Classification of League of Legends Results

I. Introduction:

League of legends is an online, 5 vs. 5 competitive PC game. It is one of the most popular games currently around. In this game, each team tries to destroy the other team's Nexus, which sits deep inside the enemy base surrounded by turrets, which can injure players quickly if they attempt to bypass them too early in the game. In this project, we are given the information of three million games, which include the game duration, first blood, first tower, first inhibitor and etc. Then we try to find some common characters in the winner team through basic classification skills. After we train a classification, we can use the classification to predict the game's winner with the given data.

II. Algorithms

The code of the algorithm can be seen in the appendix.

1. DT

The first algorithm is one using decision tree classifier. By observing the data, it is obvious that the game ID and creation time are unrelated to the winner. Therefore, we first use drop function to delete the two columns to ensure the data we use are meaningful (figure 1).

```
X=data.drop(["gameId" , "creationTime" , "winner"], axis = 1 ).values
y=data['winner'].values
```

Figure 1. extract the meaning attributes and label

After preprocessing the data, then we can create a tree classifier, and train the classifier use the given attribute and label. After being fitted, the model can then be used to predict the class of samples. There's something suppose to be further explained that the data used for test should do the same preprocessing part before being used for predicting the

winner.

2. ANN

Preprocessing the data is quite important for ANN. The ANN classifier is quite sensitive to the data scale. If one of the data is quite large, the data will influence the performance of the classifier. To eliminate the influence of the scale, the first step is to set the range of all attributes to be in the range 0 to 1. The way I normalize the data is through the function MinMaxScaler. We firstly choose the meaningful data as we did in DT classifier, then we normalize the data by setting the feature_range = (0,1) (figure2). Then we transfer the data type to tensor float type that ANN can use for training. After preprocessing, we can create a ANN classifier and set the hidden layer and learning

```
scaler = MinMaxScaler(feature_range=(0, 1))
X = scaler.fit transform(X)
```

Figure 2.set the range of the attribute

rate for the classifier (figure3).

```
class ANN(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(in_features=18, out_features=72)
        self.output = nn.Linear(in_features=72, out_features=3)
        nn.Softmax(dim=1)
    def forward(self, x):
        x = torch.sigmoid(self.fc1(x))
        x = self.output(x)
        x = F.softmax(x)
        return x
```

Figure 3. create a ANN classifier and set the hidden layer

After training, the classifier will predict the class label. It worth mentioning that the algorithm for testing is different from the DT (figure 4).

```
predict_out = model(X_test)
_, predict_y = torch.max(predict_out, 1)
```

Figure 4. the part of ANN used for prediction is different from DT

3. bagging

The algorithm for bagging classifier is similar to the process we do for DT, we drop

the meaningless data from the attributes first. After that, we create a bagging classifier with basic classifier as decision tree (figure 5).

```
kfold = model_selection.KFold(n_splits=10)
cart = DecisionTreeClassifier()
num_trees = 10
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees)
```

Figure 5. create a bagging classifier with basic classifier as decision tree

The tree number we set for the bagging is ten, we can then train the classifier using the same way as we did in DT by using fit function and predict the class (figure 6).

```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Figure 6. bagging use fit function to train and predict function to predict the class

4. boosting

Boosting classifier is one that is similar to bagging classifier. With only difference in detail, we can just change the a small part of the bagging classifier to boosting classifier. Here is the part that we changed (figure 7).

```
kfold = model_selection.KFold(n_splits=10)
model = AdaBoostClassifier(n_estimators=num_trees)
```

Figure 7. we change the bagging classification to boosting classification to see the performance

III. Experimental Requirements

In this experiment, Jupyter Notebook is used to compile the code with python 3.7 edition. To run the code above, sklearn and pytorch packages should be installed first. It should be noted that the training process and test process are in the same .py file as shown above. The meaning of each step are noted in the annotation part with red color.

IV. Experimental Results

1. DT

```
the basic information of classifier:
 DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                        max_depth=3, max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort='deprecated',
                        random_state=None, splitter='best')
Accuracy: 0.9514232973865734
```

2. ANN

```
the basic information of classifier:
 ANN(
  (fc1): Linear(in_features=18, out_features=72, bias=True)
  (output): Linear(in_features=72, out_features=3, bias=True)
)
Accuracy: 0.9687651802195667
```

3. bagging

