Machine Learning Engineer Nanodegree

Capstone Project

Dhrubojyoti Jasu
October 21st, 2018

I. Definition

Project Overview

For this capstone project, I will use an English Premier League Football dataset from http://football-data.co.uk/englandm.php . This is a binary classification problem similar to the CharityML in Supervised Learning section of the MLND. The legal sports-betting market in the U.S. was worth an estimated USD270 million in 2017 -- with another USD2.5 billion to USD3 billion in black market betting, according to research firm Eilers & Krejcik Gaming, LLC.

Even though it is not legalized in many other countries like India, but for my own interest, I want to build a predictive model capable of predicting if the home team will win a football match. Usually betting is conducted with human instincts but now we can use some machine learning algorithm to predict the result of the future matches also.

There are reports related to sports prediction using machine learning. Some of them I have listed below:

- Using Machine Learning to Predict the Outcome of English County twenty over Cricket Matches

  "It is possible to predict the winner of English county twenty-twenty cricket games in almost two-thirds of instances."

- https://qz.com/233830/world-cup-germany-argentina-predictions-microsoft/

  For the 2014 World Cup, Bing correctly predicted the outcomes for all of the 15 games in the knockout round.

- Predicting Football Results With Statistical Modelling

  "Something that becomes clear from the results is that Twitter contains enough information to be useful for predicting outcomes in the Premier League"

I am a regular viewer of football around the globe, my favorite club is Manchester United and I follow English Premier League religiously and now I am excited I can use my knowledge of machine learning to have some fun with data.

Problem Statement

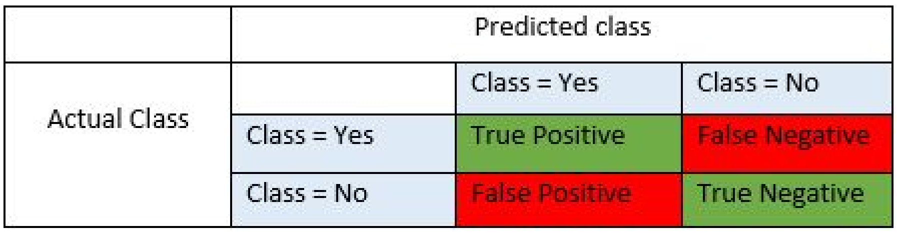
The problem is to use the existing dataset of EPL obtained through http://football-data.co.uk/englandm.php and use it to train some supervised learning algorithms to predict the matches of EPL. I want to predict whether a home team is gonna win the match by training some supervised learning algorithms. This is a pure binary classification problem and II will use a **Logistic Regression** model as a benchmark model. Then I will use **Support Vector Machine** machine algorithm and finally I will implement *Ensemble Learning* to beat my benchmark model score. This dataset has enough points to train a machine learning algorithm & predict results on the data.

Metrics

I will be using accuracy_score & f1_score from sklearn.metrics to evaluate both the benchmark and my final model. The goal of this project is to predict the winner of 'Home Team' accurately. So accuracy as a metric to evaluate a model's performance is appropriate. However, predicting a team is not going to win is not that much important, hence, a model's ability to precisely predict the winner of a 'home team' is *more important* than the model's ability to recall those teams.

For this reason, we can use the f1-score as a metric which considers both precision & recall.

Let’s discuss these two metrics in detail:



	Predicted class		
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

True positive and true negatives are the observations that are correctly predicted and therefore shown in green. We want to minimize false positives and false negatives so they are shown in red color. These terms are a bit confusing. So let’s take each term one by one and understand it fully.

True Positives (TP) - These are the correctly predicted positive values which mean that the value of the actual class is yes and the value of the predicted class is also yes.

E.g. if the actual class value indicates that if a home team is going to win and predicted class tells you the same thing.

True Negatives (TN) - These are the correctly predicted negative values which mean that the value of the actual class is no and value of the predicted class is also no. E.g. if actual class says the home team didn't win and predicted class tells you the same thing.

False positives and false negatives, these values occur when your actual class contradicts with the predicted class.

False Positives (FP) – When actual class is no and predicted class is yes. E.g. if actual class says the home team didn't win but predicted class tells you that this passenger will survive.

False Negatives (FN) – When actual class is yes but predicted class in no. E.g. if the actual class value indicates that the home team wins and predicted class tells you that away team is going to win.

Once you understand these four parameters then we can calculate Accuracy, Precision, Recall and F1 score.

Accuracy - Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost the same. Therefore, you have to look at other parameters to evaluate the performance of your model.

Accuracy = TP+TN/TP+FP+FN+TN

Precision - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labeled as survived, how many actually survived? High precision relates to the low false positive rate.

Precision = TP/TP+FP

Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to all observations in actual class - yes.

Recall = TP/TP+FN

F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have a similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall. I

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

II. Analysis

Exploratory Data Analysis of EPL sessions

In the Datasets I have downloaded all the sessions data separately from the http://football-data.co.uk/englandm.php . Then I consolidated all the datasets into a final dataset called My_Capstone_Dataset.csv file.

Dataset Exploration:

```
# Lets read the data
data = pd.read_csv('My_Capstone_Dataset.csv')

# Remove first 3 matchweeks
# As in first three weeks EPL team's performance are too much unpredictable in nature
# I am removing these datapoints because I don't want to include those results into my analysis

data = data[data.MW > 3]

data.drop(['Unnamed: 0','HomeTeam', 'AwayTeam', 'Date', 'MW', 'HTFormPtsStr', 'ATFormPtsStr', 'FTHG', 'FTAG',
           'HTGS', 'ATGS', 'HTGC', 'ATGC','HomeTeamLP', 'AwayTeamLP','DiffPts','HTFormPts','ATFormPts',
           'HM4','HM5','AM4','AM5','HTLossStreak5','ATLossStreak5','HTWinStreak5','ATWinStreak5',
           'HTWinStreak3','HTLossStreak3','ATWinStreak3','ATLossStreak3'],1, inplace=True)


# Preview data.
display(data.head())
```

	FTR	HTP	ATP	HM1	HM2	HM3	AM1	AM2	AM3	HTGD	ATGD	DiffFormPts	DiffLP
30	H	1.25	1.00	D	D	W	D	W	L	0.50	0.25	0.25	-16.0
31	NH	0.75	0.25	L	L	W	D	L	L	-0.50	-0.75	0.50	-2.0
32	H	1.00	1.00	L	D	W	D	W	L	0.00	0.25	0.00	-3.0
33	NH	0.75	0.50	L	L	W	D	L	D	-0.25	-0.25	0.25	3.0
34	NH	1.00	1.50	D	L	W	W	W	L	0.00	0.75	-0.50	3.0

