驭风计划：实验 1

实验名称： 基于决策树的英雄联盟游戏胜负预测

姓名：王灿

UID：11031310

联系方式：13145318073

报告正文

1. 数据集

1.1. 数据集简介

1.1.1. 样本数量：数据集 League of Legends Diamond Ranked Games (10 min) 来自 Kaggle，包含了 9879 场钻一到大师段位的单双排对局，对局双方几乎是同一水平。每条数据是前 10 分钟的对局情况，每支队伍有 19 个特征，红蓝双方共 38 个特征，涉及了金钱，视野，塔防等多个方面，具体特征和说明可见参考 1。

1.1.2. 特性：根据官网介绍和绘制了箱图（图 1）进行数据分析，可以了解到其具有以下 4 条特性。

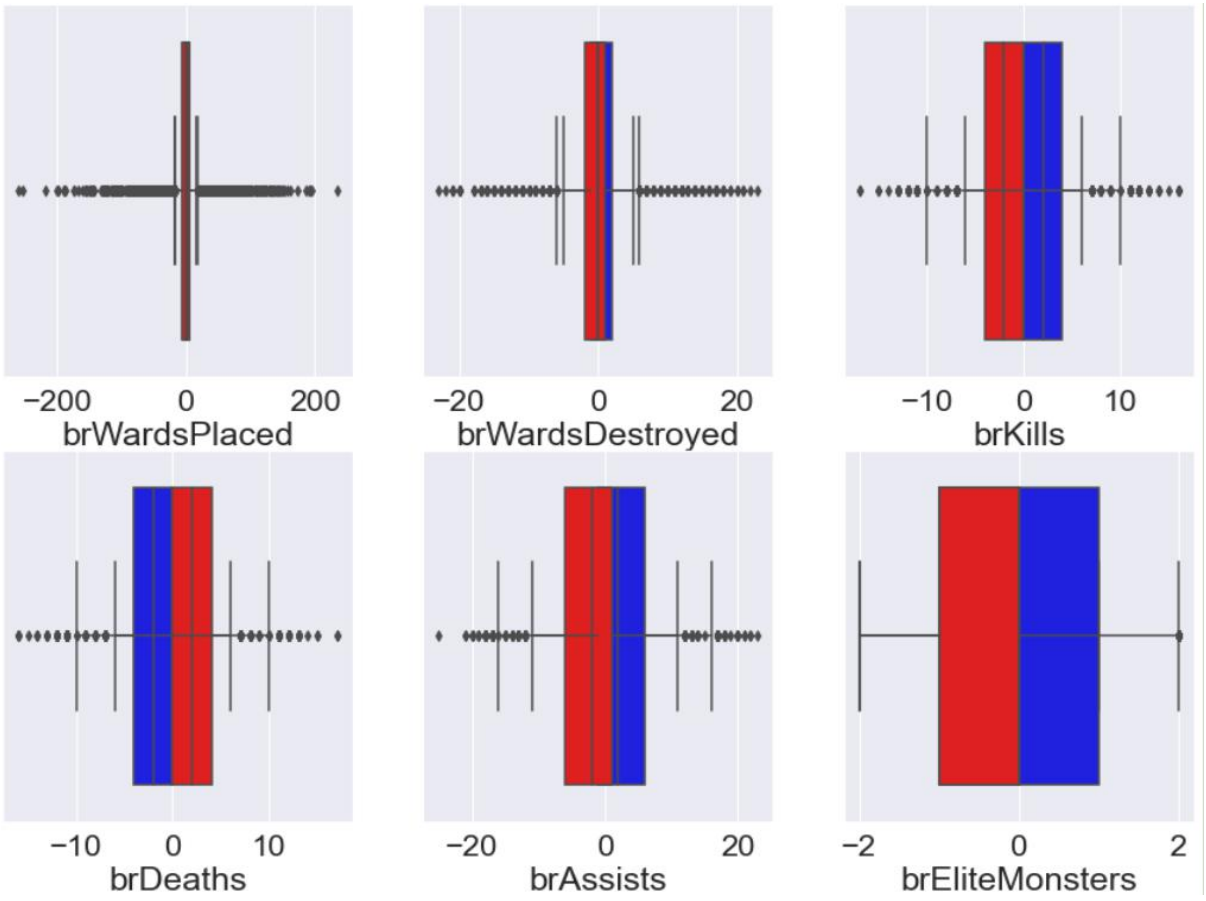

图 1 不同特征分布箱图（部分）

1.1.2.1. 存在冗余特征，如每分钟击杀数和击杀总数，可以通过与时间相乘得到等式，所以在开始要进行**特征的筛选和删除**。

1.1.2.2. 部分数据是连续的，而决策树算法一般是基于离散特征的，所以预处理部分将对特征进行离散化，即**将一定范围内的值映射成一个值**。

1.1.2.3. 数据是对称的，可以知道每局游戏红蓝双方的表现是无关的，也就说明可以通过做差来获得新的特征，并且因为对称，存在部分数据是均匀分布于正负样本之间的，所以**对特征的奇数划分**将更为合理。

1.1.2.4. 数据是正确且合理的，即使存在着离群值。通过代码进行检验了非法条件（比如摧毁视野数大于放置视野数的情况），发现并没有非法数据存在。且可以发现离群值分布松散但连续，每个特征中约有 1000 个为离群值，说明非噪点所为。那么这部分数据必须被保留并考虑，因为未知空间也可能存在这样的数据。同时，**对于这类数据，按照密度划分特征明显比按照区间划分更为合理。**

1.1.3. 训练集与测试集划分：已有训练样本和测试样本比例为 4：1。后文将使用 5 折交叉验证，将训练样本再随机划分为 5 份，每次取不同的 1 份作为验证集，用于超参数的调节和模型的改进，**最终得到比例构成为 4：1：1.25 的数据集**。

1.2. 预处理与特征表示

1.2.1. **无用特征的剔除：**首先去除无用特征，共有两部分。第一部分是可以通过换算得来已有特征的，比如每分钟金币，经验，击杀。另一部分是根据游戏规则可以判断无影响或影响极小的特征，比如对局编号，第一滴血。

