# Classification of LOL Results

Li Yuanzhuo

October 2020

## 0.1 Problem Introduction

League of Legends(LOL) is an online, $5vs5$ competitive PC game. There are lots of strategies that can make one team win the battle. In this project, we are going to utilize the advanced algorithms, for instance, Decision Tree(DT) and Artificial Neural Network(ANN), to make the classification of different strategies thus figure out strategies that can lead to victory depend on the original data, after that, we will make a prediction of the test data to see whether our classification is good or not. What's more, in view of the defect of the classifier, we attempt to optimize the classifier in order to fit the practical situation better.

## 0.2 Algorithms

### 0.2.1 Decision Tree

We choose decision tree as our first classifier due to its relatively easier algorithm and higher precision than other algorithms. In order to make use of every available attributes in the original data, we need to set up several layers nodes and each node stands for the classification of one attribute. Under this white box mechanism, we are able to see the detailed process of the classification. However, when the depth of the tree is too deep, the classifier is more likely tend to overfitting, which causes the poor performance in prediction.

To get around this problem, after training this model, the depth of the decision tree should be carefully determined, which impacts not only the prediction accuracy, but the efficiency of the program. We have explored an advanced method which considers the impact factors and quantifies this impact so that successfully tackle this problem.

The one parameter needed to set up is the depth of the tree *depth*, which directly determine the tree's time complexity and space complexity. The other parameter is *entropy*

```
for dpt in range(1,22):
    time_1 = time.time()
    clf = DecisionTreeClassifier(criterion="entropy",max_depth=dpt)#total depth 21
```

### 0.2.2 Artificial Neural Network

We choose ANN as our second classifier because of its outstanding ability in dealing with multi-features classification. It combines different attributes with different weights so that make sure every attribute has its own contribution to the result. With the black box mechanism, we only need to input the attributes and obtain the results after the model is trained.

However, there also exists some defects. When the data size is too large and learning rate is too small, the model also tends to overfitting and local optimizing. What's more, because of the back propagation algorithm, the algorithm needs to run hundreds of loops to converge to the optimal parameters, which makes the training part consume time seriously. So we are going to find the relatively better learning rate to training the model so that avoid overfitting and time-consuming situation.

The parameters we set up for the neural network are learning rate $lr$ ,looping times *epoch*, input and output features of hidden layer and output layer.

```
class ANN(nn.Module):
    def __init__(self):
        super().__init__()
        self.hidden_1 = nn.Linear(in_features=16,out_features=17)#in_features = number of features
        self.output = nn.Linear(in_features=17,out_features=2)#out_features = kinds of output
    def forward(self,x):
        x = torch.sigmoid(self.hidden_1(x))
        x = self.output(x)
        x = F.softmax(x,dim=1)
        return x
model = ANN()
```

```
index = []
for rate in np.arange(0.01,0.9,0.01):
    cost = nn.CrossEntropyLoss()
    back_propagation = torch.optim.Adam(model.parameters(),lr=rate)
    back_propagation.zero_grad()
    epochs = 200
    errors = []
    #index = []
    for i in range(epochs):
        y_hat = model.forward(X_train)
        error = cost(y_hat,y_train)
        errors.append(error)
        back_propagation.zero_grad()
        error.backward()
        back_propagation.step()
        #index.append(i)
        #print(f"Epoch :{i} Loss:{error}")
```

## 0.3 Experimental Requirements

### 0.3.1 Packages for Decision Tree

```
import time
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import MinMaxScaler as MMS
from sklearn.impute import SimpleImputer as SI
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import export_graphviz
from six import StringIO
from sklearn.preprocessing import MinMaxScaler as MMS
from sklearn.impute import SimpleImputer as SI
from IPython.display import Image
import pydotplus
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
```

### 0.3.2 Packages for Artificial Neural Network

```
import torch
import pandas as pd
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler as MMS
from sklearn.impute import SimpleImputer as SI
```

## 0.4 Experimental Results

### 0.4.1 Decision Tree

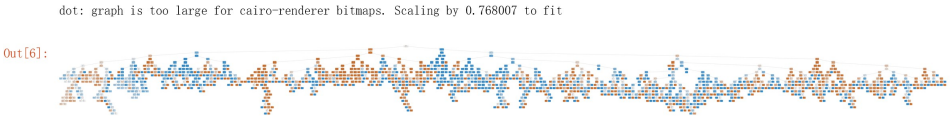
1. Set up the decision tree model and find out the best split in each node in the criterion of Info Entropy:

$$E = -\sum_{i=0}^{c-1} p_i(t) log_2 p_i(t) \tag{1}$$

In this equation, $p_i(t)$ represents that at node t, the rate of a sample that belong to class i. So the entropy $E$ can be view as the separation degree of sample, the lower $E$, the better split is selected.

