



中山大學  
SUN YAT-SEN UNIVERSITY

课 程: 2019 机器学习与数据挖掘

项 目: Titanic: Machine Learning  
from Disaster (Kaggle 赛题)

院 系: 数据科学与计算机学院

专 业: 软件工程

学生姓名: 陈吉凡

学 号: 16340020

授课教师: 苏勤亮

2020 年 5 月 16 日

## 目录

一、作业说明和赛题介绍 .....	1
A. 作业说明 .....	1
B. 赛题介绍 .....	1
二、数据分析 .....	1
A. 数据 .....	1
B. 数据分析 .....	2
三、数据预处理 .....	2
A. 数据清洗 .....	3
B. 缺失值处理 .....	3
C. 字符串转换 .....	3
D. 类别数据处理 .....	4
E. 归一化 (Normalize) 问题 .....	5
四、特征工程 .....	6
A. 特征选择 .....	6
B. 具体数据变换方法 .....	6
五、模型选择及调参 .....	7
A. 模型选择 .....	7
B. 训练方法 .....	10
C. 参数调节 .....	11
六、训练结果和分数 .....	13
A. 得到的文件 .....	13
B. 得分 .....	13
七、遇到的困难和解决方法 .....	13
八、总结 .....	14
九、心得 .....	14
十、其他 .....	14
A. 代码展示 .....	15
B. 文件说明 .....	15
C. 参考文献 .....	15

---

## 一、 作业说明和赛题介绍

### A. 作业说明

本次实践我选择完成 Kaggle 平台上的一个题目，并以此报告来作为苏勤亮老师的机器学习与数据挖掘课程的期末作业。

赛题网址：<https://www.kaggle.com/c/titanic/overview>

### B. 赛题介绍

**赛题背景：**当今时代，空难、海难频频发生，无数生命、财产毁于一旦。在灾难发生的同时，我们希望能从中提取出一些蕴含价值的信息，对灾难防范、善后等步骤都有积极的作用。

**赛题说明：**泰坦尼克号的沉没举世震惊，本次赛题的数据正是从这次事故中取得的。赛题要求通过原始数据集给出的信息，预测乘客是否能够幸存，原始信息包括泰坦尼克号事故相关乘客的性别、年龄、票价、舱位、上船地点等。

## 二、 数据分析

### A. 数据

比赛数据为泰坦尼克号事故中乘客的基本信息，共三个数据集（train.csv, test.csv, gender\_submission.csv），除去最后测试结果文件外，剩下的两个分别为训练集和测试集，两个文件结构大致相同，仅是后者没有 Survived 列（即待预测列）。

具体内容：

列名	含义	类型
PassengerId	乘客唯一标识	整数
Survived	乘客是否幸存标识，1 为是，0 为否	整数

Pclass	票的级别，分 123 等	整数
Name	乘客姓名	字符串
Sex	乘客性别，male 或 female	字符串
Age	乘客年龄	浮点数
SibSp	乘客在船上的兄弟姐妹/配偶数	整数
Parch	乘客在船上的父母数	整数
Ticket	票的编号	字符串
Fare	票价	浮点数
Cabin	船舱号	字符串
Embarked	上船地点代号	字符串

## B. 数据分析

本赛题相当于只需要处理一张表，其中 PassengerId 为编号，Name 表示名字，这三列的信息对乘客是否生还的影响最小，基本可以忽略不记（Name 列含有性别信息但是属于冗余信息），那么可以考虑直接将这两列去除。

### 剩余列的分析：

1. 票级别、票价：与一个乘客的财富、地位有关，影响存活的几率；
2. 性别、年龄：老人、小孩、女人势必更受到保护，也有可能因为没有成年男性强壮而产生影响。
3. 家庭成员：存在协作逃生、互相保护的影响；
4. 船舱、票号、上船地点：和船舱的位置有关系，离出口距离、设施等都有可能产生影响。

### 补充分析：

1. Age 中有约 200 条是没有值，对这部分数据需要进行填充等处理；
2. Cabin 中的有效数据项很少，只有 200 多条有效，共 76 类，但含有 A,B 等前缀，可能含有有用信息；
3. Fare、Embarked 也存在空缺值。

