Predicting Whether Home Team Will Win or Not Using Random Forest & Gradient Boosting

-Within EPL Football League-

Department of DataScience
2020047029
JunYeong Ahn

I. Introduction

I dare say you know 'Messi', 'Manchester United' and 'Real Madrid'! In 2020, SPORTS SHOW(sportsshow.net) picked soccer as the most popular sports in the world based on 15 criteria. And European leagues, especially EPL(England Premier League) is said to be the most exciting since EPL has many star players and is well-balanced in terms of performances of each team. Of course, guessing which team is going to win the match is always a hot issue between fans – regardless of gamble or simple bet. This paper aims to predict either home team or away team will win with 'European Soccer Database' in Kaggle, using Random Forest and Gradient Boosting. I also suggest some other possible analysis related to this work here. One thing readers should notice is that 'Draw' is merged into 'Lose' to make the problem binary classification – win, or not.

One thing I want to talk about this paper is that I have no choice but to write with some omissions(skips), assuming that the reader has some basic understanding of this data since the data is way bulky and hard to be well-explained due to its silly form. Please refer to the attached codes for more details.

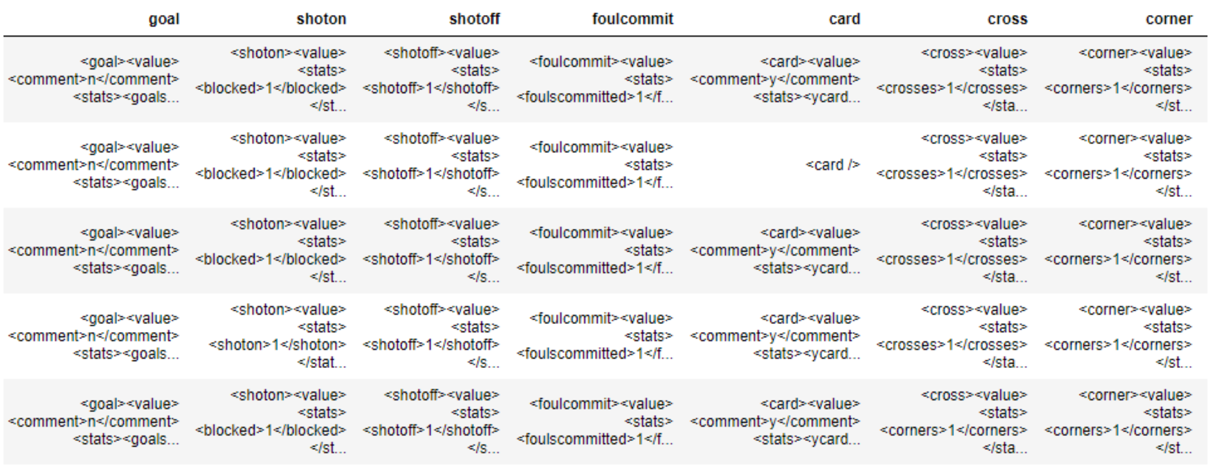
II. Data

The data consists of seven tables – *County, League, Match, Player, Player_Attributes, Team* and *Team_Attributes* – which are somewhat related to each other, with +200k rows and 199 columns in total. Based on tables, we have three broad choices; 1) inspecting mainly on team performances; 2) inspecting mainly on player performances; 3) inspecting mainly on matches. To predict 'Win or Not', 3) is the best, however, I'll leave the possibility of combining them by trying to add a unique column associated with 2) not to reproduce dull studies. Anyway, I should extract EPL data from *Match* table and it may contain 3040 rows – (20*19)/2 (# of combinations between 20 EPL teams) * 2 (Home/Away) * 8 (total 8 years observed).

After filtering EPL data from *Match* table using 'league_id' in *League* table, I face the two noticeable problems with loaded data;

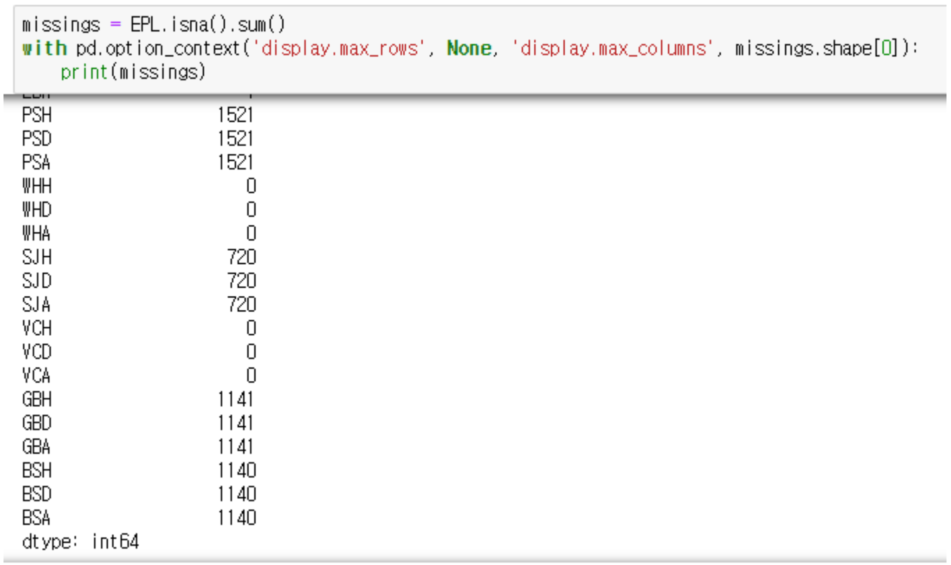
- Eight columns ('goal' ~ 'possession') are lost after loading the data(**FIGURE II.1**).
- Specific columns about betting odds have too many missing values(**FIGURE II.2**).