```
 the basic information of classifier:
  BaggingClassifier(base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                          class_weight=None,
                                                          criterion='gini',
                                                          max_depth=None,
                                                          max_features=None,
                                                          max_leaf_nodes=None,
                                                          min_impurity_decrease=0.0,
                                                          min_impurity_split=None,
                                                          min_samples_leaf=1,
                                                          min_samples_split=2,
                                                          min_weight_fraction_leaf=0.0,
                                                          presort='deprecated',
                                                          random_state=None,
                                                          splitter='best'),
                    bootstrap=True, bootstrap_features=False, max_features=1.0,
                    max_samples=1.0, n_estimators=10, n_jobs=None,
                    oob_score=False, random_state=None, verbose=0,
                    warm_start=False)
 Accuracy: 0.966870688817643
```

4. boosting

```
the basic informaton of classifier AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                   n_estimators=10, random_state=True)
Accuracy: 0.9544836296512192
```

	DT	ANN	bagging	boosting
accuracy	0.9514	0.9688	0.9669	0.9545
training time	2s	43s	7s	8s

In this experiment, three different classifiers have been applied to predict the winner of each game. The first classifier is decision tree classifier, which can reach the accuracy of 0.951. The second classifier is ANN, the accuracy of ANN classifier is 0.968. The third classifier is bagging classifier with the basic classifier as decision tree, the accuracy of this classifier is 0.968. the accuracy of the boosting classifier is 0.9545,which is higher than DT and lower than others.

V. Comparison and Discussion

It's obvious that both the bagging classifier and ANN classifier can reach a high

accuracy. In fact, the data used to train ANN classifier is only 2000. If using more data to train the ANN classifier, the accuracy can reach even higher. However, compared to bagging classification, the training time of ANN is much longer even though the training data is only 2000 for ANN (the data used to train bagging classifier is more than 30000). Based on this view, bagging classifier has an edge over the ANN.

In a similar way, when we compare the bagging classifier with Decision tree, we can compare the accuracy and time of the two classifiers. In this experiment, we use the bagging classifier as a combination of 10 different decision classifiers. It seems that the accuracies are both high. However, if we measure the error rate, we can see the probability that DT makes an error is two times of the probability that bagging classifier makes an error. Moreover, the training time are small for the two classifiers. The bagging classifier and boosting classifier are both ensemble classifiers. it is obvious that the accuracy of both of the classifiers are higher than decision tree, which shows that the advantage of using ensemble methods. However, in this experiment, the accuracy of bagging classifier is higher than boosting classifier. We know that in common cases, boosting classifiers should be as good as bagging classifiers or even better than classifiers, the reason that the lower accuracy of boosting maybe because of overfitting. As we know, bagging can handle overfitting quite well, but boosting classifier cannot.

VI. Summery and Future work

In conclusion, in this project, three different classifiers have been applied to predict the winner of an lol game. Compared to DT, ANN and boosting classifier, the bagging classifier can reach a high accuracy and time-saving at the same time. However, to further improve the accuracy is hard for the bagging classifier as the accuracy for DT is limited. Therefore, if we need to find a better classifier to predict the winner team, we can try to use bagging classifier with the basic classifier being ANN, as ANN can reach higher accuracy than DT. With the consideration of time-consuming, we need to optimize the algorithm to shorten the running time. In the future work, we will

explore ways that can minimize the training time. Furthermore, if given more powerful computing resources and time, I would like to experiment with a larger dataset and try other ways in the purpose that the classification can reach higher accuracy in shorter training time.

Appendix

Codes:

1. Decision tree

```
# coding: utf-8
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# read the training data
data = pd.read_csv('new_data.csv')

X=data.drop(["gameId" , "creationTime" , "winner"], axis = 1 ).values
y=data['winner'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

#create a decision tree classifier
clf = DecisionTreeClassifier()
clf=clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)# test error using the data exacting from training data
print("Accuracy:",accuracy_score(y_test, y_pred))


# read the test data and test the classification accuracy
test=pd.read_csv('test_set.csv')
test_x=test.drop(["gameId" , "creationTime" , "winner"], axis = 1 ).values
test_y=test['winner'].values
pred_y=clf.predict(test_x)
print("the basic information of classifier", clf)
```