## 三、数据预处理

---

## A. 数据清洗

**分析：**数据清洗的作用在于避免无项的干扰，清洗标准可以为在一行中空缺值超过 80%即认为该行信息没有价值并删去。但是通过对数据的观察，该数据均为有效行，无需进行数据清洗的工作。

## B. 缺失值处理

**分析：**出现缺失值的列有 Age、Fare、Cabin 和 Embarked 列，其中 Age 较多出现，其他两列偶尔出现。对于这些缺失值不可以简单的填充为 0 或者空字符串这些没有代表性的数据。

**处理方式：**将 Age、Fare 用平均数填充，Cabin 列使用额外增加一个类别的形式（原共 76 类，填充为第 77 类），Embarked 列使用众数填充。

```
1. baseTable['Age'] = baseTable['Age'].fillna(baseTable['Age'].mean())
2. baseTable['Embarked'] = baseTable['Embarked'].fillna(baseTable['Embarked'].mode()[0])
3. baseTable['Fare'] = baseTable['Fare'].fillna(baseTable['Fare'].mean())
```

## C. 字符串转换

数据项中的 Sex,Cabin,Embarked 均为字符串类型，我们需要将它们转换为训练模型可以处理的类型。

**处理方法：**将相同字符串的划为一类，用数字 1, 2...来代替。

**具体转换：**比较复杂，看以下源码：

```
1. class2id = {}
2. id2class = {}
3. def processStr(baseTable):
4.     resTable = baseTable
5.     cat_columns = ['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked']
6.     for c in cat_columns:
7.         resTable[c] = resTable[c].apply(lambda x: x if type(x)==str else str(x))
8.         sort_temp = sorted(list(set(resTable[c])))
```

```
9.         class2id[c+'2id'] = dict(zip(sort_temp, range(1, len(sort_temp)+1)))

10.         resTable[c] = resTable[c].apply(lambda x: class2id[c+'2id'][x])

11.     return resTable
```

## D. 类别数据处理

**分析：**经过上一步 C 中的处理之后，数据集中只存在数字类型，其中表示类别的列有：Pclass,Sex,Cabin,Embarked，其他列为数值大小有意义的列（除去 PassengerId,Name,Survived 三列），需要考虑是否需要前者进行处理。

**目的：**采用将数据扩展到欧式空间等方式，使不同类数据间的距离相同。

**思路：**一开始是决定用 one-hot encode 或者 hash encode，在这方面花费了很多时间，但是后来查找了很多资料，包括对基于树的模型和三类主流 Boost 模型（xgBoost, anaBoost, GBDT，本比赛使用的是 GBDT 模型，是基于树的 Boost 模型）的具体实现进行深入学习的博客，和 lightGBM 的官方文档等，发现基于树的模型是可以不考虑类别这个问题的，大体原因是基于树的模型采用树作为分类器，因而不受类型间距离的影响。而且 lightGBM 框架也自带了识别类别的方法，实现方法也优于 OHE，实现大致为每次会分为两大类，不断细分，只需要  $2^N$  次分类，因而树的深度也会降低，缺点是容易导致过拟合。

**不同处理方法区别如下：**

### 1) OHE

How it learns : Treats each of the binary levels independently.

How it counts to importance : Each level is counted separately.

Therefore, especially for the organisation type that as far as I remember has over 20 levels, each split will be not only weaker but also counted for only one of the OHE columns.

### 2) Categorical LGBM encoding

How it learns: The split can be made based on the variable being of one specific level or any subset of levels. You have  $2^N$  splits available instead of 4 for OHE.

How it counts to importance: Each split is counted. For the above example, you'll get 100 and therefore higher importance.

### 3) Numerical encoding (pd.factorize)

How it learns:The variable is treated as a numerical one. There is as

---

many splits available as in OHE, but they are made with the constraint of forced, artificial order introduced to a categorical variable.  
How it counts to importance: all of the counts for this variable are counted.

### LightGBM 关于类别特征模块的官方文档如下:

#### Optimal Split for Categorical Features:

Instead of one-hot encoding, the optimal solution is to split on a categorical feature by partitioning its categories into 2 subsets. If the feature has  $k$  categories, there are  $2^{(k-1)} - 1$  possible partitions. But there is an efficient solution for regression trees[8]. It needs about  $O(k * \log(k))$  to find the optimal partition.

