PREDICTING THE OUTCOMHASE OF 2020 ENGLISH PREMIER LEAGUE (EPL) FOOTBALL MATCHES

Group Name: `Group D`
Department of Computer Science
University College London
London, WC1E 6BT

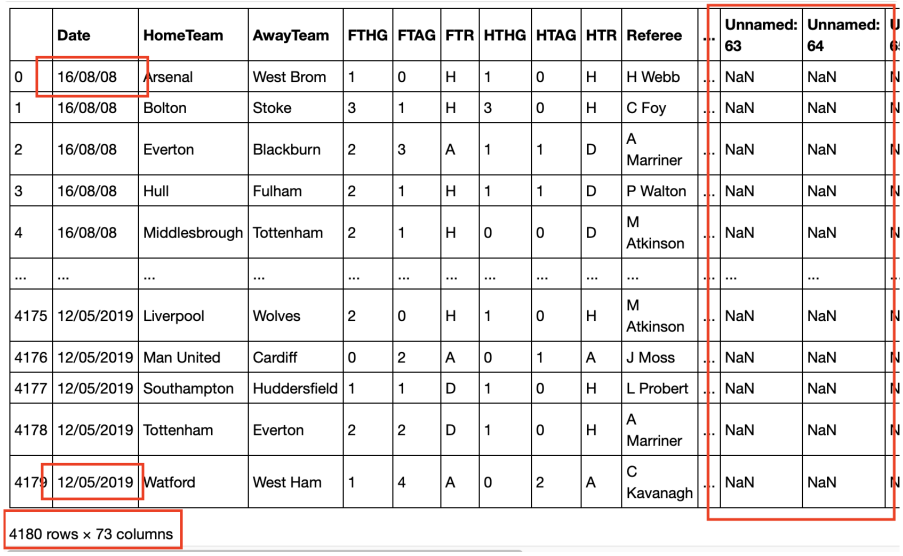
January 6, 2020

1 Introduction

2 Data Transformation & Exploration

At first sight, we found that:

- The shape of the data frame is 4180 rows x 73 columns, but some columns are empty and unnamed.
- There are two different date formats, "%d%m%y" and "%d%m%Y".
- The involved data is from 2008-08-16 to 2019-05-12 (i.e. totally 11 seasons).



	Date	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	Referee	...	Unnamed: 63	Unnamed: 64	U
0	16/08/08	Arsenal	West Brom	1	0	H	1	0	H	H Webb	...	NaN	NaN	N
1	16/08/08	Bolton	Stoke	3	1	H	3	0	H	C Foy	...	NaN	NaN	N
2	16/08/08	Everton	Blackburn	2	3	A	1	1	D	A Marriner	...	NaN	NaN	N
3	16/08/08	Hull	Fulham	2	1	H	1	1	D	P Walton	...	NaN	NaN	N
4	16/08/08	Middlesbrough	Tottenham	2	1	H	0	0	D	M Atkinson	...	NaN	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4175	12/05/2019	Liverpool	Wolves	2	0	H	1	0	H	M Atkinson	...	NaN	NaN	N
4176	12/05/2019	Man United	Cardiff	0	2	A	0	1	A	J Moss	...	NaN	NaN	N
4177	12/05/2019	Southampton	Huddersfield	1	1	D	1	0	H	L Probert	...	NaN	NaN	N
4178	12/05/2019	Tottenham	Everton	2	2	D	1	0	H	A Marriner	...	NaN	NaN	N
4179	12/05/2019	Watford	West Ham	1	4	A	0	2	A	C Kavanagh	...	NaN	NaN	N

4180 rows × 73 columns

Fig. 1. First sight of training data

2.1 Data Cleaning

After we dropped the unnamed columns, the number reduced to 22.

We verified that there is no row containing invalid values (i.e., None, NaN, infinite or overflowed number), so we don't need to drop any rows. The size remains 4180.

We then unified the date formats, converting into "%Y-%m-%d" for later exploration and transformation.

2.2 Initial Data Exploration

2.2.1 Number of matches per season

The full set is of huge amount. To help learn the data, we separated rows by date from August to May (i.e., one season) to check how many matches there are per season.

```
2008 [380 rows x 22 columns]
2009 [380 rows x 22 columns]
2010 [380 rows x 22 columns]
2011 [380 rows x 22 columns]
2012 [380 rows x 22 columns]
2013 [380 rows x 22 columns]
2014 [380 rows x 22 columns]
2015 [380 rows x 22 columns]
2016 [380 rows x 22 columns]
2017 [380 rows x 22 columns]
2018 [380 rows x 22 columns]
```

Fig. 2. Number of matches per season

We found that the number of matches each season stays constant (380).

