Use Supervised Learning Algorithm in Scikit-learn Module to Achieve Classification for LOL Win/Loss

■ Introduction

This project is going to apply classification algorithms to achieve accurate classification. It is going to find the best strategy in lol to win the game. Two datasest consisting in all fifty thousand samples are used in this project, one for training and the other one for test. To achieve the binary classification, scikit-learn module in python are used in this project. The base method is as follow, firstly observe the data characteristic and then base on this to do data preprocessing, next apply four classification algorithms and grid search to select the best parameters, finally based on different kinds of evaluation methods and training time to evaluate the model and make the discussion. In random forest features with higher importance will be visualized.

■ Algorithms

Algorithms	Algorithm Introduction	Used Parameters	Parameters Introduction
Decision Tree	The decision tree algorithm adopts tree structure and uses layer upon layer reasoning to achieve the final classification. In the prediction, a certain attribute value is used to judge at the internal node of the tree, and according to the judgment result to decide which branch node is entered, until it reaches the leaf node and the classification result is obtained.	max_depth	The maximum depth of the tree

Random Forest	Random forest is one kind of bagging in ensemble learning. Random forest is composed of many decision trees, and there is no correlation between different decision trees. When we carry out the classification task, new input samples come in and each decision tree in the forest is judged and classified separately. Each decision tree will get its own classification result. Which of the classification results of the decision tree has the most classification, then the random forest will regard this result as the final result	max_ features	The number of features to consider when looking for the best split
		n_ estimators	The number of trees in the forest
		max_depth	The maximum depth of the tree
Adaboost	AdaBoost for short decision trees. After the first tree is created, the performance of the tree on each training instance is weighted so that the next tree created should be concerned with the attention of each training instance. Training data that is difficult to predict are given more weight, while instances that are easy to predict are given less weight. The models are created sequentially, one after the other, and each model updates the weights on the training instance that affect the learning performed by the next tree in the sequence.	base_ estimator	The base estimator from which the boosted ensemble is built.
		n_ estimators:	The maximum number of estimators at which boosting is terminated.
		learning_rate	Learning rate shrinks the contribution of each classifier by learning_rate. There is a trade-off between learning_rate and n_estimators.

MLP	A multilayer perceptron is a deep, artificial neural network. It is composed of more than one perceptron. They are composed of an input layer to receive the signal, an output layer that makes a decision or prediction about the input, and in between those two, an arbitrary number of hidden layers that are the true computational engine of the MLP.	hidden_ layer_ sizes	The ith element represents the number of neurons in the ith hidden layer.
		learning_ rate_ init	The initial learning rate used. It controls the step-size in updating the weights.

■ **Requirements**

sklearn (algorithms), pandas (reading csv files), time (record training time), numpy(visualization of feature importance in random forest)

■ **Results**

● **Algorithms Results**

Algorithms	Decision Tree	Random Forest	Adaboost	MLP
Accuracy	0.967016	0.972068	0.971534	0.971971
AUC score	0.967030	0.972054	0.971516	0.971956
F1 score	0.966947	0.972282	0.971784	0.972202
Precision	0.971725	0.967572	0.966016	0.966938
Recall	0.962217	0.977039	0.977620	0.977523
Traning time	0.62	34.84	26.50	33.84
Selected Parameters	max_depth: 8	max_features: 5 n_estimators: 200 max_depth: 9	base_estimator: Decision_Tree (max_depth=2) n_estimators: 100	hidden_layer_size: (100,) learning_ rate_init:

