LOL Winner Prediction Based on Multiple Classifiers

Author: Lin Ze

1.Introduction

In many games, we can always predict which team or which one will win the game correctly. You may wonder how we can make these predictions. It is hard for us to describe it. We will always answer that 'This team (or this guy) is stronger.' Now we'd love to ask the machine to make the prediction. So here comes a question, how should we define or quantify strength? To finish this task, first of all, we can regard win and defeat as two classes. Then the problem can convert into classification. Therefore, we are required to understand the basis classification methods and apply different classification algorithms to classifier the data. The requirement of this project is to use the LOL matches record to train the classifiers and predict which team will win the match. In this report, I will use Python to analysis about 3 Million match record and build several classifiers (Decision Tree, Voting ensembles, etc) to adapted it.

2.Algorithms

Decision Trees (DT) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. So how can we achieve it? First of all, tree algorithms include ID3, C4.5, C5.5 and CART.

Scikit-learn uses an optimized version of the CART algorithm. When CART is used as a classification tree, the characteristic attributes can be either continuous or discrete, but the observation attributes (that is, label attributes or classification attributes) must be discrete types. The CART classification tree uses the Gini index when the nodes are split, and the Gini index measures the impurity of the data partition or the training data set. The smaller the Gini value, the higher the purity of the sample (that is, the higher the probability that the sample belongs to the same category). After measuring the Gini index of all the values of a certain feature in the data set, the Gini Split info of the feature, which is Gini Gain, can be obtained. Regardless of pruning, the process of recursive creation of the classification decision tree is to select the smallest node of Gini Gain as the bifurcation point each time until the sub-data sets belong to the same category or all the features are used up.

The formula of Gini index is as below ($p_i(t)$ is the frequency of class *i* at node *t*, and *c* is the total number of classes):

$$Gini\ index = 1 - \sum_{i=0}^{c-1} p_i(t)^2$$

Comparing with the calculation of entropy, the calculation of Gini index avoids a large number of logarithmic operations.

The essence of the classification learning process is the reduction of sample uncertainty (ie the entropy reduction process), so the feature split with the smallest Gini index should be selected. The sample set corresponding to the parent node is D, and CART selects feature A to split into two child nodes, and the corresponding sets are DL and DR; the Gini index after splitting is defined as follows:

$$G(D, A) = \frac{|D_L|}{|D|} Gini(D_L) + \frac{|D_R|}{|D|} Gini(D_R)$$

CART algorithm flow:

1. If the conditions for stopping splitting are met (the number of samples is less than the predetermined threshold, or the Gini index is less than the predetermined threshold (the samples basically belong to the same category, or there are no features to split), then stop splitting;

2. Otherwise, select the smallest Gini index to split;

3. Recursively execute steps 1-2 until the splitting stops.

The following part is to introduce the parameters of the algorithm I used in DT. First one is the criterion, we can choose ‘gini’ or ‘entropy’. Because we chose the CART algorithm, the criterion should be ‘gini’. The next one is the splitter, its default is ‘best’, but the amount of data is too large. To avoid overfitting, we’d better choose ‘random’. Additionally, max_depth is also a very significant parameter. The accuracy can reach 100% if the depth of the tree is deep enough, though it means overfitting. Therefore, it is necessary to limit the depth, which lower than 20 can be regarded as acceptable. That’s all the parameters I consider in DT algorithm.

The second classifier I chose is KNN. The core idea of the KNN algorithm is: if most of the k nearest neighbors of a data in the feature space belong to a certain category, then the sample also belongs to this category (similar to voting), and has the characteristics of the sample in this category. In layman's terms, for a given test sample and based on a certain distance measurement, the classification result of the current sample is predicted by the k closest training samples. For numerical attributes, the KNN algorithm uses a distance formula to calculate the distance between any two-sample data.

The optimal value of k (parameter: n_neighbor) needs to be determined through experiments. Starting from k=1, use the test data set to estimate the error rate of the classifier. Repeat the process, each time k is increased by 1, allowing one neighbor to be added, and k that produces the smallest error rate is selected. Generally speaking, the more training data sets, the larger the value of k, so that classification prediction can be based on a larger proportion of the training data set.

The third classifier I chose is Voting (soft) in ensemble learning. Ensemble learning is to combine multiple weakly-supervised models to obtain a better and more comprehensive strong-supervised model. The underlying idea of ensemble learning is that even if a weak classifier gets a wrong prediction, other weak classifiers can also correct the error back.