2.2.2 Relationship between attributes

We plotted a Pearson Correlation Heatmap (Fig. 3) to see the top 10 features related to the match result (FTR).

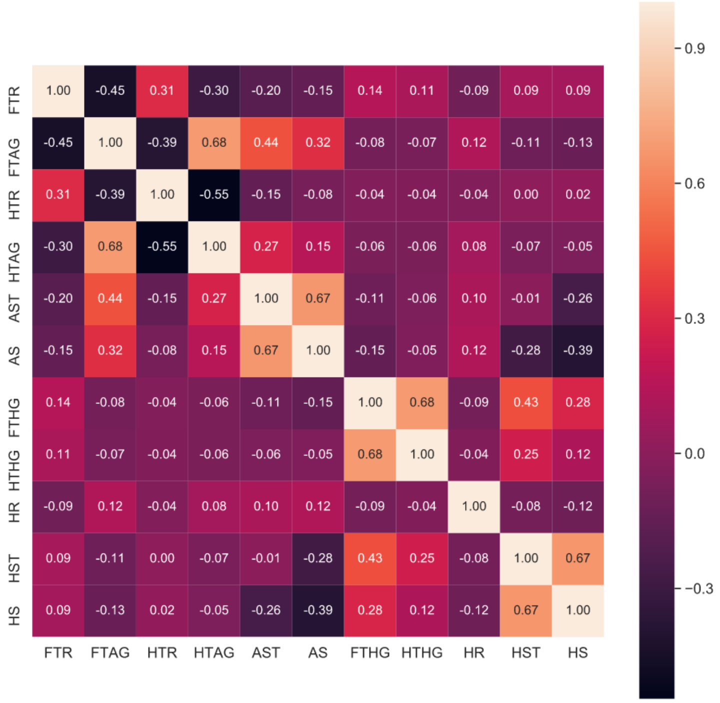


Fig. 3. The top 10 features related to FTR

As shown in the graph, the top 10 features are:

HTR, FTHG, HTHG, HST, HS, HR, AS, AST, HTAG, FTAG,

ordered from the greatest to least.

It is notable that the goal scored at full time (FTHG, FTAG) & goal scored at half time (HTHG, HTAG) and the total number of shots on goal (HS, AS) & that on target (HST, AST) are the two pairs of data which are highly correlated (> 0.65).

2.3 Feature Construction

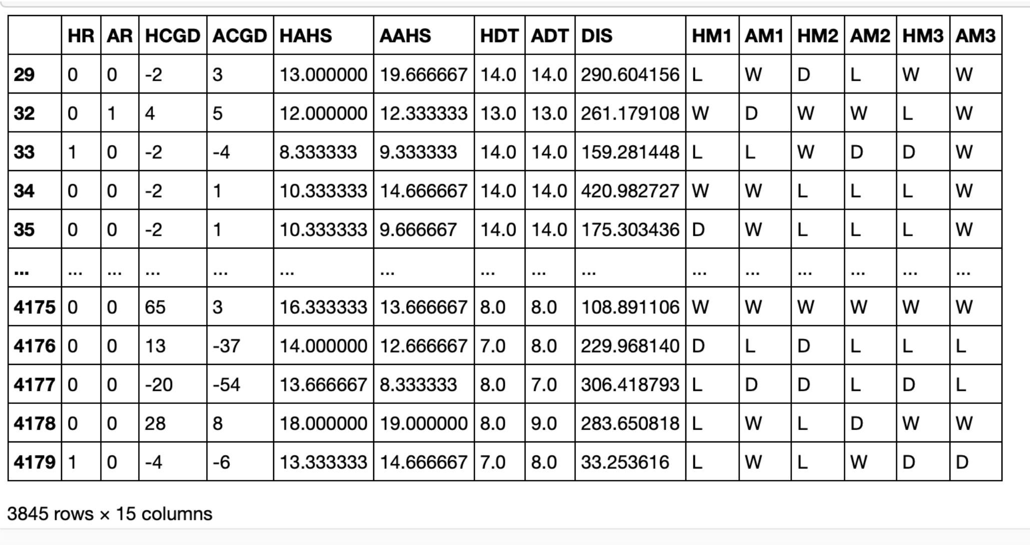
So, within the top 10 we picked FTHG, FTAG , HS, AS, HR, AR to create features:

- FTHG, FTAG ⇒ the cumulative full-time goal difference by home team and away team [HCGD, ACGD]
- HS, AS ⇒ the average number of shots on goal in the past 3 matches by home team and away team [HAS, AAS]
- HR, AR (as features directly)

Apart from that, we also derived features from the following attributes:

- Date ⇒ the delta time from last match of home team and away team [HDT, ADT]
- HomeTeam, AwayTeam ⇒ the distance needed to travel for the away team (with the help of extra data source) [DIS]
- FTR ⇒ the performance of past 3 matches of the home team and away team [HM1,AM1, HM2,AM2. HM3,AM3]

Due to the lack of data in the beginning of each year, there are a few rows containing empty values. After removing these rows and also the intermediate data (which we used to create features), the feature set is shown in Fig. 4.



