1.Introduction:

At present, the League of Legends Global Finals (S10) is in full swing. League of Legends (LoL) is one of the most played eSports in the world at the moment. A lot of people focus on this international competition. Arguably, the fun of games or sport lies in the uncertainty of their outcomes. In every game, the audience pay more attention to the winner at last, but in fact, we can predict the winner of a game in the first 10 minutes.

In this project, the students will have access to about 3 Million match records of solo gamers as training set. Each record comprises of all publicly available game statistics of a match played by some gamers. In sum, the fields of the data include Game ID, Creation Time (in Epoch format), Game Duration (in seconds), Season ID, Winner (1 means team1 won and 2 means team2 won), first Baron, dragon, tower, blood, inhibitor and Rift Herald (1= team1, 2 = team2, 0 = none), and the number of tower, inhibitor, Baron, dragon and Rift Herald kills each team has. The students are required to use the match records to create one or more classifier to predict the winner of a game.

To find the best classifier of the match records, I had used four common classifiers—Decision tree(DT), Artificial Neural Network(ANN), Support Vector Machine(SVM ） and K-NearestNeighbor(KNN) to build the model and then compared their accuracy and training time to find the beat classifier.

2. Algorithms:

2.1 The introduction of the classification methods:

①Decision tree: A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. Decision trees typically consist of three different elements: root node, branches and leaf node. Root node is the top-level node and

represents the ultimate objective, or big decision you're trying to make. Branches, which stem from the root, represent different options—or courses of action—that are available when making a particular decision. They are most commonly indicated with an arrow line and often include associated costs, as well as the likelihood to occur. The leaf nodes—which are attached at the end of the branches—represent possible outcomes for each action.

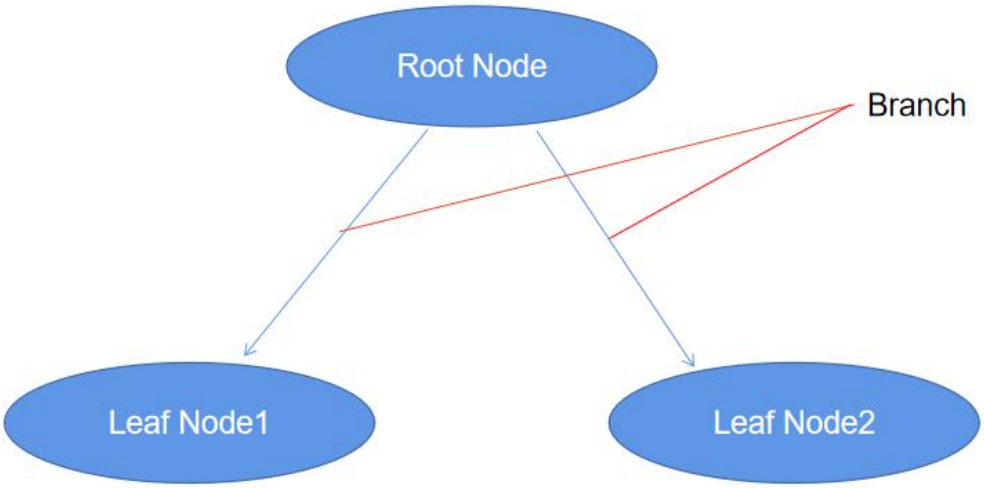

Figure 1 The basic three different elements of DT

② Artificial Neural Network: An artificial neural network (ANN) is the piece of a computing system designed to simulate the way the human brain analyzes and processes information. It is the foundation of artificial intelligence (AI) and solves problems that would prove impossible or difficult by human or statistical standards. ANNs have self-learning capabilities that enable them to produce better results as more data becomes available.

Multi-Layer Perceptron(MLP) is one kind of ANN. Besides input and output layers, MLP also had hidden layers which are intermediary layers between input and output layers. At the same time, MLP also has more general activation functions. MLP can solve any type of classification task involving nonlinear decision surfaces.

