```
--2021-02-18 03:14:03--  https://jsc370.github.io/events.csv.zip
Resolving jsc370.github.io (jsc370.github.io)... 185.199.108.153, 18
5.199.111.153, 185.199.109.153, ...
Connecting to jsc370.github.io (jsc370.github.io)|185.199.108.153|:4
43... connected.
HTTP request sent, awaiting response... 200 OK
Length: 21797089 (21M) [application/zip]
Saving to: ‘events.csv.zip.1’

events.csv.zip.1    100%[===================>]  20.79M  45.2MB/s
in 0.5s

2021-02-18 03:14:04 (45.2 MB/s) - ‘events.csv.zip.1’ saved [21797089
/21797089]

Archive:  events.csv.zip
replace events.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: N
--2021-02-18 03:14:09--  https://jsc370.github.io/ginf.csv.zip
Resolving jsc370.github.io (jsc370.github.io)... 185.199.108.153, 18
5.199.111.153, 185.199.109.153, ...
Connecting to jsc370.github.io (jsc370.github.io)|185.199.108.153|:4
43... connected.
HTTP request sent, awaiting response... 200 OK
Length: 344339 (336K) [application/zip]
Saving to: ‘ginf.csv.zip.1’

ginf.csv.zip.1      100%[===================>] 336.27K  --.-KB/s
in 0.04s

2021-02-18 03:14:09 (8.24 MB/s) - ‘ginf.csv.zip.1’ saved [344339/344
339]

Archive:  ginf.csv.zip
replace ginf.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: N
--2021-02-18 03:14:13--  https://jsc370.github.io/dictionary.txt
Resolving jsc370.github.io (jsc370.github.io)... 185.199.108.153, 18
5.199.111.153, 185.199.109.153, ...
Connecting to jsc370.github.io (jsc370.github.io)|185.199.108.153|:4
43... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1306 (1.3K) [text/plain]
Saving to: ‘dictionary.txt.1’

dictionary.txt.1    100%[===================>]   1.28K  --.-KB/s
in 0s

2021-02-18 03:14:13 (89.4 MB/s) - ‘dictionary.txt.1’ saved [1306/130
6]
```

Soccer!

Haiyue Yang, Yun Shen, Xin Peng

The success of any football match lies in the coaches and managers of soccer teams, since they make decisions for selecting players for matches, planning the strategy and instructing players on the pitch and they are responsible for holding the team together. However, the process of selecting players, deciding the best strategy and supporting players based on their own attributes is always very difficult in real life, and it is hard for humans to make objective decisions based on so many attributes. So in this report, we are going to design a decision-support tool that can aid coaches and managers' decision making process and lead to a successful outcome.

We are using the 'Football Events' dataset from Kaggle (https://www.kaggle.com/secareanualin/football-events (https://www.kaggle.com/secareanualin/football-events)), which contains 9074 games, 941009 events from soccer leagues: England, Spain, Germany, Italy, France from 2011/2012 season to 2016/2017 season.

And we have explored the data, brainstormed what factors could help improve the decision-support tool, proposed several models, and implemented three variants below.

Add-on Value of the Data

- Design strategies (location-wise) for players: if we analyze the hottest spots in the game, we may develop a strategy to guide players where to stand during the game
- Help coaches to better understand what’s important in training
- Focus on certain assist methods: some methods are more efficient
- Focus on training certain body parts: which muscle is crucial during practice

Key Factors to Measure Success

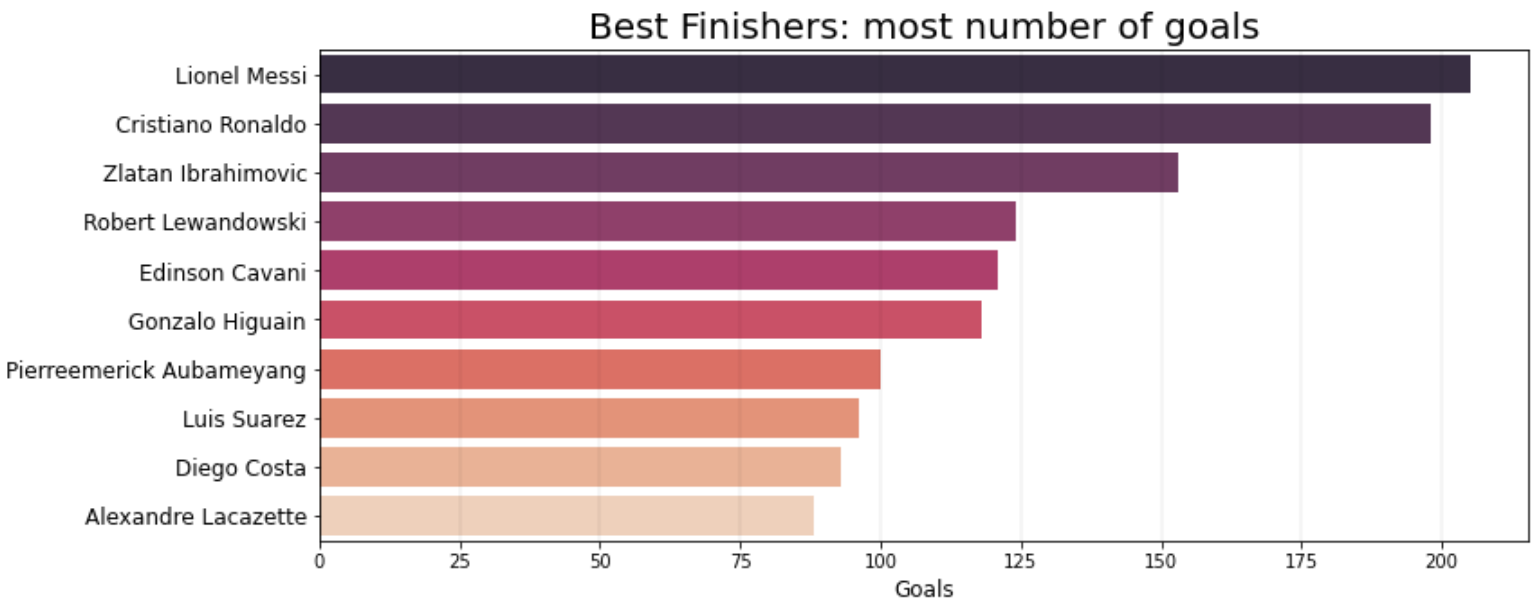
- Winning of the game, which is the ultimate goal
- Saving from unnecessary training: save time and money from things that are less important to the game
- Less injury: players are beneficial from special training to avoid injury
- Model: best accuracy and least running complexity, since our dataset is very imbalanced, so our model is very cost-sensitive and we must take that into account.
- We might also look at F1-score to determine how good our model is, since when the data is so imbalanced that accuracy might not be the only criteria.

Data Exploration

Here, I want to discover the performance of each player.

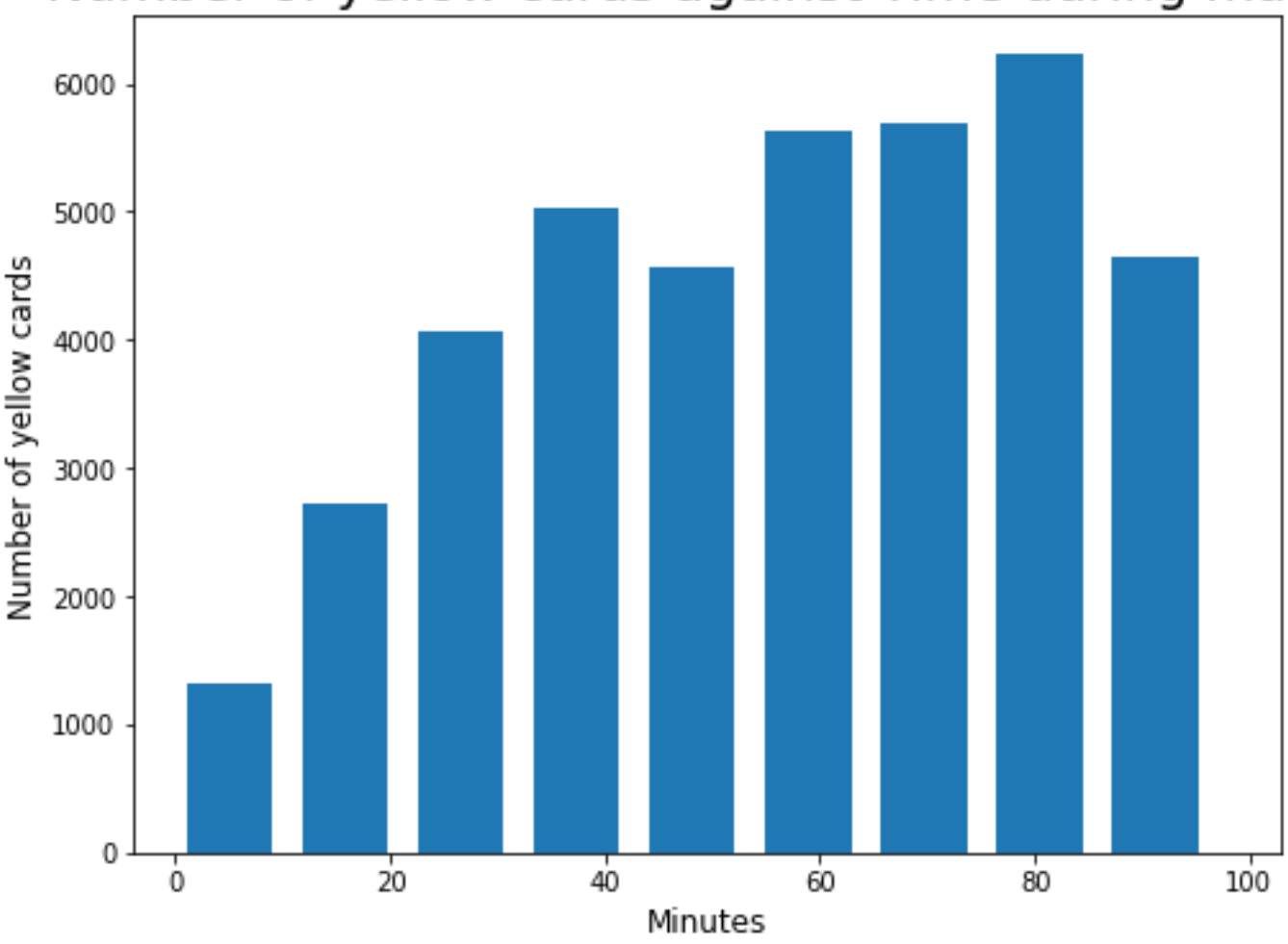
	rank	player	Goals
0	1	Lionel Messi	205
1	2	Cristiano Ronaldo	198
2	3	Zlatan Ibrahimovic	153
3	4	Robert Lewandowski	124
4	5	Edinson Cavani	121
5	6	Gonzalo Higuain	118
6	7	Pierreemerick Aubameyang	100
7	8	Luis Suarez	96
8	9	Diego Costa	93
9	10	Alexandre Lacazette	88

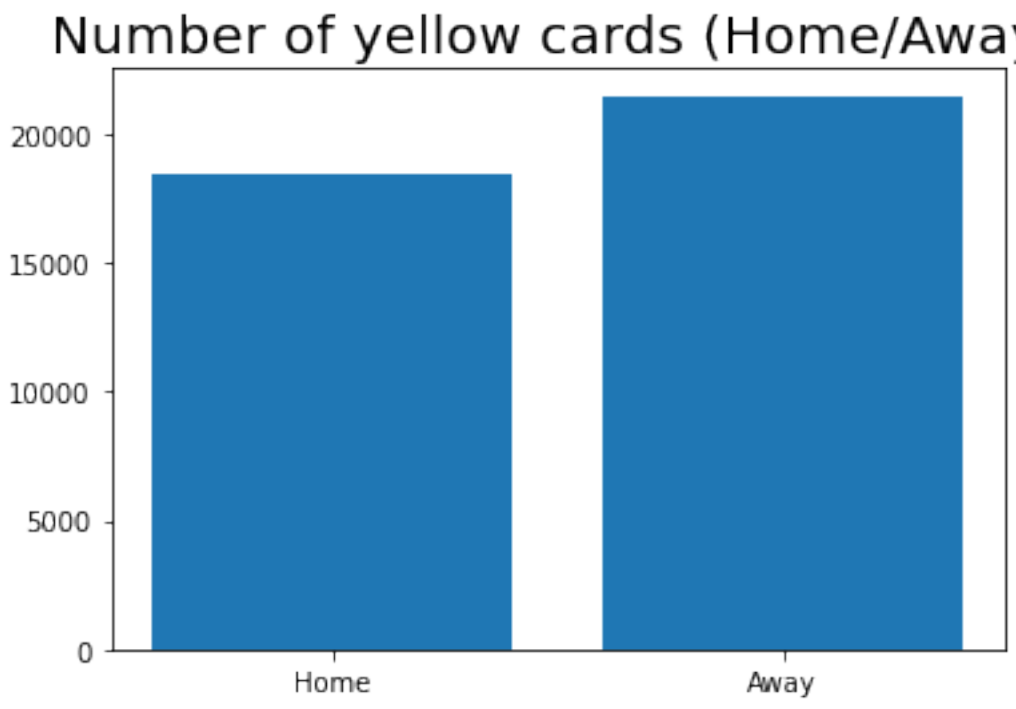
Table I: Top 10 players with the maximum number of goals.



