深圳大学考试答题纸

(以论文、报告等形式考核专用)

二○一八～二○一九学年度第二学期

课程编号	2706019	课程名称	数据仓库与数据挖掘	主讲教师	陈小军	评分	

学　　号	1810272058	姓名	徐荣钦	专业年级	计算机技术　研究生一年级

教师评语：

项目名称：　基于泰坦尼克号生还预测的研究和分析

深圳大学课程项目报告

课程名称：数据挖掘导论

项目名称：基于泰坦尼克号生还预测的研究和分析

学　　院：计算机与软件学院

专　　业：计算机技术

任课教师：陈小军

报 告 人：徐荣钦　　学号：**1810272058**

提交时间：**2019 年 06 月 17 日**

教 务 处 制

目录

一、项目要求

1.1 研究背景

机器学习(Machine Learning, ML)是一门多领域交叉学科，涉及概率论、统计学、逼近论凸分析、算法复杂度理论等多门学科。专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构使之不断改善自身的性能。它是人工智能的核心，是使计算机具有智能的根本途径，其应用遍及人工智能的各个领域，它主要使用归纳、综合而不是演绎。主要的机器学习模型有三种：有监督学习、无监督学习和半监督学习。

数据挖掘（英语：Data mining），又译为资料探勘、数据采矿。它是数据库知识发现（英语：Knowledge-Discovery in Databases，简称：KDD)中的一个步骤。数据挖掘一般是指从大量的数据中通过算法搜索隐藏于其中信息的过程。数据挖掘通常与计算机科学有关，并通过统计、在线分析处理、情报检索、机器学习、专家系统（依靠过去的经验法则）和模式识别等诸多方法来实现上述目标。

简单地说就是，机器学习和数据库是数据挖掘的基石。机器学习致力于研究如何通过计算的手段，利用经验来改善自身的性能，而这里的经验指的就是数据。我们研究出了一系列在理论层面上可行的算法，即（learning algorithm），然后将数据集应用到这些算法上，产生学习模型，这个学习模型可以应用到新的数据集上，对新数据集的一些情况进行预测和判断。

1.2 研究选题

项目题目可以从 DataCastle，天池，和 kaggle 比赛网站上选择。本项目主要从 Kaggle 网站中进行选择，帅选条件为“Getting Started”即为正在进行中的比赛。如下图 1-1 所示。经过综合考量和个人兴趣，我选择了其中的“Titanic: Machine Learning for Disaster”，即泰坦尼克号的生还预测，作为本项目的研究题目。它要求参赛选手通过训练数据集分析出什么类型的人更可能幸存，并预测出测试数据集中的所有乘客是否生还。

泰坦尼克号的沉没是历史上最震惊世界的沉船事故之一。1912 年 4 月 15 日，泰坦尼克号在处女航中撞上冰山沉没，2224 名乘客和船员中 1502 人遇难。这一耸人听闻的悲剧震惊了国际社会，并导致了更好的船舶安全条例。沉船造成了如此巨大的人员伤亡的原因之一

是没有足够的救生艇来容纳乘客和船员。虽然在沉船事件中幸存下来也有一些运气的因素，但是有些人比其他人更有可能幸存下来，比如妇女、儿童和上层阶级等。

该选题主要是一个二分类问题，因此，我们通过机器学习的经典算法进行分类预测问题。本次实验的平台是搭建在 Windows 10 （Intel I7，16G 内存） + Pycharm 2018.02，编程语言为 Python 3.7 以及 numpy / sklearn / pandas 等常用的数据操作包和机器学习模型包等等。

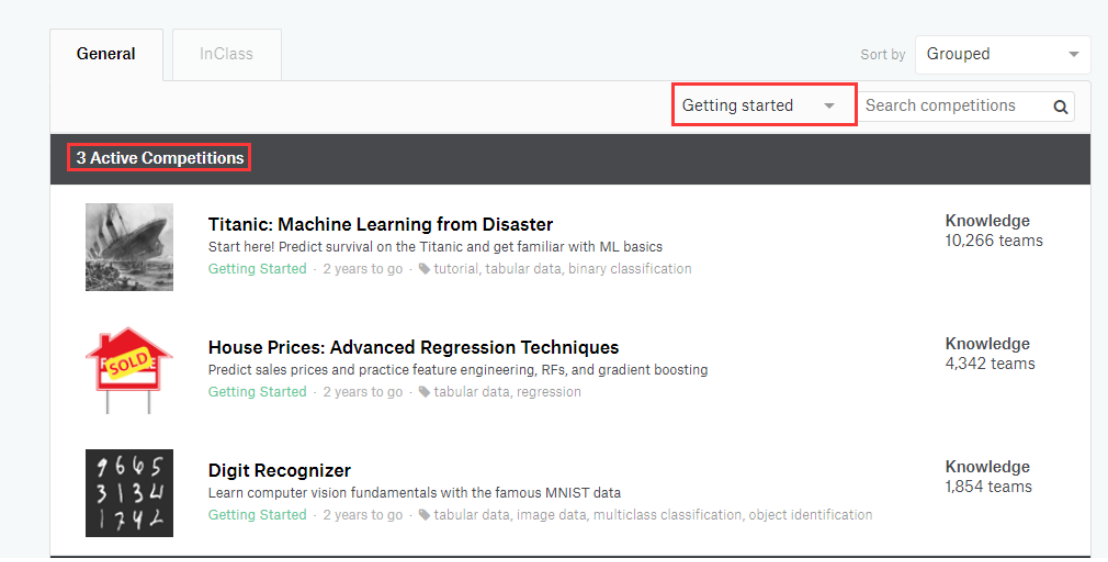


图 1-1： Kaggle 平台中的几道正在进行的比赛题目

1.3 任务描述

图 1-2 展示了泰坦尼克号幸存预测问题的问题描述。在泰坦尼克号豪华游艇倒了，大家都惊恐逃生，可是救生艇的数量有限，无法人人都有，副船长发话了“lady and kid first!”，所以是否获救其实并非随机，而是基于一些背景有 rank 先后的。该选题提供了 CSV 格式的训练集 train.csv 以及测试集 test.csv。训练集和测试集都包含每个乘客的 PassengerId、Sex、Name、Age 等信息。唯一不同的是，训练集 train.csv 中包含了是否存活的标签 Survived（1 表示幸存，0 表示死亡）。训练集 train.csv 是一些乘客的个人信息以及存活状况，要尝试根据它生成合适的模型并预测在测试集 test.csv 中的其他人的存活状况。这是一个二分类问题，是经典的机器学习模型如决策树、Logistic Regression、随机森林、KNN 等模型所能处理的范畴。模型的预测结果保存为 CSV 格式并且通过图 1-2 中的“Submit predictions”提交到系统进行在线判题。我这里保存为 submission.csv，注意，该文件的格式必须是 csv 且只包含以下两列：

1)、PassengerId（按任意顺序排序）

2)、Survived（包含二进制预测：1 表示幸存，0 表示死亡）

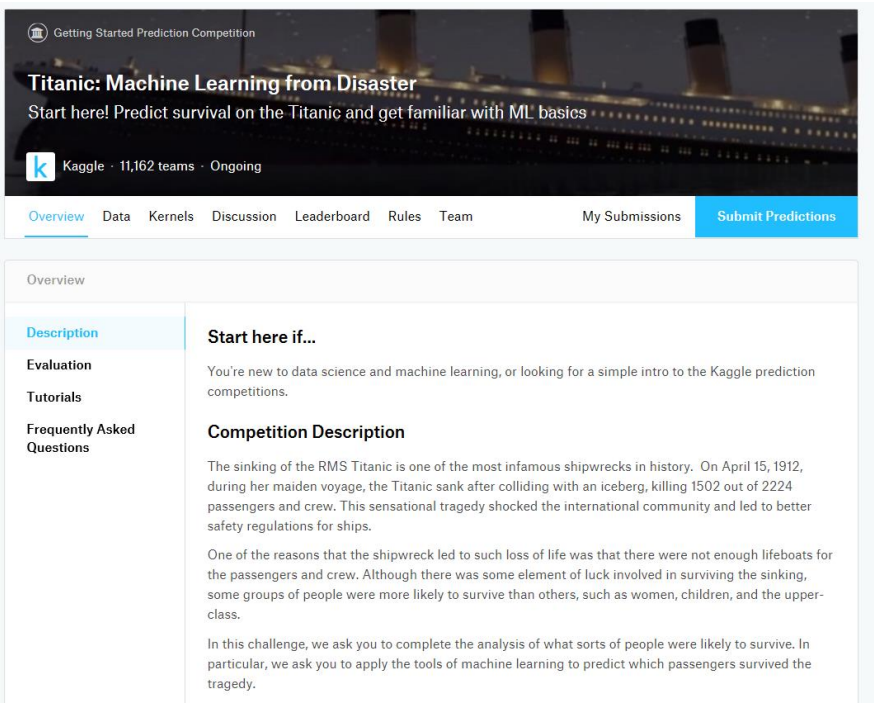


图 1-2：泰坦尼克号幸存预测的问题描述。

1.4 数据集描述

训练集 train.csv 用于构建机器学习的模型，每位乘客都带着标签（1 代表幸存，0 代表死亡），共计 891 条数据。 测试集（test.csv）是没有对乘客进行标注幸存或死亡标签的，主要用来测试，并且做出预测，共计 419 条数据。

训练集/测试集的数据字典（即每一个属性字段的表示和具体含义）如表 1-1 所示。

表 1-1：训练集/测试集的数据字典，属性的字段及其表示意义

属性字段	意义	注释
PassengerId	乘客ID	
Survived	生还与否	0表示死亡，1表示生还
Pclass	乘客等级(1/2/3等舱位)	1/2/3分别表示高级仓位、中级仓位、低级仓位
Name	乘客姓名	
Sex	性别	
Age	年龄	
SibSp	堂兄弟/妹个数	
Parch	父母与小孩个数	
Ticket	船票信息	主要是由字母和阿拉伯数字组成的字符串
Fare	票价	浮点型数据
Cabin	客舱	主要是由字母和阿拉伯数字组成的字符串
Embarked	登船港口	C = Cherbourg, Q = Queenstown, S = Southampton

二、项目知识点

项目的知识点主要有以下几个，具体将会在下面的“实验过程”部分进行。

1. 数据预处理；
2. 数据可视化；
3. 特征工程；
4. 模型选择；
5. 实验结果的评价；

三、实验过程

在这一部分中，我们先对训练集 train.csv 和 test.csv 读进来，然后观察它们的某些属性字段是否数据缺失。如果存在数据缺失，则需要根据属性字段的数据特征进行缺失值填补。此外，在机器学习中，离群点（Outlier）又称为异常点，会影响预测模型的稳定性和泛化性能。因此，我们需要对训练集的数据进行离群点检测和剔除离群点。

3.1 数据预处理

3.1.1 数据集观察

首先，我们利用读入训练集 train.csv 和测试集 test.csv。如图 3-2 所示，是训练集的前 5 行数据以及每个数据样本的特征属性信息。此外，图 3-3 展示了每个属性字段的数值类型和数目，我们可以从图 3-3 看出——train、test 数据集中的 Age、Cabin 发生了大量数据缺失，train 数据集中 embarked 有少量缺失，test 数据集中 Fare 有少量缺失。图 3-1 是具体的代码。

```
# 加载训练集test和测试集test为dataFrame格式
train = pd.read_csv('../input/train.csv')
test = pd.read_csv('../input/test.csv')

# 显示train的列名和具体数值信息等等
print('---------------head of train set------------')
print(train.head())

# 显示每一列的详细信息，包括数值类型、数值总数、文件大小等
print('---------------info of train set------------')
print(train.info())
print('---------------describe of train set------------')
print(train.describe())
```

图 3-1 读入数据和显示数据预览的代码

代码分析和说明：read_csv(“CSV 文件路径”)函数读入训练集和测试集，分别为 DataFrame 格式的 train、test 变量，然后利用 head()函数预览测试集的前 5 行的数据。此外，info()函数统计了每个属性字段的数值类型和数目。

```
--------------head of train set------------
   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
```

图 3-2：训练集的数据预览。

```
----------------info of train set------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId     891 non-null int64
Survived        891 non-null int64
Pclass          891 non-null int64
Name            891 non-null object
Sex             891 non-null object
Age             714 non-null float64
SibSp           891 non-null int64
Parch           891 non-null int64
Ticket          891 non-null object
Fare            891 non-null float64
Cabin           204 non-null object
Embarked        889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
None
```

```
============= Test Set Info =================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
PassengerId     418 non-null int64
Pclass          418 non-null int64
Name            418 non-null object
Sex             418 non-null object
Age             332 non-null float64
SibSp           418 non-null int64
Parch           418 non-null int64
Ticket          418 non-null object
Fare            417 non-null float64
Cabin           91 non-null object
Embarked        418 non-null object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
None
```

