady	1
Sir	1
Countess	1
Jonkheer	1
Dona	1
Mme	1

Figure 14: 不同称谓出现次数统计

即将出现次数较少的男性称谓 'Capt', 'Col', 'Don', 'Dr', 'Jonkheer', 'Major', 'Rev', 'Sir' 和女性称谓 'Countess', 'Lady', 'Mlle', 'Mme', 'Ms', 'Dona' 分别用 'Male Rare' 和 'Female Rare' 替代。

```
df['Title'] = df['Title'].replace(['Capt', 'Col', 'Don', 'Dr',  
'Jonkheer', 'Major', 'Rev', 'Sir'], 'Male_Rare')  
df['Title'] = df['Title'].replace(['Countess', 'Lady', 'Mlle', 'Mme', 'Ms', 'Dona'], 'Female_Rare')
```

同时增加姓名长度属性

```
df['NameLength'] = df.Name.apply(lambda x: len(x))
```

由于年轻女性的存活概率较大，因此将年龄小于 14 岁或有父母陪同的女性称谓改为 'Girl'

```
def girl(aa):  
    if (aa.Age != 0) & (aa.Title == 'Miss') & (aa.Age <= 14):  
        return 'Girl'  
    elif (aa.Age == 0) & (aa.Title == 'Miss') & (aa.Parch != 0):  
        return 'Girl'  
    else:  
        return aa.Title
```

至此姓名属性特征提取结束。

3.2.2 年龄

由于年龄属性存在大量缺失值，因此需要填补缺失值，填补方式为将缺失的年龄设置为同样称谓的未缺失年龄的中位数。

```
# fill missing ages according to title  
def set_missing_ages_title(df):  
    Tit = ['Mr', 'Miss', 'Mrs', 'Master', 'Girl', 'Male_Rare', 'Female_Rare']  
    for i in Tit:  
        df.loc[(df.Age == 0) & (df.Title == i), 'Age'] = df.loc[df.Title == i, 'Age'].median()  
    return df
```

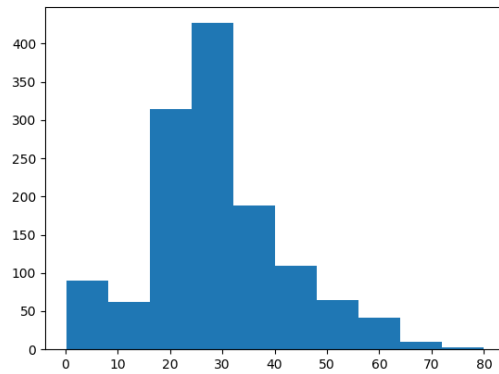


Figure 15: 年龄分布

填补好缺失值以后，对于年龄值的分布进行初步统计
将年龄值按照年龄段划分

```
def encode_age(aa):
    if aa.Age > 0 and aa.Age < 14:
        return 0
    if aa.Age >= 14 and aa.Age < 30:
        return 1
    if aa.Age >= 30 and aa.Age < 45:
        return 2
    if aa.Age >= 45 and aa.Age < 60:
        return 3
    return 4
```

3.2.3 客舱，登船口

对于客舱这一属性，形式为字符串，因此只将字符串的首字母提取出来，首字母有可能反映客舱在船上的位置，对于是否获救会产生影响。

```
df['CabinCat'] = pd.Categorical.from_array(
    (df.Cabin.fillna('0').apply(lambda x: x[0])).codes
```

对于登船口这一属性，由于其只有'S', 'C', 'Q' 三个值，直接将其转换为编码。

```
df['EmbarkedCat'] = pd.Categorical.from_array(df.Embarked).codes
```

同样将称谓属性转换为编码：

```
title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3,
                  "Master": 4, "Girl": 5, "Male_Rare": 6, "Female_Rare": 7}
df["TitleCat"] = df.loc[:, 'Title'].map(title_mapping)
```

对于票价属性，由于很难从中提取到有用特征，因此直接将其删除。

3.2.4 家庭成员

对于'SibSp' 和'Parch' 属性，增加'FamilySize' 属性，计算兄弟及子女的数量和，同时根据家庭成员人数将其划分为'Large Family', 'Media Family', 'Small Family' 和'Single'

```
def get_family_size(df):
    df['FamilySize'] = df["SibSp"] + df["Parch"] + 1
    df['Single'] = df['FamilySize'].map(lambda s: 1 if s == 1 else 0)
    df['SmallF'] = df['FamilySize'].map(lambda s: 1 if s == 2 else 0)
    df['MedF'] = df['FamilySize'].map(lambda s: 1 if 3 <= s <= 4 else 0)
    df['LargeF'] = df['FamilySize'].map(lambda s: 1 if s >= 5 else 0)
```

