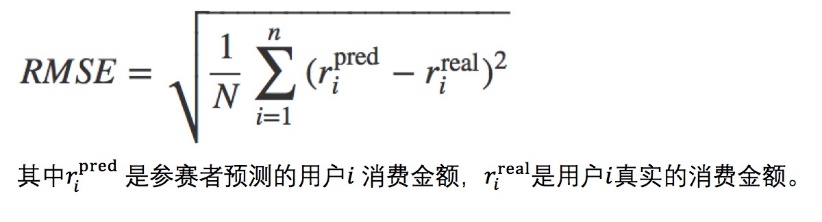
游戏玩家付费金额预测大赛

<http://www.dcjingsai.com/common/cmpt/%E6%B8%B8%E6%88%8F%E7%8E%A9%E5%AE%B6%E4%BB%98%E8%B4%B9%E9%87%91%E9%A2%9D%E9%A2%84%E6%B5%8B%E5%A4%A7%E8%B5%9B_%E7%AB%9E%E8%B5%9B%E4%BF%A1%E6%81%AF.html>

当你选择了一个比赛后，可以先“热热身”，稍微熟悉一下数据，粗略地跑出一些简单的模型，看看自己在榜上的排名，然后再去慢慢迭代。

评分标准 RMSE(root-mean-square error)，即均方根误差。



## 数据预处理

### 先取部分数据，用于快速实验

1减法train=0.1.csv ：训练集的1/10，样本数= 228800

2变换train=0.1-fe.csv ：train=0.1.csv 特征工程后的，注册日期转换为天数

3变换tap\_fun\_test-fe.csv ：tap\_fun\_test.csv 特征工程后的，注册日期转换为天数

4加法train=0.1-fe-top10.csv ：train=0.1-fe.csv 添加重要特征（top 10）的多项式

5加法tap\_fun\_test-fe-top10.csv ：tap\_fun\_test-fe.csv 添加重要特征（top 10）的多项式

6加法train=0.1-fe-top20.csv ：train=0.1-fe.csv 添加重要特征（top 20）的多项式

7加法tap\_fun\_test-fe-top20.csv ：tap\_fun\_test-fe.csv 添加重要特征（top 20）的多项式

8减法train=0.1-fe-coor0.3.csv ：train=0.1-fe.csv 取相关性 > 0.3的特征47个

9减法tap\_fun\_test-fe-coor0.3.csv ：tap\_fun\_test-fe.csv 取相关性 > 0.3的特征47个

10减法train=0.1-fe-coor0.3-pay.csv ：train=0.1-fe-coor0.3.csv 去掉不付费的记录，只保留付费记录

11减法train=0.1-fe-pay.csv ：train=0.1-fe.csv 去掉不付费的记录，只保留付费记录

### 特征工程

* 计算每天的平均值：输入特征/7、目标值/45，预测值\*45
* 计算在线时长平均值：除了付费之外的输入特征/在线时长
* 只训练pay\_price, pay\_count, prediction\_pay\_price
* 长尾分布转化为正态发布：对数变换，对最终结果取反对数。

<https://blog.csdn.net/laolu1573/article/details/80308554>

### 样本平衡问题

利用EasyEnsemble方法，产生多个不同的训练集，训练多个分类器，组合多个分类器的结果得到最终结果。

由于只有1/50的玩家付费=正例，所以把数据集分成50份，每份包含所有的正例和1/50反例。

每个数据集，采用5折交叉验证。

train=0.1-01.csv ~ train=0.1-50.csv

### 最后用全部数据，最佳策略训练

tap\_fun\_train.csv ：整个训练集，样本数= 2288004

## 模型融合

我们使用 greedy blending 策略和两层结合了其他的图像特征的集成学习算法将模型结果融合起来。

确保所有的模型都要经过4折交叉验证，尽管这会增加计算开销，但是却能降低过拟合的风险。使用固定的k-fold 避免Valid Set过拟合。

### 基础学习器 K-折交叉验证

* 线性回归 lasso ：效果最好
* LightGBM：效果不太好
* Elastic Net Regression：
* SVM回归，核函数
* 神经网络回归 <https://blog.csdn.net/roguesir/article/details/79383122>
* Gradient Boosting Regressor