			learning_rate: 0.8	0.001

- **Captured Results**

➢ **Data View**

```
      gameId  creationTime  gameDuration  seasonId  winner  firstBlood
0  3326086514  1.504280e+12          1949         9       1           2
1  3229566029  1.497850e+12          1851         9       1           1
2  3327363504  1.504360e+12          1493         9       1           2
3  3326856598  1.504350e+12          1758         9       1           1
4  3330080762  1.504550e+12          2094         9       1           2

   firstTower  firstInhibitor  firstBaron  firstDragon  firstRiftHerald
0           1               1           1            1                2
1           1               1           0            1                1
2           1               1           1            2                0
3           1               1           1            1                0
4           1               1           1            1                0
```

```
   t1_towerKills  t1_inhibitorKills  t1_baronKills  t1_dragonKills  \
0             11                  1              2               3
1             10                  4              0               2
2              8                  1              1               1
3              9                  2              1               2
4              9                  2              1               3

   t1_riftHeraldKills  t2_towerKills  t2_inhibitorKills  t2_baronKills
0                   0              5                  0              0
1                   1              2                  0              0
2                   0              2                  0              0
3                   0              0                  0              0
4                   0              3                  0              0
```

```
   t2_dragonKills  t2_riftHeraldKills
0               1                   1
1               0                   0
2               1                   0
3               0                   0
4               1                   0
```

```
             gameId  creationTime  gameDuration  seasonId        winner
count  3.090400e+04  3.090400e+04  30904.000000   30904.0  30904.000000
mean   3.306337e+09  1.502933e+12   1830.401987       9.0      1.490195
std    2.935730e+07  1.971271e+09    510.642352       0.0      0.499912
min    3.214824e+09  1.496890e+12    190.000000       9.0      1.000000
25%    3.292400e+09  1.502030e+12   1530.000000       9.0      1.000000
50%    3.320250e+09  1.503850e+12   1830.000000       9.0      1.000000
75%    3.327100e+09  1.504350e+12   2145.000000       9.0      2.000000
max    3.331764e+09  1.504700e+12   4728.000000       9.0      2.000000
```

	firstBlood	firstTower	firstInhibitor	firstBaron	firstDragon
count	30904.000000	30904.000000	30904.000000	30904.000000	30904.000000
mean	1.469454	1.448874	1.305171	0.927679	1.437904
std	0.520280	0.542926	0.675593	0.840543	0.569615
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	1.000000	1.000000	0.000000	1.000000
50%	1.000000	1.000000	1.000000	1.000000	1.000000
75%	2.000000	2.000000	2.000000	2.000000	2.000000
max	2.000000	2.000000	2.000000	2.000000	2.000000

	firstRiftHerald	t1_towerKills	t1_inhibitorKills	t1_baronKills
count	30904.000000	30904.000000	30904.000000	30904.000000
mean	0.731426	5.721363	1.019544	0.372929
std	0.821019	3.800538	1.259828	0.583594
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	0.000000	0.000000
50%	0.000000	6.000000	1.000000	0.000000
75%	1.000000	9.000000	2.000000	1.000000
max	2.000000	11.000000	10.000000	5.000000

	t1_dragonKills	t1_riftHeraldKills	t2_towerKills	t2_inhibitorKills
count	30904.000000	30904.000000	30904.000000	30904.000000
mean	1.392926	0.253818	5.516179	0.980520
std	1.203411	0.435202	3.868673	1.258287
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	2.000000	0.000000
50%	1.000000	0.000000	6.000000	0.000000
75%	2.000000	1.000000	9.000000	2.000000
max	6.000000	1.000000	11.000000	10.000000

	t2_baronKills	t2_dragonKills	t2_riftHeraldKills
count	30904.000000	30904.000000	30904.000000
mean	0.414865	1.393347	0.238804
std	0.615466	1.220642	0.426360
min	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	1.000000	0.000000
75%	1.000000	2.000000	0.000000
max	4.000000	6.000000	1.000000

Some features have only two values, acting as a judgement, but some features have continuous values, which has the meaning of the number.

```
RangeIndex: 30904 entries, 0 to 30903
Data columns (total 21 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   gameId              30904 non-null  int64
 1   creationTime        30904 non-null  float64
 2   gameDuration        30904 non-null  int64
 3   seasonId            30904 non-null  int64
 4   winner              30904 non-null  int64
 5   firstBlood          30904 non-null  int64
 6   firstTower          30904 non-null  int64
 7   firstInhibitor      30904 non-null  int64
 8   firstBaron          30904 non-null  int64
 9   firstDragon         30904 non-null  int64
 10  firstRiftHerald     30904 non-null  int64
 11  t1_towerKills       30904 non-null  int64
 12  t1_inhibitorKills   30904 non-null  int64
 13  t1_baronKills       30904 non-null  int64
 14  t1_dragonKills      30904 non-null  int64
 15  t1_riftHeraldKills  30904 non-null  int64
 16  t2_towerKills       30904 non-null  int64
 17  t2_inhibitorKills   30904 non-null  int64
 18  t2_baronKills       30904 non-null  int64
 19  t2_dragonKills      30904 non-null  int64
 20  t2_riftHeraldKills  30904 non-null  int64
```

There are some columns not contributing to winner classification, which should be removed in training. After removing some useless information, the feature dimension is 16 which is not that large. The smaple size is 30904 so in MLP algorithm the default 'adam' solver can be used. The dataset has high quality.