The basic idea is to sort the categories according to the training objective at each split. More specifically, LightGBM sorts the histogram (for a categorical feature) according to its accumulated values ( $\text{sum\_gradient} / \text{sum\_hessian}$ ) and then finds the best split on the sorted histogram.

**具体实现:** 不需要一开始的 OHE, 直接使用 lightGBM 的分类方法:

```
1. fea_name = list(X.columns)
2. cata=['Pclass', 'Sex', 'Cabin', 'Havef', 'Fc', 'Ac', 'Embarked']
3. #...
4. model = lgb.train(param, train, feature_name=fea_name, categorical_feature=cata, ...)
```

## E. 归一化 (Normalize) 问题

**目的:** 1) 把数据变成  $(-1, 1)$  或者  $(0, 1)$  之间的小数。主要是为了数据处理方便提出来的, 把数据映射到  $0 \sim 1$  范围之内处理, 更加便捷快速。2) 把有量纲表达式变成无量纲表达式, 便于不同单位或量级的指标能够进行比较和加权。归一化是一种简化计算的方式, 即将有量纲的表达式, 经过变换, 化为无量纲的表达式, 成为纯量。

**思路:** 一开始考虑对票价等一些数据值进行归一化, 之后了解了 lightGBM 的

---

一些具体实现，发现它是基于桶的，归一化并没有什么意义，遂作罢

## 四、特征工程

### A. 特征选择

#### 1. 乘客的年龄 Age

**思路：**乘客的年龄在一定区间内的特点是趋于一致的，我们可以将年龄划为不同区间，从而提取出更具意义的信息。

#### 2. 乘客的家庭成员信息 SibSp 和 Parch

**思路：**乘客的家庭成员有无是一个质变，和一名乘客的性格、旅行是否为家庭旅行都有千丝万缕的联系。我们可以提取出这一信息。

#### 3. 票价 Fare

**思路：**和年龄一个思路，我们也可以观察到 Fare 很明显被分为了几类，每类间的数值差异是比较大的，且和另外的票级别 Pclass 密切相关。我们将 Fare 也进行分类。

#### 4. 票编号 Ticket

**思路：**这里观察数据不难看出，Ticket 有英文前缀和数字编号两部分，数字编号是没有利用价值的，我们只能利用前缀分类，有些没有前缀的我们划分到新一类中去。

### B. 具体数据变换方法

#### 1. Age 和 Fare

```
1. baseTable['Ac'] = pd.cut(baseTable['Age'], bins=[0, 12, 20, 50, 120], labels=[1,2,3,4]);
2. baseTable['Fc'] = pd.qcut(baseTable['Fare'], 4, labels=[1, 2, 3, 4]);
```

#### 2. SibSp 和 Parch

```
1. baseTable['Havef'] = baseTable['Fsize'] != 0;
```



---

### 3. Ticket

```
1. temp = []
2. for i in list(baseTable["Ticket"]):
3.     if not i.isdigit() :
4.         temp.append(i.replace(".", "").replace("/", "").strip().split(' ')[0])
5.     else:
6.         temp.append("Null")
7. baseTable["Ticket"] = temp
```

## 五、模型选择及调参

### A. 模型选择

**模型：**使用 lightGBM 框架。它是一个梯度 Boosting 框架，使用基于决策树的学习算法。它可以说是分布式的，高效的。

**优势：**

- 1) 更快的训练效率
- 2) 低内存使用
- 3) 更高的准确率
- 4) 支持并行化学习
- 5) 可以处理大规模数据

与常见的机器学习算法对比，速度是非常快的

lightGBM 和 XGBoost 对比速度提升了 10 倍左右，占用内存下降了 3 倍左右。因为他是基于决策树算法的，它采用最优的叶明智策略分裂叶子节点，然而其它的提升算法分裂树一般采用的是深度方向或者水平明智而不是叶，明智的。因此，在 LightGBM 算法中，当增长到相同的叶子节点，叶明智算法比水平-wise 算法减少更多的损失。因此导致更高的精度，而其他的任何已存在的提升算法都不能够达。