### 次级学习器

LightGBM、Xgboost

### 最终结果：加权平均

# 日程记录

## 2018.06.24 数据探索

### 各个特征的含义

tap4fun数据字段-分析.xls

### 特征之间的相关性

# 升序排列，取 prediction\_pay\_price 相关性 > 0.3 有 48 个特征

| 特征名称 | **pay\_price** | **pay\_count** | **prediction\_pay\_price** |
| --- | --- | --- | --- |
| **sr\_infantry\_atk\_level** | 0.350232 | 0.656094 | 0.320464 |
| **magic\_add\_value** | 0.401094 | 0.349691 | 0.328770 |
| **cavalry\_add\_value** | 0.390930 | 0.538951 | 0.333436 |
| **sr\_rss\_e\_gather\_level** | 0.412443 | 0.431104 | 0.335987 |
| **sr\_cavalry\_atk\_level** | 0.372536 | 0.667096 | 0.341669 |
| **sr\_march\_size\_level** | 0.489823 | 0.204157 | 0.344306 |
| **sr\_outpost\_tier\_4\_level** | 0.489823 | 0.204157 | 0.344306 |
| **sr\_guest\_troop\_capacity\_level** | 0.489823 | 0.204157 | 0.344306 |
| **sr\_shaman\_atk\_level** | 0.387948 | 0.657870 | 0.354319 |
| **sr\_outpost\_tier\_3\_level** | 0.389628 | 0.530950 | 0.390737 |
| **sr\_troop\_attack\_level** | 0.599588 | 0.268091 | 0.403624 |
| **shaman\_add\_value** | 0.526324 | 0.557001 | 0.409337 |
| **infantry\_add\_value** | 0.470065 | 0.564580 | 0.409438 |
| **sr\_gathering\_march\_speed\_level** | 0.564067 | 0.403341 | 0.435076 |
| **sr\_pvp\_march\_speed\_level** | 0.570141 | 0.416220 | 0.442595 |
| **sr\_shaman\_tier\_4\_level** | 0.687182 | 0.333949 | 0.448995 |
| **sr\_alliance\_march\_speed\_level** | 0.617740 | 0.414411 | 0.455331 |
| **sr\_cavalry\_tier\_4\_level** | 0.739613 | 0.383548 | 0.511766 |
| **sr\_cavalry\_tier\_3\_level** | 0.603622 | 0.452134 | 0.522416 |
| **sr\_infantry\_tier\_3\_level** | 0.593288 | 0.453258 | 0.535145 |
| **sr\_shaman\_tier\_3\_level** | 0.595050 | 0.457640 | 0.541889 |
| **sr\_infantry\_tier\_4\_level** | 0.782337 | 0.404025 | 0.563373 |
| **building\_acceleration\_add\_value** | 0.680214 | 0.797693 | 0.589272 |
| **pay\_count** | 0.703309 | 1.000000 | 0.598916 |
| **sr\_troop\_defense\_level** | 0.704995 | 0.425913 | 0.607208 |
| **sr\_cavalry\_hp\_level** | 0.704053 | 0.422663 | 0.607486 |
| **building\_acceleration\_reduce\_value** | 0.703206 | 0.784215 | 0.608038 |
| **sr\_shaman\_def\_level** | 0.758099 | 0.427712 | 0.614711 |
| **sr\_infantry\_def\_level** | 0.728575 | 0.409429 | 0.621730 |
| **sr\_shaman\_hp\_level** | 0.715633 | 0.414688 | 0.622085 |
| **sr\_cavalry\_def\_level** | 0.835937 | 0.451455 | 0.629569 |
| **meat\_add\_value** | 0.792775 | 0.623492 | 0.650798 |
| **reaserch\_acceleration\_add\_value** | 0.798939 | 0.830616 | 0.671408 |
| **sr\_infantry\_hp\_level** | 0.814188 | 0.474725 | 0.681264 |
| **training\_acceleration\_add\_value** | 0.815864 | 0.743238 | 0.686599 |
| **reaserch\_acceleration\_reduce\_value** | 0.834280 | 0.782629 | 0.695218 |
| **meat\_reduce\_value** | 0.858870 | 0.713156 | 0.700602 |
| **training\_acceleration\_reduce\_value** | 0.844732 | 0.704375 | 0.712416 |
| **wood\_reduce\_value** | 0.870434 | 0.725854 | 0.712524 |
| **general\_acceleration\_add\_value** | 0.886887 | 0.793226 | 0.725020 |
| **wood\_add\_value** | 0.891059 | 0.684707 | 0.732175 |
| **stone\_add\_value** | 0.905230 | 0.634133 | 0.737585 |
| **general\_acceleration\_reduce\_value** | 0.895368 | 0.762830 | 0.740558 |
| **ivory\_reduce\_value** | 0.911758 | 0.603684 | 0.743139 |
| **ivory\_add\_value** | 0.909818 | 0.643655 | 0.746421 |
| **stone\_reduce\_value** | 0.924619 | 0.672541 | 0.747822 |
| **pay\_price** | 1.000000 | 0.703309 | 0.803923 |
| **prediction\_pay\_price** | 0.803923 | 0.598916 | 1.000000 |