In **FIGURE II.2**, all columns ending with H are the odds for the Home team in the match. Likewise, columns ending with D are odds for Draw, & A are for Away team. Most of missing values are within these betting odds columns.



goal	shoton	shotoff	foulcommit	card	cross	corner
<goal><value> <comment>n</comment> <stats><goals...	<shoton><value> <stats> <blocked>1</blocked> </st...	<shotoff><value> <stats> <shotoff>1</shotoff> </s...	<foulcommit><value> <stats> <foulscommitted>1</f...	<card><value> <comment>y</comment> <stats><ycard...	<cross><value> <stats> <crosses>1</crosses> </sta...	<corner><value> <stats> <corners>1</corners> </st...
<goal><value> <comment>n</comment> <stats><goals...	<shoton><value> <stats> <blocked>1</blocked> </st...	<shotoff><value> <stats> <shotoff>1</shotoff> </s...	<foulcommit><value> <stats> <foulscommitted>1</f...	<card />	<cross><value> <stats> <crosses>1</crosses> </sta...	<corner><value> <stats> <corners>1</corners> </st...
<goal><value> <comment>n</comment> <stats><goals...	<shoton><value> <stats> <blocked>1</blocked> </st...	<shotoff><value> <stats> <shotoff>1</shotoff> </s...	<foulcommit><value> <stats> <foulscommitted>1</f...	<card><value> <comment>y</comment> <stats><ycard...	<cross><value> <stats> <crosses>1</crosses> </sta...	<corner><value> <stats> <corners>1</corners> </st...
<goal><value> <comment>n</comment> <stats><goals...	<shoton><value> <stats> <shoton>1</shoton> </stat...	<shotoff><value> <stats> <shotoff>1</shotoff> </s...	<foulcommit><value> <stats> <foulscommitted>1</f...	<card><value> <comment>y</comment> <stats><ycard...	<cross><value> <stats> <crosses>1</crosses> </sta...	<corner><value> <stats> <corners>1</corners> </st...
<goal><value> <comment>n</comment> <stats><goals...	<shoton><value> <stats> <blocked>1</blocked> </st...	<shotoff><value> <stats> <shotoff>1</shotoff> </s...	<foulcommit><value> <stats> <foulscommitted>1</f...	<card><value> <comment>y</comment> <stats><ycard...	<cross><value> <stats> <corners>1</corners> </sta...	<corner><value> <stats> <corners>1</corners> </st...

FIGURE II.1. – After some manipulations on NaN values of eight columns, it turns out that they are actually written in XML format containing multiple informations at one dataframe block, not in single number. This may require XML parsing to be properly displayed in our data.



```
missings = EPL.isna().sum()
with pd.option_context('display.max_rows', None, 'display.max_columns', missings.shape[0]):
    print(missings)
```

```
PSH             1521
PSD             1521
PSA             1521
WHH                0
WHD                0
WHA                0
SJH              720
SJD              720
SJA              720
VCH                0
VCD                0
VCA                0
GBH             1141
GBD             1141
GBA             1141
BSH             1140
BSD             1140
BSA             1140
dtype: int64
```

FIGURE II.2. – ――H, ――D and ――A are each the betting odds of Home team win/ Draw / Away team win in certain website. Some betting odds including PSH, GBH and many else have almost a thousand of missing values out of 3040 observations, which is too large to get rid of or fill in with average values.

My solution against them is;

- Trying to extract at least a high level overview of all these data using XML parser.
- Drop them all for maintaining data reliability (plus, many other betting odds still survived and enough to represent the loss due to their similarity.)

Also I dropped 'home_player_X1'~'away_player_Y11'(coordinates information of each player in the field), which is related to team formation but not used in my Home vs. Away prediction.

Then 'home_outcome' target column is created, whose classes are binary; 'Win' and 'Not Win' – the subject is home team - based on 'home_team_goal' and 'away_team_goal' columns since I cannot classify the target whose number of classes is three – only binary class is possible for me to classify. When the team wins, they get 3 score / Draw : 1 / Lose : 0. This is why I merged 'Draw' into 'Lose', not into 'Win' – it is closer to 'Lose' in terms of score, although it is not considered in this dataset.

Before wrapping up pre-processing, I tried to add a column named '*starplayer*' to see if there could be a specific person on the team who has a big impact on 'Win or Not' – for this, I made a table ('Top20Rank' in codes) which contains top 20 ranks players in terms of overall ratings in each year. I expected that the existence of such influential person might be associated with the result of matches since we empirically know that the ace who has a great leadership affects games. 'Starplayer' is categorical variable that consists of three classes – 'Home'(at least one person in home team) , 'Away'(at least one person in away team) and 'No'(no one in both teams). This could become more sophisticated to get improved with some further tasks like dividing into more classes or tuning N in top N ranks players(here 20, considering around two people in each position).

Irrelevant variables such as 'country_id' and 'league_id' are also dropped, then categorical values, including 'starplayer', are diverted into dummy values so that they can be taken in consideration during training the models. As a result, I have 40 columns left and now draw a heatmap to identify the correlations between features. So far, I was reluctant to draw any plots since there are too many features but now just am trying so that I can come up with some outlines for further pre-processing.