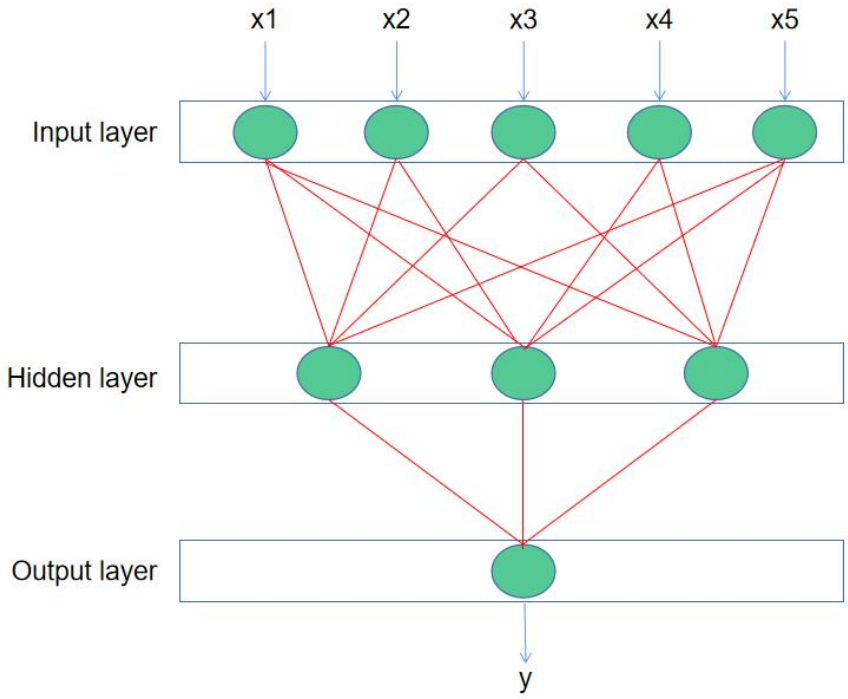

Figure 2 The most simple MLP

③Support Vector Machine: Support vector machine (SVM) is a powerful yet flexible supervised machine learning algorithm which can be used both for classification and regression. But generally, they are used in classification problems. SVM has its unique way of implementation as compared to other machine learning algorithms. SVM is extremely popular because of its ability to handle multiple continuous and categorical variables. 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 (MMH). If the data set is linearly separable, using the SVM can let the classification problem be formulated as Quadratic Optimization Problem and solve for ω and b. If the data set is non-linearly separable, SVM also can solve the problem with adding a slack variables or using a kernel.

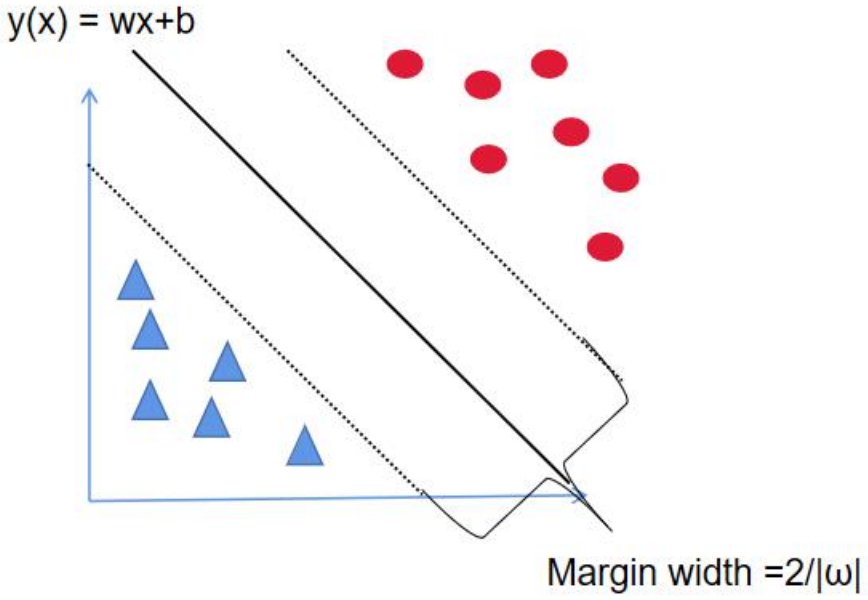

Figure 3 Using SVM to solve Linearly Separable classification problem

④ K-NearestNeighbor: K-Nearest Neighbors is a machine learning technique and algorithm that can be used for both regression and classification tasks. K-Nearest Neighbors examines the labels of a chosen number of data points surrounding a target data point, in order to make a prediction about the class that the data point falls into. K-Nearest Neighbors (KNN) is a conceptually simple yet very powerful algorithm, and for those reasons, it's one of the most popular machine learning algorithms.

