A binary classification of League of Legends (LoL) match records by team performances

Wang Shiqi

Abstract

This paper investigated four different approaches for automated League of Legends match records' labels (win or not) classification based on the teams' performances and match records. The team scores and some other pieces of information are included in the team's performance.

Introduction

At present, League of Legends (abbreviated as LoL) is one of the most popular and played eSports in the world. It is undeniable that the interesting point of watching games lies on the uncertainty, which implies that each team has the probability to win and sometimes an unexpected team even turns the table and snatched the victory eventually. However, it is hard for most people to watch the whole game and some companies are interested in investing in some promising teams, hence team performance data and predicted winning probability being essential.

In this project, about thirty thousand match records of two solo team gamers (team 1 and 2) are given, based on the given data, each team is supposed to be classified into two groups, winners and losers. 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. These attributes such as "FirstBlood" provide information to assist grouping. Predicting the game outcome mounts to classify the teams by their performance. The aim of this project is to build several classifiers to predict the winner in the given games, to compare all the selected classifiers according to their performance and characteristics, and to validate the prediction power through the accuracy (, precision, recall, and F1-score).

Since there are only two teams, this problem can be regarded as a binary classification problem, dividing all the match records into two label, label 1 and 2. As explained above, label 1 means team 1 is the winner in the game or vice versa. In this project, four types of classifiers are adopted, which are Decision Tree classifier, Support Vector Machine classifier, Multi-Layer Perceptron (aka. Artificial Neural Network), and ensembles (including Bagging classifier, Boost classifier, Voting classifiers), in order to automatedly solve the binary classification problem. Additionally, the test accuracies of the test sets are computed to help evaluate the models, the PR graph (Precision-Recall graph) is shown to explicitly indicate the relationship between the precision and recall, also evaluating the models, even though accuracy is good enough for the evaluation of the balanced dataset.

Automatic team records classification of LoL based on the teams' scores not only speeds up the prediction process by providing a list of suggestion but the result may potentially be more accurate than an experts, because of the uncertainty of games. What's more, this automatic classification provides the companies who intend to invest in teams with a reference.

Algorithms

1. Brief Introduction of the classifiers

As mentioned in the Introduction, four main types of classifiers are utilized to better classify the match records, which are:

1. Decision Tree
2. SVM (Support Vector Machine)
3. MLP (Multi-Layer Perceptron)
4. Ensemble

All of the four kinds of classifiers are appropriate for binary classification, with high accuracy. A general description of the used classifiers is shown in the Fig.1

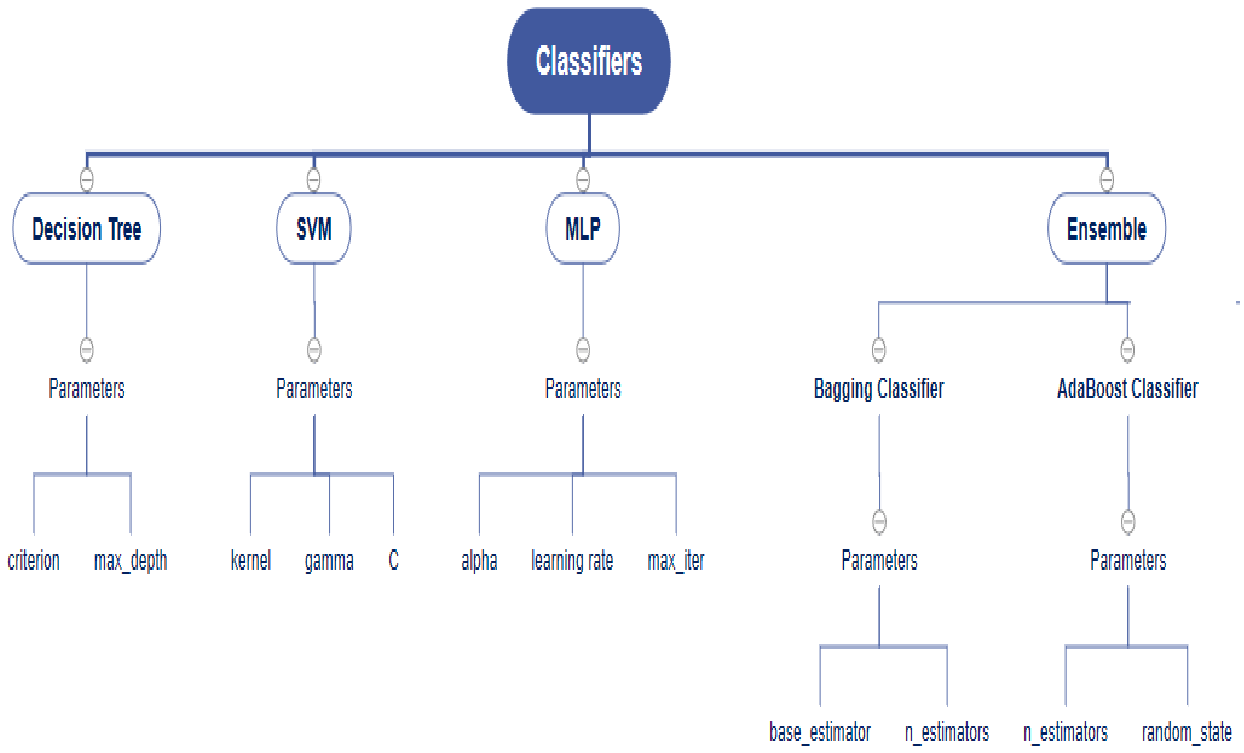


Fig.1 A general description of the used classifiers

The adopted classifiers are different with their principles and parameters, which will be introduced in the latter.

