Classification of League of Legends (LoL) Results

Zhao Qiuran

South China University of Technology

October 7, 2020

Abstract

League of Legends (LoL) is one of the most played eSports in the world at the moment. Arguably, the fun aspect of games or sport lies within the uncertainty of their outcomes. The creators of LoL have tried their best to match players with teammates and opponents of as similar skill level as possible. In this project, 3 million match records of solo gamers will be accessible as training set. Each record comprises of all publicly available game statistics of a match played by some gamer, including an important field called "winner". If "winner" is “1”, the team 1 won the match, or vice versa.

I. Introduction

We would like to investigate what are the better strategy to win a LoL game. In this project, several classifiers, trained by millions of match records of solo players, which contain features (events in a game) like first blood, first baron and so on, will be used to make evaluation about the results based on those events provided in a game. There is a minimum requirement of the task that the evaluation result should at least be higher than 50% (random guess). In the program, several classifiers will be used, including Decision Tree, Multi-layer Perceptron, Support Vector Machine and k-Nearest Neighbor, given the same training set, measure scores and time cost will serve as a standard for comparison among them.

II. Algorithms

Several classification algorithms are used in the program and realized through python.

Decision Tree

Decision tree is the most powerful and popular tool for classification and prediction. A decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. Three main steps are included in creating a decision tree: feature selection, generation and pruning. For node splitting, the feature with the highest information gain can be selected as the test feature, entropy and gini index can be used to calculate information gain, in this project we choose entropy. Assume that there are m classes in set D, |D| and $|C_{i,D}|$ are the number of samples in them, the information entropy of D is given by

$$\boldsymbol{Info(D) = -\sum_{i=1}^{m} p_i log_2 p_i}$$

the information gain is given by

$$\boldsymbol{Gain(C) = Info(D) - Info_C(D)}$$

The feature with the largest information gain will be chosen as the splitting feature. The basic algorithm is as following:

```
1. If all tuples in D belongs to the same class $C_j$ Add a leaf node labeled as Cj
    Return            // Termination condition
2. Select an attribute $A_i$
3. Partition $D = \{D_1, D_2, ..., D_p\}$ based on p different values of $A_i$ in D
4. For each $D_k \in D$
    Create a node and add an edge between D and $D_k$ with label as the Ai 's attribute value in $D_k$
5. For each $D_k \in D$
    BuildDT($D_k$ )            // Recursive call
6. Stop
```

In this project, a machine learning library in python called sklearn is used to do the algorithm, the parameters are all default except for the feature selection criteria which use entropy.

Multi-layer Perceptron (MLP)

A multilayer perceptron is a feedforward Artificial Neural Network (ANN) that generates a set of outputs from a set of inputs. An MLP is characterized by several layers of input nodes connected as a directed graph between the input and output layers. It can solve more complex and nonlinear problems. Figure II-1 represents three layers of perceptron with three inputs, two outputs and a hidden layer of four neurons.

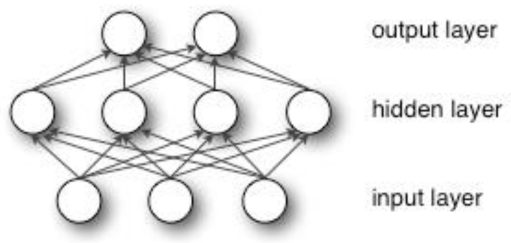


Figure II-1

All neurons in MLP are similar. Each of them has several input links (it takes the output values of several neurons in the previous layer as input) and several output links (it passes the response to several neurons in the next layer). The values retrieved from the previous layer are added to some weights, and a bias term is added to each individual neuron. Using activation functions to transform the sum, activation function can also be different for different neurons. The neuron of the hidden layer is fully connected to the input layer. The whole MLP model is summed up in a formula:

$$f(x) = G(b_2 + W_2(s(b_1 + W_1)))$$

with bias vectors b_1, b_2; weight matrices W_1, W_2 and activation functions G and S. Assuming that the input layer is represented by vector x, *$h(x)=s(W_1x+b_1)$* constitutes the hidden layer, the function s can be the sigmoid function or tanh function, but for the data in this project, a simple linear function $f(x) = x$ can do as well as these complex functions and cost less time. The output vector is then obtained as: $o(x)=G(b_2+W_2h(x))$, G represents the Softmax function. So, all the parameters of an MLP are the connection weights and biases between the layer. For a specific problem, solving the best parameter is an optimization problem. Stochastic Gradient Descent (sgd) and Adam Optimizer are two ways to solve this, here we choose Adam for its higher efficiency and ability to process large-scale data. The sklearn in python is also used, the parameters are all default except for the activation function which use 'identity'.

k-Nearest Neighbor (k-nn)