考虑到属于同一家庭的获救概率会更大一些，因此增加'Family Id' 属性，根据姓氏划分每位乘客所属的家庭。

```
def get_family_id(row):
    global family_id_mapping
    last_name = row["Name"].split(",")[0]
    family_id = "{0}{1}".format(last_name, row["FamilySize"])
```

```

if family_id not in family_id_mapping:
    if len(family_id_mapping) == 0:
        current_id = 1
    else:
        current_id = max(family_id_mapping.values()) + 1
    family_id_mapping[family_id] = current_id
return family_id_mapping[family_id]

```

由于'Family Id' 的属性值太多，因此将家庭成员数小于 3 的'Family Id' 设为-1

```

family_ids = df.apply(get_family_id, axis=1)
family_ids[df["FamilySize"] < 3] = -1

```

3.2.5 性别，年龄

由于性别，年龄这两个属性对于是否获救的影响很大，因此将这两个属性结合起来生成'Person' 属性，将乘客划分为'child', 'female adult', 'male adult'

```

def get_person(passenger):
    age, sex = passenger
    child_age = 14
    if (age < child_age):
        return 'child'
    elif (sex == 'female'):
        return 'female_adult'
    else:
        return 'male_adult'

```

'person' 属性的分布如下

male_adult	782
female_adult	410
child	117

Figure 16: 'person' 属性值统计

最后将'person', 'Sex', 'Pclass', 'Title' 属性转换为 one-hot 编码

```

def dummy(df):
    sex_dummies = pd.get_dummies(df['Sex'])
    person_dummies = pd.get_dummies(df['person'])
    class_dummies = pd.get_dummies(df['Pclass'], prefix= 'Pclass')
    title_dummies = pd.get_dummies(df['Title'])
    df = pd.concat([df, sex_dummies, person_dummies], axis = 1)
    df.drop(['Sex', 'person'], axis = 1, inplace = True)
    return df

```

至此特征提取部分结束，下面建立模型进行回归预测。

4 构造模型

4.1 随机森林模型

4.1.1 基本原理

随机森林主要是决策树的集成，是一个能够回归和分类的模型。它具备处理大数据的特性，而且它有助于估计变量，是非常重要的基础数据建模。在随机森林中，每个决策树的 n 个分类特征是在所有特征中随机选择的。每棵决策树都是一个分类器，对于一个输入样本， N 棵树会有 N 个分类结果。而随机森林集成了所有的分类投票结果，将投票次数最多的类别指定为最终的输出。随机森林最重要的参数主要有两个，'n_estimators' 和 'max_features'。'n_estimators' 表示随机森林中决策树的数目，但是并不是取得越大就会越好，预测效果最好的将会出现在合理的树个数。'max_features' 表示随机选择特征集合的子集合，并用来分割节点。子集合的个数越少，方差就会减少的越快，但同时偏差就会增加的越快。

4.1.2 调节参数

随机森林中重要的参数如下

- max_depth: 设置树的最大深度，默认为 None，这样建树时，会使每一个叶节点只有一个类别，或是达到 min_samples_split
- min_samples_split: 根据属性划分节点时，每个划分最少的样本数
- min_samples_leaf: 叶子节点最少的样本数
- max_leaf_nodes: 叶子树的最大样本数
- min_weight_fraction_leaf: (default=0) 叶子节点所需要的最小权值
- n_estimators: 决策树的个数，越多越好，但是性能就会越差

使用交叉验证的方式调节寻找最优参数

```
def build_model_rfc(X_train, y_train):
    RFC = RandomForestClassifier()
    ## Search grid for optimal parameters
    rf_param_grid = {"max_depth": [None],
                     "max_features": [1, 3, 10],
                     "min_samples_split": [2, 3, 10],
                     "min_samples_leaf": [1, 3, 4, 10],
                     "bootstrap": [False],
                     "n_estimators": [100, 300],
                     "criterion": ["gini"]}
    kfold = StratifiedKFold(n_splits=10)
    gsRFC = GridSearchCV(RFC, param_grid =
rf_param_grid, cv=kfold, scoring="accuracy", n_jobs= 4, verbose = 1)
    gsRFC.fit(X_train, y_train)
    RFC_best = gsRFC.best_estimator_
    return RFC_best, gsRFC.best_score_
```

其中 `rf_param_grid` 定义候选参数，`StratifiedKFold` 将训练集平均分成十份，九份用作训练集，剩下一份用作测试集。`GridSearchCV` 遍历所有候选参数用来寻找最优参数，返回最优参数对应的模型。再用最优参数对应的模型拟合训练样本，返回模型及其对应的分数。