The result corresponds to our common sense. We see some famous names: Messi, Ronaldo and Ibrahimović are on top of the chart. Messi socres over 200 goals in the league, whereas Ronaldo is slightly below him and scores 198 goals. Besides these two superstars, other top scoring players score around 100 goals.

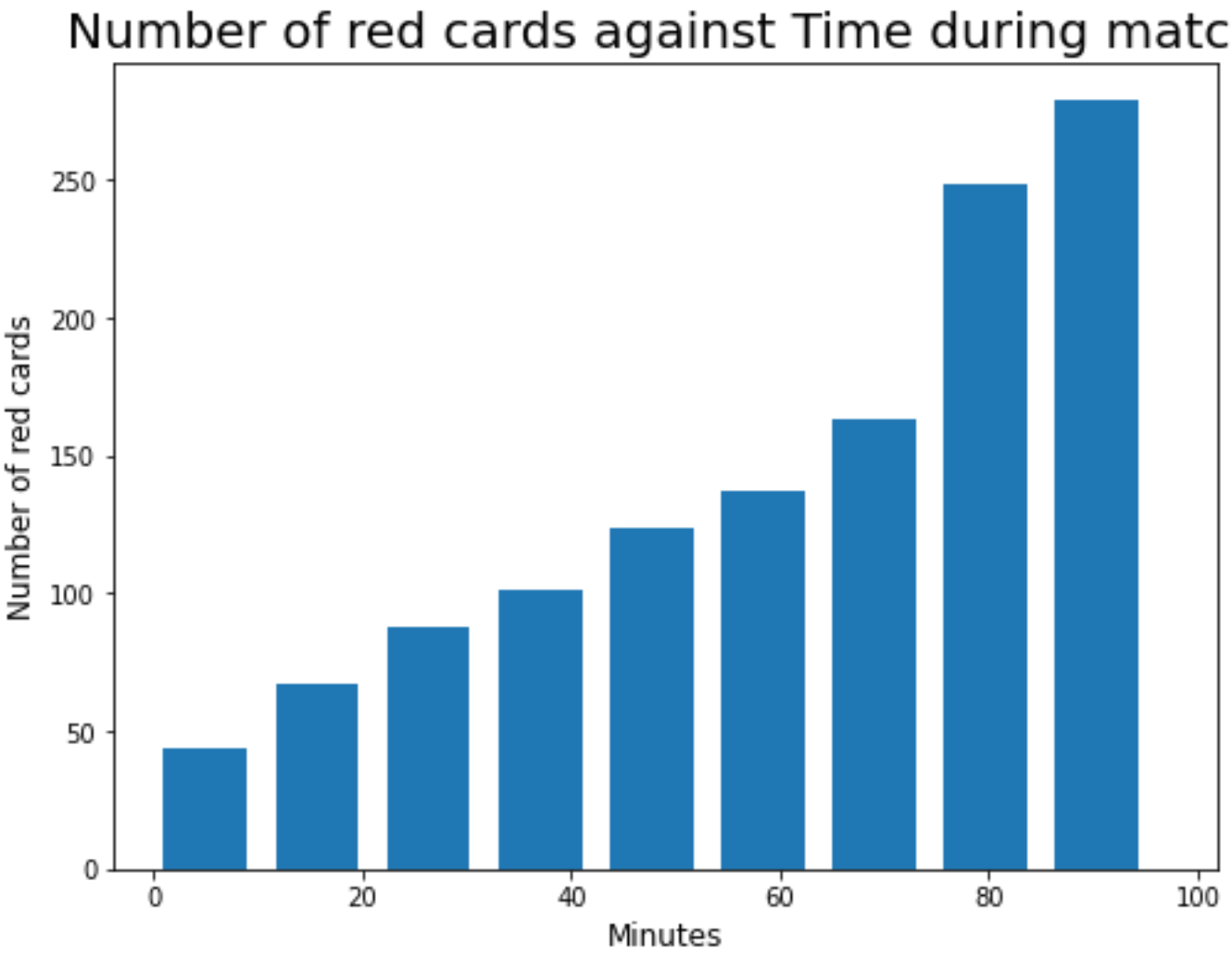
Next, I will do some analysis on yellow cards and red cards.

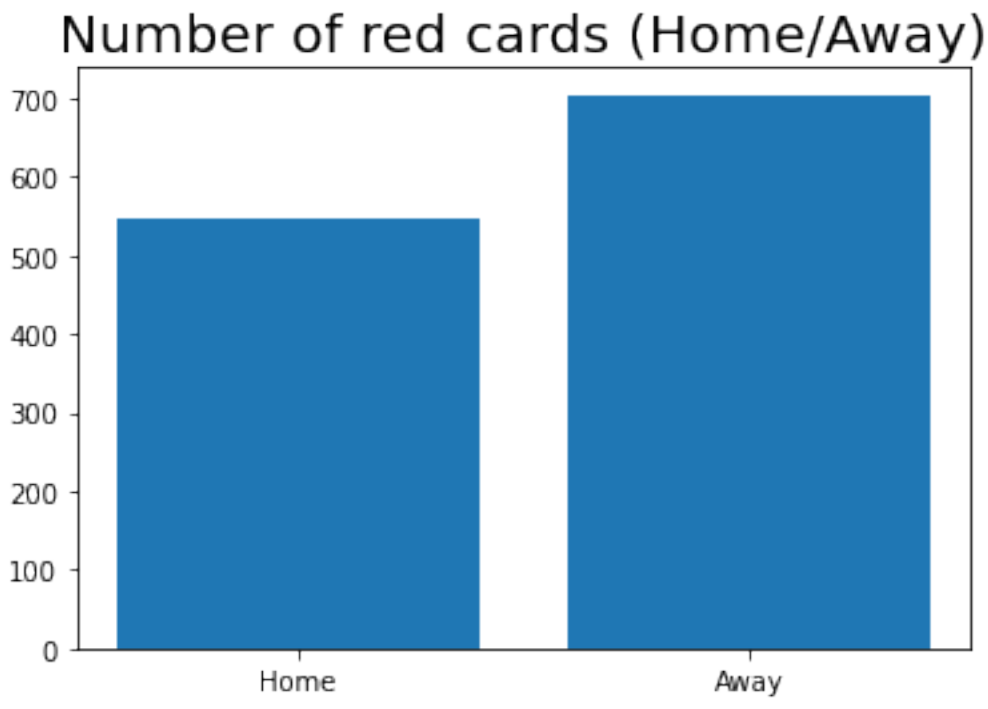




For yelloe cards, I observe that most of the yellow cards are issued in the second half of the game, especially around 80mins. The reasons might be that players' temper rises as the game moves on and they are more likely to take risky moves to secure the game.

More yellow cards has been issued to the away team compared to the home team.



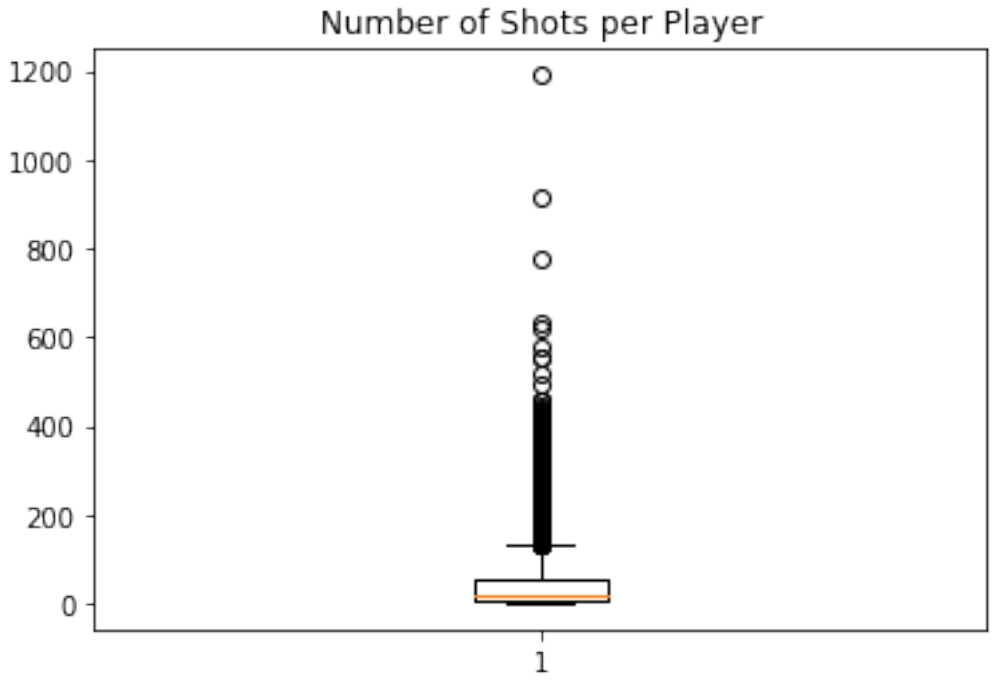


For the red cards, there is a clear increasing pattern where the probability of receiving a red card is higher as the game moves through. There is a sharp increase at about 80mins, which id similar to the yellow cards.

Also, more red cards has been issued to the away team compared to the home team. So, I am curious whether the referee has a favour to the home team.

Problems in Data

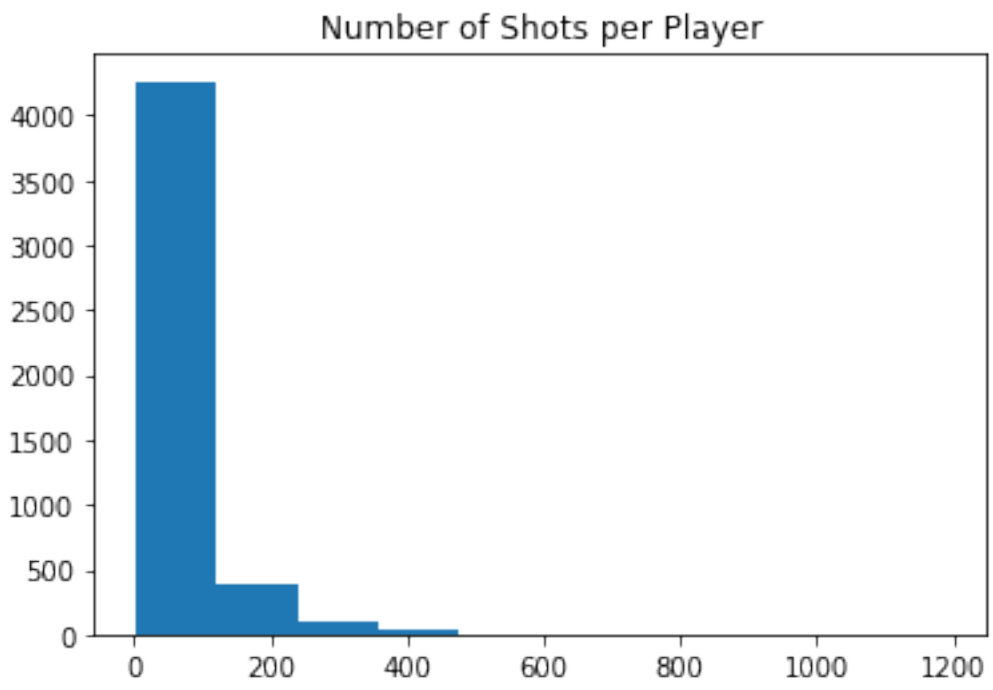
	id_event
count	4787.000000
mean	47.863380
std	73.901184
min	1.000000
25%	6.000000
50%	21.000000
75%	57.000000
max	1190.000000



player	id_event
cristiano ronaldo	1190
lionel messi	914
zlatan ibrahimovic	774
robert lewandowski	633
edinson cavani	623
pierreemerick aubameyang	580
antonio candreva	556
gonzalo higuain	552
antonio di natale	515
antoine griezmann	493

