Machine Learning Engineer Nanodegree

Capstone Project

Raul Britto
May 20th, 2019

Definition

Project Overview

Currently, sporting bets companies have become very popular with marketing campaigns on websites, TV channels, and disclosure sports groups. Hence, the betting volume and total value of bettings have increased with the years. This growing of sporting bets companies is based on the increase of data and statistics sporting, increased of sports transmission, publicity and the possibility of sports betting being made from home on the Internet.

Thus more and more people are encouraged to do sporting bets and follow sports that they love it. For example, It is estimated that Brazilian annually about R$ 2 billion in sports betting [1]. However, most parts of them did bets without using any prediction model or statistical model and these companies earn more and more money with amateur gamblers who betting as a hobby. According to a survey published by the Economist magazine, Brazilian bettors lost US $ 4.1 billion in 2014 on lottery and betting sites[2].

In this project, it was created a supervised learning model to predict if a Premier League match will be finished as a Home Team victory, Away Team victory or Draw between the teams.

Project Statement

The goal of this project is to create a sporting bet model to soccer matches, specifically in the Premier League[3], using data from the last ten seasons and statistics such as number of goals, corners and yellow cards for each team to predict if the match on analysis will be a draw or which team will be the victorious team.

The most part of the data was collected on the Football Data website[4], where it is available to users details and data about soccer games of several championships and odds given by the sportsbook. The files are separated by seasons and specifically were collected

data from the season 2009/2010 to the season 2018/2019. Other data were collected from the official website of the Premier League[3], saved and manipulated. These data were added to the database after treatment, the average of each team was calculated from the relevant statistics.

As of the dataset created, the data will be preprocessed (normalization, encoding, splitting, etc) and used as input by some supervised algorithms, like SVM, AdaBoost, and XGBoost. Thus, each model will be trained and evaluated, the two best algorithm results will be select and optimized and confronted against the benchmark model.

To split the data into two sets: training and test set. It was followed by the recommendation given by “A machine learning framework for sport result prediction” [7]. In the case where it is used data from multiple seasons, a common approach is to use the earlier seasons as training data, to predict the later season as the test data set. So, in our case, it was used the first nine seasons completed and almost half of the last season like a training set and the test set is composed by the most recent 200 matches, being the last ten matches forming in the benchmark input.

Metrics

To evaluate model performance, the match results were classified into home wins, away wins and draws and then look at the number of matches that the model has correctly identified, using a standard confusion matrix.

However, there is an imbalance in the class values for the data, because home teams usually have a bigger probability to win the game. This is a common phenomenon known as home advantage phenomenon. So, it was applied the ROC curve evaluation as well as a confusion matrix.

ROC curve is one of the best performance models for classification problems because tells how much the model under analysis is capable of differentiate between classes. In our study, a higher ROC tells us that how many matches ended as Home Victories were predict as being really a home victory by our model. The same analysis is valid to games ended in draw and away victory.

ROC curve is defined by the expression: TPR/FPR. Where TPR is the true positive rate, given by: $True\ Positives/\ (True\ Positives\ +\ False\ Negatives)$ and the FPR is given by: $1\ -\ True\ Negatives\ /\ (True\ Negatives\ +False\ Positives).$

Analysis

Data Exploration and Visualization

The most part of the data was collected on the Football Data website [4], where it is available to users details and data about soccer games of several championships and odds

given by the sportsbook. The files are separated by seasons and specifically were collected data from the season 2009/2010 to the season 2018/2019.

The data contained in these files are:

- *Away Team*;
- *HomeTeam*;
- *Referee*;
- Date of match (later separated into seasons with the *Season* column);
- Total goals in the first half for each team (*HTHomeGoals / Away*);
- Which team won in the first half (*HTResult*);
- Total goals in the match by team (*FTHomeGoals / Away*);
- Which team won in time the match (*FTResult*);
- Number of corners (*HomeTeamCorners/Away*);
- Number of shoots (*HomeShots/Away*);
- How many of these shoots were in the target (*HomeTeamShotsTarget/Away*);
- Number of yellow and red cards (*HomeTeamRedCards/Away* and *HomeTeamYellowCards/Away) ;*
- Number of fouls by team (*HomeTeamFouls/Away*);
- Quotas from betting websites 365 Bet [6] for win of each team or draw (*B365H, B365A, B365D*);

Other data were collected from the official website of the Premier League [3], saved and manipulated. These data were added to the database after treatment, the average of each team was calculated from the relevant statistics. Below it follows the description of the data and its respective attribute added to the base:

- Goals of each team per season (*MeanGoalsHome / Away*);
- Goals conceded from each team per season (*MeanGoalsConHome / Away*);
- Corners by each team per season (*MeanCornersHome / Away*);
- Yellow cards for each season (*MeanCardsHome / Away*);
- Shoots of each team per season (*MeanShotsHome / Away*).

A new column is added into the dataset, the column ‘Season’. This attribute starts equal to 1 for the season 09-10 and finishes with 10 for the latest season 18-19. The data are read by season, each corresponding file has 38 rounds with 10 games each one. So in the total, a unique season file has 380 rows (matches) and 38 columns (attributes) as it is possible to see in figure 1.

