Report

■ Introduction:

a) Objectives and Requirements: create a classifier that take as inputs from data; the evaluation result should at least be higher than 50%; On program time cost and test score, select score;

b) Methodology: data preprocessing at first, then using CART algorithm to train Decision tree model, using Cross-validation get the score.

■ Algorithms;

a) methods: CART decision tree uses Gini index to select partition attributes. The smaller the Gini index, the higher the purity of the dataset. Therefore, in the candidate feature set, the feature which makes the Gini index minimum is selected as the optimal partition attribute.

b) Parameters:

   1) Criterion: Gini index (Feature selection criteria)
   2) Splitter: best (Feature classification criteria)
   3) Max_depth: 14 (Maximum depth of decision tree)
   4) min_samples_leaf: 6 (Minimum sample number of leaf nodes)

■ Requirements

a) numpy: It is an open source numerical computation extension of Python

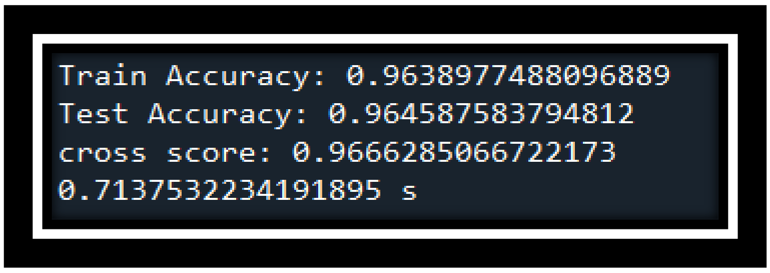
b) pandas: A data analysis tool

c) sklearn: Machine learning library

d) time: Module for processing date and time

■ Results

Train accuracy	0.963897
Test accuracy	0.964587
Cross score	0.966628
time	0.71375s



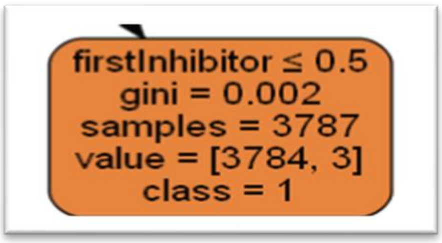
```
Train Accuracy: 0.9638977488096889
Test Accuracy: 0.964587583794812
cross score: 0.9666285066722173
0.7137532234191895 s
```

■ Discussion:

a) Data preprocessing:

1) Discrete data processing:

for the features like "fristBlood" and "firstTower", it contains three types of data "0", "1", "2". In the model, it is closer to dividing continuous values from '0' to '2' (In fact, we know that continuous values are transformed into discrete values by dichotomy)



But I don't think '0' to '2' is an efficient way to divide the threshold, I chose to map '2' to '-1'. This can reduce the number of repeated partitions of the same feature.

Before mapping:

For "0" or team 2 get it, it needs to be divided by one time.

For team 1 get it, it needs to be divided by two time.

After mapping:

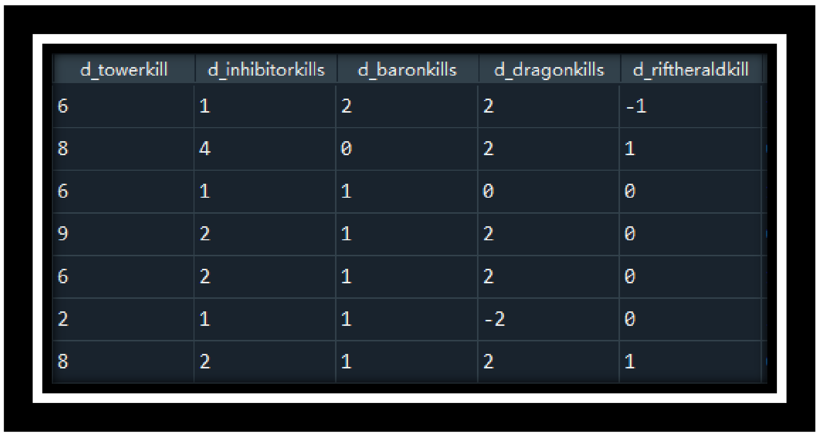
For team 1 or team 2 get it, it only needs to be divided by one time.

For else situation ('0'), it needs to be divided by two time.

We know that in a normal game, 'team 1 get it' or 'team 2 get it' are almost certain happen. We get it the number of '1' and '2' must bigger than '0'. Therefore, the model will be batter after mapping.