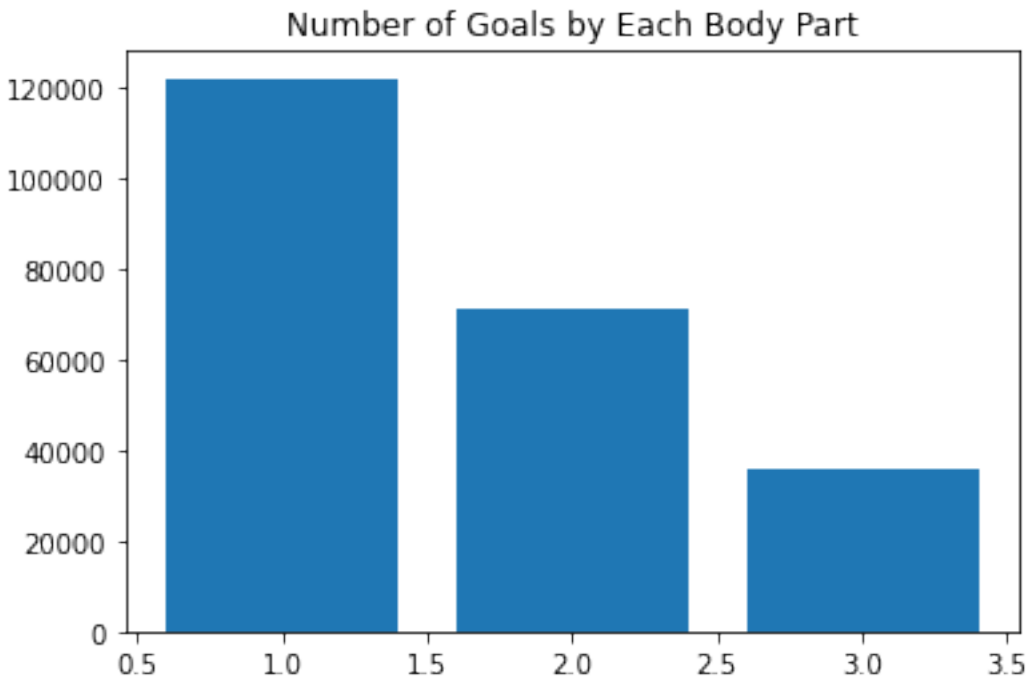
From the boxplot, we discovered that there are lots of outliers at the larger end. However, these outliers are not a probelm indeed. They are the records for the world famous players who did have a lot of attempts to goal. The real problem is that most of the users do not have many records in the data sets. 25% of the players have fewer than 6 attempts to goal and even there are a large proportion of them only have 1 record. Therefore, predictions for these players might not be very believable.

Shots per Player



The distribution of the number of shots per player is strongly skewed to the right. Most of the players have less than 100 shots.

Goals by Body Part



The shots are only classified into 3 different body part. However, the data can be finer-grained by classifying the body parts in a more precise way. For instance, by left foot, by right foot, by head, by knee, or by chest. The performance of a shot by head and a shot by knee can be different, but if they are both classified as the part ‘other’, the difference between outcomes cannot be identified.

Brainstorm a comprehensive list of factors that could affect the recorded data.

Before building up a model, it is important to notice that there are many factors that could conceivably influence the recorded data in real life, like the wrong data and many subjective and objective factors that affect whether a shot is a goal. Here are some of these factors:

- Incorrect data due to using webscraping, integrating and regex, since the author of the dataset webscraped and integrated different data sources, and then used regex to derive data.
- Missing data when recording, since the author of the dataset said "there are games that have been played during these seasons for which I could not collect detailed data".
- Incorrect data in the original data sources or websites.
- Weather condition.
- Whether the team adapts to the field.
- Team's injuries.
- Team's schedules.
- Team's moltivation.
- Team's playing strategy.
- Team's last game outcome, since this will affect player's mental conditions.
- Synergy or conflict between teammates
- Psychological aspects like the ability under pressure.
- Medical conditions of the players.
- Whether the player is a substitute player or not.
- Refs bad call or missed call.

Complementary Sources of Data

Since many factors that are not recoreded in this data can also affect the game outcome, or whether a shot is a goal or not, here are some complementary sources of data that might be useful in providing some extra information.

- More detailed information about players, e.g., their position, style of play, how many minutes they have played in a specific game, joining time, medical condition.
- More detailed information about teams, e.g., injuries, schedules.
- Information about commentaries.
- Information about some objective conditions, e.g., weather, field.
- Information about the goalkeeper, e.g., goalkeeper's performance, whether the goalkeeper is facing the attempt.
- Video recording from multiple positions.
- Dynamic analysis of the shot.

Model Staircase

Here are some models we proposed.

- Logistic regression using "bodypart", "situation" (baseline model).
- Use mean number of goals by each player assuming the probability that a shot is a goal for each player remains the same.
- Logistic regression using "side", "location", "bodypart", "assist_method", "situation".
- Resampling training dataset - undersample majority class (Variant 1 subvariant 1).
- Resampling training dataset - oversample minority class (Variant 1 subvariant 2).
- Resampling training dataset - oversample minority class + add prior correction for the bias term (Variant 1 subvariant 3).
- Change the threshold of baseline logistic regression from 0.5 to some smaller value like 0.2.
- Use neural network (Variant 2).
- Use decision tree as the classifier (Variant 2 subvariant).
- Use random forest as the classifier (Variant 2 subvariant).
- Use AdaBoost as the classifier (Variant 2 subvariant).
- Use gradient boosting as the classifier (Variant 2 subvariant).
- Use KNN as the classifier (Variant 2 subvariant).
- Add a latent variable for each player and each attempt (Varaint 3).

Baseline Model(Logistic Regression)

The baseline model uses logistic regression.

$$z = w^T x + b$$

$$y = \sigma(z) = \frac{1}{1+e^{-z}}$$

And it uses cross entropy loss $\mathcal{L}_{CE}(y, t) = -tlogy - (1-t)log(1-y)$ and variables 'bodypart' and 'situation'.

And the testing and training data is obtained by simply splitting the original data into random training and testing subsets.

From the result, it seems that the baseline model predicts well since its training accuracy is 89.30707 and testing accuracy is 89.43854. However, from the histogram, we can see that this model simply predicts all observations to be 0, and its f_1 score for 'is_goal=1' is 0.

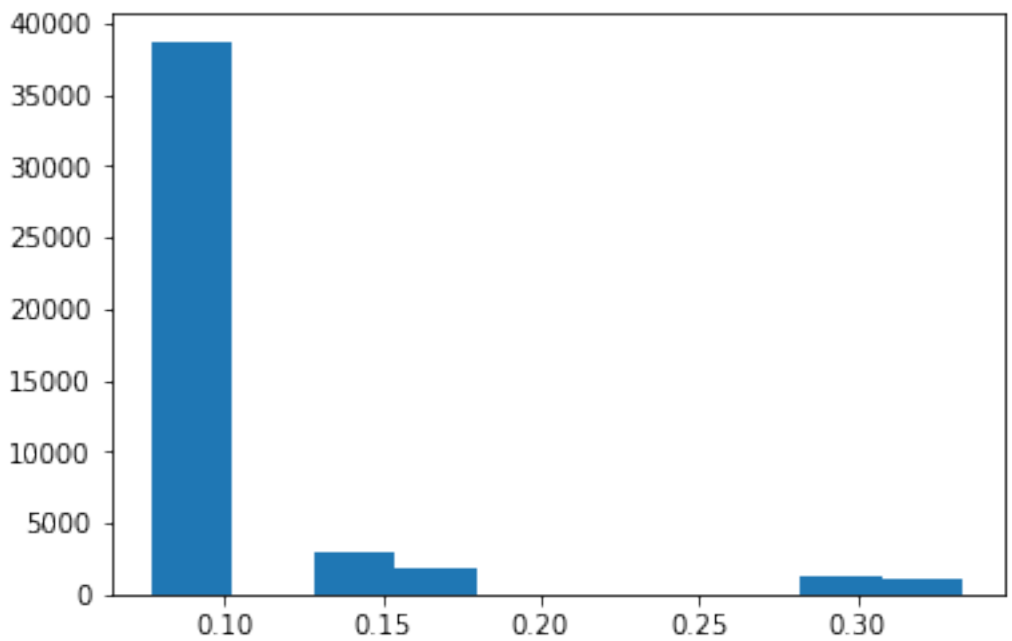
So in the variants below, we are trying to improve the model performance, decrease loss and increase f_1 score for 'is_goal=1'.

Number of epoch, training loss, training accuracy, testing accuracy using baseline model:

```
[0.39307082 0.2068895  0.12631471 0.05986647 0.34710723 0.00963406
 0.02761745 0.00871211]
0 0.62918705 89.30707 89.43854
50 0.334494 89.30707 89.43854
100 0.3323569 89.30707 89.43854
150 0.3316706 89.30707 89.43854
200 0.3312175 89.30707 89.43854
250 0.33089313 89.30707 89.43854
300 0.33065045 89.30707 89.43854
350 0.33046293 89.30707 89.43854
400 0.3303147 89.30707 89.43854
450 0.33019555 89.30707 89.43854
500 0.33009854 89.30707 89.43854
550 0.3300188 89.30707 89.43854
600 0.32995275 89.30707 89.43854
650 0.32989773 89.30707 89.43854
700 0.3298515 89.30707 89.43854
750 0.3298124 89.30707 89.43854
800 0.32977915 89.30707 89.43854
850 0.3297507 89.30707 89.43854
900 0.32972625 89.30707 89.43854
950 0.329705 89.30707 89.43854
1000 0.32968655 89.30707 89.43854
1050 0.3296703 89.30707 89.43854
1100 0.32965598 89.30707 89.43854
1150 0.3296434 89.30707 89.43854
1200 0.3296321 89.30707 89.43854
1250 0.32962206 89.30707 89.43854
1300 0.329613 89.30707 89.43854
1350 0.32960486 89.30707 89.43854
1400 0.32959744 89.30707 89.43854
1450 0.32959077 89.30707 89.43854
1500 0.3295847 89.30707 89.43854
1550 0.32957914 89.30707 89.43854
1600 0.32957405 89.30707 89.43854
1650 0.3295694 89.30707 89.43854
1700 0.32956502 89.30707 89.43854
1750 0.3295611 89.30707 89.43854
1800 0.32955745 89.30707 89.43854
1850 0.32955414 89.30707 89.43854
1900 0.329551 89.30707 89.43854
1950 0.32954812 89.30707 89.43854
2000 0.32954547 89.30707 89.43854
2050 0.329543 89.30707 89.43854
2100 0.3295407 89.30707 89.43854
2150 0.32953852 89.30707 89.43854
```

Predictions of testing data using baseline model.

```
(array([3.8702e+04, 1.0000e+00, 3.0090e+03, 1.7520e+03, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 1.3180e+03, 1.0450e+03])
,
 array([0.07692171, 0.10258147, 0.12824124, 0.153901  , 0.17956077,
        0.20522052, 0.23088029, 0.25654006, 0.2821998 , 0.30785957,
        0.33351934], dtype=float32),
 <a list of 10 Patch objects>)
```

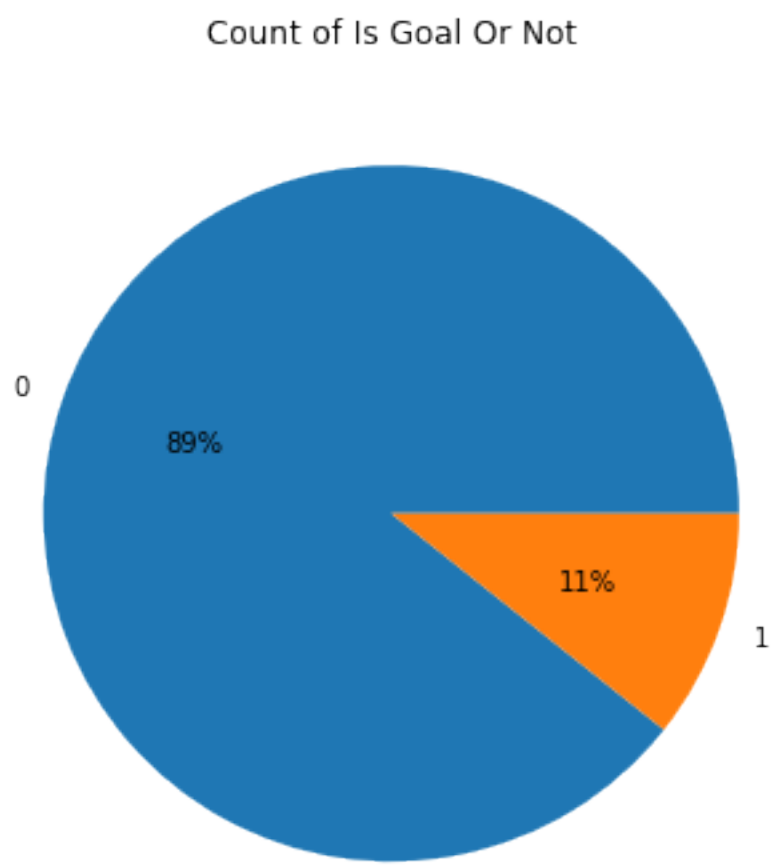


	precision	recall	f1-score	support
0.0	0.89	1.00	0.94	40987
1.0	0.00	0.00	0.00	4840
accuracy			0.89	45827
macro avg	0.45	0.50	0.47	45827
weighted avg	0.80	0.89	0.84	45827

Variant 1: Adapt training to handle imbalanced data.