图 3-3：train、test 数据集中的数据属性类型和个数等具体信息

查看数据表的中位数和众数，如图 3-4 所示，使用 median()方法统计数据表的中位数情况，如下图 3-5 使用 mode()方法统计每个属性的众数情况。此外，查看属性的四分位极差（IQR），以及变异系数 （CV），如图 3-6 所示，其中，四分位极差（IQR）为：

$$IOR = X_{75\%} - X_{25\%}.$$

变异系数（CV）为：

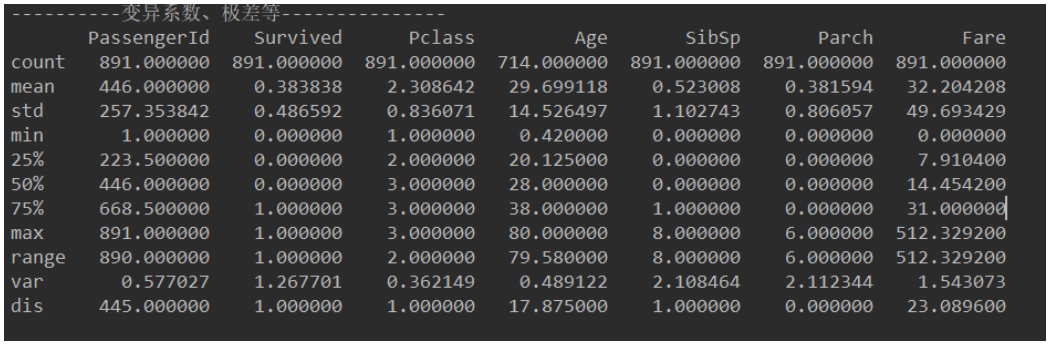
$$CV = std - \bar{X}$$

```
PassengerId    446.0000
Survived         0.0000
Pclass           3.0000
Age             28.0000
SibSp            0.0000
Parch            0.0000
Fare            14.4542
dtype: float64
```

图 3-4：train 的中位数

```
   PassengerId  Survived  Pclass                 Name   Sex   Age  SibSp  Parch  Ticket  Fare    Cabin Embarked
0            1       0.0     3.0  Abbing, Mr. Anthony  male  24.0    0.0    0.0    1601  8.05  B96 B98        S
```

图 3-5：train 的众数



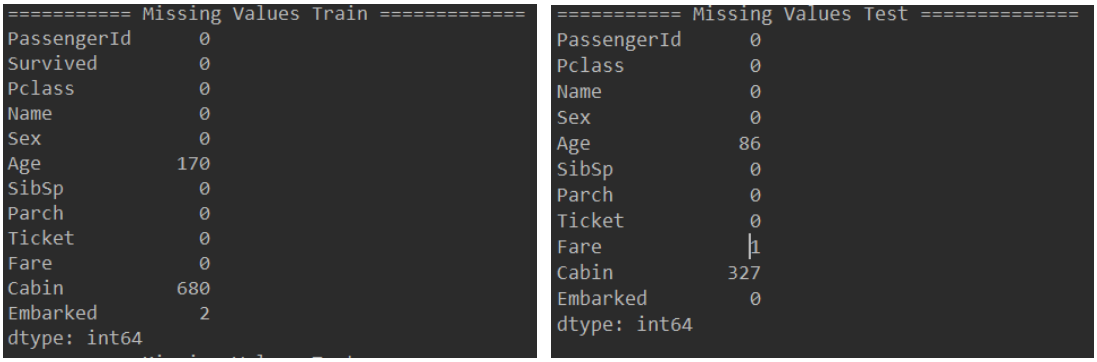
----------变异系数、极差等----------

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200
range	890.000000	1.000000	2.000000	79.580000	8.000000	6.000000	512.329200
var	0.577027	1.267701	0.362149	0.489122	2.108464	2.112344	1.543073
dis	445.000000	1.000000	1.000000	17.875000	1.000000	0.000000	23.089600

图 3-6：train 的极差和变异系数等等

3.1.2 数据缺失值填补

由 3.1.1 的分析可知，train 和 test 都存在着某些属性的数据缺失。对数据进行分析的时候要注意其中是否有缺失值。一些机器学习算法能够处理缺失值，比如神经网络，一些则不能。如图 3-7 所示，展示了 train 和 test 每个属性值缺失的值的数量。



========== Missing Values Train ==========

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

dtype: int64

========== Missing Values Test ==========

PassengerId	0
Pclass	0
Name	0
Sex	0
Age	86
SibSp	0
Parch	0
Ticket	0
Fare	1
Cabin	327
Embarked	0

dtype: int64

图 3-7：train 和 test 的缺失值数目

对于缺失值的处理，本人在 CSDN 博客的论坛上进行了资料整理和总结，一般有以下几种处理方法 [3]：

（1）、如果数据集很多，但有很少的缺失值，可以删掉带缺失值的行；

（2）、如果该属性相对学习来说不是很重要，可以对缺失值赋**均值或者众数**。比如在哪儿上船 Embarked 这一属性（共有三个上船地点），缺失两个，可以用众数赋值。

（3）、对于标称属性，可以赋一个代表缺失的值，比如‘U0’。因为缺失本身也可能代表着一些隐含信息。比如船舱号 Cabin 这一属性，缺失可能代表并没有船舱。

（4）、使用**回归随机森林**等模型来预测缺失属性的值。因为 Age 在该数据集里是一个相当重要的特征（由下面的数据可视化分析可知），所以保证一定的缺失值填充准确率是非常重要的，对结果也会产生较大影响。这里使用随机森林预测模型进行 Age 数据属性的填补。

在这一部分，根据不同缺失属性字段的数据特性，我们选择不同的缺失值填补方法。下面将展示 Cabin、Age、Embarked 等不同字段属性的缺失值填补过程过关键代码。**特别注意，若 train 与 test 都在某一字段有缺失值，则应该对 train\test 作相同的缺失值填补工作**。

1）、Cabin 缺失值填补。代码如下所示：

```
# 处理train和test data中的'Cabin'缺失值
def fix_cabin(data):
    data["Cabin"] = pd.Series([i[0] if not pd.isnull(i) else 'X' for i in data['Cabin']])
fix_cabin(train)
fix_cabin(test)
```

代码说明和分析：有些方法[4, 5]把无缺失的 Cabin 置为 1，有缺失则置为 0。但这里我的思路是把 Cabin 缺失值置为'X'。 因为，经过分析（如下面的数据可视化分析），不同字母开头的 Cabin，乘客的生还率都有所不同。

2）、Age 缺失值填补

```
from sklearn.ensemble import RandomForestRegressor

### 使用 RandomForestClassifier 填补缺失的年龄属性
def set_missing_ages(df):
    # 把已有的数值型特征取出来丢进Random Forest Regressor中
    age_df = df[['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

    # 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
    df.loc[(df.Age.isnull()), 'Age'] = predictedAges

    return df, rfr
```

代码说明和分析：一般情况下，会使用数据完整的条目作为模型的训练集，以此来预测缺失值。对于当前的这个数据，可以使用随机森林来预测也可以使用线性回归预测。这里使用随机森林预测模型 RandomForestRegressor，选取数据集中的数值属性作为特征（因为 sklearn 的模型只能处理数值属性，所以这里先仅选取数值特征，但在实际的应用中需要将非数值特征转换为数值特征）。

3）、Embarked、Fare

由于 Embarked 和 Fare 的缺失值仅为 1-2 个，这里用众数填充。

进行以上缺失值填补之后，我们再来看看填补所有缺失值之后的 Dataframe，如图 3-8 所示，

可以看到，没有缺失值了。

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            891 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          891 non-null object
Embarked       891 non-null object
```

图 3-8：填补所有缺失值后的 train。

3.1.3 离群点剔除

离群点检测也称异常检测和例外挖掘。孤立点是一个明显偏离与其他数据点的对象,它就像是由一个完全不同的机制生成的数据点一样[2]。离群点检测是数据挖掘中重要的一部分，它的任务是发现与大部分其他对象显著不同的对象。

在大规模数据集中，由于噪声、扰动、采样过程误差等等原因，会出现一些数据点偏移整个数据集。假想整个数据集由某未知分布生成，则这些点可以看做该未知分布下的噪声采样。在可视化情况下，这些点显著偏移了数据集的点群，故称为离群点。而众多机器学习算法对数据分布都存在着一定的假设或期待数据集较为“规整”。因此在数据挖掘中，常需要在预处理中去除该类点，让算法能更好地发现“正常”数据间存在的关系。Tukey Method 是一类常用方法

即使很多 CSDN 上的经典的泰坦尼克号幸存预测模型[4,5]没有用到 Outlier Detection，这里我们进行 Outlier Detection 进行数据清洗。这里感谢 Kaggle Kernel 中的博主 [6].

```python
# Outlier detection函数
def detect_outliers(df, n, features):
    """
    通过tukey方法检测离群值，输入是一个包含多个特征的dataFrame对象，
    返回一个包含n个以上异常值的观察值对应的行索引列表。
    """
    outlier_indices = []

    # iterate over features(columns)
    for col in features:
        # 1st quartile (25%)
        Q1 = np.percentile(df[col], 25)
        # 3rd quartile (75%)
        Q3 = np.percentile(df[col], 75)
        # Interquartile range (IQR)
        IQR = Q3 - Q1

        # outlier step
        outlier_step = 1.5 * IQR

        # Determine a list of indices of outliers for feature col
        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 + outlier_step)].index

        # append the found outlier indices for col to the list of outlier indices
        outlier_indices.extend(outlier_list_col)

    # select observations containing more than 2 outliers
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list(k for k, v in outlier_indices.items() if v > n)

    return multiple_outliers
```

代码说明：这里通过 Tukey 方法进行离群点检测。该方法返回 Dataframe 对象中疑似离群点的索引。这里不详细说明 Tukey 方法的原理。

如图 3-9 所示，展示了进行离群点检测后的 train.info()。我们可以看到，去除了 10 个离群点。