Div	Date	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	Referee	HS	AS	HST	AST	HF	AF	HC
E0	15/08/09	Aston Villa	Wigan	0	2	A	0	1	A	M Clattenburg	11	14	5	7	15	14	4
E0	15/08/09	Blackburn	Man City	0	2	A	0	1	A	M Dean	17	8	9	5	12	9	5
E0	15/08/09	Bolton	Sunderland	0	1	A	0	1	A	A Marriner	11	20	3	13	16	10	4
E0	15/08/09	Chelsea	Hull	2	1	H	1	1	D	A Wiley	26	7	12	3	13	15	12
E0	15/08/09	Everton	Arsenal	1	6	A	0	3	A	M Halsey	8	15	5	9	11	13	4
E0	15/08/09	Portsmouth	Fulham	0	1	A	0	1	A	M Atkinson	16	9	4	3	11	18	6
E0	15/08/09	Stoke	Burnley	2	0	H	2	0	H	S Bennett	12	9	5	5	15	10	3
E0	15/08/09	Wolves	West Ham	0	2	A	0	1	A	C Foy	19	16	11	13	9	5	8
E0	16/08/09	Man United	Birmingham	1	0	H	1	0	H	L Mason	26	6	17	4	13	7	13
E0	16/08/09	Tottenham	Liverpool	2	1	H	1	0	H	P Dowd	17	6	11	3	14	16	6

Figure 1: First 10 rows for the dataset season 09-10

Some attributes were removed because they didn't have any relevance in the analysis as the columns: 'Div', 'Date','Unnamed: 0','FTHG','FTAG'. The next step is to create a new data frame that contains all the data only in a variable. From this variable, it is possible to start to explore the data and make a statistical analysis as in figure 2.

```
Total number of records: 3800
Home wins: 1322
Away wins: 917
Draw mathces: 1561
Percentage of wins Home: 34.78947368421053%
Percentage of wins Away: 24.13157894736842%
Percentage of Draws: 41.078947368421055%
```

Figura 2: Exploratory analysis for the entire dataset by type of output class

In figure 2, it is presented the amount of each class for the entire dataset and its respective relative percentage of the total. How it is observed the classes have some imbalance between them.

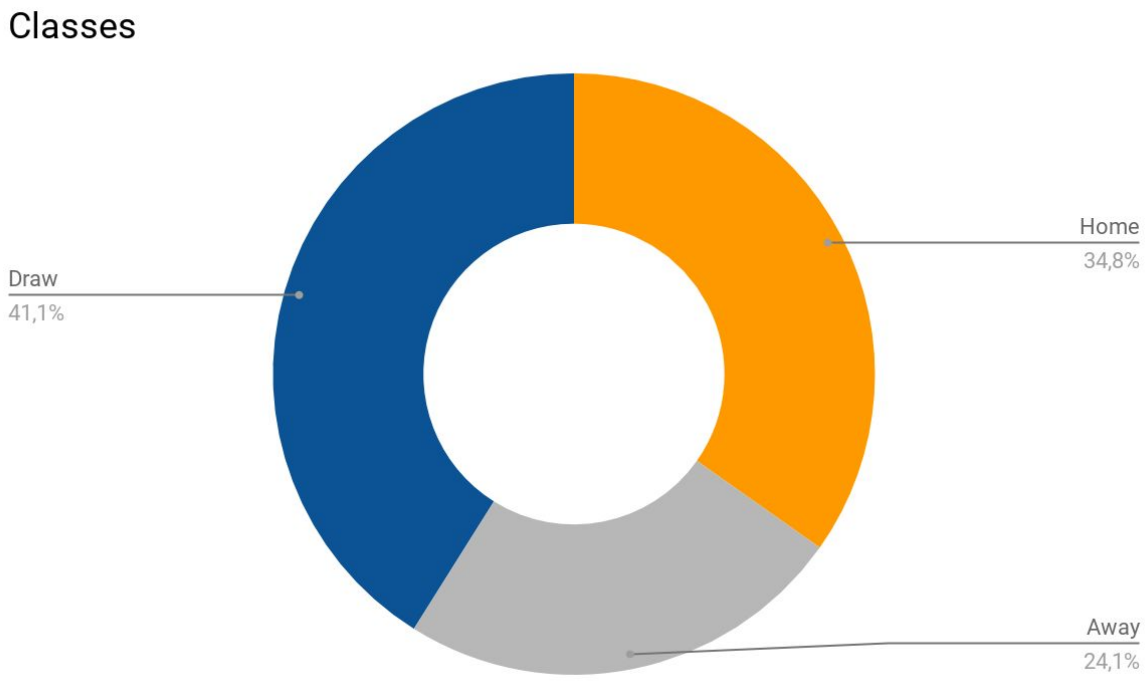


Figure 3: Distribution of classes

Featureset Exploration

In the text below, we have an exploratory description in text for each attribute analyzed. Next, we have two pieces of code a statistical description for numerical variables and another for categorical variables.