2. Classification Process

In this classification method, even though the classifiers are different, the main process of classification is similar. The working process is shown in the flow chart below:

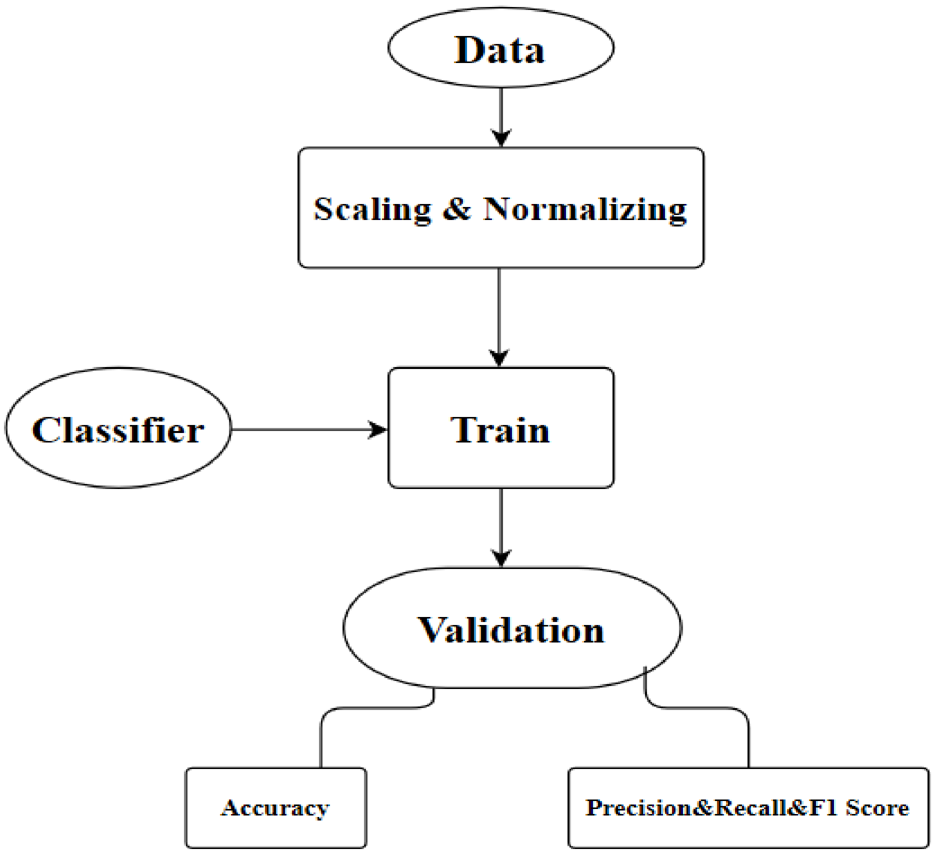
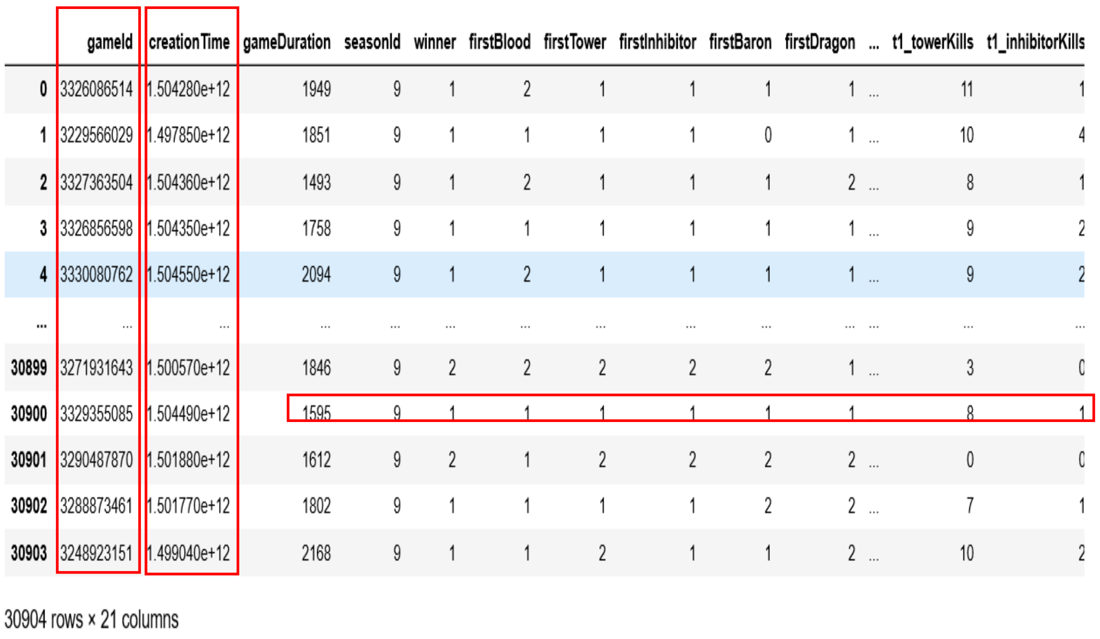


Fig.2 Classification process

2.1 Data Preprocessing

The first stage is the data preprocessing. Because the range of the multiple features in the dataset are obviously different, just as shown in the Fig.3, and some features like "gameId" and "creationTime" are just for indexing, which are of no use to the classification, the training outcome will worse without the data preprocessing. During the data preprocessing, some useless features can be deleted, and the residual features will be scaled before feeding to the models, and then the data will be normalized into the same range [0,1] (aka. Min-Max scaling).