When using KNN, a new pattern is classified by a majority vote of its k nearest neighbors (training samples).

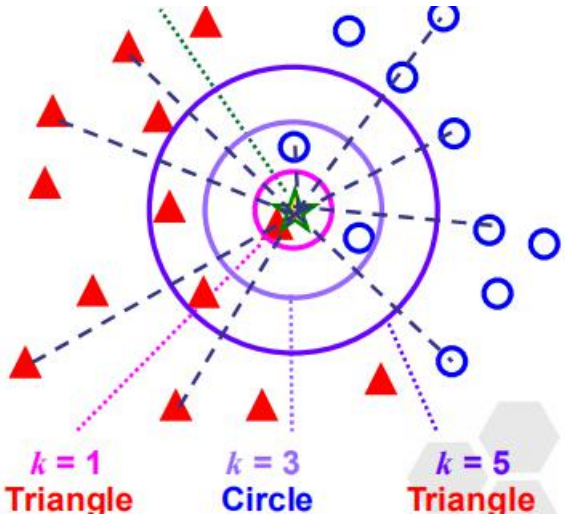

Figure 4 A example of KNN classifier

2.2 The workflow of the classifiers

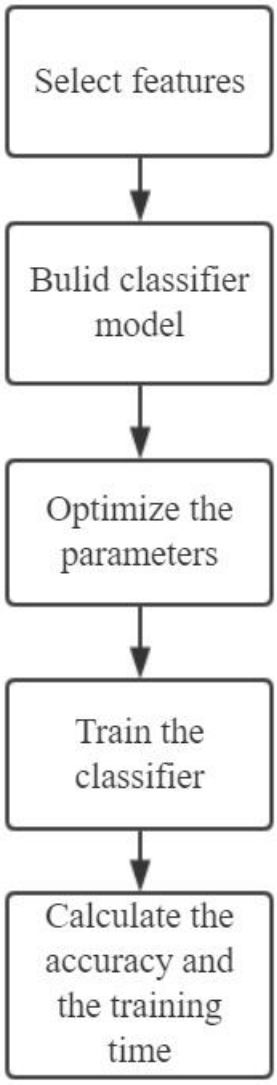

Figure 5 The workflow of building a classifier

For a machine learning problem, when we get the task and the data, we should first look at the data. Because no all the features are useful for the classifier to predict the outcomes and sometimes some data may even be the noise which may reduce accuracy of the classifier, we need to select the useful features to train the classifier. After we build a classifier, we should optimize the parameters because different classifier has different parameters and even a parameter can be different for different data sets. Sometimes, a change of a parameter may cause a big change of the accuracy of a classifier, so it is important to optimize the parameters of each classifier. After we optimize the parameters, we can train the classifier and final get the accuracy and

training time of each classifier. The four classifiers have the same workflow and it can be seen in Figure 5.

2.3 Select features

In this project, I chose to use the correlation to select useful features. Seaborn is a library based on Matplotlib. Basically what it gives us are nicer graphics and functions to make complex types of graphics with just one line of code. As a result, I used the Seaborn to plot a heatmap which can show all the correlations between variables in a dataset. The related code can be seen in the file called heatmap.py.