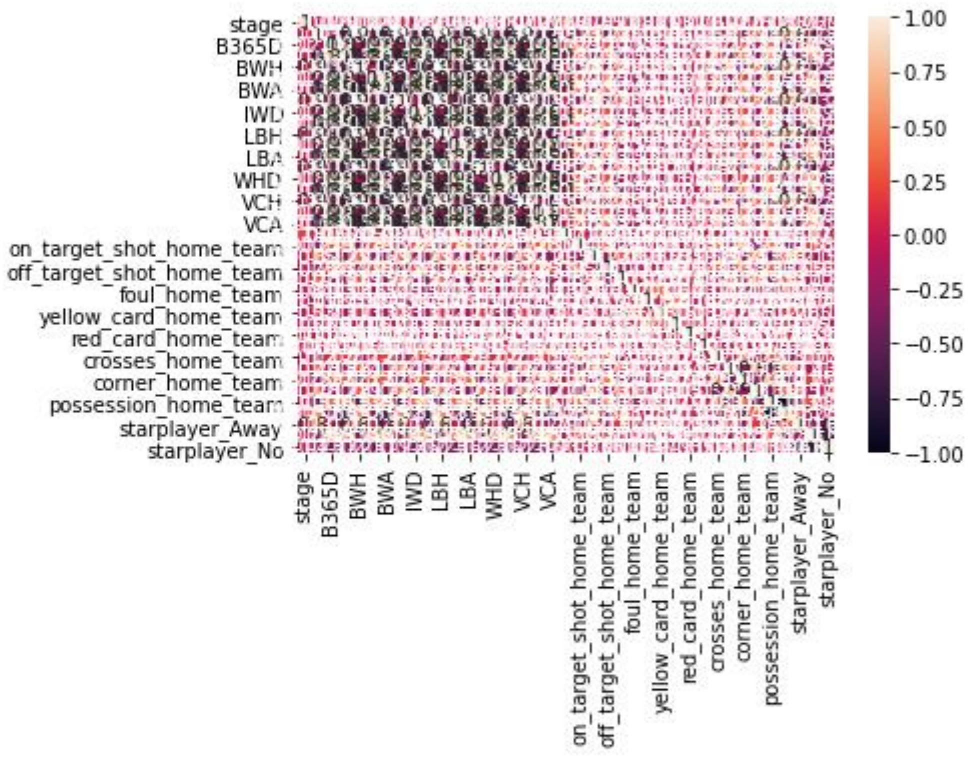
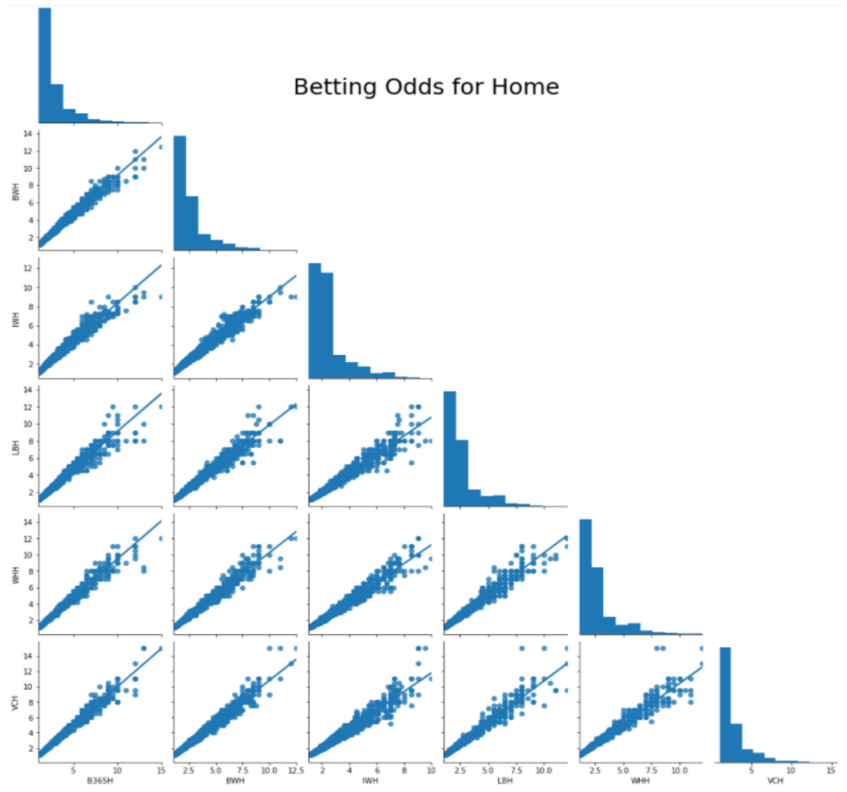
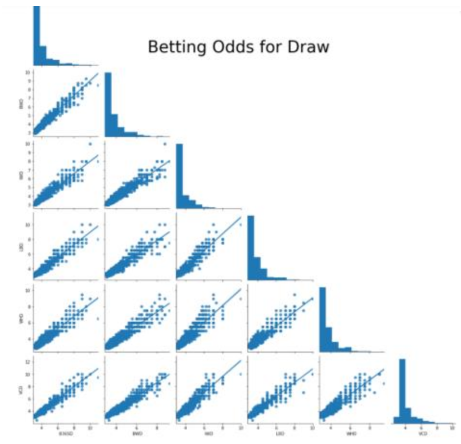


FIGURE II.3. - Although graphics are crashed a bit due to way large # of variables(40), it is clear that there is *a black zone* on the top left which means strong negative relationships are between some columns – seeing the heatmap, maybe betting odds are problematic.

Although there's no problem such as multicollinearity since random forest and gradient boosting will automatically "drop" one column among redundant ones (which have strong correlation) at each split, still there is no reason to overlook such state – dealing with this before modeling will help readers to easily interpret and understand the data later on.