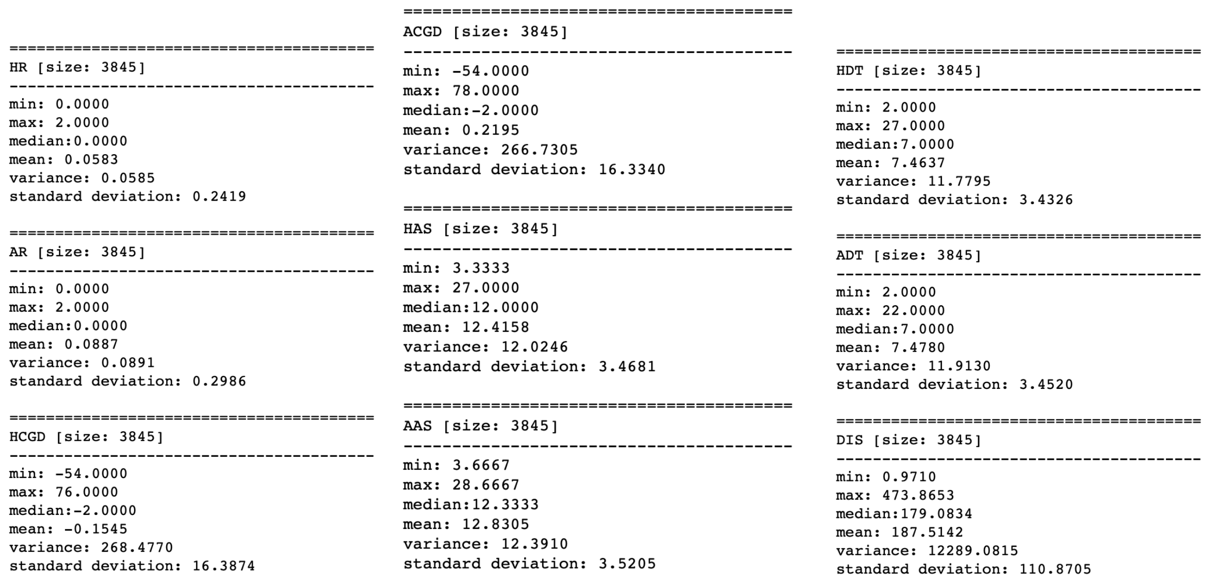
	HR	AR	HCGD	ACGD	HAHS	AAHS	HDT	ADT	DIS	HM1	AM1	HM2	AM2	HM3	AM3
29	0	0	-2	3	13.000000	19.666667	14.0	14.0	290.604156	L	W	D	L	W	W
32	0	1	4	5	12.000000	12.333333	13.0	13.0	261.179108	W	D	W	W	L	W
33	1	0	-2	-4	8.333333	9.333333	14.0	14.0	159.281448	L	L	W	D	D	W
34	0	0	-2	1	10.333333	14.666667	14.0	14.0	420.982727	W	W	L	L	L	W
35	0	0	-2	1	10.333333	9.666667	14.0	14.0	175.303436	D	W	L	L	L	W
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4175	0	0	65	3	16.333333	13.666667	8.0	8.0	108.891106	W	W	W	W	W	W
4176	0	0	13	-37	14.000000	12.666667	7.0	8.0	229.968140	D	L	D	L	L	L
4177	0	0	-20	-54	13.666667	8.333333	8.0	7.0	306.418793	L	D	D	L	D	L
4178	0	0	28	8	18.000000	19.000000	8.0	9.0	283.650818	L	W	L	D	W	W
4179	1	0	-4	-6	13.333333	14.666667	7.0	8.0	33.253616	L	W	L	W	D	D