	gameId	creationTime	gameDuration	seasonId	winner	firstBlood	firstTower	firstInhibitor	firstBaron	firstDragon	...	t1_towerKills	t1_inhibitorKills
0	3326086514	1.504280e+12	1949	9	1	2	1	1	1	1	...	11	1
1	3229566029	1.497850e+12	1851	9	1	1	1	1	0	1	...	10	4
2	3327363504	1.504360e+12	1493	9	1	2	1	1	1	2	...	8	1
3	3326856598	1.504350e+12	1758	9	1	1	1	1	1	1	...	9	2
4	3330080762	1.504550e+12	2094	9	1	2	1	1	1	1	...	9	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
30899	3271931643	1.500570e+12	1846	9	2	2	2	2	2	1	...	3	0
30900	3329355085	1.504490e+12	1595	9	1	1	1	1	1	1	...	8	1
30901	3290487870	1.501880e+12	1612	9	2	1	2	2	2	2	...	0	0
30902	3288873461	1.501770e+12	1802	9	1	1	1	1	2	2	...	7	1
30903	3248923151	1.499040e+12	2168	9	1	1	2	1	1	2	...	10	2

30904 rows × 21 columns

Fig.3 Visualization of the dataset

2.2 Classifiers & Training

The second step is to use the preprocessed data to train the selected models. The dataset will be split into two sets, the training set and the test set. Use the training set to feed the models, and then get higher accuracy. The four types of the adopted classifiers and their parameters will be introduced below.

A. Decision Tree

Decision Tree (DT) is 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. The working principle is to split the dataset based on the given condition and the measure criterion until all tuples in the leaf node belong to the same class. In this problem, the Decision Tree classifier will Select an attribute A_i first, and then partition the training dataset based on the different values of A_i. Secondly, for each $D_k \in D$, create a node and add an edge between D and D_k with label as the A_i's attribute value in D_k. Recursively repeat the former operation until all tuples in D_k belongs to the same class.

In Python, the implement of Decision Tree is based on DecisionTreeClassifier, whose parameters are criterion and max_depth. Criterion is used for measuring the quality of a split. There are two types, “gini” and “entropy”, and “entropy” is used in this project. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. Max_depth is used to set the maximum depth of the tree.

B. SVM

The classifier separates data points using a hyperplane with the largest amount of margin. The core idea of SVM is to find a maximum marginal hyperplane(MMH) that best divides the dataset into classes. The figure shown below helps illustrate the principle.

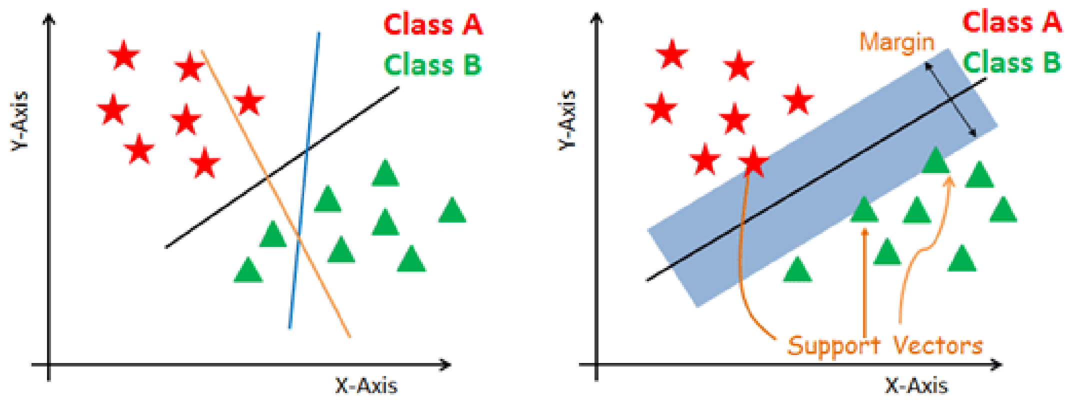


Fig.4 The principle of SVM

When classifying, firstly generate hyperplanes which segregates the classes in the best way. In Fig.4, left-hand side figure showing three hyperplanes black, blue and orange. Here, the blue and orange have higher classification error, but the black is separating the two classes correctly. Secondly, select the right hyperplane with the maximum segregation from the either nearest data points as shown in the right-hand side figure. In this project, the implementation of SVM in Python using scikit-learn is through the model "SVC" (for classification). The mainly used parameters are kernel, gamma, and C. The detailed introduction is shown in the List.1.

kernel (default='rbf')	Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used.
gamma (float)	Kernel coefficient for 'rbf', 'poly' and 'sigmoid'
C (float, default=1.0)	Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive.

List.1 The parameters of SVM

C. MLP

MLP is a model inspired by brain, it follows the concept of neurons present in our brain. Several inputs are being sent to a neuron along with some weights, then for a corresponding value neuron fires depending upon the threshold being set in that neuron. Its algorithms can be divided into three phases. The first phase is initializing phase, where the weights are initialized with random values. The second phase is the training phase, for each training sample, compute the output and use the error (the difference between real output and computed one) to update the weights. The third phase is a recall phase, repeating the functions above until stopping condition is met.

The parameters of the MLPClassifier in Python are shown in the List.2.