最后用最优模型拟合测试样本。

```
pred_rfc = rfc_model.predict(X_test)
```

4.2 迭代决策树模型

4.2.1 基本原理

迭代决策树通过一系列的迭代来优化分类结果，每迭代一次引入一个弱分类器，来克服现在已经存在的弱分类器组合的缺陷，这个缺陷的表征就是梯度，在迭代的时候选择梯度下降的方向来保证最后的结果最好。损失函数用来描述模型的“靠谱”程度，假设模型没有过拟合，损失函数越大，模型的错误率越高。如果模型能够让损失函数持续的下降，则说明我们的模型在不停的改进，而最好的方式就是让损失函数在其梯度方向上下降。

4.2.2 调节参数

迭代决策树模型重要的参数如下

- loss: 选择损失函数, 默认值为 ls(least squares)
- learning_rate: 学习率, 默认值 0.1
- n_estimators: 弱学习器的数目, 默认值 100
- max_depth: 每一个学习器的最大深度, 限制回归树的节点数目, 默认为 3
- min_weight_fraction_leaf: 叶子节点所需要的最小权值
- min_samples_split: 可以划分为内部节点的最小样本数, 默认为 2
- min_samples_leaf: 叶节点所需的最小样本数, 默认为 1

使用交叉验证的方式调节寻找最优参数

```
def build_model_gbc(X_train, y_train):
    GBC = GradientBoostingClassifier()
    gb_param_grid = {'loss' : ["deviance"],
                     'n_estimators' : [100,200,300],
                     'learning_rate': [0.1, 0.05, 0.01],
                     'max_depth': [4, 8],
                     'min_samples_leaf': [100,150],
                     'max_features': [0.3, 0.1]
    }
    kfold = StratifiedKfold(n_splits=10)
    gsGBC = GridSearchCV(GBC,param_grid =
gb_param_grid, cv = kfold, scoring="accuracy", n_jobs= 4, verbose = 1)
    gsGBC.fit(X_train,y_train)
    GBC_best = gsGBC.best_estimator_
    return GBC_best, gsGBC.best_score_
```

其中 *gb_param_grid* 定义候选参数, *StratifiedKfold* 将训练集平均分成十份, 九份用作训练集, 剩下一份用作测试集。 *GridSearchCV* 遍历所有候选参数用来寻找最优参数, 返回最优参数对应的模型。再用最优参数对应的模型拟合训练样本, 返回模型及其对应的分数。

最后用最优模型拟合测试样本。

```
pred_gbc = gbc_model.predict(X_test)
```

4.3 逻辑回归模型

4.3.1 基本原理

Logistic 回归的过程就是根据样本求解分类直线 (或者超平面) 的参数过程, 求解的方法是极大似然估计法, 但是因为似然方程最后求导无法得到解析解, 所以用了梯度下降法去逐步优化得到极值。

4.3.2 构造模型

下面用逻辑回归模型拟合训练数据并预测结果

```
lrc = LogisticRegression()
X_training,X_val, y_training, y_val =
    train_test_split(X_train,y_train,test_size=0.3, random_state=0)
lrc.fit(X_training, y_training)
val_lrc = lrc.predict(X_val)
```

将训练集按照 0.3 的比例划分为训练集和交叉验证集, 在训练集上训练模型, 并在交叉验证集上测试模型。最后在测试集上预测结果

```
pred_lrc = lrc.predict(X_test)
```

5 结果分析与讨论

测试发现，随机森林模型在交叉验证集上的效果最好，因此对随机森林模型在交叉验证集上的表现进行分析讨论。

对于随机森林模型，用返回的最优模型拟合测试数据，并输出在交叉验证集上的结果。

	precision	recall	f1-score	support
0.0	0.81	0.95	0.88	110
1.0	0.90	0.64	0.75	69
avg / total	0.84	0.83	0.83	179

Figure 17: 随机森林模型交叉验证结果

下面对预测错误的样本进行进一步分析

对于随机森林模型，首先将交叉验证集上预测错误的样本筛选出来

```
val_rfc = pd.DataFrame(val_rfc, columns=['Predicted'])
X_val = X_val.reset_index(drop = True)
y_val = y_val.reset_index(drop = True)
y_val = y_val.to_frame()
y_val.columns = ['Truth']
X_val_concat = pd.concat([X_val, y_val, val_rfc], axis = 1)
bad_case = X_val_concat[X_val_concat.Truth != X_val_concat.Predicted]
```

发现如下样本预测错误

	female_adult	male_adult	Truth	Predicted
5	0	1	1.0	0.0
15	1	0	0.0	1.0
27	1	0	1.0	0.0
33	0	1	1.0	0.0
35	0	1	1.0	0.0
40	0	1	1.0	0.0
50	0	1	1.0	0.0
54	0	1	1.0	0.0
59	0	1	1.0	0.0
87	1	0	1.0	0.0
99	0	1	1.0	0.0
112	0	1	1.0	0.0
114	0	1	1.0	0.0
120	0	1	1.0	0.0
127	0	1	1.0	0.0
130	1	0	0.0	1.0
143	1	0	0.0	1.0
151	1	0	1.0	0.0
157	0	1	1.0	0.0
158	1	0	0.0	1.0
161	0	1	1.0	0.0