The original dataset has 6080 data points and 42 input variables, I reduced the dimension to 12 features which are relevant to predict the target variable which is **FTR (Full Time Result.)**

Feature set Explanation:

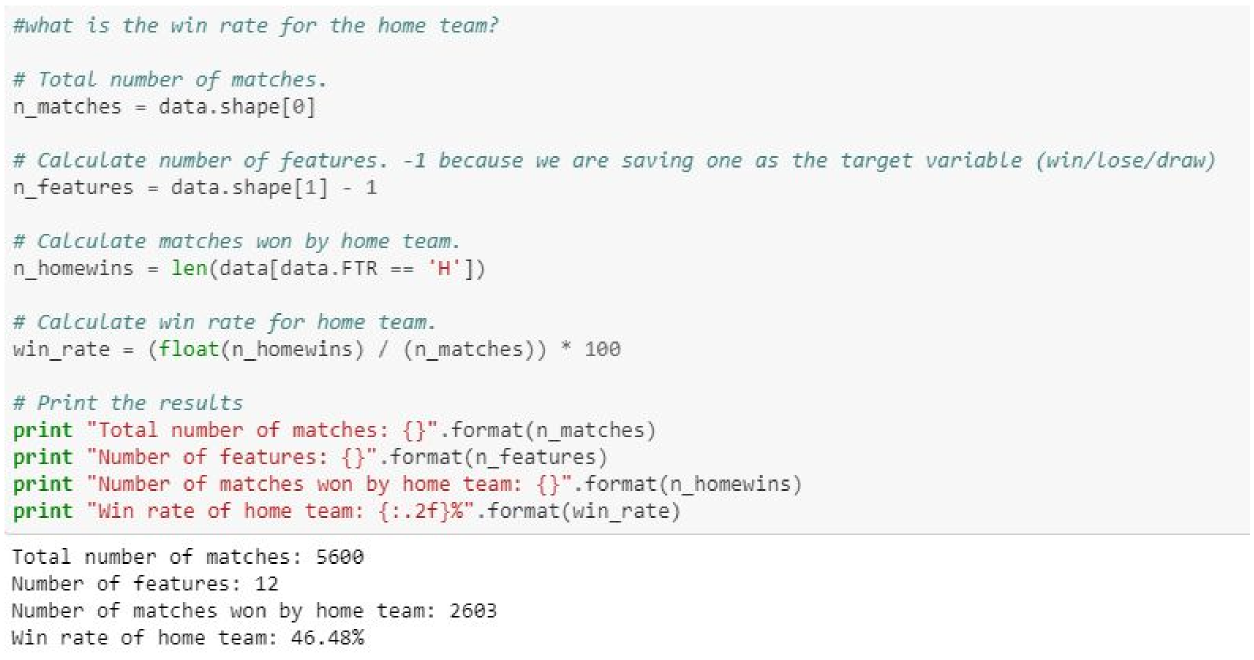
Input Feature Set Explanation

- HTP - Home team points
- ATP - Away team points
- HTGD - Home team goal difference
- ATGD - away team goal difference
- DiffFormPts - Difference in points
- DiffLP - Difference in last years prediction
- HM - Home Match
- AM - Away Match

Target Variable

FTR: Full-Time Result (H=Home Win, D=Draw, A=Away Win)

Let's explore this data a little bit



```
#what is the win rate for the home team?

# Total number of matches.
n_matches = data.shape[0]

# Calculate number of features. -1 because we are saving one as the target variable (win/lose/draw)
n_features = data.shape[1] - 1

# Calculate matches won by home team.
n_homewins = len(data[data.FTR == 'H'])

# Calculate win rate for home team.
win_rate = (float(n_homewins) / (n_matches)) * 100

# Print the results
print "Total number of matches: {}".format(n_matches)
print "Number of features: {}".format(n_features)
print "Number of matches won by home team: {}".format(n_homewins)
print "Win rate of home team: {:.2f}%".format(win_rate)
```

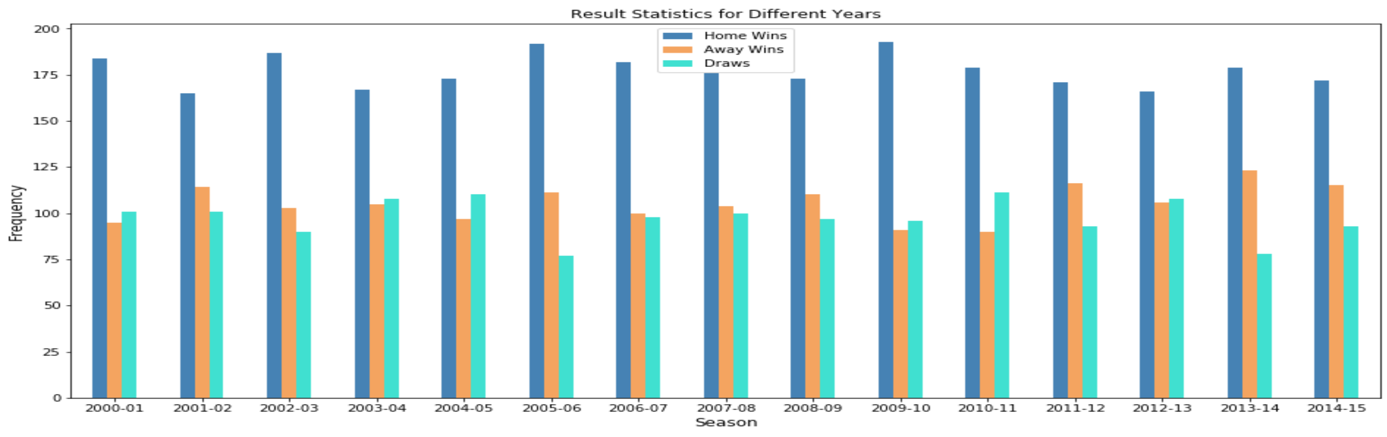
```
Total number of matches: 5600
Number of features: 12
Number of matches won by home team: 2603
Win rate of home team: 46.48%
```