```
----------------after outlier detection----------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 881 entries, 0 to 880
Data columns (total 12 columns):
PassengerId    881 non-null int64
Survived       881 non-null int64
Pclass         881 non-null int64
Name           881 non-null object
Sex            881 non-null object
Age            881 non-null float64
SibSp          881 non-null int64
Parch          881 non-null int64
Ticket         881 non-null object
Fare           881 non-null float64
Cabin          881 non-null object
Embarked       881 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 82.7+ KB
None
```

图 3-9：离群点检测后的 train。

3.2 数据可视化

在这一部分，我们以生还率为观察核心，观察不同的数据特征属性，例如 Age、Sex、Name、Cabin 等不同情况下的生还率。目的是探索和分析训练集中数据之间的关系和所遵循的规律，这将为后续的特征工程以及模型选择有很大的帮助作用。

3.2.1 数值型变量之间的相关性

在这一部分，我们首先观察每个特征之间的相关性，这里，我采用皮尔逊（Pearson） 相关系数进行计算，假设有两个特征变量 $\boldsymbol{x}$ 和 $\boldsymbol{y}$，且 n 为维度，则 Pearson 的公式为 [7]：

$$r=\frac{\sum_{i=1}^{n}(x_i-\overline{x})(y_i-\overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\overline{x})^2}\cdot\sqrt{\sum_{i=1}^{n}(y_i-\overline{y})^2}}$$

如图 3-10 展示了数值型属性之间的相关系数矩阵，可以看到，Pclass、Age 与 Survived=1 呈负相关性，说明这两者的值越小，则越有可能获救！

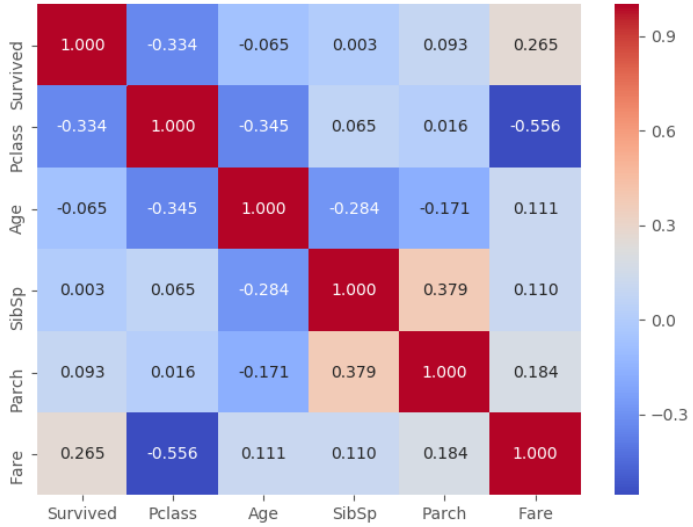


图 3-10：数值型属性之间的相关系数矩阵。

3.2.2 不同数据属性与 Survived 之间的关系

3.2.2.1 SibSp vs Survived

如图 3-11 所示，展示了不同数目的 SibSp（兄弟姐妹/配偶）下的生还率。很明显，有更多兄弟姐妹/配偶的人存活的几率较小。孤独的人生存的机会也相对较低。

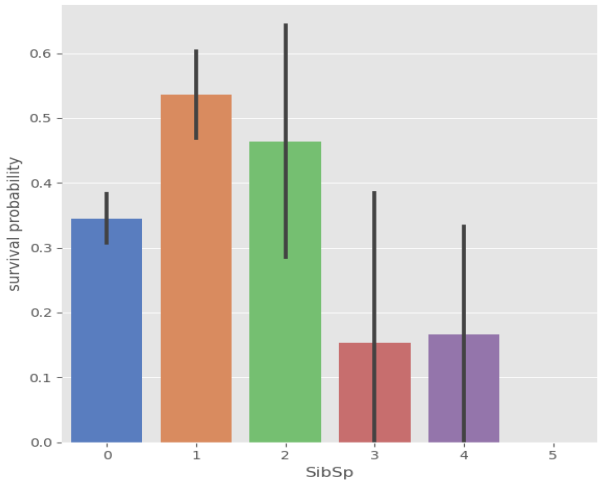


图 3-11：SibSp vs Survived

3.2.2.2 Parch vs Survived

如图 3-12 所示，展示了不同数目的 Parch（孩子/父母）下的生还率。同样，有更多孩子/父母在船上的人没有活下来，独自旅行的人也没有活下来。细心观察可以发现，Parch==3 的情况下生还率最高，也许，这个数字使家庭成员之间相互帮助从而最大化获救率的最佳大小。

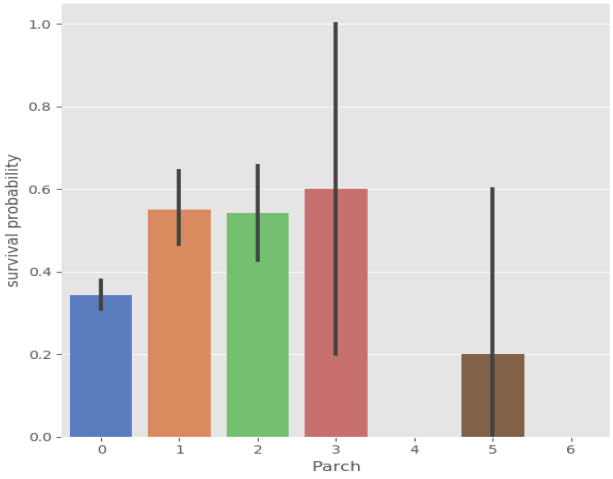


图 3-12：Parch vs Survived

3.2.2.3 Age vs Survived

如图 3-13 所示，展示了生还和死亡两种情况下的年龄分布。我们可以清楚地看到，年轻人比年龄大的活得更多，但是也有更多的年轻人失去了生命，这意味着船上有更多的年轻

人。此外，图 3-14 直观地看出，在获救的年龄分布中（蓝色的 Survived），在 0-10 岁中有一个波峰，这说明了**年龄小的小孩子优先登上获救船**！

图 3-13：生还和死亡两种情况下的年龄分布（1）。

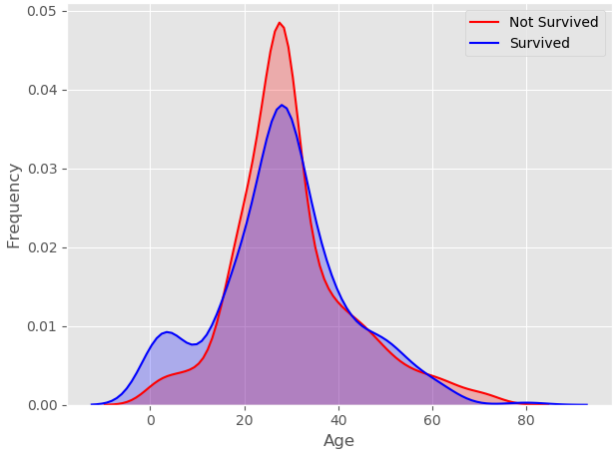


图 3-14：生还和死亡两种情况下的年龄分布（2）。

3.2.2.4 Sex vs Survived

如图 3-15 所示，展示了不同性别下的生还率。我们可以看到，与男性相比，**女性存活下来的人数更多，这只能意味着女性和儿童更倾向于乘坐救生艇**。

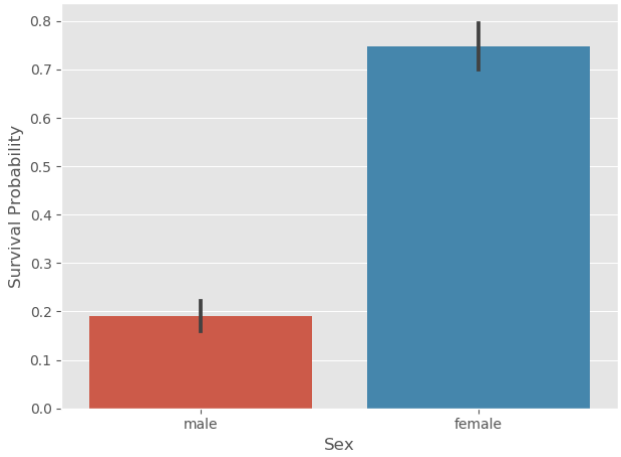


图 3-15：不同性别下的生还率。

3.2.2.5 Pclass vs Survived

图 3-16 展示了不同乘客等级的生还率。我们可以看到，等级越高的乘客，优先被营救。

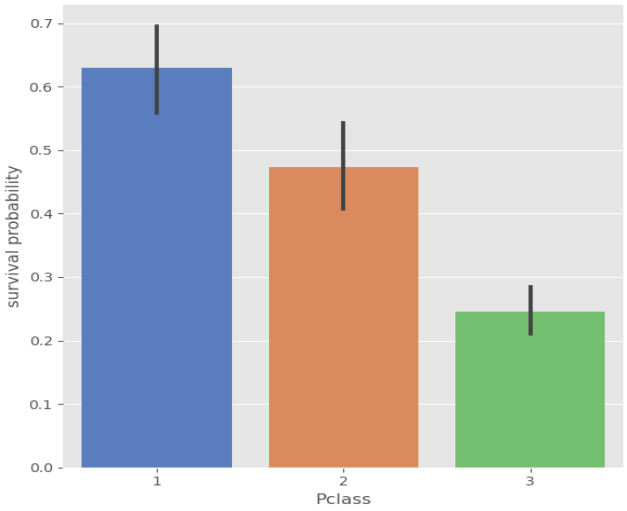


图 3-16：不同乘客等级的生还率。

3.2.2.6 Embarked vs Survived

图 3-17 展示了来自不同地方的乘客的生还率。在这里，与昆士敦（Queenstown，Q）和南安普敦（Southamptom，S）相比，来自切尔堡（Cherbourg，C）的乘客存活率最高。

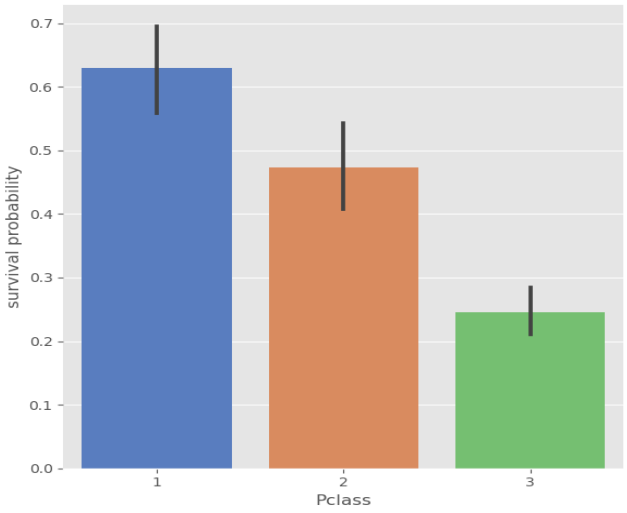


图 3-17：来自不同地方的乘客的生还率。

3.2.2.7 Pclass vs Embarked on Survived

图 3-18 的三个子图展示了来自三个不同地方的不同等级的乘客的数目。结合图 3-17 和图 3-18，我们发现：大多数来自 Southamptom 的人属于 3 级，这就是为什么他们的生存机会最低的原因。来自 Cherbourg 的人非常少，但大多数是高级乘客，而 Queenstown 的乘客大多是低级乘客。

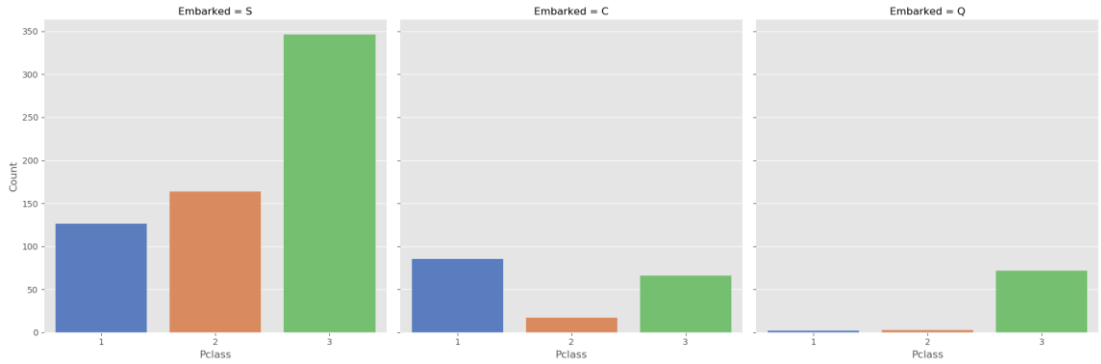


图 3-18：来自三个不同地方的不同等级的乘客的数目。

3.2.2.8 Cabin vs Survived

图 3-19 展示了不同字母开头的仓位 Cabin 的数目。图 3-20 展示了不同字母开头的仓位 Cabin 的生还率比较。因此，**我们不能因为 Cabin 缺失值太多而直接把 Cabin 丢弃！**要把它作为一个特征加入到后续的特征工程和模型选择中。

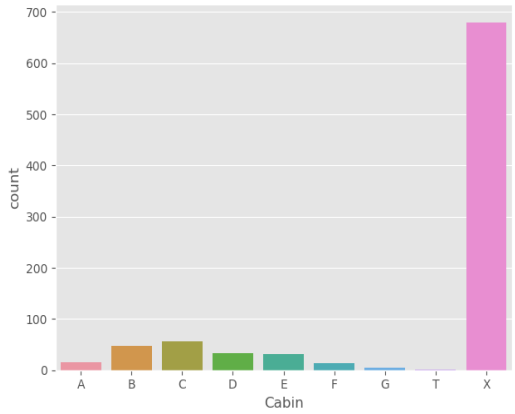


图 3-19：不同字母开头的 Cabin 的数目。

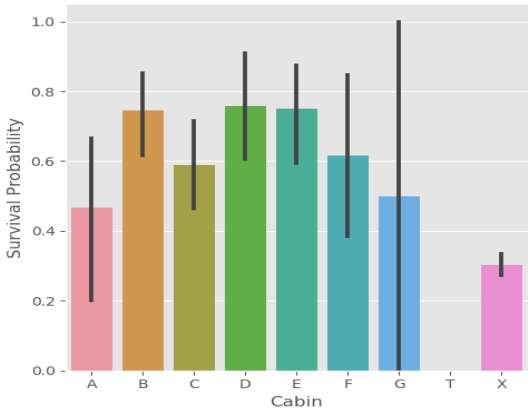


图 3-20：不同字母开头的 Cabin 的生还率。

3.3 特征工程

在这一部分中，我们对训练集 train 和测试集 test 进行特征工程（feature engineering）。特征工程是利用数据的领域知识来创建使机器学习算法工作的特征的过程。例如，我们从 Name 中提取称呼头衔 Title，并观察 Title 与生还率等数据特性的关系，以决定是否把新生成的隐藏特征 Title 加入到训练集 train 中。此外，机器学习算法只能处理数值型数据，我们需要利用 one-hot 等编码把非数值型数据（例如 Sex、Title、Pclass 等）转化为数值型数据。

在特征工程的阶段，注意，我们需要将训练集 train 和测试集 test 作相同的特征工程变化，使得二者具有相同的数据类型和数据分布，特征的性质是一样的。对数据进行特征工程，也就是从各项参数中提取出对输出结果有或大或小的影响的特征，将这些特征作为训练模型的依据。

3.3.1 Name 特征属性的特征工程

Name 字段表示乘客的姓名，包括称呼（例如，Mrs、Miss、Mr、Master 等）。Name 为非数值型数据，我们不能直接把这一列去掉，因为它包含着很多有用的信息（请看下面的分析）。因此，我们要对 Name 字段分析，挖掘其中蕴含的信息，派生出新的特征如下面提到的 Title 特征。

首先，如下所示，我们提取了不同乘客的称呼 Title 并且进行分析，代码如下所示。乘客姓名中包含了一些特定的称谓如“Mr”、“Miss”、“Dr”、“Major”等，这些称谓不仅可以反映出乘客的性别、年龄、是否结婚、甚至社会地位，这些都是影响其生存与否的重要因素，

所以可以从乘客姓名中提出一个隐含的“Title”特征。

