中 国 地 质 大 学

研究生课程报告

课 程 名 称：机器学习方法与应用

教 师 姓 名：刘袁缘

研究生姓名：郑康、徐鸿飞、张泽岩、梅杭

所 在 院 系：地理与信息工程学院

日 期：2021 年 11 月 28 日

PUBG Finish Placement Prediction

1 背景介绍

1.1 Kaggle 简介

Kaggle 是一个为数据分析爱好研究者提供举办机器学习竞赛、托管数据库、编写和分享代码的平台。对于数据分析和机器学习研究来说，数据集是十分重要的，Kaggle 为参赛者提供了各种各样高质量的数据集，基于这些数据集来举办一些算法的竞赛，比赛期间参赛人员还能够通过社区进行讨论与分享。除了高质量的数据集和开放的交流平台，Kaggle 还为排名靠前的队伍提供一定的奖励，这样的机制吸引了全球众多的相关领域的研究人员与爱好者。

Kaggle 上的比赛很多，可以主要分为以下两大类：

（1）竞赛型：竞赛型的比赛的目的十分明确，该类型比赛要求参赛人员在一定的时间段内（通常是 90 天左右）使用平台提供的数据完成相应的任务。参赛人员可以自行组队，平台最终会根据队伍解决方案的排名给予一定的奖励，队伍除了可以获得奖金之外，还可能会受到赞助商的邀请进入企业工作或者参加一些重大会议。

（2）数据集型：数据集型比赛是赞助商为了解决某些问题，在平台上免费发布了内部的一些脱敏的数据，平台上的所有人员都被允许使用这些数据来进行相关研究，从而对目前的模型进行一定的改进或对现有的解决方案进行优化，但是改进后的模型与解决方案是能够不公开的，因此此类比赛没有奖金，仅仅可以与其他的模型或方案进行对比。

1.2 问题背景

PUBG 是一款十分火热的射击生存类游戏，在 PUBG 游戏中，每场比赛玩家数量最多为 100 名，每场比赛玩家最终的排名取决于玩家在被淘汰时仍存在玩家或者队伍的数量。玩家在进入游戏后，可以收集不同的装备和武器来攻击敌人，在受到伤害后也可以使用收集到的医疗品恢复自身的血量，也能够救起被击倒但是没有被杀死的队友，在游戏过程中，玩家还可以开车前往目的区，开船或者游泳来度过河流，逃跑与追击方式十分丰富。为了在比赛中取得更好的名词，玩家需要去收集更高质量的武器、装备以及医疗品，以便于消灭敌人，躲避不断缩小的“毒圈”。因此，如果玩家想要取得一个好的名次，必须要考虑许多因素，例如，合适的跳伞位置能够帮助玩家更迅速地收集装备，恰当的转移路线可以帮助玩家更好地躲避敌人的攻击。

1.3 问题分析

本题中平台提供了大量匿名的 PUBG 游戏统计数据，其格式设置为每行包含一个玩家的游戏后统计数据，需要创建一个模型，根据平台提供的最终统计数据预测玩家的相应排名。鉴于先前玩家在每场比赛期间的统计数据和训练数据中的排名信息，我们需要训练模型根据每场比赛期间的统计数据预测测试组中球员的排名。目标标签排名将是 0 到 1 之间的百分比值，更高的百分比表示该匹配中的更高排名。在这个过程中，我们将分析诸多因素对获胜的影响。

2 数据分析与处理

2.1 数据介绍

题目给出的数据如表 1 所示：

表 1 所有数据及说明表

DBNOs	玩家击倒的敌人数量
assists	玩家造成伤害且被队友所杀死的敌人数量
boosts	玩家使用的增益性物品数量
damageDealt	玩家造成的总伤害-玩家所受的伤害
headshotKills	玩家通过爆头杀死的敌人数量
heals	玩家使用的救援类物品数量
Id	玩家的 ID
killPlace	玩家杀死敌人数量的排名
killPoints	基于杀戮的玩家外部排名。
killStreaks	玩家在短时间内杀死敌人的最大数量
kills	玩家杀死的敌人的数量
longestKill	玩家和玩家在死亡时被杀的最长距离。
matchDuration	比赛时间
matchId	比赛的 ID
matchType	单排/双排/四排；标准模式是“solo”,“duo”,“squad”,“solo-fpp”,“duo-fpp”和“squad-fpp”；其他模式来自事件或自定义匹配。
rankPoints	类似 Elo 的玩家排名。
revives	玩家救援队友的次数
rideDistance	玩家使用交通工具行驶了多少米
roadKills	玩家在交通工具上杀死敌人的数目
swimDistance	玩家游泳的距离

teamKills	该玩家杀死队友的次数
vehicleDestroys	玩家毁坏的交通工具数目
walkDistance	玩家步行距离
weaponsAcquired	玩家捡枪数量
winPoints	基于赢的玩家外部排名。
groupId	队伍的 ID
numGroups	在该局比赛中有玩家数据的队伍数量
maxPlace	在该局中已有数据的最差的队伍名次
winPlacePerc	预测目标，是以百分数计算的，介于 0-1 之间，1 对应第一名，0 对应最后一名。 它是根据 maxPlace 计算的，而不是 numGroups，因此匹配中可能缺少某些队伍。

2.2 常用 EDA 方法与步骤

2.2.1 数据质量分析

（1）缺失值分析

首先看数据缺失的类型，分为三种方式：完全随机缺失、随机缺失和非随机缺失，三者之间的主要区别是缺失的值与其他的变量的关系，完全随机缺失不依赖于其他的变量，而随机缺失与非随机缺失分别依赖于其他完全变量与不完全变量。对于缺失值，直接删除记录是不合适的，随机缺失可以通过已知的其他变量进行估计，而完全随机缺失与非随机缺失还没有好的解决方式。