```
print("Accuracy:",accuracy_score(test_y,pred_y ))
```

2. ANN

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer

# read the training data
data = pd.read_csv('E:/new_data.csv')

#create a Simple Imputer
SimpleImputer = SimpleImputer()

# extract the attributes and change the range of each attribute to [0,1]
X = data.drop(['gameId','creationTime','winner'], axis=1).values
X = SimpleImputer.fit_transform(X)
scaler = MinMaxScaler(feature_range=(0, 1))
X = scaler.fit_transform(X)

# extract the label
y = data['winner'].values
# change the attibutes and labels to float type
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)
X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.LongTensor(y_train)
y_test = torch.LongTensor(y_test)
# define an ANN classifier
class ANN(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(in_features=18, out_features=72)
        self.output = nn.Linear(in_features=72, out_features=3)
        nn.Softmax(dim=1)
    def forward(self, x):
        x = torch.sigmoid(self.fc1(x))
        x = self.output(x)
```

```
            x = F.softmax(x)
            return x

model = ANN()
# set the loss and learning rate
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.05)

# training procedure, the data used for learning is 2000
epochs = 2000
loss_arr = []
for i in range(epochs):
    y_hat = model.forward(X_train)
    loss = criterion(y_hat, y_train)
    loss_arr.append(loss)
    if i % 10 == 0:
        print(f'Epoch: {i} Loss: {loss}')
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# predict the label of each x_test
predict_out = model(X_test)
_,predict_y = torch.max(predict_out, 1)
# calculate the accuracy rate using data from training data
from sklearn.metrics import accuracy_score
print("The accuracy is ", accuracy_score(y_test, predict_y) )


# read and process the data used for test
test=pd.read_csv('test_set.csv')
test_x = test.drop(["gameId" , "creationTime" , "winner"], axis = 1 ).values
test_x = SimpleImputer.fit_transform(test_x)
scaler = MinMaxScaler(feature_range=(0, 1))
test_x = scaler.fit_transform(test_x)
test_x = torch.FloatTensor(test_x)

# use the ANN classifier to predict label
pred_out = model(test_x)
_,pred_y = torch.max(pred_out, 1)

test_y = test['winner'].values
test_y = torch.FloatTensor(test_y)
# the final accuracy rate is 0.967
```

```
print(“the basic information of classifier”, model)
print("Accuracy:",accuracy_score(test_y,pred_y ))
```

3. bagging

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

data = pd.read_csv('new_data.csv') # read the training data

X=data.drop(["gameId" , "creationTime" , "winner"], axis = 1 ).values  # drop the meaningless data and the target label
y=data['winner'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) # take twenty percent of all the data for test

kfold = model_selection.KFold(n_splits=10) # create a bagging classification with decision tree classificaton
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees)

results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold)
print(results.mean()) # the training accuracy

model.fit(X_train,y_train)
y_pred = model.predict(X_test) # test error using the data exacting from training data
print("Accuracy:",accuracy_score(y_test, y_pred))


# read the test data and test the classification accuracy
test=pd.read_csv('test_set.csv')
test_x=test.drop(["gameId" , "creationTime" , "winner"], axis = 1 ).values
test_y=test['winner'].values
pred_y=model.predict(test_x)
print(“the basic information of classifier”, model)
print("Accuracy:",accuracy_score(test_y,pred_y))
#the classification is good, and can reach the accuracy of 0.968
```

4.boosting

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

data = pd.read_csv('new_data.csv') # read the training data

X=data.drop(["gameId" , "creationTime" , "winner"], axis = 1 ).values# drop the meaningless data and the target label
y=data['winner'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

num_trees = 10
kfold = model_selection.KFold(n_splits=10)
model = AdaBoostClassifier(n_estimators=num_trees) # create a boosting classification with decision tree classificaton
results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold)

model.fit(X_train,y_train)
y_pred = model.predict(X_test)
print("Accuracy:",accuracy_score(y_test, y_pred))


# read the test data and test the classification accuracy
test=pd.read_csv('test_set.csv')
test_x=test.drop(["gameId" , "creationTime" , "winner"], axis = 1 ).values
test_y=test['winner'].values
pred_y=model.predict(test_x)
print("the basic informaton of classifier",model)
print("Accuracy:",accuracy_score(test_y,pred_y ))
```