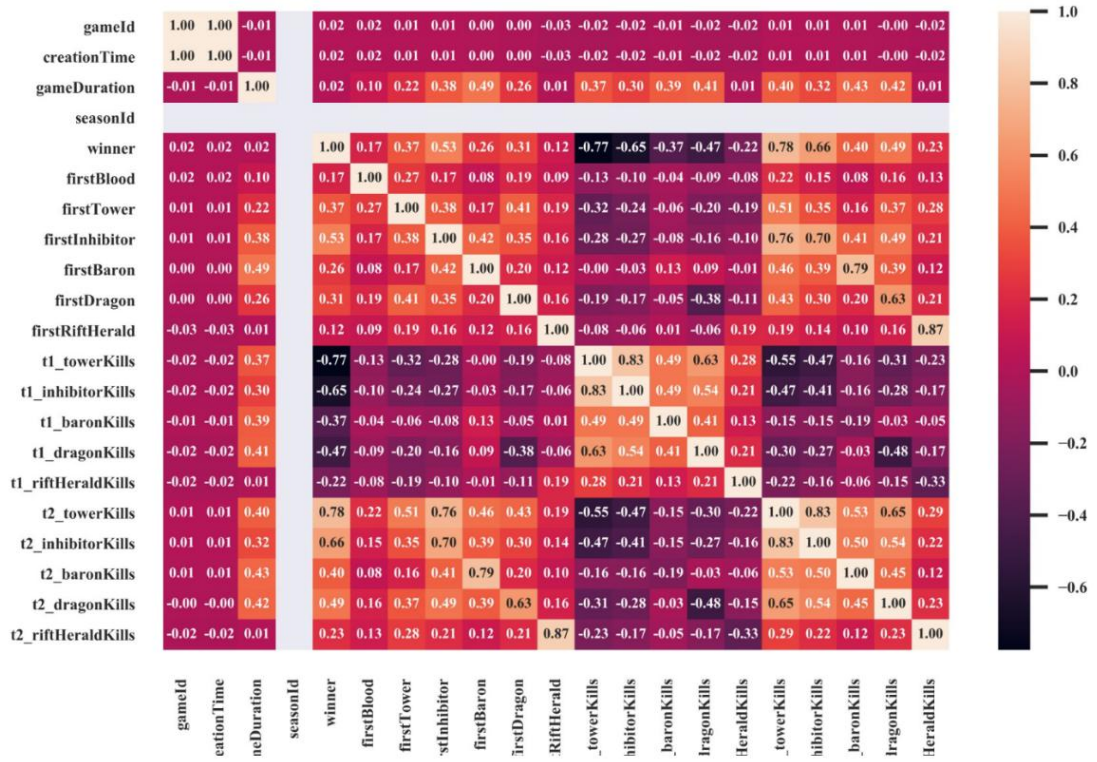

Figure 6 All the correlations between variables in a dataset

From the Figure 6, we can find that the features—gameId, creation time, gameDuration and seasonId have very low correlations with the winner. As a result, I only selected the else 17 features' data as train set.

```
data1 = pd.read_csv("new_data.csv")
data1 = data1.iloc[:, 0:21]
data1 = np.array(data1)
train_label = data1[:,4]
train_set = data1[:,5:21]
```

2.4 Optimize the parameters

When I first tried to optimize the parameters of the DT classifier, I firstly used cross validation to optimize the parameters max depth and the minsplit. The following is the code of using cross validation to optimize the max depth.

```
def cv_score(d):
    clf = tree.DecisionTreeClassifier(max_depth=d)
    clf.fit(train_set, train_label)
    return(clf.score(train_set,train_label), clf.score(test_set, test_label))
depths = np.arange(1,10)
scores = [cv_score(d) for d in depths]
tr_scores = [s[0] for s in scores]
te_scores = [s[1] for s in scores]
tr_best_index = np.argmax(tr_scores)
te_best_index = np.argmax(te_scores)
```

The result:

```
print("bestdepth:", te_best_index+1, " bestdepth_score:", te_scores[te_best_index], '\n')
bestdepth: 7  bestdepth_score: 0.9666763820071893
```

Figure 7 The result of the two optimized parameters

```
bestmin: 0.0
bestscore: 0.9668221121150297
```

Next, I used the two parameters to create a Decision Tree classifier and then used F1-score, recall and precision to measure the effect of classification of the DT classifier.

```
model = tree.DecisionTreeClassifier(max_depth=7, min_impurity_decrease=0.0)
model.fit(train_set, train_label)
from sklearn import metrics
print("tees_score:", model.score(test_set,test_label))
y_pred = model.predict(test_set)
print("precision:",metrics.precision_score(test_label, y_pred))
print("recall rate :",metrics.recall_score(test_label, y_pred))
print("F1_score:",metrics.f1_score(test_label, y_pred))
```

The result is:

```
precision: 0.97080078125
recall rate : 0.9630885487308661
F1_score: 0.9669292870343352
```

Figure 8 The result of precision ,recall and F1 score

We can see the effect of this two parameter is good for improving the effect of classification of the DT classifier. However, I then found that there are two problems with the above model optimization methods. Firstly, the parameter is instable. Each time the training set test set is reallocated, the original parameter is not optimal. The solution is to average it multiple times. Secondly, it cannot choose more than two parameters to optimize at a times. As a result, I then use GridSearchCV to optimize the parameters.