1.2.2. **特征转换：**将剩余 28 个特征进行蓝红做差，得到为以前特征值一半数量 14 的数据。绘制热力图（图 2）查看其相关性。

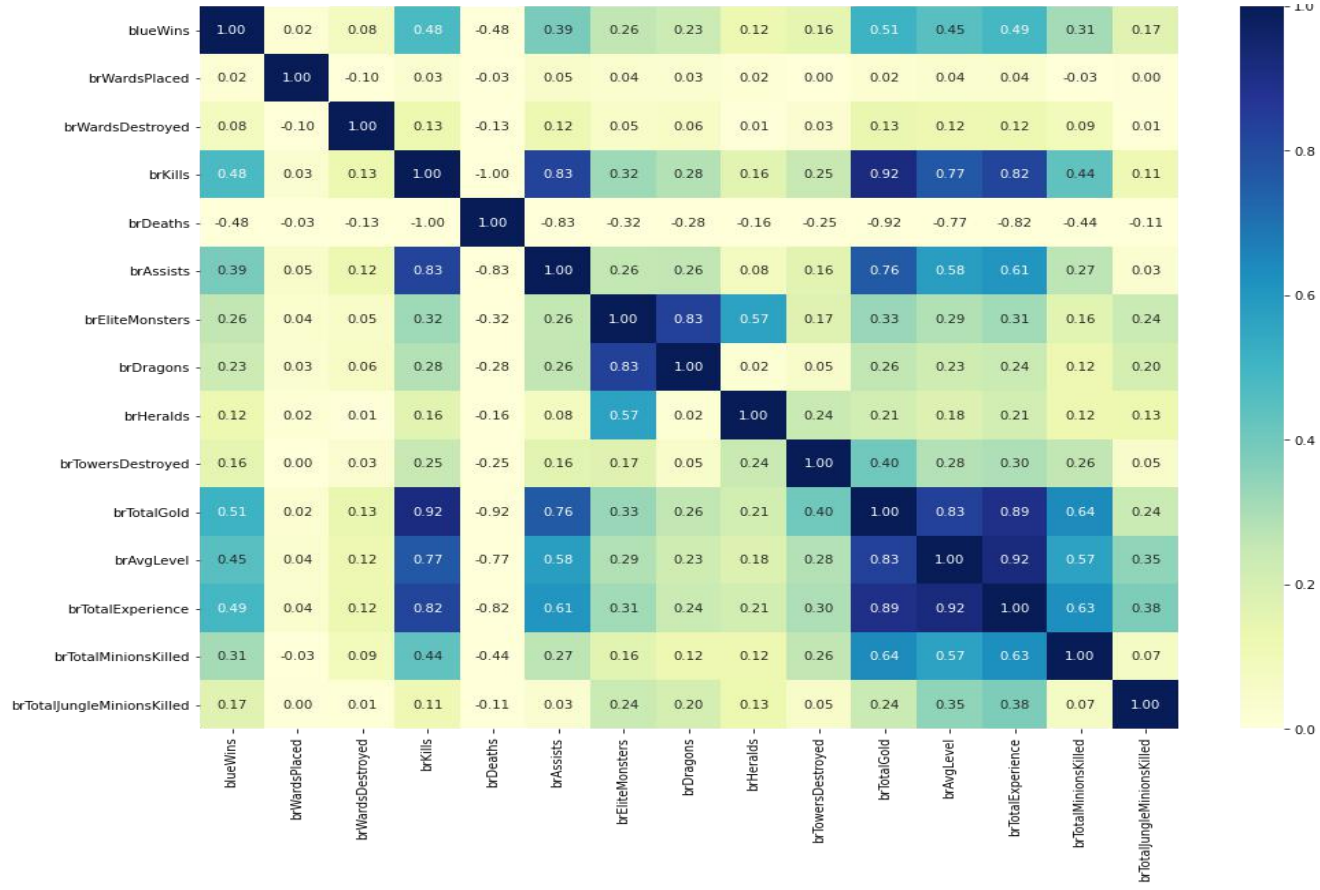

图 2 特征之间的相关性

值得一提的是，最开始凭借线性回归的经验，为了消除相关性使用了 **PCA 主成分分析法进行降维**，结果发现效果并不尽人意，准确率没有提升。查阅资料（参考 2）发现决策树和随机森林的预测能力不受多重共线性影响，有以下 2 点原因：

1.2.2.1. 统计分析中，作推断时，如果自变量存在共线性，将无法区分它们对因变量的影响，因此无法对结果进行清楚的解释。但是作预测时，我们并不关心如何解释自变量对因变量的影响。

1.2.2.2. 决策树用贪婪算法进行变量选择，只有新变量对结果影响比较大时，才会被加入到模型中。在决策树模型中，每一个树的构建都是贪婪的，因此，它们决定分裂时，树只会选择一个完全相关的特征。

1.2.3. **弱相关性特征剔除：**设置阈值将与是否获胜（标签）为弱相关性的特征剔除，阈值设定为 0.1，即相关性低于 0.1 时认为影响很小，共舍弃了 'brWardsDestroyed','brWardsPlaced', 'brTowersDestroyed', 'brTotalMinionsKilled', 'brHeralds' 五个弱相关特征。**最后获得数据为 9879×9 的特征。**

1.2.4. **数据离散化：**如上文 1.1.2 中所提，通过将连续的特征映射到不同区间来进行离散化，首先区间划分数目一定为奇数，也就是 1，3，5……其次，决策树的前几层倾向于与结果相关性最高的特征，也就是总金币和经验，这样可以尽可能的保证树的当前路径最短。通过遍历 1~11 之间的奇数，观察每一层树的结构，**取层数和特征相关性的加权平均得分（记树的第 i 层权重为 2^(总层数-当前层数)）最高的一颗树即可。**实验结果为 3，也就是每个特征将被离散化为 3 段，每个结点有 3 个子结点来构建一颗多叉树，实验代码如下。

```
def discrete_fea(dis_thre=10,method = 'qcut'):
    '''
    离散化每一列特征，即discrete_df[c] = ...
    dis_thre是对于有些特征本身取值就很少，可以跳过的阈值
    method是使用等区间(cut)或等密度(qcut)划分
    '''
    global df, discrete_df
    for c in df.columns[1:]:
        # 些特征本身取值就很少，可以跳过
        if len(set(discrete_df[c])) <= dis_thre:
            continue
        elif method=='cut':
            discrete_df[c] = pd.cut(df[c], dis_thre, labels=[i for i in range(dis_thre)])
        else:
            discrete_df[c] = pd.qcut(
                df[c], dis_thre, precision=0, labels=False, duplicates='drop')

discrete_fea(3, 'qcut')
```