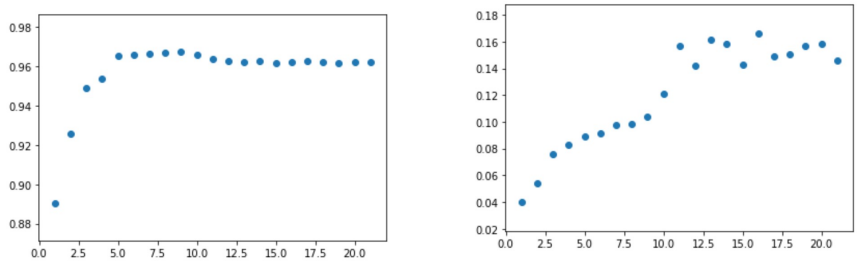
2. After that, use this model with different depths to fit the data and count the duration of each depth in training.

3. Plot the trained decision tree with defaulted depth of 21:



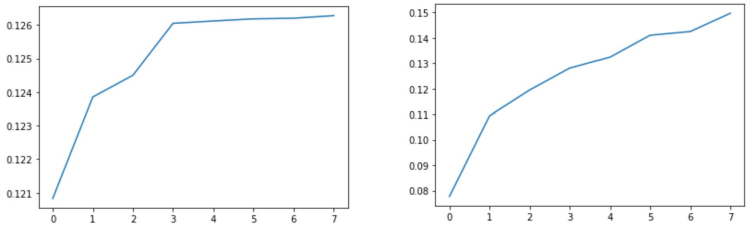

It is obviously that this decision tree is far more large and complex, which is easier tend to overfitting and less efficient. So we need to find the suitable depth that guarantee both efficiency and accuracy.

4. Visualizing the accuracy figure and time figure.




The right figure represent n-accuracy plot where n is the depth of tree, the left figure represents n-time plot.
From the right figure we can see that the accuracy increases according to the increase of depth until depth equal to 5, then the accuracy basically remains constant or somehow decrease slightly.
From the left figure we can find that the time consumed by training the model basically increase with the depth increase.
For better efficiency and accuracy, it is reasonable for us to choose only the depth equals or less than 9 for optimization.

5. Normalizing accuracy and time so that they are in the same scale(0,1) and it is convenient for us to combine them together to evaluate the efficiency of different trees. The normalized figures are as follows:

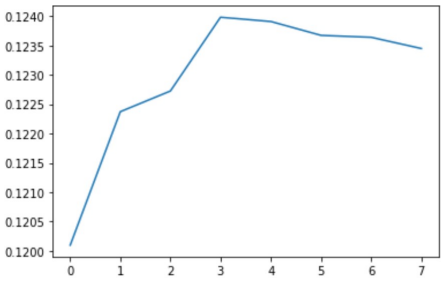



6. Access the efficiency by combining two factors
   In ideal cases, we want to have a classifier whose accuracy is high enough and training time is short enough. Thus, we want to define an equation where the accuracy has the direct ratio with the depth and time has the inverse ratio with the depth. Finally, the equation likes this:

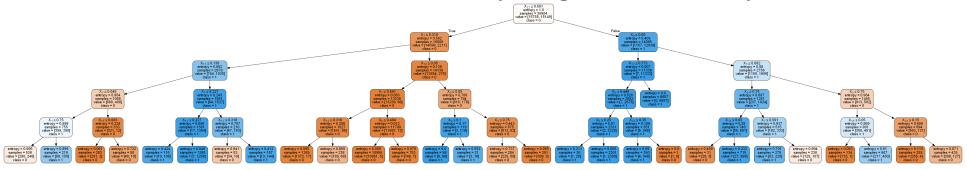
$$E_f(d) = A(d) \times [1 - T(d)^2] \tag{2}$$

   In equation (2), $E_f(d)$ represents the efficiency function of classifier at depth $d$, $A(d)$ represents the accuracy function and $T(d)$ represents the time function.