- **HomeTeam**: Aston Villa', 'Blackburn', 'Bolton', 'Chelsea', 'Everton', 'Portsmouth', 'Stoke', 'Wolves', 'Man United', 'Tottenham','Sunderland', 'Wigan', 'Birmingham', 'Burnley', 'Hull','Liverpool', 'Arsenal', 'Man City', 'Fulham', 'West Ham','West Brom', 'Newcastle', 'Blackpool', 'QPR', 'Swansea', 'Norwich','Reading', 'Southampton', 'Crystal Palace', 'Cardiff', 'Leicester','Bournemouth', 'Watford', 'Middlesbrough', 'Brighton','Huddersfield'.
- **AwayTeam**: 'Wigan', 'Man City', 'Sunderland', 'Hull', 'Arsenal', 'Fulham', 'Burnley', 'West Ham', 'Birmingham', 'Liverpool', 'Chelsea', 'Wolves', 'Portsmouth', 'Man United', 'Tottenham', 'Stoke', 'Bolton', 'Blackburn', 'Everton', 'Aston Villa', 'West Brom', 'Blackpool', 'Newcastle', 'Norwich', 'Swansea', 'QPR', 'Southampton', 'Reading', 'Cardiff', 'Crystal Palace', 'Leicester', 'Watford', 'Bournemouth', 'Middlesbrough', 'Huddersfield', 'Brighton'.
- **FTResult**: continuous, ranging from 0 to 9.
- **HTHomeGoals**: continuous, ranging from 0 to 5.
- **HTAwayGoals**: continuous, ranging from 0 to 5.
- **HTResult** : 'A', 'D', 'H'.

- **Referee**: 'M Clattenburg', 'M Dean', 'A Marriner', 'A Wiley', 'M Halsey', 'M Atkinson', 'S Bennett', 'C Foy', 'L Mason', 'P Dowd', 'M Jones', 'L Probert', 'P Walton', 'H Webb', 'S Attwell', 'K Friend', 'St Bennett', 'Mn Atkinson', 'A Taylor', 'M Oliver', 'N Swarbrick', 'J Moss', 'R East', 'C Pawson', 'R Madley', 'P Tierney', 'K Stroud', 'G Scott', 'S Hooper', 'C Kavanagh', 'I Mason', 'D Coote', 'A Madley', 'J Linington'.
- **HomeShots** : continuous, ranging from 0 to 43.
- **AwayShots** : continuous, ranging from 0 to 30.
- **HomeTeamShotsTarget** : continuous, ranging from 0 to 24.
- **AwayTeamShotsTarget** : continuous, ranging from 0 to 20.
- **HomeTeamFouls** : continuous, ranging from 0 to 24.
- **AwayTeamFouls** : continuous, ranging from 0 to 26.
- **HomeTeamCorners**: continuous, ranging from 0 to 19.
- **AwayTeamCorners**: continuous, ranging from 0 to 19.
- **HomeTeamYellowCards**: continuous, ranging from 0 to 7.
- **AwayTeamYellowCards**: continuous, ranging from 0 to 9.
- **HomeTeamRedCards**: continuous, ranging from 0 to 2.
- **AwayTeamRedCards**: continuous, ranging from 0 to 2.
- **Season**: continuos, ranging from 1 to 10.
- **MeanCornersHome**: continuous.
- **MeanCornersAway**: continuous.
- **MeanShotsHome**: continuous.
- **MeanShotsAway**: continuous.
- **MeanGoalsHome**: continuous.
- **MeanGoalsAway**: continuous.
- **MeanGoalsConHome**: continuous.
- **MeanGoalsConAway**: continuous.
- **B365H**: continuous.
- **B365D**: continuous.
- **B365A**: continuous.

Correlation Matrix

The correlation between the attributes was calculated, and the largest of them was found between columns B365A and B365D, but they were not removed because this result shared the same explanation of the home advantage phenomenon. If an away team is more strong the another the victory by the home team will decrease and will be distributed between the odds of away victory and draw. Another example of correlation in the attributes was between MeanGoalsHome/Away and MeanShotsHome/Away; and MeanGoalsHome/Away and MeanCornersHome /Away. This can be explained by the own

soccer rules: to make goals it is necessary to shoot and a corner is originated by an attempt of a shoot on goal.

