REPORT OF PROJECT ONE

Introduction:

League of Legends (LoL) is one of the most played eSports in the world at the moment. Arguably, the fun aspect of games or sport lies within the uncertainty of their outcomes. There is nothing more boring than playing or watching a game, be it soccer, basketball, or any video games, with a predictable ending. This is why La Liga, despite the fact that it used to have two of the best soccer players of the modern time-Lionel Messi and Cristiano Ronaldo-has been widely considered to be less entertaining than her counterpart Premier League in England. The reason is because Barcelona and Real Madrid are usually classes above the rest of the league. To make La Liga more fun to watch, there must be a way to close the skill gaps among teams. Similarly, the creators of LoL have tried their best to match players with teammates and opponents of as similar skill level as possible. In this project, I will have access to about 3 Million match records of solo gamers as training set. 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.

In sum, the fields include:

- Game ID
- Creation Time (in Epoch format)
- Game Duration (in seconds)

- Season ID
- Winner (1 = team1, 2 = team2)
- First Baron, dragon, tower, blood, inhibitor and Rift Herald (1 = team1, 2 =team2, 0 = none)
- The number of towers, inhibitor, Baron, dragon and Rift Herald kills each team has.

gameId	creationTi	gameDura	seasonId	winner	firstBlood	firstTower	firstInhibit	firstBaron
3.22E+09	1.50E+12	203	9	2	0	0	0	0
3.32E+09	1.50E+12	1799	9	1	2	2	1	1
3.3E+09	1.50E+12	1876	9	2	1	2	2	0
3.32E+09	1.50E+12	1423	9	1	1	2	1	1
3.33E+09	1.50E+12	2093	9	1	1	2	1	0
3.32E+09	1.50E+12	1649	9	1	1	1	1	0
3.32E+09	1.50E+12	1365	9	2	2	2	2	2
3.33E+09	1.50E+12	1645	9	1	1	1	0	0
3.32E+09	1.50E+12	2479	9	2	1	1	2	2
3.32E+09	1.50E+12	2088	9	2	1	2	2	2
3.31E+09	1.50E+12	1508	9	1	2	2	0	2
3.25E+09	1.50E+12	1683	9	1	1	1	1	1
3.3E+09	1.50E+12	1913	9	2	1	1	2	0
3.25E+09	1.50E+12	2030	9	2	1	1	2	2
3.29E+09	1.50E+12	192	9	2	0	0	0	0
3.33E+09	1.50E+12	1737	9	1	2	1	1	0
3.32E+09	1.50E+12	1726	9	1	1	2	1	1
3.22E+09	1.50E+12	1659	9	2	1	2	2	2
3.28E+09	1.50E+12	2512	9	2	2	1	2	1
3.33E+09	1.50E+12	1348	9	1	2	1	1	0
3.28E+09	1.50E+12	1233	9	2	2	2	2	0
3.33E+09	1.50E+12	2621	9	2	2	2	2	1
3.33E+09	1.50E+12	2066	9	2	1	2	2	2
3.28E+09	1.50E+12	1578	9	1	1	1	1	1
3.32E+09	1.50E+12	1807	9	2	2	1	2	2
3.32E+09	1.50E+12	1504	9	1	1	1	1	1
3.32E+09	1.50E+12	1489	9	2	2	2	2	0
3.32E+09	1.50E+12	2623	9	2	2	2	2	1
3.32E+09	1.50E+12	1597	9	1	2	1	1	1
3.29E+09	1.50E+12	1949	9	1	1	1	1	1
3.32E+09	1.50E+12	2797	9	2	2	1	1	2
3.3E+09	1.50E+12	2344	9	1	2	2	1	2
3.24E+09	1.50E+12	1685	9	2	2	2	2	0
3.33E+09	1.50E+12	1565	9	1	1	1	1	0