对于缺失值的处理主要有丢弃、补全、真值转化和不处理四种方法。对于不同类型的数据使用不同的方法：对于连续型的数据，可以直接删除，也可以进行填充，对于缺失值少的，可以使用固定值插补、均值插补、中值插补、最近数据插补、回归插补、拉格朗日插值、牛顿插值、分段插值、K-means、KNN 等方法进行插补，对于缺失值多的可以使用回归预测、极大似然估计、多重插补、随机森林等方法进行拟合，也可以使用衍生法或者直接不处理；对于类别型值，若缺失值少，如果缺失值没有业务含义，可以使用众数即最多的数进行填充，否则填充默认值，若缺失值多则应该剔除缺失率大于百分之八十并且之后还是会缺失的变量，若缺失值适中可以将缺失当作新的类来处理；对于时间类型可以通过其他变量来估计时间。

（2）异常值分析

首先要查看异常情况，主要方法有：画出数据散点图，观察是否有偏差很大的数据再判断是否异常值；画出箱形图，如果数据是服从正态分布的则超过平均值的三倍标准差的值则可能异常；另外还有基于模型预测、KNN 离群点检测、基

于密度的离群点检测、K-means 聚类等方法。然后要对异常值进行处理，主要有四种方式：直接当作缺失值来处理、直接删除、不处理和进行平均值修正。

2.2.2 数据特征分析

这里都是一些多元统计分析中常见的方法。

（1）分布

可以绘制数据对频率直方分布图、饼图和条形图等来查看数据的分布，以及观察样本分布是否偏斜，计算分布等偏度、峰度。

（2）对比

主要有绝对数对比和相对数对比两种方式。

（3）常见统计量

常见统计量主要分为两种，第一种是集中趋势度量，有均值、中位数、众数，另一种是离中趋势度量，有标准差、极差、变异系数等。

（4）周期性

根据帕累托法则，也被称为 80/20 法则、关键少数法则、八二法则来判断，约仅有 20%的因素影响 80%的结果。也就是说：所有变因中，最重要的仅有 20%，虽然剩余的 80%占了多数，影响的幅度却远低于“关键的少数”。

（5）相关性

判断连续变量间线性相关的程度，可以使用绘制散点图和三点钟矩阵、计算相关系数（Pearson product-moment correlation coefficient、spearman 相关系数、判定系数）等方法。

2.3 特征处理与分析

采用单独分析与综合分析相结合的方式，对诸多因素进行筛选，选出最重要的影响因素，并分析其对最终排名的影响，当然我们首先进行了单变量、双变量的数据探索性分析，得到诸多结论，并结合这些结论进行多变量数据探索分析，得到了我们最终的模型。

数据中除去 ID 三列外，其他数据都是数字类型，所以可以考虑进行内存数据压缩，这样可以减少内存消耗以及加快运算速度。

通过图 2.1 可以看到将训练数据通过这个函数进行压缩后减少了 70.7%的内存占用。

通过图 2.2 可知，大多数人杀敌数是为 0 的，部分杀敌数为 1 和 2，极少数能够杀死更多的敌人。这也是和我们自己的游戏经验所符合的。

```
In [2]:
def reduce_mem_usage(df):
    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))
    for col in df.columns:
        col_type = df[col].dtype
        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
    end_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))
    return df

In [43]:
train_data = reduce_mem_usage(train_data)

Memory usage of dataframe is 983.90 MB
Memory usage after optimization is: 288.39 MB
Decreased by 70.7%
```

图 2.1 数据压缩

```
plt.rcParams['figure.facecolor'] = 'azure'

data = train_data.copy()
data.loc[data['kills'] > most_kill] = "More"
plt.figure(figsize=(15,10))
sns.countplot(data['kills'].astype("str").sort_values())
plt.title("Kill Count", fontsize=15)
plt.show()
# 显然，我们通过该图象可知，大多数人杀敌数是为0的，部分杀敌数为1和2，极少数能够杀死更多的敌人。这也是和我们自己的游戏经验所符合的，
# 由于大多数人没有杀敌数，因此，我们认为对伤害量分析是很有必要的。
```

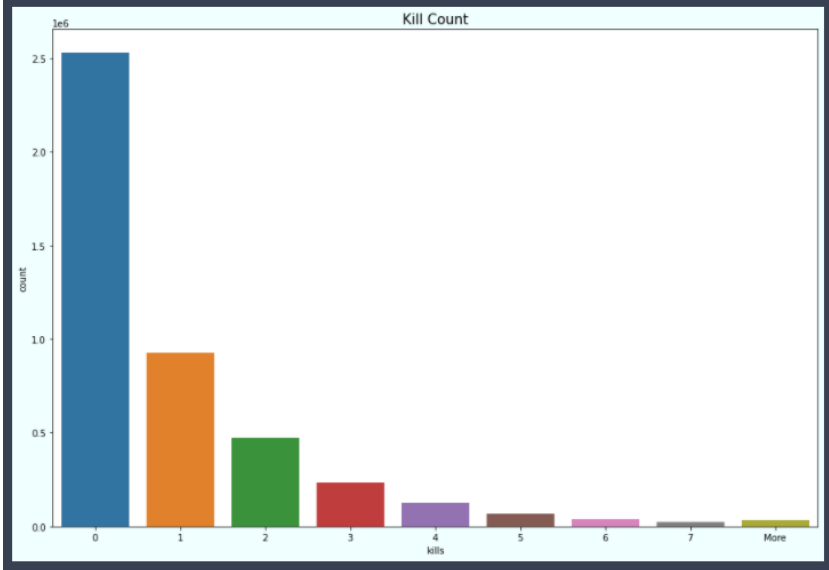

图 2.2 杀敌数量统计

```
# 伤害量
# damageDealt 在这里定义为 玩家造成的总伤害 - 玩家所受的伤害 ，damageDealt为一个连续量，取值较为分散，故在此处我们将使用直方图的方式来形象地表示该数据，
data = train_data.copy()
plt.figure(figsize=(15,10))
plt.title("Damage Dealt ",fontsize=15)
sns.distplot(data['damageDealt'])
plt.show()
#直方图：先分箱，然后计算每个分箱频数的数据分布，
# 和条形图的区别，条形图有空隙，直方图没有，条形图一般用于类别特征，直方图一般用于数字特征（连续型）
# 多用于y值和数字（连续型）特征的分布画图
```