3845 rows × 15 columns

Fig. 4. Feature set

2.4 Second Data Exploration - Analyse Numerical Features

To learn the characteristics of each feature, we derived the minimum, maximum, median, mean, variance and standard deviation:



```
========================================
HR [size: 3845]
----------------------------------------
min: 0.0000
max: 2.0000
median:0.0000
mean: 0.0583
variance: 0.0585
standard deviation: 0.2419

========================================
AR [size: 3845]
----------------------------------------
min: 0.0000
max: 2.0000
median:0.0000
mean: 0.0887
variance: 0.0891
standard deviation: 0.2986

========================================
HCGD [size: 3845]
----------------------------------------
min: -54.0000
max: 76.0000
median:-2.0000
mean: -0.1545
variance: 268.4770
standard deviation: 16.3874
```

(a)

```
========================================
ACGD [size: 3845]
----------------------------------------
min: -54.0000
max: 78.0000
median:-2.0000
mean: 0.2195
variance: 266.7305
standard deviation: 16.3340

========================================
HAS [size: 3845]
----------------------------------------
min: 3.3333
max: 27.0000
median:12.0000
mean: 12.4158
variance: 12.0246
standard deviation: 3.4681

========================================
AAS [size: 3845]
----------------------------------------
min: 3.6667
max: 28.6667
median:12.3333
mean: 12.8305
variance: 12.3910
standard deviation: 3.5205
```

(b)

```
========================================
HDT [size: 3845]
----------------------------------------
min: 2.0000
max: 27.0000
median:7.0000
mean: 7.4637
variance: 11.7795
standard deviation: 3.4326

========================================
ADT [size: 3845]
----------------------------------------
min: 2.0000
max: 22.0000
median:7.0000
mean: 7.4780
variance: 11.9130
standard deviation: 3.4520

========================================
DIS [size: 3845]
----------------------------------------
min: 0.9710
max: 473.8653
median:179.0834
mean: 187.5142
variance: 12289.0815
standard deviation: 110.8705
```

(c)

Fig. 5. Statistics of each feature column

From the figure, we can draw such conclusions:

- HR & AR: The range is very small (2). From the median, the mean and also the small variance we can know that most values are 0 (as these two features are discrete) while value=2 is of low occurrence.
- HCGD & ACGD: Large range (> 130) with negative values involved. The median and the mean demonstrates that there is a relatively greater number of negative values within the data set.
- AAS & AAS: Moderate range (around 25) with all positive values. The median and the mean is at the half of the range while the variance is reasonable.
- HDT & ADT: Similar moderate range (around 25) and variance with the above pair of data. But the median and the mean is at the one third of the range. Outliers may exist.
- DIS: Large range (> 450) with all positive values. Reasonable median and mean. But from the variance we can know that the value fluctuates significantly.
- Comparing to the other features, the values of HR & AR are too small while that of DIS too large.

2.5 Data Transformation

2.5.1 Label mapping

We mapped the label (i.e., FTR) into numbers for later model training by the rule:

- 'H' → 1
- 'A → 0
- 'D' → 2

2.5.2 Rescale and standardize numerical features

With the conclusions from 3.3, we applied the z-score standardization and min-max rescaling to the numerical features.

2.5.3 Transform categorical features

The categorical data within the feature set is:

HM1,AM1, HM2,AM2, HM3,AM3,

which only take the values 'W', 'L', 'D'.

So we introduced the binary features

HM1_W, HM1_L, HM1_D
AM1_W, AM1_L, AM1_D
......
AM3_W, AM3_L, AM3_D

such that if, for example, HM1 takes the value of 'W', then HM1_W = 1, HM1_L= 0, HM1_D = 0.

3 Methodology Overview

This section will explain what additional data set have we used. Followed by a brief introduction of some base classifiers we applied. After that, we will discuss what techniques that optimize the performance of classifiers have we used.

3.1 Data set

Firstly, we add a feature DIS, which is the distance between the stadiums of the two teams. This is done by using a library 'geopy' which allowed us to estimate the distance of latitude and longitude of the city. Besides, we found the stadium location data from `https://github.com/jokecamp/FootballData/blob/master/other/stadiums-with-GPS-coordinates.csv`. Although it does not cover every team that appears in the provided data set, we did manage to add missing data manually.

Other features were derived from origin training data. For example, HTdd, home team date delta, means the time between the current match and the previous match of the home team.

3.2 Base model

In this problem, we need to classify match result(FTR) into three categories, so we need to build a multi-class classification model to achieve our goal. Hence we found several classic classification models that are suitable for this problem: Multinomial Logistic Regression, Gaussian Naive Bayes, Linear Discriminant Analysis(LDA), Quadratic Discriminant Analysis(QDA), Decision Tree Classifier, and Multilayer Perceptron(MLP, a class of feedforward artificial neural network). These classifiers can be directly imported from Scikit-learn library, and detailed information of these classifiers can be found at: `https://scikit-learn.org/stable/modules/multiclass.html`

3.3 Model evaluation

3.3.1 F1-score

After the first step, we need to evaluate and optimise our model. As a result, we need to find an evaluation standard for our classifiers. We use f1-score as an analysis of the accuracy of the classification model.
The result will be sorted into the matrix. For example, if a result is predicted as false, but the actual result is positive, it will be regarded as TN.