firstDrago	firstRiftHe	t1_towerK	t1_inhibitc	t1_baronK	t1_dragon	t1_riftHera	t2_towerK	t2_inhibitc	t2_baronK	t2_dragon	t2_riftHeral
0	0	0	0	0	0	0	0	0	0	0	0
2	0	8	1	1	0	0	4	0	0	2	0
2	0	3	0	0	0	0	7	1	0	3	0
2	1	8	2	1	1	1	3	0	0	2	0
1	0	10	2	0	3	0	4	0	0	0	0
1	0	8	1	0	3	0	3	0	0	0	0
2	2	0	0	0	0	0	11	3	1	2	1
2	0	6	0	0	1	0	0	0	0	1	0
1	0	4	0	0	1	0	10	2	2	4	0
1	2	1	0	0	1	0	9	3	1	3	1
2	1	6	0	0	0	1	3	0	1	2	0
1	0	11	3	1	3	0	0	0	0	0	0
2	0	4	0	0	0	0	9	2	0	3	0
2	1	3	0	0	0	1	10	2	2	4	0
0	0	0	0	0	0	0	0	0	0	0	0
1	0	11	3	0	2	0	1	0	0	0	0
1	1	11	1	1	2	1	2	0	0	1	0
2	2	2	0	0	1	0	10	1	1	2	1
2	0	6	0	1	1	0	8	1	0	4	0
1	0	5	1	0	2	0	4	0	0	0	0
0	0	0	0	0	0	0	8	1	0	0	0
2	2	6	1	1	2	0	11	4	1	3	1
2	0	2	0	0	0	0	9	1	1	4	0
1	1	11	3	1	3	1	0	0	0	0	0
2	0	1	0	0	0	0	10	2	1	2	0
1	0	6	1	1	2	0	3	0	0	0	0
2	2	2	0	0	0	0	8	1	0	3	1
2	2	3	0	1	1	0	10	2	1	5	1
1	1	9	1	1	3	1	0	0	0	0	0
2	0	10	2	1	2	0	3	0	0	1	0
1	1	7	2	1	2	1	8	1	3	3	0
2	0	9	1	0	3	0	4	0	2	2	0
2	0	1	0	0	0	0	7	2	0	3	0
1	1	11	3	0	1	1	0	0	0	1	0

My task is to create one or more classifiers that take as inputs from any fields from above (except “winner”) such match record, and labels this record as a “1” or a “2”.

In this program, I have used two methods which are Decision Tree and Artificial Neural Network. Decision Tree (hereafter this text will be abbreviated as DT) is a decision analysis method which is based on the known probability of occurrence of various situations and through the formation of a decision tree to obtain the probability that the expected value of net

present value is greater than or equal to zero, evaluate the project risk, and judge its feasibility. It is a graphic method that intuitively uses probability analysis. Because this decision branch is drawn into a graph, it is very similar to the branches of a tree, so it is called decision tree. Artificial Neural Network (hereafter this text will be abbreviated as ANN) is an operational model, which is composed of a large number of nodes (or neurons) connected with each other. Each node represents a specific output function, which is called activation function. Each connection between two nodes represents a weighted value of the signal passing through the connection, which is called the weight, which is equivalent to the memory of the artificial neural network. The output of the network varies with the connection mode, weight value and excitation function of the network. The network itself is usually the approximation of some algorithm or function in nature, and may also be the expression of a logic strategy.

■ Algorithms:

In my algorithms, for there are a few features, I have used 17 of them which are ‘seasonId’, 'firstBlood', 'firstTower', 'firstInhibitor', 'firstBaron', 'firstDragon', 'firstRiftHerald', 't1_towerKills', 't1_inhibitorKills', 't1_baronKills', 't1_dragonKills', 't1_riftHeraldKills', 't2_towerKills', 't2_inhibitorKills', 't2_baronKills', 't2_dragonKills' and 't2_riftHeraldKills' and the label is ‘winner’

which is ‘1’ for team 1 wins or ‘2’ for team 2 wins.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,random_state=1
# 80% training and 20% test

clf = DecisionTreeClassifier(criterion="entropy", max_depth=10)# Create Decision Tree
clf = clf.fit(x_train,y_train)# Train Decision Tree Classifer
y_pred = clf.predict(x_test)#Predict the response for test dataset
#print(y_pred)
a = accuracy_score(y_test, y_pred)
```

DT is a decision analysis method which is based on the known probability of occurrence of various situations and through the formation of a decision tree to obtain the probability that the expected value of net present value is greater than or equal to zero, evaluate the project risk, and judge its feasibility. It is a graphic method. Because this decision branch is drawn into a graph, it is very similar to the branches of a tree, so it is called decision tree.