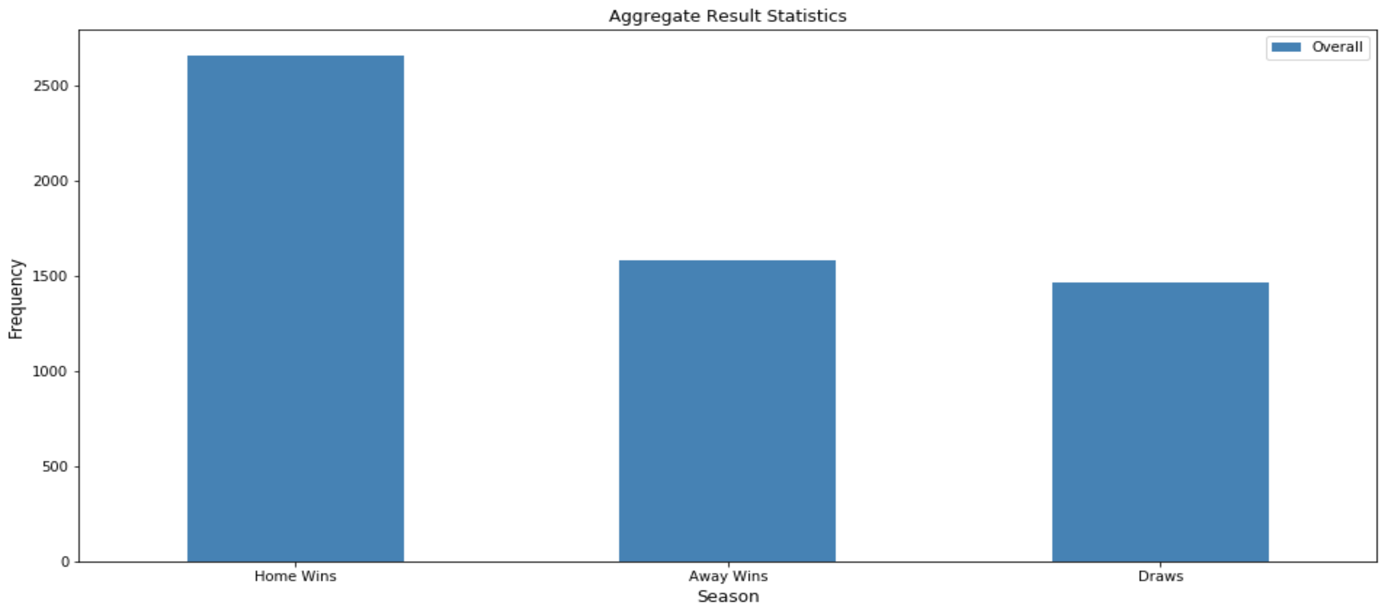
Now I will use all separate sessions CSV files to find answers to some interesting questions listed below:

- How many matches have been won/drawn by home teams?

- What is the win percentage for home teams & away teams?

While searching for above two answers I will create some visualization for clear understanding.



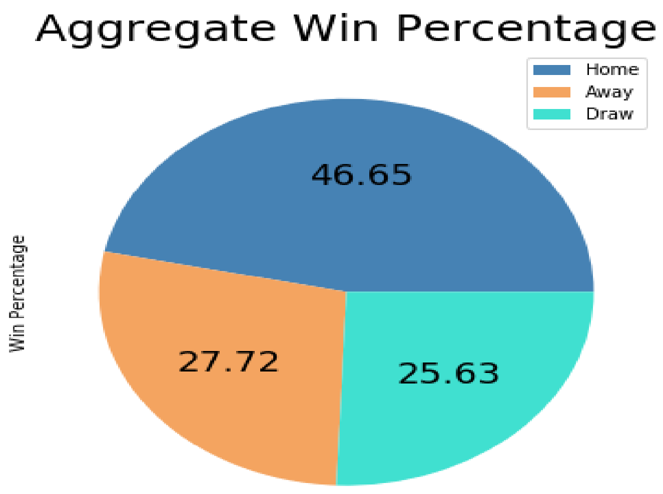


Displaying the overall results

	Home wins	Away Wins	Draws
Overall	2659	1580	1461

So our first question was *How many matches have been won/drawn by home teams?*

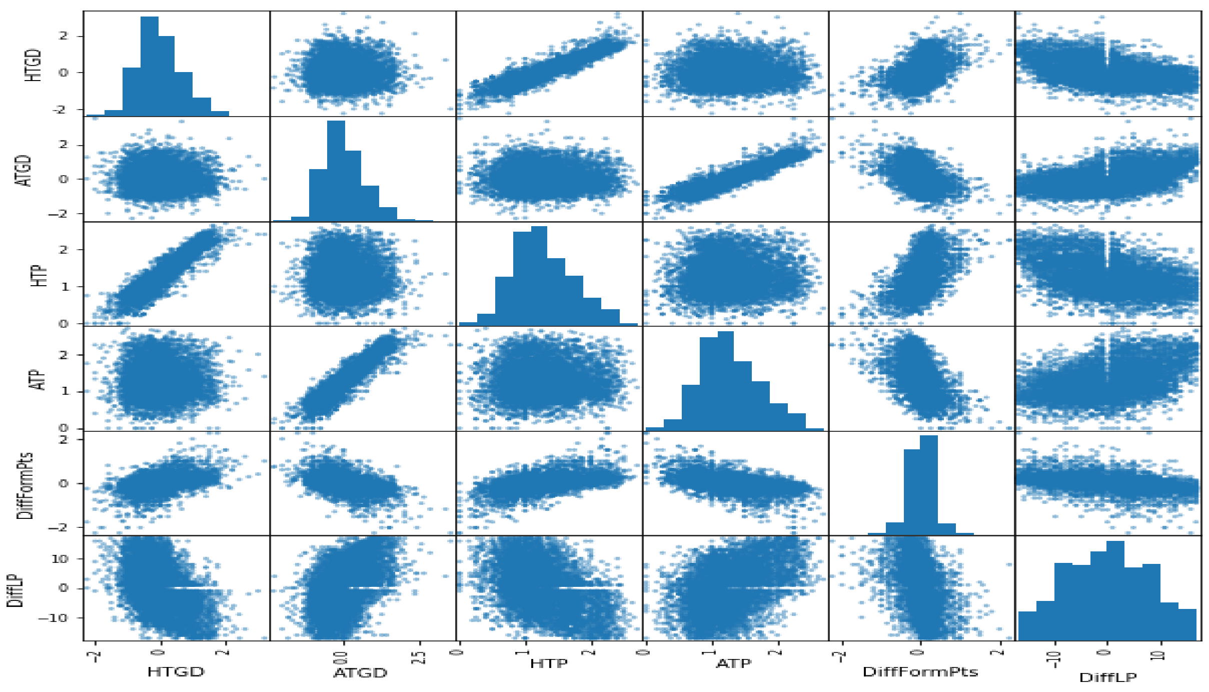
From the above visualization, we can clearly see more matches have won by the home teams only. It is too common in football matches as home team can feel advantage while the home crowd is supporting behind them and pitch conditions and other playing factors are also in favor of them. Now, let's visualize the win percentage for Home & Away teams.



So from above visualization also we can see 46% times Home team wins, 28% time Away team wins and 25% time matches are drawn. So the majority of the times home team usually wins.

