Classification of League of Legends (LoL) Results Using Ensemble Model

I. Problem Introduction

The problem is applying classification algorithms to predict the game result of LOL based on the given game data. A number pieces of labeled data are given to train the model. The test set is used to test the performance of the model. The features of the given data include first baron, dragon, tower, blood, inhibitor and so on. In this project, tree classification, support vector classification and multi-layer perception classification algorithms are used separately and then a boosting ensemble is implemented. Detailed algorithm introduction is discussed in part two. Part three and four talk about the experiments, which contain model results and evaluation. Comparison between different models and discussion about the performance are given in part five.

II. Algorithm

The procedure of all three models are similar, which is create a classifier, fit the data to train, do the prediction and finally model performance evaluation.

Fig. 1 shows a detailed execution flow.

The first used algorithm is decision tree classifier. 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 main reason to choose decision tree is that it is a white-box. So, the result can be explained very easily. What's more, we can not only do the classification but also can know what are the important features dominating the result of a game.

Gini criterion and best split method are chosen as the parameters. There is no limitation on the number of maximum depth and minimum samples split.

The second classifier is support vector classifier (SVC). Before fitting the data, a pipeline is made to standardize the features and set the parameters of classifier. The kernel chosen is rbf (radial-basis function kernel) and the gamma is set to be auto. The reason to choose rbf is that this kernel is infinitely differentiable and covariance function has mean square derivatives of all orders, and are thus very smooth. So, it is stable when the features locate in high dimensional space.

The third classifier is multi-layer perception classifier with 10x10 sized hidden layer and the activation function tanh.

iv. After all three models are trained, a boosting ensemble is used to get the final prediction. Every classifier votes a result, take the candidate with maximum votes as the final result.

As the accuracy of each classifier is near, the weight of vote is uniform.

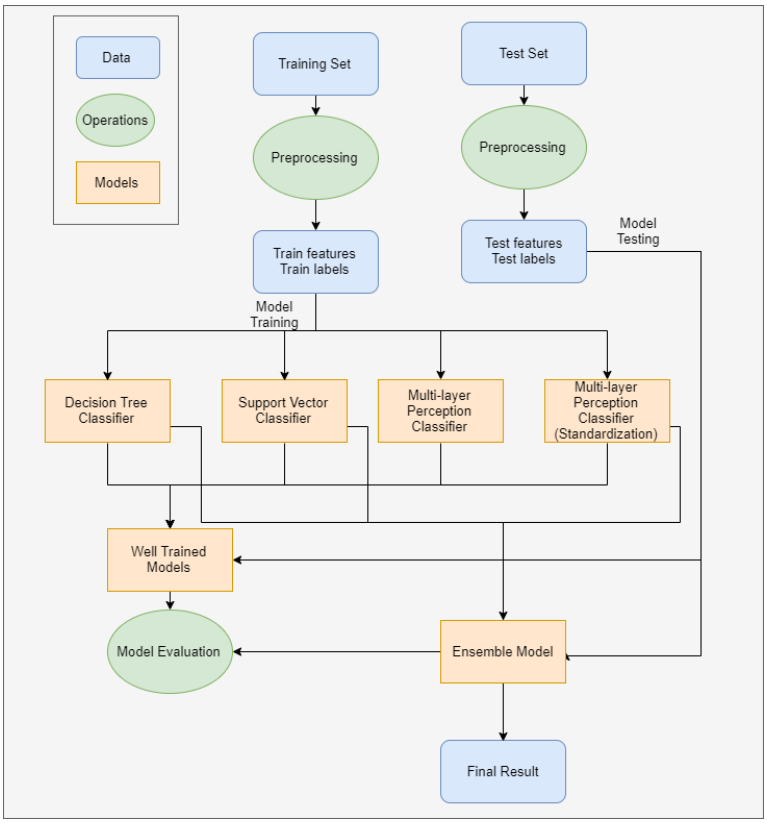

Fig. 1 Flow Chart of Execution

III. Experimental Requirements

i. Environments

The project mainly uses scikit-learn library. Scikit-learn is an open source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities. Numpy and pandas are libraries containing matrix operation which are used for data preprocess. The version of scikit-learn, numpy and pandas are 0.23.2, 1.16.5 and 0.25.1.

ii. Data preprocessing

In order to reduce the training time and increase the accuracy, some unrelated features are abandoned. The unrelated features are thought to be game ID, game duration, season ID. Furthermore, both train and test set are split to features and labels. After preprocessing, the shape of training features is (30904,17) and label is (30904,1). The shape of test features is (20586,17) and the label is (20586,1).