```
class ANN(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(in_features=17, out_features=30)
        self.output = nn.Linear(in_features=30, out_features=3)
    def forward(self, m):
        m = torch.sigmoid(self.fc1(m))
        m = self.output(m)
        m = F.softmax(m)
        return m

model = ANN()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

epochs = 100
loss_arr = []
for i in range(epochs):
    n_hat = model.forward(m_train)
    loss = criterion(n_hat, n_train)
    loss_arr.append(loss)
    if i%2 == 0:
        print(f'Epoch:{i} Loss:{loss}')
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

ANN is an operational model, which is composed of a large number of nodes (or neurons) connected with each other. Each node represents a specific output function, which is called activation function. Each connection between two nodes represents a weighted value of the signal passing through the connection, which is called the weight, which is equivalent to the memory of the artificial neural network. The output of the network varies with the connection mode, weight value and excitation function of the network. The network itself is usually the approximation of some algorithm or function in nature, and may also be the expression of a logic strategy. When I get two outputs of accuracy of DT and ANN, there will be a procedure that judges

which is more large and then the result will be more accurate one.

■ Requirements.

The prerequisite packages I use for my code are ‘pandas’, ‘torch’, ‘sklearn’, ‘datatime’ and ‘matplotlib’. Torch is used for ANN. Sklearn is used for DT’s classification, test and calculating the accuracy. Matplotlib is used for plot DT’s picture. Datatime is used for recording programing time.

■ Results.

DT		ANN		
Duration	Accuracy	Duration	Accuracy	Frequency
0.212	0.9634	0.743	0.9628	1
0.218	0.9635	0.717	0.9611	2
0.213	0.9631	0.755	0.9616	3
0.216	0.9635	0.721	0.9618	4
0.208	0.9635	0.728	0.9616	5

```
<built-in method values of Tenso
Epoch:0 Loss:1.0404645204544067
Epoch:2 Loss:0.9484430551528931
Epoch:4 Loss:0.8789101243019104
Epoch:6 Loss:0.8228802680969238
Epoch:8 Loss:0.776783287525177
Epoch:10 Loss:0.7366601228713989
Epoch:12 Loss:0.7026914358139038
Epoch:14 Loss:0.675834059715271
Epoch:16 Loss:0.655470609664917
Epoch:18 Loss:0.6404291987419128
Epoch:20 Loss:0.6295273900032043
Epoch:22 Loss:0.621616005897522
Epoch:24 Loss:0.6157917380332947
Epoch:26 Loss:0.611490190029144
Implicit dimension choice for so
as an argument.
  m = F.softmax(m)
```

```
Epoch:28 Loss:0.6082834005355835
Epoch:30 Loss:0.6058500409126282
Epoch:32 Loss:0.603943407535553
Epoch:34 Loss:0.6024153828620911
Epoch:36 Loss:0.6011592149734497
Epoch:38 Loss:0.6001250743865967
Epoch:40 Loss:0.5992494821548462
Epoch:42 Loss:0.5984821915626526
Epoch:44 Loss:0.597811758518219
Epoch:46 Loss:0.5972111225128174
Epoch:48 Loss:0.5966894030570984
Epoch:50 Loss:0.5962257385253906
Epoch:52 Loss:0.5958145260810852
Epoch:54 Loss:0.5954291224479675
Epoch:56 Loss:0.5950750708580017
Epoch:58 Loss:0.5947526097297668
Epoch:60 Loss:0.5944589376449585
Epoch:62 Loss:0.5941850543022156
Epoch:64 Loss:0.5939295291900635
Epoch:66 Loss:0.5936901569366455
Epoch:68 Loss:0.5934667587280273
Epoch:70 Loss:0.5932565331459045
Epoch:72 Loss:0.593056321144104
Epoch:74 Loss:0.5928662419319153
Epoch:76 Loss:0.5926889777183533
Epoch:78 Loss:0.5925168991088867
Epoch:80 Loss:0.5923580527305603
Epoch:82 Loss:0.5922045111656189
Epoch:84 Loss:0.5920592546463013
Epoch:86 Loss:0.5919189453125
Epoch:88 Loss:0.5917807221412659
Epoch:90 Loss:0.5916460156440735
Epoch:92 Loss:0.5915207266807556
Epoch:94 Loss:0.5913988947868347
Epoch:96 Loss:0.5912802815437317
Epoch:98 Loss:0.5911683440208435
ANN is more accurate whose accuracy is : 0.9635745507527926
```

■ Comparison and discussion:

There are many things needed to be improve. The classifiers I used are just a few of all and add more classifiers can increase the accuracy of result. When I put more classifiers into my program, I can select the best one by voting. In addition, my planning is not good. The unreasonable time allocation has made a bad influence on my project.