Usefulness of Ensembled Methods for creating xG model

Hemant Banke, Sayantan Deb Barman, Soham Majumdar

19/06/2022

Introduction:

Football is still far behind most other sports in the matters of analytics advancements, but every year more and more teams are starting to invest money and resources in it. One of the reasons why football is behind basketball and baseball on sports analytics is because it is much harder to get access to data. Although some data providers such as StatsBomb are starting to release public datasets to try to draw more people into working in this field, it is still expensive to get access to large and complete datasets. Another issue is that the structure of the game of football is continuous, unlike the stop-start nature of other sports such as baseball or basketball, and therefore the collection and quantification of metrics becomes that much more difficult.
Techniques to predict the outcome of professional football matches have traditionally used the number of goals scored or the number of shots taken by each team as a base measure for evaluating a team's performance and estimating future results. However, the number of goals scored during a match possesses an important random element which leads to large inconsistencies in many games between a team's performance and number of goals scored or conceded. The development of more advanced metrics can help give more precise insights into the data.

What are Expected Goals?

Expected goals (or xG) measures the quality of a chance by calculating the likelihood that it will be scored from a particular position on the pitch during a particular phase of play. This value is based on several factors from before the shot was taken. xG is measured on a scale between zero and one, where zero represents a chance that is impossible to score and one represents a chance that a player would be expected to score every single time.
That is why this project aims to develop an xG model.
In this project, we develop a model 'Expected Goals' metric to provide us with a more precise and accurate way to evaluate a team's performance, instead of using the actual number of goals scored. We start with building some basic models using Logistic Regression and Decision Trees and then improve upon these by using different Ensembling methods such as bagging, boosting and stacking and compare the performances of each of them.

Objective:

Building and comparing the performance of expected goals (xG) model created using Ensemble Methods, using it to predict the outcome of a football match.

Background:

Logistic Regression:

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is dichotomous, which means there would be only two possible classes. Here we will use a Binary Logistic Regression. In Binary Logistic regression, the dependent variable is binary in nature having data coded as either 1 (stands for success/yes) or 0 (stands for failure/no).
In our model, 1 denotes that the shot results in a goal and 0 denotes that the shot doesn't result in a goal.
The explanatory variables may be continuous or categorical.
Suppose we have p explanatory variables that are continuous, then the model can be written as:

$$log(\frac{\pi}{1-\pi}) = \alpha + \beta_1 x_1 + \ldots + \beta_p x_p$$

where π denotes the probability of success.
Suppose we have r explanatory variables in the model that are categorical, with the kth variable having q_k classes, coded as 1,2,...,q_k, then the model can be written as:

$$log(\frac{\pi}{1-\pi}) = \alpha + \gamma_{11} I(Z_1 = 1) + \ldots + \gamma_{1(q_1-1)} I(Z_1 = q_1 - 1) + \ldots + \gamma_{r1} I(Z_r = 1) + \ldots + \gamma_{r(q_r-1)} I(Z_r = q_r - 1)$$

Here for each of the explanatory variables, the last class acts as the reference class.
Any model which contains both continuous and categorical explanatory variables, we write as a combination of the last two models.
Assumptions of Logistic Regression: We must be aware of the following assumptions while fitting a logistic regression –

1> In case of binary logistic regression, the target variables must be binary always and the desired outcome is represented by the factor level 1.

2> There should not be any multi-collinearity in the model, which means the independent variables must be independent of each other.

3> We must include meaningful variables in our model. Hence variables which have NA as their values must be removed before fitting a logistic regression.

We should choose a large sample size for logistic regression.

Penalized Logistic Regression:

Penalized logistic regression imposes a penalty to the logistic model for having too many variables. This results in shrinking the coefficients of the less contributive variables toward zero. This also helps to solve the problem of multicollinearity that may seep into our data, which in turn violates the assumptions of a Logistic Regression.
The most commonly used penalized regression include:

Ridge regression: variables with minor contribution have their coefficients close to zero. However, all the variables are incorporated in the model. This is useful when all variables need to be incorporated in the model according to domain knowledge.
Lasso regression: the coefficients of some less contributive variables are forced to be exactly zero. Only the most significant variables are kept in the final model.
Elastic net regression: the combination of ridge and lasso regression. It shrinks some coefficients toward zero (like ridge regression) and set some coefficients to exactly zero (like lasso regression).

The objective function for logistic regression is the penalized negative binomial log-likelihood and is:

$$\min_{(\beta_0,\beta)\in R^{p+1}} -[\frac{1}{N}\sum_{i=1}^{N} y_i(\beta_0 + x_i^T\beta) - log(1 + e^(\beta_0 + x_i^T\beta))] + \lambda[(1-\alpha)||\beta||_2^2/2 + \alpha||\beta||_1)]$$

α: the elastic net mixing parameter. Allowed values include:
“1”: for lasso regression
“0”: for ridge regression
a value between 0 and 1 (say 0.3) for elastic net regression.
λ: a numeric value defining the amount of shrinkage.

Decision Trees:

A decision tree is a supervised learning technique that involves stratifying or segmenting the predictor space into a number of simple regions. Decision trees can be applied to both regression problems and classification problems.
In our case, we will deal with only the classification problem.

Classification Trees
A classification tree is used to predict a qualitative response. For making a decision tree, we usually perform recursive binary splitting. If the predictor variable is continuous, we can divide the prediction space into regions like $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$. That is, for any j and s, we define the pair of half-planes
$R_1(j,s) = \{X|X_j < s\}$ and $R_2(j,s) = \{X|X_j \geq s\}$
A natural alternative to RSS as a splitting criterion is the classification error rate. However, it turns out that this is not sufficiently sensitive for tree-growing, and in practice other measures such as the Gini index or alternatively, *entropy* are preferred. We seek the value of j and s that minimize the Gini index, which is defined as

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

where $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class. The total number of classes is K.
The Gini index takes small values if $\hat{p}_{mk}$ is close to 0 or 1.
The predictor variable can be qualitative instead of being continuous. Suppose a predictor X_j has p levels, then we can divide the predictor space into 2 regions similarly as: $R_1(j) = \{X|X_j \in \{x_1, x_2, .., x_r\}\}$ and $R_2(j) = \{X|X_j \in \{x_{r+1}, .., x_s\}\}$
where $x_i's$ denotes the levels of the predictor variable X_j.

Bagging

Bagging, also known as Bootstrap aggregation, is a general purpose procedure for reducing the variance of a statistical learning method.
The decision trees discussed previously suffer from high variance. This means that if we split the training data randomly into two parts and fit a decision tree on each part, we will get very different fits.
A solution to this problem is by using Bagging.
If we draw a set of n iid observations $Z_1, .., Z_n$ each having variance σ^2, then the variance of the mean of these random variables is given by σ^2/n, i.e. using mean reduces the variance.
Bagging enables us to extend the same idea to Decision tree classifier. We take repeated samples from a single training data set. Thus we generate B bootstrapped training sets. We then train the decision tree for all the B bootstrapped datasets. For a given test observation, we record the class predicted by each of the B decision trees. Then we use the method of majority vote, i.e. the overall prediction is the most commonly occurring class among the B predictions.

Random Forest:

Random Forests use the idea of Bagging, except that it makes a small tweak in the bagging algorithm that *decorrelates* the decision trees.
As in bagging, random forests consider a number of decision trees based on bootstrapped training samples. But while making these trees, a random sample of m predictors is considered instead of the full sample of p predictors. Typically, $m \approx \sqrt{p}$.
The purpose of this is suppose we have a very strong predictor in the data along with some moderately strong predictors, then in case of bagging, almost all the trees will have the strong predictor at the top split. Consequently, all the trees would look similar. And averaging over highly correlated quantities do not lead to a large reduction in variance. Considering a random sample of only m of the predictors, we decorrelate the decision trees.

Boosting:

Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees. Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set. Essentially, this technique attempts to build a strong classifier from a number of weak classifiers. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added.

AdaBoost:

There are two forms of AdaBoost algorithm that we will go into- AdaBoost.M1 and Real AdaBoost.
AdaBoost.M1: AdaBoost.M1 is the most popular boosting algorithm due to Freund and Schapire (1997). Consider a two class problem with output coded as $Y \in -1, 1$. Given a vector of predictor variables X, a classifier G(X) produces a prediction taking one of the two values {-1,1}. The error rate on the training sample is:

$$err = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq G(x_i))$$

Algorithm for AdaBoost.M1
1> Initialize the observation weights $w_i = 1/N$, i=1,2,..,N
2> For m=1,2,..,M (where M denotes the number of iterations)
a> Fit a classifier (here a decision tree classifier) $G_m(x)$ to the training data using weights w_i.
b> Compute

$$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$

c> Compute $\alpha_m = log((1 - err_m)/err_m)$
d> Set $w_i = w_i . exp[\alpha_m . I(y_i \neq G_m(x_i))], i = 1, 2, . . , N$
3> Output $G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$
The last line of the algorithm combines the predictions of all the weak classifiers through a weighted majority vote to produce the final prediction. Here $\alpha_1, . . , \alpha_m$ are computed by the boosting algorithm.
Observe that $\alpha_m > 0$ if $err_m < 0.5$ i.e. any classifier that works better than a classifier based purely on chance should have $\alpha_m > 0$.
Consequently, we see that the weights of the observations that are misclassified for such a classifier get increased by a factor of $exp[\alpha_m . I(y_i \neq G_m(x_i))]$ while the weights of the observations that were classified correctly remain unchanged. Thus in subsequent iterations, more weightage is given to observations that are misclassified and less weightage is given to observations that are classified correctly.

Real AdaBoost

If instead of returning discrete prediction (as in case of AdaBoost.M1), the base classifiers return real valued predictions (eg a probability mapped to the interval [-1,1]), then the boosting algorithm can be modified appropriately to what is known as Real AdaBoost.
Algorithm for Real AdaBoost:
1> Initialize the observation weights $w_i = 1/N$, i=1,2,..,N
2> For m=1,2,..,M (where M denotes the number of iterations)
a> Fit the classifier to obtain a class probability estimate $p_m(x) = \hat{P}_w(y = 1|x) \in [0, 1]$, using weights w_i on the training data.
b> Set $f_m(x) = \frac{1}{2} log(p_m(x)/(1 - p_m(x)) \in R$ (contribution to the final model).
c> Set $w_i = w_i exp[-y_i f_m(x_i)]$, i=1,2,..,N, and renormalize that $\sum_{i=1}^{N} w_i = 1$
3> Output the classifier $sign[\sum_{m=1}^{M} f_m(x)]$
We observe that $f_m(x)$ is positive if $p_m(x) > 0.5$. For $y_i = 1$, if $p_m(x_i) > 0.5$, we understand that the base classifier is classifying the observation to class {Y=1} with a high probability i.e. it is classifying correctly. In this case, the weights are altered by a factor of $exp[-y_i f_m(x_i)]$, which becomes <1. That is the weight of the observation classified correctly gets reduced in the subsequent iterations.
Similarly, we can observe that for observations that are misclassified, $exp[-y_i f_m(x_i)]$ becomes >1. Therefore, the weights of such observations in the subsequent iterations increase.

Gradient Boost

Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error. In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.
Now we will go into details in the mathematics that goes into gradient boosting.
A gradient boost involves three main components: loss function, weak learner andadditive model.
Loss function: The role of the loss function is to estimate how good the model is at making predictions with the given data. Our objective would be to minimise the loss function.
Weak Learner - A weak learner is one that classifies our data but does so poorly.
Additive Model - This is the iterative and sequential approach of adding the weak learners one step at a time. After each iteration, we need to be closer to our final model. In other words, each iteration should reduce the value of our loss function.

We are provided with data in the form of (y_i, x_i), i=1,..,n where the $x_i's$ are the input variables that we feed into our model and $y_i's$ are the target variables that we are trying to predict. The $y_i's$ are in the form of 0's and 1's.
The log likelihood of the observed data can be written as:

$$y_i * log\ p + (1 - y_i) * log\ (1 - p)$$

where p is the predicted probability
The goal should be to maximise the log likelihood, hence to minimise -(log likelihood). Hence it makes sense to take -(log likelihood) as our loss function.
In terms of log odds, this can be written as:

$$-y_i\ log(odds) + log(1 + e^{log(odds)})$$

This function is differentiable and on differentiating this we get:

$$\frac{d}{d(log\ odds)}(-y_i\ log(odds) + log(1 + e^{log(odds)})) = -y_i + \frac{e^{log(odds)}}{1 + e^{log(odds)}}$$

This can also be written as -Observed+Predicted
Now that we have found our loss function we can dive into the algorithm.
1> Initialize $f_0(x) = arg\ min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2> For m=1,..,M
a> For i=1,2,..,n compute:

$$r_{im} = -[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f=f_{m-1}}$$

We have seen that $-[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]$= Observed-Predicted. Hence this quantity may act as pseudo-residuals, which serve as the input of our next regression tree.
b> Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, j=1,2,..,J_m
c> Each terminal region may contain several observations, hence we need to combine them to get a common predicted probability. For j=1,2,.., J_m compute

$$\gamma_{jm} = arg\ min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

d> Update $f_m(x) = f_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$ where ν denotes the learning rate.
Learning Rate is used to scale the contribution from the new tree. This results in a small step in the right direction of prediction. Empirical evidence has proven that taking lots of small steps in the right direction results in better prediction with a testing dataset i.e the dataset that the model has never seen as compared to the perfect prediction in 1st step. Learning Rate is usually a small number like 0.1
3> Output $\hat{f}(x) = f_M(x)$
This gives us the log(odds). To predict the probability of classifying the object into class 1, we use the formula:

$$P = \frac{e^{log(odds)}}{1 + e^{log(odds)}}$$

If this probability is greater than the threshold, we classify the observation into class 1, else we classify it into class 2.

Boosted Logistic Regression:

We can boost the Logistic Regression to obtain better results. We are fitting additive logistic regression by stagewise optimization of Bernoulli log-likelihood. Here we will only consider 2 classes.

LogitBoost (2 classes, population version) :
The LogitBoost Algorithm uses Newton Steps for fitting an additive Logistic model by maximum likelihood approach. If response takes values 0/1 for each outcome, we represent probability of y = 1 by,

$$p(x) = \frac{e^{F(x)}}{e^{F(x)} + e^{-F(x)}}$$

where $F(x)$ is the additive model of form $\sum_{m=1}^{M} c_m f_m(x)$.

The algorithm is as follows :
1> Let y be the response. y=-1 if the observation belongs to Class 1 and y=1 if the observation belongs to Class 2. We define a new variable y'= (y+1)/2. y' takes the values 0 and 1.
2> Start with weights $w_i = 1/N$, i=1,2,...,N, F(x)=0 and probability estimates $p(x_i) = 0.5$.
3> Repeat for m=1,2,..,M:
(a) Compute the working response and weights

$$z_i = \frac{y_i' - p(x_i)}{p(x_i)(1 - p(x_i))}$$

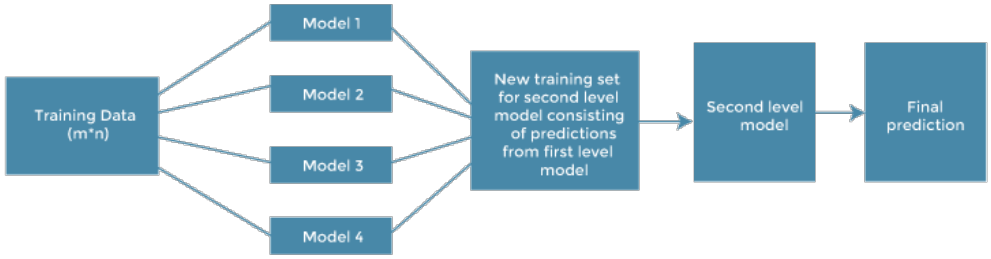
$$w_i = p(x_i)(1 - p(x_i))$$

(b) Fit the function f_m(x) by a weighted least squares regression of z_i to x_i using weights w_i.
(c) Update $F(x) = F(x) + 0.5 * f_m(x)$ and p(x)=$\frac{e^{F(x)}}{e^{F(x)} + e^{-F(x)}}$
4> Output the classifier sign[F(x)]= sign[$\sum_{m=1}^{M} f_m(x)$]

Stacking/Blending:

Stacking is one of the most popular ensemble machine learning techniques used to predict multiple nodes to build a new model and improve model performance. Stacking enables us to train multiple models to solve similar problems, and based on their combined output, it builds a new model with improved performance.
In stacking, an algorithm takes the outputs of sub-models as input and attempts to learn how to best combine the input predictions to make a better output prediction.

Architecture of Stacking: The architecture of the stacking model is designed in such as way that it consists of two or more base/learner's models and a meta-model that combines the predictions of the base models. These base models are called level 0 models, and the meta-model is known as the level 1 model. So, the Stacking ensemble method includes original (training) data, primary level models, primary level prediction, secondary level model, and final prediction. The basic architecture of stacking can be represented as shown below the image.



Algorithm for stacking:
1> Divide the training set into k folds.
2> Fit base model 1 on the first k-1 folds, and use it to predict the kth fold classification.
3> Store these predictions in x1_train array.
4> Repeat steps 2 and 3 for the remaining k-1folds.
5> Fit the model on the entirety of the training set and use it to classify the test set. Store the outcome in y1_test array.
6> Repeat the procedure for base model 2. Subsequently we get x2_train array and y2_test array.
7> Use the predictions of the base models to train the Level 1 model (also known as the Meta Model).

Data Description:

The data is extracted from StatsBomb open data (https://github.com/statsbomb/open-data (https://github.com/statsbomb/open-data)).