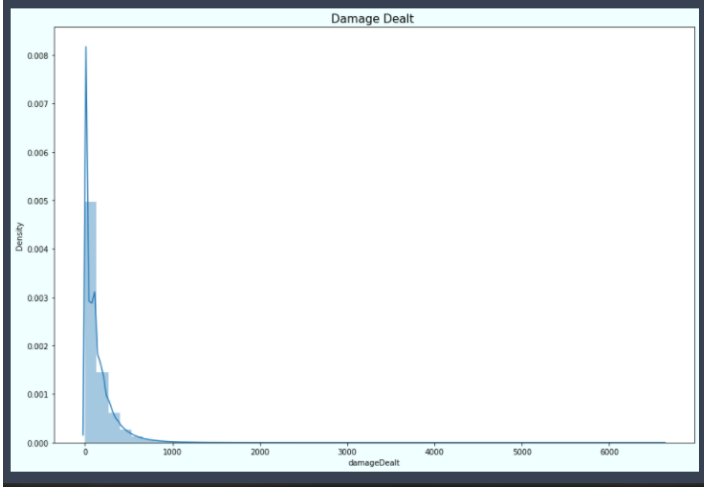

图 2.3 伤害量统计

通过对伤害量进行作图 2.3 可以看出过大部分玩家的伤害量也是趋于 0。

```
[47]:
# 我们再具体查看杀敌0人的玩家造成的伤害
data = train_data.copy()
data = data[data['kills']==0]
plt.figure(figsize=(15,10))
plt.title("Damage Dealt by 0 killers",fontsize=15)
sns.distplot(data['damageDealt'])
plt.show()
```

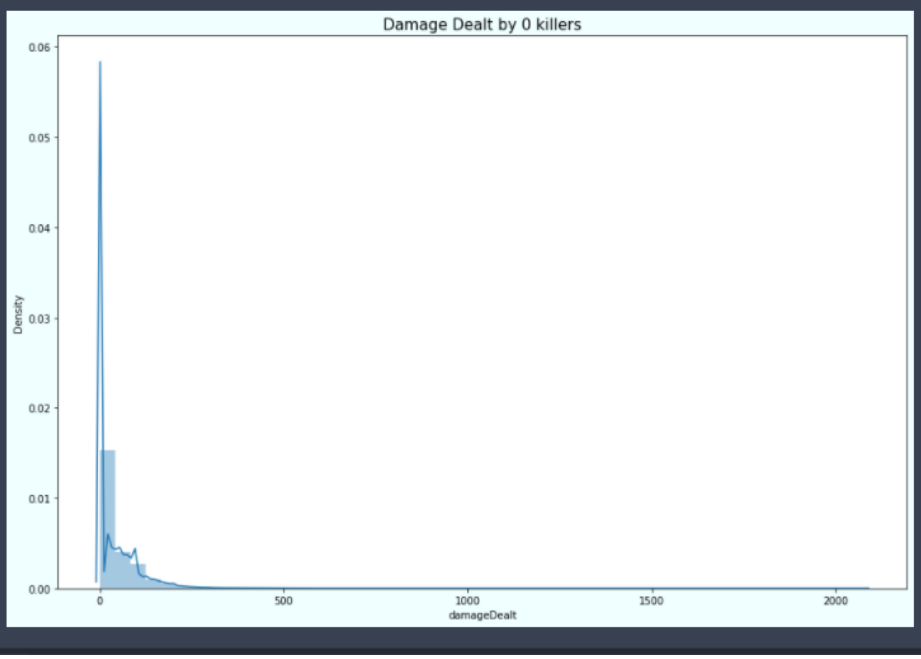

图 2.4 杀敌数量 0 玩家的伤害量

原计划分别计算最终胜利（第一名）的人平均伤害量以分析，以表示伤害量高的人胜利概率最大，但这里数值过于庞大，难以在可接受时间内计算出，故我们转换思路，我们查看伤害量小于 0 的人以及杀敌为 0 的人能够取胜的概率，反向推理出伤害量和杀敌数与取胜的密切相关性。

```
data = train_data[train_data['kills'] == 0].copy()
print("{} players ({:.4f}%) have won without a single kill!".format(len(data[data['winPlacePerc']==1]), 100*len(data[data['winPlacePerc']==1])/len(train_data)))

data1 = train_data[train_data['damageDealt'] == 0].copy()
print("{} players ({:.4f}%) have won without dealing damage!".format(len(data1[data1['winPlacePerc']==1]), 100*len(data1[data1['winPlacePerc']==1])/len(train_data)))
```

```
16666 players (0.3748%) have won without a single kill!
4770 players (0.1073%) have won without dealing damage!
```

图 2.5 是否吃鸡与杀敌数、伤害量统计

从图 2.5 的数据可以看出，玩家是否能取胜与杀敌数和伤害量关系密切。

```
# 步行距离
print ("平均步行距离为：{:.0f}".format(train_data['walkDistance'].astype('float32').mean()))
print ("99%的步行距离小于该值：",train_data['walkDistance'].quantile(0.99))
print ("数据集中最大的步行距离：",train_data['walkDistance'].max())
```

```
平均步行距离为：1154
99%的步行距离小于该值： 4396.0
数据集中最大的步行距离： 25780.0
```

图 2.6 平均步行距离

```
data = train_data[train_data['walkDistance'] < train_data['walkDistance'].quantile(0.99)]
plt.figure(figsize=(15,10))
plt.title("The Walking Distance")
sns.distplot(data['walkDistance'])   #distplot直方图
plt.show()
```