GridSearchCV includes two parts, GridSearch and CV. GridSearch means in the selection of all the candidate parameters, through loop through, trying every possibility, and the best performance of the parameters is the final result. It's like finding the maximum value in an array. CV is the optimization method that is mentioned above—

cross validation. By using GridSearchCV, we can optimize the several parameters at the same time. However, it is a very time-consuming process so I only used it to optimize three to four parameters at the same time.

The related code and the result:

```
from sklearn.model_selection import GridSearchCV
entropy_thresholds = np.linspace(0, 1, 100)
gini_thresholds = np.linspace(0, 0.2, 100)
#Set parameters matrix
param_grid = [{'criterion': ['entropy'],
                    'min_impurity_decrease':entropy_thresholds},
                    {'criterion': ['gini'], 'min_impurity_decrease': gini_thresholds},
                    {'max_depth': np.arange(2,10)},
                    {'min_samples_split': np.arange(2,30,2)}]
clf = GridSearchCV(tree.DecisionTreeClassifier(), param_grid, cv=5)
clf.fit(train_set, train_label)
clf.fit(test_set,test_label)
print("best param:{0}\nbest score:{1}".format(clf.best_params_, clf.best_score_))
```

```
best param:{'min_samples_split': 26}
best score:0.9684738518300847
```

Figure 9 The result of GridSearchCV in DT classifier

The other classifiers all used the GridSearchCV to optimize the parameters.

2.5 The parameters of the classifiers

Table 1 The parameters of the classifiers

The classifier	The parameters
DT	max_depth=7,min_samples_split=26
ANN(Using torch)	in_features=16, hidden_features=96, out_features=3
MLP(Using sklearn)	hidden_layer_sizes=(10,10)
SVM	kernel='linear',gamma='auto'
KNN	n_neighbors=13,weights='uniform',algorithm='auto', leaf_size=30

The parameters that are not mentioned above are the default in the classifiers.

2.6 Two approaches to create the ANN classifier

When I firstly tried to create the ANN, I had use torch which we had learned in the class. However, when I then calculated the accuracy and training time, I found that the accuracy was lower than other. I think that is because I cannot optimize the related parameters well. I tried many times but the result is still no very well, so I used sklearn to create the MLP, and the result is better than that created by torch.

2.7 Ensemble

In ensemble algorithms, bagging methods form a class of algorithms which build several instances of a black-box estimator on random subsets of the original training set and then aggregate their individual predictions to form a final prediction. These methods are used as a way to reduce the variance of a base estimator, by introducing randomization into its construction procedure and then making an ensemble out of it. In many cases, bagging methods constitute a very simple way to improve with respect to a single model, without making it necessary to adapt the underlying base algorithm.

As they provide a way to reduce overfitting, bagging methods work best with strong and complex models, in contrast with boosting methods which usually work best with weak models.

To improve the classification effect, I had used the ensemble algorithm. In my ensemble algorithm, there are four classifiers, MLP (using sklearn), KNN, SVM, DT. The final accuracy is better than three of its components. In general, the ensemble had improved overall efficiency. The related code can be seen in the file called "ensemble.py".

3.Requirements:

Table 2 The IDE and package used in the project

IDE or packages	Jupyter Notebook	Python	sklearn	pandas	numpy	Seaborn	torch
version	6.0.3	3.7.6	0.23.1	1.0.1	1.18.1	0.10.0	1.6.0+cpu

4. Comparison and discussion:

4.1 Results:

Decision Tree

```
accuracy: 0.9676479160594579
training time: 0.08397865295410156
```

Figure 10 The accuracy and train time of DT

ANN (using torch)

```
accuracy : 0.9621587486641406
training time: 8.905752897262573
```

Figure 11 The accuracy and train time of ANN(torch)

MLP (using sklearn)

```
accuaracy : 0.9713397454580783
training time: 8.029109954833984
```

Figure 12 The accuracy and train time of MLP(sklearn)