IV. Experimental Results

i. Decision tree

```
In [18]: 1 accuracy_score(test_labels, dt_pred)
Out[18]: 0.9602642572622171
```

Fig. 2 Accuracy of DT

As shown in Fig. 2, the accuracy of DT is 0.96. In order to see whether

both player 1 and 2 are well predicted and both precision and recall are good, f1-score is computed.

$$f1 = \frac{2 * precision * recall}{(precision + recall)}$$

The confusion matrix and f1-score of DT model are shown in Fig. 3 and Fig. 4.

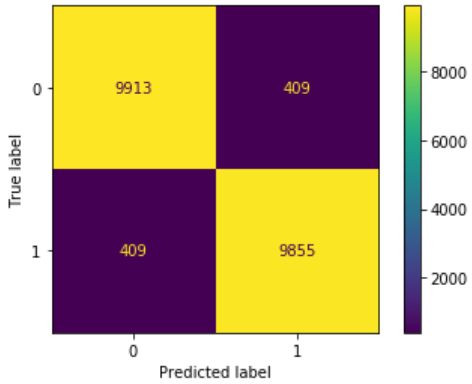

Fig. 3 Confusion Matrix of DT

```
In  [30]:  1  f1_score(test_labels, dt_pred)
Out[30]:  0.9603758961441581
```

Fig. 4 F1-score of DT

The split and growth of the tree are saved in tree.pdf.

What's more, a question we think highly of is what is the most important stuff resulting win or loss of a game. The relative rank of a feature used as a decision node in a tree can be used to asses the relative importance of that feature. The method forest.importance_ can be used to compute the importance of each features. After computing, rank them form most important to least important, then use bar plot to represent the result as shown in Fig. 5 Fig. 6.

```
Feature ranking:
1. feature t2_towerKills (0.241728)
2. feature t1_towerKills (0.208507)
3. feature firstInhibitor (0.153350)
4. feature t1_inhibitorKills (0.083066)
5. feature t2_inhibitorKills (0.081228)
6. feature firstBaron (0.035274)
7. feature firstTower (0.033729)
8. feature t1_dragonKills (0.032117)
9. feature t2_dragonKills (0.030585)
10. feature creationTime (0.025952)
11. feature t2_baronKills (0.019391)
12. feature t1_baronKills (0.019244)
13. feature firstDragon (0.014136)
14. feature firstBlood (0.007023)
15. feature t2_riftHeraldKills (0.006123)
16. feature t1_riftHeraldKills (0.004616)
17. feature firstRiftHerald (0.003931)
```

Fig. 5 Feature Ranking

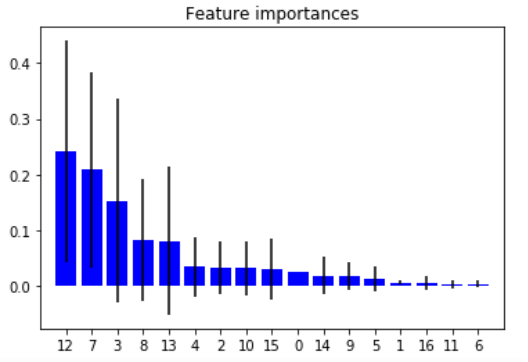

Fig. 6 Importance of Features

After the analysis we know killing enemy under tower is the most important feature dominating the result of game. First inhibitor is also of high importance.

ii. SVC

The accuracy and f1-score of SVC are shown in Fig. 7.

```
In  [38]:  1  svc.score(test_features, test_labels)
Out[38]:  0.9706110949188769

In  [39]:  1  f1_score(test_labels, svc_pred)
Out[39]:  0.9708419682876283
```

Fig. 7 Accuracy and f1-score of SVC

One thing should be pointed out is that the training time of SVC is far longer than the other two method, which is 48.4 seconds. And the other two methods are

just 0.13s and 26.0s.

iii. MLP

At first, the performance of MLP classifier is the worst, with accuracy only 0.50 and f1-score 0.67 as shown in Fig. 8.

```
In [43]: 1 mlp.score(test_features, test_labels)
Out[43]: 0.5014087243757893

In [44]: 1 f1_score(test_labels, mlp.predict(test_features))
Out[44]: 0.667917691212631
```

Fig. 8 Accuracy and f1-score of MLP 1

Then a feature standardization is used and train MLP classifier again, the results are improved significantly, as Fig. 9 shows.

```
In [48]: 1 mlp.score(X_test, test_labels)
Out[48]: 0.9675993393568445

In [49]: 1 mlp_pred = mlp.predict(X_test)

In [50]: 1 f1_score(test_labels, mlp_pred)
Out[50]: 0.9675851679059144
```

Fig. 9 Accuracy and f1-score of MLP 2

iv. Ensemble

After all three models are well trained, a ensemble is made to generate the final result. The method of ensemble is boosting. The weights of each models are the same since there is no significant difference among the performance of three models.