The ensemble method is a meta-algorithm that combines several machine learning techniques into a predictive model to achieve the effect of reducing bagging, boosting or improving prediction (stacking). The Voting has two types, one is ‘hard’ and another is ‘soft’. ‘Hard’ uses predicted class labels for majority rule voting. ‘Soft’, predicts the class label based on the argmax of the sums of the predicted probabilities, which is recommended for an ensemble of well-calibrated classifiers. In my project, I use soft voting because in this case, the minority obeying the majority does not apply, so a more reasonable voting method should be a worthy voting method. I combined KNN, SVM, DT and MLP four classifiers together as the soft voting classifier.

The parameter of Voting itself is the type of voting, I chose ‘soft’. Besides, in the estimators, I apply KNN with n_neighbors=10 (10 nearest point around), SVM with kernel=‘rbf’ and probability=True (Output probability instead of label), DT with the max_depth=20. It can make sure that they are well-calibrated.

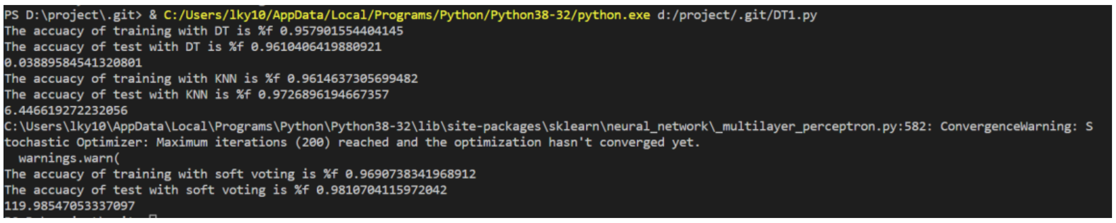
3.Requirement

The passages that I used in the code are as follows:

```
import sklearn

import numpy as np                                              ##Deal with the data.

import pandas                                                   ##To read the csv file

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split            ##To divide the data into
traing set and test set

from sklearn import metrics
from sklearn.neural_network import MLPClassifier

import pickle as pkl                                        ##To save the classifiers

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.ensemble import VotingClassifier

import time                                                 ##Record the time cost
```

4.Result

Result / Classifiers	Training accuracy	Test accuracy	Time cost
DT	0.958	0.961	0.039s
KNN	0.961	0.973	6.447s
Soft Voting	0.969	0.981	119.985s



```
PS D:\project\.git> & C:/Users/lky10/AppData/Local/Programs/Python/Python38-32/python.exe d:/project/.git/DT1.py
The accuacy of training with DT is %f 0.957901554404145
The accuacy of test with DT is %f 0.9610406419880921
0.03889584541320801
The accuacy of training with KNN is %f 0.9614637305699482
The accuacy of test with KNN is %f 0.9726896194667357
6.446619272232056
C:\Users\lky10\AppData\Local\Programs\Python\Python38-32\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:582: ConvergenceWarning: S
tochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
The accuacy of training with soft voting is %f 0.9690738341968912
The accuacy of test with soft voting is %f 0.9810704115972042
119.98547053337097
```

5. Comparison and discussion

Through this project, I have deeper understanding of classification, especially the DT, KNN, ensemble voting and so on. It is also very important to deal with the data. For example, at the beginning, I take all the feature for training, whose accuracy is only 40%-60%. After analysis the feature that I was given, I found that the first 4 feature: game ID; creation time; gameDuration and seasonId may have nothing to do with the result of a game. Therefore, the program would not choose the first four features to classifier. The performance of the classifiers has huge improvement. The accuracy is increase from 50% to about 96%. We should know that more features and more data don’t mean better performance and higher accuracy. It’s necessary to delete some unusual or irrelevant data. (sometime called noise)

In further learning, if time permitted, I will try to improve my model using the ways below:

1. Adapt more classifiers.
2. Delete more irrelevant data.
3. Try more ensemble ways to improve the accuracy and reduce the time cost at the same time

There are still plenty of classifiers that we can use like: SVM, ANN, MLP, Stacking and so on. They have different advantages and disadvantages. For this reason, they may have good performance in this project. Delete more irrelevant data can help us reduce the influence of noise, which will make the classifier unreliable. It can also reduce the amount of unnecessary calculation, which will get higher accuracy and cost less time. The third one, as we can see, its accuracy is the highest one. However, its time cost is much higher than others. We need to consider that it is worthy or not. A good classifier should be both accurate and quick.

Other ensemble ways like random forest and so on can be tried, I can also change the parameter to find a balance point between time and accuracy. That’s what I learn from this project.

6. Appendix

You can also clone the code and data from: https://github.com/WilliamSCUT/Project1