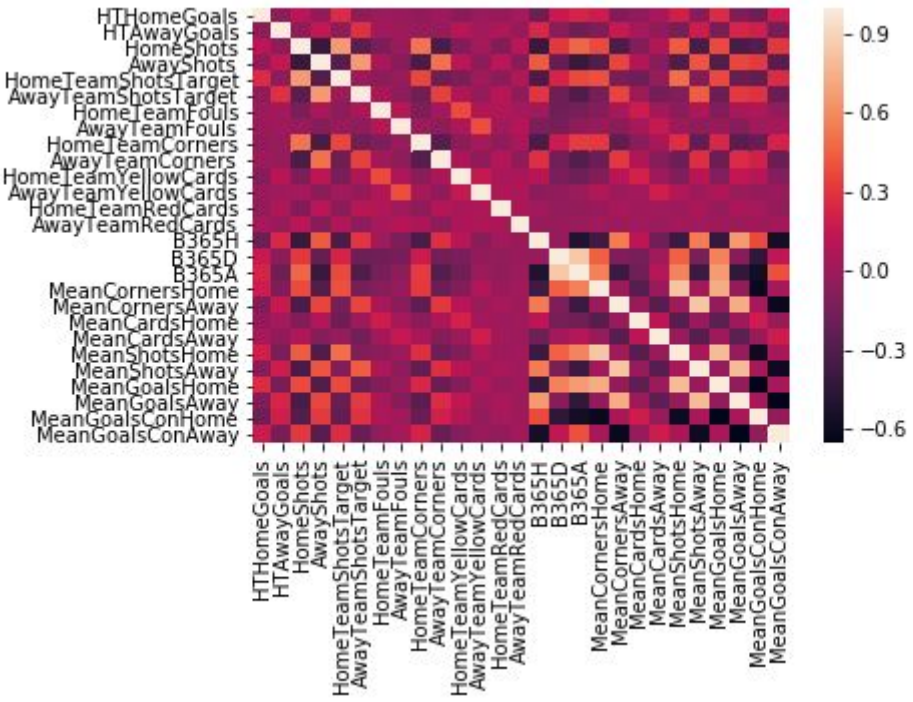


Figure 4: Correlation Matrix

Algorithms and Techniques

The following algorithms will be analyzed: DecisionTree, SVM, AdaBoost, and XGBoost. The choice of the algorithms was made as follows: every algorithm was evaluated with the default parameters and the two that have obtained the two best results were optimized. The choice of each algorithm was made for the following reasons:

1. **Decision Tree:** it is an algorithm that works like a decision support tool, easy to use with low computational cost and able to handle both categorical and numerical data, and graphically allow you to interpret the data. A decision tree has a structure similar to a flow chart in which each node or division represents an attribute being evaluated and each possible branch is the answer given by the algorithm analysing a specific instance. The paths from root to leaf represent classification rules.

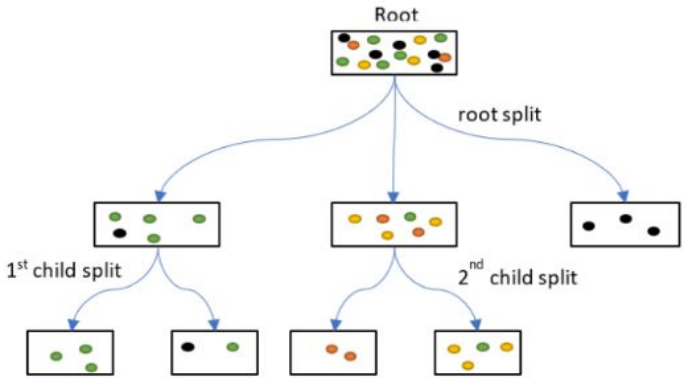


Figure 5: Decision Tree Model

2. **Support Vector Machine (SVM):** SVM is a supervised learning models used for classification and regression analysis. In SVM the input data point is read as a n-dimensional vector, and it tries to know whether it is possible to separate such points with a (n-1)-dimensional hyperplane (linear classifier). Another interesting SVM feature is that there are many hyperplanes that could classify the data, but the best choice is that hyperplane that represents the largest separation, or margin, between the two classes or more classes. So, it is utilized the hyperplane so that the distance from it to the nearest data point on each side is maximized.
   SVM also has good points for the our specific case: works well in small datasets without noise. The limits could be non-linear but need to be well defined.
3. **AdaBoost:** it is an ensemble machine learning method that can be used in conjunction with many other types of learning algorithms to improve performance. The input for the AdaBoost are the output from other machine learning algorithms, known as ‘'weak learners', so the weak learners outputs are combined into a weighted sum that represents the final output of the boosted classifier (AdaBoost). The main purpose of AdaBoost method is the individual learners could be weak, but as long the performance of each one is slightly better than random guessing, so the final model can become a ‘strong learner’.

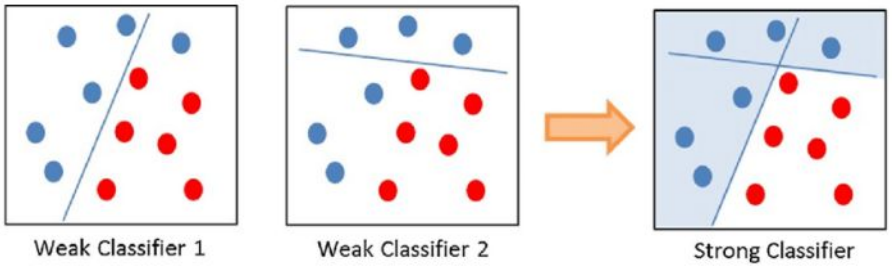


Figure 6: AdaBoost concept

4. **XGBoost:** is an implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm that attempts to accurately predict a target variable by combining the estimates of a set of simpler, 'weaker leaners' similarly to AdaBoost. Typically the ‘weak learns’ are

Decision Trees and has good performance and velocity. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function[8].
The main idea of gradient boosting can be interpreted as an optimization algorithm on a suitable cost function.That is, algorithms that optimize a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction.