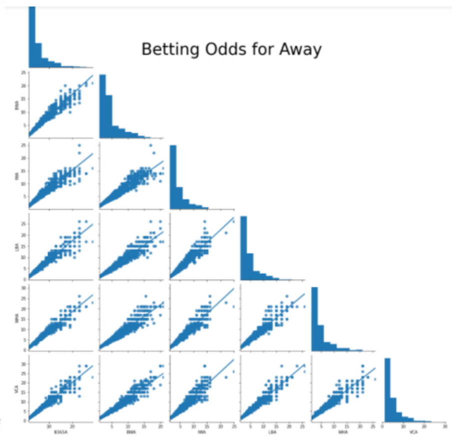


FIGURE II.4, 5, 6. - Seeing the correlation plot for each type of betting odds('H'ome/'D'raw/'A'way), there are almost linear relationships between betting odds in each website, which means we are likely to get more parsimonious models later without any score loss if we select representative one website.

Seeing **FIGURE II.4, 5, 6**, here I picked 'B365H', 'B365D' and 'B365A', for they are the biggest and most reliable website. Thus, the final EPL data, which I named 'EPL_final' in code, contains 25 columns:

```
'home_outcome'(target variable), 'stage', 'B365H', 'B365D', 'B365A',
'year', 'on_target_shot_home_team', 'on_target_shot_away_team',
      'off_target_shot_home_team', 'off_target_shot_away_team',
      'foul_home_team', 'foul_away_team', 'yellow_card_home_team',
      'yellow_card_away_team', 'red_card_home_team', 'red_card_away_team',
'crosses_home_team', 'crosses_away_team', 'corner_home_team',
'corner_away_team', 'possession_home_team', 'possession_away_team',
'starplayer_Away', 'starplayer_Home', 'starplayer_No'
```

One possible concern that can arise here would be this; "Is it okay to include betting odds as final features, although they are elaborately calculated based on the possibility that each team wins?" Since I don't know whether this study will be used in real gambling or in just practices of someone, I'll include it first then show another result without those odds features later on.

III. Modeling

Two different tree-based ensemble models are used to predict 'Win or Not' – Random Forest & Gradient Boosting. Like random forests gradient boosting is a set of decision trees. The two main differences are:

1. **How trees are built:** random forests build each tree independently while gradient boosting builds one tree at a time. This additive model (ensemble) works in a forward stage-wise manner, introducing a weak learner to improve the shortcomings of existing weak learners.
2. **Combining results**: random forests combine results at the end of the process (by averaging or "majority rules") while gradient boosting combines results along the way.

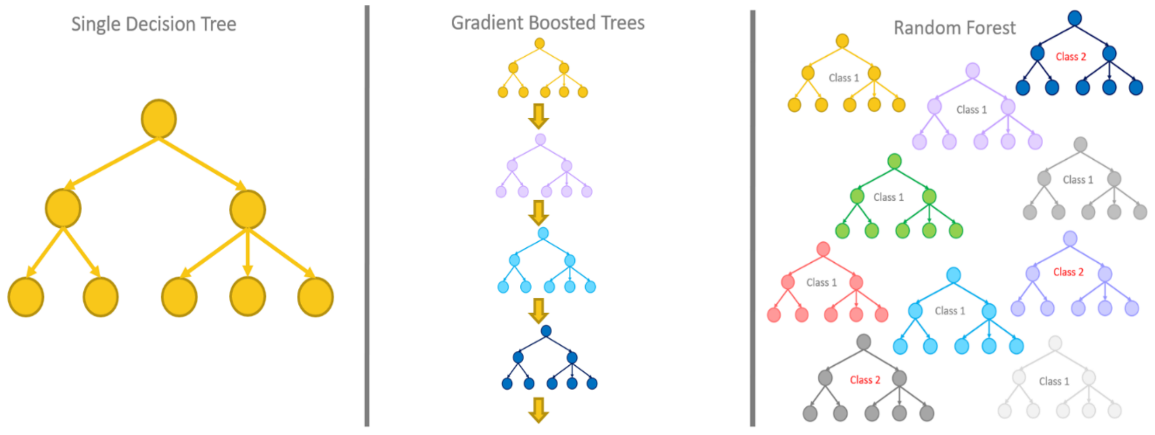


FIGURE III.1 – Gradient boosting sequentially produces new regressor learning from the previous one, whereas random forest classifier produces multiple weak classifiers (decision trees) at once. Both are almost always better than a single decision tree in that they are robust to train data.

As evaluation metrics, Accuracy will be calculated and AUC as well not to be swayed by a change in single performance score – the highest X score doesn't always mean the highest Y score. In this study, *classification accuracy(Acc)* is the ratio of correct predictions to total predictions made. *The Area Under the Curve (AUC)* is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes – how true positive rate and false positive rate trade off. Both are for classification problem and will help me to objectively identify when (and WHY) the model gets improved, stays still and gets worse.

In each model, there are two broad steps; 1) Inspecting on the change in scores (+runtime) as certain parameter changes; 2) Inspecting on the change in scores (+runtime) when feature selection is conducted with certain method. All the results are those of 5-fold cross validation(CV), since CV is regarded as 'gold standard' among evaluating models since it has many strengths in general compared to traditional train/test splitting: 1) CV generalizes better since it uses all the data in evaluating 2) It brings higher accuracy since it uses all the data in training 3) ,, We have 3030 observations here so 5 folds would be sufficient.