Visualizing distribution of data

The scatter matrix is plotting each of the columns specified against each other column. You would have observed that the diagonal graph is defined as a histogram, which means that in the section of the plot matrix where the variable is against itself, a histogram is plotted. Scatter plots show how much one variable is affected by another. The relationship between two variables is called their correlation negative vs positive correlation.



Algorithms & Techniques

Some questions I think are right can be asked which are listed below:

- What model should we use?
- What are the features (the aspects of a game) that matter the most to predicting a team wins?
- Does being the home team give a team the advantage?

I will use three *supervised learning* algorithm listed below to train and predict on my dataset. I will deploy a train_test_split to shuffle & split data into a training & testing set and will compare the metrics of *three* learning algorithms and will choose the best one amongst them. Finally, I will perform a hyperparameter tuning to optimize my final classifier. Then I will try some prediction on my test data set.

Support Vector Machine(SVM):

What is a Support Vector Machine?

“Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyperplane that differentiates the two classes very well. Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

How does SVM works?

Support vector machines focus only on the points that are the most difficult to tell apart, whereas other classifiers pay attention to all of the points.

The intuition behind the support vector machine approach is that if a classifier is good at the most challenging comparisons (the points in B and A that are closest to each other in Figure 2), then the classifier will be even better at the easy comparisons (comparing points in B and A that are far away from each other.

In SVM, it is easy to have a linear hyperplane between two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyperplane. No, SVM has a technique called the kernel trick. These are functions which take low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels. It is most useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then find out the process to separate the data based on the labels or outputs you've defined

Tuning the parameters of SVM:

Tuning parameters value for machine learning algorithms effectively improves the model performance. Let's look at the list of parameters available with SVM

```
sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma=0.0, coef0=0.0, shrinking=True, probability=Fals
e,tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, random_state=None)
```

I am going to discuss one parameter having a higher impact on model performance, "kernel".

we have various options available with the kernel, like, "linear", "rbf", "poly" and others (default value is "rbf"). Here "rbf" and "poly" are useful for non-linear hyper-plane. Let's look at the example, where we've used the linear kernel on two feature of iris data set to classify their class. I will use 'rbf' as my data points are non-linear here and features are only 12.

Reference:

https://stats.stackexchange.com/questions/23391/how-does-a-support-vector-machine-svm-work

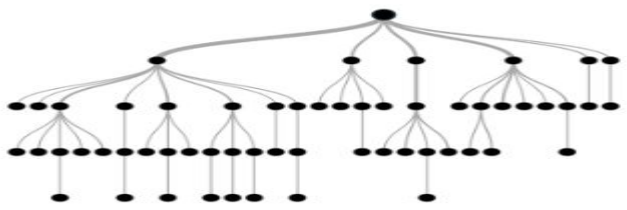
Random Forest:

What is Random Forest?

Random Forest is a versatile machine learning method capable of performing both regression and classification tasks. It also undertakes dimensional reduction methods, treats missing values, outlier values and other essential steps of data exploration, and does a fairly good job. It is a type of ensemble learning method, where a group of weak models combines to form a powerful model.

What is a Decision Tree?

A decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter/differentiator in input variables.



How Does Random Forest Work?

Random forest is like a bootstrapping algorithm with Decision tree (CART) model. Say, we have 1000 observation in the complete population with 10 variables. Random forest tries to build multiple CART model with a different sample and different initial variables. For instance, it will take a random sample of 100 observation and 5 randomly chosen initial variables to build a CART model. It will repeat the process (say) 10 times and then make a final prediction on each observation. Final prediction is a function of each prediction. This final prediction can simply be the mean of each prediction.

Reference:

https://www.analyticsvidhya.com/blog/2014/06/introduction-random-forest-simplified/

XGBoost:

What is boosting?

Boosting is an ensemble technique in which the predictors are not made independently, but sequentially. This technique employs the logic in which the subsequent predictors learn from the mistakes of the previous predictors. Therefore, the observations have an unequal probability of appearing in subsequent models and ones with the highest error appear most. (So the observations are not chosen based on the bootstrap process, but based on the error). The predictors can be chosen from a range of models like decision trees, regressors, classifiers etc. Because new predictors are learning from mistakes committed by previous predictors, it takes less time/iterations to reach close to actual predictions. But we have to choose the stopping criteria carefully or it could lead to overfitting on training data. Gradient Boosting is an example of a boosting algorithm.

What is gradient boosting?

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. (Wikipedia definition).

The objective of any supervised learning algorithm is to define a loss function and minimize it. Let's see how maths work out for Gradient Boosting algorithm. Say we have the mean squared error (MSE) as loss defined as:

$$Loss = MSE = \sum (y_i - y_i^p)^2$$

where, y_i = ith target value, y_i^p = ith prediction, $L(y_i, y_i^p)$ is Loss function

We want our predictions, such that our loss function (MSE) is minimum. By using gradient descent and updating our predictions based on a learning rate, we can find the values where MSE is minimum.

$$y_i^p = y_i^p + \alpha * \delta \sum (y_i - y_i^p)^2 / \delta y_i^p$$

which becomes, $y_i^p = y_i^p - \alpha * 2 * \sum (y_i - y_i^p)$

where, α is learning rate and $\sum (y_i - y_i^p)$ is sum of residuals

So, we are basically updating the predictions such that the sum of our residuals is close to 0 (or minimum) and predicted values are sufficiently close to actual values.

What is XGBoost?

XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost stands for eXtreme Gradient Boosting.

As Tianqi Chen said:

> The name xgboost, though, actually refers to the engineering goal to push the limit of computations resources for boosted tree algorithms. Which is the reason why many people use xgboost

Generally, XGBoost is fast. Really fast when compared to other implementations of gradient boosting.