From the data, we are interested in working with only FA Women's Super League (which corresponds to competition id=37).

```
View(Comp)
```

A data.frame: 3 × 12

competition_id	season_id	country_name	competition_name	competition_gender	competition_youth	competition_international	season_name	match_updated	match_updated_360	match_ava
<int>	<int>	<chr>	<chr>	<chr>	<lgl>	<lgl>	<chr>	<chr>	<chr>	
37	90	England	FA Women's Super League	female	FALSE	FALSE	2020/2021	2021-07-01T18:14:40.756	2021-06-13T16:17:31.694	
37	42	England	FA Women's Super League	female	FALSE	FALSE	2019/2020	2021-06-01T13:01:18.188	2021-06-13T16:17:31.694	
37	4	England	FA Women's Super League	female	FALSE	FALSE	2018/2019	2022-05-27T12:02:24.272045	2021-06-13T16:17:31.694	

We extract the data on the shots for the individual matches.

match_id	match_date	kick_off	home_score	away_score	match_status	match_status_360	last_updated	last_updated_360	match_week	...	competition_stage.id	competition_stage.na
<int>	<chr>	<chr>	<int>	<int>	<chr>	<chr>	<chr>	<chr>	<int>	...	<int>	<ch
3775648	2021-02-28	15:00:00.000	0	4	available	scheduled	2021-03-01T07:42:29.838322	2021-06-13T16:17:31.694	11	...	1	Regular Seas
3775609	2021-04-28	20:30:00.000	2	0	available	scheduled	2021-05-28T15:32:57.267	2021-06-13T16:17:31.694	13	...	1	Regular Seas
3775633	2021-02-06	13:30:00.000	1	0	available	scheduled	2021-04-29T05:55:14.897296	2021-06-13T16:17:31.694	14	...	1	Regular Seas
3775570	2021-03-28	13:30:00.000	0	5	available	scheduled	2021-04-13T21:46:33.264800	2021-06-13T16:17:31.694	18	...	1	Regular Seas
3775581	2021-03-28	15:30:00.000	2	0	available	scheduled	2021-03-31T04:18:30.437117	2021-06-13T16:17:31.694	18	...	1	Regular Seas
3775579	2021-05-02	15:00:00.000	3	2	available	scheduled	2021-05-05T18:33:52.405043	2021-06-13T16:17:31.694	21	...	1	Regular Seas

The data on shots contains variables like 'distance of shot from goal', 'angle of shot wrt the goal', 'part of body with which shot was taken', 'the play pattern (corner, freekick, counter-attack etc.) that ended in taking the shot', 'type of pass' and so on that we will use to construct our xG model.

Variable Description

A brief description of the variables that we have used in our xG model prediction is provided:
1> **under_pressure:** binary variable, true if the action was performed while being pressured by an opponent
2> **play_pattern.id:** qualitative variable denoting the pattern of play which resulted in the shot coded as:
1- Regular Play; 2- From a corner; 3- From Free Kick; 4- From Throw In; 5- Other; 6- From Counter; 7- From Goal Kick; 8- From Keeper; 9- from Kick off

3> **position.id:** qualitative variable denoting the play position of the player who took the shot, coded as:

Position Number	Position Abbreviation	Position Name
1	GK	Goalkeeper
2	RB	Right Back
3	RCB	Right Center Back
4	CB	Center Back
5	LCB	Left Center Back
6	LB	Left Back
7	RWB	Right Wing Back
8	LWB	Left Wing Back
9	RDM	Right Defensive Midfield
10	CDM	Center Defensive Midfield
11	LDM	Left Defensive Midfield
12	RM	Right Midfield
13	RCM	Right Center Midfield
14	CM	Center Midfield
15	LCM	Left Center Midfield
16	LM	Left Midfield
17	RW	Right Wing
18	RAM	Right Attacking Midfield
19	CAM	Center Attacking Midfield
20	LAM	Left Attacking

4> **shot.one_on_one:** binary variable, true if the shot was taken when the attacker was one on one with the opponent keeper
5> **shot.open_goal:** binary variable, true if the shot was taken with an open goal
6> **shot.aerial_won:** binary variable, true if the shot was taken as a result of winning an aerial duel
7> **shot.deflected:** binary variable, true if the shot was redirected by another player's touch
8> **shot.type.id:** qualitative variable, which denotes the type of the shot taken coded as:
61- corner; 62- free kick; 87- open play; 88- penalty; 65- kickoff
9> **shot.body_part.id:** qualitative variable, which denotes the body part from which the shot was taken coded as:
37- head, 38- left foot; 40- right foot, 70- other
10> **DistToGoal:** continuous variable; distance of the point of origin of the shot from goal
11> **DistToKeeper:** continuous variable; distance of the point of origin of the shot from keeper
12> **AngleDeviation:** continuous variable; denotes the absolute angle between goalkeeper and shot location
13> **avevelocity:**: continuous variable; denotes the average velocity with which the ball travelled in the shot
14> **shot.technique.id:** qualitative variable, which denotes the technique of the shot, coded as:
89- Backheel; 90- Diving Header; 91- Half Volley; 92- Lob; 93- Normal; 94- Overhead Kick; 95- Volley
15> **DefendersInCone:** integer type variable; denotes the number of defenders present in the incone
16> **MaxAcuteAngle:** (defined later)
17> **AttackersBehindBall:** integer type variable; denotes the number of attackers behind the line of the ball
18> **density.incone:** (defined later)
19> **pass_throughball:** binary variable; true if last pass cuts the last line of defence
20> **shot.first_time:** binary variable; true if the shot was a first touch
21> **shot.redirect:** binary variable: true if the shot is redirected towards goal

Feature Engineering:

We need to perform the following steps for fitting a model.
* Cleaning NA from Shots data. In binary variables, NA is replaced by FALSE.
* Treating the categorical explanatory variables as factors.
* Divide the data into training data and test data, with 80% of the data being used as training data and 20% of the data being used as test data.
Data is divided in a way to maintain the same proportion of shot outcome in train and test data.

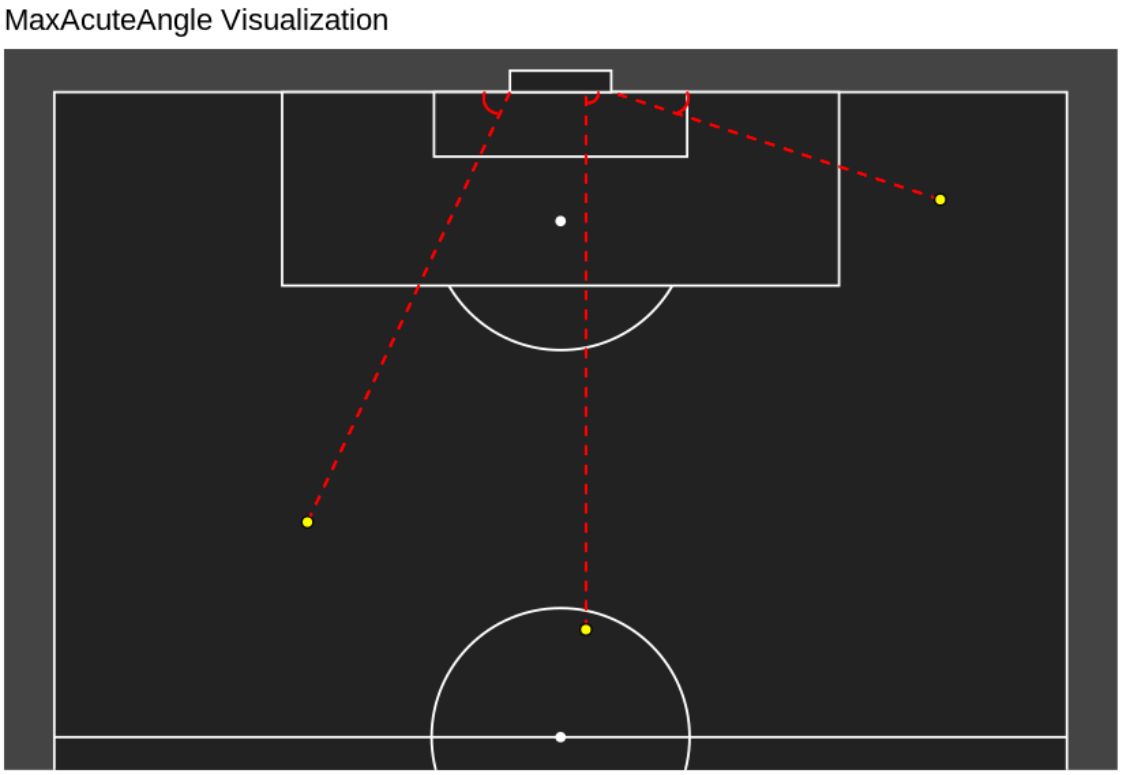
Custom variables

We introduce two custom made variables in our model. The description of these variables and the motivation of tailoring them are mentioned below.

1> Maximum Acute Angle

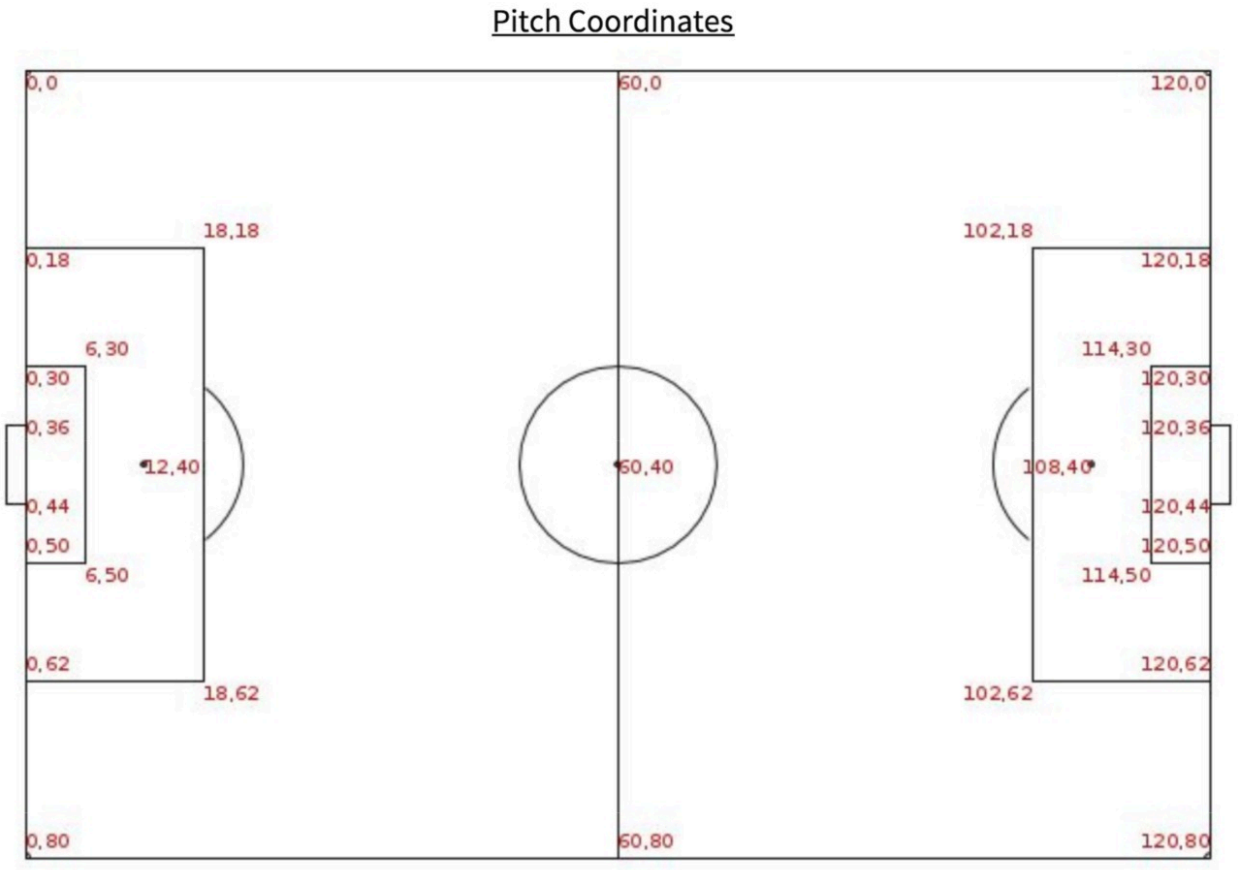
This variable is defines as the maximum acute angle that can be formed by the line made by the direction of the shot and the goalline. This variable is tailored from the variable AngletoGoal, which is the angle between the line of the shot and the goalline (can be an obtuse angle). But suppose a shot which makes an angle of 45 degrees coming from the right side of the goal should have equal probability of a similar shot which comes from the left side of the goal, having AngletoGoal as 135 degrees, when other factors are same. Hence it makes sense that the angle to goal is converted to acute angles only before proceding.

This variable is well represented by the following diagram



2> Incone Density

To understand this variable, we first look at the football field with the coordinates of the various points on it.



We define two cones-

* a cone formed by the point of origin of the shot to the ends of the goal post [with coordinates (120,36) and (120,44) when we consider the right half of the pitch].

* a wider cone formed by the point of origin of the shot to the ends of the goal post [with coordinates (120,33) and (120,47) when we consider the right half of the pitch]. The motivation behind forming a wider cone is that we want to include the effect of the defenders who might be out of the line of a shot on target initially, but may come running in to block a shot.

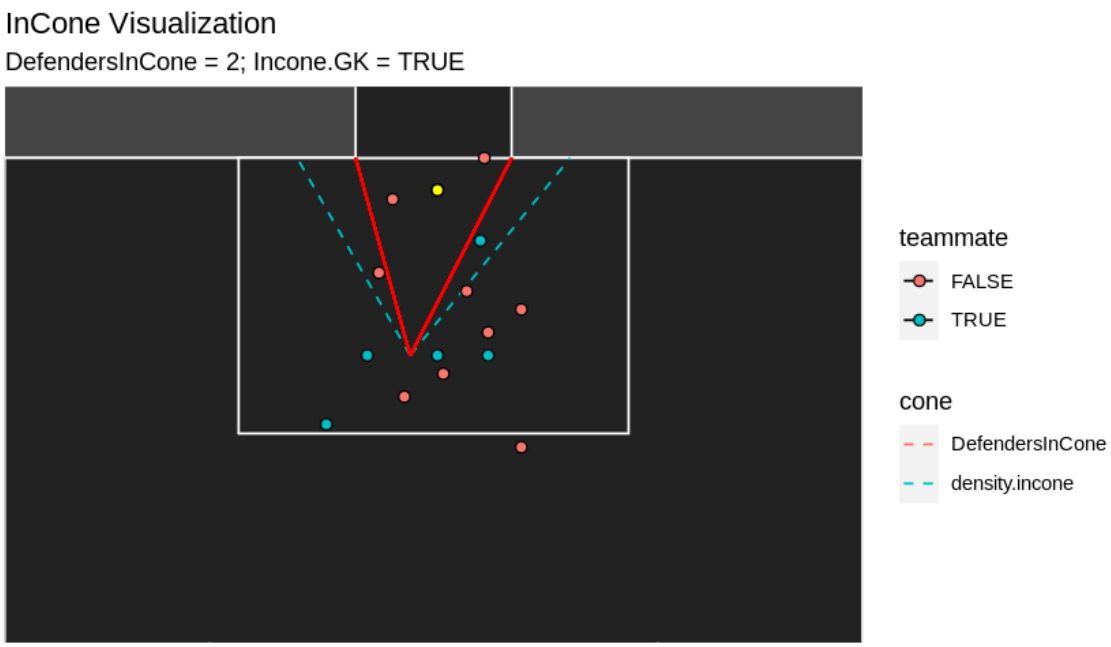
$$Incone\ density = \sum_{i \in A} 1/d_i$$

where A denotes the set of all defenders inside the wider cone and d_i denotes the distance of the defender from the point of the shot.

If there are more defenders in the cone, the reciprocal of their distances get added and incone density gets large. Also if the defenders are closer to the point of origin of the shot, the incone density gets larger.

The motivation behind including this variable is that we may expect that if incone density is large, there will be less chances of the shot ending in

a goal.
The incone density can be explained better with the following diagram:



Training the models:

To train our model, we will be using Resampling Techniques. These involve repeatedly drawing samples from a training set and refitting a model of interest on each sample in order to obtain additional information about the fitted model.
The Resampling procedure we will use is k-fold repeated cross validation. We will perform Resampling to tune the models i.e. find the best possible hyper parameters to be considered in the model. ROC AUC is used as the performance metric, the hyper parameters that maximize this metric are selected as the final hyper parameters.
Other schemes for selecting model can be used. Breiman et al (1984) suggested the “one standard error rule” for simple tree-based models. In this case, the model with the best performance value is identified and, using resampling, we can estimate the standard error of performance. The final model used was the simplest model within one standard error of the (empirically) best model. With simple trees this makes sense, since these models will start to over-fit as they become more and more specific to the training data. But we will stick to simply choosing the “best” model (hyperparameters giving maximumm ROC AUC).

```
Define sets of model parameter values to evaluate
for each parameter set do
    for each resampling iteration do
        Hold–out specific samples
        [Optional] Pre–process the data
        Fit the model on the remainder
        Predict the hold–out samples
    end
    Calculate the average performance across hold–out predictions
end
Determine the optimal parameter set
Fit the final model to all the training data using the optimal parameter set
```

The descriptions of these are mentioned below:

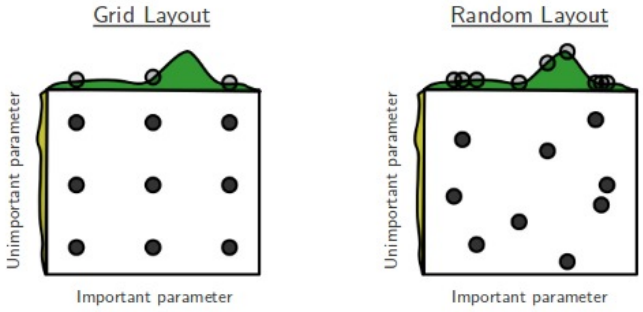
k-fold Cross Validation:

This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. One of the groups is treated as a validation set, and the model is fit on the remaining k-1 groups. The measure of error is calculated once for every group taken as a validation set. The final estimate of error is calculated as the average of all the estimates.

Repeated k-fold Cross Validation:

To further reduce the variance of the estimate of the error, we use k-fold Cross Validation with repetitions. Suppose we have r repetitions, we compute the estimate of the measure of error as mentioned previously for each of the repetitions. The final estimate is the mean over all the repetitions.

We use package 'caret' to tune the hyper parameters. There are two major approaches to tuning :
1. Grid Search :
We provide the model with the relevant grid of all possible values of hyper parameters. The training algorithm only considers these set of values.
2. Random Search :
Independent draws from a uniform density from the same configuration space as would be spanned by a regular grid, as an alternative strategy for producing a trial set {λ^(1), …, λ^(S)}. Random search has all the practical advantages of grid search (conceptual simplicity, ease of implementation, trivial parallelism) and trades a small reduction in efficiency in low-dimensional spaces for a large improvement in efficiency in high-dimensional search spaces.