```
# 从Name特征属性提取称呼(Title)
def get_title(data):
    data_title = [i.split(",")[1].split(".")[0].strip() for i in data["Name"]]# 提取并且构建称呼Title
    data["Title"] = pd.Series(data_title)# 把'Title'作为一个列添加到DataFrame对象data中

get_title(train)
get_title(test)
print(train['Title'].head())
print(test['Title'].head())
```

代码说明和分析：自定义 get_title 获取不同乘客的称呼 Title（例如，Mrs、Miss、Mr、Master 等），提取当中的称谓，然后对称谓进行匹配。并且把 Title 作为一列加入到 Dataframe 中。

此外，图 3-21 展示了把 Title 作为一列特征属性字段加入到 train 和 test 后的信息。图 3-22 展示了不同 Title 的数目，我们可以发现，Mr 即男性的称呼比较多。

```
0      Mr
1     Mrs
2    Miss
3     Mrs
4      Mr
Name: Title, dtype: object
0     Mr
1    Mrs
2     Mr
3     Mr
4    Mrs
Name: Title, dtype: object
```

图 3-21：train 和 test 的 Title 特征属性字段。

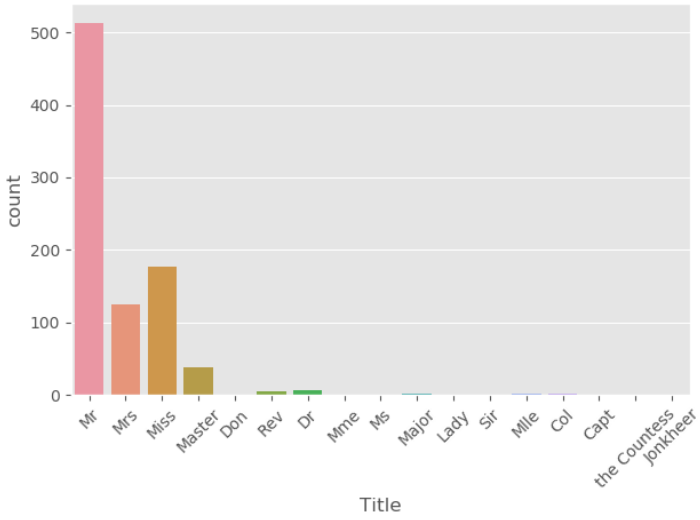


图 3-22：不同 Title 的数目

接着，我们把某一的 Title 集合映射到另外的 Title 集合，如下代码所示：

```
newtitles={
    "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"}


train['Title']=train.Title.map(newtitles)
test['Title']=test.Title.map(newtitles)
```

代码说明：称谓 Title 映射的字典以及转换（利用 map 函数）。把各种不同的 Title 分为总共六大类，分别是：Officer、 Royalty、Mr、Mrs、Miss、Master。

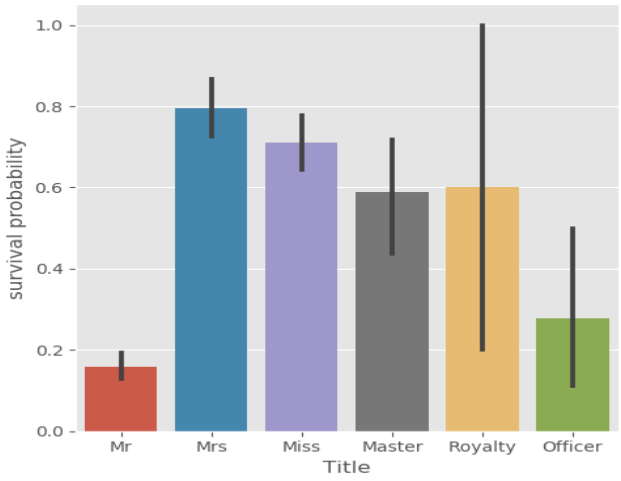


图 3-23：称谓 Title 映射后的不同 Title 的生还率。

此外，图 3-23 展示了称谓 Title 映射后的不同 Title 的生还率，我们可以看到，女士（即 Mrs 和 Miss）的生还率比男士 Mr 高，说明了女士优先营救上船的原则。此外，一些贵族（即 Royalty）和知识分子（即 Master）明显比普通的企业职工 Officer 生还率要高，说明了贵族和知识分子优先被营救！

最后，把 Name 字段丢弃，如下代码所示：

```
# ------------------------Name-------------------------- #
# Drop Name Column
train.drop(labels = ["Name"], axis = 1, inplace = True)
test.drop(labels=["Name"], axis=1, inplace=True)
```

3.3.2 Ticket 特征属性的特征工程

不少 Kaggle Kernels 或者 CSDN 博客上都会直接把 Ticket 直接丢弃，这里我分析了不同长度的 Ticket 字符串的生还率是否有差异。如下代码所示：

```
# ---------------------Ticket------------------------- #
train['Ticket2'] = train.Ticket.apply(lambda x: len(x))
test['Ticket2'] = test.Ticket.apply(lambda x: len(x))
sns.barplot('Ticket2','Survived',data=train)
plt.show()
```

代码说明和分析：这里计算每个乘客 Ticket 字段的字符串长度并且而作为一个新的列 Ticket2 加入到 Datframe 对象 train 和 test 中。

如图 3-24 所示，展示了不同船票长度 Ticket2 下的生还率。可以看到，船票编号长度为 5 和 8 的乘客，生还率明显高于其他的乘客。

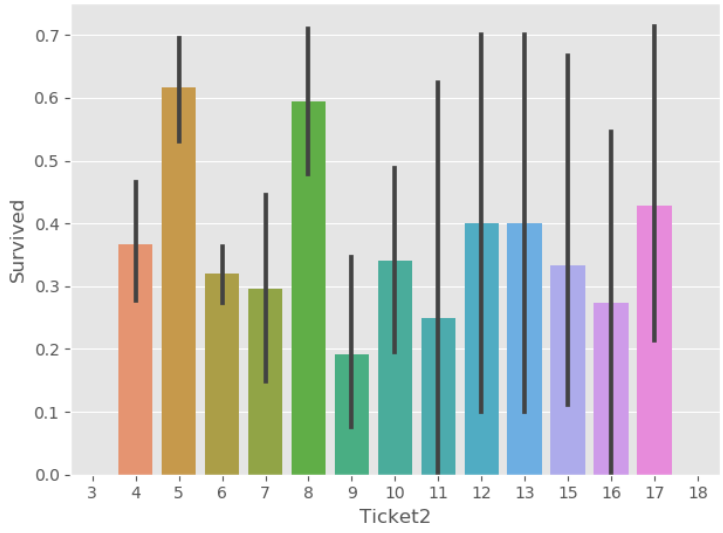


图 3-24：不同船票长度 Ticket2 下的生还率。

3.3.3 SibSp 和 Parch 特征属性的特征工程

我们在这里探索不同的家庭大小（Family Size，简写为 Fsize）与生还率的关系。很明显，Fisze = SibSp + Parch +1。下面代码展示了计算 Fsize 并且把 Fsize 作为一个新的特征属性加到 train 和 test 中，接着，再把 Fisze 按照大小进一步划分为 Single、SmallF、MedF、LargeF

四种。图 3-26 展示了在处理 Fsize 后的 train 的详细信息。

```
# ------------------Family Size-------------------- #
# 新的特征向量family size，家庭成员总数（包括本人）
train["Fsize"] = train["SibSp"] + train["Parch"] + 1
test["Fsize"] = test["SibSp"] + train["Parch"] + 1

g = sns.catplot(x="Fsize",y="Survived",data=train, kind='bar')
g = g.set_ylabels("Survival Probability")
plt.show()

# Create new feature of family size
def create_fsize(data):
    data['Single'] = data['Fsize'].map(lambda s: 1 if s == 1 else 0)
    data['SmallF'] = data['Fsize'].map(lambda s: 1 if  s == 2  else 0)
    data['MedF'] = data['Fsize'].map(lambda s: 1 if 3 <= s <= 4 else 0)
    data['LargeF'] = data['Fsize'].map(lambda s: 1 if s >= 5 else 0)

create_fsize(train)
create_fsize(test)
```

如图 3-25 所示，展示了不同家庭大小 Fsize 的生还率统计，我们可以发现，家庭大小为 3 和 4 的人生还率最高，此外，家庭大小为 5 和 6 的人获救率最小！

正如我们所看到的，大家庭和孤独的乘客生存的机会最少！图 4-7 更加详细地展示了这种情况。

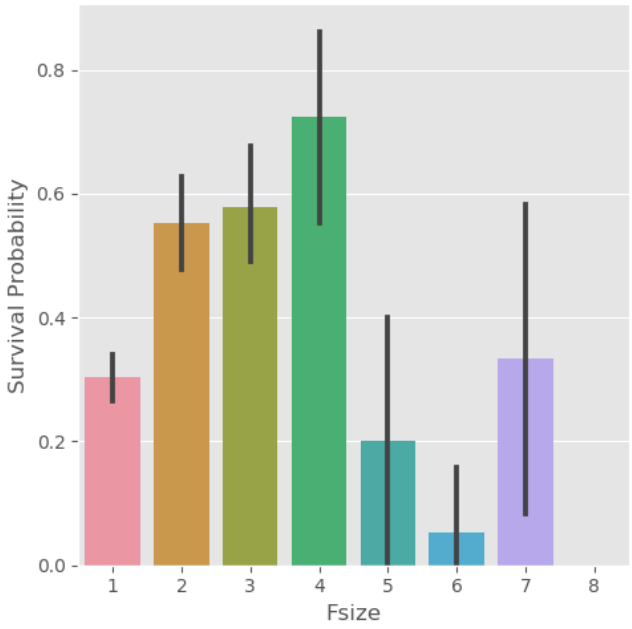


图 3-25：不同家庭大小 Fsize 的生还率统计。