2) Combined feature:

I introduced the "difference" feature (team_1 minus team_2)



d_towerkill	d_inhibitorkills	d_baronkills	d_dragonkills	d_riftheraldkill
6	1	2	2	-1
8	4	0	2	1
6	1	1	0	0
9	2	1	2	0
6	2	1	2	0
2	1	1	-2	0
8	2	1	2	1

At the first time, I want to reduce the time cost and keep the score by reducing the number of features (Delete the independent data of two groups and add the difference feature). The time cost dropped to 0.53s, while the score dropped to 0.957. That's not my intention to "increase the score". Then I tried to train with all the features and got a better score (0.962~0.963).

I got this idea from the “oblique decision tree”. If we use multiple features to divide, it can reduce part of the time cost.

For example:

The first needs to be divided into four nodes

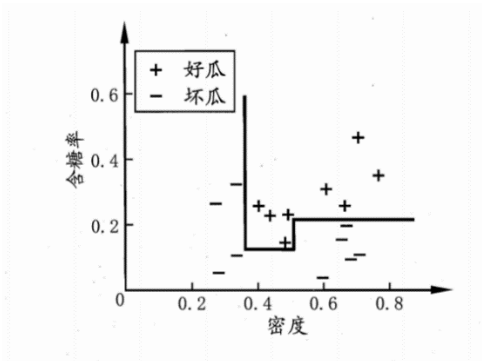


图 4.11 图 4.10 决策树对应的分类边界

The second needs only two nodes to divide

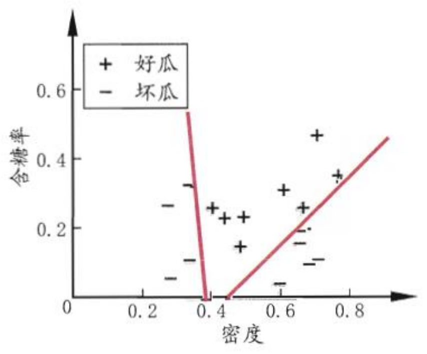


图 4.14 图 4.13 多变量决策树对应的分类边界

3) remove abnormal data:

   When I was looking at the data set, I found that many samples had time less than 300 (Generally speaking, it's between 800 and 3000)

   I try to remove it from the training set, but it has terrible effects—the train score increased to 0.970 while the test score drop to 0.957. It proves that the generalization ability of the model is reduced, and means this road is wrong.

b) pre-pruning

   1) Introduction: Decision tree model is easy to over fit. In order to prevent over fitting problem, decision tree can be pruned. In the “sklearn” page, It only includes pre-pruning function, not post-pruning function.

   2) max_depth and min_samples_leaf: In this model, I use these two parameters for pre-pruning.

      Max_depth for limiting the growth of trees

      Min_samples_leaf for improving the generalization ability of lower nodes

   3) Optimization parameters:

      I used the simplest brute force solution, which is double traversal. But when the value range is large, it will cause a lot of time cost. I prefer to use “gradient descent”, but I don’t have enough time to implement it.

c) Score: k-fold cross validation

   1) Introduction: If the number of training samples in S is m, then each subset has m / K training samples, and the corresponding subsets are called {S1,

S2 ,…., Sk}. Each time from the divided subset, take out one as the test set, and the other k-1 as the training set.

This method makes full use of all samples. But the calculation is complicated, which needs training K times and testing K times.

2) Application: call “cross_val_score() ”fuction, set the k=10

d) Experience:

1) Improve familiarity with the DT model from theory to practice
2) The ability to call different parameters has been improved

e) Extended and improved

1) “oblique decision tree”: I want to improve the decision tree by using a combination of features, which contains 'd_towerkill', 'd_inhibitorkills', …, 'd_riftheraldkill'. It should be like a linear combination

$$\mathrm{f(x)} = w^T * \mathrm{x} + \mathrm{b}$$

x: matrix of eigenvalues

w: matrix of weights

The weight can be set by our own or by calculation

We know that there is subordination on features with the same attributes.

Like : $w_{baronkills} > w_{dragonkills}$

So we can through different constraints to calculate the most appropriate “w”.

2) Achieve standard gradient descent: Gradient descent is a common heuristic minimization method, which can effectively reduce the time cost. How to get the mathematical relations to make chain derivation is a difficult problem.

$$\frac{\partial Score(\ )}{\partial l} = max(\frac{\partial Score(\ )}{\partial max\_depth} * \cos\theta + \frac{\partial Score(\ )}{\partial \mathrm{min\_samples\_leaf}} * \sin\theta)$$

$$Score(\ ) = Score(\ ) + \ \varepsilon * \frac{\partial Score(\ )}{\partial l}$$

Get the best value to maximize the score

3) On the processing of time characteristic less than 300s

   The general game match time is between 500 and 3000 second. In the data set, we found some data sets that are difficult to predict. In this game, there is a way called remake (Within 3 minutes of the beginning of the game, the system detects that a player has not entered the match and can propose a “remake”). So I think if we want to directly training this part of data is unreasonable.