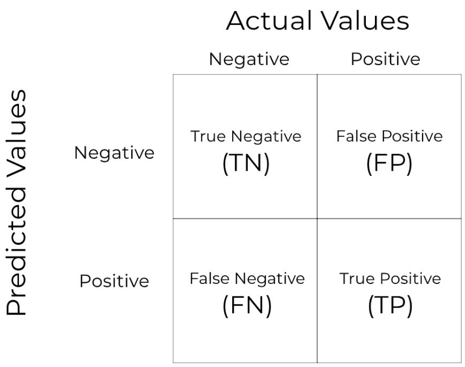
After that, the f1-score is calculated from precision(p) and recall(r), where:

$$p = \frac{TP}{TP + FP} \tag{1}$$

$$r = \frac{TP}{TP + FN} \tag{2}$$

$$f1\_score = 2 * \frac{p * r}{p + r} \tag{3}$$

It shows that f1-score indicates both precision and recall of a model, while accuracy can be contributed mainly by a large number of True Negatives. In this case, f1-score might be a better measure to use since we need to seek a balance between Precision and Recall[1]



		Actual Values	
		Negative	Positive
Predicted Values	Negative	True Negative (TN)	False Positive (FP)
	Positive	False Negative (FN)	True Positive (TP)

Fig. 6. Confusion Matrix[6]

3.3.2 Cross-Validation

Cross-Validation split training set into several folds, then train the model using all folds but one as a training set each time. The remaining folds will be regarded as a validation set.
Cross-Validation allows us to perform validation, but still maintaining the training data set. Thus it will reduce overfitting.

3.3.3 Parameter optimization

A model has many parameters, and diverse parameters produce different model performance. So we need to define a range of parameters to be adjusted for each model and find an approach to seek the best model(with highest f1-score).

3.3.4 Ensemble methods

Ensembled methods seek to consolidate the predictions of one or more base models built with a specific learning algorithm in order to improve the performance of a single model.

There are two main classes of ensemble methods, which are averaging method and boosting method. The averaging method looks to build several classifiers independently and average their predictions while in boosting methods, base estimators are built sequentially, and one tries to reduce the bias of the combined estimator.

Initially, we intended to use a bagging classifier(an averaging method). However, we found from Vu, Braga-Neto, and Dougherty [4]that an empirical basis that ensemble classification by bagging cannot increase the performance of stable classification rules, such as linear discriminant analysis. It seems bagging methods were not a fitting model in this case. As a result, we decide to use boosting methods instead. Based on research by Martínez-Muñoz[3], Marina and Robert P W[2], it seems LDA and NN can be theoretically used as the base estimator in boosting model.

4 Model Training & Validation

In this section, we created groups of parameters to be optimised for each base model, and then we got six optimised classifier with parameters that has the best f1-score. We also applied cross-validation and generate a cross-validation-score for each model as well. Next, we used them as base-estimator in boosting methods. Finally, we compared the overall performance based on f1-score and cross-validation-score and presented the best estimator.

4.1 Optimization of base estimator

4.1.1 Cross-Validation

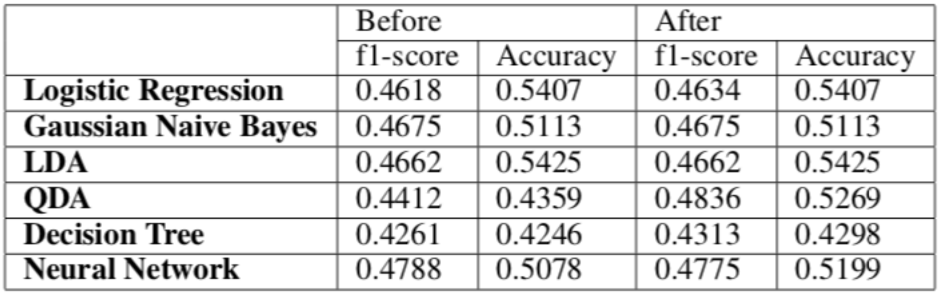
KFold function was used to split the training data into ten sets.We use model_selection.cross_val_score function, it generates a cross-validation score for a model based on the given model and k folds.