We will be using the default grid search from caret package as it's more computationally practical for the models we will train. By default, if p is the number of tuning parameters, the grid size is 3^p.

Performance of classifiers:

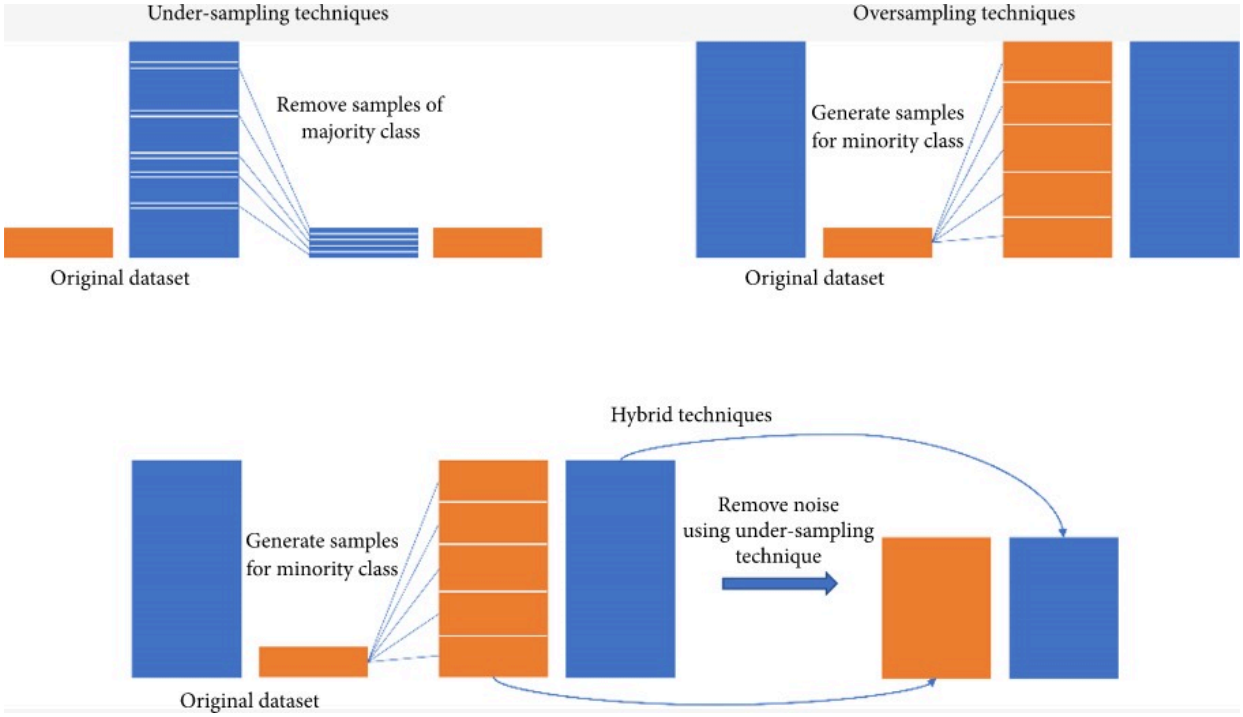
Two of the popular ways to diagnose the performance of a classifier are PR curves and ROC curves.
1> PR curve:
PR curve stands for Precision-Recall curve. The PR curve has Precision values on the Y-axis and Recall values on the X-axis.
Precision=TP/(TP+FN)
Recall=TP/(TP+FP)
TP=True Positive; FP=False Positive; TN= True Negative; FN= False Negative
2> ROC curve
ROC curve stands for Receiver Operating Characteristic Curve). It is a graph showing the performance of a classification model at all classification thresholds. The ROC curve has True Positive Rate (or Recall) on the X-axis and False Positive Rate on the Y-axis.
True Positive Rate (TPR)= TP/(TP+FN)
False Positive Rate (FPR)= FP/(FP+TN)
For both the curves, a good classifier should have high Area Under Curve (AUC) value. These metrics are essential in presence of class imbalance, which is true in our case.

We observe that only about 12% of our data contains shots that resulted in a Goal. Hence we will also need to deal with class imbalance in the data set while performing classification.

Subsampling For Class Imbalances

In classification problems, a disparity in the frequencies of the observed classes can have a significant negative impact on model fitting. In our data, we have a majority of shots that don't end up in a goal, and a minority of shots that end up in a goal, hence leading to a class imbalance. One technique for resolving such a class imbalance is to subsample the training data in a manner that mitigates the issues. Examples of sampling methods for this purpose are:

```
- down-sampling: randomly subset all the classes in the training set so that their class frequencies match the le
ast prevalent class. For example, suppose that 80% of the training set samples are the first class and the remain
ing 20% are in the second class. Down-sampling would randomly sample the first class to be the same size as the s
econd class (so that only 40% of the total training set is used to fit the model). caret contains a function (dow
nSample) to do this. \
- up-sampling: randomly sample (with replacement) the minority class to be the same size as the majority class. c
aret contains a function (upSample) to do this. \
- hybrid methods: techniques such as SMOTE and ROSE down-sample the majority class and synthesize new data points
in the minority class. There are two packages (DMwR and ROSE) that implement these procedures.\
```



The above procedures are independent of resampling methods such as cross-validation and the bootstrap. In practice, one could take the training set and, before model fitting, sample the data. There are two issues with this approach :

```
- during model tuning the holdout samples generated during resampling are also glanced and may not reflect the cl
ass imbalance that future predictions would encounter. This is likely to lead to overly optimistic estimates of p
erformance. \
- the subsampling process will probably induce more model uncertainty. The model results can differ under a diffe
rent subsample. As above, the resampling statistics are more likely to make the model appear more effective than
it actually is. \
```

The alternative is to include the subsampling inside of the usual resampling procedure. The two disadvantages are that it might increase computational times and that it might also complicate the analysis in other ways (Sparsely represented categories in factor variables may turn into zero-variance predictors or may be completely sampled out of the model, For some models that require more samples than parameters, a reduction in the sample size may prevent us from being able to fit the model).

We will use ROSE subsampling procedure inside of the k-fold repeated cross validation resampling for our training/tuning algorithm.

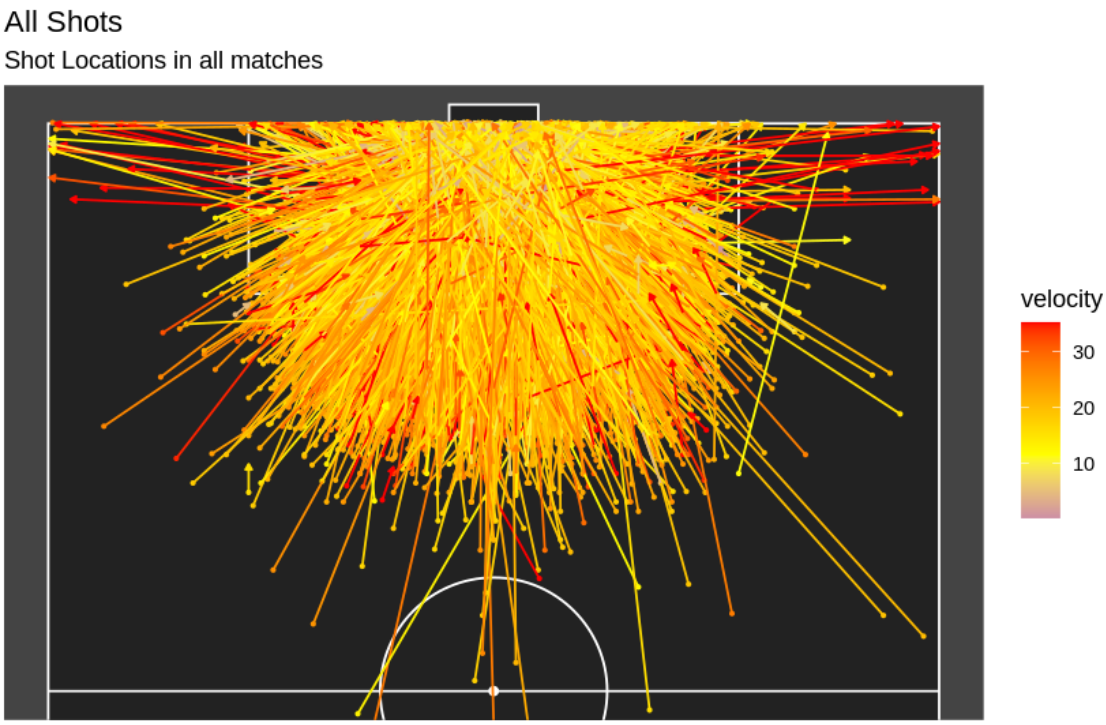
ROSE (Random Over-Sampling Examples):

It is a bootstrap-based technique which aids the task of binary classification in presence of rare classes. It creates a sample of synthetic data by enlarging the features space of minority and majority class examples. Operationally, the new examples are drawn from a conditional kernel density estimate of the two classes.
Let $'y'$ be the binary response and $'x'$ be vector of numeric predictors observed on n subjects i, (i = 1, . . . , n). Synthetic examples with class label k,(k = 0, 1) are generated from a kernel estimate of the conditional density $f(x|y = k)$. Essentially, ROSE selects an observation belonging to the class k and generates new examples in its neighborhood, where the width of the neighborhood is determined by an asymptotically optimal smoothing matrix.

Implementation and Inference

Visualisation

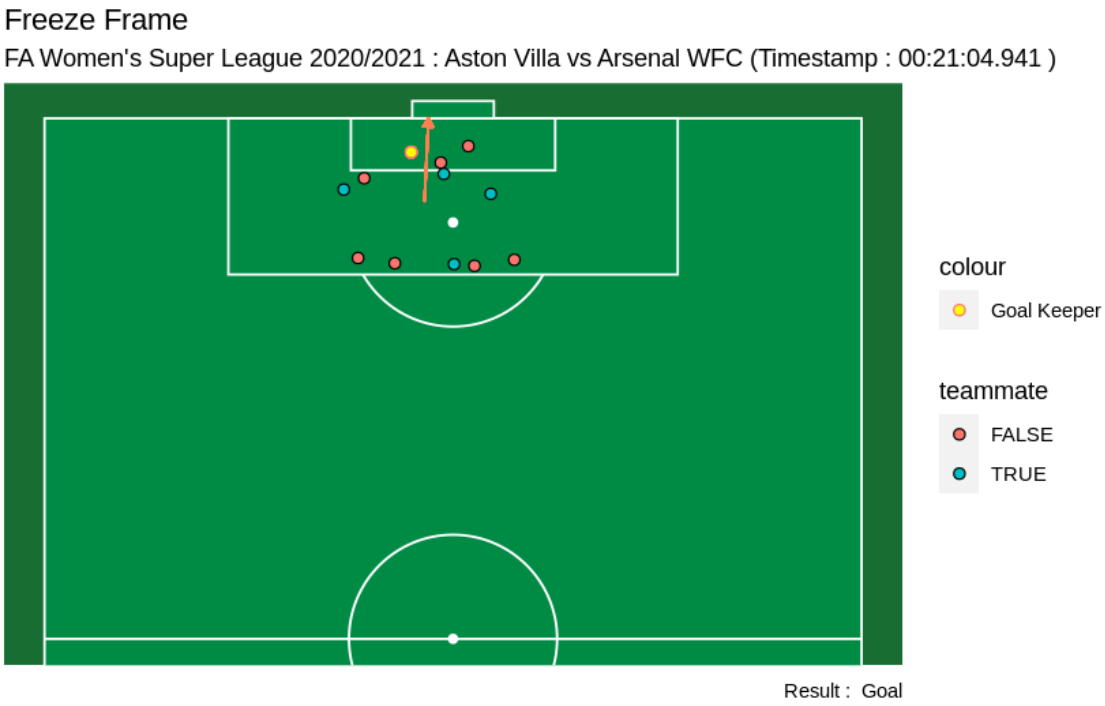
Once we have the data in hand, we are interested to visualise the shot positions for all the matches. To do the same, we have made use of three kinds of plots.
Plot depicting shot positions
In this plot, we have made use of lines connecting the position from where the shot originated and the position where it terminated. The different coloured lines depict the different velocities of the shots. This shows majority data is coming from shot locations close to the goal.



Freeze Frame

A freeze frame provides precise goalkeeper and defender positionings for the shot. It is a snapshot taken at the moment of the shot that captures

the location of all players involved in the event. It enables us to more accurately evaluate the context around each shot and measure the impacts of things we expect to have an impact on the outcome of those chances, such as defensive pressure.



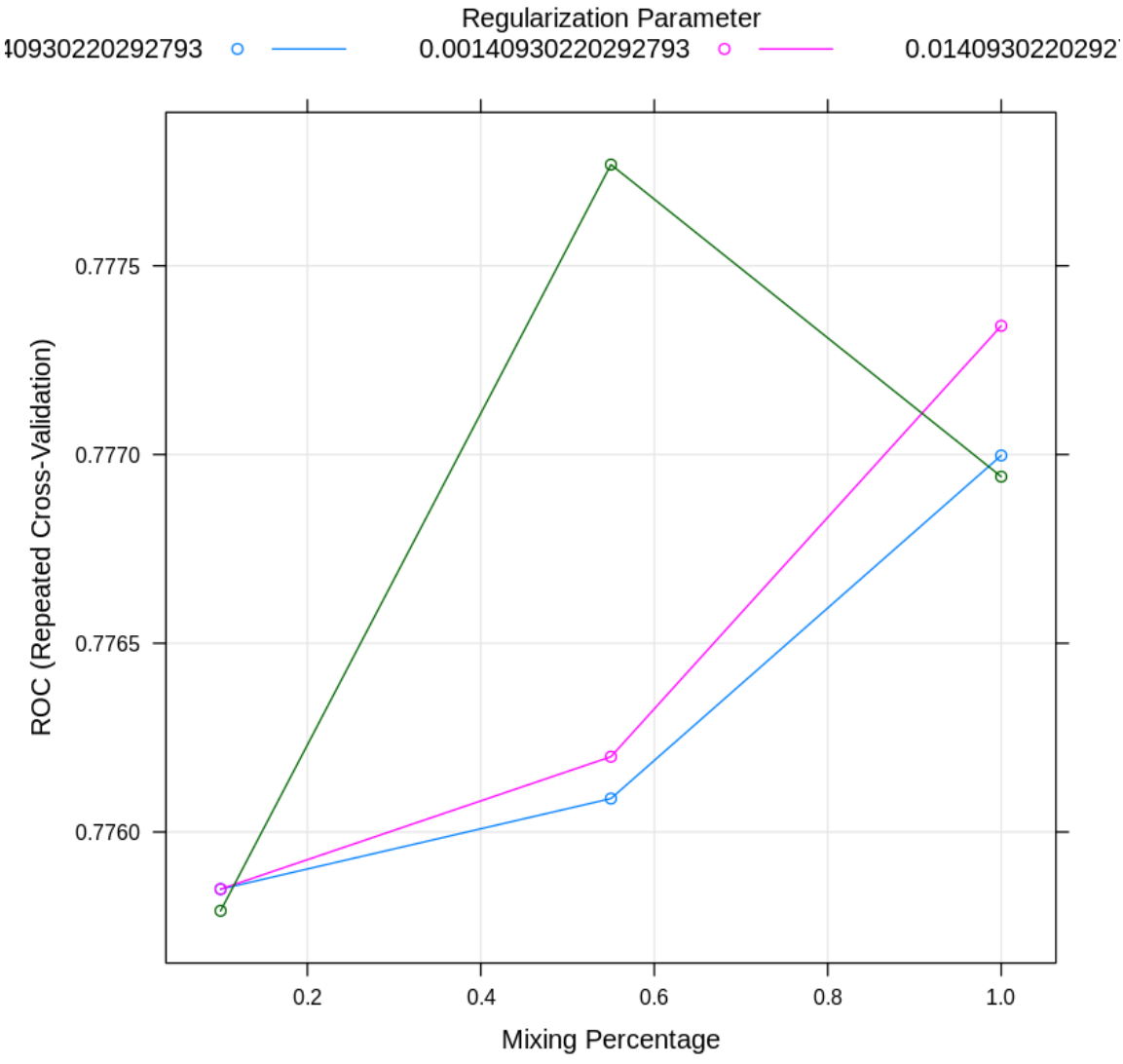
Animation The last plotting method that we will use to visualise the shots is to use an animation. The dots on the animation depicts the football as it travels in each shot, colour coded according to velocity. This indicates how our training data is distributed in terms of shot location and shot velocity.

Penalized Logistic Regression:

In R:
We can fit a Penalized Logistic Regression using the glmnet package in R. We use ROSE method for resampling, and k-fold repeated CV for training the model. The number of folds is chosen as 20 and the number of repetitions is chosen as 10. ROC is used as a measure to determine our tuning parameters α and λ.