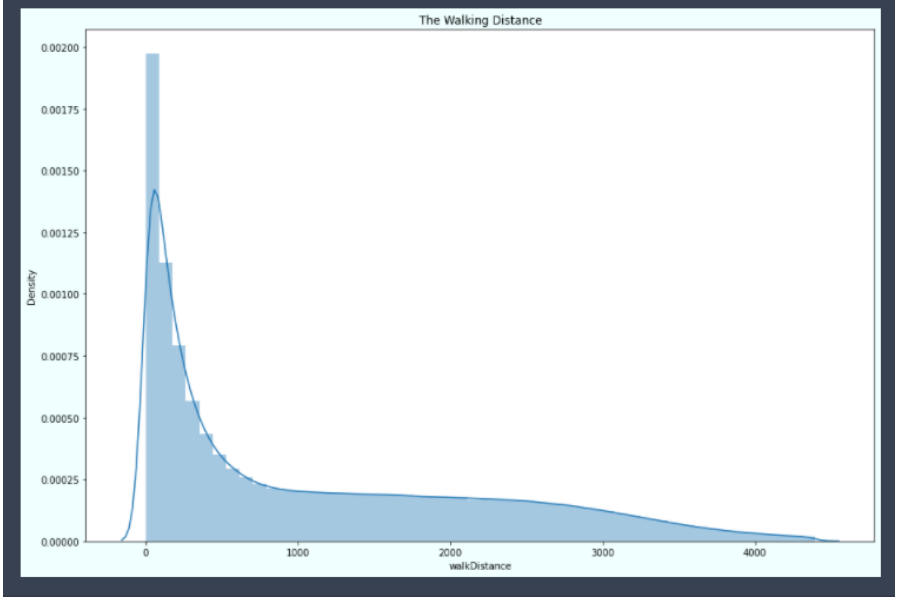

图 2.7 步行距离统计

步行距离是指玩家行走距离，PUBG 游戏随着游戏的进行会逐渐缩小安全区域，逼迫存活的玩家不断地移动，通常来说，一局中排名越高的玩家，步行距离往往越大，行走距离较多的玩家偏向于更大概率完成吃鸡，我们将用数据说明这一点。

```
winner_data=train_data[train_data['winPlacePerc'] == 1].copy()
winner_data = winner_data[winner_data['walkDistance'] < winner_data['walkDistance'].quantile(0.99)]
plt.figure(figsize=(15,10))
plt.title("The winner's walking distances")
sns.distplot(winner_data['walkDistance'])   #distplot直方图
plt.show()
# 很显然，我们可知最终赢家通常步行距离在3000m左右，也就是较高的步行距离
```

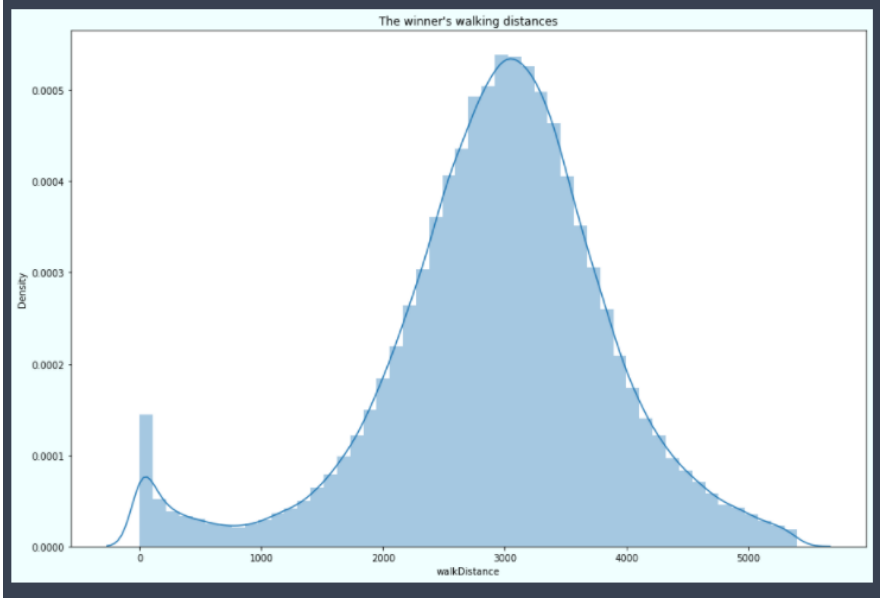

图 2.8 吃鸡玩家步行距离

```
zero_meter_num = len(train_data[train_data['walkDistance'] == 0])
print(f'{zero_meter_num} players walk 0m, the rate is {zero_meter_num/len(train_data) * 100:.3f}%')
sns.jointplot(x="winPlacePerc", y="walkDistance",  data=train_data, height=10, ratio=3, color="lime")
plt.show()
```

```
99603 players walk 0m, the rate is 2.240%
```

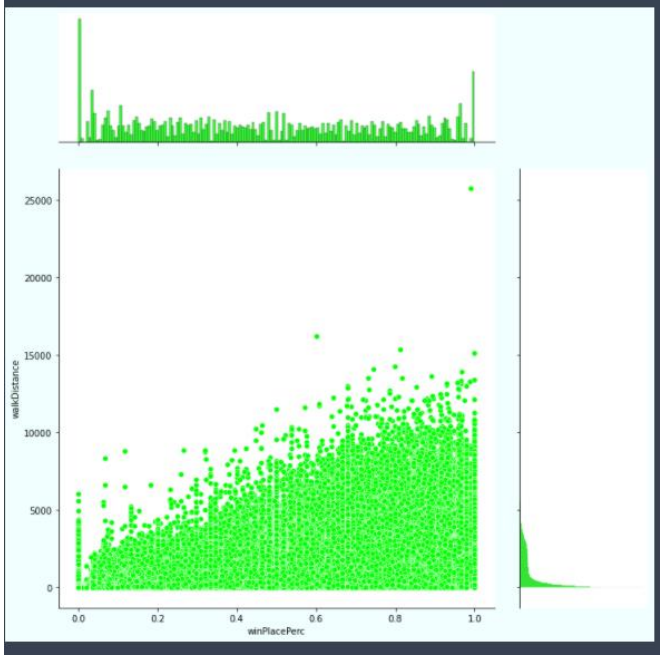

图 2.9 步行距离与最终排名

从图2.8可知最终排名通常步行距离在3000m左右，也就是较高的步行距离。从图2.9可知步行距离与最终排名关系密切。

```
# 增益性和治疗性物品
new_data = train_data.copy()
new_data = new_data[new_data['heals'] < new_data['heals'].quantile(0.99)]
new_data = new_data[new_data['boosts'] < new_data['boosts'].quantile(0.99)]
f, ax1 = plt.subplots(figsize = (10, 6))
sns.lineplot(x = 'heals', y = 'winPlacePerc', data = new_data, color = 'red')
sns.lineplot(x = 'boosts', y = 'winPlacePerc', data = new_data, color = 'blue')
plt.text(5, 0.45, 'Heals', color = 'red', style = 'italic')
plt.text(5, 0.5, 'Boosts', color = 'blue', style = 'italic')
plt.xlabel('Number of heal/boost items',fontsize = 15,color='blue')
plt.ylabel('Win Percentage',fontsize = 15,color='blue')
plt.title('Heals vs Boosts',fontsize = 20,color='blue')
plt.grid()
plt.show()
del new_data
```