4.1.2 Parameter optimisation

A model has many parameters in it. However, we cannot optimse all of them since that will take a massive amount of time to compute. So we select several parameters that could potentially affect the most. After we specified a group of parameters for each model, we then pass that as a parameter to train_predict function together with an f1-scorer. It will use GridSearchCV function to choose parameter with the highest f1-score and produce the best estimator.

4.2 Result

Now we get the result of the prediction on test set by the six optimised models which is shown below:



	Before		After	
	f1-score	Accuracy	f1-score	Accuracy
Logistic Regression	0.4618	0.5407	0.4634	0.5407
Gaussian Naive Bayes	0.4675	0.5113	0.4675	0.5113
LDA	0.4662	0.5425	0.4662	0.5425
QDA	0.4412	0.4359	0.4836	0.5269
Decision Tree	0.4261	0.4246	0.4313	0.4298
Neural Network	0.4788	0.5078	0.4775	0.5199

Fig. 7. Parameter optimisation result

We noticed LDA seems do not change after optimisation, and this could because the parameter group we selected is not large enough, or the default model is already relatively optimised. However, after we tried different possible parameters(some of the parameters are not supported under some particular condition like shrinkage is not supported when the solver is "svd") and increase the range of parameter selection, the result still did not change. So we supposed we could consider the origin setting to be approximately optimised.

In addition, the f1-score of NN slightly decreased after optimisation, and this could because there was some error during fitting or it was slightly overfitted.

In general, the performance of most of the models slightly increased after optimisation. This means our optimisation is valuable.

4.3 Ensemble methods

4.3.1 Ada boost

We used AdaBoostClassifier from scikit-learn as an ensemble method. An AdaBoost classifier is a meta-estimator that starts by fitting a classifier on the original training set and then fits additional copies of the classifier on the same training set but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.[5]

It allows using a single classifier as a base estimator so that we can feed our optimised classifier into it. However, there is some pre-condition to be the base estimator. First, it must provide the "predict_proba" method, which is not supported by QDA and LDA. We tried implementing the function ourselves and switching algorithms in AdaBoostClassifier, but none of them worked. As a result, we have to give up feed LDA and QDA in ensemble methods.

MLP classifier is not supported as well, but we do find an approach to implement the missing attribute in MLP classifier. It now works after we implemented customMLPClassifier class, although the run time for that is remarkably long. We found this solution from: `https://stackoverflow.com/questions/55632010/using-scikit-learns-mlpclassifier-in-adaboostclassifier`

Moreover, we tried to optimise parameter of the AdaBoostClassifier as well. However, it will take forever to compute. So we dropped this task for this classifier.

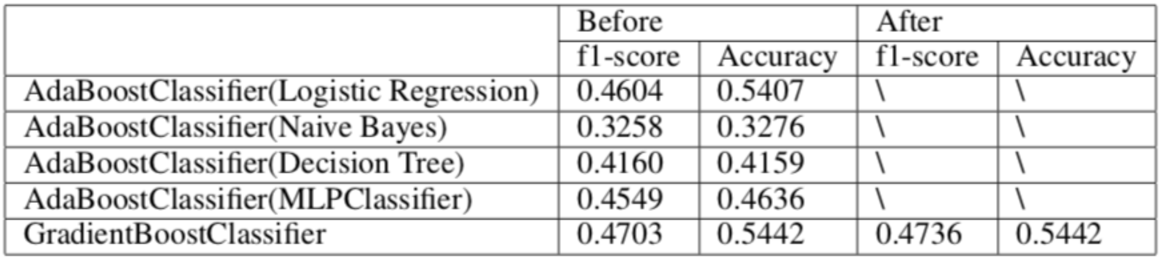
4.3.2 Gradient boost

We also found another classifier provided called GradientBoostClassifier. It builds an additive model in a forward stage-wise fashion. it allows for the optimization of arbitrary differentiable loss functions. In each stage n_classes regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function. [5]

There is no base_estimaor parameter in GradientBoostClassifier. It only has loss as a parameter, which is the loss function, and it can take two values: deviance and exponential. Fortunately, the deviance option is the same with Logistic regression so that we can regard it as a kind of ensemble model on the Logistic Regression classifier.