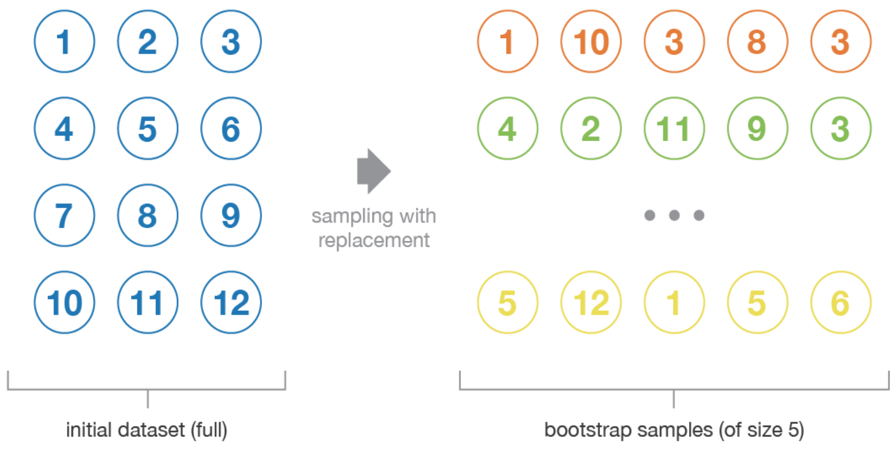
alpha (float, default=0.0001)	L2 penalty (regularization term) parameter
learning_rate ({'constant', 'invscaling', 'adaptive'}, default='constant')	Learning rate schedule for weight updates.
max_iter (int, default=200)	Maximum number of iterations.

List.2 The parameters of MLP

D. Ensemble

D(1) Bagging classifier

A Bagging classifier is an ensemble meta-estimator that fits base classifiers (Decision Tree) each on random subsets of the original dataset and then aggregate their individual predictions. Bagging is a powerful ensemble method which helps to reduce variance, and by extension, prevent overfitting. The most significant part in this approach is the bootstrapping, which is the process of generating bootstrapped samples from the given dataset. The samples are formulated by randomly drawing the data points with replacement. The resampled data contains different characteristics that are as a whole in the original data. Tree classifiers are built on each bootstrap sample, and the individual weak learners are independent of each other. Finally, the different outcomes of the classifiers are combined, improving the accuracy. The bootstrapping process is represented below:



The parameters of the BaggingClassifier are base_estimators and n_estimators. The detailed information are displayed in the List.3.

base_estimators	The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a decision tree.
n_estimators	The number of base estimators in the ensemble.

List.3 The parameters of BaggingClassifier

D(2) AdaBoost Classifier

AdaBoost classifier combines multiple Decision Tree classifiers to increase the accuracy of classifiers. AdaBoost is an **iterative** ensemble method. AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that you will get high accuracy strong classifier. The basic concept behind AdaBoost is to set the weights of classifiers and training the data sample in each iteration such that it ensures the accurate predictions of unusual observations. Ada is short for adaptive, and the reason is that this approach is an iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records. The mainly used parameters are n_estimator and random_state, since the base classifiers are Decision Tree, it will not be repeatedly introduced. The parameters are displayed in the following list.

n_estimators	The maximum number of estimators at which boosting is terminated.
random_state	Controls the random seed given at each base_estimator at each boosting iteration. Thus, it is only used when base_estimator exposes a random_state.

List.4 The parameters of the AdaBoost Classifier

3. Validation

The final section is to validate the classification models mentioned above. The general measure is taking advantage of the accuracy. However, using accuracy to evaluate the prediction power is appropriate for balanced dataset. When it comes to the imbalanced dataset, precision, recall, and F1 Score are more suitable.

Requirements

The prerequisite package used in this project is the “sklearn”, which is essential for machine learning in Python. Sklearn contains simple and efficient tools for predictive data analysis, such as classification models, regression models, clustering models, and preprocessing modules. In this project, in order to implement the classification, the package “sklearn” is supposed to be installed, via the command “conda install -c anaconda scikitlearn” in Anaconda.

Results

The performance of the four types of classification models and the classifiers training time are shown in Table.1

	Accuracy	Precision	Recall	Time(s)
Decision Tree	0.9638	0.9638	0.9638	0.1151
SVM(default)	0.9677	0.9677	0.9677	2.9875
SVM(parameters)	0.9635	0.9635	0.9635	5.2569
MLP(parameters)	0.9692	0.9692	0.9692	52.7766

MLP(parameters)	0.9691	0.9691	0.9691	44.7167
Bagging	0.9688	/	/	41.1585
Adaboost	0.9654	/	/	11.0708

Table.1 Training time, test accuracy, precision and recall of all classification models.

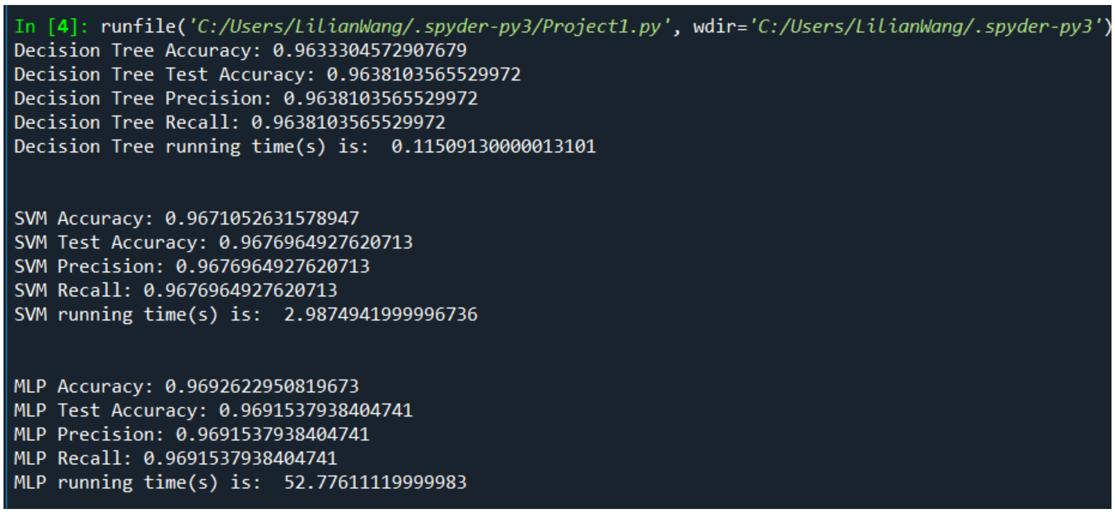
The evaluation is based on the following metrics:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

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

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

The default parameters work better for SVM and MLP models. The results of the classification on IPython Console are shown in the figures.