A k-nearest-neighbor algorithm is an approach to data classification that estimates how likely a data point is to be a member of one group or the other depending on what group the data points nearest to it are in. It is an example of a "lazy learner" algorithm, meaning that it does not build a model using the training set until a query of the data set is performed. The general procedure for k-nn is:

```
Let m be the number of training data samples.
Let p be an unknown point.
1. Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).
2. for i=0 to m:
    Calculate the distance d(arr[i], p).
3. Make set S of K smallest distances obtained. Each of these distances corresponds to an already classified data point.
4. Return the majority label among S.
```

In k-nn algorithm, by calculating the distance between objects as the non-similarity index between each object, the matching problem between objects is avoided. Euclidian distance or and Manhattan distance are generally used, here we choose Euclidian distance for the dimension of data is not that high:

$$d(x, y) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2}$$

The value of k is important, it can be determined through cross-validation (the sample data is divided into training data and validation data in a certain proportion). Starting from a small value of k, and increased continuously, then the variance of the validation set is calculated to finally find an appropriate value of k, python can be used to realize it, the plot can be drawn like Figure II-2

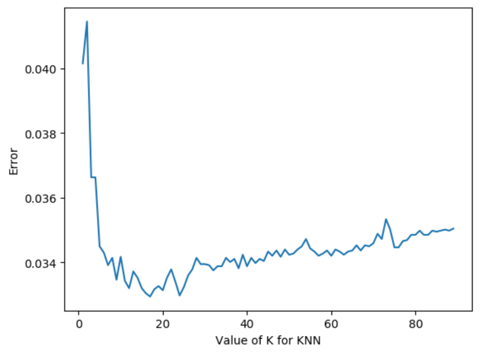


Figure II-2

Obviously, the error is the smallest when k is 17, so take k=17. In sklearn library, we use KNeighborsClassifier with parameters k=17, weights='distance', algorithm='kd_tree', others by default.

Support Vector Machine (SVM)

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used for classification problems like the one in this project. An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (Figure II-3).

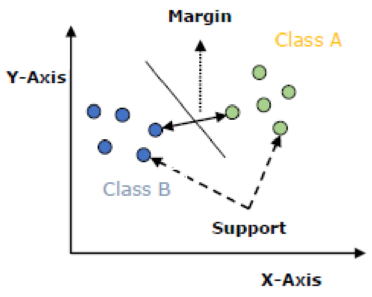


Figure II-3

Maximum margin classifier only depends on few samples, called Support Vectors. SVM uses a technique called kernel in which kernel transforms a low dimensional input space into a higher dimensional space, which means that it converts non-separable problems into separable problems by adding more dimensions to it. In this project, we choose linear kernel:

$$K(x, x_i) = sum(x * x_i)$$

Also, sklearn library is used in the program, all parameters by default except for the kernel by 'linear'.

III. Requirements

- Anaconda 3 with Python 3.7
- Python libraries including sklearn, pandas, time, matplotlib
- PyCharm used as IDE

IV. Results

	Precision	Recall	F-Measure	Time cost
DT	0.96558	0.95679	0.96116	0.048

MLP	0.96716	0.95011	0.95856	1.399
k-nn	0.96325	0.97258	0.96789	2.345
SVM	0.97056	0.94856	0.95943	4.521

Table 1

The evaluation is based on the following metrics:

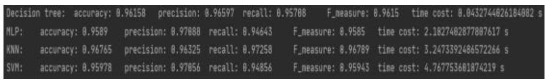
$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

where TP, FP and FN are the numbers of true positive, false positive and false negative results respectively. Table 1 shows the precision, recall, F-measure and time cost for training and testing by different algorithms.

Figure IV-1 shows the screenshot of the result of the program,



```
Decision tree:  accuracy: 0.96158   precision: 0.96597  recall: 0.95788    F_measure: 0.9615   time cost: 0.0432744026184082 s
MLP:    accuracy: 0.9589    precision: 0.97088  recall: 0.94643    F_measure: 0.9585   time cost: 2.1827402877807617 s
KNN:    accuracy: 0.96765   precision: 0.96325  recall: 0.97258    F_measure: 0.96789  time cost: 3.2473392486572266 s
SVM:    accuracy: 0.95978   precision: 0.97056  recall: 0.94856    F_measure: 0.95943  time cost: 4.767753601874219 s
```

Figure IV-1

V. Comparison and Discussion

Experience Gained

In this project, the knowledge of various classifiers like Decision Tree, Artificial Neural Network learned in class was practiced, which deepened my understanding of the basic algorithm of them as well as some other classifiers like k-nn and SVM. For the tools used in this project like python and its libraries, I also try to take good advantage of them, use them to realize the algorithms and draw plots to directly reflect the result for comparison.

Possible Improvement

The program is just a simple one, all of the algorithms are based on existing machine learning libraries, and we can only optimize the results by adjusting the parameters in them, because of the data set we used, the result like precision score and recall score are very positive, but for more complex situations in reality, more adjustments will be needed, for example, an ensemble which can combine multiple learning algorithms to achieve better performance can be a great method to satisfy the goal.