R output:

```
glmnet

6561 samples
  20 predictor
   2 classes: 'NG', 'G'

No pre-processing
Resampling: Cross-Validated (20 fold, repeated 10 times)
Summary of sample sizes: 6233, 6233, 6233, 6233, 6233, 6232, ...
Addtional sampling using ROSE

Resampling results across tuning parameters:

  alpha  lambda        ROC        Sens       Spec
  0.10   0.0001409302  0.7758487  0.7126382  0.7033333
  0.10   0.0014093022  0.7758487  0.7126382  0.7033333
  0.10   0.0140930220  0.7757911  0.7108062  0.7059722
  0.55   0.0001409302  0.7760885  0.7133570  0.7030556
  0.55   0.0014093022  0.7761994  0.7130491  0.7034722
  0.55   0.0140930220  0.7777682  0.7012354  0.7150000
  1.00   0.0001409302  0.7769975  0.7163515  0.7043056
  1.00   0.0014093022  0.7773409  0.7151872  0.7069444
  1.00   0.0140930220  0.7769414  0.6908766  0.7222222

ROC was used to select the optimal model using the largest value.
The final values used for the model were alpha = 0.55 and lambda = 0.01409302.
```

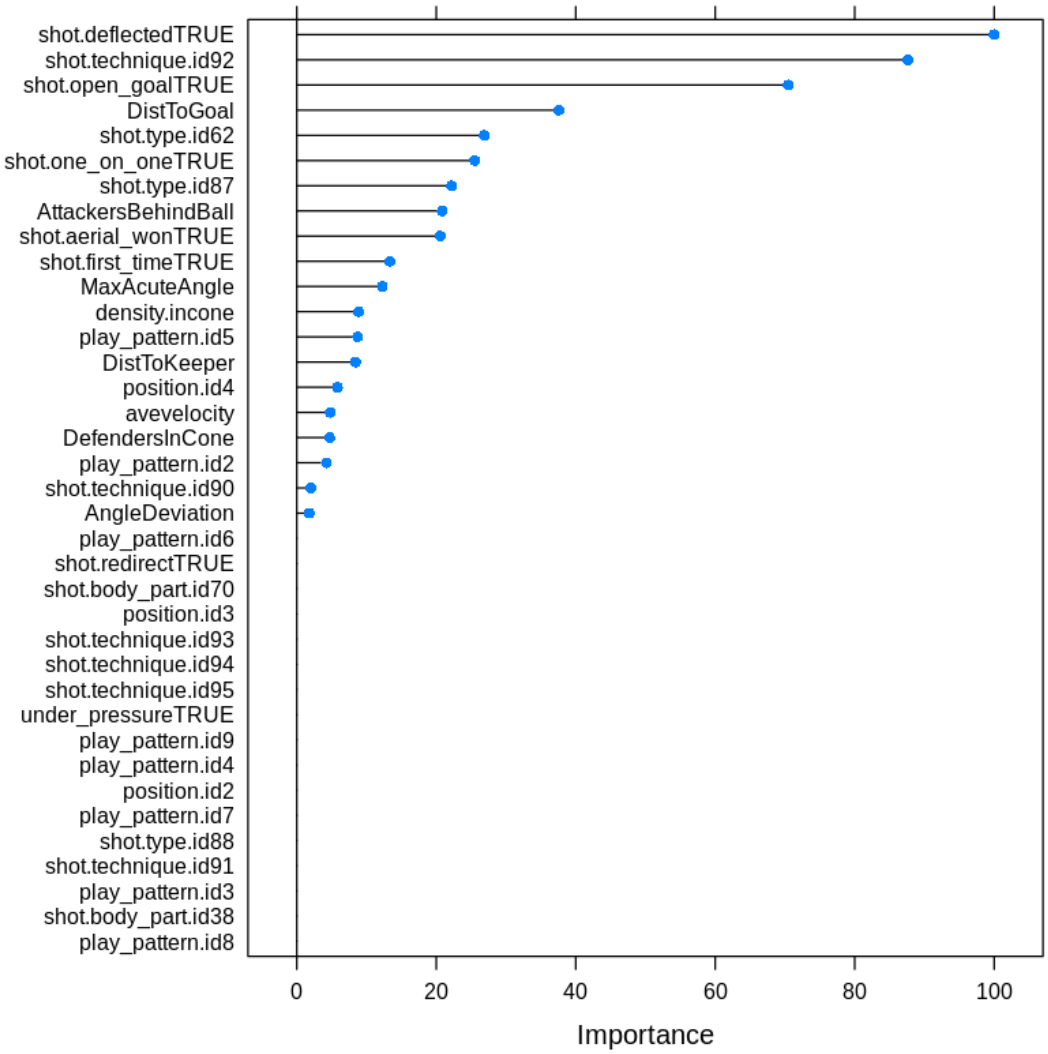


We see that the maximum ROC was attained for $\alpha = 0.55$ and $\lambda = 0.01409302$, i.e. we go for elastic regression with shrinkage parameter = 0.01409302. Even though there is a penalty α of 0.55, the effect of that penalty is very low as $\lambda = 0.014$.

Variance Importance Plot:

Variable importance plot provides a list of the most significant variables in descending order by a mean decrease in Gini. The top variables contribute more to the model than the bottom ones and also have high predictive power in the classification. The mean decrease in Gini index is expressed relative to the maximum.

We also do a variance importance plot, to check for the predictors present in our model which influence the classification most.



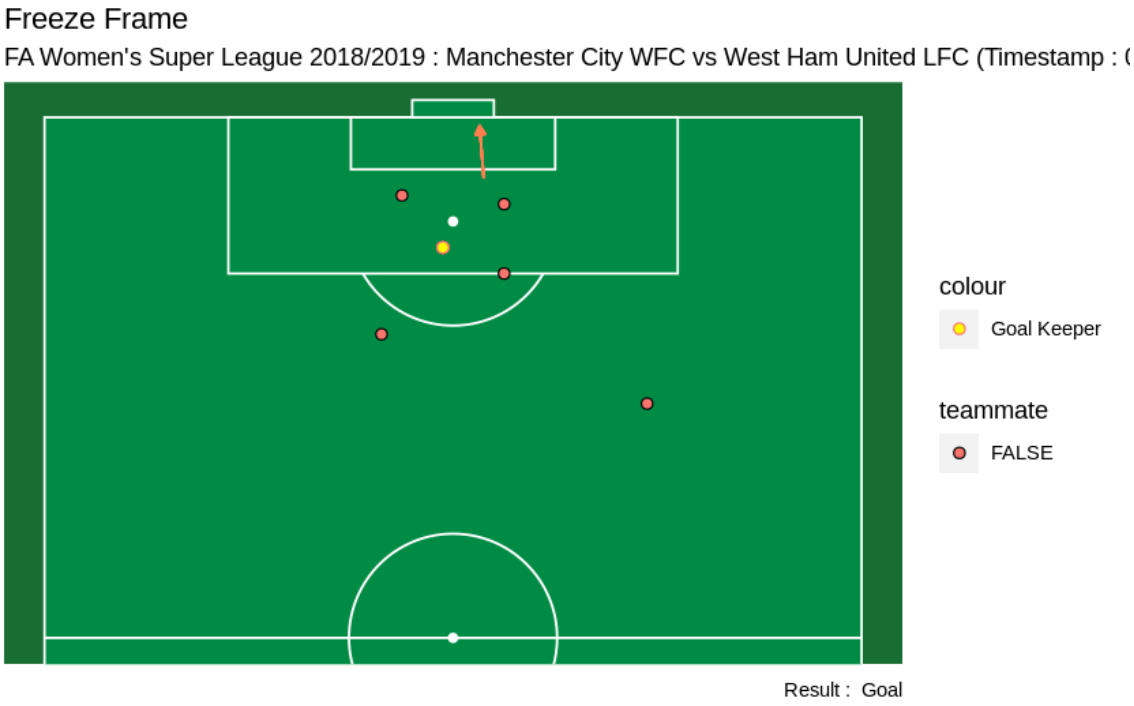
Prediction:

Now we will use this model to predict the outcome of a shot in the test set.

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
0.08865 0.33290 0.43555 0.44332 0.54472 0.88632 
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
0.00000 0.02402 0.05221 0.10672 0.13263 0.90459 
0.88776946
```

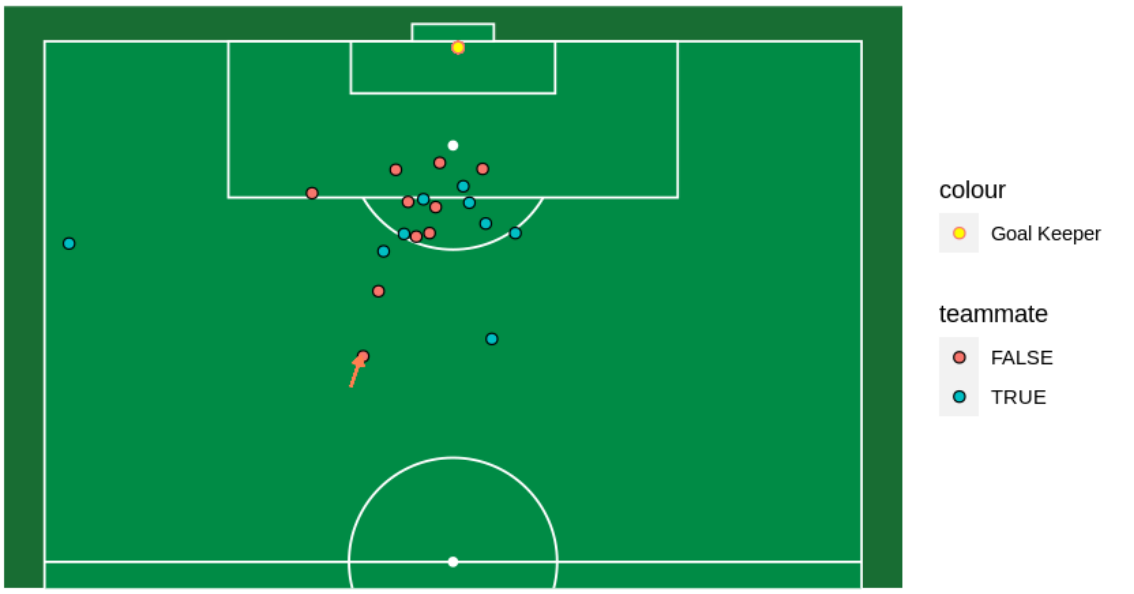
We will plot the freeze frame of the shot that had the highest predicted XG and the shot that had the lowest predicted XG as a means for verification. We expect that the shot with the highest predicted XG should end up in a goal, while the shot with the lowest predicted XG should

not end up in a goal.



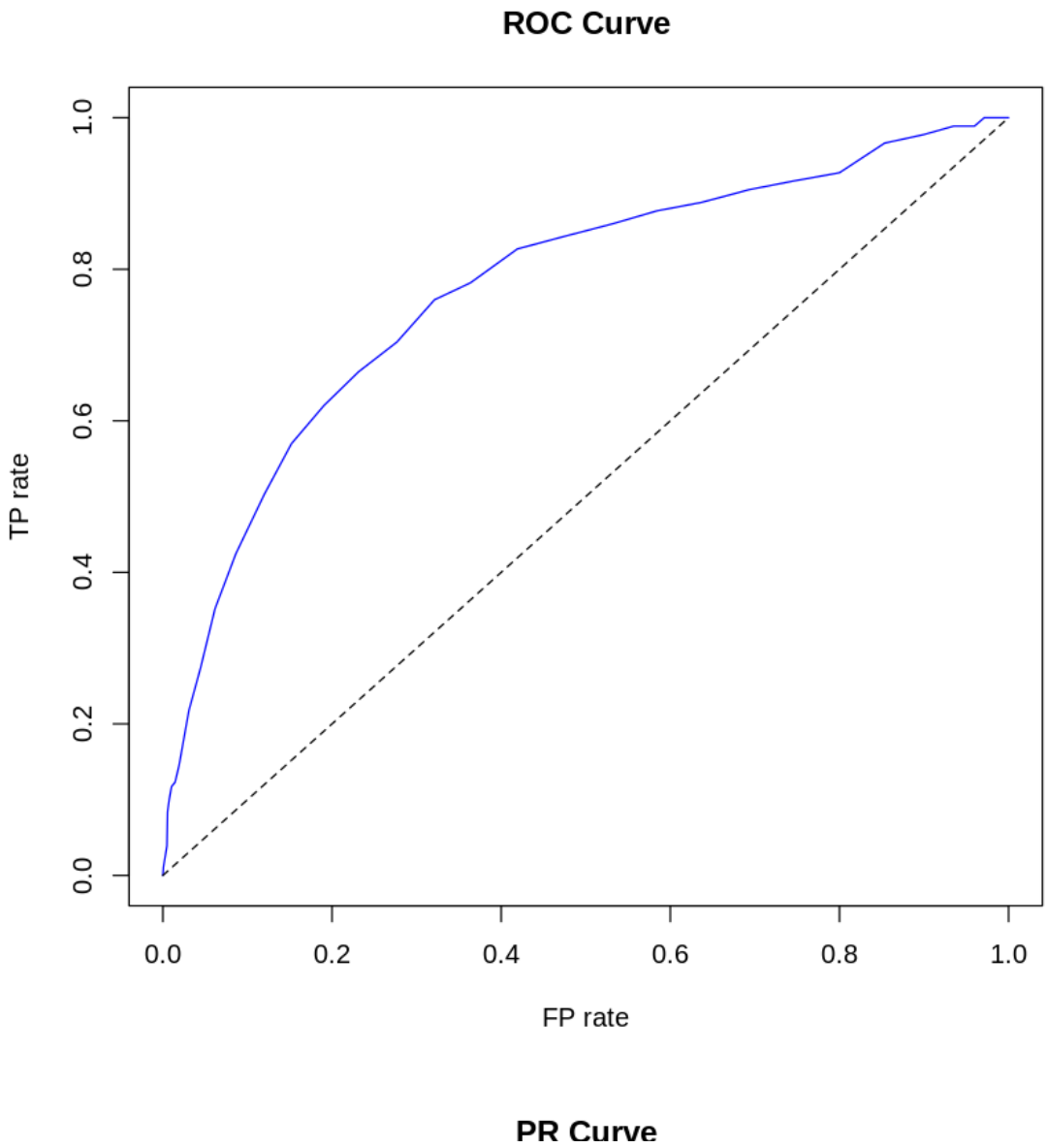
Freeze Frame

FA Women's Super League 2020/2021 : Manchester City WFC vs Manchester United (Timestamp : 00:2

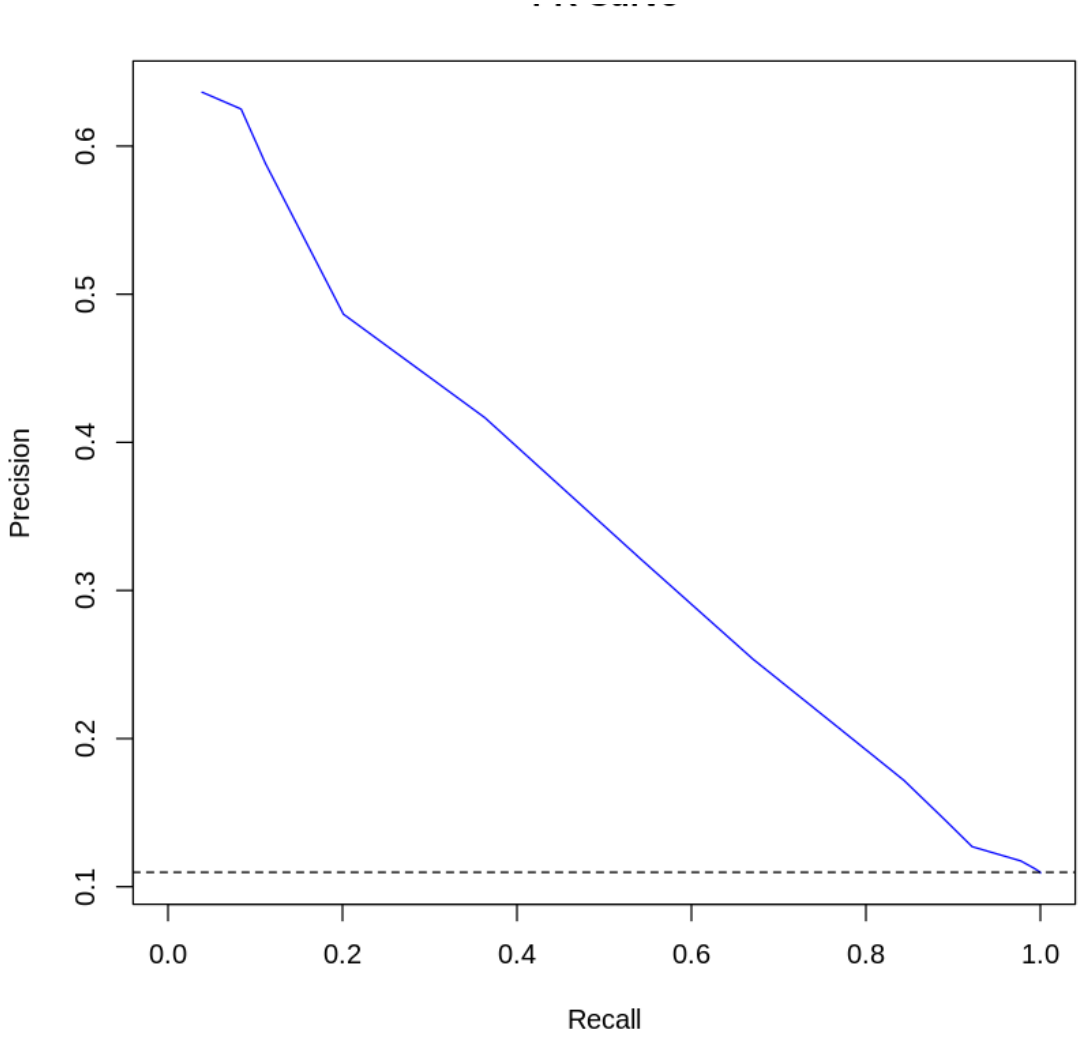


Result : Blocked

Further we will use ROC curve and PR curve to diagnose the performance of this classifier.

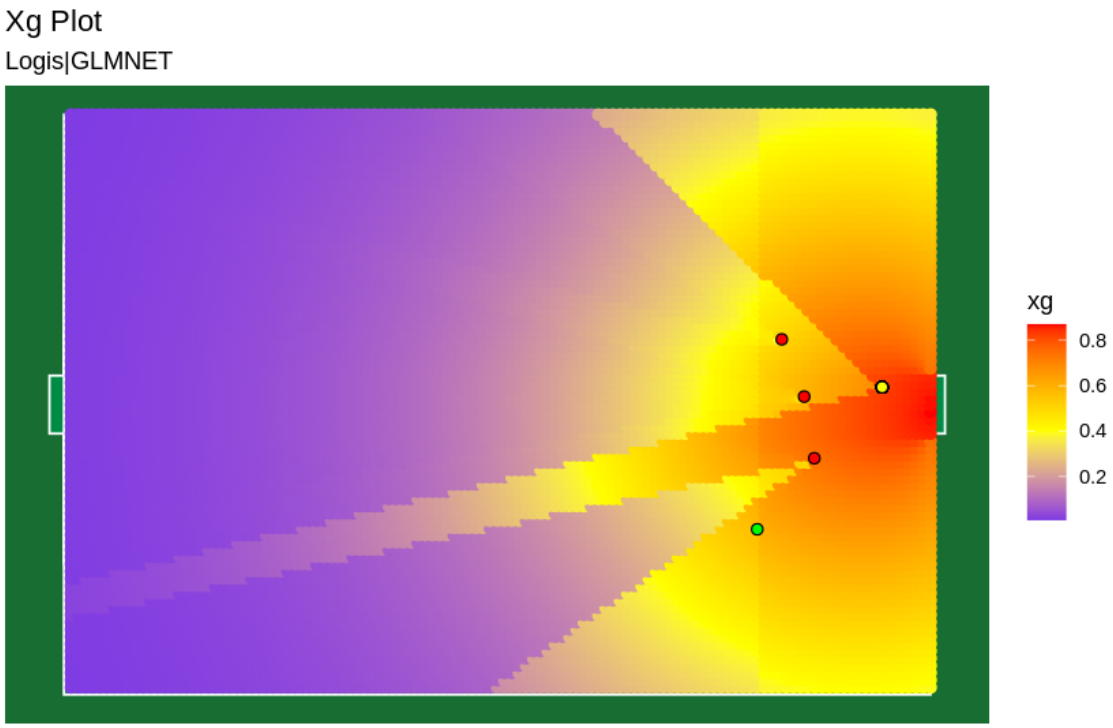


PR Curve



In the ROC curve, the dotted line (having AUC=0.5), depicts a classifier that works on pure chance. Our penalized logistic regression model has significantly higher AUC than the chance based classifier. Hence we can say that this classifier works well. This is further supported by high diagonal entries in the confusion matrix and high AUC in the PR curve.

Heatmap:

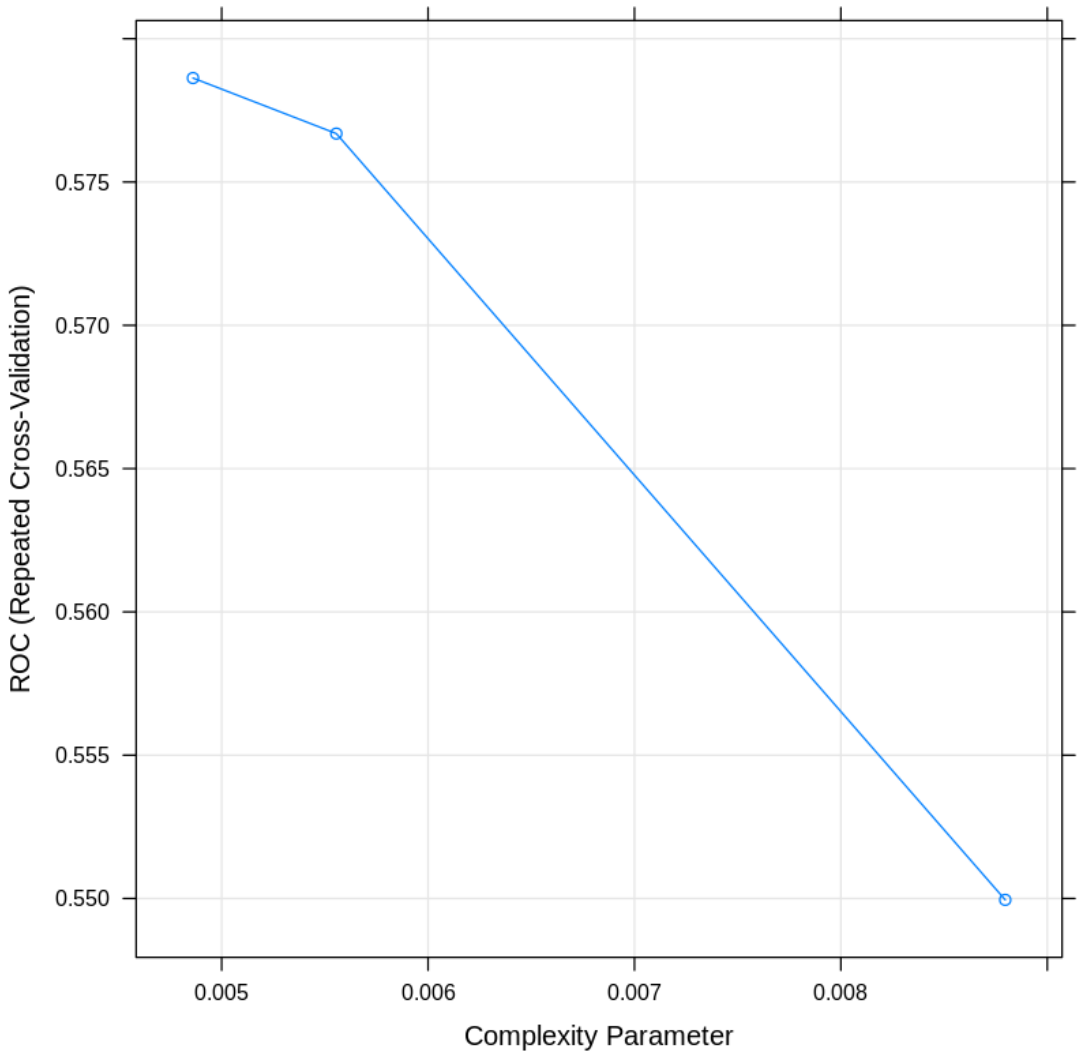


Decision Tree:

In R:

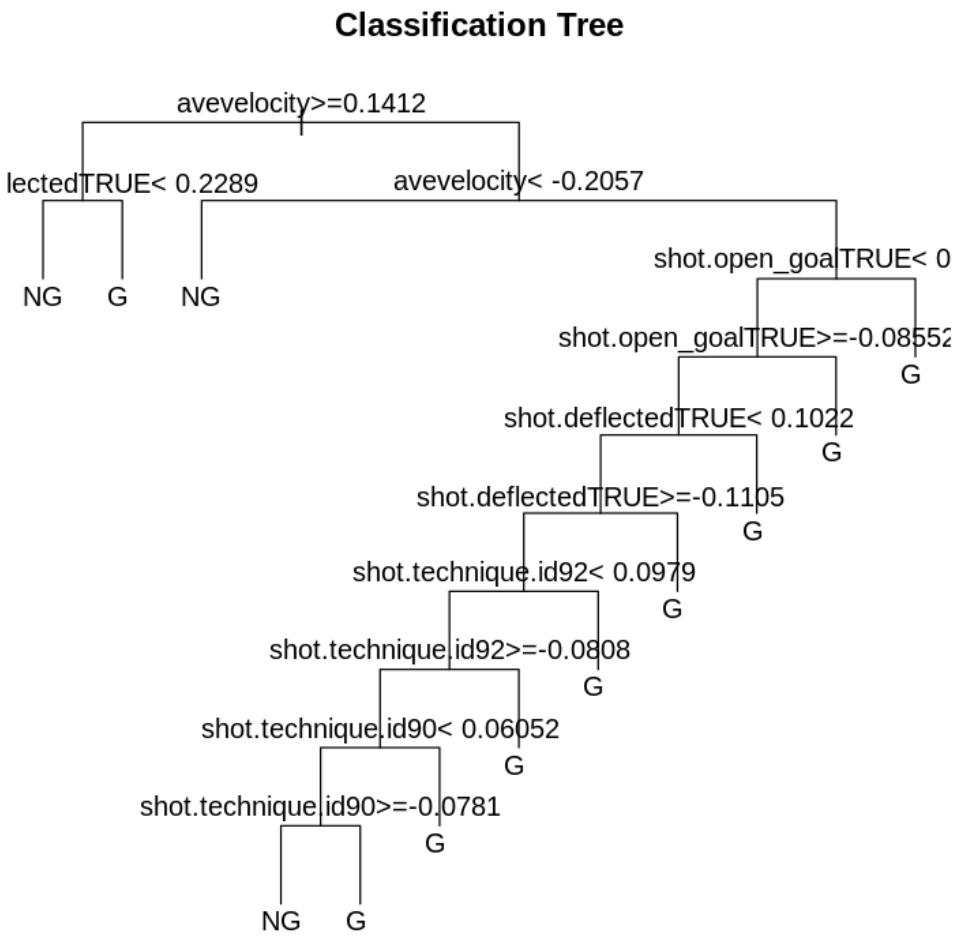
We use rpart library in R to make a decision tree. We use ROSE method of resampling and k-fold repeated CV method to train our model. The number of folds is chosen to be 10 and repeated 3 times. We use ROC as a measure to fix our tuning parameters.