We have already seen that the logistic regression simply predicts all observations to be 0. And this is because we have an imbalanced data.

```
Text(0, 0.5, '')
```



In the original dataset, only 11% of the data is goal, while 89% is not goal, which is quite imbalanced. And this kind of imbalanced data will lead to the accuracy paradox as what we have when using the baseline model.

So in this variant, we have tried several ways to handle imbalanced data, including using resampling techniques, using rare event corrections and changing to algorithms that are robust to imbalanced data.

And in this variant, we are going to variables "side", "location", "bodypart", "assist_method", "situation" to predict instead of only using "bodypart" and "situation", since variables like "side", "location", "assist_method" will also affect if a shot is a goal or not.

Subvariant 1: Resampling - Undersample majority class.

We used undersampling here to remove some observations of the majority class, i.e., the class where is_goal = 0. So we randomly selected samples where is_goal equals 0 without replacement from the original training data, and

the number of the samples where is_goal = 0 we selected here = (the number of observations where is_goal = 1 in training data) × 4.

So in this new training dataset,

number of observations where is_goal is 0 : number of observations where is_goal is 1 = 4 : 1.

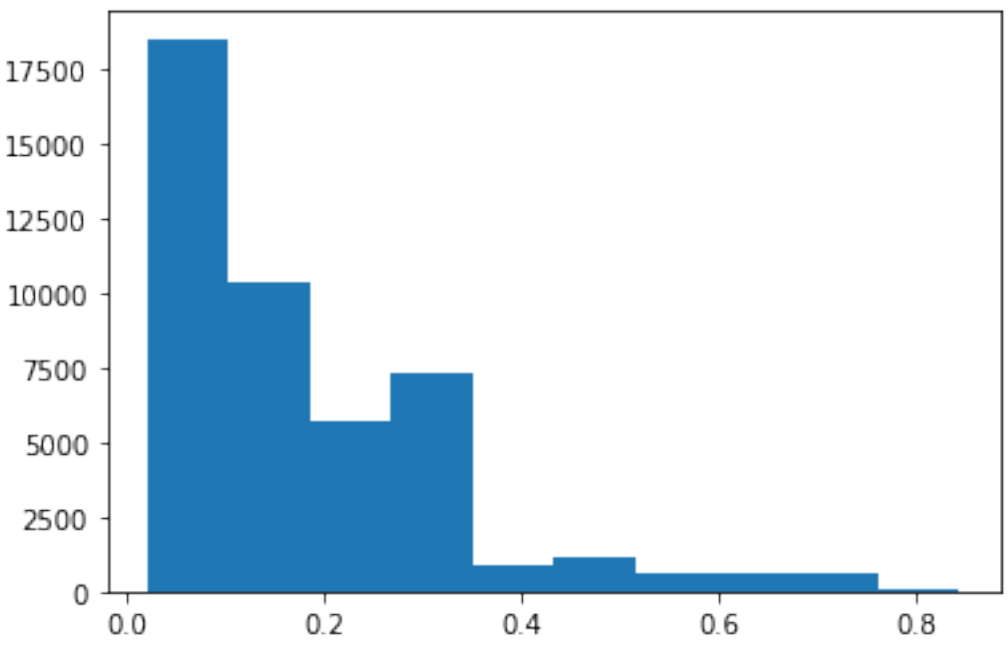
Number of epoch, training loss, training accuracy, testing accuracy using variant 1 subvariant 1:

```
[ 0.30000025  0.16434376  0.13565642  0.07129738  0.00280088  0.0036
4267
  0.00352023  0.02998826  0.00260701  0.02888118  0.00241824 -0.0074
3839
 -0.00700985  0.16729254  0.00358655  0.00366308  0.00055099 -0.0058
0072
  0.1565533   0.09849498  0.04495178  0.07145554  0.16247138  0.0535
4828
  0.00984134  0.00268354  0.2737209   0.00124484  0.01882047  0.0062
1397]
0 0.6686507 80.0 89.43854
50 0.4742415 80.0 89.43854
100 0.46250963 80.0 89.43854
150 0.45436007 80.0 89.43854
200 0.44780594 80.0 89.43854
250 0.44239628 79.998985 89.43854
300 0.437849 80.08367 89.47346
350 0.43397 80.11734 89.48872
400 0.43062037 81.220345 89.9557
450 0.4276976 81.559105 90.02335
500 0.42512405 81.73358 90.056076
550 0.4228397 81.786644 90.034256
600 0.42079768 81.80297 90.04299
650 0.41896072 82.03867 90.145546
700 0.41729885 82.26519 90.245926
750 0.41578767 82.27947 90.245926
800 0.41440722 82.311104 90.22629
850 0.41314098 82.58559 90.287384
900 0.41197512 82.71006 90.31575
950 0.41089803 82.816185 90.36812
1000 0.40989977 82.8172 90.35285
1050 0.40897202 82.87026 90.33758
1100 0.40810746 82.92536 90.32448
1150 0.40729985 82.923325 90.32448
1200 0.40654367 83.07025 90.34848
1250 0.4058343 83.17025 90.37467
1300 0.40516737 83.1733 90.35285
1350 0.40453935 83.17229 90.35285
1400 0.4039468 83.26718 90.39431
1450 0.40338686 83.33554 90.42704
1500 0.40285683 83.362076 90.41395
1550 0.4023544 83.362076 90.41395
1600 0.40187758 83.395744 90.40522
1650 0.40142435 83.395744 90.40522
1700 0.40099293 83.72634 90.566696
1750 0.40058184 83.96102 90.69544
1800 0.40018967 83.98347 90.688896
1850 0.3998151 84.00082 90.68016
```

```
1900 0.39945686 83.999794 90.68016
1950 0.39911404 83.999794 90.68016
2000 0.39878553 84.0151 90.65398
2050 0.39847037 84.0151 90.65398
2100 0.3981678 84.0151 90.65398
2150 0.3978771 84.058975 90.60161
```

Predictions of testing data using variant 1 subvariant 1.

```
(array([18504., 10310.,  5744.,  7312.,   890.,  1119.,   617.,   62
5.,
          636.,    70.]),
 array([0.02135868, 0.10370312, 0.18604755, 0.268392  , 0.3507364 ,
        0.43308085, 0.51542526, 0.59776974, 0.68011415, 0.7624586 ,
        0.84480304], dtype=float32),
 <a list of 10 Patch objects>)
```



	precision	recall	f1-score	support
0.0	0.92	0.98	0.95	40987
1.0	0.62	0.28	0.38	4840
accuracy			0.91	45827
macro avg	0.77	0.63	0.67	45827
weighted avg	0.89	0.91	0.89	45827

By using undersampling, logistic regression does not predict all observations into a single class anymore, and it predicts some observations to have is_goal=1. And this variant has increased testing accuracy and f_1 score for both classes. And we will go back to this result in summary section.

Subvariant 2: Resampling - Oversample minority class.

Similarly to undersampling, oversampling adds more copies of observations from the minority class, i.e., the class where is_goal = 1. So we randomly selected samples where is_goal equals 1 with replacement from the original training data, and the number of the samples where is_goal = 1 we selected here = (the number of observations where is_goal = 0 in training data) / 4. So in this new training dataset, number of observations where is_goal is 0 : number of observations where is_goal is 1 = 4 : 1.

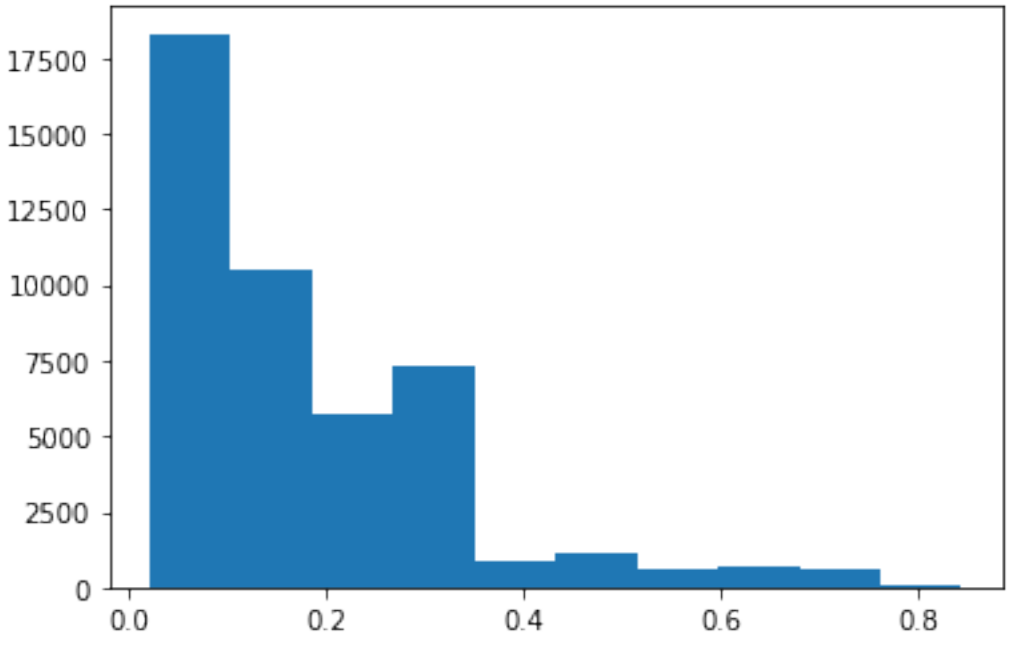
Number of epoch, training loss, training accuracy, testing accuracy using variant 1 subvariant 2:

```
[ 0.30000287  0.1633778   0.13662507  0.07056778  0.00272439  0.0037
3107
  0.0036089   0.03021507  0.00262665  0.02832388  0.00258023 -0.0069
1482
 -0.00706387  0.16754626  0.00365532  0.00376772  0.00047891 -0.0058
4461
  0.15675619  0.09866442  0.04458226  0.0719972   0.16185313  0.0533
8093
  0.00985911  0.00291253  0.27445954  0.00059863  0.01881905  0.0061
256 ]
0 0.66863245 80.00029 89.43854
50 0.47402453 80.00029 89.43854
100 0.46223047 80.00029 89.43854
150 0.4540596 80.00029 89.43854
200 0.447508 80.00029 89.43854
250 0.4421153 80.00029 89.43854
300 0.43759328 80.043785 89.4669
350 0.43374458 80.8159 89.73094
400 0.43042803 81.24789 89.92951
450 0.42753953 81.54208 90.001526
500 0.4250003 81.73804 90.03644
550 0.42274988 81.791794 90.03644
600 0.4207407 82.041016 90.14773
650 0.41893512 82.048836 90.145546
700 0.41730306 82.25457 90.221924
750 0.41581994 82.2878 90.21537
800 0.41446584 82.33569 90.221924
850 0.4132243 82.449554 90.30921
900 0.4120814 82.613266 90.28302
950 0.41102555 82.717354 90.34412
1000 0.410047 82.80434 90.36158
1050 0.40913764 82.81753 90.32885
1100 0.40828997 82.87764 90.322296
1150 0.40749812 82.89181 90.30266
1200 0.40675655 83.05943 90.32448
1250 0.4060604 83.06285 90.30921
1300 0.405406 83.14494 90.31139
1350 0.4047895 83.17769 90.31793
1400 0.40420756 83.24855 90.35067
1450 0.40365747 83.24855 90.35067
```

```
1500 0.40313673 83.26028 90.357216
1550 0.4026429 83.313545 90.39213
1600 0.40217403 83.31452 90.39213
1650 0.40172818 83.348724 90.387764
1700 0.40130365 83.67027 90.54924
1750 0.40089908 83.669785 90.54924
1800 0.40051305 83.70155 90.53615
1850 0.4001441 83.93123 90.667076
1900 0.3997913 83.94736 90.69544
1950 0.3994535 83.9669 90.688896
2000 0.39912966 83.9669 90.688896
2050 0.39881906 83.966415 90.688896
2100 0.39852077 83.982056 90.66271
2150 0.3982341 83.9972 90.65398
```

Predictions of testing data using variant 1 subvariant 2.

```
(array([18303., 10523.,  5745.,  7294.,   877.,  1159.,   574.,   64
6.,
          640.,    66.]),
 array([0.02067448, 0.10308277, 0.18549106, 0.26789936, 0.35030764,
        0.43271592, 0.5151242 , 0.5975325 , 0.6799408 , 0.76234907,
        0.8447574 ], dtype=float32),
 <a list of 10 Patch objects>)
```



	precision	recall	f1-score	support
0.0	0.92	0.98	0.95	40987
1.0	0.61	0.29	0.39	4840
accuracy			0.91	45827
macro avg	0.76	0.63	0.67	45827
weighted avg	0.89	0.91	0.89	45827

The result we get using this variant is quite similar to the result when using undersampling, since what oversampling does here is almost the same as what undersampling does. And oversampling also increases testing accuracy and f_1 score for both classes.