2. 实验环境

2.1. 计算机配置

使用计算机为 Win10 系统，内存大小 16G，64 位 8 核处理器，其它配置如下图 3 所示。

图 3 计算机配置

2.2. 编程语言与平台

采用 python 编写，版本为 3.8.5。编程平台为网页交互式编程环境 jupyter notebook。

3. 实验过程（此处仅给出部分重点方法的说明）

3.1. 决策树设计

整个多叉树将以递归的形式创建，采用贪心的方法。因经过特征的删减，解空间并不庞大，可以遍历搜索。**贪心策略为：通过不纯度的做差得到信息增益，优先贪心增益最大的特征。**

3.1.1. 不纯度的计算：根据传入参数的不同，可以采用信息熵和基尼系数两种做法。都是先获取每个取值的比例，再根据公式计算，代码如下。

```
def impurity(self, label):
    '''
    计算不纯度，根据传入参数计算信息熵或gini系数
    label是numpy一维数组：根据当前特征划分后的标签组成
    '''
    cnt, total = Counter(label), float(len(label))
    probs = [cnt[v]/total for v in cnt]
    if self.impurity_t == 'gini':
        return 1 - sum([p*p for p in probs])
    return -sum([p * np.log2(p) for p in probs if p > 0])
```

3.1.2. 增益的计算：计算当前结点未分裂时的不纯度，和依据每个“尚未使用”过的特征来分裂后的不纯度。计算增益时，考虑了信息增益率，即在标签无关时的不纯度，以防止对特征取值多的天然偏置带来的问题。但因为上文离散化时将特征和最大需要划分阈值设为同一个，所以此处并未用到。

```
def gain(self, feature, label) -> tuple:
    '''
    计算某个结点对应不同特征信息增益
    feature是numpy一维数组：当前待分结点某个特征的值，
    label是numpy一维数组：对应的标签

    返回元组：
        信息增益率，以及一个分类字典f_index{特征值：数组下标}，以供获取子节点使用
    '''
```

3.1.3. 递归结点函数：将结点定义为两类，第一类是叶结点，只需要返回当前类别即可。第二类为可以继续分裂的结点，此时其结构为一个元组，元组内容为（特征下标，特征取值和对应的子节点，到达当前节点的样本中最多的类别）。

```
def expand_node(self, feature, label, depth, used_features) -> tuple:
    '''
    训练时递归函数分裂节点，考虑不同情况。最后返回该函数返回根节点。
    feature为二维numpy（n*m）数组，每行表示一个样本，有m个特征
    label为一维numpy（n）数组，表示每个样本的分类标签
    depth记录了当前节点的深度
    used_features表示已经用过的特征，不再重复使用
```

3.1.3.1. 递归终止条件：只有一种类别无需分裂或达到分裂阈值，此时返回叶结点，取当前结点最多的类别。

```
if len(set(label)) == 0:
    return label[0]
most = Counter(label).most_common(1)[0][0]
if depth > self.max_depth or len(label) < self.min_samples_split:
    return most
```

3.1.3.2. 遍历所有未使用特征，调用 gain()找到最佳分裂特征。如果找不到有用的分裂特征，也结束递归。遍历特征的每种取值，递归调用 expand_node 进行建树。同时记录当前结点的层数。

```
new_used_features = used_features+[bestf]
for v in bestf_idx:
    c_idx = bestf_idx[v]
    children[v] = self.expand_node(feature[c_idx, :], \
                                   label[c_idx], depth + 1, new_used_features)
self.tree[depth].append(self.features[bestf])
```

3.2. 数据预测

数据的预测也将以递归的形式遍历。如果输入是一维 numpy（m）数组，则是一个样本，包含 m 个特征，返回一个类别值即可。如果是二维 numpy（n*m）数组，则表示 n 个样本，，每个样本包含 m 个特征，分别调用递归函数，返回一个 numpy 一维数组。

```
if len(feature.shape) == 1:  # 如果是一个样本
    return self.traverse_node(self.root, feature)  # 从根节点开始路由
# 如果是很多个样本
return np.array([self.traverse_node(self.root, f) for f in feature])
```