Reference:
https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

Benchmark

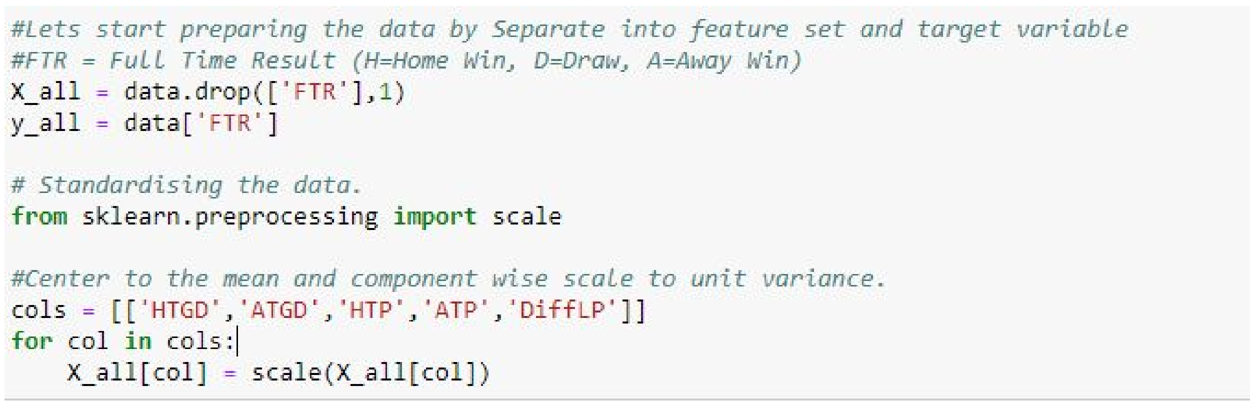
I will use an untuned Logistic Regression classifier from 'sklearn' to benchmark my final result. When the outcome (dependent variable) has only a limited number of possible values (in our cases hometeam win or loss), Logistic Regression is used when the response variable is categorical in nature.

III. Methodology

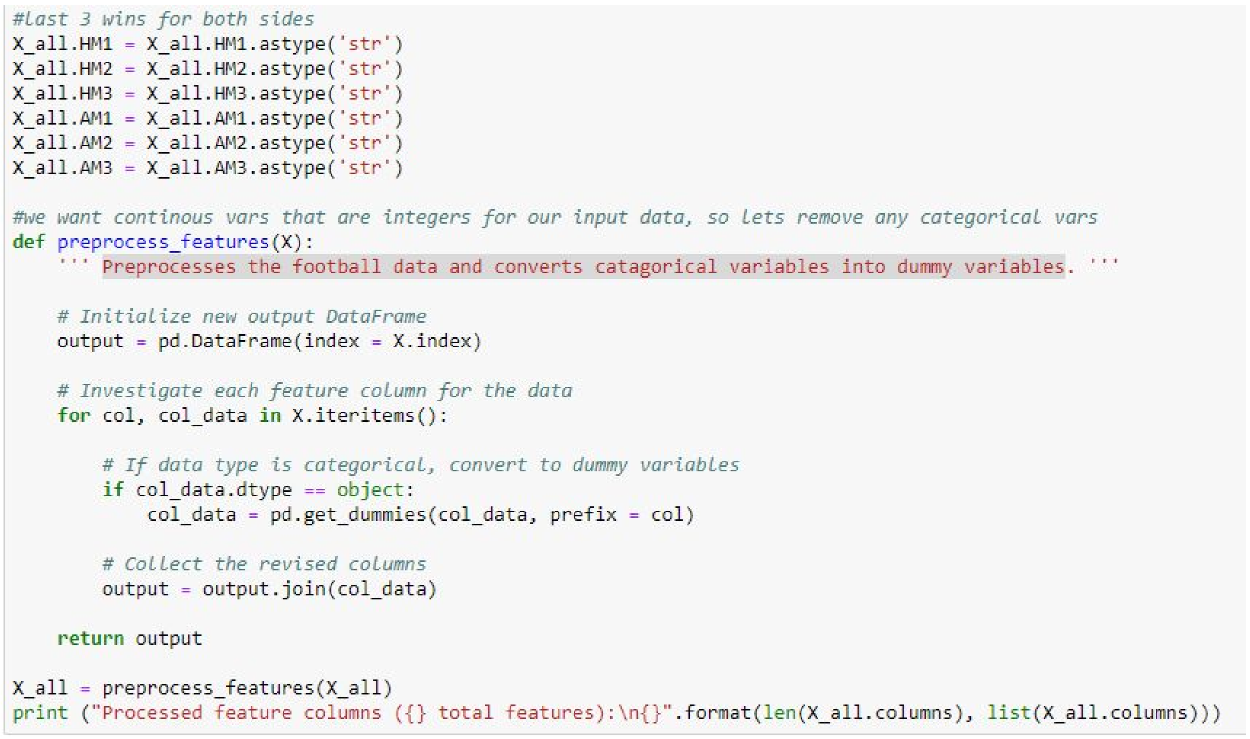
Data Preprocessing:

- Here the data is prepared by Separate into feature set and the target variable FTR = Full-Time Result (H=Home Win, D=Draw, A=Away Win)
- I have preprocessed the football data and converts categorical variables into dummy variables

Below are my code snippet from `My_Capstone_Report.ipynb` file.



```python
#Lets start preparing the data by Separate into feature set and target variable
#FTR = Full Time Result (H=Home Win, D=Draw, A=Away Win)
X_all = data.drop(['FTR'],1)
y_all = data['FTR']

# Standardising the data.
from sklearn.preprocessing import scale

#Center to the mean and component wise scale to unit variance.
cols = [['HTGD','ATGD','HTP','ATP','DiffLP']]
for col in cols:
    X_all[col] = scale(X_all[col])
```

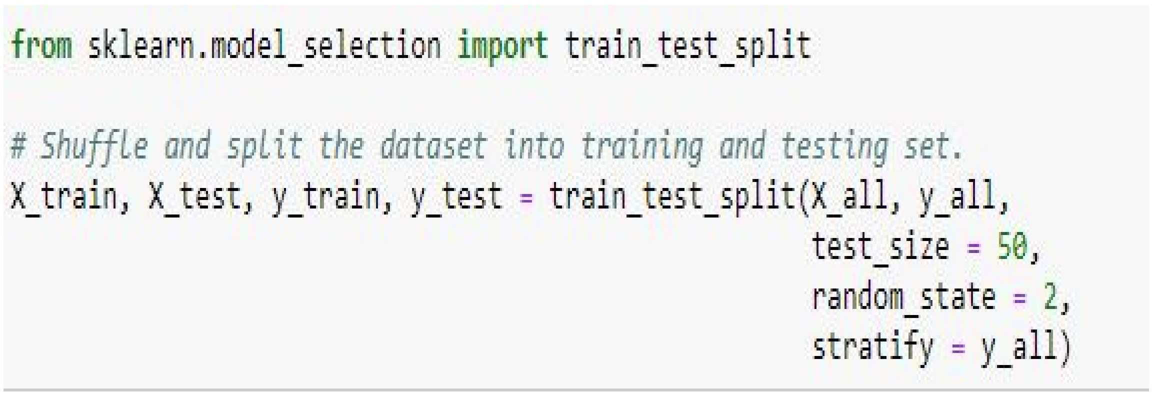


```python
#last 3 wins for both sides
X_all.HM1 = X_all.HM1.astype('str')
X_all.HM2 = X_all.HM2.astype('str')
X_all.HM3 = X_all.HM3.astype('str')
X_all.AM1 = X_all.AM1.astype('str')
X_all.AM2 = X_all.AM2.astype('str')
X_all.AM3 = X_all.AM3.astype('str')

#we want continous vars that are integers for our input data, so lets remove any categorical vars
def preprocess_features(X):
    ''' Preprocesses the football data and converts catagorical variables into dummy variables. '''

    # Initialize new output DataFrame
    output = pd.DataFrame(index = X.index)

    # Investigate each feature column for the data
    for col, col_data in X.iteritems():

        # If data type is categorical, convert to dummy variables
        if col_data.dtype == object:
            col_data = pd.get_dummies(col_data, prefix = col)

        # Collect the revised columns
        output = output.join(col_data)

    return output

X_all = preprocess_features(X_all)
print ("Processed feature columns ({} total features):\n{}".format(len(X_all.columns), list(X_all.columns)))
```