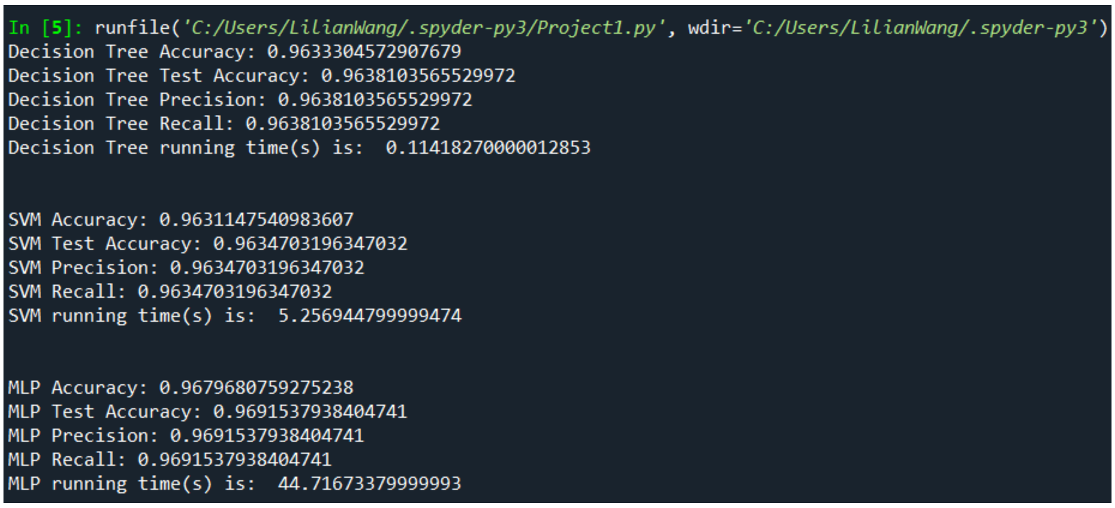


```
In [4]: runfile('C:/Users/LilianWang/.spyder-py3/Project1.py', wdir='C:/Users/LilianWang/.spyder-py3')
Decision Tree Accuracy: 0.9633304572907679
Decision Tree Test Accuracy: 0.9638103565529972
Decision Tree Precision: 0.9638103565529972
Decision Tree Recall: 0.9638103565529972
Decision Tree running time(s) is:  0.11509130000013101


SVM Accuracy: 0.9671052631578947
SVM Test Accuracy: 0.9676964927620713
SVM Precision: 0.9676964927620713
SVM Recall: 0.9676964927620713
SVM running time(s) is:  2.9874941999996736


MLP Accuracy: 0.9692622950819673
MLP Test Accuracy: 0.9691537938404741
MLP Precision: 0.9691537938404741
MLP Recall: 0.9691537938404741
MLP running time(s) is:  52.77611119999983
```

Fig.5 result_1 default Decision Tree, SVM and MLP

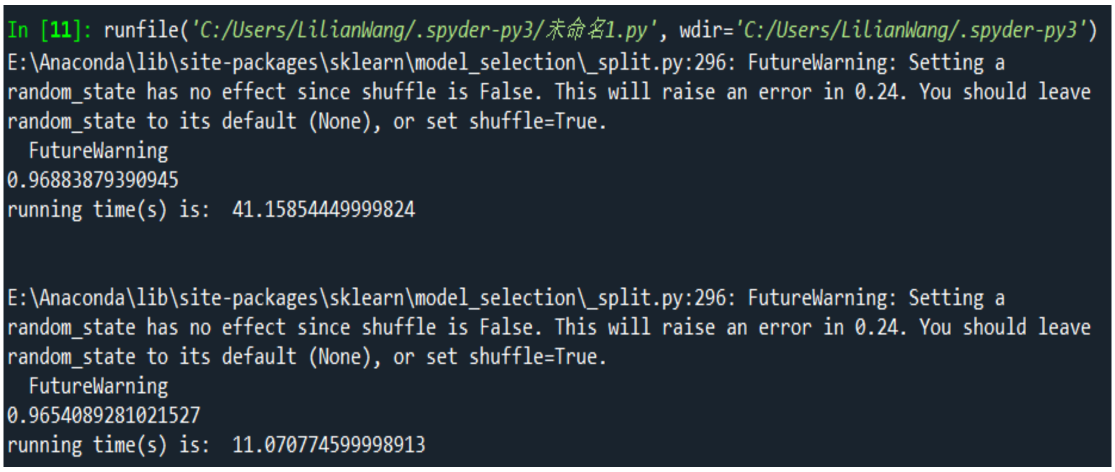


```
In [5]: runfile('C:/Users/LilianWang/.spyder-py3/Project1.py', wdir='C:/Users/LilianWang/.spyder-py3')
Decision Tree Accuracy: 0.9633304572907679
Decision Tree Test Accuracy: 0.9638103565529972
Decision Tree Precision: 0.9638103565529972
Decision Tree Recall: 0.9638103565529972
Decision Tree running time(s) is:  0.11418270000012853


SVM Accuracy: 0.9631147540983607
SVM Test Accuracy: 0.9634703196347032
SVM Precision: 0.9634703196347032
SVM Recall: 0.9634703196347032
SVM running time(s) is:  5.256944799999474


MLP Accuracy: 0.9679680759275238
MLP Test Accuracy: 0.9691537938404741
MLP Precision: 0.9691537938404741
MLP Recall: 0.9691537938404741
MLP running time(s) is:  44.71673379999993
```