```
CART

6561 samples
  20 predictor
   2 classes: 'NG', 'G'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 5905, 5904, 5905, 5905, 5905, 5905, ...
Addtional sampling using ROSE

Resampling results across tuning parameters:

  cp           ROC        Sens       Spec
  0.004861111  0.5786282  0.8668962  0.2694444
  0.005555556  0.5766899  0.8756862  0.2569444
  0.008796296  0.5499515  0.6952638  0.3990741

ROC was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.004861111.
```



Here we tune the complexity parameter (cp) which is the minimum improvement in the model needed at each node.

Plotting our decision tree:



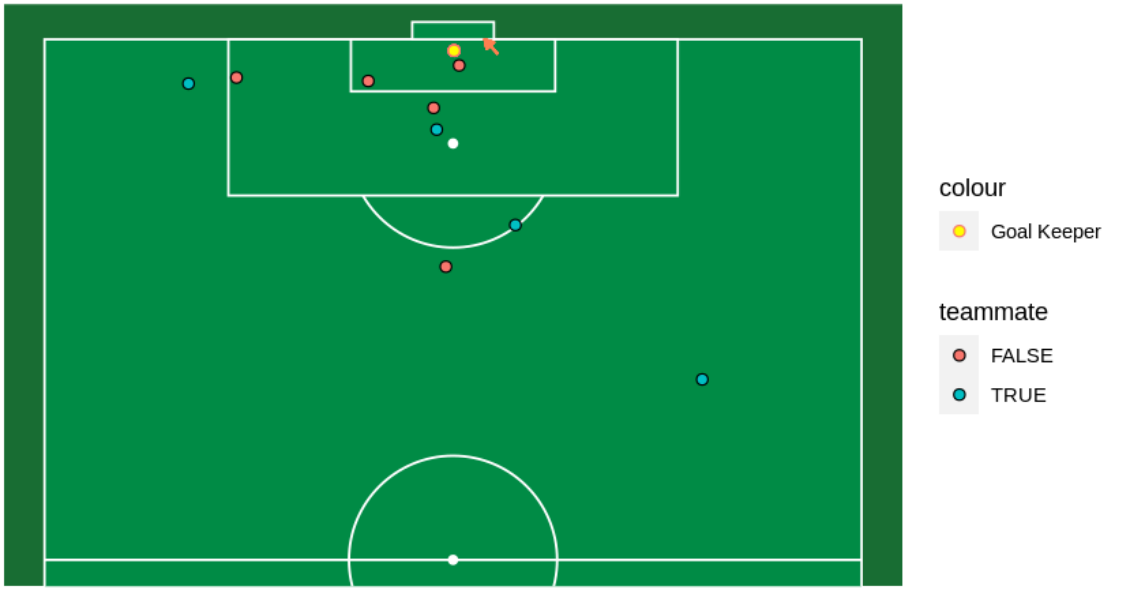
We use the fitted decision tree to make predictions on the training set.