Subvariant 3: Oversampling + Prior correction for bias term.

Oversampling will not affect the slope estimates, however it will affect the intercept estimate, and any computation based on all parameters are incorrect. Gary King and Langche Zeng have proposed an adjustment of the intercept by computing the usual logistic regression MLE for intercept, and correcting the estimate based on prior information about proportion of ones in the population, and proportion of ones in our sample.

$\beta_0 = \hat{\beta}_0 - ln[(\frac{1-\tau}{\tau})(\frac{\bar{y}}{1-\bar{y}})]$, where τ is the fraction of 1s in the population, and $\bar{y}$ is the fraction of 1s in our sample data (King, G., & Zeng, L. 2001).

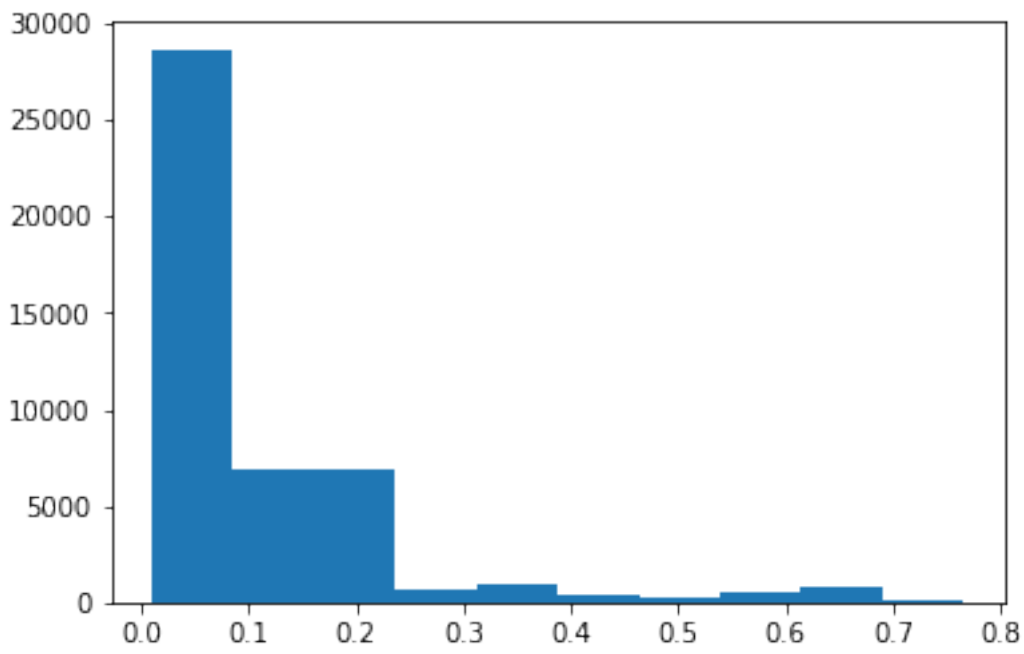
Number of epoch, training loss, training accuracy, testing accuracy using variant 1 subvariant 3:

```
[ 0.30000287  0.1641475   0.1358554   0.0709343   0.00269996  0.0038
4835
  0.00359424  0.03001715  0.0026462   0.02843871  0.00230657 -0.0073
5952
 -0.00709319  0.1677515   0.00367487  0.00380437  0.00047646 -0.0057
371
  0.15693945  0.09849828  0.04456515  0.07189699  0.16249576  0.0530
3397
  0.00965631  0.00291986  0.27391222  0.00069637  0.01901941  0.0063
7483]
0 0.51452905 80.00029 89.43854
50 0.49578005 80.00029 89.43854
100 0.47911647 80.00029 89.43854
150 0.46812543 80.00029 89.43854
200 0.46039286 80.00029 89.43854
250 0.45462102 79.9998 89.43854
300 0.4501262 80.08533 89.47346
350 0.44651338 80.10829 89.49091
400 0.44353867 80.13712 89.49964
450 0.44104236 80.17035 89.504005
500 0.43891546 80.940025 89.77022
550 0.43708026 81.38374 89.990616
600 0.43547985 81.446785 90.00371
650 0.43407133 81.706276 90.10409
700 0.43282163 81.88806 90.167366
750 0.43170482 81.89881 90.16955
800 0.43070036 81.93009 90.18482
850 0.42979157 81.93009 90.18482
```

```
900 0.4289648 81.93009 90.18482
950 0.42820886 82.1451 90.29611
1000 0.42751482 82.449554 90.36812
1050 0.42687482 82.54534 90.398674
1100 0.42628226 82.6636 90.49251
1150 0.4257317 82.73788 90.54924
1200 0.42521855 82.82535 90.531784
1250 0.42473885 82.95973 90.555786
1300 0.42428896 83.11074 90.564514
1350 0.42386615 83.11074 90.564514
1400 0.42346773 83.1078 90.54924
1450 0.42309135 83.1772 90.57979
1500 0.42273512 83.21727 90.60597
1550 0.42239732 83.21727 90.60597
1600 0.4220765 83.21727 90.60597
1650 0.42177117 83.555435 90.76745
1700 0.42148012 83.79636 90.89838
1750 0.42120242 83.79636 90.89838
1800 0.42093697 83.79636 90.89838
1850 0.420683 83.79636 90.89838
1900 0.4204395 83.79636 90.89838
1950 0.42020613 83.79636 90.89838
2000 0.41998202 83.790985 90.90275
2050 0.41976658 83.790985 90.90275
2100 0.41955936 83.790985 90.90275
2150 0.41935986 83.790985 90.90275
```

Predictions of testing data using variant 1 subvariant 3.

```
(array([28671.,  6848.,  6841.,   608.,   905.,   379.,   232.,   51
9.,
          767.,    57.]),
 array([0.00914319, 0.08478262, 0.16042206, 0.23606148, 0.3117009 ,
        0.38734034, 0.46297976, 0.5386192 , 0.61425865, 0.6898981 ,
        0.7655375 ], dtype=float32),
 <a list of 10 Patch objects>)
```

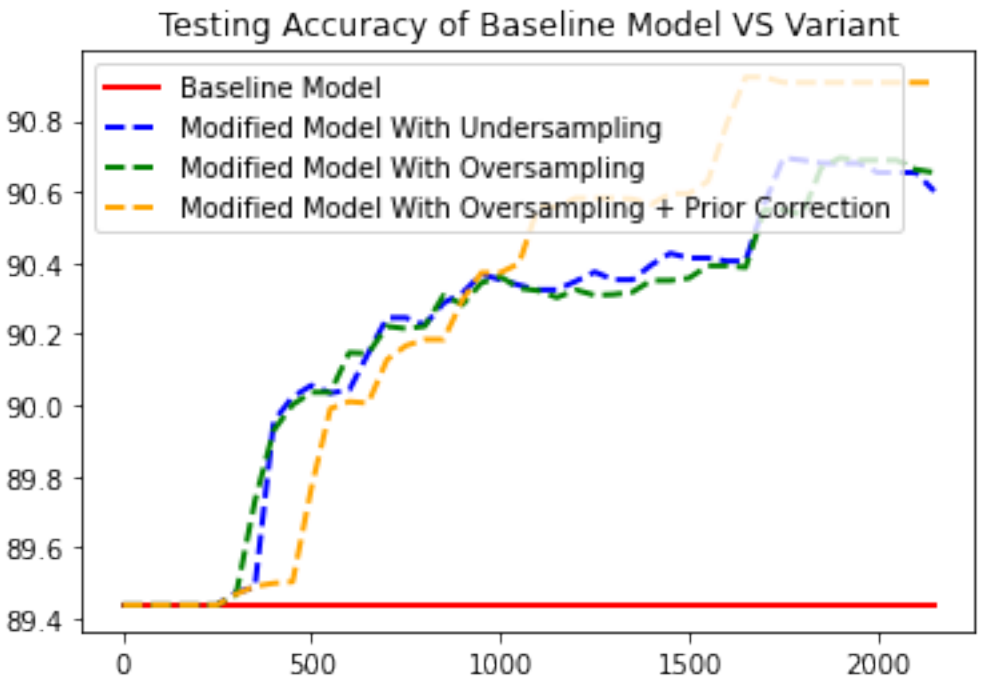


	precision	recall	f1-score	support
0.0	0.92	0.99	0.95	40987
1.0	0.73	0.22	0.34	4840
accuracy			0.91	45827
macro avg	0.82	0.61	0.65	45827
weighted avg	0.90	0.91	0.89	45827

Using oversampling and prior correction for bias term together, we can get 90.90711% testing accuracy, which is 1.64% higher than using the baseline logistic regression, and 0.24% higher than only using oversampling, without using prior correction. And the precision for is_goal=1 is 0.73, which is 12.31% higher than only using oversampling, and recall for is_goal=0 is 0.99, which is also higher than only using oversampling.

Variant 1 Model Summary and Sanity Check

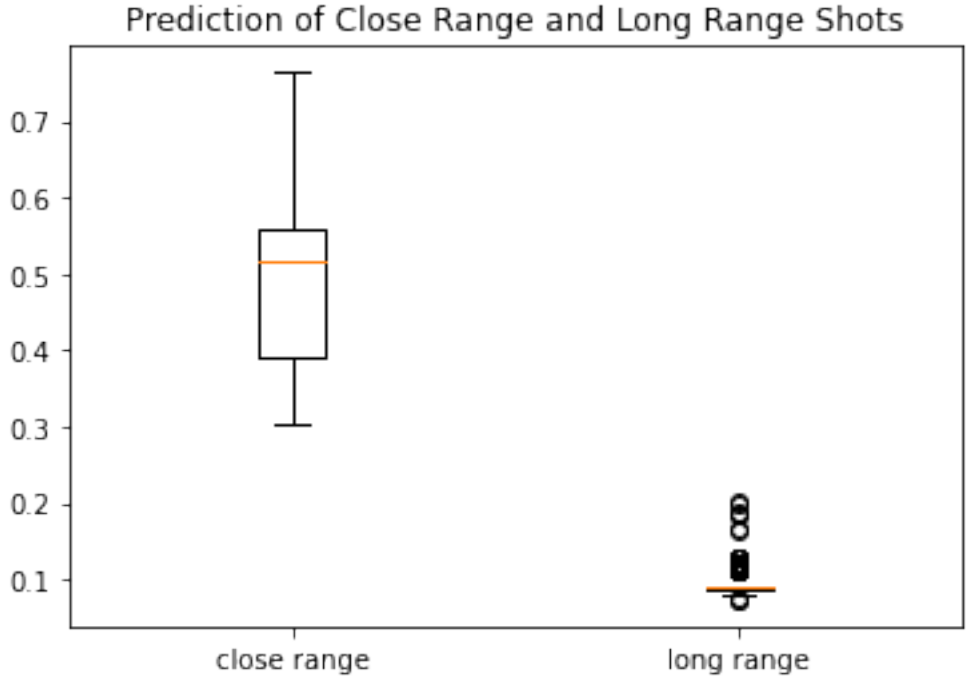
In variant 1, we are handling the imbalanced data. And the three methods we used here, including undersampling, oversampling, oversampling + prior correction all achieve a better testing accuracy, and a higher f_1 score for not goal class. And they all help avoid accuracy paradox, i.e., simply predict all classes to be not goal, so the precision and recall for is goal class also get higher. The summary plot and table for variant 1 are shown below.



	$Training Loss$	$Testing Accuracy$	$F1 Score (Not Goal)$	$F1$
$Baseline Model$	$**0.3295**$	89.4385	0.94	
$Undersampling$	0.3991	90.6802	0.95	*
$Oversampling$	0.3983	90.6889	0.95	*
$Oversampling + Prior Correction$	0.4199	$**90.9071**$	0.95	

From the plot and table above, oversampling+prior correction can achieve the highest testing accuracy, which is 90.9071%, and is 1.64% higher than using the baseline model. Also the precision of is_goal=1 is 0.73, which means that for all the shots that are predicted to be goals, 73% of them are predicted correctly. Precision is quite important for coaches to make decisions, since if they are using this decision-support tool to train the players, they will rely on the shots that are predicted to be goals to make strategy and train players. For example, if the decision-support tool predicts that a shot from long distance and using head will be a goal, then coach will train players to shot from long distance and use head. However, this prediction is incorrect, and should not be a goal. And this will become a big decision-making problem. That's why having a higher precision is quite important here, and our new variant has achieved a relatively good precision.

And to do sanity check for this variant, we will take a look at the final prediction for shots from close range and shots from long range using our variant model.



From the boxplot above, it is very clear that the predictions for shots from close range are closer to 1, and predictions for shots from long range are closer to 0. And this makes sense, since shots from far away are less likely to succeed, and shots that are in close range are more likely to succeed.

Variant 1 Result and Limitation

In variant 1, the main problem we are dealing with is to handle the imbalanced data, and avoid accuracy paradox. In addition, since we are using resampling to handle the imbalanced data, we also introduce prior correction to correct the bias term that is affected by using resampling.