IV. Results & Analysis

- Random Forest(clf = RandomForestClassifier(n_estimators = N(this may be tuned), max_depth = None, min_samples_split = 3, criterion = 'entropy', random_state = rand_st)

No. of trees	Accuracy	AUC	5Fold CV Runtime
10	0.67 (+/- 0.04)	0.72 (+/- 0.06)	0.2234036922454834
20	0.69 (+/- 0.05)	0.75 (+/- 0.05)	0.43694591522216797
50	0.70 (+/- 0.05)	0.76 (+/- 0.05)	1.0492475032806396
200	0.71 (+/- 0.04)	0.77 (+/- 0.04)	4.153926372528076
500	0.71 (+/- 0.03)	0.77 (+/- 0.04)	10.414162158966064
1000	0.71 (+/- 0.04)	0.77 (+/- 0.04)	20.819379806518555
2000	0.71 (+/- 0.04)	0.77 (+/- 0.04)	41.26173543930054

** Figures behind '+/-' are 2*standard deviation, which means they are expression of error range that guarantees 95% confidence.*

TABLE IV.1. – With No. of trees that Random Forest model produces being increased from 10 to 2000, Accuracy also increased from 0.67 to 0.71 and did AUC from 0.72 to 0.77 as well. Also error range decreased when average values get better, which means it guarantees obvious improvement in scores. However, it seems scores converge after certain boundary(=200) of 'No. of trees'.

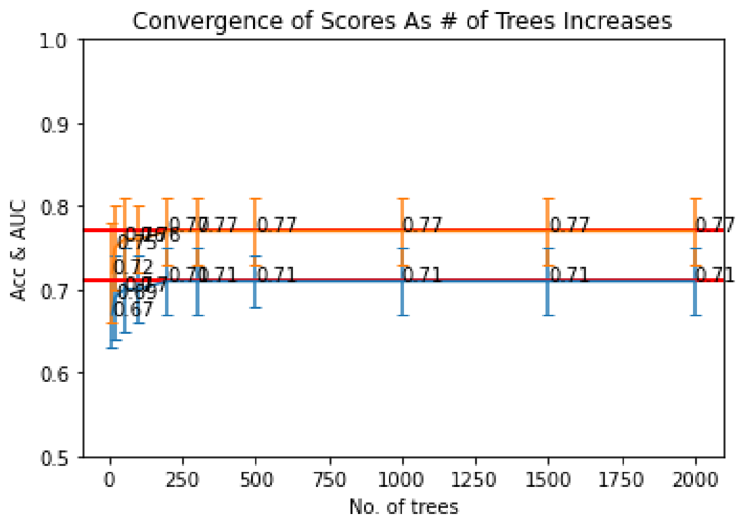


FIGURE IV.1 – With a subtle change of error ranges, both scores converge after around No. of trees = 200.

Also, runtime increased with in No. of trees direct proportion because runtimes are associated with the number of calculations within folds. Folds are fixed to 5, so No. of trees determine runtimes;
10 : 20 : 50 : 100 : 200 : 300 : 500 : 1000 : 1500 : 2000 ≒ 0.2(when No. of trees = 10) : 0.43(=20) : 1.04(=50) : 2.12(=100) : 4.15(=200) : 6.28(=300) : 10.41 (=500) : 20.81 (=1000) : 31.43 (=1500) : 41.26 (=2000). This means that scores are the highest for the time when No. of trees is equal to 200.

- Random Forest (No. of trees : 200)

	Accuracy	**AUC**	**CV Runtime**	**Features**
No Feature Selection	0.71 (+/- 0.04)	0.77 (+/- 0.04)	4.353926372528076	'stage', 'B365H', 'B365D', 'B365A', 'year', 'on_target_shot_home_team', 'on_target_shot_away_team', 'off_target_shot_home_team', 'off_target_shot_away_team', 'foul_home_team', 'foul_away_team', 'yellow_card_home_team', 'yellow_card_away_team', 'red_card_home_team', 'red_card_away_team', 'crosses_home_team', 'crosses_away_team', 'corner_home_team', 'corner_away_team', 'possession_home_team', 'possession_away_team', 'starplayer_Away', 'starplayer_Home', 'starplayer_No'
Wrapper-Based Feature Selection	0.69 (+/- 0.04)	0.75 (+/- 0.04)	3.064138650894165	'stage', 'B365H', 'B365D', 'B365A', 'on_target_shot_home_team', 'off_target_shot_home_team', 'off_target_shot_away_team', 'foul_home_team', 'foul_away_team', 'crosses_home_team', 'crosses_away_team', 'corner_home_team', 'possession_home_team', 'possession_away_team'

Feature-Importance Based Feature Selection	0.69 (+/- 0.04)	0.75 (+/- 0.04)	3.199074602127075	'stage', 'B365H', 'B365D', 'B365A', 'on_target_shot_home_team', 'off_target_shot_home_team', 'off_target_shot_away_team', 'foul_home_team', 'foul_away_team', 'crosses_home_team', 'crosses_away_team', 'corner_home_team', 'possession_home_team', 'possession_away_team'

TABLE IV.2. – After two different feature selection methods – Wrapper−Based & Feature−Importance−Based − , the same 14 features have survived in each method among original 24 features. Though Acc and AUC dropped by 0.02, the number of features became almost half and runtimes got shorter, which means we can comparative parsimonious model with only subtle loss of scores.

I also conducted normalization against all the features but it didn't make any changes in scores. This may stem from the fact that soccer has time limit of 90 mins and two comparative teams (although sometimes one side is overwhelming) play, which makes it hard to get something problematic like outliers. So far, the optimized condition is 'No. of trees = 200' & 'Doing Feature Selection' with 0.69 of Acc, 0.75 of AUC and 14 variables.