```
     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.009194 0.193431 0.193431 0.212521 0.193431 0.985140
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.02402 0.05221 0.10672 0.13263 0.90459
0.5296389
```

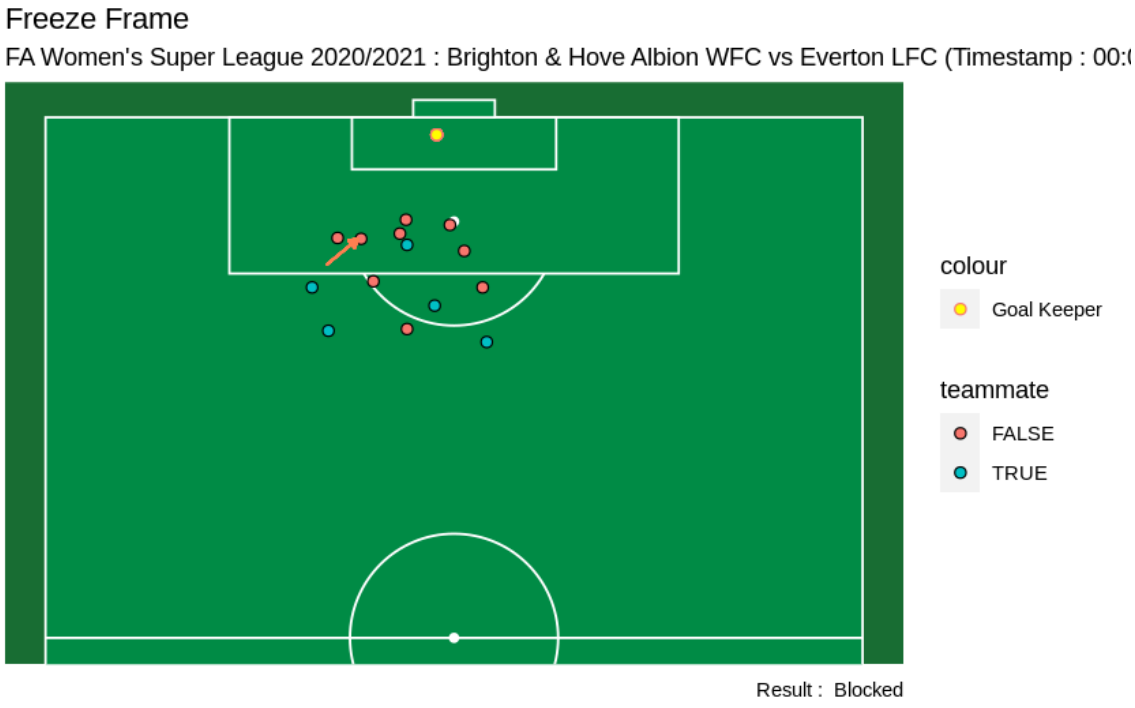
We again use the freeze frame of the shot with the highest and lowest predicted XG to check for the validity of our model.

Freeze Frame

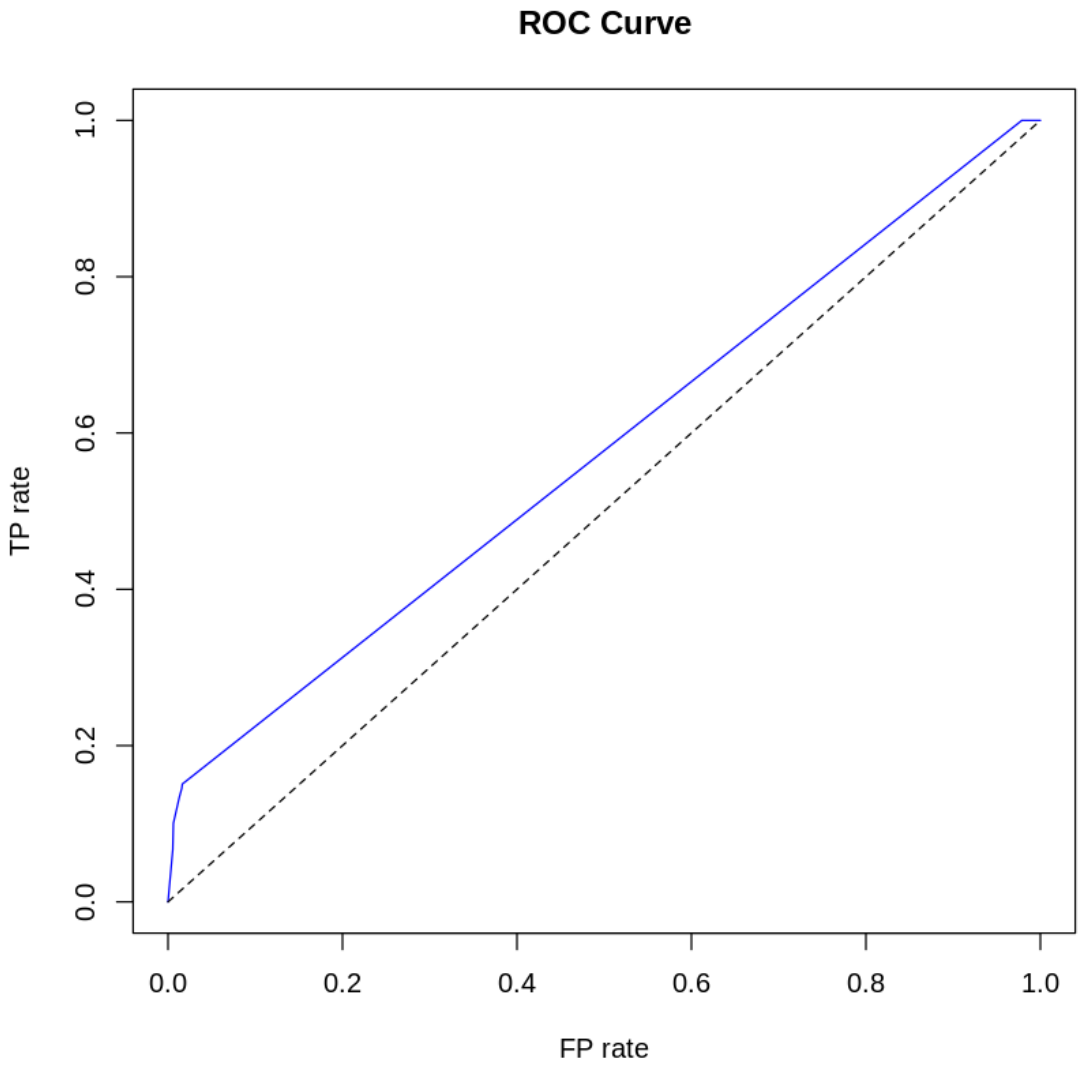
FA Women's Super League 2020/2021 : Brighton & Hove Albion WFC vs Arsenal WFC (Timestamp : 00:

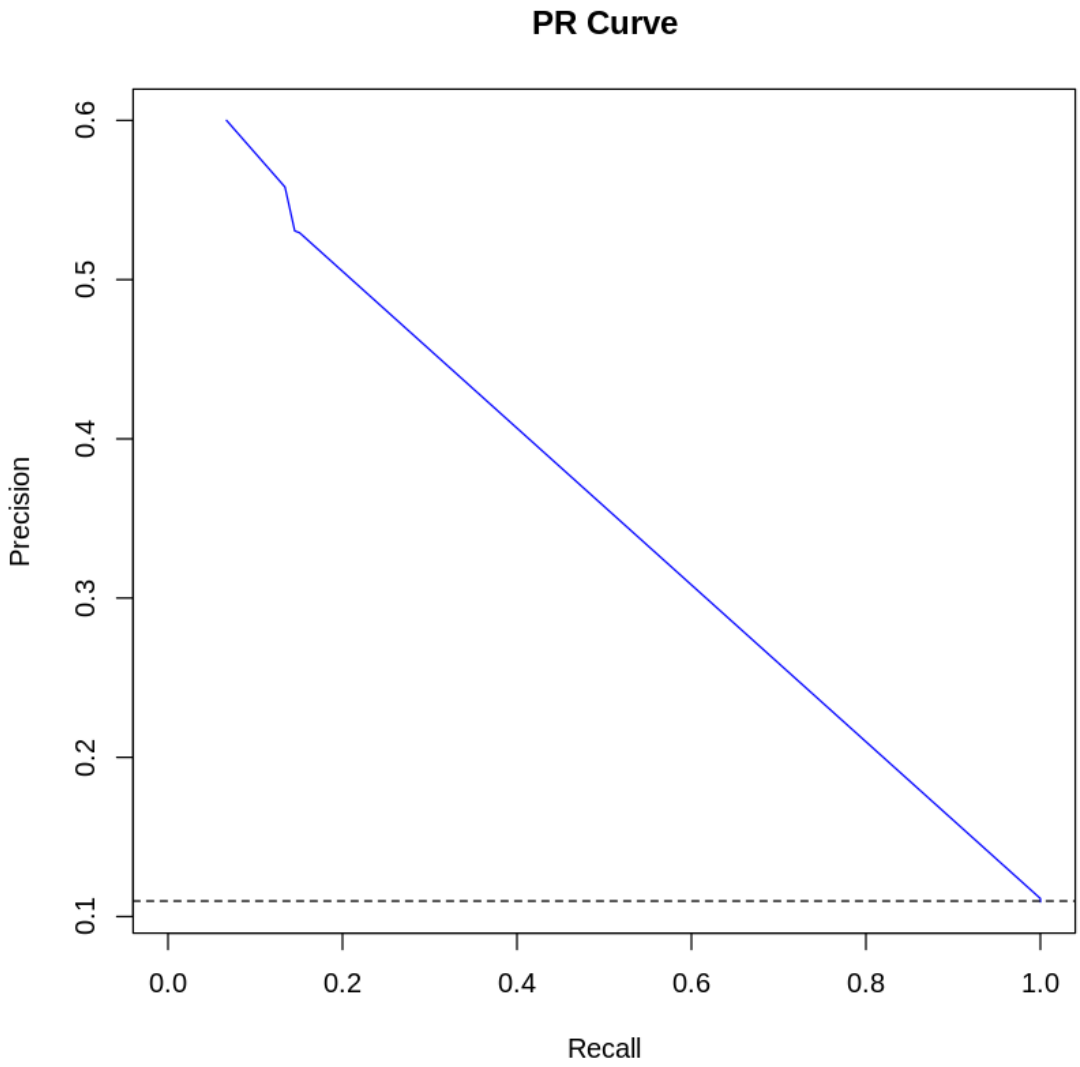


Result : Goal

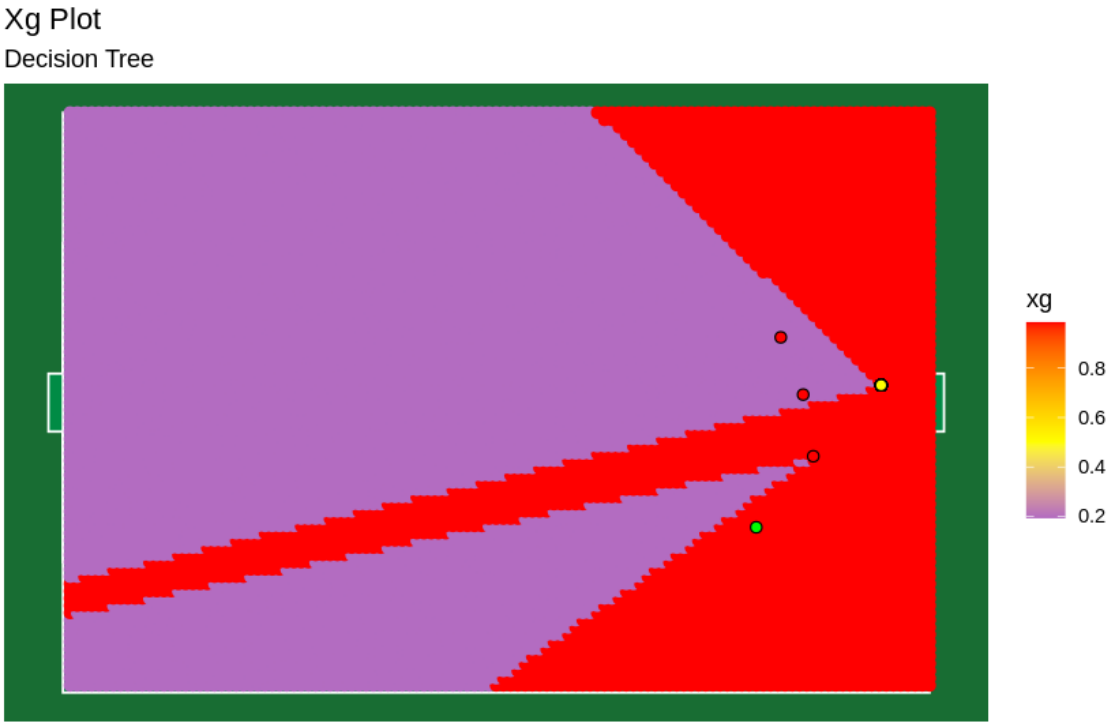


Again we use PR curve and ROC curve to diagnose the efficiency of the classifier.


ROC Curve
TP rate
FP rate
0.0
0.2
0.4
0.6
0.8
1.0



Heatmap:

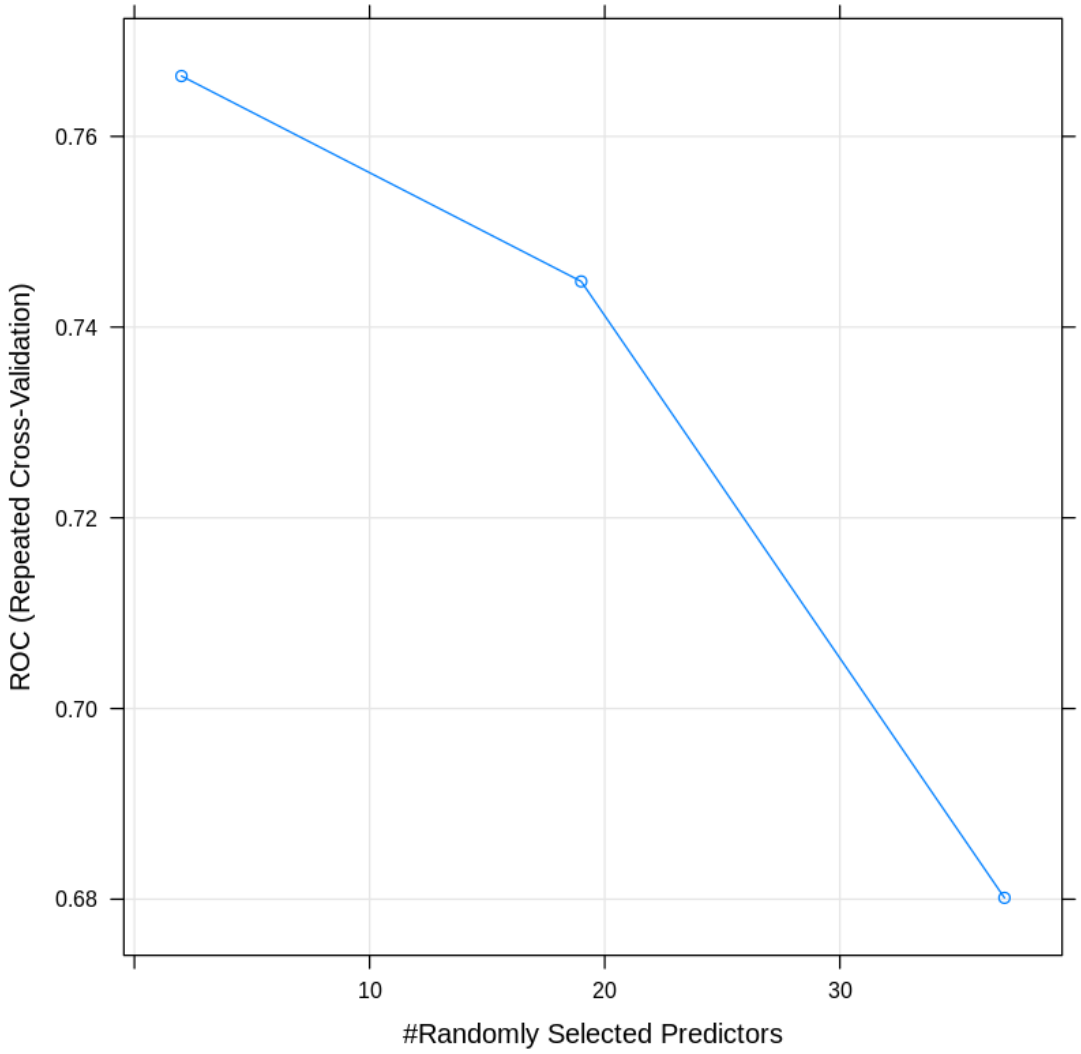


Random Forests:

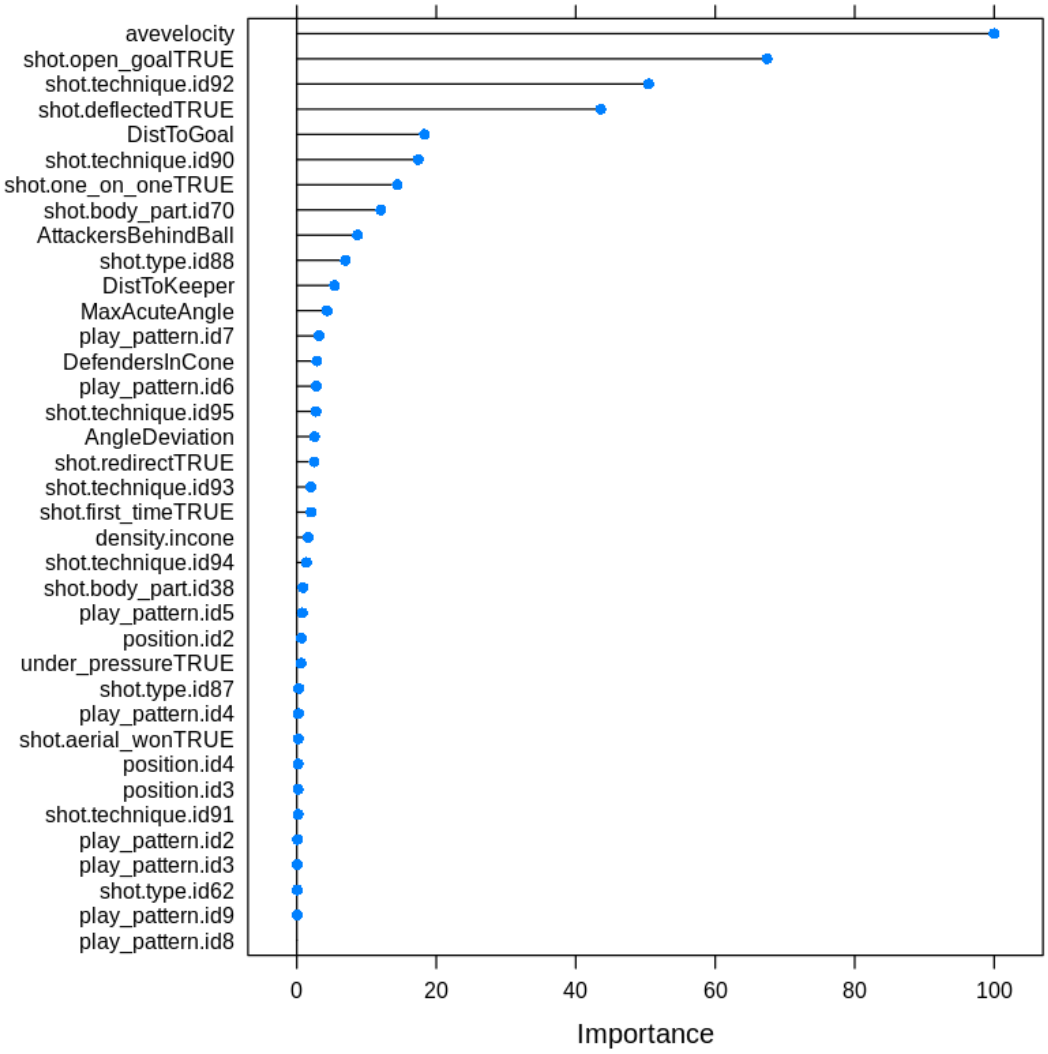
In R:

For using Random Forest in R, we use the randomForest library. We use ROSE method of resampling and k-fold repeated CV method to train our model. The number of folds is chosen to be 3 and repeated 3 times. The tuning parameter here is the number of randomly selected predictors out of all the predictors. We use ROC as a measure to fix our tuning parameter.

```
Random Forest

6561 samples
  20 predictor
   2 classes: 'NG', 'G'

No pre-processing
Resampling: Cross-Validated (3 fold, repeated 3 times)
Summary of sample sizes: 4374, 4374, 4374, 4374, 4374, 4374, ...
Addtional sampling using ROSE

Resampling results across tuning parameters:

  mtry  ROC        Sens       Spec
   2    0.7663294  0.9097187  0.2828704
  19    0.7447890  0.7259602  0.4115741
  37    0.6801438  0.7364036  0.3976852

ROC was used to select the optimal model using the largest value.
The final value used for the model was mtry = 2.
```



Here mtry denotes the number of randomly selected parameters. We see that when mtry=2, ROC is highest.
We also do a Variance Importance model to check which predictors have maximum influence on our classification.



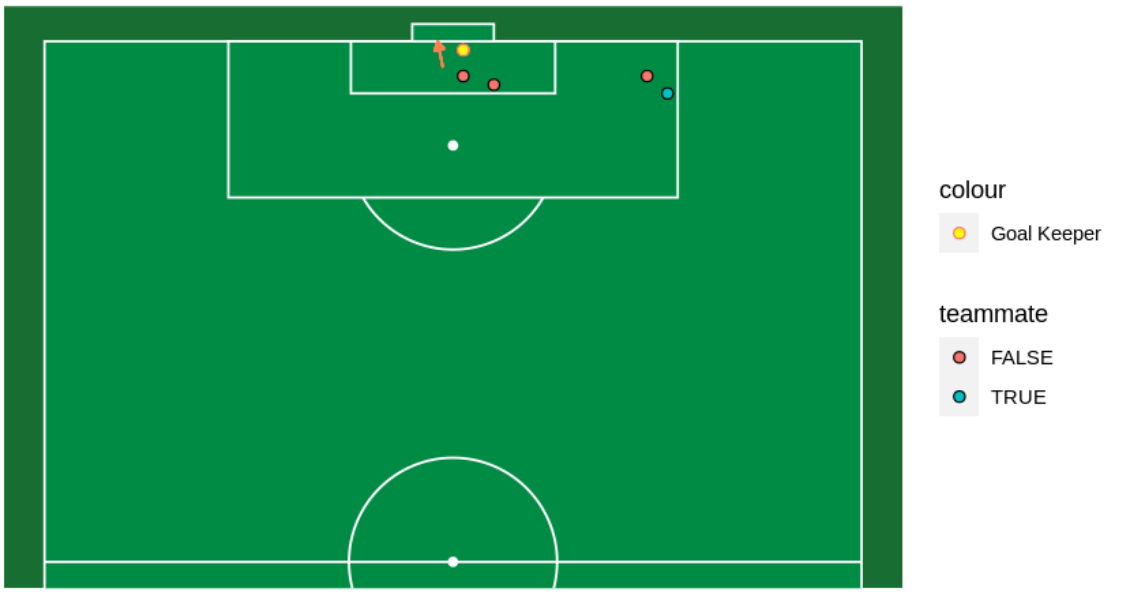
We use the fitted decision tree to make predictions on the training set.

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.0100  0.1100  0.1740  0.2102  0.2820  0.8360 
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
0.00000 0.02402 0.05221 0.10672 0.13263 0.90459 
0.6502246
```

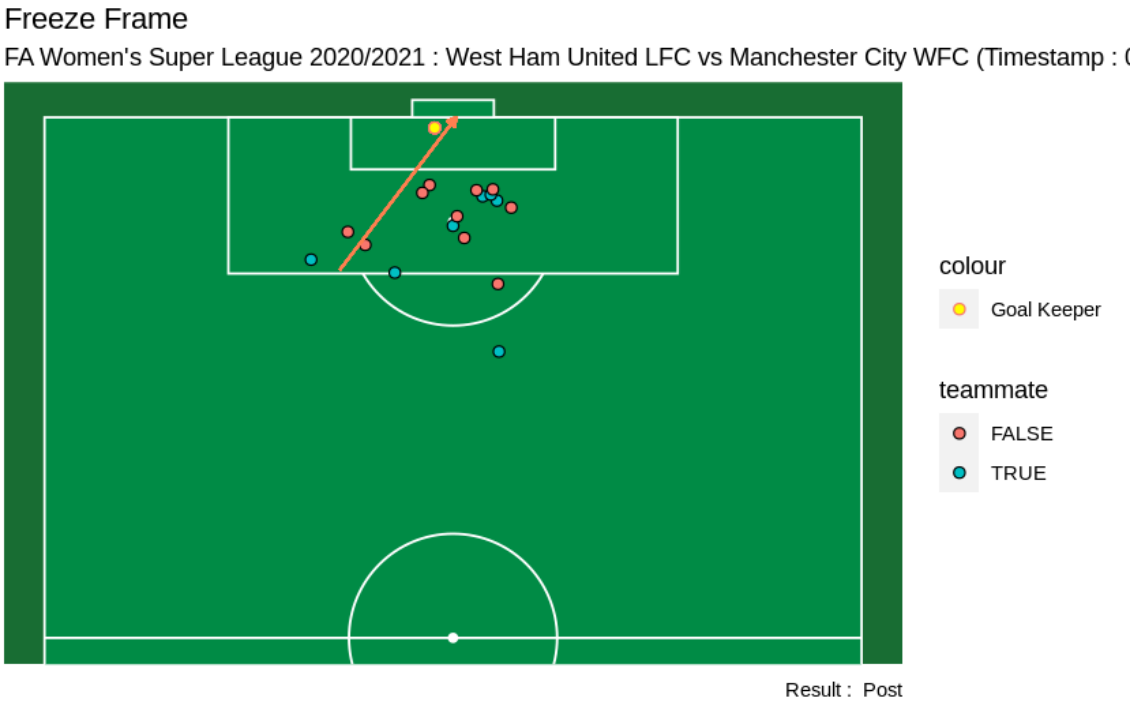
We again use the freeze frame of the shot with the highest and lowest predicted XG to check for the validity of our model.