And we used undersampling the majority class by removing some observations of the majority class, i.e., the class where is_goal = 0. Similarly, we have also tried oversampling the minority class by adding more copies of observations from the minority class, i.e., the class where is_goal = 1. These two resampling methods help handle imbalanced data by randomly selecting samples to achieve number of observations where is_goal is 0 : number of observations where is_goal is 1 = 4 : 1.

The oversampling+prior correction variant can achieve 90.9071% testing accuracy, which is 1.64% higher than using the baseline model as shown in Section 'Variant 1 Model Summary and Sanity Check'. In addition, its precision for is_goal=1 is 0.73, which is relatively good compared to 0 in baseline logistic regression model.

And 27.52% of the soccer matches end in a draw, so using this variant to support decision-making such as planning strategy and instructing players, it leads to 0.45% more games won compared to using the baseline model.

However, oversampling may cause some problems such as overfitting since it duplicates examples from minority class in the training dataset. So if more shots that are not goal occur, and fewer shots that are goal occur, in other words, if the data becomes more imbalanced, then the risk of getting overfitting will be higher.

Variant 2: Neural Networks

The second moedel for our model staircase is to build a neural network. While simple models only calculate the distances to predict the classes, complex models such as neural network are more sophisticated and may perform well for the imbalanced data.

For our neural networks, we use Pytorch to simplify the process. Our input is the number of the training data, and our output is a label 0 or 1 to indicate whether it's a goal or miss. Our training data consists of side, location, bodypart, assist_method, situation and fast_break. We have two hidden layers, 16 and 4, resulting from tuning the hyperparameters. Since we are not dealing with imagaes, it's hard to determine the size of the hidden layers as well as the convolution, which adds each element of an image to its local neighbors, weighted by a kernel, or a small matrix, that helps us extract certain features from the input image. We also use the rectified-linear activation function and sigmoid function over x in our forward function. For our loss function, we use a binary cross entropy loss and an Adam optimizer with a learning rate 0.01. The number of epoch is 2500.

Number of epoch, training loss, testing accuracy using variant 2:

```
epoch 0
    Train set - loss: 0.8644660115242004, accuracy: 0.10692932456731
796
    Test  set - loss: 0.8648345470428467, accuracy: 0.10561459511518
478

epoch 50
    Train set - loss: 0.287494421005249, accuracy: 0.893070697784423
8
    Test  set - loss: 0.28471657633781433, accuracy: 0.8943853974342
346

epoch 100
    Train set - loss: 0.26673242449760437, accuracy: 0.9059942960739
136
    Test  set - loss: 0.26368656754493713, accuracy: 0.9068016409873
962

epoch 150
    Train set - loss: 0.26514506340026855, accuracy: 0.9088801145553
589
    Test  set - loss: 0.26208066940307617, accuracy: 0.9093111157417
297

epoch 200
    Train set - loss: 0.26430007815361023, accuracy: 0.9101893901824
951
    Test  set - loss: 0.26131927967071533, accuracy: 0.9106858372688
293

epoch 250
    Train set - loss: 0.263606995344162, accuracy: 0.910342156887054
4
    Test  set - loss: 0.26069843769073486, accuracy: 0.9107949733734
131

epoch 300
    Train set - loss: 0.26301828026771545, accuracy: 0.9103748798370
361
    Test  set - loss: 0.26017239689826965, accuracy: 0.9107730984687
805

epoch 350
    Train set - loss: 0.26249659061431885, accuracy: 0.9103530645370
483
    Test  set - loss: 0.2596947252750397, accuracy: 0.91086041927337
65
```

```
epoch 400
    Train set - loss: 0.26206910610198975, accuracy: 0.9103530645370
483
    Test  set - loss: 0.2593104839324951, accuracy: 0.91086041927337
65

epoch 450
    Train set - loss: 0.26177945733070374, accuracy: 0.9104076027870
178
    Test  set - loss: 0.25908029079437256, accuracy: 0.9107730984687
805

epoch 500
    Train set - loss: 0.26152998208999634, accuracy: 0.9104076027870
178
    Test  set - loss: 0.2588808536529541, accuracy: 0.91075128316879
27

epoch 550
    Train set - loss: 0.2613106966018677, accuracy: 0.91040217876434
33
    Test  set - loss: 0.25873419642448425, accuracy: 0.9107512831687
927

epoch 600
    Train set - loss: 0.26111292839050293, accuracy: 0.9104076027870
178
    Test  set - loss: 0.2585940957069397, accuracy: 0.91075128316879
27

epoch 650
    Train set - loss: 0.260935515165329, accuracy: 0.910407602787017
8
    Test  set - loss: 0.2584634125232 6965, accuracy: 0.9107294678688
049

epoch 700
    Train set - loss: 0.2607730031013489, accuracy: 0.91041308641433
72
    Test  set - loss: 0.25833654403686523, accuracy: 0.9107294678688
049

epoch 750
    Train set - loss: 0.2606198489665985, accuracy: 0.91041308641433
72
    Test  set - loss: 0.2582103908061981, accuracy: 0.91072946786880
49

epoch 800
    Train set - loss: 0.26047053933143616, accuracy: 0.9104185104370
117
    Test  set - loss: 0.25810036063194275, accuracy: 0.9107512831687
```

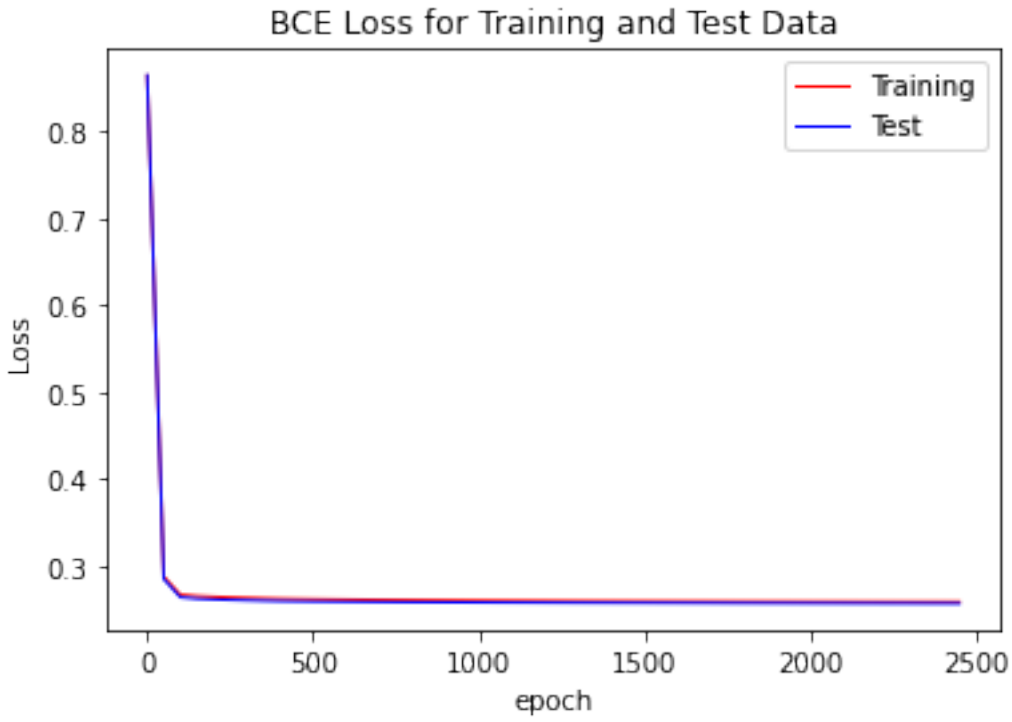
```
927

epoch 850
    Train set - loss: 0.260336399078369 14, accuracy: 0.9104130864143
372
    Test  set - loss: 0.25801485776901245, accuracy: 0.9107294678688
049

epoch 900
    Train set - loss: 0.26021575927734375, accuracy: 0.9104130864143
372
    Test  set - loss: 0.25793009996414185, accuracy: 0.9107294678688
049

epoch 950
    Train set - loss: 0.26010510325431824, accuracy: 0.9104130864143
372
    Test  set - loss: 0.25784558057785034, accuracy: 0.9107294678688
049

epoch 1000
    Train set - loss: 0.26000189781188965, accuracy: 0.9104349017143
25
    Test  set - loss: 0.25776129961013794, accuracy: 0.9107949733734
131

epoch 1050
    Train set - loss: 0.2599065601825714, accuracy: 0.91055488586425
78
    Test  set - loss: 0.257679283618927, accuracy: 0.910882234573364
3

epoch 1100
    Train set - loss: 0.2598131597042084, accuracy: 0.91055488586425
78
    Test  set - loss: 0.25760865211486816, accuracy: 0.9108604192733
765

epoch 1150
    Train set - loss: 0.25972282886505127, accuracy: 0.9105385541915
894
    Test  set - loss: 0.2575533390045166, accuracy: 0.91088223457336
43

epoch 1200
    Train set - loss: 0.25963449478149414, accuracy: 0.9105439782142
639
    Test  set - loss: 0.25747647881507874, accuracy: 0.9108822345733
643

epoch 1250
    Train set - loss: 0.25954633951187134, accuracy: 0.9105548858642
578
```

```
    Test  set - loss: 0.257419556379318 24, accuracy: 0.9108822345733
643

epoch 1300
    Train set - loss: 0.2594752311706543, accuracy: 0.91053855419158
94
    Test  set - loss: 0.25736191868782043, accuracy: 0.9108822345733
643

epoch 1350
    Train set - loss: 0.25940534472465515, accuracy: 0.9105712771415
71
    Test  set - loss: 0.2573074698448181, accuracy: 0.91077309846878
05

epoch 1400
    Train set - loss: 0.25934574007987976, accuracy: 0.9105712771415
71
    Test  set - loss: 0.257267564535141, accuracy: 0.910794973373413
1

epoch 1450
    Train set - loss: 0.2592931389808655, accuracy: 0.91055488586425
78
    Test  set - loss: 0.2572230398654938, accuracy: 0.91081678867340
09

epoch 1500
    Train set - loss: 0.25924307107925415, accuracy: 0.9105385541915
894
    Test  set - loss: 0.2571963667869568, accuracy: 0.91081678867340
09

epoch 1550
    Train set - loss: 0.2591937482357025, accuracy: 0.91053307056427
    Test  set - loss: 0.25715389847755 43, accuracy: 0.91079497337341
31

epoch 1600
    Train set - loss: 0.2591495215892792, accuracy: 0.91052216291427
61
    Test  set - loss: 0.25712159276008606, accuracy: 0.9107730984687
805

epoch 1650
    Train set - loss: 0.2591056227684021, accuracy: 0.91052216291427
61
    Test  set - loss: 0.25708478689193726, accuracy: 0.9107730984687
805

epoch 1700
    Train set - loss: 0.25906285643577576, accuracy: 0.9105221629142
761
```

