Project 1

Classification of unlabeled LOL match records with “Win/Loss

Hao Ying

- **Introduction**

As one of the most played eSports in the world, League of Legends (LOL) is a confrontation game with uncertain outcomes when teammates and opponents are quite different. In this project, the task is to use different classifiers to build models and train the match records data, by learning features to predict which team will be the winner. Besides, there will be a factor importance estimation to predict which factor weighs more to help gamers win. This project is realized by python and some libraries are used to realize the implementation.

- **Algorithms**

Given datasets include each game’s game Id, creation Time, game Duration, season Id, winner, first Blood, first Tower, first Inhibitor, first Baron, first Dragon, first Rift Herald, t1_towerKills, t1_inhibitorKills, t1_baronKills, t1_dragonKills, t1_riftHeraldKills, t2_towerKills, t2_inhibitorKills, t2_baronKills, t2_dragonKills, t2_riftHeraldKills. When training and testing, winner is used to be the label as training results. If "winner" is “1”, the team 1 won the match, or vice versa. Information from datasets other than winner, game Id, creation Time, season Id will be used as features to help training. Game Id, creation time and season Id are not chosen to be features for training since they seem not directly related to the winning of the game.

To solve prediction problem and evaluate test datasets, I build models for four classifiers including Decision Tree, SVM, KNN, MLP. Then, I build ensembles using above models. Finally, we can make prediction with model that has best accuracy. When building models, I set different parameters for some of classifiers to help better training according to given datasets.

a) For Decision Tree, I let {max_depth=33, min_samples_split=100}. max_depth limits the depth of tree. min_samples_split limits samples used for training. Setting the two values can help avoid overfitting.

b) For SVM, I let {kernel='rbf', C=1e3, gamma=0. 0001}. Radial basis function (RBF), is a good choice in most cases so I choose it and it performs well. C is the penalty coefficient. Large C value indicates great penalty for the error. Gamma is small so that there are more support vectors, which affects the speed of training and prediction.

c) For KNN, I let {n_neighbors=20, weights='uniform'}. I choose "uniform" for weights so that all nearest neighbor samples have the same weight.

d) For MLP, I let {alpha=0.05, solver="adam", batch_size=800, max_iter=200, beta_1=0.85, beta_2=0.7}. The default solver "adam" is chosen since it performs well in terms of training time and validation scores for large data sets. Batch_size decides the size of the minibatch used for random optimizer. Max_iter stands for the maximum number of iterations. During several training processes, I find that making max_iter small can help decrease training time. Beta stands for the exponential decay rate of the moment vector.

e) For bagging, I let {base_estimator=tree.DecisionTreeClassifier(), n_estimators=9, max_samples=20586, max_features=17} since there are 20586 samples and 17 features to be tested.

All in all, adjusting parameters is not an easy process, although sometimes it gives a good accuracy, you actually do not know why. This needs further research if we want to have a deeper understanding.

Besides, datasets are pre-processed so that training can work better. According to given datasets, the ranges of the feature game duration time are not the same as other features, which may cause a problem that one small change in game duration time might affect a lot. To address this problem, I divide game duration time by 1000. After comparing the accuracy and training time of classifiers before the data preprocessing and after it, I find they almost all perform better, especially KNN and MLP.

To estimate the importance of different factors, I use two ways. First, I train every model again without one factor dividedly using KNN classifier and decide its importance according to the changes of model's accuracy. The one giving lowest accuracy means the corresponding removed factor has relatively biggest effect on the chance of winning. On the contrary, it has relatively lowest effect on the chance of winning. The second way is to use feature_importances_, which is the parameters of DT, to estimate factor importance. At first, I try the first way and find that almost every training work well, and accuracies have few differences, so I try the second way.

■ Requirements

To realize the implementation, I use some prerequisite packages and modules: numpy for data processing, pandas for datasets reading, time for time recording, texttable for table building,

matplotlib for picture drawing, metrics for accuracy counting, tree for Decision Tree classifier, SVC for SVM classifier, KNeighborsClassifier for KNN, MLPClassifier for MLP classifier, BaggingClassifier for bagging.

■ Results

This is the result for classification.

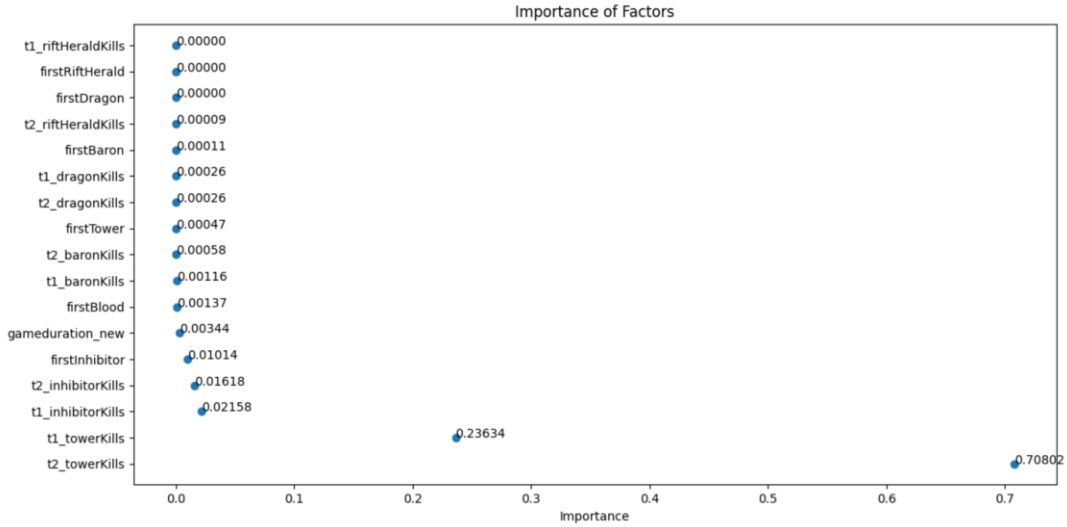
classifier	parameters	accuracy	training time
DT	{'max_depth': 33, 'min_samples_split': 100}	0.967	0.195
SVM	{'kernel': 'rbf', 'C': 'les3', 'gamma': 0.0001}	0.962	12.951
K-NN	{'n_neighbors': 20, 'weights': 'uniform'}	0.966	11.109
MLP	{'alpha': 0.05, 'solver': 'adam', 'batch_size': 800, 'max_iter': 200, 'beta_1': 0.85, 'beta_2': 0.7}	0.970	36.526
bagging	{'base_estimator': 'tree.DecisionTreeClassifier()', 'n_estimators': 9, 'max_samples': 20586, 'max_features': 17}	0.968	1.011