Freeze Frame

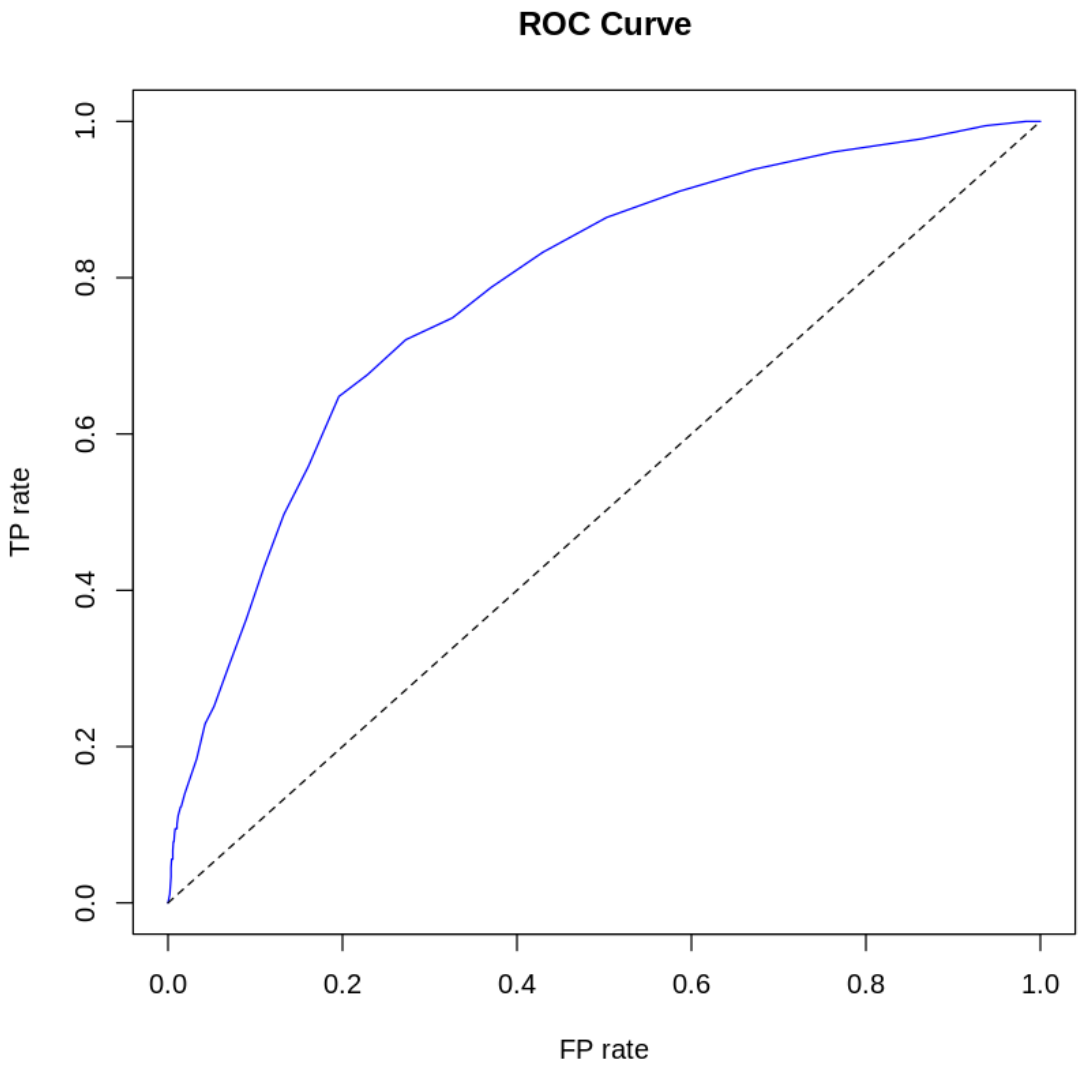
FA Women's Super League 2018/2019 : Liverpool WFC vs West Ham United LFC (Timestamp : 00:03:1



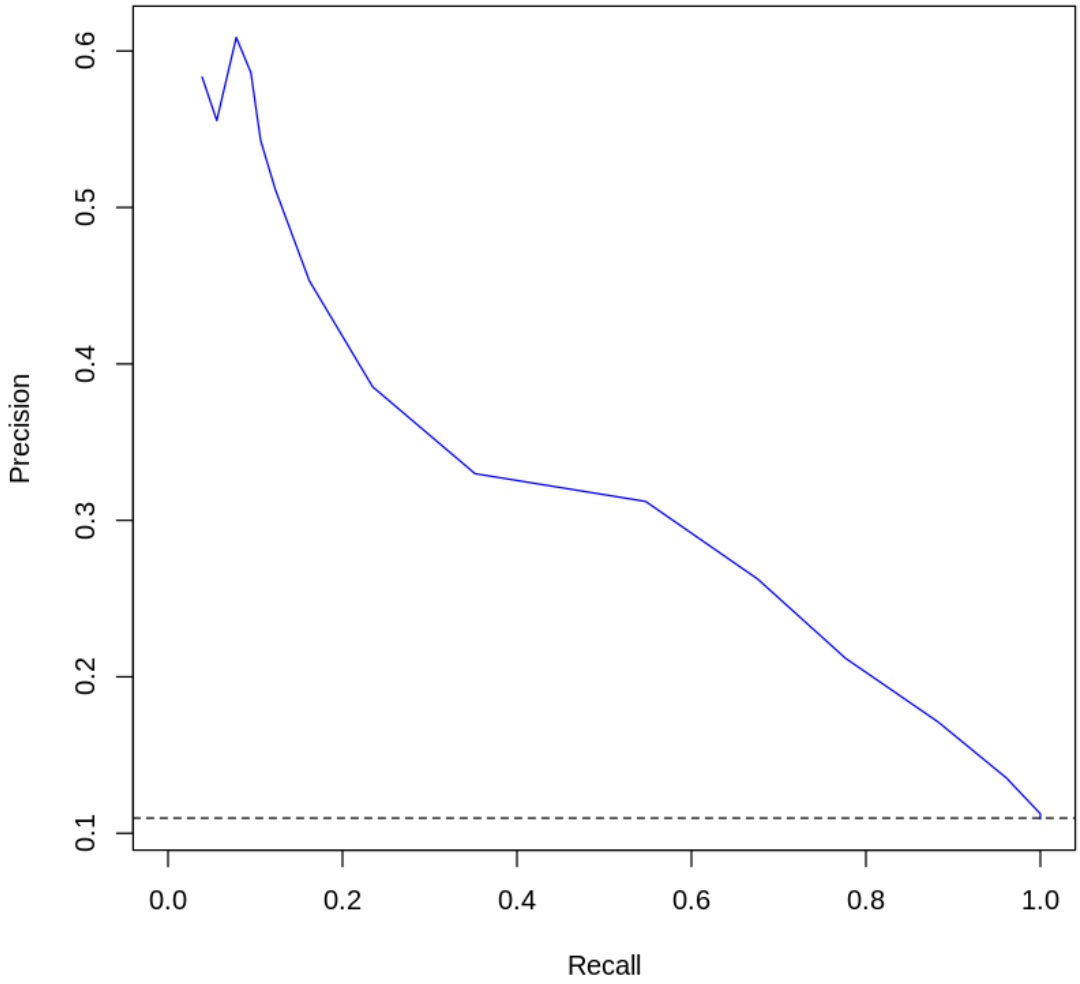
Result : Goal



Again we use ROC curve and PR curve to diagnose the effectiveness of our model.

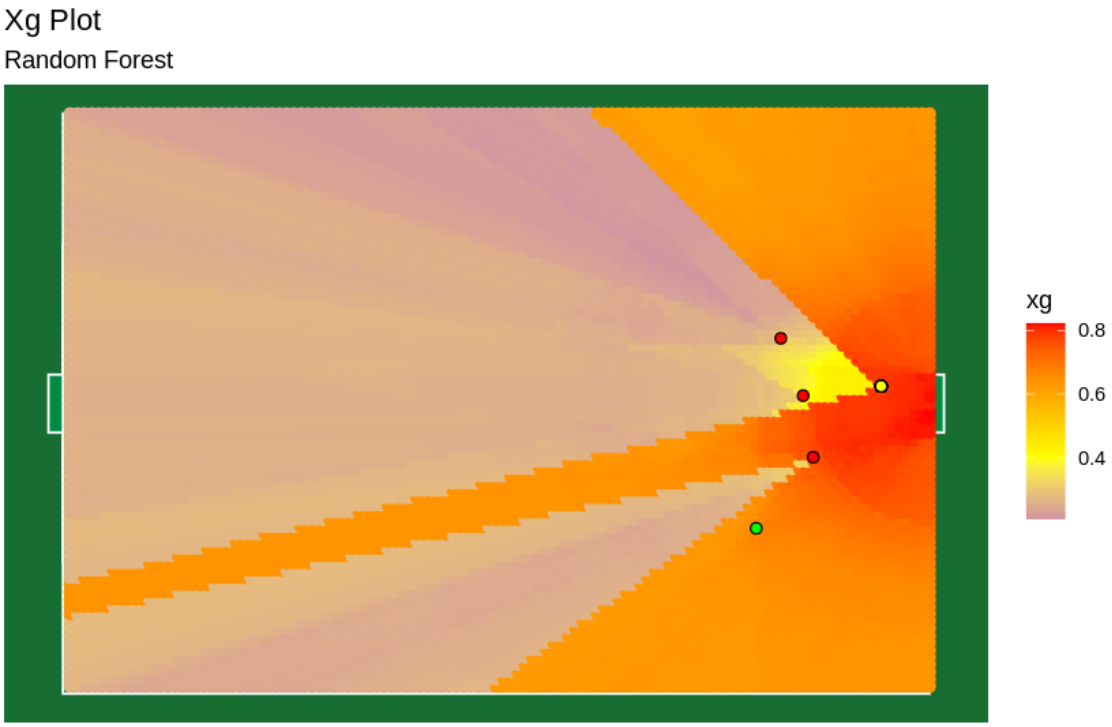


PR Curve



We see that the AUC for the ROC is much higher than the AUC of an estimator that is purely based on chance. Hence we can say that our classification model is effective.

Also from the confusion matrix, we see that the diagoanl entries have higher values, which confirms our interpretation.
Heatmap:

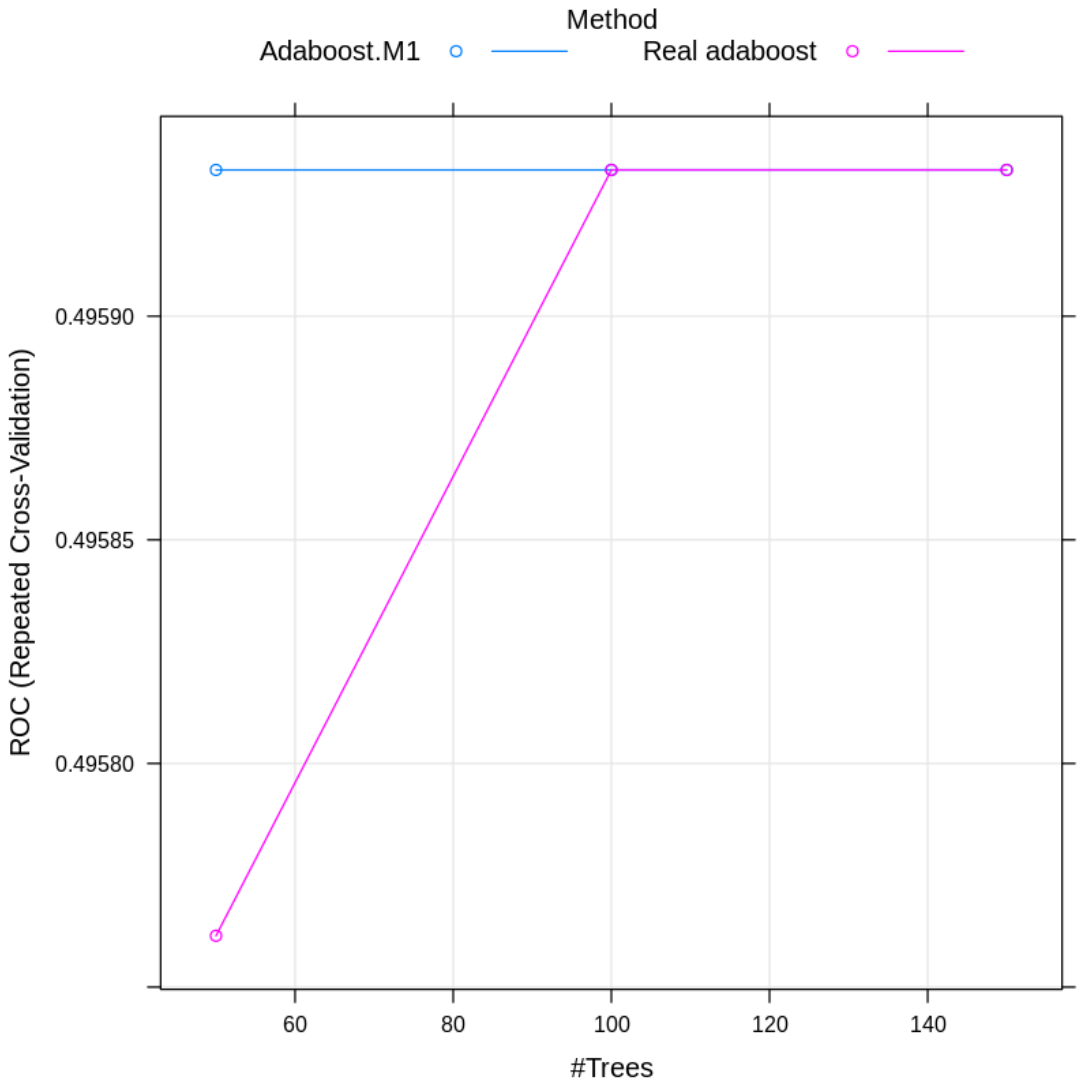


AdaBoost

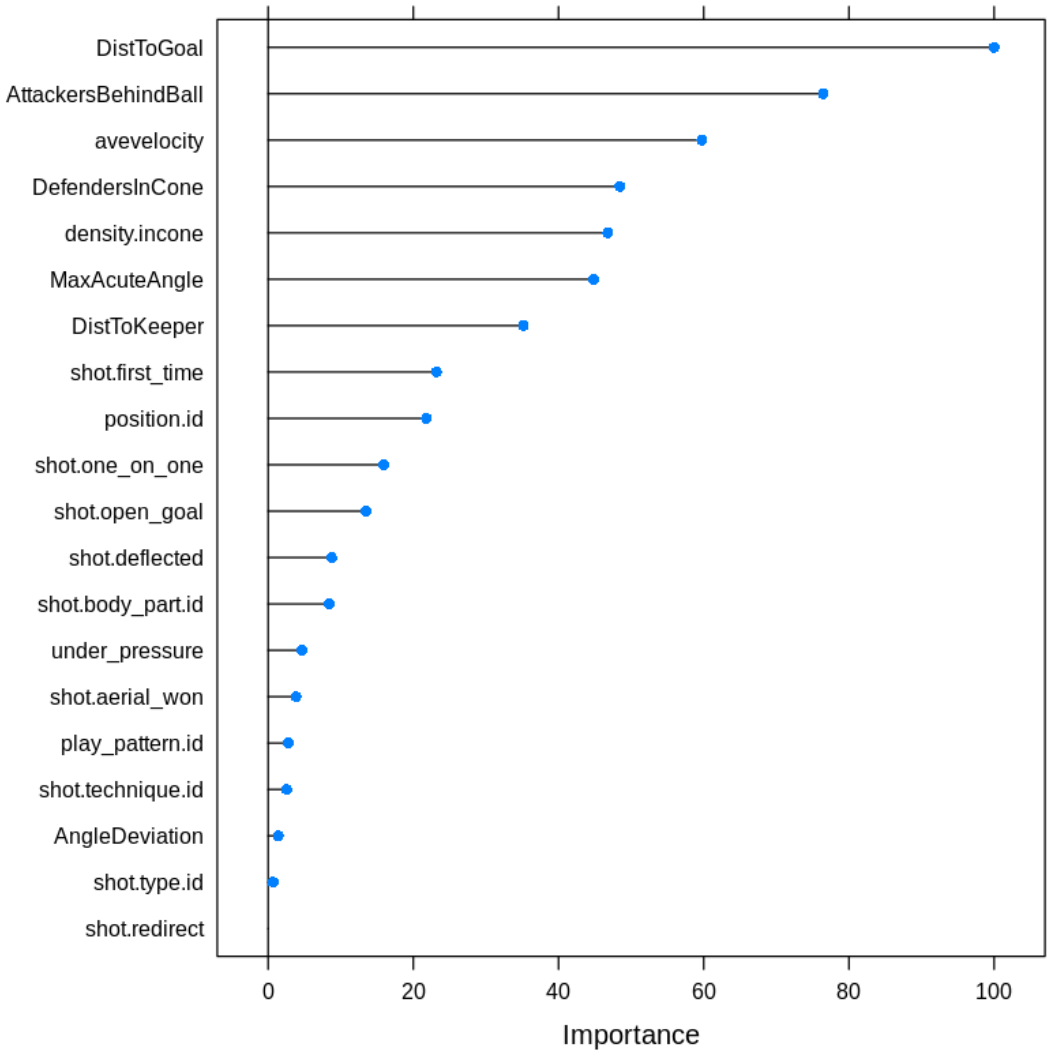
In R
For using AdaBoost in R, we use the fastAdaboost library. We use ROSE method of resampling and k-fold repeated CV method to train our model. The number of folds is chosen to be 2 and repeated 2 times. The tuning parameter here is the number of iterations. Also we have to

consider two methods of AdaBoost- AdaBoost.M1 and Real AdaBoost. We use ROC as a measure to fix our tuning parameter and method.

```
AdaBoost Classification Trees

6561 samples
  20 predictor
   2 classes: 'NG', 'G'

No pre-processing
Resampling: Cross-Validated (2 fold, repeated 2 times)
Summary of sample sizes: 3280, 3281, 3280, 3281
Addtional sampling using ROSE

Resampling results across tuning parameters:

  nIter  method         ROC        Sens         Spec
   50    Adaboost.M1    0.4959327  0.004365412  0.9875
   50    Real adaboost  0.4957615  0.004022947  0.9875
  100    Adaboost.M1    0.4959327  0.004365412  0.9875
  100    Real adaboost  0.4959327  0.004365412  0.9875
  150    Adaboost.M1    0.4959327  0.004365412  0.9875
  150    Real adaboost  0.4959327  0.004365412  0.9875

ROC was used to select the optimal model using the largest value.
The final values used for the model were nIter = 50 and method = Adaboost.M1.
```