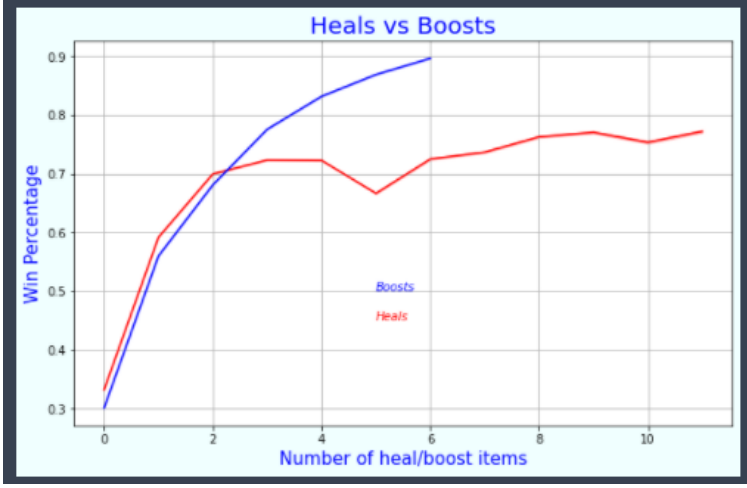

图 2.10 吃鸡概率与增益性、治疗性物品使用

图 2.10 可看出增益性和治疗性物品与吃鸡概率是正相关的，也就是能够最终吃鸡的玩家通常都有着较多的增益性和治疗性物品使用。我们将通过散点图描绘该关系，并进一步分析。

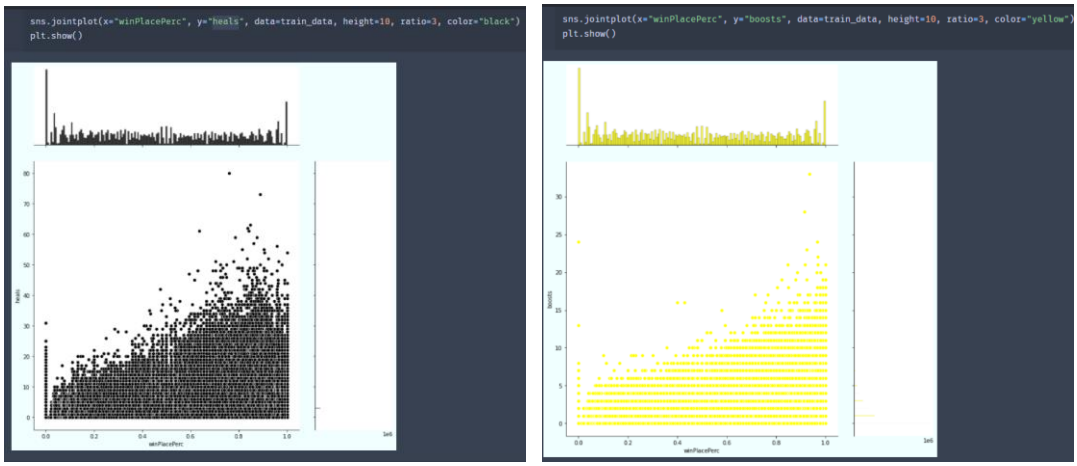

图 2.11 治疗性和增益性物品统计

从图 2.11 两幅图可看出排名较高的玩家，通常在游戏中有着较多治疗性物品。对于增益性物品，则更为明显。故我们可以下普遍性结论：获得更多的治疗性和增益性物品的玩家可以取得更好的名次，当然存在部分值异常。