### 数据集的特别之处

类似 CTR 数据集。

* 1 数据量大，数据非常稀疏。
* 2 负样本数目 >> 正样本数目

样本总量 228800

pay\_count 属性的不同取值和出现的次数

0 224599 一次都没有付费的最多，占 98.16%

1 1786

2 792

3 491

4 309

5 209

6 159

7 100

8 77

9 57

11 39

10 35

如何解决样本不平衡问题？利用EasyEnsemble的方法，即多次下采样（放回采样，这样产生的训练集才相互独立）产生多个不同的训练集，进而训练多个不同的分类器，通过组合多个分类器的结果得到最终的结果。

* 3 所有特征分布都是长尾，付费低的多而集中，付费高的少而分散。

<http://wiki.mbalib.com/wiki/%E9%95%BF%E5%B0%BE%E7%90%86%E8%AE%BA>

“长尾”案例：Google、亚马逊，涓涓细流，汇聚成河。

只要存储和流通的渠道足够大，需求不旺或销量不佳的产品共同占据的市场份额，就可以和那些数量不多的热卖品所占据的市场份额相匹敌，甚至更大。

所有特征值，取倒数，只使用1/10000以上的。

干脆去掉7日不付费的记录。训练个模型看看。

* 4 sr\_rss\_help\_bonus\_level 这个特征值，全是 0

训练集里面只有3个玩家 sr\_rss\_help\_bonus\_level > 0 而且是付费的

测试集里面也有3个玩家 sr\_rss\_help\_bonus\_level > 0 应该是付费的

## 2018.06.25 快速实验，用部分数据

**先用最小的数据集训练，看看效果。再逐渐扩大，比较效果。**

1. train=0.1-fe-coor0.3-pay.csv ：只保留付费记录，

对应测试集 tap\_fun\_test-fe-coor0.3.csv

* 无论是否标准化 train\_y ，LinearRegression、岭回归得到的预测结果相同
* 标准化 train\_y 后，Lasso回归得到的预测结果更好。
* **结论：标准化 train\_y 更好。**

LinearRegression summit0.1-coor0.3-pay-Linear.csv **= 97.8753**

岭回归 summit0.1-coor0.3-pay-ridge.csv **= 86.01859**

Lasso回归 标准化train\_y summit0.1-coor0.3-pay-lasso.csv **= 82.12618**

Lasso回归 不标准化train\_y summit0.1-coor0.3-pay-lasso\_1.csv **=** **91.08**

LightGBM 默认参数 summit0.1-coor0.3-pay-lgbm\_1.csv =99.10\*\*\*4

参数调优后 summit0.1-coor0.3-pay-lgbm-2.csv = 91.57\*\*\*5

1. train=0.1-fe-coor0.3.csv ：取相关性 > 0.3的特征47个

对应测试集 tap\_fun\_test-fe-coor0.3.csv

Lasso回归 summit0.1-coor0.3-lasso-1.csv **= 78.14601**

岭回归 summit0.1-coor0.3-RidgeCV-1.csv **= 86.35\*\*9**

1. train=0.1-fe-pay.csv ：只保留付费记录

对应测试集 tap\_fun\_test-fe.csv

Lasso回归 summit=0.1-fe-pay-lasso-1.csv **= 82.14\*\*2**

岭回归 summit=0.1-fe-pay-RidgeCV-2.csv **= 85.58\*\*7**

1. train=0.1-fe.csv ：特征工程后的

对应测试集 tap\_fun\_test-fe.csv

Lasso回归 summit=0.1-fe-lasso-1.csv **= 78.00881**

岭回归 summit=0.1-fe-RidgeCV-3.csv **= 86.00\*\*3**

ElasticNetCV summit=0.1-fe-ElasticNetCV-1.csv **=？？**

1. train=0.1-fe-top10.csv ：添加重要特征（top 10）的多项式

对应测试集 tap\_fun\_test-fe-top10.csv

Lasso回归 summit=0.1-fe-top10-lasso-1.csv **= 78.61\*\*\*5**

岭回归 summit=0.1-fe-top10-RidgeCV-1.csv **= 82.50\*\*5**

1. train=0.1-fe-top20.csv ：添加重要特征（top 20）的多项式

对应测试集 tap\_fun\_test-fe-top20.csv

Lasso回归 summit=0.1-fe-top20-lasso-1.csv **= ？？**

岭回归 summit=0.1-fe-top20-RidgeCV-1.csv **= ？？**

# 训练策略

## 快速实验，用部分数据

寻找最佳训练策略

## 训练基本学习器，用全部数据

使用最佳训练策略

## 训练次级学习器

模型融合：基础学习器

## 模型融合：次级学习器

采用加权平均，权重取二级模型评分。

* 注意 sr\_rss\_help\_bonus\_level > 0 的玩家是付费的。
* 训练集的玩家注册时间长118，测试集的时间短89，预测值按比例118/89看看。
* 通过用户注册账户后7天之内的游戏数据，预测用户在45天内的消费金额