```
----------train head after Fsize settings---------
   PassengerId  Survived  Pclass     Sex   Age  SibSp  Parch            Ticket     Fare Cabin Embarked  Title  Ticket2  Fsize  Single  SmallF  MedF  LargeF
0            1         0       3    male  22.0      1      0         A/5 21171   7.2500     X        S      2        9      2       0       1     0       0
1            2         1       1  female  38.0      1      0          PC 17599  71.2833     C        C      1        8      2       0       1     0       0
2            3         1       3  female  26.0      0      0  STON/O2. 3101282   7.9250     X        S      1       16      1       1       0     0       0
3            4         1       1  female  35.0      1      0            113803  53.1000     C        S      1        6      2       0       1     0       0
4            5         0       3    male  35.0      0      0            373450   8.0500     X        S      2        6      1       1       0     0       0
          train head after Embarked / Cabin / Pclass
```

图 3-26：在处理家庭大小特征属性 Fsize 后的 train。

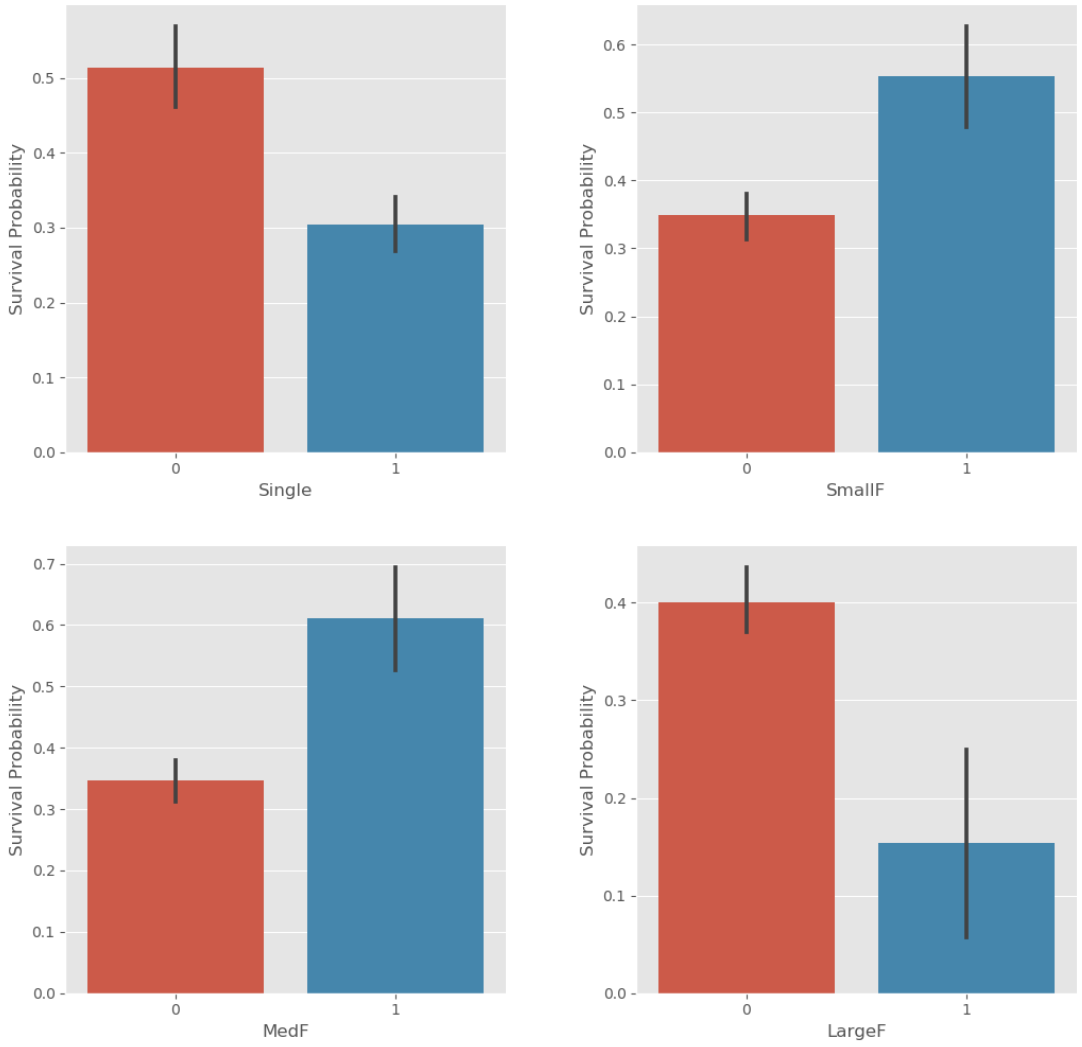


图 3-27：不同家庭大小下（Single、SmallF、MedF、LargeF）乘客的生还率比较。

3.3.4 Cabin 和 Parch 特征属性的特征工程

由上面数据描述和分析部分可知，Cabin 有较多的缺失且缺失的被置为“X”。此外，Pclass 会较大地影响到乘客的生还率，因此，这里利用 ont-hot 对 Pclass 进行特征化。关键代码如下所示：

```
# --------------------Cabin------------------------ #
train['Cabin'] = train['Cabin'].map({'X': 1, 'C': 2, 'B': 3, 'D': 4, 'E': 5, 'A': 6, 'F': 7, 'G': 8, 'T': 9})
test['Cabin'] = test['Cabin'].map({'X': 1, 'C': 2, 'B': 3, 'D': 4, 'E': 5, 'A': 6, 'F': 7, 'G': 8, 'T': 9})

# --------------------Pclass------------------------ #
train["Pclass"] = train["Pclass"].astype("category")
train = pd.get_dummies(train, columns = ["Pclass"],prefix="Pc")
test["Pclass"] = test["Pclass"].astype("category")
test = pd.get_dummies(test, columns = ["Pclass"],prefix="Pc")
```

代码说明和分析：这里主要通过 map 函数把 Cabin 由单字母映射为数值 1-9。此外，通过 pandas 中的 get_dummies 函数把 Pclass 以 one-hot 形式特征化！转化为 Pclas_1、Pclass_2、Pclass3

共 3 列并且加到 train 和 test 中。

3.3.5 Embarked 和 Title 特征属性的特征工程

Title 属性已经在前面进行了映射为 6 类 Title，Embarked 特征属性也是由 S\Q\C 三种字母表示不同的地点，这里也是通过 get_dummies 函数把 Embarked 和 Title 特征化为数值型数据。关键代码如下所示。

```
# --------------------Embarked & Title---------------------- #
def encode_train_embark(data):
    '''把特征属性Embarked转化为数值向量并且加入到DataFrame data中 '''
    data = pd.get_dummies(data, columns=["Title"])# get_dummies 实现one hot encode（即 0 0 1; 0 1 0; 1 0 0;...）
    data = pd.get_dummies(data, columns=["Embarked"], prefix="Em")

encode_train_embark(train)
encode_train_embark(test)
```

图 3-28：在处理家庭大小特征属性 Fsize 后的 train。

3.3.6 Sex 特征属性的特征工程

ML 算法只接受数值型输入，把非数值型或类别性质的特征属性如 Sex 等转化为数值型因此，这里把 Sex 二值化，关键代码如下所示，

```
from sklearn.preprocessing import LabelEncoder
def impute_cats(df):
    '''ML算法只接受数值型输入，把非数值型或类别性质的特征属性如Sex等转化为数值型'''
    # Find the columns of object type along with their column index
    object_cols = list(df.select_dtypes(exclude=[np.number]).columns)
    object_cols_ind = []
    for col in object_cols:
        object_cols_ind.append(df.columns.get_loc(col))

    # Encode the categorical columns with numbers
    label_enc = LabelEncoder()
    for i in object_cols_ind:
        df.iloc[:,i] = label_enc.fit_transform(df.iloc[:,i])

# 把非数值型或类别性质的特征属性如Sex/Embarked等转化为数值型
impute_cats(train)
impute_cats(test)
```

3.3.7 特征属性的展示

在进行以上所有特征的特征工程，图 3-29 展示了 train 的预览。注意，要把 PassengerID 和原来的 Ticket 列去掉。图 3-30 显示了 train 和 test 的数据类型（如 int32、float 等）统计，他们都在经过特征工程后转化为了 ML 模型可以接受的数值型数据。

	Survived	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked	Title	Ticket2	Fsize	Single	SmallF	MedF	LargeF	Pc_1	Pc_2	Pc_3
0	0	1	22.0	1	0	7.2500	1	2	2	9	2	0	1	0	0	0	0	1
1	1	0	38.0	1	0	71.2833	2	0	1	8	2	0	1	0	0	1	0	0
2	1	0	26.0	0	0	7.9250	1	2	1	16	1	1	0	0	0	0	0	1
3	1	0	35.0	1	0	53.1000	2	2	1	6	2	0	1	0	0	1	0	0
4	0	1	35.0	0	0	8.0500	1	2	2	6	1	1	0	0	0	0	0	1

图 3-29：特征工程后的 train 概览。

```
Train Dtype counts:
int64      10
uint8       3
int32       3
float64     2
dtype: int64
Test Dtype counts:
int64      8
uint8      3
int32      3
float64    3
dtype: int64
```

图 3-30：train 和 test 的数据类型

3.3.8 特征属性相关性

此外，我们计算了 train 中每个特征之间的相关性，如下图 3-31 所示。**根据颜色的深浅可以判断，与 Survived=1 最呈负相关性分别是：Sex（男为 1，女为 0）、Title、Pc_3，说明这两个特征很大程度影响生还！Pc_1、Fare、Cabin 与 Survived=1 最呈正相关性，说明这三个特征很大程度影响生还！**

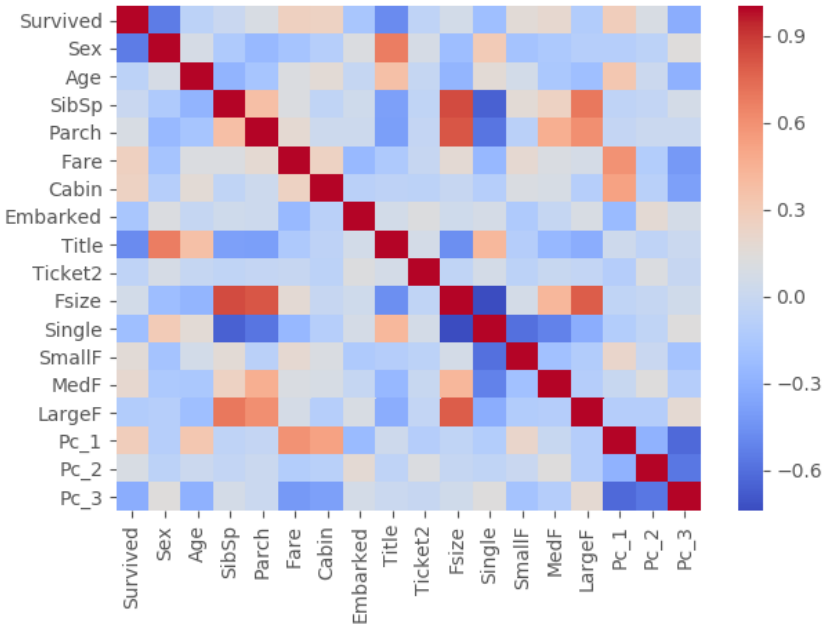


图 3-31：所有特征属性之间的相关系数矩阵图。

3.4 模型选择

在这部分我们选择模型对测试集 test 进行预测，并且给出预测结果。首先，我们在 3.4.1 节采用单个模型进行预测并且展示提交后的预测准确率 accuracy。接着，在在 3.4.2 节，

为了提高预测准确率 accuracy，我们将会展示如何对模型进行融合，也即集成学习（ensembling），并且展示在 test 上的预测结果。

这里用到了 python 的多种机器学习集成包，如下所示，

```
# ---------------------------------------------------------模型选择-------------- ----------------------------
# import the models
from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier

# import evaluators, stacking, etc.
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score, StratifiedKFold, train_test_split, GridSearchCV
# Package for stacking models
from vecstack import stacking
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

代码说明：在用单一机器学习模型或集成机器学习模型进行回归或者分类的时候，通过导入 sklearn、vecstack、lightgbm、xgboost 等包的建立模型并且进行预测。

在利用单一模型预测后，利用集成学习的方法选择多个表现好的单一模型进行集成学习主要有 **Bagging、Boosting、Stacking 三种** [5]，主要目的是提高模型的预测准确率。这里步探讨他们的具体原理和适用性等等。

3.4.1 单一模型预测

首先选择单一模型，接着得出单一模型的预测结果并且进行讨论分析。

3.4.1.1 模型选择

为了观察每个模型的表现，并且为了后面做集成学习提高准确率，这里我基于 train 数据集训练了多个分类模型，分别是：逻辑回归（Logistic Regression）、支持向量机（Support Vector Machine）、随机森林（Random Forest Classifier）、线性识别分析（Linear Discriminant Analysis）、多层感知机（MLPClassifier）、LGBM 分类器（Light Gradient Boosting Machine）、XGBClassifier（Extreme Gradient Boosting Classifier）。需要注意的是，这里的随机森林（Random Forest Classifier）是集成学习中的 Bagging，LGBMClassifier 和 XGBClassifier 是集成学习中的 boosting [9]。

一般来说，得到每一个单一模型的步骤是：初始化模型→验证模型（K 折交叉验证）→拟合得到最优的模型。为了方便展示，这里定义 train_model 函数计算 K 折交叉验证的模型

的准确度，如下所示：

```
def train_model(classifier, name="Classifier"):
    '''这个函数是用户训练所有模型并且打印每个模型的accuracy'''

    # 分层采样，KFold不同的是，保证不同类别的样本在训练集和测试集的比例一致
    folds = StratifiedKFold(n_splits=5, random_state=42)
    accuracy = np.mean(cross_val_score(classifier, X, y, scoring="accuracy", cv=folds, n_jobs=-1))# 求取所有"折"的accuracy均值
    if name not in alg_list:
        alg_list.append(name)# 把算法名字加到List对象alg_list中
    print(f"{name} Accuracy: {accuracy}")
    return accuracy

# 交叉值平均值和算法名称的列表
cv_means = []
alg_list = []
```

接着，我展示生成每个单一模型的关键代码。注意，训练集 train 被划分为特征属性数据和标签，即 train = X（特征属性） + y (labels， 0 为死，1 为生)。这里计算 5 折交叉验证下的准确率。