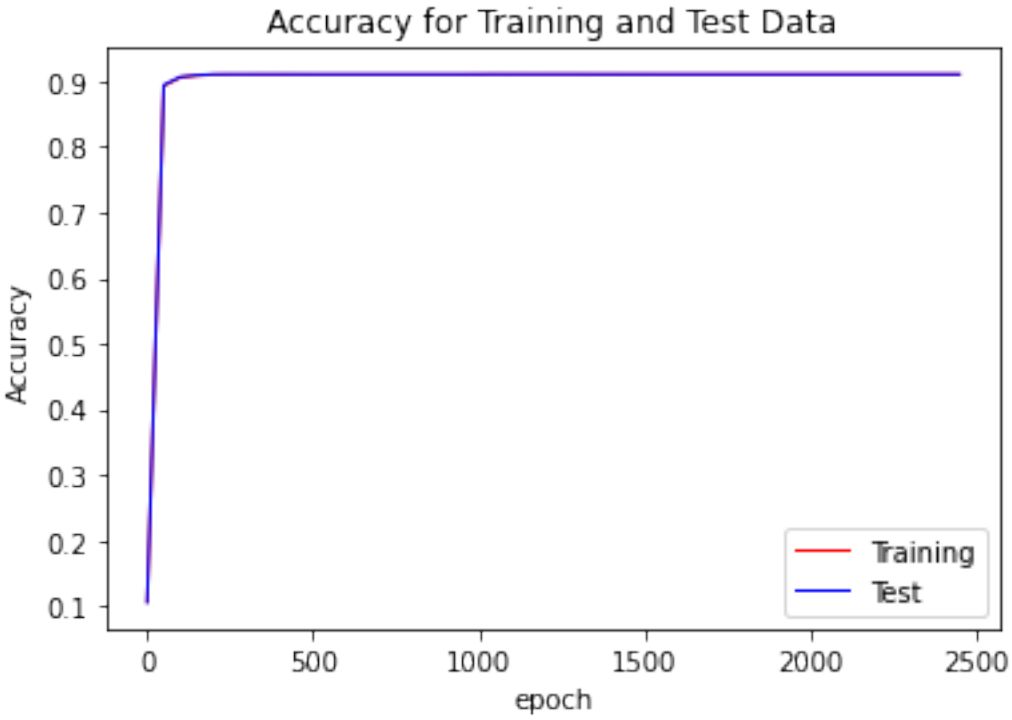
```
    Test  set - loss: 0.2570442855358124, accuracy: 0.91077309846878
05

epoch 1750
    Train set - loss: 0.25901755690574646, accuracy: 0.9105167388916
016
    Test  set - loss: 0.2569862902164459, accuracy: 0.91079497337341
31

epoch 1800
    Train set - loss: 0.2589762806892395, accuracy: 0.91052216291427
61
    Test  set - loss: 0.2569381594657898, accuracy: 0.91077309846878
05

epoch 1850
    Train set - loss: 0.2589346170425415, accuracy: 0.91052216291427
61
    Test  set - loss: 0.25689783692359924, accuracy: 0.9107730984687
805

epoch 1900
    Train set - loss: 0.25889673829078674, accuracy: 0.9105221629142
761
    Test  set - loss: 0.25686103105545044, accuracy: 0.9107730984687
805

epoch 1950
    Train set - loss: 0.25886058807373047, accuracy: 0.9105221629142
761
    Test  set - loss: 0.25683102011680603, accuracy: 0.9107730984687
805

epoch 2000
    Train set - loss: 0.25882625579833984, accuracy: 0.9105221629142
761
    Test  set - loss: 0.2567976117134094, accuracy: 0.91077309846878
05

epoch 2050
    Train set - loss: 0.2587929666042328, accuracy: 0.91052764654159
55
    Test  set - loss: 0.2567676603794098, accuracy: 0.91077309846878
05

epoch 2100
    Train set - loss: 0.25876113772392273, accuracy: 0.9105221629142
761
    Test  set - loss: 0.25673907995224, accuracy: 0.9107949733734131

epoch 2150
    Train set - loss: 0.2587301433086395, accuracy: 0.91052764654159
55
```

```
    Test  set - loss: 0.256712406873703, accuracy: 0.910773098468780
5

epoch 2200
    Train set - loss: 0.2587002217769623, accuracy: 0.91052764654159
55
    Test  set - loss: 0.2566919922828674 3, accuracy: 0.9107730984687
805

epoch 2250
    Train set - loss: 0.25867101550102234, accuracy: 0.9105276465415
955
    Test  set - loss: 0.256661593914032, accuracy: 0.910773098468780
5

epoch 2300
    Train set - loss: 0.2586424648761749, accuracy: 0.91052764654159
55
    Test  set - loss: 0.25664156675338745, accuracy: 0.9107730984687
805

epoch 2350
    Train set - loss: 0.2586153745651245, accuracy: 0.91052764654159
55
    Test  set - loss: 0.2566141188144684, accuracy: 0.91077309846878
05

epoch 2400
    Train set - loss: 0.2585885524749756, accuracy: 0.91052764654159
55
    Test  set - loss: 0.2565942108631134, accuracy: 0.91077309846878
05

epoch 2450
    Train set - loss: 0.2585621774196625, accuracy: 0.91052764654159
55
    Test  set - loss: 0.25656771659851074, accuracy: 0.9107730984687
805
```

Model Summary and Sanity Check



When we look at the training curve, the BCE loss converges to around 0.25 as epoch number increases. Since we don't see the curev oscillating as epoch number increases, there seems not to have signs of overfitting. Thus, we are happy with our neural networks and we want to verify its performance in terms of accuracy to furthur justify the validity of the model.



For the training and test accuracy, the result is definitely reasonable. Since our baseline logistic regression model always predict 0s due to the imbalance of the data, it's reasonable to see that in the first 100 epoches our neural network predicts the same result. However, the neural network is able to catch the deep information of our data and starts to predict 1s, increasing both traing and test accuracy to slightly above 90%. The accuracy also converges, so there is no obvious indication of overfitting or underfitting. We are happy with the result so far.

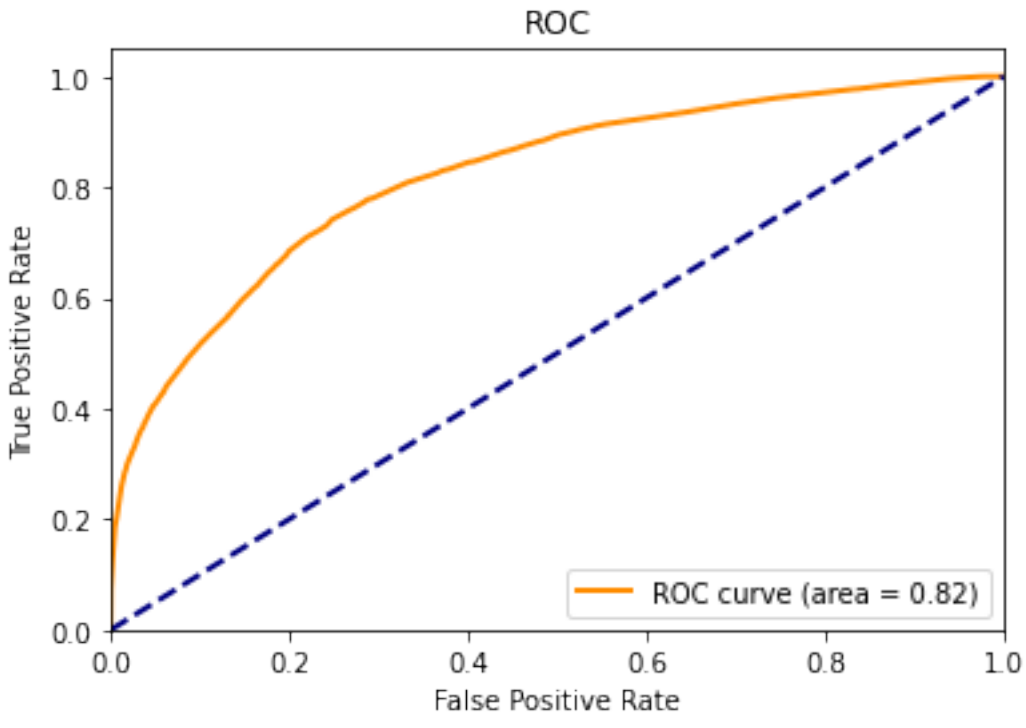
Reault and Limitations

Overall, our neural network performs better than the baseline logistic regression model. After both finish training, the loss decreases from 0.3295 to 0.2560. The training accuracy increases from 0.8931 to 0.9107 and the test accuracy increases from 0.8944 to 0.9108. The test accuracy increases by 1.64%.

	precision	recall	f1-score	support
0.0	0.92	0.99	0.95	40987
1.0	0.71	0.26	0.38	4840
accuracy			0.91	45827
macro avg	0.81	0.62	0.67	45827
weighted avg	0.90	0.91	0.89	45827

When we look at the classification report, we see that the precision for class-0 (miss) increases from 0.89 to 0.92. The precision for class-1 (goal) increases from 0 to 0.71, which is a dramatical increase and the model finally has the ability to predict 1s. The recall for class-1 increases from 0 to 0.26, meaning that the model is finally able to find positive instances. The F1 score for class-1 also increases from 0 to 0.38.

Our neural network obtains an ROC-AUC of 82%



Confusion Matrix:

```
[[40467   520]
 [ 3569  1271]]
```

Furthermore, we obtain a pretty good ROC-AUC metric of 82%. This looks very impressing. We also look at the confusion matrix. From all the shots that were missed, our model correctly predicts 40473 as no-goals, and makes a mistake in 3527 cases in which they should be goals. From the other column, we see that it correctly predicts 514 goals, but fails to predict 1268 as goals. This sill reflects that our model is not good enough to predict actual goals. In fact, our model achieves such high accuracy because even predicting all 0s would give an accuracy of 89%.

Subvaraint: Use of Different Classifiers

In addition of the neural netwrok, I tried to implement multiple classifiers to see whether the result would be better. I used five built-in classifiers from scikit-learn, and they are decision tree, random forest, ada boost, gradient boosting, and KNN.

```
DecisionTreeClassifier
Accuracy: 91.0729%
RandomForestClassifier
Accuracy: 91.0839%
AdaBoostClassifier
Accuracy: 90.8897%
GradientBoostingClassifier
Accuracy: 91.0555%
KNeighborsClassifier
Accuracy: 90.5776%
```

We see that the Gradient Boosting Classifier achieves the best result, and it has a test accuracy of 91.06%. I want to discover it further.

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse',
init=None,
                           learning_rate=0.1, loss='deviance', max_d
epth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_s
plit=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimator
s=100,
                           n_iter_no_change=None, presort='deprecate
d',
                           random_state=None, subsample=1.0, tol=0.0
001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

	precision	recall	f1-score	support
0	0.92	0.99	0.95	40939
1	0.72	0.26	0.38	4888
accuracy			0.91	45827
macro avg	0.82	0.62	0.67	45827
weighted avg	0.90	0.91	0.89	45827

`Our Gradient Boosting Classifier obtains an ROC-AUC of 82%`

Overall, the performance of the gradient boosting classifier is almost the same as the neural network. They have very similar accuracy, precision, recall, and F1 score. As its name says, it's a boosting classifier, with many weak models such as decision trees. Gradient Boosting is a sequential process and thus every time it makes an incorrect prediction, it focuses more on that incorrectly predicted data point. In our case, it will focus more on those data which actually scores. With a number of iterations, the correctness will rise.

Variant 3: Latent Variable Per-Player

The baseline model considered only the `bodypart` and `situation` of shots. In this variant, we add a latent variable for each player in addition to the baseline model.

There are 3 different `bodypart` : right_foot, left_foot, and header, and 4 different `situation` : open_play, set_piece, corner, and free_kick, so for simplicity we define the 12 different combinations of `bodypart` and `situation` as a new variable `attempt` , which represents the type of attempt of a shot. And because the sample is imbalanced, with too many shots not goal, we drop the data of player and attempt combinations that never goals. It is not meaningful to predict whether a player is able to goal in a situation that he never goals before.

After adjusting the dataset, we transform both the training and testing sets into sparse matrices such that each row represents a `player` and each column represents a type of `attempt` . If at least 50% of shots of $player_i$ and $attempt_j$ goal, then the ij^{th} entry of the sparse matrix is 1; otherwise, the ij^{th} entry of the sparse matrix is 0.

IRT Model

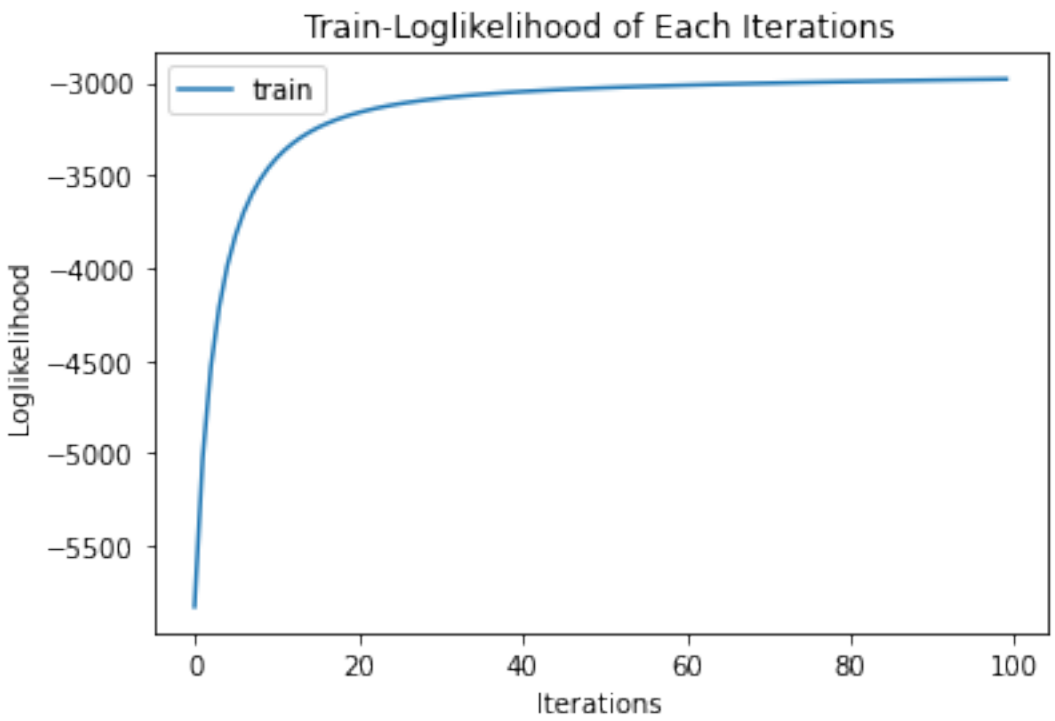
In this variant, we implemented an Item-Response Theory (IRT) model to predict player's goal rates. The IRT model assigns each player an ability value and each attempt a difficulty value to formulate a probability distribution. In the one-parameter IRT model, β_j represents the difficulty of the attempt j, and θ_i represents the ability of student i. Then, the probability that the shots of player i and attempt j could goal is formulated as:

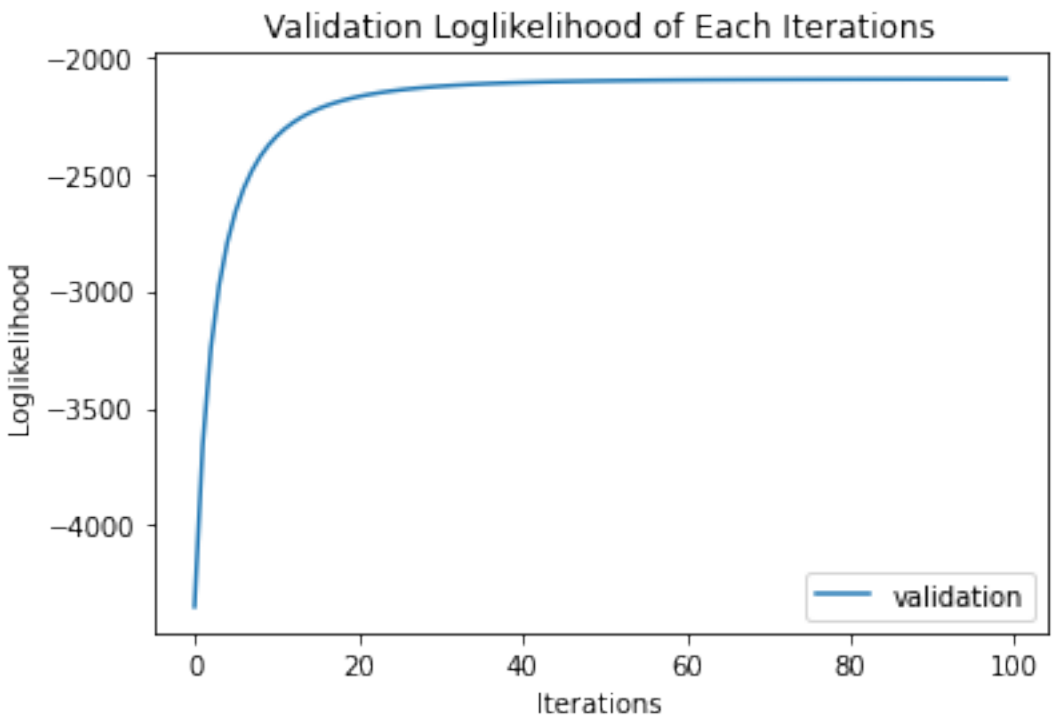
$$p(c_{ij} = 1|\theta_i, \beta_j) = \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}$$

```
train_nllk: 5826.5951997869      test_nllk: 4346.725969291417
train_nllk: 5023.893795284591    test_nllk: 3657.761924299098
train_nllk: 4533.658399805418    test_nllk: 3243.485177569156
train_nllk: 4212.261962079653    test_nllk: 2976.474630353644
train_nllk: 3987.9316286411417   test_nllk: 2793.295392899025
train_nllk: 3823.638972926018    test_nllk: 2661.40469200411
train_nllk: 3698.8432292855796   test_nllk: 2562.8758383880686
train_nllk: 3601.3331722222442   test_nllk: 2487.134938314903
train_nllk: 3523.415731230749    test_nllk: 2427.5756149454587
train_nllk: 3460.009164087826    test_nllk: 2379.87096819181
train_nllk: 3407.624175572745    test_nllk: 2341.0752846236646
train_nllk: 3363.7871428427316   test_nllk: 2309.117909256505
train_nllk: 3326.6970718953103   test_nllk: 2282.503758051289
train_nllk: 3295.0127919478127   test_nllk: 2260.128534170584
train_nllk: 3267.7161027982897   test_nllk: 2241.160557162045
train_nllk: 3244.020998716411    test_nllk: 2224.9628218565103
train_nllk: 3223.31181821814     test_nllk: 2211.040195021951
train_nllk: 3205.1001037772903   test_nllk: 2199.002794112964
train_nllk: 3188.993885273901    test_nllk: 2188.5400592810297
train_nllk: 3174.6754064683855   test_nllk: 2179.4020570953867
train_nllk: 3161.884708703874    test_nllk: 2171.3857768004564
train_nllk: 3150.407353543851    test_nllk: 2164.3249374293778
train_nllk: 3140.0651190099748   test_nllk: 2158.0823051652633
train_nllk: 3130.7088644433456   test_nllk: 2152.5438326902977
train_nllk: 3122.212998605176    test_nllk: 2147.614139158225
train_nllk: 3114.4711478530357   test_nllk: 2143.2129889819894
train_nllk: 3107.3927328975697   test_nllk: 2139.272523334893
train_nllk: 3100.900240688993    test_nllk: 2135.735064901689
train_nllk: 3094.9270332924334   test_nllk: 2132.551363465884
train_nllk: 3089.41557531545     test_nllk: 2129.6791835690233
train_nllk: 3084.3159902935536   test_nllk: 2127.0821598315956
train_nllk: 3079.5848776237963   test_nllk: 2124.728863347013
train_nllk: 3075.1843373563206   test_nllk: 2122.5920357366663
train_nllk: 3071.0811619308224   test_nllk: 2120.6479572900234
train_nllk: 3067.2461628474102   test_nllk: 2118.875923022198
```

train_nllk: 3063.653607047548 test_nllk: 2117.2578061086215
train_nllk: 3060.2807429947566 test_nllk: 2115.7776924646264
train_nllk: 3057.1074004804964 test_nllk: 2114.4215735606317
train_nllk: 3054.1156513263672 test_nllk: 2113.177087144693
train_nllk: 3051.289520621919 test_nllk: 2112.033297562258
train_nllk: 3048.6147400861028 test_nllk: 2110.9805089509073
train_nllk: 3046.078536688078 test_nllk: 2110.010105844663
train_nllk: 3043.669450899187 test_nllk: 2109.114416722876
train_nllk: 3041.3771799404785 test_nllk: 2108.2865968392825
train_nllk: 3039.192442191128 test_nllk: 2107.520527310772
train_nllk: 3037.1068595726665 test_nllk: 2106.8107279658543
train_nllk: 3035.112855253095 test_nllk: 2106.1522818754493
train_nllk: 3033.203564447925 test_nllk: 2105.540769833276
train_nllk: 3031.3727564509636 test_nllk: 2104.9722133354244
train_nllk: 3029.6147663211295 test_nllk: 2104.443024840829
train_nllk: 3027.924434894666 test_nllk: 2103.949964285987
train_nllk: 3026.2970559940723 test_nllk: 2103.490100986044
train_nllk: 3024.728329873614 test_nllk: 2103.0607801863744
train_nllk: 3023.2143220822372 test_nllk: 2102.659593638936
train_nllk: 3021.751427043117 test_nllk: 2102.284353669854
train_nllk: 3020.3363357487706 test_nllk: 2101.9330702820957
train_nllk: 3018.9660070548593 test_nllk: 2101.6039309022813
train_nllk: 3017.637642127178 test_nllk: 2101.295282435727
train_nllk: 3016.3486616568916 test_nllk: 2101.005615340438
train_nllk: 3015.0966855107213 test_nllk: 2100.7335494703525
train_nllk: 3013.8795145268164 test_nllk: 2100.477821471861
train_nllk: 3012.695114204792 test_nllk: 2100.237273546365
train_nllk: 3011.5416000707582 test_nllk: 2100.010843416265
train_nllk: 3010.4172245259983 test_nllk: 2099.797555352853
train_nllk: 3009.320365011925 test_nllk: 2099.5965121427334
train_nllk: 3008.24951334468 test_nllk: 2099.406887884979
train_nllk: 3007.203266090619 test_nllk: 2099.2279215247363
train_nllk: 3006.1803158695143 test_nllk: 2099.0589110406163
train_nllk: 3005.1794434857793 test_nllk: 2098.899208213315
train_nllk: 3004.1995107997673 test_nllk: 2098.7482139116582
train_nllk: 3003.239454261462 test_nllk: 2098.6053738398814
train_nllk: 3002.298279037806 test_nllk: 2098.4701746965957
train_nllk: 3001.375053672763 test_nllk: 2098.3421407016644
train_nllk: 3000.468905226042 test_nllk: 2098.220830452281
train_nllk: 2999.5790148424676 test_nllk: 2098.1058340739723
train_nllk: 2998.704613709246 test_nllk: 2097.996770636102
train_nllk: 2997.8449793630502 test_nllk: 2097.893285804895
train_nllk: 2996.9994323129513 test_nllk: 2097.7950497099605
train_nllk: 2996.1673329488567 test_nllk: 2097.701755002956
train_nllk: 2995.3480787083117 test_nllk: 2097.613115089339
train_nllk: 2994.5411014773695 test_nllk: 2097.5288625162166
train_nllk: 2993.7458652037467 test_nllk: 2097.448747501108
train_nllk: 2992.961863702712 test_nllk: 2097.372536588046
train_nllk: 2992.188618638152 test_nllk: 2097.3000114188685
train_nllk: 2991.425677663004 test_nllk: 2097.23096760879
train_nllk: 2990.6726127048387 test_nllk: 2097.165213716501
train_nllk: 2989.9290183837634 test_nllk: 2097.102570300008
train_nllk: 2989.194510551083 test_nllk: 2097.0428690503204

```
train_nllk: 2988.4687249382537   test_nllk: 2096.9859519958936
train_nllk: 2987.7513159066875   test_nllk: 2096.9316707714306
train_nllk: 2987.041955289858    test_nllk: 2096.8798859452704
train_nllk: 2986.3403313199456   test_nllk: 2096.8304664001744
train_nllk: 2985.646147632005    test_nllk: 2096.7832887627947
train_nllk: 2984.9591223392695   test_nllk: 2096.738236877591
train_nllk: 2984.278987173803    test_nllk: 2096.695201321344
train_nllk: 2983.6054866872246   test_nllk: 2096.6540789547835
train_nllk: 2982.938377506702    test_nllk: 2096.6147725081814
train_nllk: 2982.277427641863    test_nllk: 2096.5771901980497
train_nllk: 2981.6224158386135   test_nllk: 2096.541245372326
train_nllk: 2980.9731309762483   test_nllk: 2096.506856181716
```





Variant 3 Model Summary and Sanity Check

Because of the imbalanced data, even though the baseline model predicting all shots to not goal had a very high accuracy(89%), it is not meaningful. Therefore, in this variant, we decided to change the measure of success to the accuracy of correctly predicting goals.

There are four outcomes:

- (0, 0): target = 0 and predict = 0
- (0, 1): target = 0 and predict = 1
- (1, 0): target = 1 and predict = 0
- (1, 1): target = 1 and predict = 1

The original accuracy is defined as:

$$acc = \frac{\#(0,0) + \#(1,1)}{\#(0,0) + \#(0,1) + \#(1,0) + \#(1,1)}$$

Now, the adjusted accuracy is defined as:

$$acc_{adj} = \frac{\#(1,1)}{\#(0,1) + \#(1,1)}$$

Because the baseline model predicts all shots to not goal, even though acc is high, the adjusted accuracy $acc_{adj} = 0$.

```
Training Accuracy: 0.5456204379562044 0.839043540328337
Test Accuracy: 0.5165441176470589 0.8591931111465476
```

	$TrainingAccuracy$	$TestingAccuracy$	$AdjustedTrainingAccuracy$	$Adjusted$
$BaselineModel$	89.30707	89.43854	0	
$Variant3$	83.90435	85.91931	54.56204	5

From the table above, both the adjusted training accuracy and the adjusted testing accuracy is higher than the baseline model. However, the overall accuracy is slightly lower than the baseline model, because although the varaint avoids more type 2 error than the baseline model, it also causes some type 1 error.

Therefore, this variant model works better for shots that successfully goal, but works worse for shots that not goal.

Result and Limitation

- This variant model is able to predict whether a `player` could goal with a given `bodypart` and in a given `situation`.
- The training and testing set is randomly splitted as in the baseline model.
- The adjusted accuracy of both the training and testing sets are 50% higher than the baseline model. Because we know only 11% of shots successfully goal, the adjusted accuracy of a random guess of goal is around 11. Therefore, the outcome of the varaint is also around 40% better than a random guess. Even though the overall accuracy of simply predicting all shots to not goal is slightly higher than this model, it is not meaningful to do such a prediction.
- If the model predicts a shot to goal, the accuracy of the prediction is more than 50%. Therefore, if the coach wants to ensure a goal in some given situations, he could make decisions based on this model. For instance, when the coach is deciding the strategy of a free-kick, he could choose a player to shot with the body part such that the model predicts that it would goal. In this situation, more than 50% of chances the free-kick could goal and earn a point for the team.
- However, because the model depends on player latents and the player latents are trained based on a player's previous shots data, many new players with few data avaliable would hurt the accuracy.