Output:

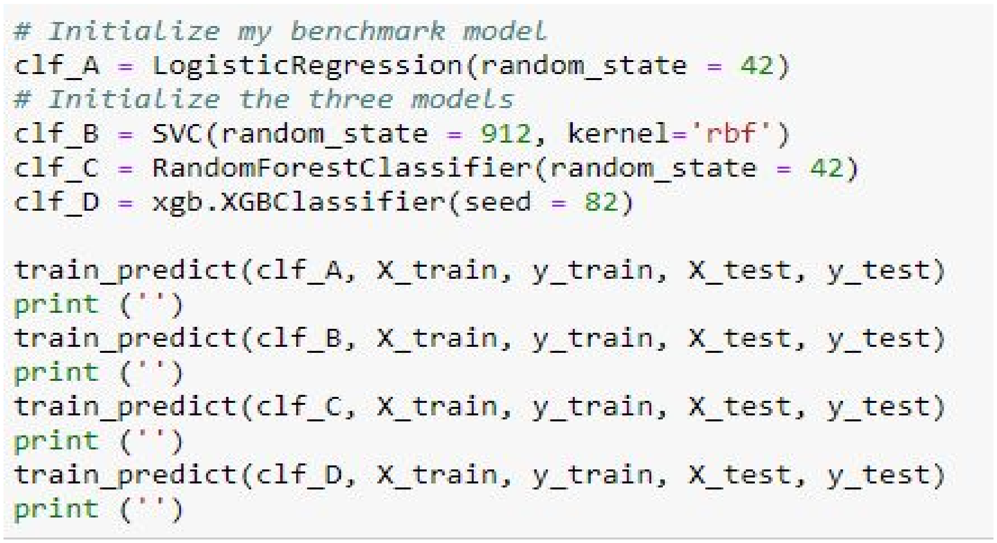
```
Processed feature columns (24 total features):
['HTP', 'ATP', 'HM1_D', 'HM1_L', 'HM1_W', 'HM2_D', 'HM2_L', 'HM2_W',
'HM3_D', 'HM3_L', 'HM3_W', 'AM1_D', 'AM1_L', 'AM1_W', 'AM2_D', 'AM2_L',
'AM2_W', 'AM3_D', 'AM3_L', 'AM3_W', 'HTGD', 'ATGD', 'DiffFormPts',
'DiffLP']
```

Shuffle & Split Data:



```
from sklearn.model_selection import train_test_split

# Shuffle and split the dataset into training and testing set.
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all,
                                                    test_size = 50,
                                                    random_state = 2,
                                                    stratify = y_all)
```

Implementation: Initial Model Evaluation



```
# Initialize my benchmark model
clf_A = LogisticRegression(random_state = 42)
# Initialize the three models
clf_B = SVC(random_state = 912, kernel='rbf')
clf_C = RandomForestClassifier(random_state = 42)
clf_D = xgb.XGBClassifier(seed = 82)

train_predict(clf_A, X_train, y_train, X_test, y_test)
print ('')
train_predict(clf_B, X_train, y_train, X_test, y_test)
print ('')
train_predict(clf_C, X_train, y_train, X_test, y_test)
print ('')
train_predict(clf_D, X_train, y_train, X_test, y_test)
print ('')
```

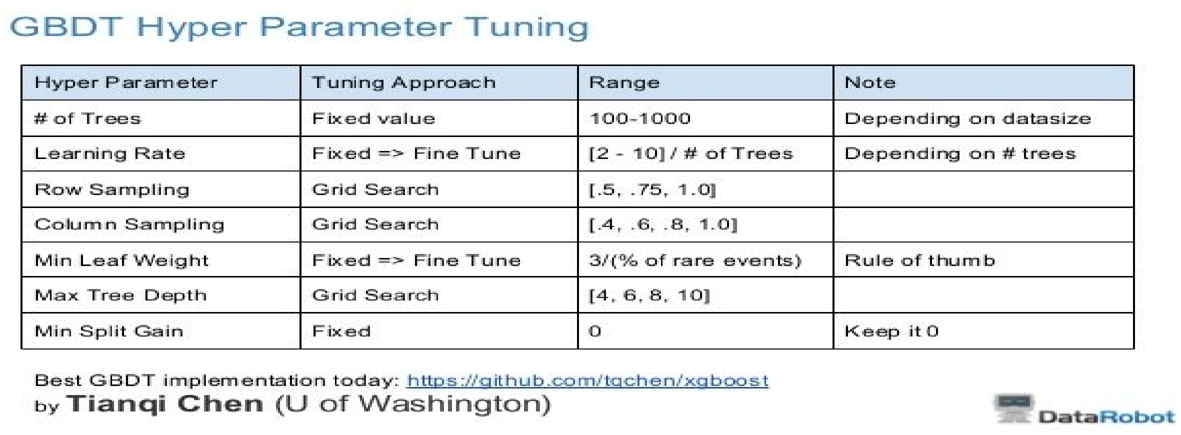
Training a LogisticRegression using a training set size of 5550. . . Trained model	Training a SVC using a training set size of 5550. . . Trained model in 1.8139	Training a RandomForestClassifier using a training set size of 5550. . . Trained model	Training a XGBClassifier using a training set size of 5550. . . Trained model

in 0.0260 seconds Made predictions in 0.0040 seconds. 0.6215610352557571 0.6654054054054054 F1 score and accuracy score for training set: 0.6216 , 0.6654. Made predictions in 0.0000 seconds. F1 score and accuracy score for test set: 0.6957 , 0.7200.	seconds Made predictions in 1.0184 seconds. 0.6204535729567822 0.6803603603603604 F1 score and accuracy score for training set: 0.6205 , 0.6804. Made predictions in 0.0110 seconds. F1 score and accuracy score for test set: 0.6818 , 0.7200.	in 0.0799 seconds Made predictions in 0.0120 seconds. 0.9853789919199692 0.9863063063063063 F1 score and accuracy score for training set: 0.9854 , 0.9863. Made predictions in 0.0020 seconds. F1 score and accuracy score for test set: 0.6923 , 0.6800.	in 0.4477 seconds Made predictions in 0.0220 seconds. 0.6521471132114238 0.694954954954955 F1 score and accuracy score for training set: 0.6521 , 0.6950. Made predictions in 0.0020 seconds. F1 score and accuracy score for test set: 0.7451 , 0.7400.