4.3.3 Ensemble methods result

The result is shown below:



	Before		After	
	f1-score	Accuracy	f1-score	Accuracy
AdaBoostClassifier(Logistic Regression)	0.4604	0.5407	\	\
AdaBoostClassifier(Naive Bayes)	0.3258	0.3276	\	\
AdaBoostClassifier(Decision Tree)	0.4160	0.4159	\	\
AdaBoostClassifier(MLPClassifier)	0.4549	0.4636	\	\
GradientBoostClassifier	0.4703	0.5442	0.4736	0.5442

Fig. 8. Ensemble methods result

We did not do parameter optimisation for AdaBoostClassifier(run-time problem), so the After column is empty. We noticed that the performance of AdaBoostClassifier with Naive Bayes, Decision Tree, and MLP Classifier is worse than that of a single classifier. Another unusual thing here is that AdaBoostClassifier with MLPClassifier as a base estimator has f1-score and accuracy of over 70 when predicting from the training set. It could be evidence of overfitting.

4.4 Final model

After combining all the results, we noticed that accuracy score of GradientBoostClassifier(after parameter optimisation) is the highest among 11 classifier we tested. Although the f1-score is slightly lower than that of QDA(-0.01) and Neural Network(MLP Classifier)(-0.029), the accuracy is much more higher(+0.0173 and + 0.0243). So we think GradientBoostClassifier is the most suitable model in this case.

5 Results [Yi]

6 Final Predictions on Test Set

6.1 Data Pre-processing

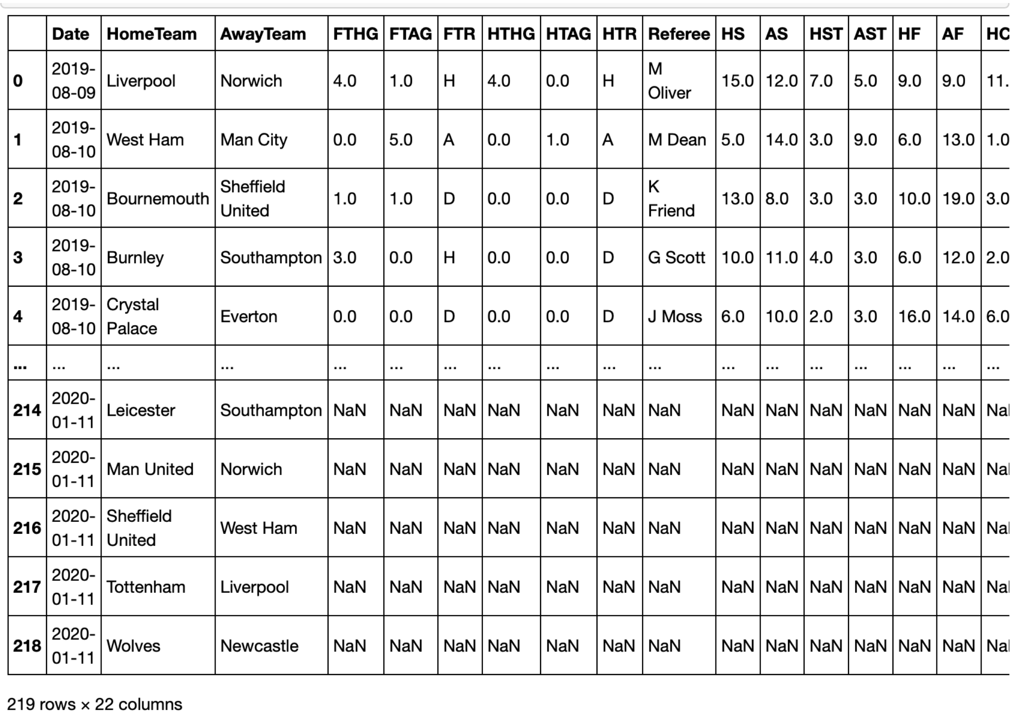
Before predicting the result we first need to process the test set to fit our model. We applied similar operations as we dealing with the training data. To derive features, we import the up-to-date data of the season 2019 from <*http://www.football-data.co.uk*> .

For the up-to-date data, we first remove all the columns that are not presented in the training set, and then check if any invalid data involved. As a result, the shape of the data set reduce from 209 rows x 106 columns to 209 rows x 22 columns.

The shape of test set is 10 rows x 3 columns. So we can get the result by simply looking at the data:

- It only contains three attributes which are all presented in the training set;
- There is no rows containing None, NaN, inifinite or overflowed values.