从根节点开始递归，如果已经到达叶节点，则返回分类结果，否则依据特征取值进入相应子节点，递归调用 traverse_node。注意，如果特征取值在训练集中未出现过，返回训练时到达当前节点的样本中最多的类别。

```
def traverse_node(self, node, feature):
    # 要求输入样本特征数和模型定义时特征数目一致
    assert len(self.features) == len(feature)
    '''
    traverse_node()预测时遍历节点，根据特征路由，考虑不同情况
    node表示当前到达的节点，例如最开始取根节点self.root
    feature是长度为m的numpy一维数组，一个样本的所有特征
    '''
```

4. 参数设置和优化

4.1. 待调参数

决策树分类中共有三类超参数需要选择：

4.1.1. 最大深度 max_depth。也是第一个可以确定的参数，随着树的深度变大，模型得分一般随着 max_depth 单调递增，之后会趋于稳定。**这个参数和下一个超参数存在着相互制约的关系，随着最大深度变大和叶子结点最小包含样本数变小，模型将趋于复杂。**

4.1.2. 叶子结点最小包含个数 min_samples_split。一个结点必须要包含至少 min_samples_split 个训练样本，这个结点才允许被分支，否则分支就不会发生。利用暂定的 max_depth 参数，绘制曲线，观察得分随着 min_samples_split（分割内部节点所需的最小样本数）的变化规律，从而确定 min_samples_split 参数的大概范围。

4.1.3. impurity_t 不纯度计算方式。这是最后确定的一个参数，通过遍历确定即可。

4.2. 优化上界

一方面，在实验中调用了 sklearn 封装好的决策树算法，得到了 0.72 的准确率，另一方面，在 kaggle 官网找到了一份使用了多种机器学习方法进行训练的 NoteBook（参考 3），并进

行运行，得到如下图 4 结果。**可以初步判断，在不对数据进行较大加工的前提下，该算法的上界约为 73%。**

	Accuracy Score
Naive Bayes	0.717611
DT	0.692308
Random Forest	0.727227
Logistic Regression	0.730263
K_nearest Neighbors	0.717105

图 4 本数据集上其它算法准确率

4.3. 优化方法

4.3.1. K 折交叉验证法。即将初始采样分割成 K 个子样本，一个子样本被保留作为验证模型的数据，其他 K-1 个样本用来训练。交叉验证重复 K 次，每个子样本验证一次，结合 K 次结果得到单一估测。查看源码后，发现 StratifiedKFold 是用于存在多分类时保证验证集和测试集的类比配比相同，如图 6 标注部分所示。本数据集并没有这样的要求，所以直接使用 cross_validate 即可。

```
cv : int, cross-validation generator or an iterable, default=None
    Determines the cross-validation splitting strategy.
    Possible inputs for cv are:

    - None, to use the default 5-fold cross validation,
    - int, to specify the number of folds in a `(Stratified)KFold`,
    - :term:`CV splitter`,
    - An iterable yielding (train, test) splits as arrays of indices.

    For int/None inputs, if the estimator is a classifier and ``y`` is
    either binary or multiclass, :class:`StratifiedKFold` is used. In all
    other cases, :class:`KFold` is used.
```

图 5 cross_val_score 分类方式与 KFlod 方式区别

4.3.2. 为获得较好性能且避免过拟合，采取**调参策略如下：首先使用树的深度来限制，不断增加树的深度，选取趋于稳定的最小深度。**然后通过不断减小 min_samples_split 来控制精度。最后通过遍历计算不纯度方法，来确定最终取值。

5. 实验结果

5.1. 各种参数配置下的实验结果

5.1.1. 如下图 6 所示，首先固定分裂直至一个结点，来观察拟合程度随着树最大深度的变化。可以发现在深度为 3，采用基尼系数的时候准确率最高，为 73.10%。