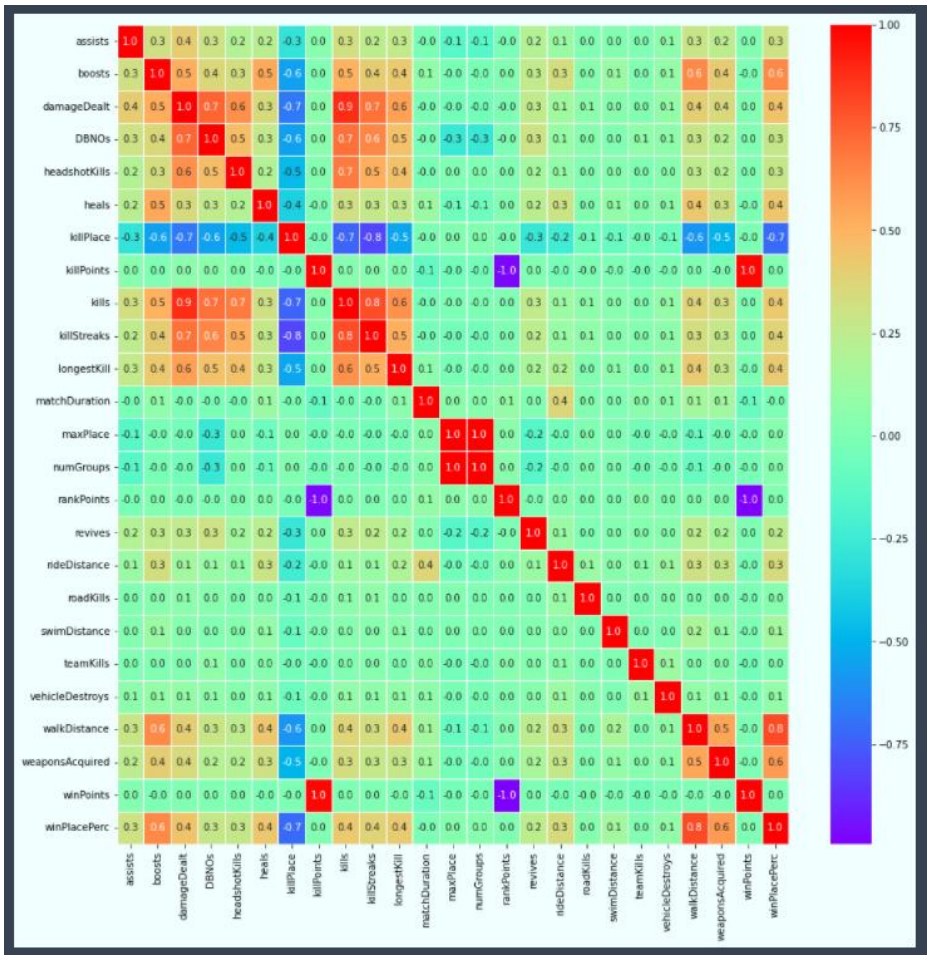

图 2.12 变量相关性热力图

我们也对其他一些特征进行了相同的分析，还构建了如图 2.12 显示的变量相关性热力图。

我们的目标变量为 winPlacePerc，根据热力图我们发现步行距离对 winPlacePerc 有着最大的正相关影响，而 killPlace（玩家杀死敌人数量的排名）则有着最大的负相关影响。

为了更为详细的分析，我们之后选取正相关影响大于等于 0.4 的部分进行分析，共 8 个变量，分别为 boosts、damageDealt、heals、kills、killStreaks、longestkill、walkDistance、weaponsAcquired，并进行了进一步的变量相关性分析。

3 模型选择与实验

3.1 特征筛选

首先是编写特征构建函数，该函数负责对训练数据进行读取、清洗、无用特征删除、增加新的特征等功能。

通过这个函数对我们将前面分析时发现的与要预测值相关性低且不影响其他特征的特征进行丢弃，并增加了包括每个特征的最高、最低、平均值等特征。

3.1 线性模型

首先我们选取了线性模型作为基线。通过前面的特征构建函数获取特征以及目标。

线性模型指的是一类可以用线性形式进行表达的模型的统称，假设一个对象可以用 n 个属性（x_1、x_2、x_3、…、x_n）进行描述，它的线性模型的基本形式是：

$$f(x) = w_1x_1 + w_2x_2 + w_3x_3 \ldots + w_nx_n + b,$$

其中x_i是第 i 个属性的取值，w_i是第 i 个属性的参数。

线性模型用矩阵形式表示是：

$$f(x) = w^Tx + b,$$

其中$w = (w_1, w_1, w_1, \ldots, w_n)$。

线性模型的预测值用$f(x_i) = w^Tx_i + b$表示，真实值用$y(x_i)$表示，线性模型的目的是使$f(x_i)$的值尽可能接近于$y(x_i)$。根据公式可以看出，线性模型的确定首先需要确定w和b，一般使用最小二乘法来确定线性模型的参数。利用均方误差来度量$f(x_i)$和$y(x_i)$之间的接近程度，让均方误差达到最小，即：

$$\begin{aligned}(w^*, b^*) &= arg\min_{(w,b)} \sum_{i=1}^{n} (f(x_i) - y_i)^2 \\ &= arg\min_{(w,b)} \sum_{i=1}^{n} (y_i - wx_i - b)^2\end{aligned}$$

将均方误差分别对 w 和 b 进行求导，即：

$$\frac{\partial E_{(w,b)}}{\partial w} = 2\left(w\sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n}(y_i - b)x_i\right),$$

$$\frac{\partial E_{(w,b)}}{\partial b} = 2\left(nb - \sum_{i=1}^{n}(y_i - wx_i)\right),$$

令上述两式等于零，就可以求得 w 和 b 的最优闭式解，即：

$$w = \frac{\sum_{i=1}^{n} y_i \left(x_i - \frac{1}{n} \sum_{i=1}^{n} x_i \right)}{\sum_{i=1}^{n} x_i^2 - \frac{1}{n} \left(\sum_{i=1}^{n} x_i \right)^2},$$

$$b = \frac{1}{n} \sum_{i=1}^{n} (y_i - w x_i),$$

利用线性模型得到的实验结果如图 3.1 所示，从实验结果中可以看出线性模型在训练集上的得分为 0.949，我们还对预测的数据进行了可视化，可视化结果如图 3.2 所示。从图中可以看出预测值与 Ground Truth 基本符合。

```
from sklearn.linear_model import LinearRegression
LR_model = LinearRegression(n_jobs=4, normalize=True)
LR_model.fit(X_train,y_train)
```

```
LinearRegression(n_jobs=4, normalize=True)
```

```
LR_model.score(X_train,y_train)
```

```
0.9491910621622454
```