We then concatenate the up-to-date data of season 2019 and the test set and unify the date. See Fig.9



	Date	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	Referee	HS	AS	HST	AST	HF	AF	HC
0	2019-08-09	Liverpool	Norwich	4.0	1.0	H	4.0	0.0	H	M Oliver	15.0	12.0	7.0	5.0	9.0	9.0	11.
1	2019-08-10	West Ham	Man City	0.0	5.0	A	0.0	1.0	A	M Dean	5.0	14.0	3.0	9.0	6.0	13.0	1.0
2	2019-08-10	Bournemouth	Sheffield United	1.0	1.0	D	0.0	0.0	D	K Friend	13.0	8.0	3.0	3.0	10.0	19.0	3.0
3	2019-08-10	Burnley	Southampton	3.0	0.0	H	0.0	0.0	D	G Scott	10.0	11.0	4.0	3.0	6.0	12.0	2.0
4	2019-08-10	Crystal Palace	Everton	0.0	0.0	D	0.0	0.0	D	J Moss	6.0	10.0	2.0	3.0	16.0	14.0	6.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
214	2020-01-11	Leicester	Southampton	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
215	2020-01-11	Man United	Norwich	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
216	2020-01-11	Sheffield United	West Ham	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
217	2020-01-11	Tottenham	Liverpool	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
218	2020-01-11	Wolves	Newcastle	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal

219 rows × 22 columns

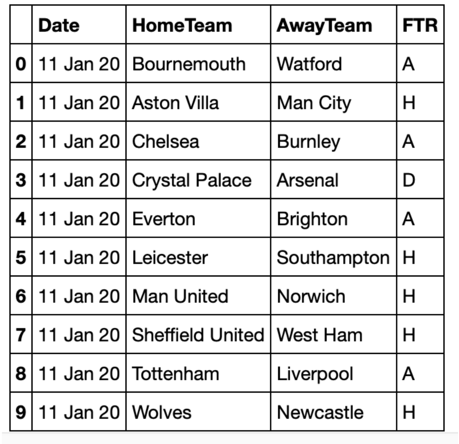
Fig. 9. Data 2019

Same process with when we handle the training data: select the basic attributes, construct features, remove invalid data and remove intermediate data (i.e., the basic attributes).

We re-scale and standardise numerical features by using the scalers that are fitted with the training data in the previous step. Categorical features are transformed using the same rule as we transforming the training data.

6.2 Result Prediction

We can now make prediction using the final model we choose to use and the processed 2019 data set. The result is shown in the Fig.10,



	Date	HomeTeam	AwayTeam	FTR
0	11 Jan 20	Bournemouth	Watford	A
1	11 Jan 20	Aston Villa	Man City	H
2	11 Jan 20	Chelsea	Burnley	A
3	11 Jan 20	Crystal Palace	Arsenal	D
4	11 Jan 20	Everton	Brighton	A
5	11 Jan 20	Leicester	Southampton	H
6	11 Jan 20	Man United	Norwich	H
7	11 Jan 20	Sheffield United	West Ham	H
8	11 Jan 20	Tottenham	Liverpool	A
9	11 Jan 20	Wolves	Newcastle	H

Fig. 10. Final prediction

7 Conclusion [Terry]

References

[1] Brownlee, Jason (2014) Classification Accuracy is Not Enough: More Performance Measures You Can Use <*https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use*>.

[2] Marina Skurichina; Robert P. W. Duin (2000) Boosting in Linear Discriminant Analysis - Multiple Classifier Systems P.190-199 Publisher: Springer Berlin Heidelberg Dec 1st 2000 ISBN: 978-3-540-45014-6 <*https://arxiv.org/abs/1909.12098v1*>.

[3] Martínez-Muñoz, Gonzalo (2019) Sequential Training of Neural Networks with Gradient Boosting Sep 26th 2019 <*https://arxiv.org/abs/1909.12098v1*>.

[4] Thang T. Vu ; Ulisses M. Braga-Neto ; Edward R. Dougherty (2019) Bagging degrades the performance of linear discriminant classifiers July 24th 2009 Published in: 2009 IEEE International Workshop on Genomic Signal Processing and Statistics DOI:10.1109/GENSIPS.2009.5174344 <*https://ieeexplore.ieee.org/document/5174344*>.

[5] Scikit-learn (2019) 1.11. Ensemble methods <*https://scikit-learn.org/stable/modules/ensemble.html#ensemble*>.

[6] Radečić, Dario (2009) A Non-Confusing Guide to Confusion Matrix Sep 28th 2009 <*https://towardsdatascience.com/a-non-confusing-guide-to-confusion-matrix-7071d2c2204f*>.

[7] Sharma, Mohit. What Steps should one take while doing Data Preprocessing?. June 20th 2018. June 1st 2019. <*https://hackernoon.com/what-steps-should-one-take-while-doing-data-preprocessing-502c993e1caa*>.

[8] J. González. Scaling/ normalisation/ standardisation: a pervasive question. Oct 18th 2018. June 3st 2019. <*https://quantdare.com/scaling-normalisation-standardisation-a-pervasive-question/*>.