（1）、Logistic Regression

```
# --------------------LogisticRegression------------------------ #
# Initialize the model
#         LogisticRegression(C=1.0, penalty='l1', tol=1e-6)
log_reg = LogisticRegression(C=5, penalty='l1',tol=1e-6, random_state=42)# penalty 有l1和l2
# Validate the model
log_reg_acc = train_model(log_reg, "Logistic Regression")
cv_means.append(log_reg_acc)
# Fit the best performing model to training data
log_reg.fit(X, y)
```

代码说明：调用 LogisticRegression 模型初始化模型并且计算交叉验证准确率，接着调用 fit 函数得到训练的最终 LogisticRegression 模型。

（2）、Support Vector Machine

```
# ---------Support Vector Machine without GridSeachCV------------ #
# Initialize the model
svm = SVC(C=5, random_state=42)
# Validate the model
svm_acc = train_model(svm, "Support Vector Machine")
cv_means.append(svm_acc)
# Fit the best performing model to training data
svm.fit(X, y)
```

代码说明：SVM 的惩罚参数 C=5.0.其他为默认。

（3）、RandomForestClassifier

```
# --------------------RandomForestClassifier---------------------- #
# Initialize the model
rf = RandomForestClassifier(n_estimators=50, max_depth=20,
                            min_samples_split=2, min_samples_leaf=5,
                            max_features="log2", random_state=12)
# Validate the model
rf_acc = train_model(rf, "Random Forest")
cv_means.append(rf_acc)
# Fit the best performing model to training data
rf.fit(X, y)
```

代码说明：弱分类器的数量是 n_estimators=50，单决策树最大深度为 20 等等。

（4）、Linear Discriminant Analysis

```
# ------------------LinearDiscriminantAnalysis--------------------- #
# Initialize the model
lda = LinearDiscriminantAnalysis(solver='lsqr')
# Validate the model
lda_acc = train_model(lda, "Linear Discriminant Analysis")
cv_means.append(lda_acc)
# Fit the best performing model to training data
lda.fit(X, y)
```

（5）、MLPClassifier

```
# ------------------------MLPClassifier----------------------- #
# Initialize the model
mlp = MLPClassifier(hidden_layer_sizes=(50, 10), activation='relu', solver='adam',
                    alpha=0.01, batch_size=32, learning_rate='constant',
                    shuffle=False, random_state=42, early_stopping=True,
                    validation_fraction=0.2, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10)
# Validate the model
mlp_acc = train_model(mlp, "MLP")
cv_means.append(mlp_acc)
# Fit the best performing model to training data
mlp.fit(X, y)
```

（6）、LGBMClassifier

```
# ------------------------LGBMClassifier----------------------- #
# Initialize the model
lgbm = LGBMClassifier(num_leaves=31, learning_rate=0.1,
                      n_estimators=64, random_state=42, n_jobs=-1)
# Validate the model
lgbm_acc = train_model(lgbm, "LGBM")
cv_means.append(lgbm_acc)
# Fit the best performing model to training data
lgbm.fit(X, y)
```

（7）、XGBClassifier

```
# ------------------------XGBClassifier----------------------- #
# Initialize the model
xgb = XGBClassifier(max_depth=5, learning_rate=0.1, n_jobs=-1, nthread=-1,
                    gamma=0.06, min_child_weight=5,
                    subsample=1, colsample_bytree=0.9,
                    reg_alpha=0, reg_lambda=0.5,
                    random_state=42)
# Validate the model
xgb_acc = train_model(xgb, "XgBoost")
cv_means.append(xgb_acc)
# Fit the best performing model to training data
xgb.fit(X, y)
```

3.4.1.2 模型预测结果

我们在训练集 train 上训练出了 7 个单一模型，并且进行交叉验证，下面图 3-32 展示了 7 个单一模型的训练准确率。我们可以看到，XGBoosting、LGBM、LogisticRegression、RandomForest 表现最好，平均训练准确率超过 82%。相比之下，MPL 和 SVM 的变现最差，平均训练准确率在 78.5%左右。

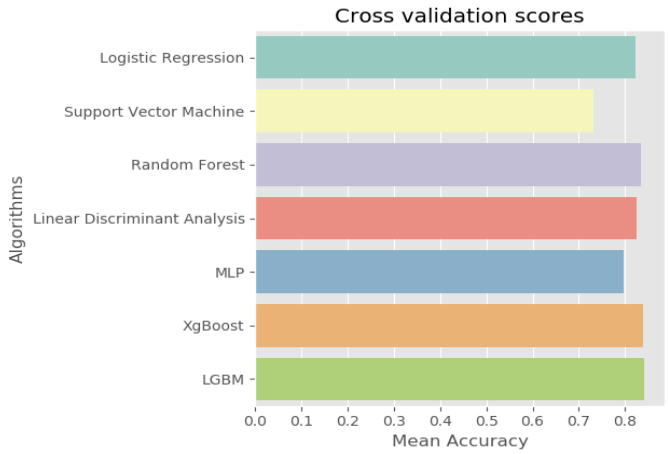
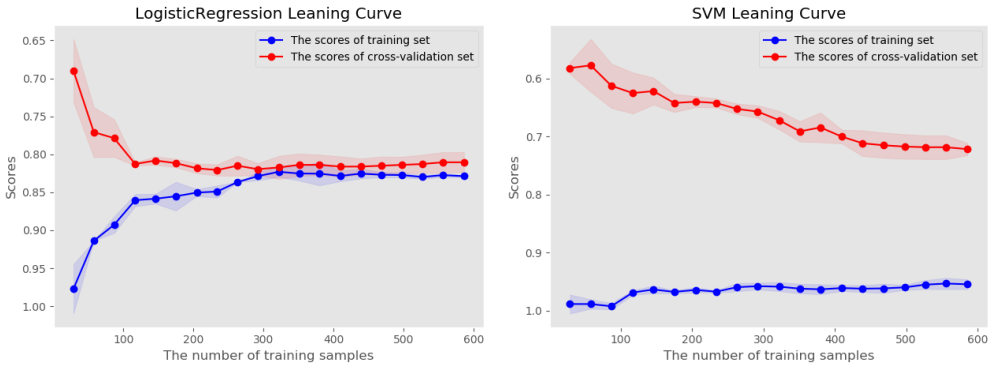


图 3-32：7 个单一模型的平均训练准确率。

此外，我们也展示了 7 个模型的学习曲线 learning curve，学习曲线的代码如下：

```
# models = [rf, lda, lgbm, log_reg, mlp, gssvm, gsrf]
plot_learning_curve(svm, u"XGB Leaning Curve", X, y)
plot_learning_curve(xgb, u"XGB Leaning Curve", X, y)
plot_learning_curve(rf, u"Random Forest Leaning Curve", X, y)
plot_learning_curve(lda, u"LDA Leaning Curve", X, y)
plot_learning_curve(lgbm, u"LGBM Leaning Curve", X, y)
plot_learning_curve(log_reg, u"LogisticRegression Leaning Curve", X, y)
plot_learning_curve(mlp, u"MLP Leaning Curve", X, y)
```

如下图 3-33 所示展示了每个模型的学习曲线，不难发现，模型 MLP 和 SVM 的学习曲线十分不稳定，说明这两个模型的泛化能力较差，数据可能出现了过拟合或者欠拟合的情况，不能很好地对数据变化进行应变。



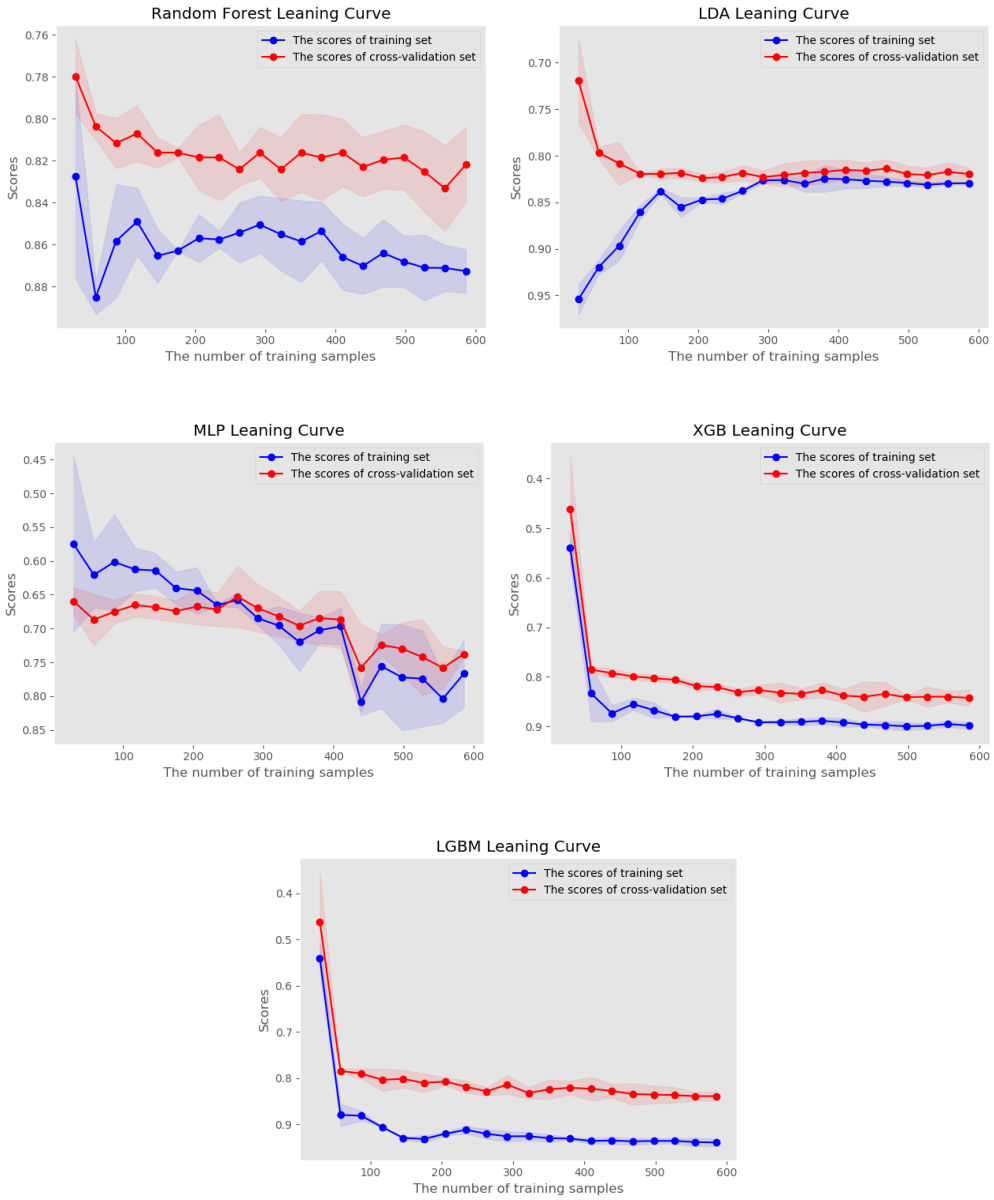


图 3-33：7 个模型在训练集上的学习曲线。

最后，这里提供每个模型最终在 Kaggle 上的测试结果，如表 3-1 所示，这个结论基本和前面的训练准确率一致，我们可以看到，这里最高为 xgboost 模型的 0.78468。

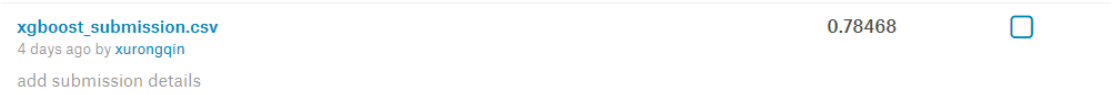


表 3-1：7 个单一模型的测试准确率。

模型	测试准确率
LogisticRegression	0.76555
SVM:0.66985	0.66985

Random Forest	0.76555
LDA	0.78468
MLP	0.65550
XGBoost	**0.78468**
LGBM：0.74162	0.74162

3.4.2 模型融合

这里，我们采用集成学习中的 stacking 进行模型的集成学习，目的是提高准确率。图 3-34 展示了 stacking 的主要思想，具体请参考博客 [10]。