- **Classifiers Result**

1) Decision Tree

```
DecisionTreeClassifier(max_depth=8)
Training Time:  0.6183409690856934
{'max_depth': 8}
Accuracy:     0.9670164189254833
AUC:          0.9670299803082747
F1 score:     0.9669473786691329
Precision:    0.9717248801487134
Recall:       0.9622166246851386
```

2) Random Forest

```
RandomForestClassifier(max_depth=8, max_features=5, n_estimators=200)
Training Time:  34.83711767196655
{'max_depth': 8, 'max_features': 5, 'n_estimators': 200}
Accuracy:    0.9720683959972797
AUC:         0.9720543510648467
F1 score:    0.9722824777054713
Precision:   0.967571716396431
Recall:      0.9770393334625073
```

Visualization of importance for each feature, which would help us determine our strategy.

```
 1) t2_towerKills          0.262317
 2) t1_towerKills          0.244277
 3) t2_inhibitorKills      0.163024
 4) t1_inhibitorKills      0.154480
 5) firstInhibitor         0.093768
 6) t2_dragonKills         0.019681
 7) t1_dragonKills         0.018088
 8) firstTower             0.012205
 9) t2_baronKills          0.010843
10) t1_baronKills          0.008947
11) firstBaron             0.003999
12) firstDragon            0.003447
13) firstBlood             0.002614
14) firstRiftHerald        0.000965
15) t2_riftHeraldKills     0.000824
16) t1_riftHeraldKills     0.000520
```

3) Adaboost

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=2),
                   learning_rate=0.8, n_estimators=100)
Training Time:  26.498143434524536
{'learning_rate': 0.8, 'n_estimators': 100}
Accuracy:    0.9715340522685321
AUC:         0.9715168552349348
F1 score:    0.9717835130970724
Precision:   0.9660156997893931
Recall:      0.977620616159659
```

4) MLP