图 3.1 线性模型实验结果

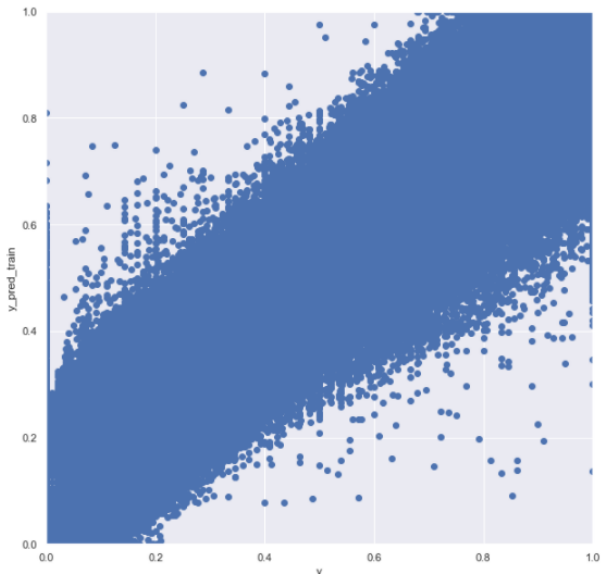

图 3.2 线性模型预测值可视化结果

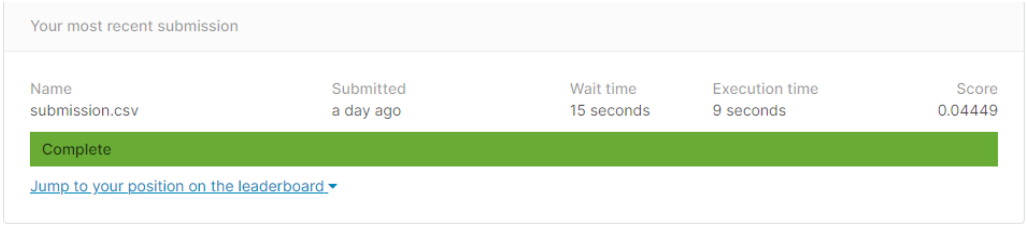

图 3.3 线性模型 kaggle 得分

我们的线性模型提交到 Kaggle 上的得分为 0.4449，处于中下等水平，因为线性回归是一个简单的模型，它的效果相对较差。这个得分指的是平均绝对误差（MAE），它的结果越小代表效果越佳。MAE 的计算公式如下式，其中 $f(x_i)$ 为预测值，y_i 为真实值。

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|h(x_i) - y_i|$$

3.1 多层感知机模型（MLP）

多层感知机是人工神经网络的一种，它包含输入层、输出层和若干个隐藏层，最简单的多层感知机只包含一个隐藏层。包含有两个隐藏层的多层感知机结构如图 3.4 所示。

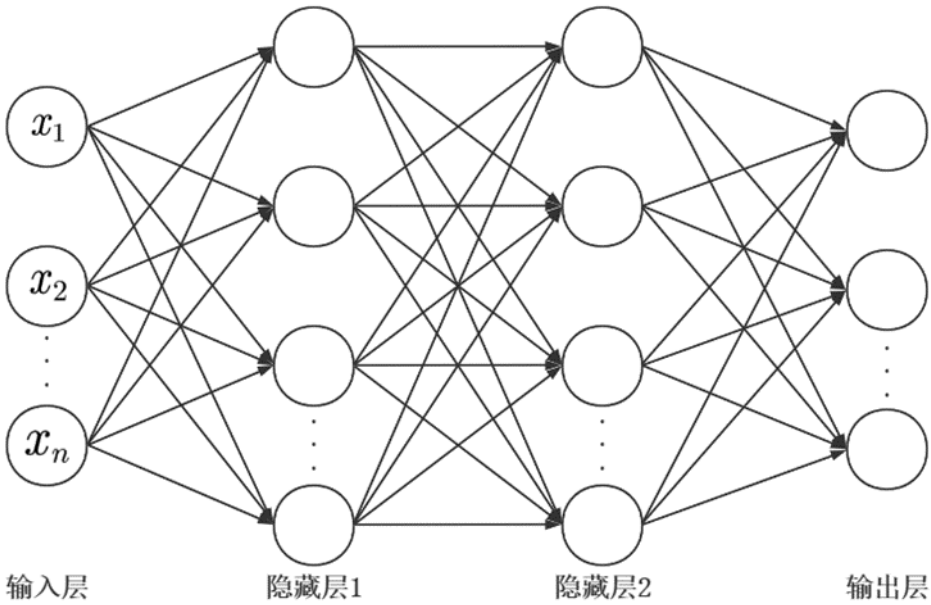

图 3.4 多层感知机结构

假设某个多层感知机只包含一个隐藏层，它一共有 m 个样本，n 个特征，隐藏层包含 h 个神经元，则输入层 $X \in R^{m\times n}$，隐藏层的神经元权重可以表示为 $W_h \in R^{n\times h}$，偏差可以表示为 $b_h \in R^{1\times h}$。假设输出层一共输出 q 个值，则输出层神经元的权重可以表示为 $W_o \in R^{h\times q}$，偏差参数可以表示为 $b_h \in R^{1\times q}$。隐藏层的输出计算公式为：

$$H = XW_h + b_h,$$

输出层的计算公式为：

$$O = HW_o + b_o。$$

从上面可以看出，多层感知机的下一层的输入就是上一层的输出，前面式子只是对数据进行简单的线性变换，事实上，为了使多层感知机达到分类的结果，还需要引入激活函数对多层感知机的每一层进行非线性变换，一般的激活函数包括 ReLU 函数、sigmoid 函数和 tanh 函数。

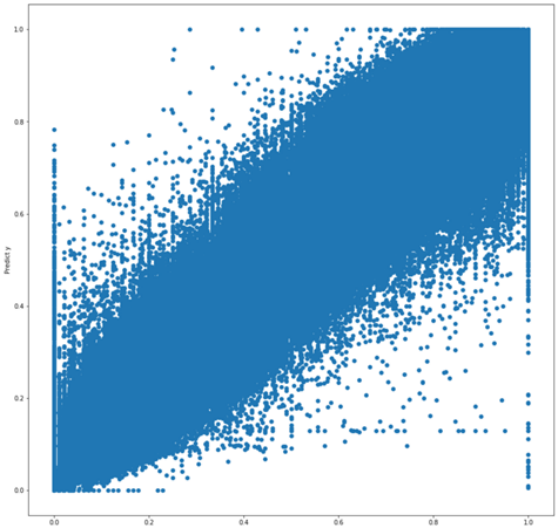

图 3.5 多层感知机模型预测值可视化结果

我们的 MLP 模型的预测结果可视化效果如图 3.5 所示。我们的 MLP 模型在训练集上也能达到 0.957 的准确度，但是提交到 Kaggle 上的 MAE 值为 0.0432，与线性模型性能基本相似。