Refinement

- From the above result, my benchmark model *Logistic Regression's* f1 score & accuracy score on the test set is 0.6957, 0.7200 respectively.
- Even though Random Forest performed very well on training data but failed on test data
- It is clearly visible from the above result that XgBoost is the best model for this problem.

Tuning the parameters of XGBoost



GBDT Hyper Parameter Tuning

Hyper Parameter	Tuning Approach	Range	Note
# of Trees	Fixed value	100-1000	Depending on datasize
Learning Rate	Fixed => Fine Tune	[2 - 10] / # of Trees	Depending on # trees
Row Sampling	Grid Search	[.5, .75, 1.0]	
Column Sampling	Grid Search	[.4, .6, .8, 1.0]	
Min Leaf Weight	Fixed => Fine Tune	3/(% of rare events)	Rule of thumb
Max Tree Depth	Grid Search	[4, 6, 8, 10]	
Min Split Gain	Fixed	0	Keep it 0

Best GBDT implementation today: https://github.com/tqchen/xgboost
by **Tianqi Chen** (U of Washington)

DataRobot

Code Snippet:

```
# Import 'GridSearchCV' and 'make_scorer'
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer


# Create the parameters list you wish to tune
parameters = { 'learning_rate' : [0.1],
               'n_estimators' : [40],
               'max_depth': [3],
               'min_child_weight': [3],
               'gamma':[0.4],
               'subsample' : [0.8],
               'colsample_bytree' : [0.8],
               'scale_pos_weight' : [1],
               'reg_alpha':[1e-5]
             }

# Initialize the classifier
clf = xgb.XGBClassifier(seed=2)

# Make an f1 scoring function using 'make_scorer'
f1_scorer = make_scorer(f1_score,pos_label='H')

# Perform grid search on the classifier using the f1_scorer as the scoring
method
grid_obj = GridSearchCV(clf,
                        scoring=f1_scorer,
                        param_grid=parameters,
                        cv=5)

# Fit the grid search object to the training data and find the optimal
parameters
grid_obj = grid_obj.fit(X_train,y_train)
```

```
# Get the estimator
clf = grid_obj.best_estimator_
print (clf)

# Report the final F1 score for training and testing after parameter tuning
f1, acc = predict_labels(clf, X_train, y_train)
print ("F1 score and accuracy score for training set: {:.4f} ,
{:.4f}.".format(f1 , acc))

f1, acc = predict_labels(clf, X_test, y_test)
print ("F1 score and accuracy score for test set: {:.4f} , {:.4f}.".format(f1
, acc))
```

Final Model Evaluation

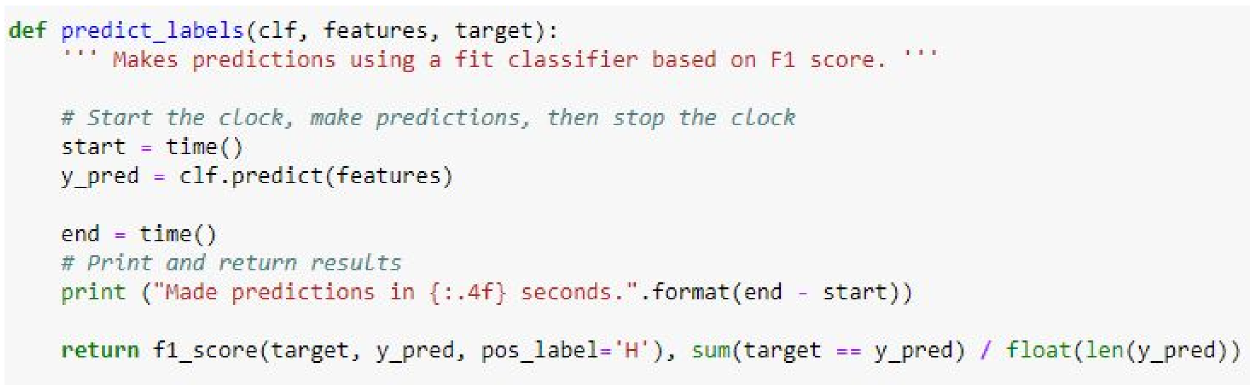
Let's find out the difference between optimized & unoptimized metrics of my final XGBoost classifier on the test set

Metric	Unoptimized Model	Optimized Model
Accuracy Score	0.74	0.80
F-Score	0.74	0.78

My **benchmark model *Logistic Regression*** f_score & Accuracy Score was 0.6957, 0.7200 respectively on the test set.

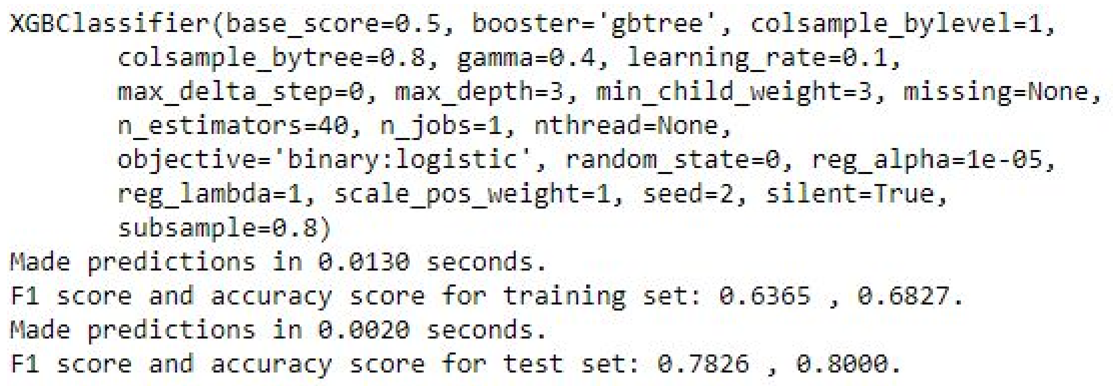
IV. Results

In *training & predicting pipeline*, I have created a function to create f1-score for us.



```
def predict_labels(clf, features, target):
    ''' Makes predictions using a fit classifier based on F1 score. '''

    # Start the clock, make predictions, then stop the clock
    start = time()
    y_pred = clf.predict(features)

    end = time()
    # Print and return results
    print ("Made predictions in {:.4f} seconds.".format(end - start))

    return f1_score(target, y_pred, pos_label='H'), sum(target == y_pred) / float(len(y_pred))
```