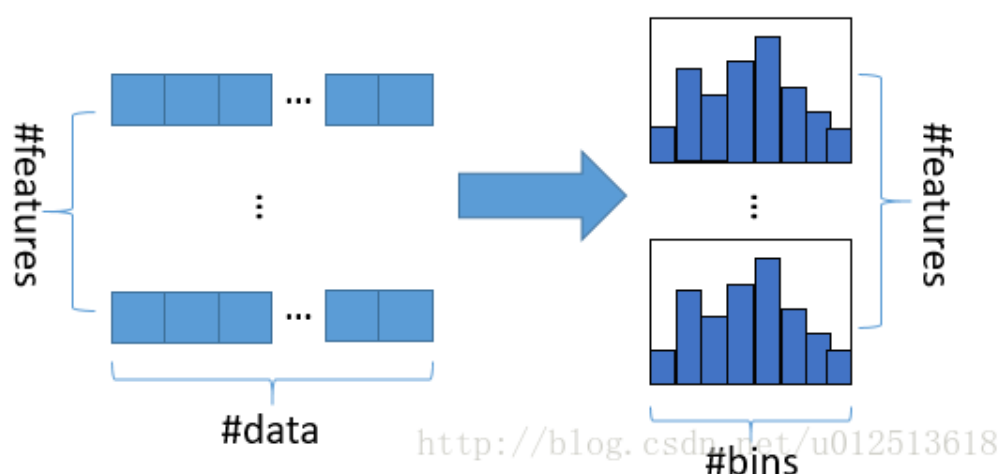
lightGBM 本质使用了 GBDT 模型（它也可以实现随机森林、DART 等，本比赛使用的为 GBDT），很多 kaggle 比赛的冠军代码使用的就是 GBDT 模型。和 XGBoost 对比：

	XGBoost	LightGBM
树木生长算法	按层生长的方式 有利于工程优化，但对学习模型效率不高	直接选择最大收益的节点来展开，在更小的计算代价上去选择我们需要的决策树 控制树的深度和每个叶子节点的数据量，能减少过拟合
划分点搜索算法	对特征预排序的方法	直方图算法：将特征值分成许多小筒，进而在筒上搜索分裂点，减少了计算代价和存储代价，得到更好的性能。另外数据结构的变化使得在细节处的变化理上效率会不同
内存开销	8个字节	1个字节
划分的计算增益	数据特征	容器特征
高速缓存优化	无	在Higgs数据集上加速40%
类别特征处理	无	在Expo数据集上速度快了8倍

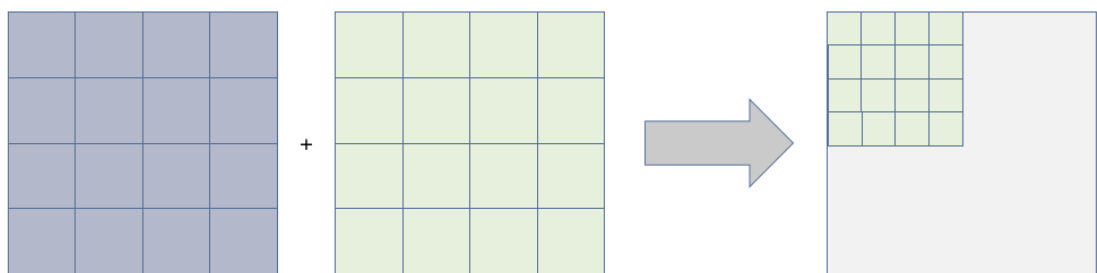
### 一些原理：

#### 1) 直方图算法

直方图算法的基本思想是先把连续的浮点特征值离散化成  $kk$  个整数，同时构造一个宽度为  $kk$  的直方图。在遍历数据的时候，根据离散化后的值作为索引在直方图中累积统计量，当遍历一次数据后，直方图累积了需要的统计量，然后根据直方图的离散值，遍历寻找最优的分割点。在 XGBoost 中需要遍历所有离散化的值，而在这里只要遍历  $kk$  个直方图的值。