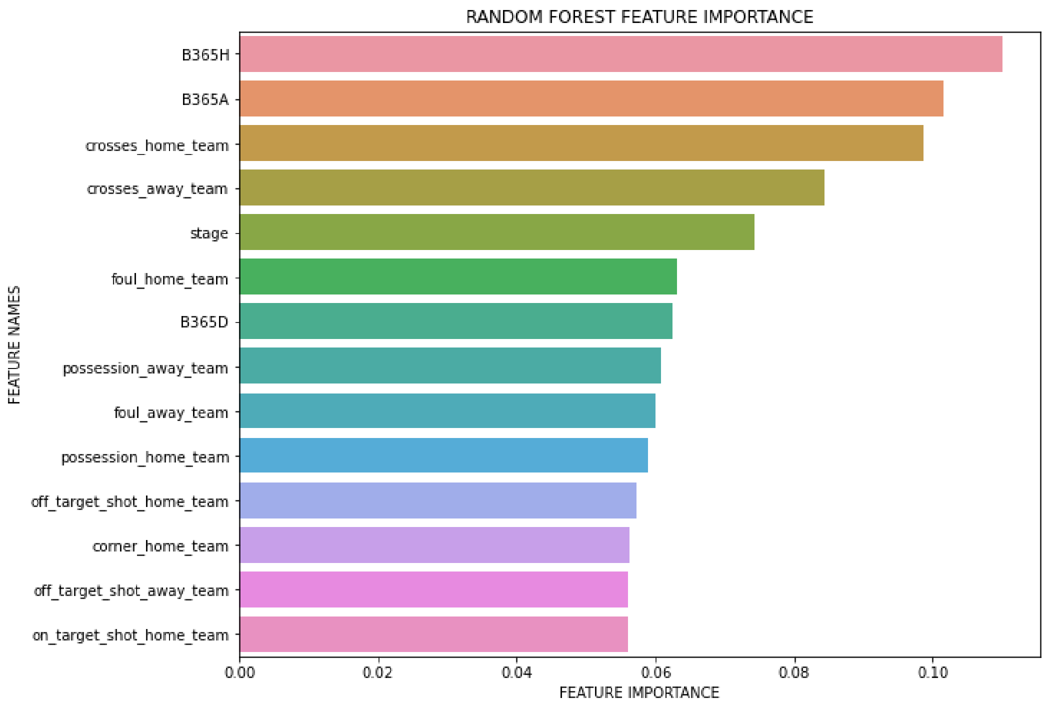


FIGURE IV.2 – As you concerned in the last part of **II. Data**, the prediction greatly relies on betting odds. If you are going to buy some lotto, this model will be of no use.

If I delete betting odds columns and run the model again with others being the same, Acc and AUC become 0.63(+/-0.05) and 0.68(+/-0.07), which are greatly dropped figures compared to those of the previous optimized condition. 14 included features are shown again **in FIGURE IV.3** displaying feature importance as well.

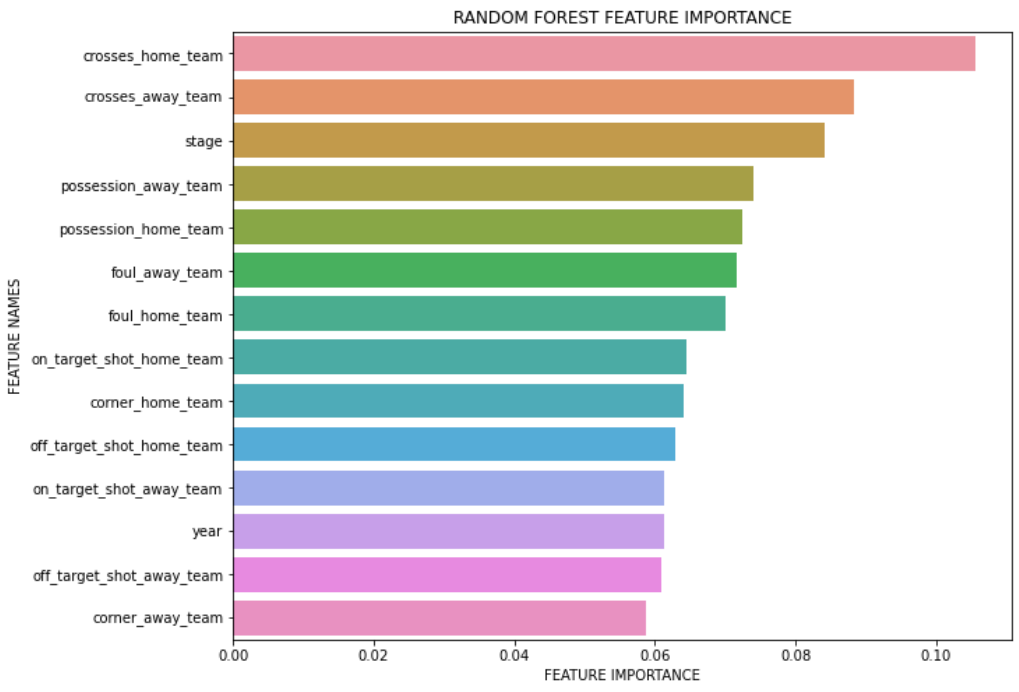


FIGURE IV.3 – Feature importance without betting odds. Crosses of both teams are turned out to be important in predicting whether home team win or not.