My final tuned XGBoost classifier is producing below result:



```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.8, gamma=0.4, learning_rate=0.1,
       max_delta_step=0, max_depth=3, min_child_weight=3, missing=None,
       n_estimators=40, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=1e-05,
       reg_lambda=1, scale_pos_weight=1, seed=2, silent=True,
       subsample=0.8)
Made predictions in 0.0130 seconds.
F1 score and accuracy score for training set: 0.6365 , 0.6827.
Made predictions in 0.0020 seconds.
F1 score and accuracy score for test set: 0.7826 , 0.8000.
```

On test set, accuracy score is **80%** and we can easily validate my final trained classifer is performing better than any other classifier I introduced earlier. Only RandomForest classifier was performing well on training set but on test set my final XGBoost model is more optimized and which is more important for the prediction job.

Justification

By using my final model we can predict the results of an EPL game which is statistically more significant than pure guessing the result of a match with human instinct.

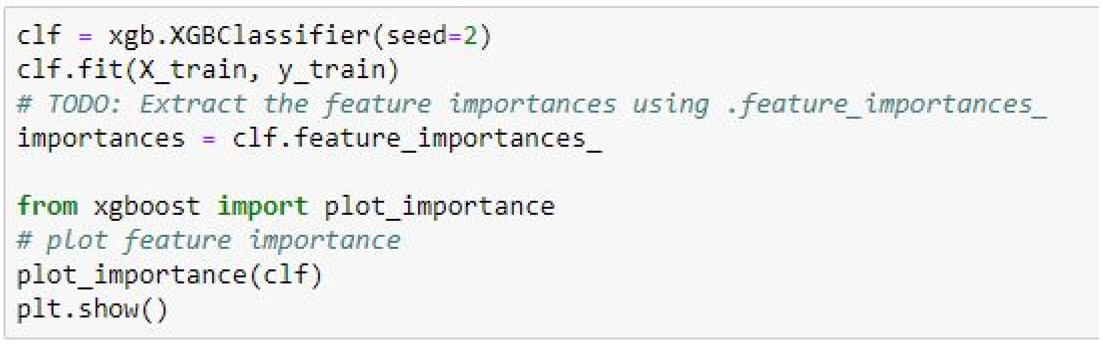
Conclusion

Free Form Visualization

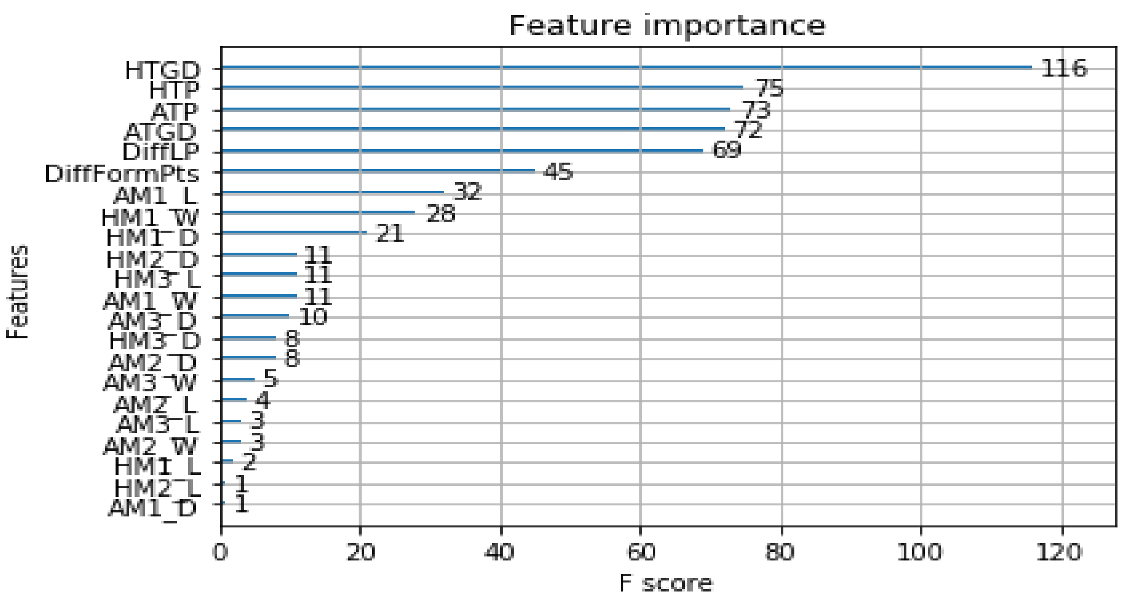
While exploring the data I guess below three features are more important to predict the future match result:

- HTP - Home team points
- HTGD - Home team goal difference
- DiffFormPts - Difference in points

Let me check with the feature importance of XGBClassifier.



```
clf = xgb.XGBClassifier(seed=2)
clf.fit(X_train, y_train)
# TODO: Extract the feature importances using .feature_importances_
importances = clf.feature_importances_

from xgboost import plot_importance
# plot feature importance
plot_importance(clf)
plt.show()
```



Seems like **HTGD(Home Team Goal Difference)** is the most determining factor in our future match prediction. **HTP(Home Team Points)** and **ATP(Away Team Points)** also have a higher importance to predict **FTR.** It is true in every football match if one team has a higher goal difference then it means the team is scoring more goals in regular interval. By using the above visualization we can cleary see this feature is a determining factor to predict the result of a match

Reflection

- In this project first I performed exploratory data analysis(EDA) with some simple visualization where it was clearly displayed that in EPL home team wins proportion is higher than draw & loss.
- I have chosen three classifiers to train & predict my data. Surprisingly Random Forest gave me a very high f1-score & accuracy score on training data but performed worst relatively on testing data. This is a very common scenario of *high train score, low test score*. RandomForest learned well from the training data but didn't perform well on testing data as compared tp XGBoost classifier. So I choose XGBoost as my final classifier.
- Pure guessing also can give some good result while predicting match results, but using my final trained model is more statistically significant for better prediction.

Improvement

- Possible improvement can be a web app taking the input of the two teams & predicting the winner based on the model trained above.

- We can implement more data features like twitter sentiment analysis on pre-match analysis by training a deep neural network. For this purpose, we can use millions of data points affecting a football match result. In FIFA 2018 world cup bing tried a similar thing by predicting some matches result accurately.
- Possibly a neural network with a much bigger feature rich dataset can predict more accurately than my final model.