Benchmark

How the 2018/2019 season has finished recently, the results of the last ten games will be used to compare the model proposed's prediction with the predictions taken from the benchmarker, the website Footballpredictions.com [5]. The results given by the website will be compared with the result from regression and classification models, so all the predictions, be they given by regression, classification or benchmarker output will be transformed into a classification output to create a common way to compare the results.

Methodology

Data Preprocessing

The pre-processing step was done as follows:

1. Season column: the season attribute as encoded using the rule the first season had the integer value 1, the second one had the value 2 and the last one had the values 10. In this way, more recent games have the biggest weight in the model;
2. Join data frames: all data frames were joined to an only data frame variable, more easier to handle;
3. Drop irrelevant attributes: attributes irrelevant to the analysis were removed from the data frame;
4. Rename columns: some columns were renamed from abbreviations to more meaningful names. For example: 'HTAG' was changed by 'HTAwayGoals';
5. Remove the target attribute: the column 'FTResult' was removed from the data frame variable to a new variable responsible to store all the outputs;
6. Encode the target attribute: the new target variable was encoded as follows: 'H' = 2, 'D' = 1 and 'A' = 0;
7. Normalizing numerical attributes: All numerical columns were mapped and normalized, figure 5.
8. One-hot encoding: All categorical features were encoded follow the one-hot encoding procedure.

	HomeTeam	AwayTeam	HTHomeGoals	HTAwayGoals	HTResult	Referee	HomeShots	AwayShots	HomeTeamShotsTarget	AwayTeamShotsTarget
0	Aston Villa	Wigan	0.0	0.2	A	M Clattenburg	0.255814	0.466667	0.208333	0.35
1	Blackburn	Man City	0.0	0.2	A	M Dean	0.395349	0.266667	0.375000	0.25
2	Bolton	Sunderland	0.0	0.2	A	A Marriner	0.255814	0.666667	0.125000	0.65
3	Chelsea	Hull	0.2	0.2	D	A Wiley	0.604651	0.233333	0.500000	0.15
4	Everton	Arsenal	0.0	0.6	A	M Halsey	0.186047	0.500000	0.208333	0.45

Figure 5: Normalization of numerical features

9. Features null: check how many features present null values;
10. Duplicate instances: verify how many rows are duplicated;
11. Benchmark creation: the benchmark was created removing the ten last rows from training and test sets, figure 6.
12. Training, test set: split the data into two sets, training set is composed by 3590 intances and test set by 200 intances;

	HTHomeGoals	HTAwayGoals	HomeShots	AwayShots	HomeTeamShotsTarget	AwayTeamShotsTarget	HomeTeamFouls	AwayTeamFouls	HomeTeamCor
3790	0.2	0.4	0.139535	0.666667	0.083333	0.45	0.500000	0.28	0.10
3791	0.0	0.0	0.325581	0.566667	0.208333	0.30	0.458333	0.08	0.21
3792	0.6	0.2	0.395349	0.533333	0.333333	0.40	0.458333	0.28	0.21
3793	0.0	0.4	0.372093	0.433333	0.083333	0.30	0.250000	0.28	0.26
3794	0.0	0.0	0.209302	0.466667	0.125000	0.20	0.375000	0.28	0.21
3795	0.2	0.0	0.302326	0.233333	0.208333	0.10	0.125000	0.40	0.21
3796	0.0	0.2	0.604651	0.433333	0.416667	0.20	0.375000	0.20	0.57
3797	0.2	0.0	0.232558	0.333333	0.125000	0.15	0.333333	0.20	0.21
3798	0.2	0.0	0.255814	0.566667	0.125000	0.45	0.416667	0.48	0.36
3799	0.0	0.4	0.395349	0.533333	0.333333	0.45	0.416667	0.36	0.36

10 rows × 146 columns

Figure 7: Benchmark set

Implementation

To run the algorithms proposed were necessary to implement the following functions:

- **Multiclass_roc_auc_score:** this function calculates the roc-score for the multiclassifcation scenario (Home, Away and Draw). Figure 7.

```
from sklearn.metrics import roc_auc_score
from sklearn import preprocessing

def multiclass_roc_auc_score(y_test, y_pred, average="macro"):

      lb = preprocessing.LabelBinarizer()
      lb.fit(y_test)

      y_test = lb.transform(y_test)
      y_pred = lb.transform(y_pred)
      return roc_auc_score(y_test, y_pred, average=average)
```