```
MLPClassifier()
Training Time:  33.8365318775177
{'activation': 'relu', 'alpha': 0.0001,
Accuracy:    0.9719712425920528
AUC:         0.9719555545269297
F1 score:    0.9722021486727369
Precision:   0.9669381887877336
Recall:      0.9775237357101337
```

- **Comparison and discussion**

- **Algorithm Comparison**

1) In this project, both Adaboost and random forest are ensemble algorithms, and the subclassifier used for both of them is decision tree. Adaboost is boost for decision tree, while random forest is a bagging of decision tree.
2) First of all, decision tree can be analyzed visually and it is easy to understand and interpret. When testing data sets, the running speed is fast. In a relatively short period of time, it can be applied on large data sources and make feasible and good results though decision tree is a very simple algorithm. Its explanation is strong, also accord with the intuitive thinking of human, but the draw back is that it prone to overfitt, it has low generalization and is not a steady algorithm.
3) Random forest is not easy to be overfitting and is also simple to implement. In this project, the training speed of random forest is relatively slower than single decision tree. However, random forest has better performance than many other classifiers under many datasets and also a relatively high training speed compared with some algorithms since trees are independent in the process of training. Besides, random forest has higher generalization than decision trees, and it can shows the importance of different features, which may help us to improve our model.
4) Compared with random forests and bagging, adaboost fully considers the weights of each classifier, realizes the collection of weak classifiers with very high precision. In this project the subclassifier is decision tree but it can also chooses different subclassifiers depending on the type of data. However, the data

imbalance will lead to the decrease of the accuracy of Adaboost, while the random forest can balance the errors of the unbalanced data set. The performance of random forest will increase as estimators increasing, but for adaboost decision tree, it is going to be overfitting if the number of estimator is to large.

5) Multilayer perceptron neural network is also used in this project. The multi-layer perceptron neural network requires a large amount of data, and the training time is long. The parameters are difficult to adjust to get better formance and it is easy to fall into local extremum. However, when the data volume is large, the default parameters can often achieve good results. And compared with other classifiers in this project, the multilayer perceptron neural network has very strong learning function, in a way of updating weight.

- **Discussion**

1) Data observation and preprocessing

It is siginificant to initialy observe data to see if there is disorder, repeatability, deficiency, etc., and also the bias, variance and data symmetry, which would help us build a model with better performance consuming less time. According to the different characteristics of data, different ways of preprocessing is decided. Meanwhile for different algorithms there should be different data preprocessing methods. For instance, standardization can effectively improve the generalization of ANN and reduce the training time.

2) Alternative between single classifier and ensemble model

Training a single model to select the best parameters is not only time-consuming and laborious, but also may produce poor results after changing a set of data. In many cases, ensemble learning has higher generalization and robustness than a single classifier, and many ensemble learning algorithms can achieve satisfactory results without adjusting many parameters.

3) Parameters adjusting

Although there are many different parameters in the model, according to the characteristics of the model, different parameter types are different and their importance are also different. Therefore, the main parameters can be selected specifically for optimization to reduce time cost. For example, under the condition of the quantity of subtrees is large, random forest does not need to consider the depth of the subtree. However, the experimental results show that controlling the maximum depth in a reasonable value still can relatively improve the effect of the model, which may due to large amount of experiment data used and limitation of quantity of subtrees. Also with more parameters adjusted, the model tends to be overfitting.

4) **Multiple evaluation methods**

   A single accuracy score as an evaluation method cannot clearly see the difference between different algorithms. Different indicators determine the performance of the classifier in different aspects, so multiple indicators are calculated to determine the overall performance of the algorithm. For example, AUC score is able to show the generalization of the model, which will distinguish the classifier with other classifiers that cannot be applied to most of situations.

5) **General model for inseparabel problem**

   In most cases the problem is linear inseparable, the application of multi-layer neural network has a good effect, whose performance using default parameters can be better than a single decision tree classification. However, the training time is generally long for MLP and it is relatively difficult to adjust parameters reasonably to make better performance.

6) **Balance between performance and time cost**

   The more subtrees in the random forest, the better the performance of the model will be, but the corresponding processing will be slower, so the time cost should be considered while ensuring the performance. Besides, compared with the neural network, the adjustment of the random forest will not change much in the classification effect.

7) **Case when ensemble is much better.**

   Ensemble model is generally superior to the single, when and only when a single model is unstable. In this experiment random forest and decision tree model have similar performance indicate that the decision tree is stable, and besides, single decision tree generated by large amount of data and high data quality is relatively reliable.

- **Improvement**

1) **Parameter Adjusting**

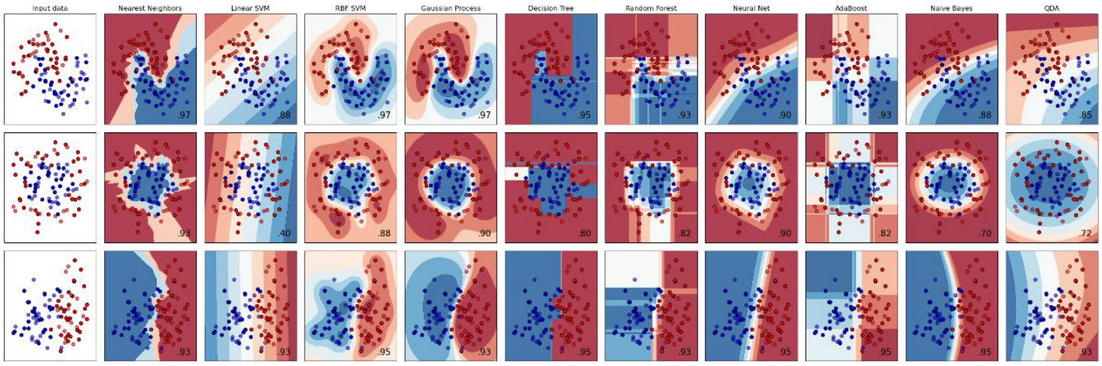
Since the data set is large, to find the optimal parameters by using the method of grid search is difficult. It should take other more advanced parameter adjusting method in the project, such as greedy parameter adjusting and bayes adjusting. By understanding the relationship between model parameters to have a deeper understanding of the model, parameters can be changed more reasonably , which will make the model on the basis of avoid overfitting to use less time to train.

2) **Feature Engineering**

Based on the original data Feature Engineering can be done, selectting features that contributes higher to predict target precision. The main implementations are: setting the threshold of variance, using pearson correlation coefficient to determine the correlation. The decision tree, random forests and adaboost used in this project are all tree models, so it can according to the phase and frequency the feature occurs to determine the importance of it. Therefore, this model can be simplified, saving storage and computing cost, making the model more easy to understand and use. Meanwhile to reduce the amount of feature dimension reduction and reduce the risk of over fitting.

3) **Classifier Boundary Diagram**

The characteristics of classifiers can be further understood through the classifier boundary distribution diagram as follows.



Input data
Nearest Neighbors
Linear SVM
RBF SVM
Gaussian Process
Decision Tree
Random Forest
Neural Net
AdaBoost
Naive Bayes
QDA
.97
.88
.97
.97
.95
.93
.90
.93
.88
.85
.93
.40
.88
.90
.80
.82
.90
.82
.70
.72
.93
.93
.95
.93
.95
.93
.95
.95
.95
.93