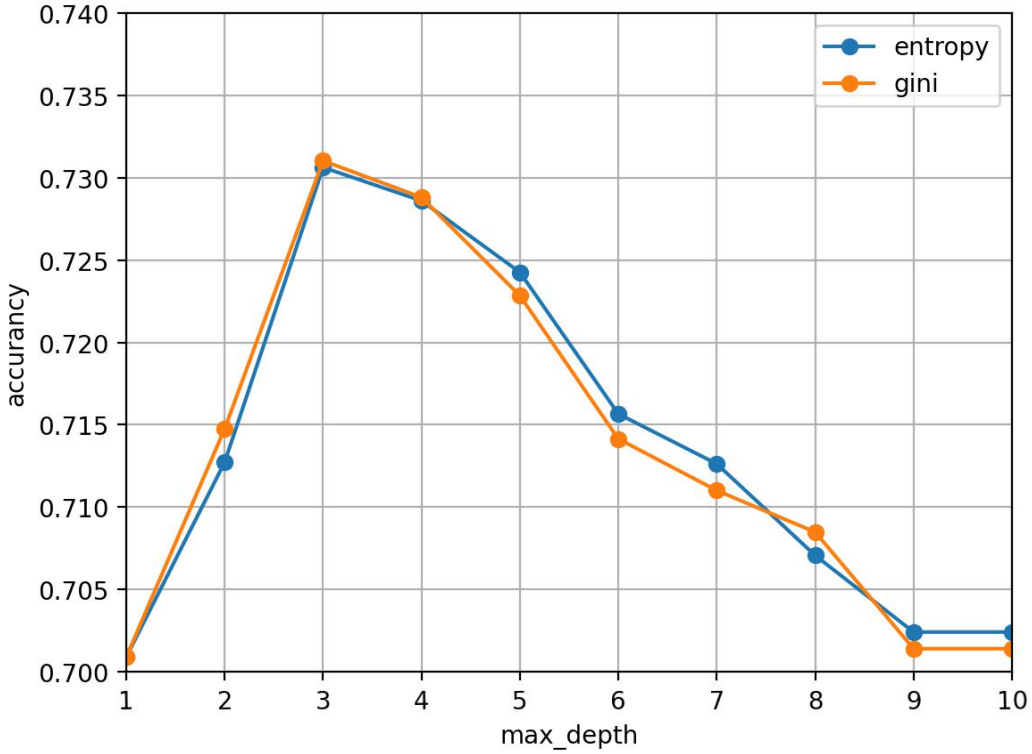

图 6 准确率随着最大深度的变化

5.1.2. 固定深度后，通过调节最小包含样本数来提高精度 K 值。可以发现，在样本数为 450，采用基尼系数的时候准确率最高，为 73.12%。

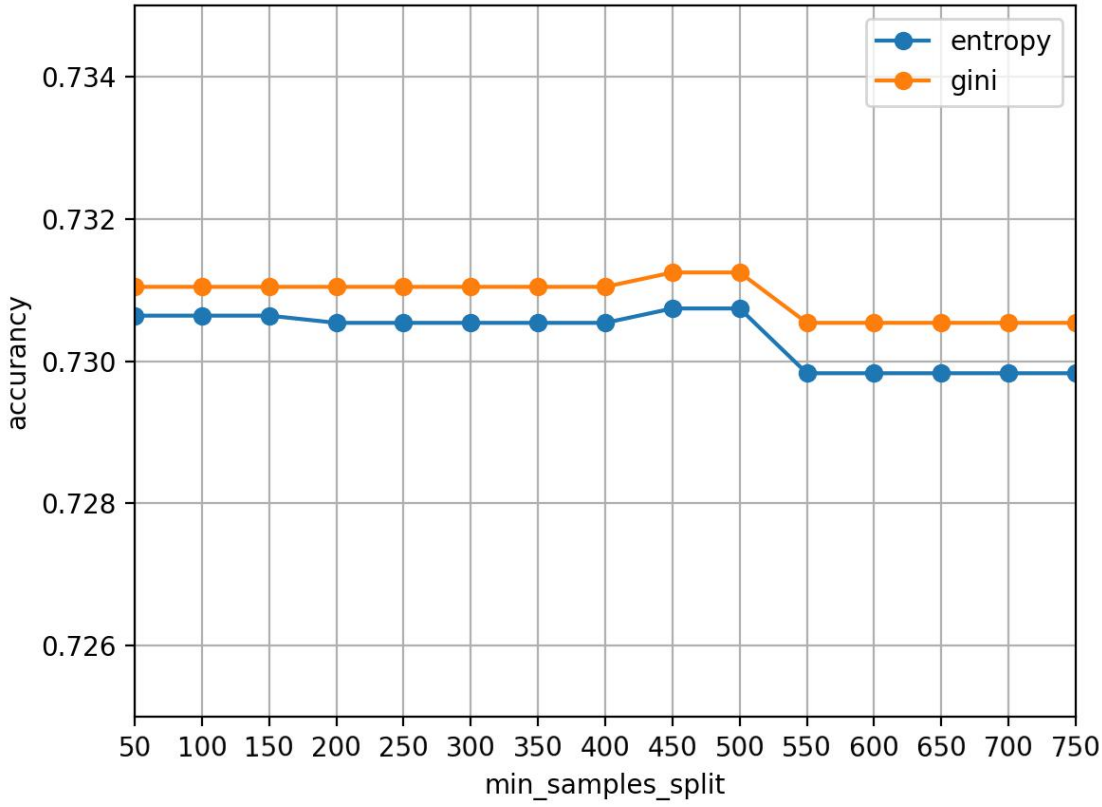

图 7 准确率随着最小包含样本数的变化

5.2. 最终准确率

将该结果（特征个数 9，特征划分区间 3，不纯度为基尼系数计算，最大深度 3，最小包含样本 450）应用于测试集，根据测试集是未知领域数据需要模型更强的泛化性，对参数进行微调，得到最终准确率 74.14%，树的结构，召回率等其余信息如下图 8 所示。

```
pred_value  [0 0 0 ... 1 0 0]
true_value  [0 1 1 ... 0 0 0]

TREE:
Layer3:['brAvgLevel', 'brDragons', 'brKills', 'brDragons', 'brDragons', 'brEliteMonsters']
Layer2:['brKills', 'brTotalExperience', 'brTotalExperience']
Layer1:['brTotalGold']

accuracy: 0.7414   precision: 0.7342   recall: 0.7226   f1_score: 0.7283
```

图 8 测试集上的准确率

5.3. 收获和展望

5.3.1. 算是第一次比较完整的实践了机器学习的方法。之前总是感觉似懂非懂，查阅了很多资料，尤其是 python 封装好的决策树给了我很多思路。然后自己一点点改代码报错，一点点的写出来，成就感很足。深刻的感受到了“绝知此事要躬行”的道理，大大加深了我对决策树的理解。比如课上说的根据密度划分的具体方法，还有树可以不仅仅是二叉树等等很多。