使用直方图算法有很多优点。首先，最明显就是内存消耗的降低，直方图算法不仅不需要额外存储预排序的结果，而且可以只保存特征离散化后的值。



Use int\_32 to store sorted index and float\_32 to store feature values

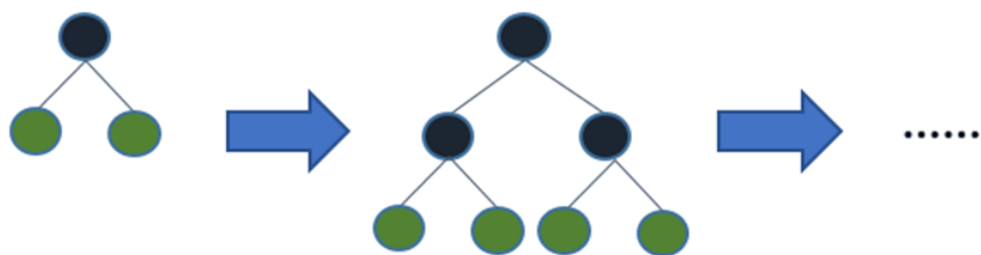
<http://t.cn/R8> Use int\_8 to store feature bins

然后在计算上的代价也大幅降低，XGBoost 预排序算法每遍历一个特征值就需要计算一次分裂的增益，而直方图算法只需要计算  $kk$  次 ( $kk$  可以认为是常数)，时间复杂度从  $O(\#data * \#feature)$  优化到  $O(k * \#features)$ 。

当然，Histogram 算法并不是完美的。由于特征被离散化后，找到的并不是很精确的分割点，所以会对结果产生影响。但在不同的数据集上的结果表明，离散化的分割点对最终的精度影响并不是很大，甚至有时候会更好一点。原因是决策树本来就是弱模型，分割点是不是精确并不是太重要；较粗的分割点也有正则化的效果，可以有效地防止过拟合；即使单棵树的训练误差比精确分割的算法稍大，但在梯度提升 (Gradient Boosting) 的框架下没有太大的影响。

## 2) 带深度限制的 Leaf-wise 的叶子生长策略

Level-wise 过一次数据可以同时分裂同一层的叶子，容易进行多线程优化，也好控制模型复杂度，不容易过拟合。但实际上 Level-wise 是一种低效的算法，因为它不加区分的对待同一层的叶子，带来了很多没必要的开销，因为实际上很多叶子的分裂增益较低，没必要进行搜索和分裂。

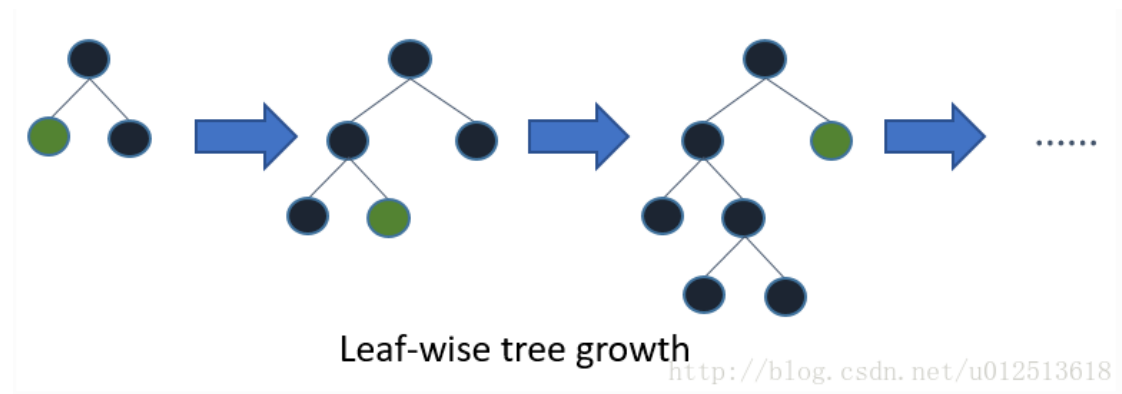


Level-wise tree growth

<http://blog.csdn.net/u012513618>

Leaf-wise 则是一种更为高效的策略，每次从当前所有叶子中，找到分裂增益最大的一个叶子，然后分裂，如此循环。因此同 Level-wise 相比，在分裂次数相同的情况下，Leaf-wise 可以降低更多的误差，得到更好的精度。Leaf-wise 的缺点是可能会长出比较深的决策树，产生过拟合。因此 LightGBM 在 Leaf-