   Then we plot $E_f(d)$ function:



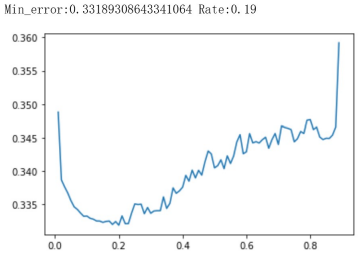

   From the figure we can easily find out when the depth = 5, the $E_f(d)$ function has the maximum value, say, highest efficiency.



   With accuracy is 0.966 and training time is 0.089s.

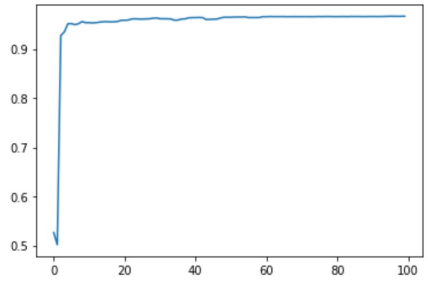
### 0.4.2 Artificial Neural Network

1. Input the attributes' values to the classifier and train the classifier with learning rate from 0.01 0.9 by 200 loops and then plot the figure of $200th$ error and learning rate.




2. From the figure we can use *min* function to see that when learning rate is 0.19, the training error is minimum, which somehow reflect the better performance of prediction. And when rate is more than 0.2, the tendency of the curve is in confusion, indicates that the classifier is overfitted.

3. We will retrain the neural network with learning rate 0.19 and predict the test data result. Then we obtain the figure of loop time and accuracy like this:




With final accuracy is 0.967 and training time is 4.109s.

### 0.4.3 Results table

The table of classifiers' accuracy and training time as follows:

| Classifier | Accuracy | Training Time |
|---|---|---|
| Decision Tree | 96.6% | 0.089s |
| Neural Network | 96.7% | 4.109s |

## 0.5 Comparison and Discussion

### 0.5.1 Decision Tree

Decision Tree classifier is a straightforward and easy classifier to train, it totally depend on the training data but sometime may dropped into overfit condition when the depth of the tree is too deep. In this project, to balance the accuracy and time so that the classifier can fit a more practical condition, we came up with a concept of efficiency function which can evaluate whether the classifier is both precise and fast. This function can effectively avoid overfitting by assigning the inverse relation between depth and efficiency.

It is remarkable that this efficiency function is not perfect and precise enough to thoroughly reflect the efficiency of the program. The accuracy variable is not sufficient to represent the work done by the program. So in the future, to optimize this function, the time complexity $T(n)$ and space complexity $S(n)$ should be considered as the depth $n$ increase.

### 0.5.2 Artificial Neural Network

Neural Networks algorithm can successfully deal with the classification problem with multiple input and output. The disadvantages are also obvious: the parameter especially learning rate is very difficult to determine, when the scale of each attributes differ a lot, the prediction result is largely depend on the attributes with larger scale. In this project, we have used transversal

method to compare the final training errors in the figure with learning rate from 0.01 to 0.9. After that, we can easily find out the learning rate with lowest training error. The figure can also help to prevent the classifier from overfitting. To deal with different scale problem, it is remarkable that we applied normalization method to transform the attributes' values into the same scale (0,1).

Altough we have find the relatively better learning rate of this classifier, there also remains the number of hidden layers to be determined. The more hidden layers, the higher the level of abstraction for the input features. So how to determine the best number of hidden layers need to make a deeper research.

## 0.6 Summary

In order to make a classification of LOL's game strategy and predict the unknown winner based on given strategies, we utilize the DT and ANN classifiers. To deal with the low efficiency problem of DT, we use a evaluation function to access the efficiency of DT model and find the best depth of the tree is 5, with accuracy is 0.966 and time is 0.089s. To deal with the learning rate problem of ANN, we use transversal method to determine the best learning rate. Finally the learning rate is 0.19 and accuracy is 0.967, time is 4.109s.