SVM

```
accuracy: 0.9597784902360827
training time: 11.686148881912231
```

Figure 13 The accuracy and train time of SVM

KNN

```
accuracy: 0.9679393762751385
training time: 8.107207298278809
```

Figure 14 The accuracy and train time of KNN

Ensemble

```
accuracy: 0.9681336830855921
```

Figure 15 The accuracy of ensemble

Table 3 The accuracy and training time of the classifiers

Classifier	Accuracy/100%	training time/s
DT	0.9676	0.0840
ANN(torch)	0.9622	8.906
MLP(sklearn)	0.9713	8.029
SVM	0.9598	11.68
KNN	0.9679	8.107
Ensemble	0.9681	

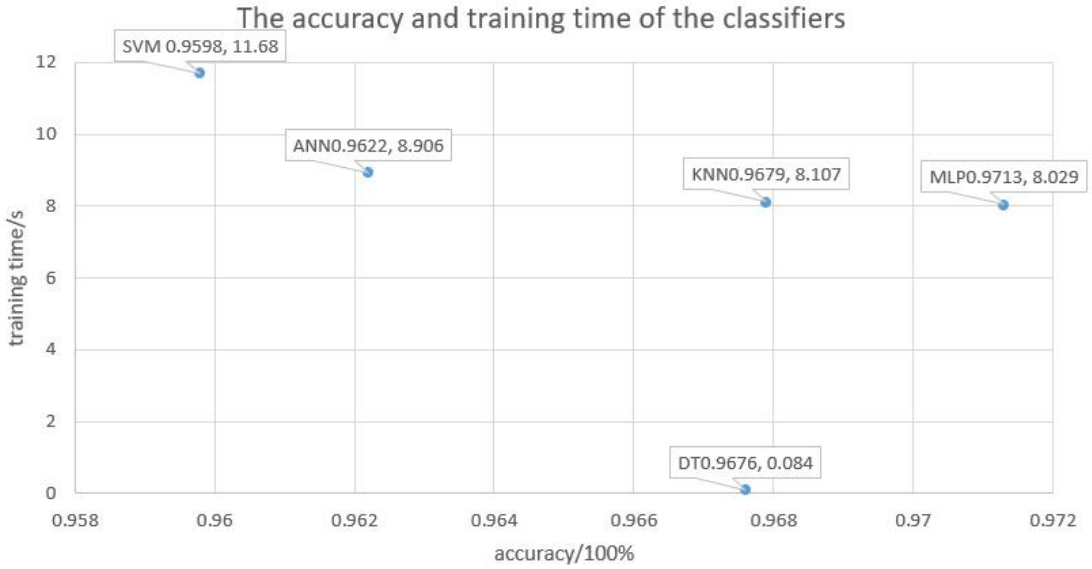

Figure 16 The accuracy and training time of the classifiers

4.2 Comparison:

Compared with other classifiers, the accuracy of the MLP(sklearn) is max. The training time of DT is least. The accuracy of SVM is the minimum and the training time of SVM is the longest.

From the results, we can know that the merit and demerit of each classifier. Decision Tree can gain feasible and effective results for large data sources in a relatively short time. Artificial Neural Network can accurately classify the data, however ANN needs

many parameters and its training time is a little long. K-NearestNeighbor is sample and effective, but its calculation progress is complicated so it training time is also a little long. Support Vector Machine can improve generalization performance but SVM has no universal solution to nonlinear problems so its training time is long.

On the whole, compared with other classifiers, I think the MLP (using sklearn) is the best classifier to predict the who will be the winner of a LoL game.

4.3 Further improvement

In this project, I had met two problems that I cannot solve. The first one is that though I used GridSearch to optimize the parameters of SVM, I cannot find the best gamma for the problem. I think this may be a reason why the training time of SVM is the longest. The second one is that when I created the ANN with torch, I firstly set the output_layer=2 because the dataset only has two label. However, the code always reports an error. I searched in the Internet but cannot find a good solution. Later, I seen one person said in a post that may add one more output layer is a solution. I did so and get a good result, but I still do not know why we should do that. The above two problems are the aspects I want to improve in the future.

5.Summary

In the project, I used 4 classifiers to classify the data. On the whole, compared with other classifiers, I think the MLP (using sklearn) is the best classifier to predict the who will be the winner of a LoL game. The accuracy is 97.13%, and the training time is 8.029s.