Fig.6 result_2 parameters-changed Decision Tree, SVM and MLP



```
In [11]: runfile('C:/Users/LilianWang/.spyder-py3/未命名1.py', wdir='C:/Users/LilianWang/.spyder-py3')
E:\Anaconda\lib\site-packages\sklearn\model_selection\_split.py:296: FutureWarning: Setting a
random_state has no effect since shuffle is False. This will raise an error in 0.24. You should leave
random_state to its default (None), or set shuffle=True.
  FutureWarning
0.96883879390945
running time(s) is:  41.15854449999824


E:\Anaconda\lib\site-packages\sklearn\model_selection\_split.py:296: FutureWarning: Setting a
random_state has no effect since shuffle is False. This will raise an error in 0.24. You should leave
random_state to its default (None), or set shuffle=True.
  FutureWarning
0.9654089281021527
running time(s) is:  11.070774599998913
```

Fig.7 result_3 Bagging and Adaboost

The decision tree diagram is shown in the Fig.8

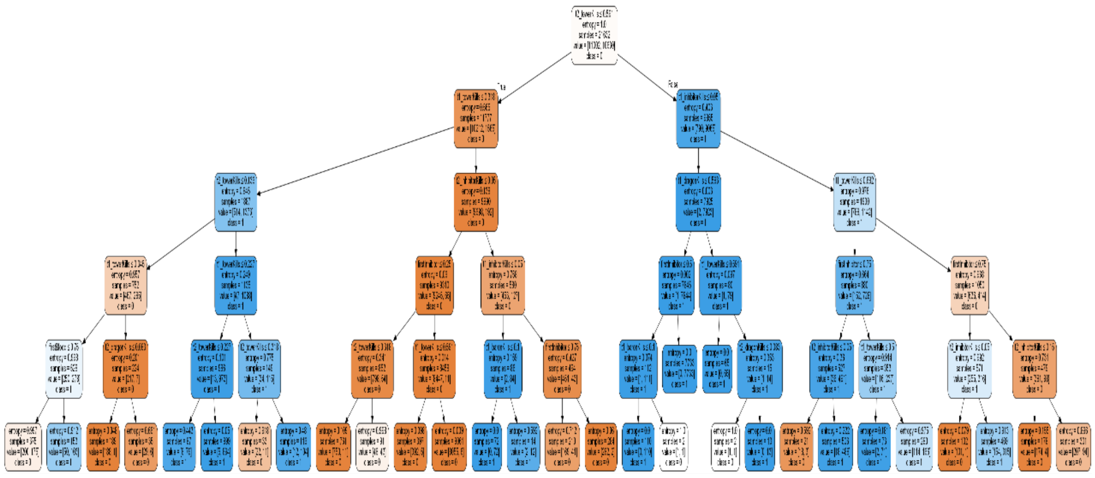


Fig.8 The decision tree diagram

Comparison and Discussion

1. Classifiers Comparison

In this project, four types of classification models are built to fit the data and classify the LoL match records, that is, to predict the winners in each game. The principle of the four classifiers are different, although their accuracies to the test data is spectacularly close. Based on the experiment results, the following will compare the each classifiers from their performance, advantages, and disadvantages.

	Accuracy	Speed	Easy to overfit?	Advantage	Disadvantage
Decision Tree	Medium	Fast	Yes	1. The splitting can be explicitly displayed 2. Not necessary	1. Easy to overfit 2. The tree occupies too much memory

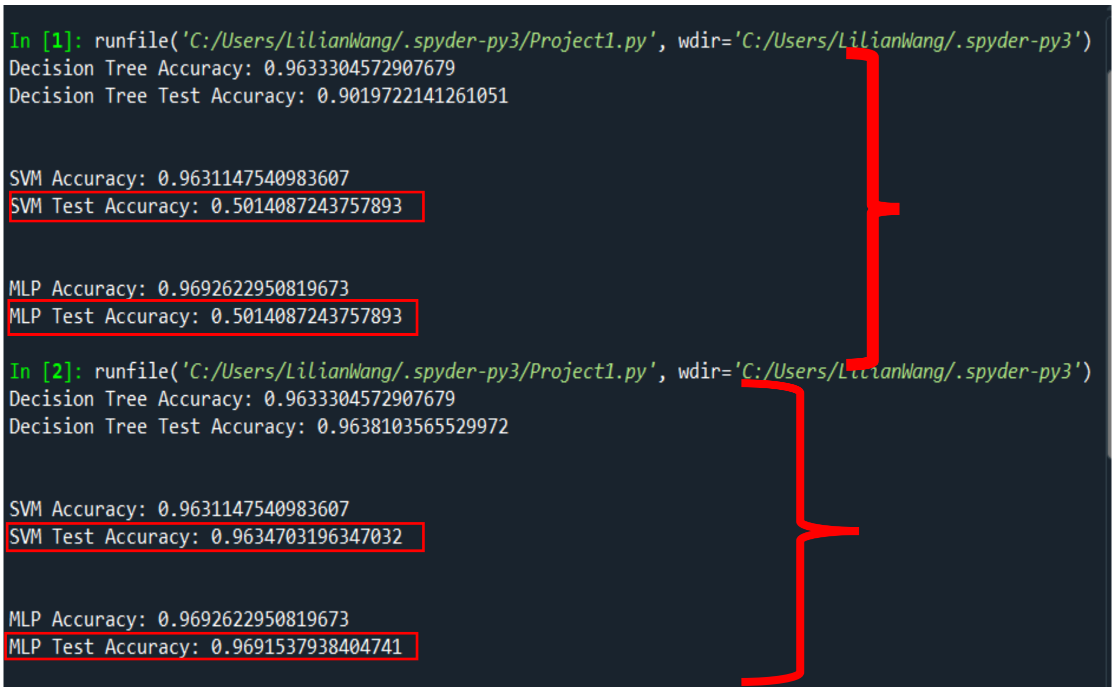
				to scale the data	3. Unstable
SVM	High	Medium	Yes	1. The accuracy is high. 2. more effective in high dimensional spaces	1. Vulnerable to noise data. 2. Accuracy reduces for large datasets. 3. The parameters are hard to modify
MLP	High	Relatively Slow	No	1. Good at mapping, and the mapping can be nonlinear 2. High accuracy	1. May be trapped by local minima 2. Hard to balance between the accuracy and training time 3. The number of hidden neurons are hard to set
Bagging	High	Medium	No	1. helps in the reduction of variance, hence eliminating the overfitting of models 2. combines weak learners to a single strong learner.	1. Highly affected by the original individual classifiers, because it uses the mean predictions 2. Time complexity is high
Adaboost	High	Relatively Slow	Yes	1. no need to use many parameters 2. improve the accuracy of weak classifiers	1. Sensitive to noisy data 2. Easy to overfit