The code of ensemble implementation is given in Fig. 10.

```
In [54]:
_1 = 0
_2 = 0
final_pred = []
#Initialisation

for i in range(0,len(dt_pred)):
    #Go through all results
    #
    if dt_pred[i]==1:
        _1+=1
    if dt_pred[i]==2:
        _2+=1

    if svc_pred[i]==1:
        _1+=1
    if svc_pred[i]==2:
        _2+=1

    if mlp_pred[i]==1:
        _1+=1
    if mlp_pred[i]==2:
        _2+=1

    if _1>=_2:
        #Comparison between votes of 1win and 2win
        final_pred.append(1)
    else:
        final_pred.append(2)

    _1=0
    _2=0
    #After each cycle, clear the number of votes.
```

Fig. 10 Implementation of Ensemble

Since there is no encapsulated method to compute the accuracy, MAE (mean absolute error) is computed to assess the accuracy.

$$MAE = \frac{\sum_{i=1}^{n} |y - \hat{y}|}{n}$$

```
In [57]: 1 error = test_labels - final_pred

In [58]: 1 ensemble_score = 1-np.sum(np.abs(error))/len(error)
```

Fig. 11 MAE Implementation

The performance of the final result is shown in Fig. 12.

```
In [59]: 1 print("The accuracy of ensembled model is %f",ensemble_score)
         The accuracy of ensembled model is %f 0    0.97066
         dtype: float64

In [60]: 1 f1_score(test_labels,final_pred)
Out[60]: 0.970790211819325
```

Fig. 12 Accuracy and f1-score of Ensemble

V. Comparison and Discussion

Fig. 13 gives a comparison of three models and the ensemble model.

Classifier	Accuracy	f1-score	Training time (s)
Decision Tree	0.959487	0.959659	0.131646633
SVC	0.970611	0.970842	48.39563012
MLP without standardization	0.501409	0.667918	0.130649805
MLP with standardization	0.967599	0.967585	26.01059914
Ensemble	0.97066	0.97079	

Fig. 13 Comparison of Models 1

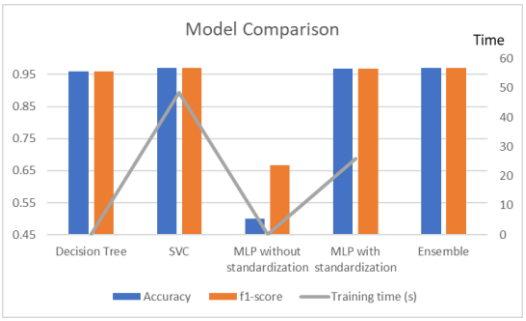

Fig. 14 Model Comparison 2

An interesting thing is why standardization improves the performance of MLP significantly but elongates the training time. This is because MLP is highly sensitive to the scale.

$$f(x) = W_2 g(W_1^T x + b_1) + b_2$$

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$Loss(\hat{y}, y, W) = -y \ln \hat{y} - (1-y) \ln (1-\hat{y}) + \alpha ||W||_2^2$$

$$Loss(\hat{y}, y, W) = \frac{1}{2} ||\hat{y} - y||_2^2 + \frac{\alpha}{2} ||W||_2^2$$

$$W^{i+1} = W^i - \epsilon \nabla Loss_W^i$$

From the result we can see that all three models perform not bad and the ensemble successfully improve the accuracy and f1-score. SVC and MLP with standardization consume more time than DT and MLP without standardization. Fig. 14 shows the result intuitively.

Explaining with less rigorous, from the learn procedure we know that if x has a larger scale, the weights will change faster, so it converges faster. However, the result will fluctuate beside the optimal. So, the accuracy is bad.

This also suggests that before applying algorithms, it is better to know the advantages and disadvantages and some implementation tips. Better preprocessing of data usually helps learn faster and gets a better performance.

VI. Summary

In this project, three different models are trained to predict the result of LOL game. After training, an ensemble is made to successfully improve the performance. The final accuracy and f1-socre of model are above 97%. Besides, more important features affecting the result are founded to help us understand the game better. From the process of improving performance of MLP classifier, I have known the importance of data preprocess

and some practical tips when applying algorithms.