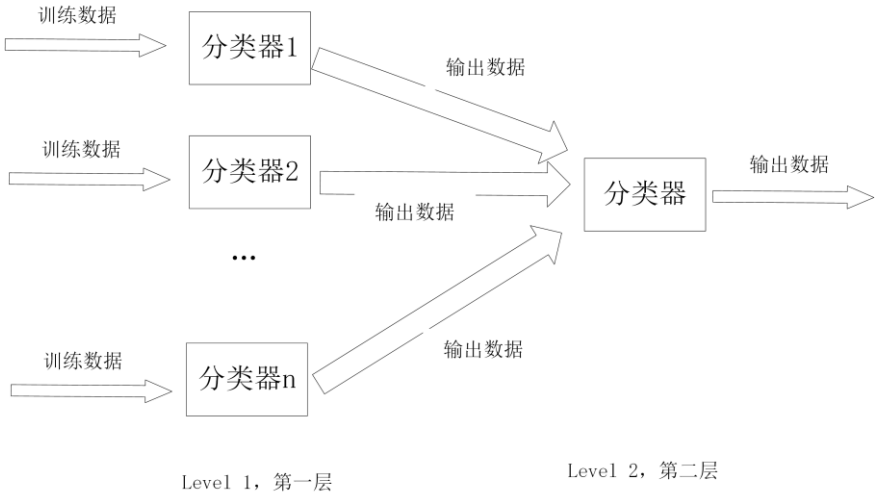


图 3-34：集成学习 Stacking 的主要思想

下面是 Stacking 的主要代码：

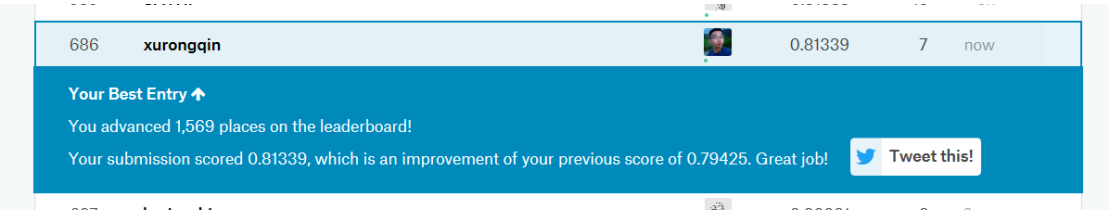
```
# ----------------------------------------------------Stacking--------------------------------------
''' 以下是多种模型的stacking'''
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
# 第一层模型
models = [rf, lda, lgbm, log_reg]
# Perform Stacking
print('-----------------performing stacking------------------')
S_train, S_test = stacking(models,
                           X_train, y_train, X_test,
                           regression=False,
                           mode='oof_pred_bag',
                           n_folds=5,
                           save_dir=None,
                           needs_proba=False,
                           random_state=42,
                           stratified=True,
                           shuffle=True,
                           verbose=2
                          )

# 进入第二层，利用第一层的输出值S_train
xgb.fit(S_train, y_train)

# 在S_test上做预测，作为最终的accuracy
stacked_pred = xgb.predict(S_test)
print('Final prediction score: [%.8f]' % accuracy_score(y_test, stacked_pred))
''' 以上是多种模型的stacking'''
```

代码说明与分析：models []变量是 stacking 第一层的几个分类器，我们在第一层的分类器选择中不是直接把上述的 7 个分类器都用上，因为 MLP 和 SVM 表现太差，我们则不把他们作为基本融合分类器叠加到最后的 stacked model 中。此外，在 stacking 的第二层中，我把 XGBoost 分类器作为第二层的分类器。最后输出分类结果。

最后，我把最终得到的 stacked model，利用 predict 方法对 test 进行预测，接着，用过 to.csv()函数把最终的预测结果写入文件中，下面是提交到 kaggle 的结果，可以看到，准确率达到了 0.81339，排名是 686，Top 6%



3．5 算法优化分析和思考

进行了模型 stacking，在 test 上的准确率由 0.78468 上升到 0.81339。但是，经过搜索资料和思考，还有以下可以调优的地方。

3.5.1 部分特征数据标准化

在处理了所有特征属性之后，Age 和 Fare 两个属性，不同乘客的之间的数值幅度变化

比较大了吧。对于大多数机器学习方法（例如，逻辑回归与梯度下降），**若某一特征属性的数据变化范围相差过大，则会大大降低模型的收敛速度，甚至可能不收敛**［8］。因此，我们先用 scikit-learn 里面的 preprocessing 模块对这两个属性做一个 scaling（标准化），就是将一些变化幅度较大的特征化到[-1,1]之内。关键代码如下所示：

```
# ---------------------------------------------------- 标准化 -----------------------------------------
import sklearn.preprocessing as preprocessing
scaler = preprocessing.StandardScaler()
age_scale_param = scaler.fit(train['Age'].values.reshape(-1, 1))
train['Age'] = scaler.fit_transform(train['Age'].values.reshape(-1, 1), age_scale_param)
fare_scale_param = scaler.fit(train['Fare'].values.reshape(-1, 1))
train['Fare'] = scaler.fit_transform(train['Fare'].values.reshape(-1, 1), fare_scale_param)

scaler = preprocessing.StandardScaler()
age_scale_param = scaler.fit(test['Age'].values.reshape(-1, 1))
test['Age'] = scaler.fit_transform(test['Age'].values.reshape(-1, 1), age_scale_param)
fare_scale_param = scaler.fit(test['Fare'].values.reshape(-1, 1))
test['Fare'] = scaler.fit_transform(test['Fare'].values.reshape(-1, 1), fare_scale_param)
```

代码说明与分析：这里主要利用 preprocessing.StandardScaler()生成的对象对 train 和 test 中的 Age、Fare 属性进行标准化到 [-1,1]。

如图 3-35 所示，展示了标准化后的 train。

```
------------train head after feature engineering---------
   Survived  Sex       Age  SibSp  Parch      Fare  Cabin  Embarked  Title  Ticket2  Fsize  Single  SmallF  MedF  LargeF  Pc_1  Pc_2  Pc_3
0         0    1 -0.565653      1      0 -0.497646      1         2      2        9      2       0       1     0       0     0     0     1
1         1    0  0.657799      1      0  0.837243      2         0      1        8      2       0       1     0       0     1     0     0
2         1    0 -0.259790      0      0 -0.483574      1         2      1       16      1       1       0     0       0     0     0     1
3         1    0  0.428402      1      0  0.458180      2         2      1        6      2       0       1     0       0     1     0     0
4         0    1  0.428402      0      0 -0.480968      1         2      2        6      1       1       0     0       0     0     0     1
```

图 3-35：标准化后的 train。

3.5.2 特征属性的选择

3.5.2.1 分析特征属性相关性

此外，我们计算了 train 中每个特征之间的相关性，如图 3-38 所示。**根据颜色的深浅可以判断，与 Survived=1 最呈负相关性分别是：Sex（男为 1，女为 0）、Title、Pc_3，说明这两个特征很大程度影响生还！Pc_1、Fare、Cabin 与 Survived=1 最呈正相关性，说明这三个特征很大程度影响生还！这个发现，**这对特征选取很有作用！

3.5.2.2 选择主要特征属性

若特征太多，则会在后面的模型训练过程中发生过拟合的现象，因此，我们需要根据 Survived 和其他特征相关性的强弱，来人工选择主要特征，剔除次要特征，主要有如下几

步：

（1）、首先，观察具体的相关性数值，如图 3-36 所示。我们注意到，**特征属性‘SibSP’和‘Parch’、‘Fsize’的信息已经被提取为新的特征属性‘Single’、‘SmallF’、‘MedF’、‘LargeF’**，且由图 3-38 和图 3-36 观察到他们和 Survived 之间的正/负相关性都很弱，因此，我们这里把‘SibSP’和‘Parch’、‘Fsize’删除。另外一方面的考虑是，弱特征太多，会加大模型的运算量！

----------Pearson Correlations values----------

	Survived	Sex	Age	SibSp
Survived	1.000000	-0.546015	-0.065338	0.003330
Sex	-0.546015	1.000000	0.080904	-0.129308
Age	-0.065338	0.080904	1.000000	-0.284270
SibSp	0.003330	-0.129308	-0.284270	1.000000
Parch	0.092819	-0.245750	-0.171010	0.379281
Fare	0.264613	-0.179277	0.110667	0.110072
Cabin	0.249399	-0.095859	0.167306	-0.042594
Embarked	-0.166252	0.111279	-0.017183	0.041002
Title	-0.477527	0.678274	0.369900	-0.384854
Ticket2	-0.051568	0.080099	-0.019426	-0.042326
Fsize	0.055279	-0.222342	-0.277243	0.847921
Single	-0.211448	0.303978	0.169826	-0.652861
SmallF	0.162092	-0.187916	0.055345	0.167642
MedF	0.192959	-0.147437	-0.153278	0.250418
LargeF	-0.119398	-0.098524	-0.210907	0.698516
Pc_1	0.282080	-0.096070	0.329162	-0.048543
Pc_2	0.091718	-0.067084	0.014454	-0.031256
Pc_3	-0.317666	0.137476	-0.295059	0.067308

图 3-36：特征属性之间的相关性值。

（2）、接着，从图 3-38 和 3-36 可知，我们剔除一些弱特征，**这里我们剔除：Pc_2、Age、Ticket2 这三个**。

进行以上两个特征属性选择和剔除之后，我们看看最后的所有特征属性之间的相关系数矩阵图，如图 3-37 所示。

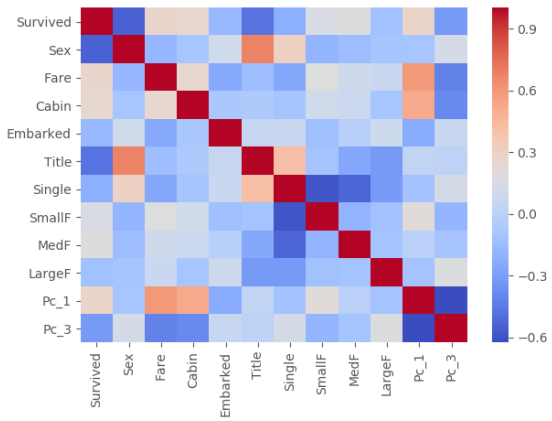


图 3-37：选择相关性强的特征后，所有特征属性之间的相关系数矩阵图。

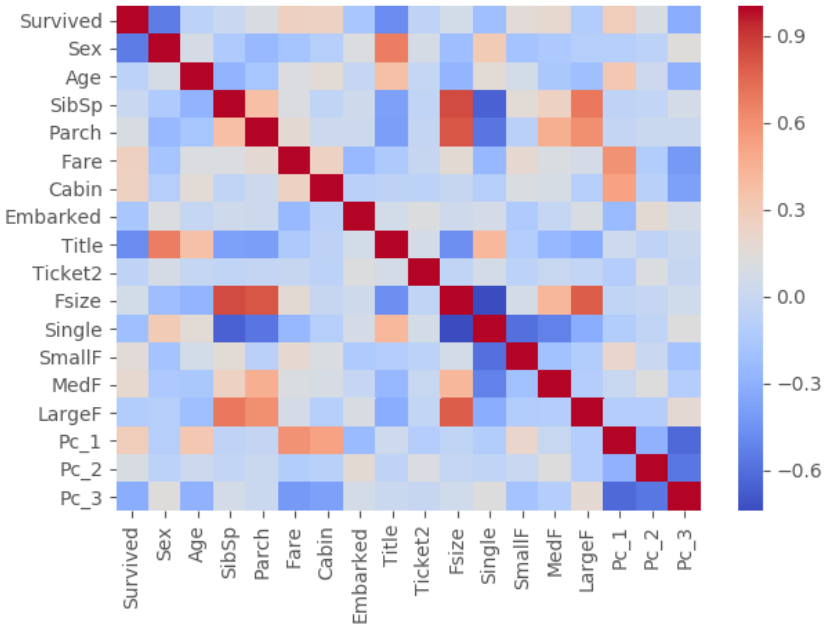


图 3-38：所有特征属性之间的相关系数矩阵图。

进行低相关特征剔除后，我们重新跑程序，可以看到，最终 stacked 模型的训练过程如下所示（**最后的 Final prediction score 为最优的训练准确率，即为 0.864406**）：

```
------------------performing stacking--------------------
task:         [classification]
n_classes:    [2]
metric:       [accuracy_score]
mode:         [oof_pred_bag]
n_models:     [7]

model  0:     [RandomForestClassifier]
    fold  0:  [0.80851064]
    fold  1:  [0.82978723]
    fold  2:  [0.79432624]
    fold  3:  [0.82978723]
    fold  4:  [0.83571429]
    ----
    MEAN:     [0.81962513] + [0.01567778]
    FULL:     [0.81960227]

model  1:     [LinearDiscriminantAnalysis]
    fold  0:  [0.80851064]
    fold  1:  [0.80851064]
    fold  2:  [0.80141844]
    fold  3:  [0.81560284]
    fold  4:  [0.83571429]
    ----
    MEAN:     [0.81395137] + [0.01176970]
    FULL:     [0.81392045]

model  2:     [LGBMClassifier]
    fold  0:  [0.82269504]
    fold  1:  [0.84397163]
    fold  2:  [0.79432624]
    fold  3:  [0.80141844]
    fold  4:  [0.81428571]
    ----
    MEAN:     [0.81533941] + [0.01738025]
    FULL:     [0.81534091]
```