Figure 8: Roc_auc_score implementation

- **Train_predict:** this function creates a template to run and evaluate (Confusion Matrix and ROC score) the model selected to test. Figure 8.

```
from sklearn.metrics import confusion_matrix

def train_predict(learner, X_train, y_train, X_test, y_test):
      '''
      inputs:
      - learner: the learning algorithm to be trained and
predicted on
      - X_train: features training set
      - y_train: income training set
      - X_test: features testing set
      - y_test: income testing set
      '''
      results = {}
      # Fit the learner to the training data
      start = time() # Get start time
      learner = learner.fit(X_train,y_train)
      end = time() # Get end time

      # Calculate the training time
      results['train_time'] = end - start
      # Print the results
      print ('Trained model in {:.4f}
seconds'.format(results['train_time']))

      # Get the predictions on the test set(X_test)
      start = time() # Get start time
      predictions_test = learner.predict(X_test)
      end = time() # Get end time

      # Calculate the total prediction time
      results['pred_time'] =  end - start
      # Print and return results
      print ('Made predictions in {:.4f}
seconds.'.format(results['pred_time']))

      # Success
      print("{} trained on {}
samples.".format(learner.__class__.__name__, len(X_train)))

      #Confusion Matrix
      classes_=[ 'Home','Draw','Away']
      plot_confusion_matrix(confusion_matrix(y_test,
predictions_test,labels=[2,0,1]), classes_)
      #Roc
      results['roc-auc'] = multiclass_roc_auc_score(y_test,
```

```
predictions_test)
      print('Roc-auc = {:.4f} \n'.format(results['roc-auc']))
      print('================================')

      # Return the results
      return results
```

Figure 9: Train_predict function implementation.

- **Main block:** this block initialize the models and collects the return from each model as well. Figure 9,

```
# Import supervised learning models from sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from xgboost.sklearn import XGBClassifier

# Initialize the models
clf_A = DecisionTreeClassifier(random_state=1)
clf_B = SVC(random_state=2)
clf_C = AdaBoostClassifier(random_state=3)
clf_D = XGBClassifier(random_state=4)


# Collect results on the learners
results = {}
for clf in [clf_A, clf_B, clf_C,clf_D]:
      clf_name = clf.__class__.__name__
      results[clf_name] = {}

      results[clf_name] = train_predict(clf, X_train, y_train,
X_test, y_test)
```

Figure 10: initializing the learning models

One of the complications during the project was calculated and plot the ROC-Score to multiclass prediction. Thus, it was necessary to create new functions to do that.

Refinement

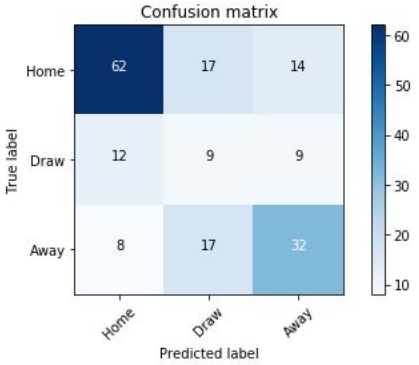
From the two best algorithms results, it was tried to optimize both of them. The optimization was done using the Grid Search method. For this, it was created a list with the parameters to be optimized by each algorithm.

Results

Model Evaluation and Validation

Decision Tree

The first algorithm evaluated was the Decision Tree presenting the following results. These results were the worst obtained by the algorithms tried.



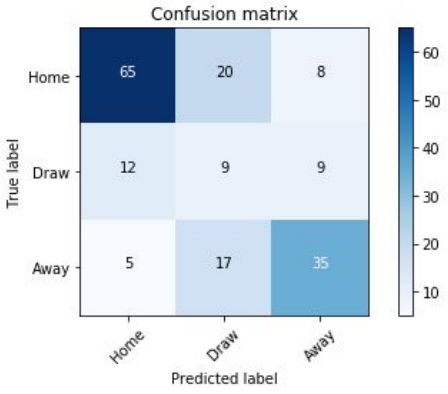

```
Roc-Score = 0.6474
==============================
```

Figure 11: Decision Tree results

SVM

The SVM results were better than they obtained by Decision Tree. ROC-Score was better than the obtained by Decision Tree algorithm and in the confusion matrix, it is possible to see that more matches were correctly classified as Home and Away Victory.

```
Trained model in 2.5034 seconds
Made predictions in 0.0842 seconds.
SVC trained on 3610 samples.
```




```
Roc-Score = 0.6721
==============================
```

Figure 12: SVM results

AdaBoost

AdaBoost results were even better than SVM results as well as ROC-Score. However, the time elapsed to train the model was significantly reduce applying the same training set.

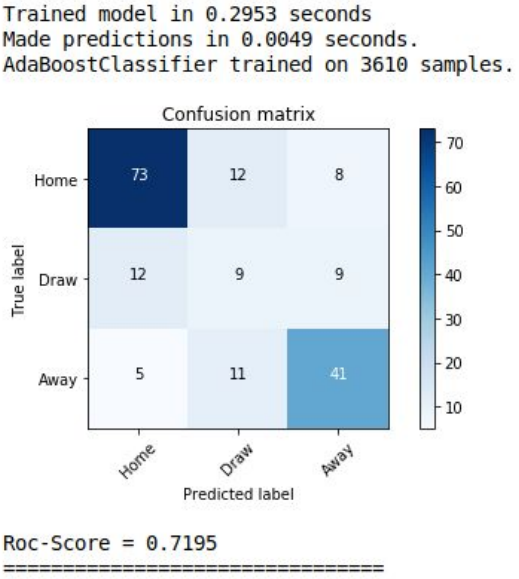


Figure 12: AdaBoost results

XGBoost

XGBoost results were approximate equals to AdaBoost results, but the computational cost to train the model was much higher now applying the same training set.