This is the result for factor importance estimation by way1.

conditons	accuracy	training time
all factors	0.959	2.458
no factor 1 (game Duration):	0.960	2.206
no factor 2 (firstBlood):	0.959	2.084
no factor 3 (firstTower):	0.958	2.128
no factor 4 (firstInhibitor):	0.955	2.287
no factor 5 (firstBaron):	0.957	2.210
no factor 6 (firstDragon):	0.960	2.139
no factor 7 (firstRiftHerald):	0.959	2.226
no factor 8 (t1_towerKills):	0.952	2.508
no factor 9 (t1_inhibitorKills):	0.956	2.108
no factor 10 (t1_baronKills):	0.958	2.189
no factor 11 (t1_dragonKills):	0.960	1.981
no factor 12 (t1_riftHeraldKills):	0.960	2.227
no factor 13 (t2_towerKills):	0.948	2.480
no factor 14 (t2_inhibitorKills):	0.958	2.138
no factor 15 (t2_baronKills):	0.959	2.358
no factor 16 (t2_dragonKills):	0.960	2.026
no factor 17 (t2_riftHeraldKills):	0.959	2.271

factor 10 has relatively little effect on the result
factor 12 has relatively big effect on the result

This is the result for factor importance estimation by way2.



■ Comparison and discussion

For prediction problem, according to results above, different classifiers all give relatively good accuracy under given parameters. Actually, they have their own classification ways theatrically.

a) For Decision Tree, the advantage is that it can visualize classification ways by the tree with leaf nodes. However, the disadvantage is that if the depth of the tree is not limited, overfitting may occur since the depth of the tree will be so large that it remembers all labels of the training data. Therefore, pruning operations are required. For instance, we can set max_depth or min_samples_split. During my DT training process, the training accuracy is higher after I set and adjust max_depth value as well as min_samples_split.

b) For SVM, it has two important parameters, C and gamma. Small gamma may lead to low accuracy, while high gamma may lead to overfitting which causes low test accuracy. SVM performs well for many datasets, but it needs careful data pre-processing and parameters adjusting. During my SVM training process, it does not perform well at first. After I make gamma value small, it performs better.

c) For KNN, the training is sensitive to training datasets, especially the value of n_neighbor. When the neighbor value is too small, it uses a small neighborhood for prediction. If the neighbor happens to be a noisy point, it will lead to overfitting. As the neighbor value continues increasing, the training accuracy of the model first increases and then decreases. The turning point is the neighbor value we are looking for.

d) For MLP, its advantage is that it can access information from large datasets and construct complicated models. However, it needs careful data preprocessing and parameters adjusting. Besides, it usually requires more training time. To adjusting parameters, the first thing is to create a network which is large enough to overfit to ensure that the network can learn the task. Then, shrink the network or increase the alpha to enhance the regularization, which can improve the generalization performance. After this, for those who chooses adam to be solver, set the batch size, the number of iterations and the beta value. In my MLP training process, it takes much more time to adjust parameters compared to other classifiers.

e) For bagging, it averages the results of every model's accuracy. Compared with a single decision tree, it decreases the variance and has a smoother classification boundary. In my

bagging training process, I find that increasing n_estimator value can improve training accuracy.

Generally speaking, different classifiers have different performance in different situations although for this dataset they all perform good. Next time when we need to decide which classifier to use, we should consider according to data size, data characteristics, number of features, number of error data, dimensional differences between data, and so on.

For factor importance prediction problem, according to results above, way1 and way2 show different results and they are not very reliable, actually.

a) For the first way, the accuracy is high for almost every condition, and it is difficult to judge which factor weighs more by this result. Besides, during the training process, I find that when value of neighbor is different, the result of factor importance could be quite different.

b) For the second way, t1_towerKills and t2_towerKills show high importance. However, we may have a question that why t2_towerKills shows higher importance than t1_ towerKills when they are both about towerKills. In a word, this needs further exploration to find better ways to estimate factor importance.

Summary

In this project, I learn how to build models based on different classifiers using given datasets with factors and results, make prediction and evaluate factor importance. During this process, I learn advantages and disadvantages of different classifiers, the meaning of their parameters, and how to adjust them for better training. These are quite meaningful since they can not only be solutions to this problem, but other problems like estimating the probability that people support one proposal and predicting the chance of admitted.

There are some extensions to this project we can explore further. One extension is to visualize classification, show how to increase the rate of winning in a more specific way, and give advice that to what degree we should do to improve that part. Another extension is to use tools to find parameters of classifiers more quickly. Besides, since two ways to weigh its importance is not very reliable, we need further explorations to estimate factor importance.