5.3.2. 代码能力得到了提高，很多库函数的调用需要查阅文档细节，比如 cut 和 qcut 的区别，取的区间开闭问题。调用 cross_validate 和 GridSearchCV 需要重写的方法等。

5.3.3. 因为是代码补全工作，所以顺着做差的思路进行了数据处理。做的途中考虑到另一种思路。可以使用双方特征，根据单独一方的数据进行获胜概率预测，比较哪个概率更大不失为另一种选择。

5.3.4. 关于数据的特征划分：这次实验非常深刻的体会到了数据的重要性，当优化模型参数可提高的程度到了极限，那就应该从数据入手。比如这里我多筛去了无关特征，才使得优化结果能逼近优化上界。同时，推测存在一种更合理的做法，即筛选两类标签的最大值和最小值，这 4 个点一定是分界点，然后再根据训练结果调整中间的特征个数。后来尝试了这种方法，准确率仍是逼近上界，没有实现突破。

5.3.5. 本实验中直接将叶节点定义为包含最多类别的标签，忽略了概率问题。因为本身竞技预测可能就存在“极限翻盘”或者不可预测现象，所以根据类别概率取可能会更加合理。

5.3.6. 本实验中使用的是预剪枝，如果使用后剪枝可能有更好的表现。

参考资料

[1] League of Legends Diamond Ranked Games (10 min) | Kaggle

[2] 为什么特征相关性非常的重要？ - 云+社区 - 腾讯云 (tencent.com)

[3] LoL - how to win | Kaggle

[4] 机器学习超详细实践攻略(9)：决策树算法使用及小白都能看懂的调参指南 - 知乎 (zhihu.com)

[5] 决策树手动实现 | tian-feng (innovation64.github.io)

[6] GitHub - RRdmlearning/Decision-Tree: CART Decision Tree

[7] 【Python 机器学习实战】决策树和集成学习（二）——决策树的实现 - 云+社区 - 腾讯云 (tencent.com)

[8] 驭风计划：培养人工智能青年人才【第三期】 - 学堂在线 - 学堂在线 (xuetangx.com)