Figure 18: 交叉验证集上错误样本

分析发现，大部分预测错误均来自于有家庭成员陪同的成年遇难女性和幸存男性。针对以上两种特殊情况，增加'perishing_mother_wife' 和'surviving_father_husband' 两个特征。

首先对于'perishing_mother_wife'，获取所有符合条件的姓氏列表。

```
perishing_female_surnames = list(set(full_data[(full_data.person == 'female_adult') &
    (full_data.Survived == 0.0) &
    ((full_data.Parch > 0) | (full_data.SibSp > 0))]['surname'].values))
```

对于姓氏在 *perishing_female_surnames* 的乘客，将'perishing_mother_wife' 属性值设为 1。

```
def perishing_mother_wife(passenger):
    surname, Pclass, person = passenger
    return 1.0 if (surname in perishing_female_surnames) else 0.0
```

同样对于'surviving_father_husband'，仍获取所有符合条件的遇难男性的姓氏列表

```
surviving_male_surnames = list(set(full_data[(full_data.person == 'male_adult') &
    (full_data.Survived == 1.0) &
    ((full_data.Parch > 0) | (full_data.SibSp > 0))]['surname'].values))
```

对于姓氏在 *surviving_male_surnames* 的乘客，将'surviving_father_husband' 属性值设为 1。

```
def surviving_father_husband(passenger):
    surname, Pclass, person = passenger
    return 1.0 if (surname in surviving_male_surnames) else 0.0
```

添加以上两个属性后，重新检验随机森林模型在交叉验证集上的表现。

	precision	recall	f1-score	support
0.0	0.86	0.97	0.91	168
1.0	0.94	0.74	0.83	100
avg / total	0.89	0.88	0.88	268

Figure 19: 添加特征后交叉验证集上结果

添加以上两个属性后，随机森林模型在交叉验证集上的表现相较添加特征之前有所提升，准确率由 0.84 提升至 0.89，召回率由 0.83 提升至 0.88，f1 值也由 0.83 提升至 0.88。

分析还可以发现，预测错误的样本很多为来自一等舱的男性乘客，因此增加'surviving_first_class_male' 属性，标记幸存的来自一等舱的男性乘客。

```
def surviving_first_class_male(passenger):
    Pclass, Sex, Survived = passenger
    if (Pclass == 1 and Sex == 'male' and Survived == 1.0):
        return 1.0
    else:
        return 0.0
```

增加这一属性后，重新评估模型在交叉验证集上的属性，结果如下

	precision	recall	f1-score	support
0.0	0.92	0.96	0.94	110
1.0	0.94	0.87	0.90	69
avg / total	0.93	0.93	0.93	179

Figure 20: 添加特征后交叉验证集上结果

添加这一属性后，随机森林模型在交叉验证集上的表现相较之前进一步提升，准确率由 0.89 提升至 0.93，召回率由 0.88 提升至 0.93，f1 值也由 0.88 提升至 0.93。

5.1 创新点，难点总结

5.1.1 创新点

- 【姓名】将出现次数较少的称谓用同一标记替代
- 【姓名】将年轻女性的称谓统一改为 Girl
- 【年龄】用同一称谓的中位数填充缺失值
- 【年龄】填补缺失值后，将年龄按段划分
- 【客舱】只保留首字母
- 【家庭】将 Sibsp 和 Parch 合并，并按总人数划分为 4 个类别
- 【家庭】增加 FamilyId 属性，按姓氏划分乘客所属家庭，将家庭成员数小于 3 的设为-1
- 【性别与年龄】将这两个影响很大的属性结合生成 Person 属性
- 【one-hot】将 Person、Sex、Pclass、Title 四个属性进行 one-hot 编码
- 【预测错误数据】增加'perishing_mother_wife', 'surviving_father_husband', 'surviving_first_class_male' 将预测错误数据特殊标识

5.1.2 难点

分别使用随机森林、迭代决策树、逻辑回归模型进行预测，寻找最优参数，并进行交叉验证，最后根据模型在交叉集结果选择最优模型预测测试集的结果。模型调优的过程是难点之一，需要遍历候选参数以选择最优参数。同时在特征提取部分，需要根据交叉验证集上的错误结果不断对提取的特征进行调整，以得到测试集上最优的结果。

6 提交结果

经过提取特征和不断调参，最终的结果为 0.82296，排名 379，截图如下所示：

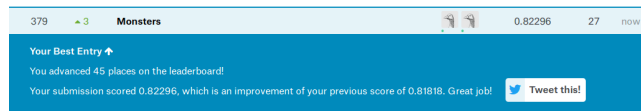


Figure 21: 最终结果及排名