- Gradient Boosting(clf=GradientBoostingClassifier(n_estimators=100, loss='deviance', learning_rate=0.1, max_depth=M(this may be tuned), min_samples_split=3, random_state=rand_st)

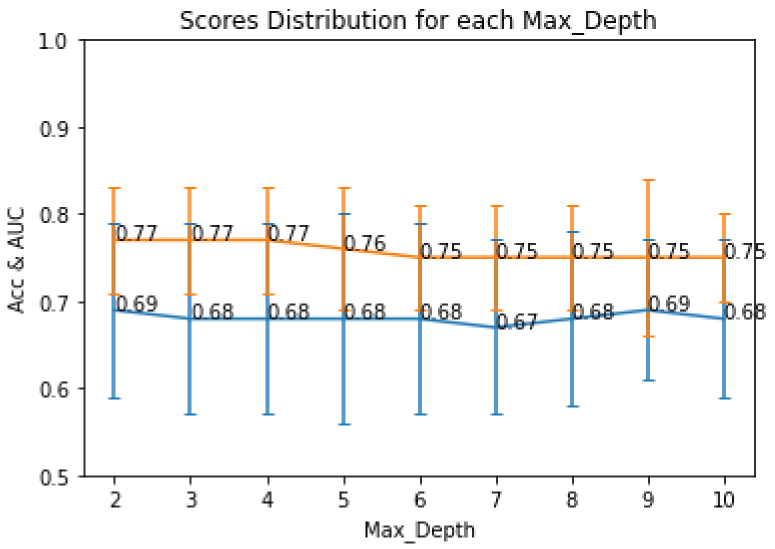


FIGURE IV.4 – 'Max−Depth' is a maximum depth of the individual regression estimators, which limits No. of nodes in tree to affect the scores. Seeing the graph, there is no

noticeable trends (such as the increase of scores as max−depth gets larger) unlike **FIGURE IV.1.** With the highest scores and the shortest error range, the Gradient Boosting model performed the best when max−depth is equal to 2 – this time I didn't report error ranges since it was lowest when max−depth is equal to 2 where scores become the highest, which means the model is undoubtedly optimized when max−depth is equal to 2.

Also runtimes naturally increased as Max_Depth increases since increase of max_depth means each tree becomes larger in size(No. of nodes), which means it takes longer to draw it. So let me fix the max_depth parameter equal to 2 from now on.

- Gradient Boosting(max_depth : 2)

	Accuracy	**AUC**	**CV Runtime**	**Features**
No Feature Selection	0.69 (+/- 0.10	0.77 (+/- 0.06)	1.4860551357269287	'stage', 'B365H', 'B365D', 'B365A', 'year', 'on_target_shot_home_team', 'on_target_shot_away_team', 'off_target_shot_home_team', 'off_target_shot_away_team', 'foul_home_team', 'foul_away_team', 'yellow_card_home_team', 'yellow_card_away_team', 'red_card_home_team', 'red_card_away_team', 'crosses_home_team', 'crosses_away_team', 'corner_home_team', 'corner_away_team', 'possession_home_team', 'possession_away_team', 'starplayer_Away', 'starplayer_Home', 'starplayer_No'
Wrapper-Based Feature Selection	0.72 (+/- 0.04)	0.79 (+/- 0.05)	0.5834403038024902	'B365H', 'B365A', 'red_card_home_team', 'red_card_away_team', 'crosses_home_team', 'crosses_away_team'

Wrapper-Based Feature Selection (without betting odds)	0.65 (+/- 0.07)	0.72 (+/- 0.08)	0.5934135913848877	'red_card_home_team', 'red_card_away_team', 'crosses_home_team', 'crosses_away_team', 'possession_home_team', 'possession_away_team' 'starplayer_Away', 'starplayer_Home'

TABLE IV.3. – After Wrapper–Based feature selection, 6 features have survived among original 24 features. Interestingly, not only did I get much more parsimonious model but also got improved scores of 0.03 and 0.02 respectively for Acc and AUC even with narrower error ranges after feature selection. If I drop betting odds columns, however, scores decreased by great figures as expected.

What is interesting in **TABLE IV.3.** is *that 'starplayer' column I manually added to data were selected after Wrapper-Based Feature Selection without betting odds (refer to* **FIGURE IV.6***).* I will deal with this in **V. Conclusion**.

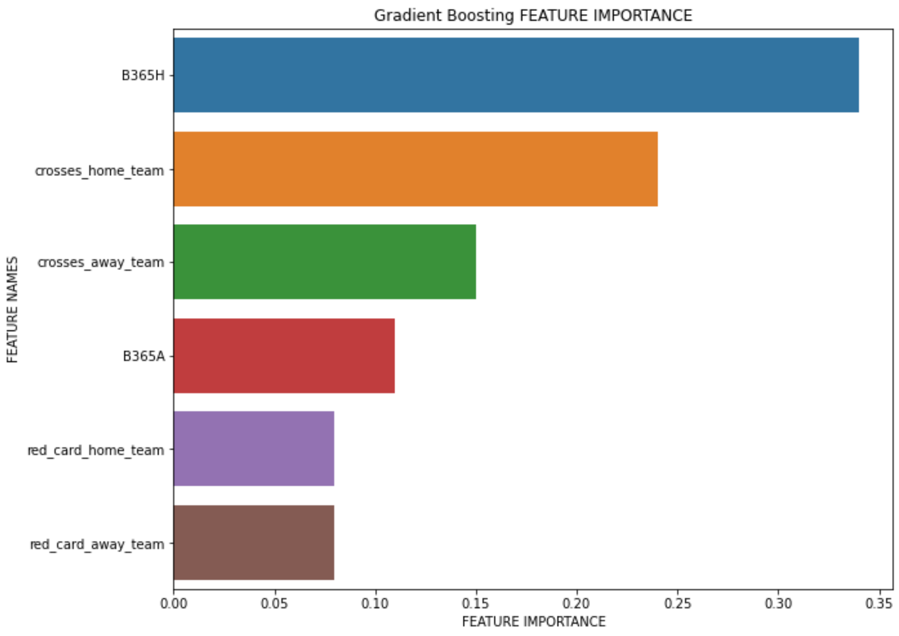


FIGURE IV.5 – Feature Importance plot for Gradient Boosting with betting odds. As 'B365H', one of betting odds variables, is regarded as the most important feature to predict target class, someone might find this useless – if the study stops here, this is not complete prediction in that the model steals another(here, the website Bet365)'s point of view.