# 参考资料

## 分享一个省内存的方法

这个比赛的训练集如果直接用pd.read\_csv来读的话，大概会占用2GB的内存，这样跑起来就慢的要死了（电脑配置差），连数据探索都吃力。

简单查看了一下数据，发现直接read\_csv的话，很多数值型的数据都是float64或者int64来存储的，但事实上很少数据需要用这么大的数据来存储，所以我们可以用https://www.jiqizhixin.com/articles/2018-03-07-3提到的方法来将int64和float64转换成内存占用较小的类型，比如float32之类的，缩减完后，训练集大概600多MB，测试集大概170多MB。

我们只需要记录下缩减完后的每个列的列名和数据类型，下次读取的时候通过指定dtype参数即可。

## Kaggle实战之一回归问题

<https://www.cnblogs.com/daniel-D/p/7868968.html>

是不是做聚类之后，计算到聚类中心的距离也是一个很好的特征呢？

* **偏度处理**：在大部分数值型特征中，通常分布不符合正态分布，而是类似于“长尾”。例如房价中，低价占大部分，豪宅属于小部分。应对这种数据分布，一般可以通过神奇的log化处理转化类近似正态分布。

#### Ensembling

* 用 ElasticNet、KernelRidge、 GradientBoostingRegressor 做底层模型，用 lasso regression 来得到每个模型的权重，Stacking models RMSE loss：0.07815
* 用 XGBRegressor 单模型得分RMSE loss：0.07851
* 用 LGBMRegressor 单模型得分RMSE loss： 0.07194
* 最后拍个权重，求平均：stacked\_pred\*0.70 + xgb\_pred\*0.15 + lgb\_pred\*0.15 得到最终预测RMSE loss： 0.07524

## 【干货】Kaggle 数据挖掘比赛经验分享

<https://blog.csdn.net/buptgshengod/article/details/71713659>

<https://blog.csdn.net/liushuikong/article/details/72630230>

* 对于稀疏型特征，一般使用线性模型，譬如 Linear Regression 或者 Logistic Regression。Random Forest 和 GBDT 等树模型不太适用于稀疏的特征。
* 对于稠密型特征，推荐使用 XGBoost 进行建模，简单易用效果好；
* 数据中既有稀疏特征，又有稠密特征，可以考虑使用线性模型对稀疏特征进行建模，将其输出与稠密特征一起再输入 XGBoost/DNN 建模。

## 数据预处理-对数变换

<https://blog.csdn.net/laolu1573/article/details/80308554>

不会做特征工程的 AI 研究员不是好数据科学家！

<https://www.leiphone.com/news/201801/T9JlyTOAMxFZvWly.html>

Log变换：fcc\_survey\_df['Income\_log'] = np.log((1+fcc\_survey\_df['Income']))

## 大数据时代下，如何挖掘长尾应用

<http://column.iresearch.cn/b/201511/750097.shtml>

石榴掌游也是一个背后依托强大机器学习算法的应用，它可以根据用户手机的自有应用预测用户可能喜欢的应用，也可以根据用户的浏览和下载记录判断用户的喜好，从而为用户提供更精准的推荐。

## 长尾数据如何进行回归分析？

<https://www.zhihu.com/question/41871926>

做多元分析的时候，自变量和因变量的分布不需要考虑。

如果线性回归拟合情况不理想，可以试试添加交叉项或者幂函数项，以及其他非线性的函数。你也可以试试神经网络。

## 神经网络回归

<https://blog.csdn.net/weixin_38195506/article/details/75302445>

解决回归问题的神经网络一般只有一个输出节点，这个节点的输出值是预测值。

mse=tf.reduce\_mean(tf.square(y\_-y)),其中y代表了神经网络的输出答案，y\_代表了标准答案。

<https://deeplearning4j.org/cn/linear-regression>

神经网络与回归

<https://blog.csdn.net/roguesir/article/details/79383122>

【Tensorflow】Python实现神经网络回归

<https://blog.csdn.net/roguesir/article/details/79383122>

## Elastic Net Regression

你应该掌握的七种回归技术

<https://www.csdn.net/article/2015-08-19/2825492>

弹性网络（ Elastic Net）

<https://blog.csdn.net/qq_21904665/article/details/52315642>

scikit-learn学习：elastic net

<https://blog.csdn.net/u014662865/article/details/55251790>

LinearRegression,RidgeCV,LassoCV,ElasticNetCV各自使用场景

<https://blog.csdn.net/hb707934728/article/details/69676833>

## Gradient Boosting Regressor

Gradient Boosting算法简介

<https://www.cnblogs.com/zhubinwang/p/5170087.html>

scikit-learn : GBR (Gradient boosting regression)

<https://blog.csdn.net/SA14023053/article/details/51817650>