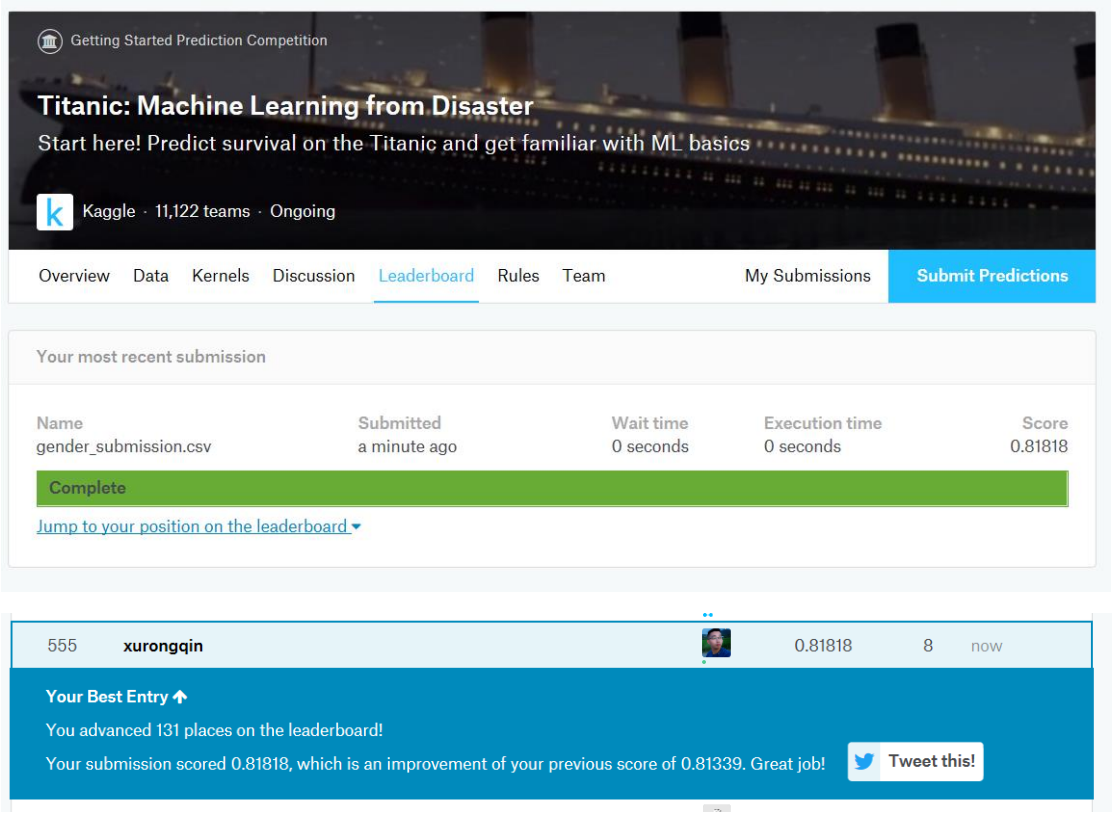
```
model  4:     [MLPClassifier]
    fold  0:  [0.73758865]
    fold  1:  [0.70921986]
    fold  2:  [0.67375887]
    fold  3:  [0.65248227]
    fold  4:  [0.75714286]
    ----
    MEAN:     [0.70603850] + [0.03880060]
    FULL:     [0.70596591]

model  5:     [GridSearchCV]
    fold  0:  [0.80141844]
    fold  1:  [0.81560284]
    fold  2:  [0.82269504]
    fold  3:  [0.80851064]
    fold  4:  [0.85000000]
    ----
    MEAN:     [0.81964539] + [0.01675261]
    FULL:     [0.81960227]

model  6:     [GridSearchCV]
    fold  0:  [0.82269504]
    fold  1:  [0.81560284]
    fold  2:  [0.77304965]
    fold  3:  [0.83687943]
    fold  4:  [0.83571429]
    ----
    MEAN:     [0.81678825] + [0.02328610]
    FULL:     [0.81676136]

Final prediction score: [0.86440678]
```

最终的提交结果为：可以看到，由 0.81339 提升到了 0.81818！排在 Top 5%。



3.5.3 单一模型的优化

我们可以从图 3-32 看到，因此 SVM 和 MLP 这两个模型的表现差而没有加到最终融合的 stacked 模型中。因此，这里的一个可以考虑的方向是，利用 GridSearch 等参数优化方法，寻求 SVM 和 MLP 这两个模型最优的参数 [6,11]，从而优化 SVM 和 MLP 模型发挥最大的潜能。

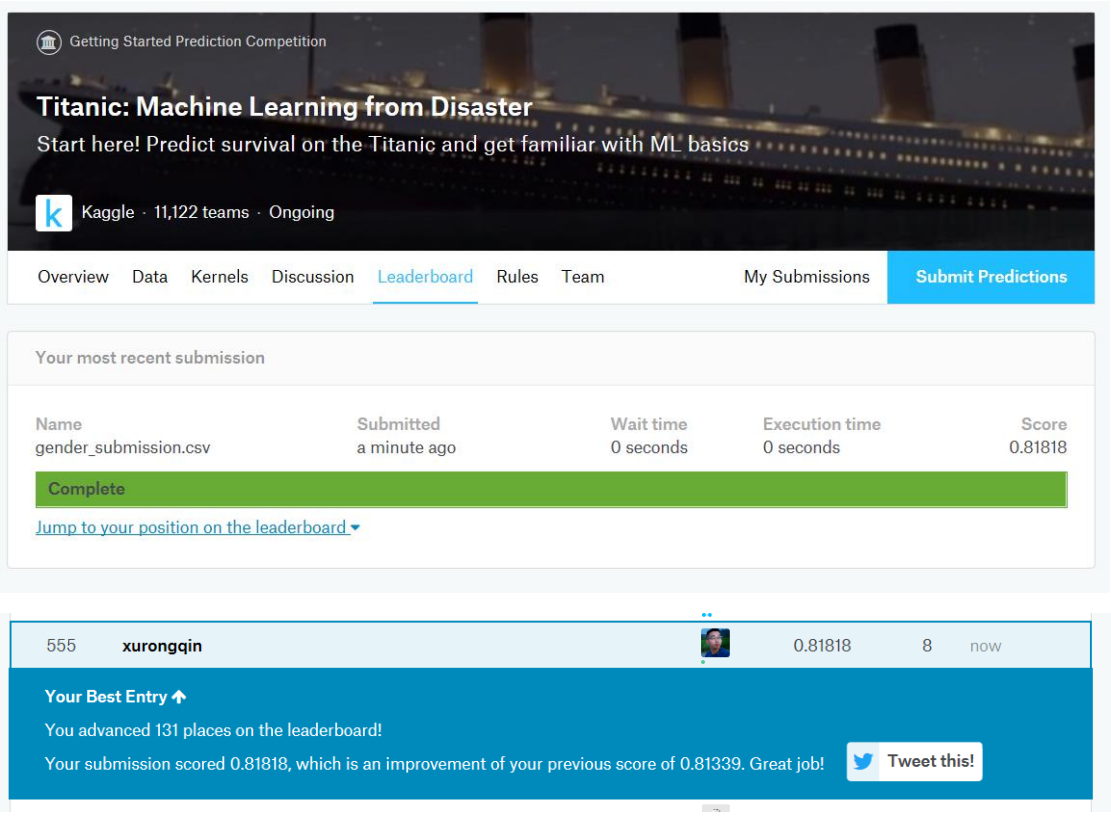
此外，本项目使用的 stacked 模型貌似有点复杂，我们可以选择更加简单的、合适的基模型去完成 stacking，完成预测任务，文献[6]给出了一种例子。

3.5.4 特征选取改进

寻求更加高效的特征属性选取方法！经过调查，多数预测准确率高的选手，都会根据原有的特征，派生出更多的特征或者删除部分低相关的特征，然后利用 PCA 等手段进行数据降维等等，这都是本项目可以继续探索，继续改进的一种方式之一。

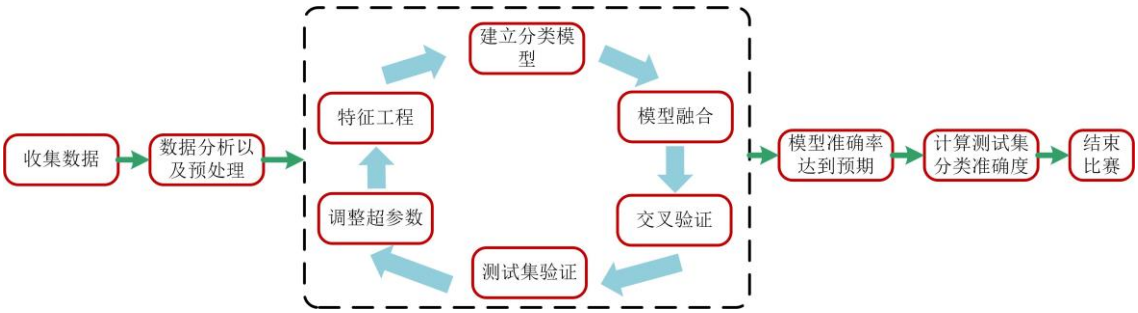
四、项目结果

最终的提交结果为：**由 0.81339 提升到了 0.81818！排在 Top 5%（其他过程结果在上述“实验过程”部分）。**



五、项目总结

本文主要以泰坦尼克号生还预测为主题，详细地介绍了如何利用机器学习方法去解决预测问题。本文从介绍项目目的开始，先详述了如何通过数据预览去探索式数据分析，然后缺失数据填补，删除关联特征以及派生新特征等方法，最后在单一模型的基础上，通过集成学习的方法建立起一个高效地、准确率高的预测模型，并且对模型进行了评价、分析和调优，最后在 Kaggle 的 Titanic 幸存预测这一分类问题竞赛中获得排名为前 5%，准确率为 0.81818 的具体方法，具体过程如下图所示。



经过这一些列的实验，我发现了要建立一个泛化能力强的机器学习模型，一般是先建立多个基本的单一模型，然后选择其中表现得好的基本模型进行集成学习（例如，Bagging，Boosting 和 stacking）。经过试验观察，我发现在泰坦尼克预测中，Stacking 方法表现最好。此外，**通过实验观察，我发现，“数据缺失值填补”、“特征工程（包括删除关联特征以及派生新特征等）”、“模型选择”、“模型融合”，对预测结果的影响十分大，这个需要经验也需要时间去进行揣测**。

我要感谢陈小军老师这一学期的任课，使我对数据挖掘这门课有了系统的认识。再次，通过此项目，我了解了在 Kaggle 平台上打比赛的流程和步骤，感觉十分喜欢。再次，我了解了很多数据挖掘、机器学习的知识点和技巧，特别在处理特征工程、建立模型的时候，花了很长的时间，最终才取得满意的效果。

我还需要向高手们学习，在实验过程中，参考了 CSDN 热门文章和 Kaggle Kernel 上的相关研究和分析，受到了他们的启发，特别是在特征工程的章节里面，我向 CSDN 的高手学习了了不少！ 此外，我后续会考虑如何用最简单的模型进行继续提高。

本项目预测结果为：准确率为 0.81818，排名为 555/11122，Top 5%，。

参考文献

[1] https://www.kaggle.com/ldfreeman3/a-data-science-framework-to-achieve-99-accuracy.

[2] https://blog.csdn.net/mw21501050/article/details/75389267.

[3] https://blog.csdn.net/Koala_Tree/article/details/78725881.

[4] https://blog.csdn.net/han_xiaoyang/article/details/49797143.

[5] https://blog.csdn.net/Koala_Tree/article/details/78725881.

[6]https://www.kaggle.com/abhinand05/machine-learning-for-everyone-lb-top-5/notebook#Building,-Training-and-Validating-our-Models.

[7] 皮尔逊系数，https://blog.csdn.net/laozhaokun/article/details/25156923

[8] 数据归一化与模型收敛，https://blog.csdn.net/leiting_imecas/article/details/54986045

[9] LGBM 和 XGBoosting，https://www.jianshu.com/p/8d61703e01b4

[10] https://blog.csdn.net/qq1483661204/article/details/80157365

[11] https://www.kaggle.com/goldens/classification-81-3-with-simple-model-nested-cv#3---Submission