我们认为主要原因是未建立合适的结构，在此处 4 个较小的隐藏层无法提取较多的有效信息。我们在之后会更加深入研究并采用更合适的结构。

3.1 Adaboost模型

Adaboost 简单来说就是多个弱分类器的组合，我们常用的神经网络、KNN 等模型都可以称为弱分类器，通过若干个弱分类器的分类，并将其结果和分类函数权重进行加权，最终可以得到强分类器的结果。Adaboost 的算法流程如图 3.6 所示。

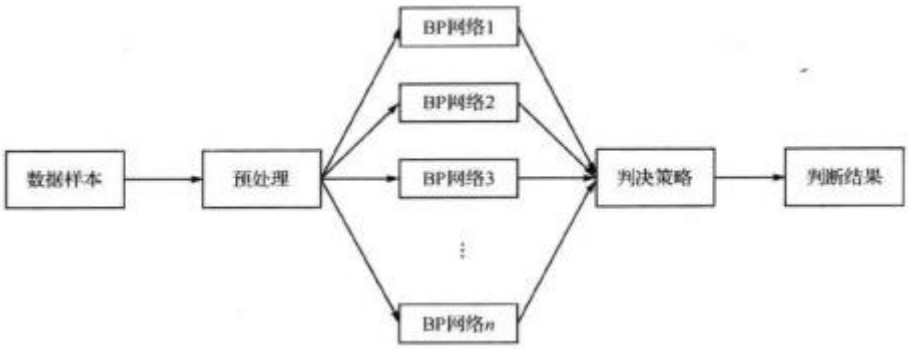

图 3.6 Adaboost 算法流程

Adaboost 的算法步骤如下。

第一步：在样本空间中任意选取 m 个训练数据，将测试数据的分布权重初始化为

$D_i(i)=\frac{1}{m}$，根据模型的输入以及输出来确定神经网络的结构，对神经网络的权重和阈值进行初始化。

第二步：在训练第 t 个弱分类器时，使用训练数据来训练 BP 神经网络，得到训练数据的预测输出，然后计算预测序列$g(t)$的预测误差和e_i，计算公式如下，其中$g(t)$为预测结果，y 为期望分类结果。

$$e_i=\sum_i D_i(i)\ \ i=1,2,3,...,m(g(t)\neq y)$$

第三步：根据上一步计算的预测误差e_i来计算权重a_i，计算公式如下。

$$a_i=\frac{1}{2}\ln\left(\frac{1-e_i}{e_i}\right)$$

第四步：根据上一步计算的权重a_i来调整下一轮训练的权重，调整公式如下，其中B_t是归一化因子，它的作用是使得分布权重的总和为 1。

$$D_{t+1}(i)=\frac{D_t(i)}{B_t}*\exp[-a_t y_i g_t(x_i)]\ \ i=1,2,3,...,m$$

第五步：经过 T 轮的训练后，可以得到 T 组弱分类函数$f(g_t,a_t)$，将 T 组弱分类器进行组合得到强分类函数$h(x)$，即：

$$h(x)=sign\left[\sum_{t=1}^{T}a_t\cdot f(g_t,a_t)\right]。$$

我们选取决策树作为底层回归器。在训练后，在 Kaggle 上的提交结果如图 3.7 所示。模型能在训练集上达到 98.7%的精度，提交后得到的分数为 0.02057，效果比较好。

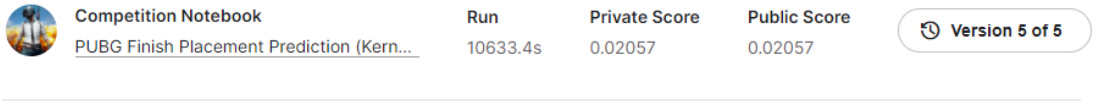

图 3.7 Adaboost 模型提交结果

4 总结与展望

起初面对浩瀚的题库，我们不知如何选择，在一番筛选之后，我们最终选择了这道关于 PUBG 数据分析的题目，这完全是兴趣使然，强烈的好奇心驱使着我们研究分析题目的步伐。尽管之前在选修数据挖掘课程时，我们已经对 Kaggle 有所了解，但是在拿到题目后还是有些不知应如何下手。

讨论之后，我们首先对数据进行了简单处理与分析，通过数据探索我们对其中的一些数据有了较为直观的认识。紧接着我们通过对变量之间关系的分析以及多变量综合分析，将变量进行了筛选，并集中考虑重要变量。在建模过程中，我们先后采取了三种不同的方式，首先我们选取了线性模型作为基线，通过前面设定的特征构建函数获取特征以及目标，使用线性模型最后的得分为 0.0449，结果不太理想；之后我们使用了 MLP（多层感知机），MLP 模型在训练集上能够大道 95.7%的准确度，但是最终提交后得到的 MAE 值为 0.0432，相较线性模型几乎没有提升；最终我们又选择了 Adaboost 模型，同时选取决策树作为底层回归器，模型在训练集上达到了 98.7%的准确度，提交后得到的 MAE 分数为 0.02057，效果达到了我们的预期。

本题排名第一的模型为 Light GBM，它是 GBDT（Gradient Boosting Decision Tree）的改进模型，Light GBM 是 boosting 集合模型中的新成员，由微软提供，它和 XGBoost 都是对 GBDT 的高效实现，原理上和 GBDT 以及 XGBoost 类似，全都采用损失函数的负梯度作为当前决策树的残差近似值，然后去拟合新的决策树。之后我们会进一步深入学习该模型，进一步研究我们目前的模型能否改进，从而取得更好的表现效果。

5 参考文献：

[1]Anzai Y. Pattern recognition and machine learning[M]. Elsevier, 2012.

[2]李航. 统计学习方法[M]. 清华大学出版社, 2012.

[3]Zheng A, Casari A. Feature Engineering for Machine Learning Models: Principles and Techniques for Data Scientists[J]. 2016.

[4]周志华. 机器学习与数据挖掘[J]. 中国计算机学会通讯, 2007, 3(12): 35-44.

[5]Mohri M, Rostamizadeh A, Talwalkar A. Foundations of machine learning[M]. MIT press, 2018.