wise 之上增加了一个最大深度的限制，在保证高效率的同时防止过拟合。



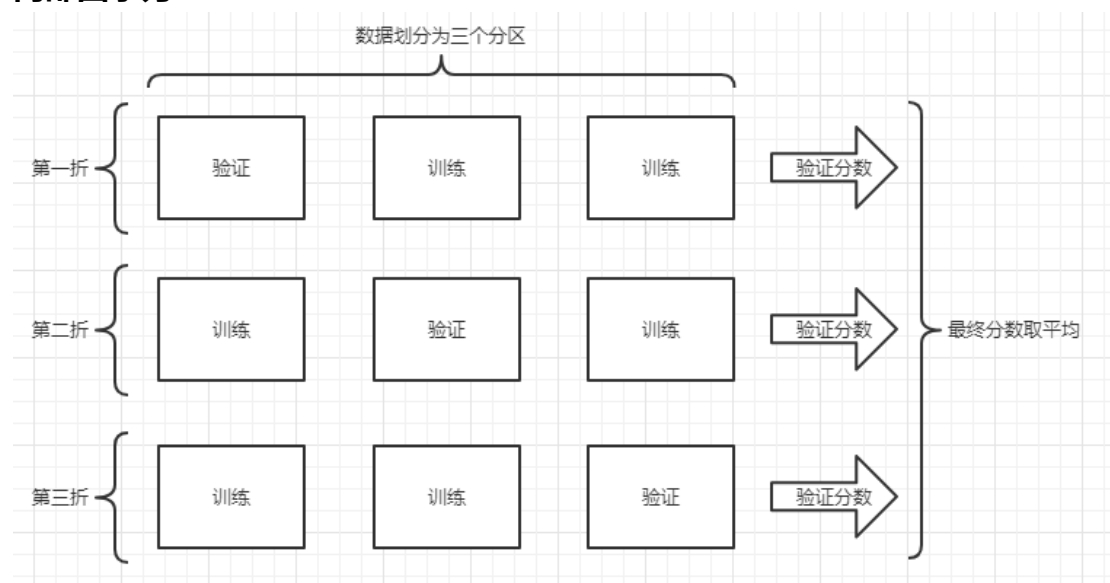
## B. 训练方法

**训练方法：**我使用的是 **3 折交叉验证**。K 折交叉验证是将初始采样（样本集  $X$ ,  $Y$ ）分割成  $K$  份，一份被保留作为验证模型的数据（test set），其他  $K-1$  份用来训练（train set）。交叉验证重复  $K$  次，每份验证一次，平均  $K$  次的结果或者使用其它结合方式，最终得到一个单一估测。这个方法的优势在于，同时重复运用随机产生的子样本进行训练和验证，每次的结果验证一次

具体实现：使用 `sklearn.model_selection` 的 `StratifiedKFold`。

`StratifiedKFold` 和 `KFold` 相比采用的是分层采样，可确保训练集，测试集中各类别样本的比例与原始数据集中相同。

**内部图示为：**



**代码：**

```
skf = StratifiedKFold(n_splits=3, random_state=1030, shuffle=True)
```

n\_splits: 表示划分几等份

random\_state: 随机数种子

shuffle: 在每次划分时, 是否进行洗牌

①若为 False 时, 其效果等同于 random\_state 等于整数, 每次划分的结果相同;

②若为 True 时, 每次划分的结果都不一样, 表示经过洗牌。

## C. 参数调节

**微软官方文档地址:**

<https://github.com/Microsoft/LightGBM/blob/master/docs/Parameters.md>

**思路:** 与大多数使用 depth-wise tree 算法的 GBM 工具不同, 由于 LightGBM 使用 leaf-wise tree 算法, 因此在迭代过程中能更快地收敛; 但 leaf-wise tree 算法较容易过拟合; 为了更好地避免过拟合, 需要重点留意以下参数:

**重要参数:**

1. num\_leaves. 这是控制树模型复杂性的重要参数。理论上, 可以通过设定  $\text{num\_leaves} = 2^{(\text{max\_depth})}$  去转变成为 depth-wise tree。但这样容易过拟合, 因为当这两个参数相等时, leaf-wise tree 的深度要远超 depth-wise tree。因此在调参时, 往往会把 num\_leaves 的值设置得小于  $2^{(\text{max\_depth})}$ 。例如当 max\_depth=6 时 depth-wise tree 可以有个好的准确率, 但如果把 num\_leaves 设成 127 会导致过拟合, 要是把这个参数设置成 70 或 80 却有可能获得比 depth-wise tree 有更好的准确率。事实上, 当用 leaf-wise tree 时, 我们可以忽略 depth 这个概念, 毕竟 leaves 跟 depth 之间没有一个确切的关系。