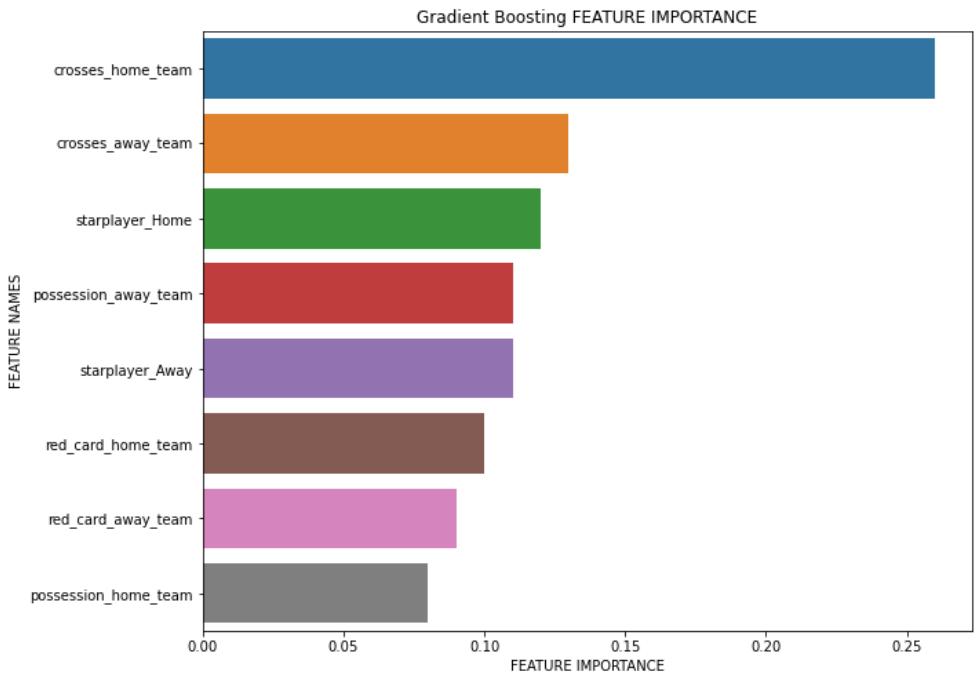


FIGURE IV.6 – Feature Importance plot for Gradient Boosting *without* betting odds. 'Starplayer—Home' and 'Starplayer—Away' features are newly regarded as important ones.

V. Conclusion

*All I call 'improvement' and 'degradation' in this section is considering error ranges as well though there is no specific mention about them.

EPL, one of the hottest football leagues, tends to be a target of wagers of guessing 'which team is going to win' between friends, fans and any other groups. This paper tried to make accurate prediction of winning team based on collected data about players, teams and matches of each league (European Soccer Database in Kaggle) using two tree-based ensemble models – Random Forest(RF) and Gradient Boosting(GB). After pre-processing the data, the initial models of RF and GB showed <u>0.71 of Acc & 0.77 of AUC</u> and <u>0.69 of Acc & 0.77 of AUC</u> – averaged figures through 5-fold CV - when only the number of trees and the maximum depth are tuned respectively without other manipulations. *RF* lost its scores a bit after feature selection (Acc : 0.71 -> 0.69 , AUC: 0.77 -> 0.75) but became much more parsimonious with the number of its features being almost half (24 -> 14). When I excluded betting odds features considering some concerns about using them in prediction, however, the performance of RF was greatly degraded (Acc : 0.69 -> 0.63 , AUC: 0.75 -> 0.68). Thus, if the readers are going to use the model in study, they should be aware of this degradation.

What was more interesting is the performance change of GB with this EPL data. Scores of GB got improved (Acc : 0.69 -> 0.72, AUC : 0.77 -> 0.79) with the number of features being almost a quarter (24 -> 6), which means much more accurate parsimonious model. Still, the performance was degraded (Acc : 0.72 -> 0.65, AUC : 0.79 -> 0.72) without betting odds. Therefore, using GB is recommended to predict the winning team in terms of scores and simplicity in real-world (without betting odds).

Both models picked '<u>crosses</u>' as the most crucial feature (in **FIGURE IV.3, 6.**) for predicting whether home team will win or not. Ranks for other features are different a bit but 'possessions' (ball possessing rate of each team) also seems to be important. 'Starplayer' features are regarded as important ones in GB, which implies there could be more deep relationship between the existence of certain person (= leadership) and 'home_outcome' (target class).

The study figured out that tree-based ensemble models (especially Gradient Boosting performed well) can quite accurately predict winning team using some simple features including 'crosses' and 'possessions' with the classification accuracy of around +0.65 on average. Also the study indicates that using betting odds to predict target variable is very helpful, but we should carefully use it according to the purpose of using this study.

VI. Future Work

Sadly, I've merged 'draw's into 'lose's because I cannot classify multiclass (more than 2) variables confined within the fence of simple classification. Much more accurate and practical study could be conducted if the target classes remains win, draw and lose. Also, more sophisticated hyperparameter tuning for each model will lead to improvement in evaluation metrics (Acc & AUC), which means it is more likely for models to succeed in prediction.

We also found that there is a chance a single (or many) player affects the result of matches through introducing 'starplayer' column - I identified whether both teams of each match have top ranks 20 players in starting lineup, but the effect of the column will change if the criterion ranks are calculated changes or the number of top ranks players considered changes from 20 to N.

One thing more I can do based on this study is to create new column associated with team formation through utilizing some coordinates information for each player – columns 'home_player_X1'~'away_player_Y11' which I dropped at the last part of 2nd page of this paper. More profound research will make these possible and if really done I think we can expect the classification accuracy of +0.7, even with three target classes (Win/Draw/Lose) .