Since the dataset of the LoL match records is large and the classification model is supposed to be generalized to untrained data to realize the function of predicting the winning probability for profits or something else, the best appropriate model in this project is the Bagging classifier according to the Comparison Table. Because its accuracy to the test data is high, and it combines the prediction results of weak tree classifiers to a single stronger learner. Furthermore, this model reduces the probability of overfitting, improving the generalization ability.

2. Analysis and Discussion

2.1 The understanding of scaling

As mentioned in the Algorithms, scaling is an essential part of data preprocessing, help to reduce the negative impact on the classification models. During the experiment, a fact has been observed that the test data prediction accuracy is not heavily affected without test data scaling when using Decision Tree, but the accuracy is greatly influenced without test data scaling when using SVM and MLP. The figure below helps illustrate this observation.



```
In [1]: runfile('C:/Users/LilianWang/.spyder-py3/Project1.py', wdir='C:/Users/LilianWang/.spyder-py3')
Decision Tree Accuracy: 0.9633304572907679
Decision Tree Test Accuracy: 0.9019722141261051


SVM Accuracy: 0.9631147540983607
SVM Test Accuracy: 0.5014087243757893


MLP Accuracy: 0.9692622950819673
MLP Test Accuracy: 0.5014087243757893

In [2]: runfile('C:/Users/LilianWang/.spyder-py3/Project1.py', wdir='C:/Users/LilianWang/.spyder-py3')
Decision Tree Accuracy: 0.9633304572907679
Decision Tree Test Accuracy: 0.9638103565529972


SVM Accuracy: 0.9631147540983607
SVM Test Accuracy: 0.9634703196347032


MLP Accuracy: 0.9692622950819673
MLP Test Accuracy: 0.9691537938404741
```

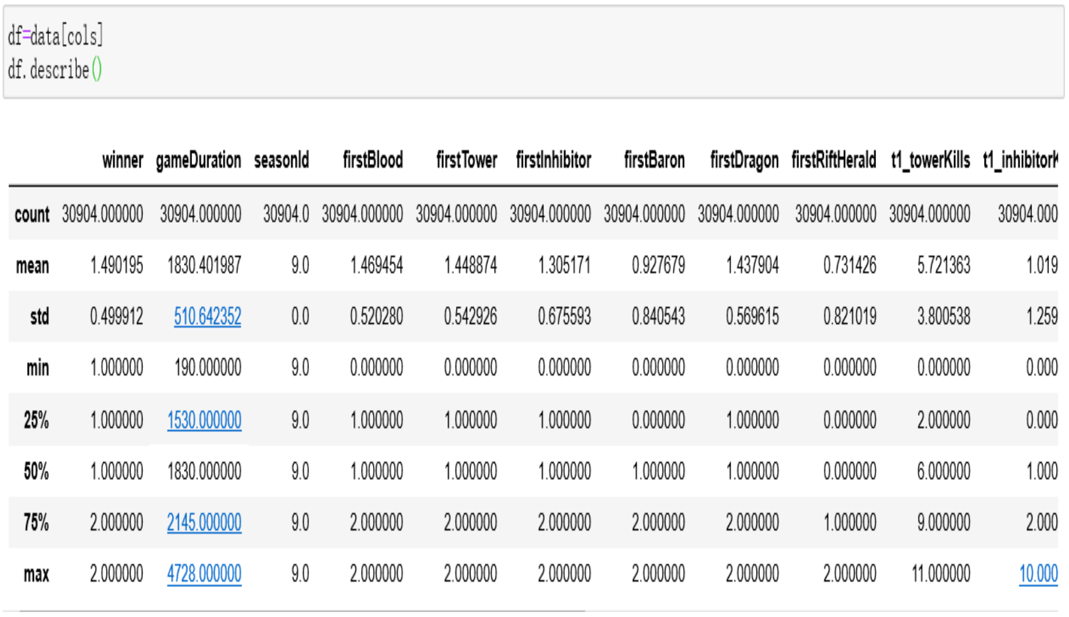
Fig.9 The outcome comparison before and after scaling

The upper is the outcome before scaling the test data, the test accuracy of SVM and MLP after scaling the test data have been improved by 45% around. From this observation, an assumption can be drawn that SVM and MLP is much more sensitive to data scaling. MLP is a approach that uses gradient descent as an optimization technique. The presence of feature value will affect the step size of the gradient descent, so the difference in ranges of features will cause different step sizes for each feature. Hence, in order to ensure the gradient descent converges, the data need to be scaled. As for SVM, it is a distanced-based algorithm, so it is most affected by the range of features. However, the tree-based algorithm is insensitive to the scale of the features. The reason is that a decision tree splits a node only based on a single feature each time and this split on a feature is not influenced by other features.