2. min\_data\_in\_leaf. 这是另一个避免 leaf-wise tree 算法过拟合的重要参数。该值受到训练集数量和 num\_leaves 这两个值的影响。把该参数设的更大能够避免生长出过深的树, 但也要避免欠拟合。在分析大型数据集时, 该值区间在

---

数百到数千之间较为合适。

3. `max_depth`. 可以通过设定 `max_depth` 的值来限制树算法生长过深。

### **提高速度的参数**

- 通过设定 `bagging_fraction` 和 `bagging_freq` 来使用 bagging 算法
- 通过设定 `feature_fraction` 来对特征采样
- 设定更小的 `max_bin` 值
- 使用 `save_binary` 以方便往后加载数据的速度
- 设定平行计算参数

### **提高精度的参数**

- 设定更大的 `max_bin` 值(但会拖慢速度)
- 设定较小的 `learning_rate` 值, 较大的 `num_iterations` 值
- 设定更大的 `num_leaves` 值(但容易导致过拟合)
- 加大训练集数量 (更多样本, 更多特征)

### **避免过拟合的参数**

- 设定较小的 `max_bin`
- 设定更小的 `num_leaves`
- 设定 `min_data_in_leaf` 和 `min_sum_hessian_in_leaf`
- 通过设定 `bagging_fraction` 和 `bagging_freq` 来使用 bagging 算法
- 通过设定 `feature_fraction` 来对特征采样。
- 加大训练集数量 (更多样本, 更多特征)
- 通过设定 `lambda_l1`, `lambda_l2` 以及 `min_gain_to_split` 来采取正则化措施
- 通过设定 `max_depth` 以避免过拟合



### **最终具体设置如下:**


```
'learning_rate': 0.002,  
'lambda_l1': 0.1,  
'lambda_l2': 0.2,  
'max_depth': 4,  
'objective': 'multiclass',
```

```
'num_class': 7,  
'num_leaves': 15,  
'min_data_in_leaf': 10,  
'max_bin': 300,  
'metric': 'multi_error'
```