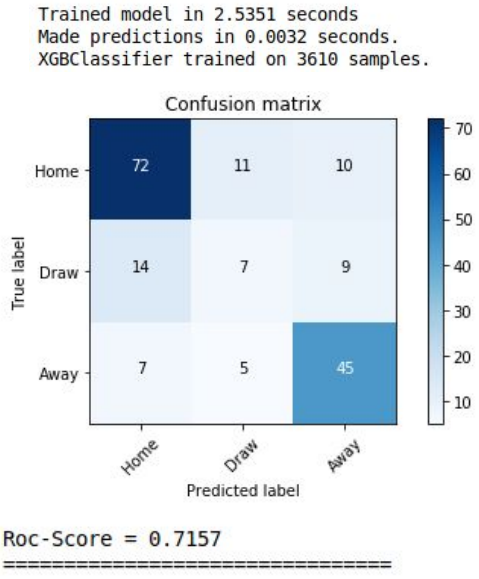


Figure 13: XGBoost results

So, in this way, the two algorithms with the better results were AdaBoost and XGBoost. Each one of these was optimized through the Grid Search method using an individual parameter set. For Decision Tree algorithm were used the following parameters and values and the results obtained were:

```
parameters = {'n_estimators': [10,25,50,75,100,150],
              'learning_rate':[1,0.75,0.5,0.25,0.1, 0.05, 0.01],
              'algorithm' : ['SAMME', 'SAMME.R']}
```

Figure 12: AdaBoost parameters optimization

```
Unoptimized model
------
ROC-Score on testing data: 0.7257

Optimized Model
------
ROC-Score on the testing data: 0.7226
```

Figure 14: AdaBoost parameters optimization results

In this way, it is not possible to reach an optimized model with better results using Grid Search. The results present almost the same ROC-Score, this could be caused because the function stucks in a maximum local or the function is very close to the global maximum.XGBoost optimization task was done in a simialliry way, but using the following parameters:

```
params={'max_depth': [5], #[3,4,5,6,7,8,9],
      'subsample': [0.6], #[0.4,0.5,0.6,0.7,0.8,0.9,1.0],
      'colsample_bytree': [0.5], #[0.5,0.6,0.7,0.8],
      'n_estimators': [2000],
      'reg_alpha': [0.03] #[0.01, 0.02, 0.03, 0.04]}
```

Figure 15: XGBBoost parameters optimization

The same way that happened to AdaBoost the optimization process does not have significant variations on results, in the figure 15.

```
Unoptimized model
------
ROC-Score on testing data: 0.7289

Optimized Model
------
ROC-Score on the testing data: 0.7201
```

Figure 16: XGBoost parameters optimization results

Benchmark

Since we already have the optimized models we will confront them with the proposed benchmark. The benchmark proposed was to get the ten last games of the last Premier League season predicted by Footballpredictions.com[8], but how the website give their prediction in possibly match score it was transformed the score to a class proposed in the study. Then our benchmark model is given by:

Home Team	Away Team	Score	Prediction Class	Correct Classification?
Brighton	Man City	1-3	A	Yes
Burnley	Arsenal	1-2	A	Yes
Crystal Palace	Bournemouth	3-3	D	No
Fulham	Newcastle	1-1	D	No
Leicester	Chelsea	1-2	A	No
Liverpool	Wolves	2-1	H	Yes
Man United	Cardiff	2-0	H	No
Southampton	Huddersfield	2-2	D	Yes
Tottenham	Everton	1-1	D	Yes
Watford	West Ham	1-1	D	No

Table 1: Benchmark results from Footballpredictions

In this way, the benchmark model proposed has five hits on ten matches. The results of each optimized model when given the benchmark matches as input are:

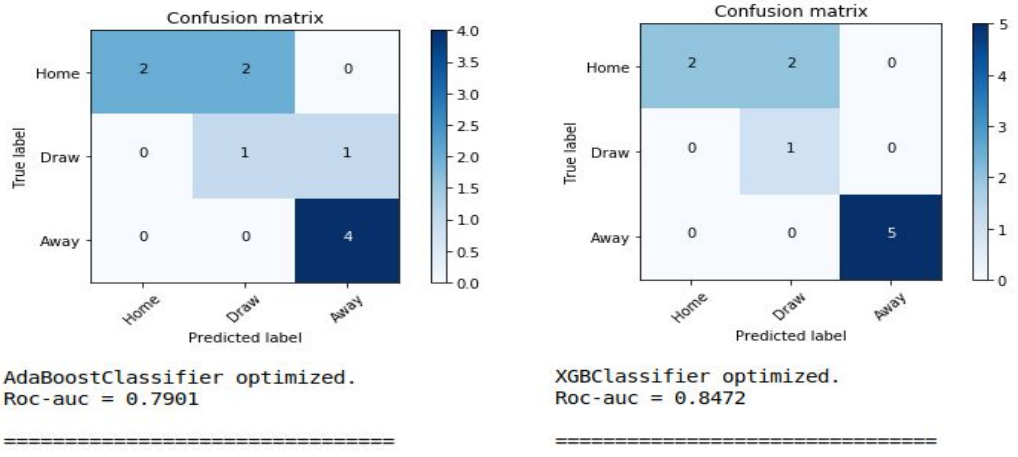


Figure 17: AdaBoost and XGBoost benchmark result

Thus, both models are able to beat the benchmark model. AdaBoostClassifier had seven hits on ten matches, while XGBoost Classifier was even better with eight hits on ten matches.

Justification

The models analyzed have had a better result than the prediction provided by the benchmark. However, the best benchmarker to be applied would be matches that did not happen. So, we can really compare more difficult inputs for both methods. It is important to notice that both models have more difficult to predict matches will be draws, although the 'Draw' class is the class with more instances.

Conclusion

Free-Form Visualization

The below table compares the result class with the classes predicted by each agent (Benchmark, AdaBoost, and XGBoost). As it is possible to see in the table below both of models proposed have won against the Benchmark.

	Home Team	Away Team	Benchmark Class	AdaBoost Class	XGBoost Class	Real Class
	Brighton	Man City	A	A	A	A
	Burnley	Arsenal	A	D	A	A
	Crystal Palace	Bournemouth	D	H	H	H
	Fulham	Newcastle	D	A	A	A
	Leicester	Chelsea	A	D	D	D
	Liverpool	Wolves	H	H	H	H
	Man United	Cardiff	H	A	A	A
	Southampton	Huddersfield	D	H	H	D
	Tottenham	Everton	D	H	H	D
	Watford	West Ham	D	A	A	A
Total			5/10	7/10	8/10	

Reflection

The process used in this project can be summarized in the following steps:

1. Find an initial problem with public datasets;
2. The data was download and preprocessed from different data source;
3. To find and propose and plausible benchmark;
4. Select classifier models to be trained in this project;
5. Preprocessed the data;
6. Implement the models proposed using the data obtained;
7. To find a good and reasonable evaluate method.

I thought the steps 4 and 7 the most difficult, to find a good and appropriate model to the problem was a little bit difficult. I have tried some algorithms like CatBoost and LightGBM, but both of them had similar results with high computational cost and time elapsed. To evaluate the model and have found an excellent article[7] that gives the correct direction.

Improvements

To improve the models I suggest to increase more data like:

- **Streaks**: a number of last game results, like last five results by team, example: ‘W-D-L-W-W’. The number of matches analyzed can vary as a parameter;
- **Performance in earlier encounters**: to get the historic matches between two teams;
- **Coach:** a specific coach can havê direct impact in the final result;
- **Matches with special importance**: sometimes a team can prioritize some specific matches then another;
- **Ball Possession**: this statistical analysis has a very important mean, how much time a team has the ball implies in how much time the team were in the attack;
- **Promotion to higher leagues**: usually a team who came from a lower division has more difficult in a higher division;
- **Soccer skills**: use skill grades on team level offense, defense, possession, and fatigue;
- **Strategy and style**: a tactical training or special plays can define a match;
- **Betting odds**: betting odds were used in the proposed model, but to use more than a betting by differents sportsbook;
- **Club budgets**: team budgets can influence directly team performance. The rich team has budgets to buy and pay better players.

Other points that can have improved try to use artificial neural networks and deep learning as well and try a better function or way of optimization the function.

References

[1] Mercado de apostas esportivas movimenta R$ 2 bilhões no Brasil, 2018.
https://www.terra.com.br/noticias/dino/mercado-de-apostas-esportivas-movimenta-r-2-bilhoes-no-brasil-segundo-pesquisa,5e91353bb264cfb927b0b93d8a94e1a397u8wbih.html

[2] Brasileiros perderam US$ 4,1 bilhões em sites de apostas e loterias, 2015.
https://epocanegocios.globo.com/Informacao/Resultados/noticia/2015/09/brasileiros-perderam-us-41-bilhoes-em-sites-de-apostas-e-loterias.html

[3] Premier League Live Scores, Stats & Blog.
www.premierleague.com/stats/top/clubs

[4] Your Football Betting Odds, Football Results, Free Bets, Football Scores Football Betting, Scores & Results Service.
http://www.football-data.co.uk/

[5] Footballpredictions.com Premier League Predictions
https://footballpredictions.com/footballpredictions/premierleaguepredictions/

[6] Bet 365. https://www.bet365.com/

[7] A machine learning framework for sport result prediction: https://www.sciencedirect.com/science/article/pii/S2210832717301485

[8] Gradient boosting :https://en.wikipedia.org/wiki/Gradient_boosting