2.2 The characteristic of the dataset

In practical, a great many classification problems are imbalanced, wit skewed classes. According to the confusion matrix, the accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$, but if the dataset is imbalanced, assuming that there are two class and the number of Class 1 samples are much larger than that of Class 2, when all test examples are predicted as Class 1, then the accuracy is still close to 100%. Consequently, the accuracy is not suitable for this

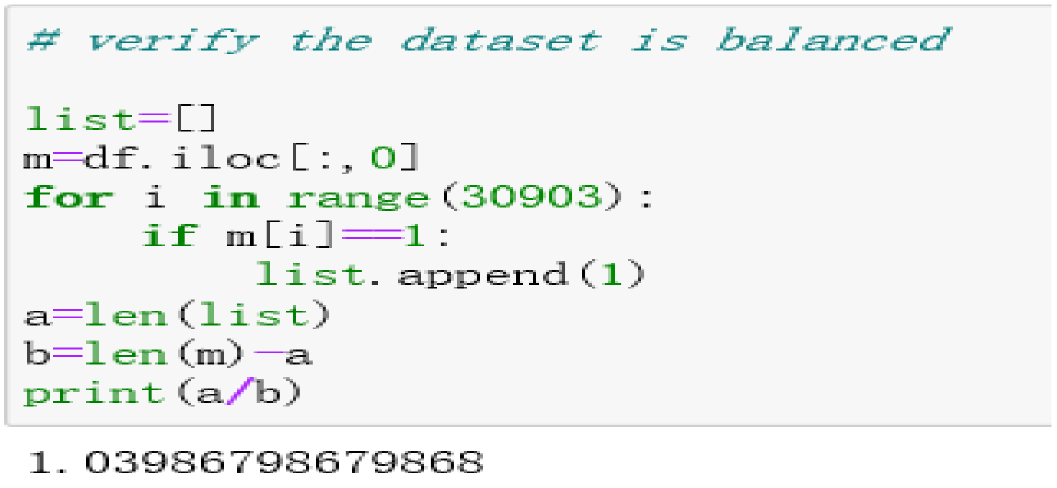
situation. Precision and recall are more appropriate in such case.
Before testing the prediction power, the type of the dataset is supposed to check. In this project, I use the Jupyter Notebook to visualize the LoL match records dataset, and to describe how the data is distributed. The result is shown below.



```
df=data[cols]
df.describe()
```

	winner	gameDuration	seasonId	firstBlood	firstTower	firstInhibitor	firstBaron	firstDragon	firstRiftHerald	t1_towerKills	t1_inhibitork
count	30904.000000	30904.000000	30904.0	30904.000000	30904.000000	30904.000000	30904.000000	30904.000000	30904.000000	30904.000000	30904.000
mean	1.490195	1830.401987	9.0	1.469454	1.448874	1.305171	0.927679	1.437904	0.731426	5.721363	1.019
std	0.499912	510.642352	0.0	0.520280	0.542926	0.675593	0.840543	0.569615	0.821019	3.800538	1.259
min	1.000000	190.000000	9.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	1.000000	1530.000000	9.0	1.000000	1.000000	1.000000	0.000000	1.000000	0.000000	2.000000	0.000
50%	1.000000	1830.000000	9.0	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	6.000000	1.000
75%	2.000000	2145.000000	9.0	2.000000	2.000000	2.000000	2.000000	2.000000	1.000000	9.000000	2.000
max	2.000000	4728.000000	9.0	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	11.000000	10.000

Fig.10 Data Distribution



```
# verify the dataset is balanced

list=[]
m=df.iloc[:,0]
for i in range(30903):
        if m[i]==1:
                list.append(1)
a=len(list)
b=len(m)-a
print(a/b)
```

```
1.0398679867986
```

Fig.11 verify the dataset is balanced

As Fig.11 shown, the number of samples of the two samples are close, therefore it is a balanced dataset. Furthermore, there is no need using precision and recall to evaluate the models. However, precision and recall measures can be imported to increase the result of the models. What's more, in balanced-class problem,

Conclusion

In this project, the LoL match records binary classification problem is solved by building four kinds of classifiers, with the winner team of each game being predicted, based on the match records features. The accuracy of each model is higher than 96%,

which validates the prediction power of the classification models and the reliability of the given dataset. Since the dataset is balanced, which has been proved in the Comparison and Discussion, the precision, recall scores are the same as the accuracy score. The Bagging classifier is the most suitable model for this problem, which improves model precision by using ensembles and prevent overfitting. During this project, the principles of the classification models have been learned, and the strengths and weaknesses of each model have been discussed. Furthermore, the importance of data preprocessing has been observed, which have a great impact on model training and testing.
This project makes sense for winning probability prediction, especially for those companies who intend to invest in LoL teams.

References

[1]. Introduction to Bagging and ensemble methods
https://blog.paperspace.com/bagging-ensemble-methods/
[2]. A Guide to AdaBoost: Boosting To Save The Day
https://blog.paperspace.com/adaboost-optimizer/
[3]. https://www.scientific.net/paper-keyword/imbalanced-dataset
[4]. API references
https://scikit-learn.org/stable/modules/classes.html
[5]. Support Vector Machines with Scikit-learn
https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python
[6]. Feature scaling for machine learning
https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/