## 六、训练结果和分数

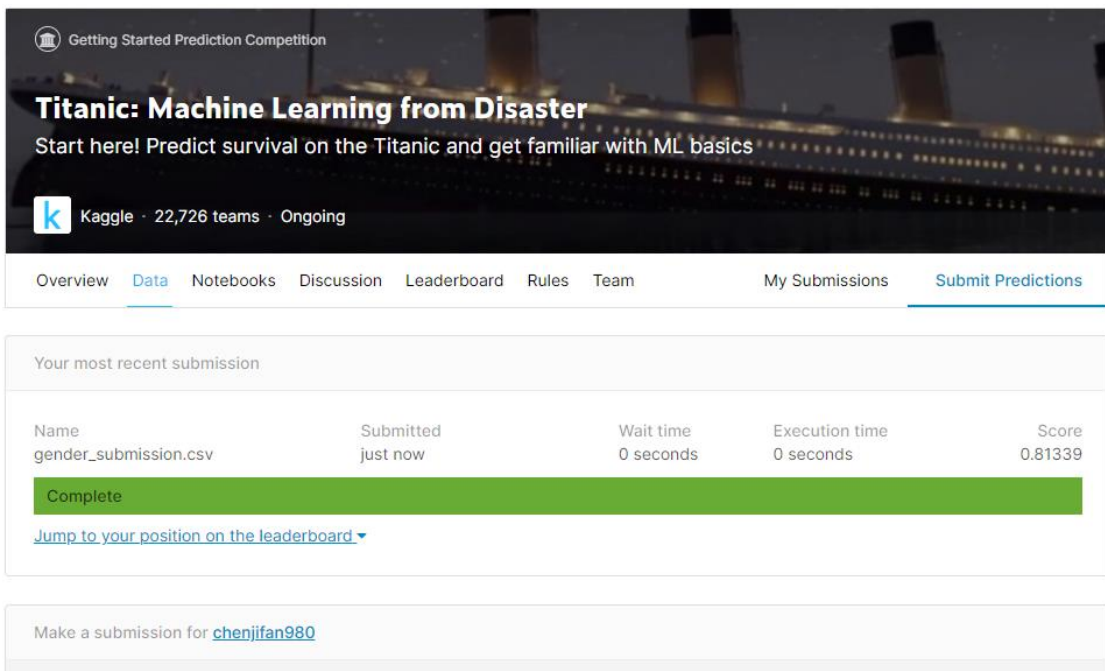
### A. 得到的文件

两个特征集:  train\_prod2.csv  test\_prod2.csv

一个结果集:  gender\_submission.csv

### B. 得分

**最好得分: 0.81339**



The screenshot shows the Kaggle competition interface for "Titanic: Machine Learning from Disaster". The submission table lists the file "gender\_submission.csv" as submitted "just now" with a score of 0.81339. A green bar indicates the submission is "Complete".

Name	Submitted	Wait time	Execution time	Score
gender_submission.csv	just now	0 seconds	0 seconds	0.81339

## 七、遇到的困难和解决方法

1. 问题: 预处理数据时不会将字符串类型转为相应的数字类别。

---

**解决：**查看很多 python 相关语法函数和 panda 的函数，也进行了多次尝试最后得以使用到了 list,set,dict, sorted 和 zip 来较好的实现。

2. **问题：**对类别数据理解不通。一开始多次使用 OHE 没有成效，不知原因。

**解决：**查阅 Kaggle 问答等论坛，查阅官方文档，深入学习各模型的特点，理清了基于树的模型和类别数据之间的关系，最后采用 LGB 的 Categorical 功能。

## 八、总结

**总结：**这次作业我完整体验了一次机器学习实现数据挖掘的过程，学习了很多知识，包括：各类模型的特点和内部的一些算法实现，如何调参，如何选择特征，以及提取特征的技巧，同时也我也对 python 语言的各种操作更加的熟悉了，总之，“炼丹”的过程很有趣，结果很充实。

**可以提高的地方：**

1. 可以更精确的对每列进行缺失值填充，同时也不仅仅考虑平均值/众数。
2. 不只使用 lightGBM 模型，可以考虑加上 LR 或 NN。
3. 可考虑加上对 Name 的信息提取，例如：姓氏，称呼前缀等，而不是直接删去。

## 九、心得

上这门课程之前我对数据挖掘理解为 0，python 基础也为 0，但是通过一点点不断研究的资料，我学习了很多数据挖掘的知识，也学会了 python 这门语言的基本使用，我感受到了它在可视化和表操作上的便利，同时激起了我对数据挖掘相关算法的浓厚兴趣，我现在对这门科学有了一定程度的理解。

在这过程中当然遇到了许多困难，一些知识盲点如 OHE 等利用只能通过网上的资料来学习，并且我一开始在特征提取过程中的表操作方面很吃力，但经过了自己这次代码经验的积累，现在已经很熟悉了，成就感十足。十分感谢老师和 TA 的付出，辛苦了！

## 十、其他



---

## A. 代码展示

附件: ./src/dm.py

Github 地址: <https://github.com/Chenjiff/Titanic-Machine-Learning-from-Disaster-Kaggle->

## B. 文件说明

src 文件夹下是训练的源代码;

data 文件夹下是赛题提供的表格;

res 文件夹下是最终训练并提交的结果;

## C. 参考文献

(<https://www.tinymind.cn/articles/3736>)

(<https://www.kaggle.com/c/talkingdata-mobile-user-demographics>)

([https://blog.csdn.net/huacha\\_/article/details/81057150](https://blog.csdn.net/huacha_/article/details/81057150))

(<https://www.cnblogs.com/jiangxinyang/p/9337094.html>)

(<https://www.cnblogs.com/bambipai/p/7658311.html>)

(<https://blog.csdn.net/Softdiamonds/article/details/80062638>)

(<https://www.jianshu.com/p/95a8f035c86c>)

([https://blog.csdn.net/wqh\\_jingsong/article/details/77896449](https://blog.csdn.net/wqh_jingsong/article/details/77896449))