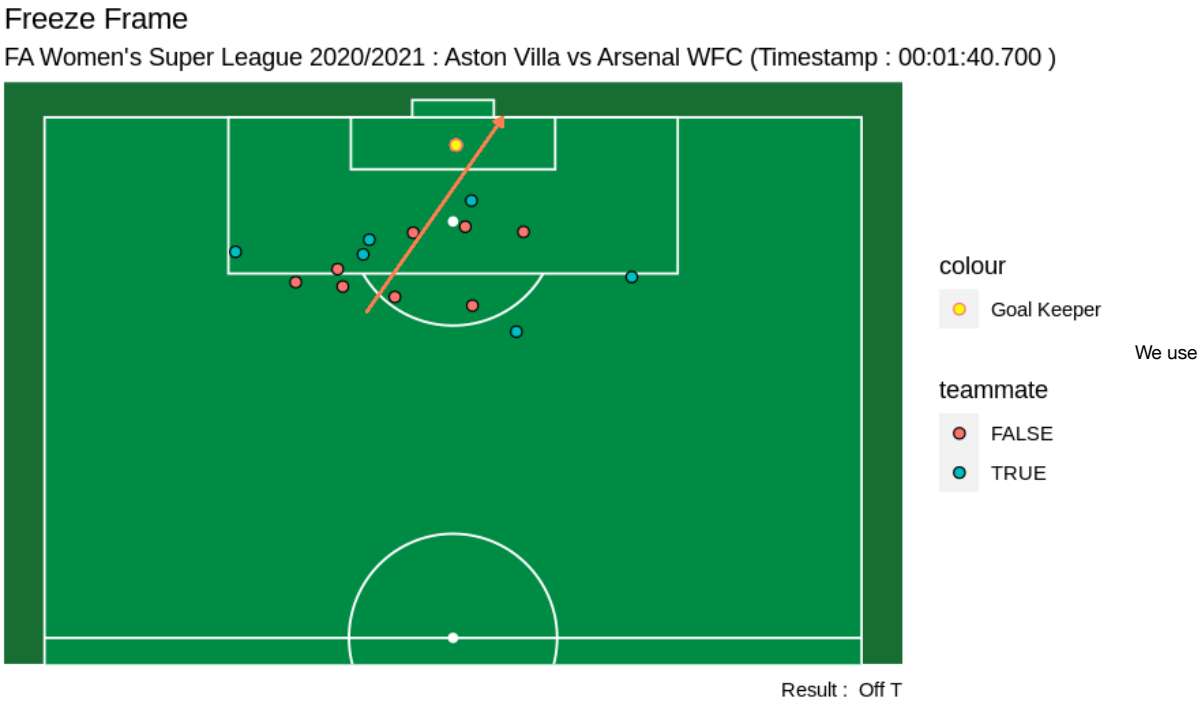
We see that the maximum ROC is obtained when we use AdaBoost.M1 with number of iterations as 50.
Plotting a Variance Importance plot.



We use this model to make predictions on the test dataset.

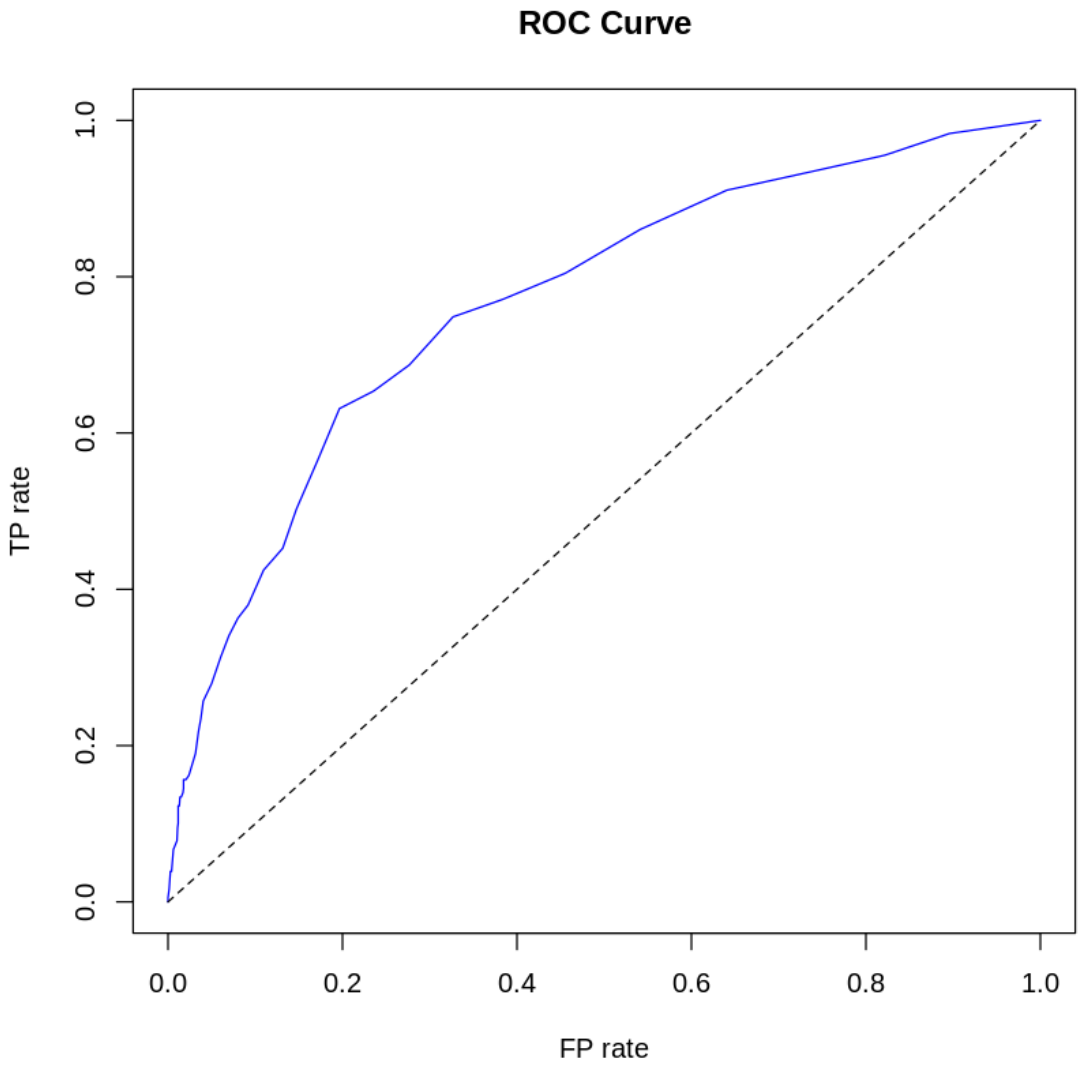
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.06192 0.12135 0.17295 0.22099 0.96074
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.02402 0.05221 0.10672 0.13263 0.90459
```

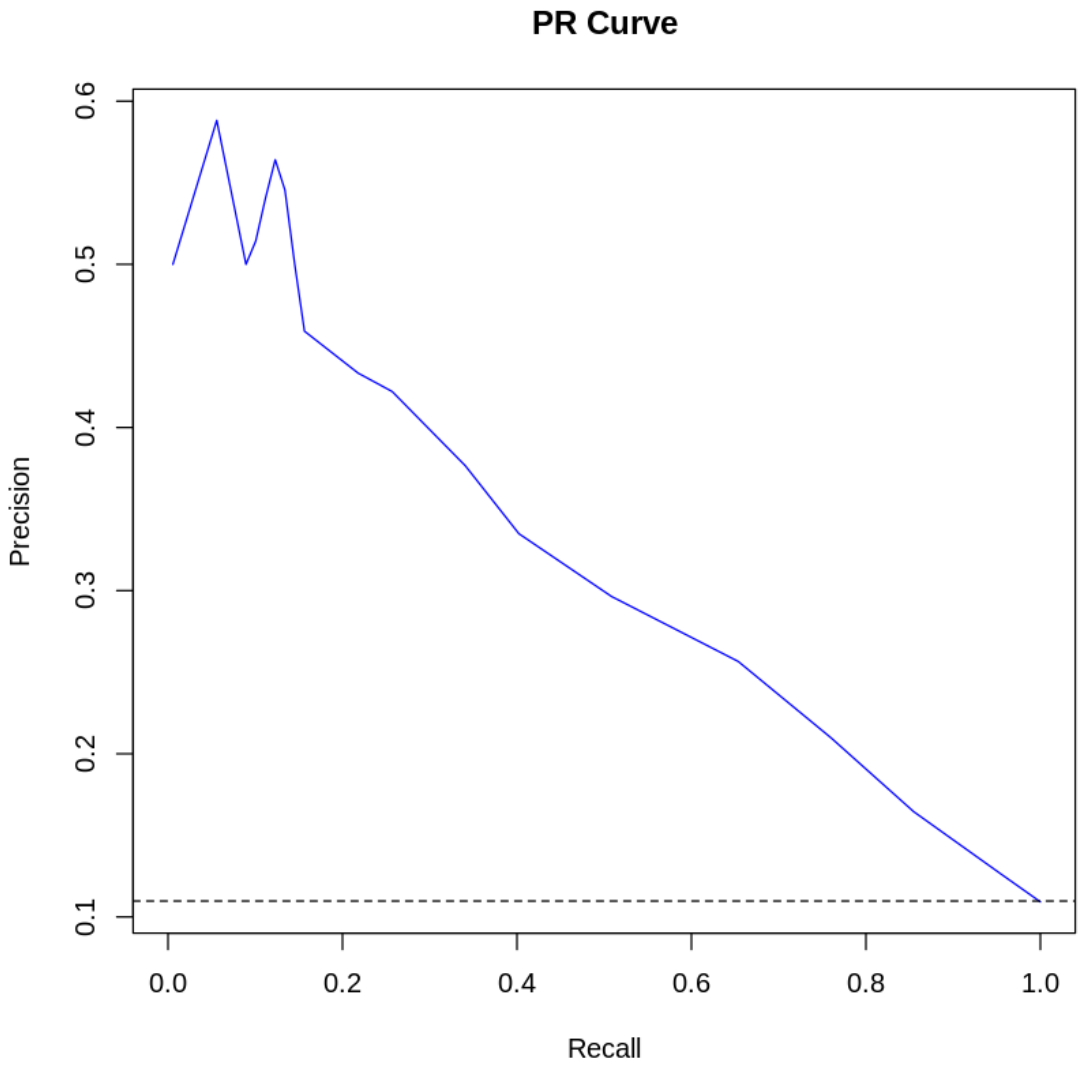
Again we plot the freeze frames of the shots with the maximum and minimum predicted xG, to check for the validity of our model.



We use

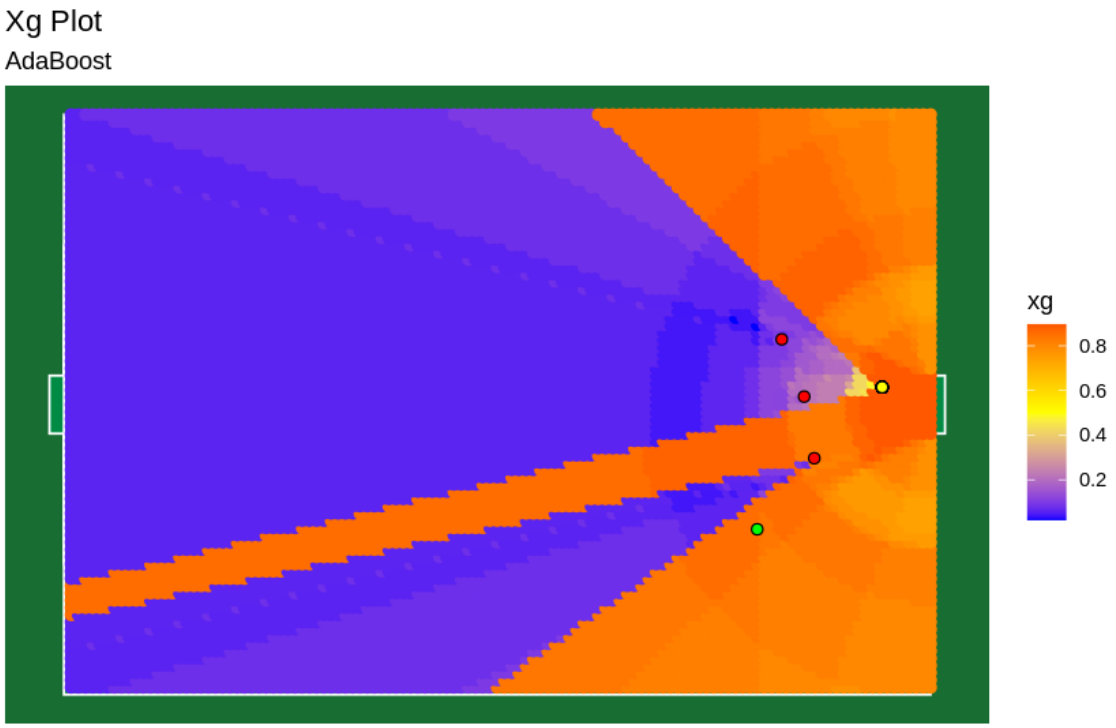
ROC and PR curves to diagnose the efficiency of our model.


ROC Curve
TP rate
FP rate
0.0
0.2
0.4
0.6
0.8
1.0



The AUC for the ROC is much higher than what we expect from a model that is purely based on chance. Hence we can say that our model is efficient. Further the diagonal elements of the confusion matrix has high values, which confirms our conclusion.

Heatmap:

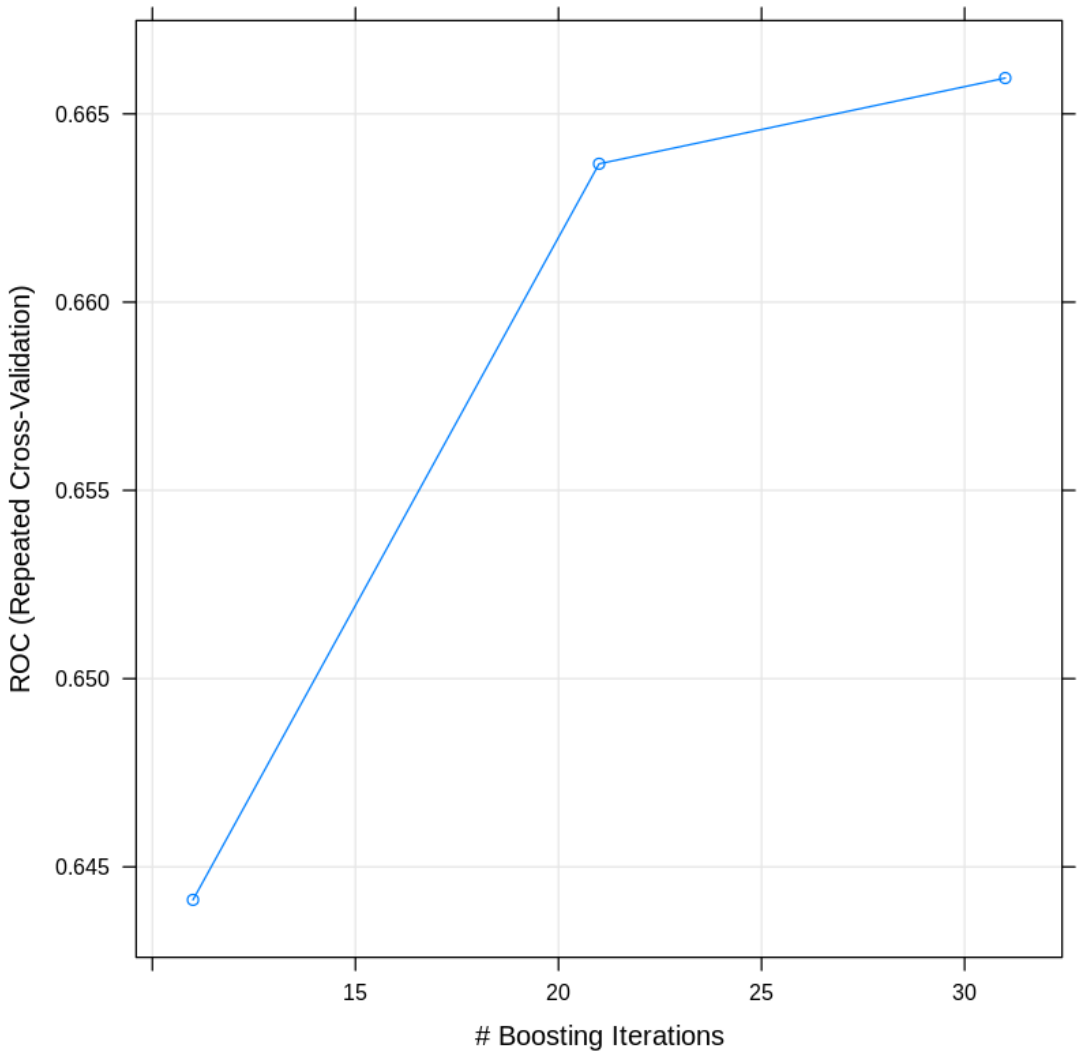


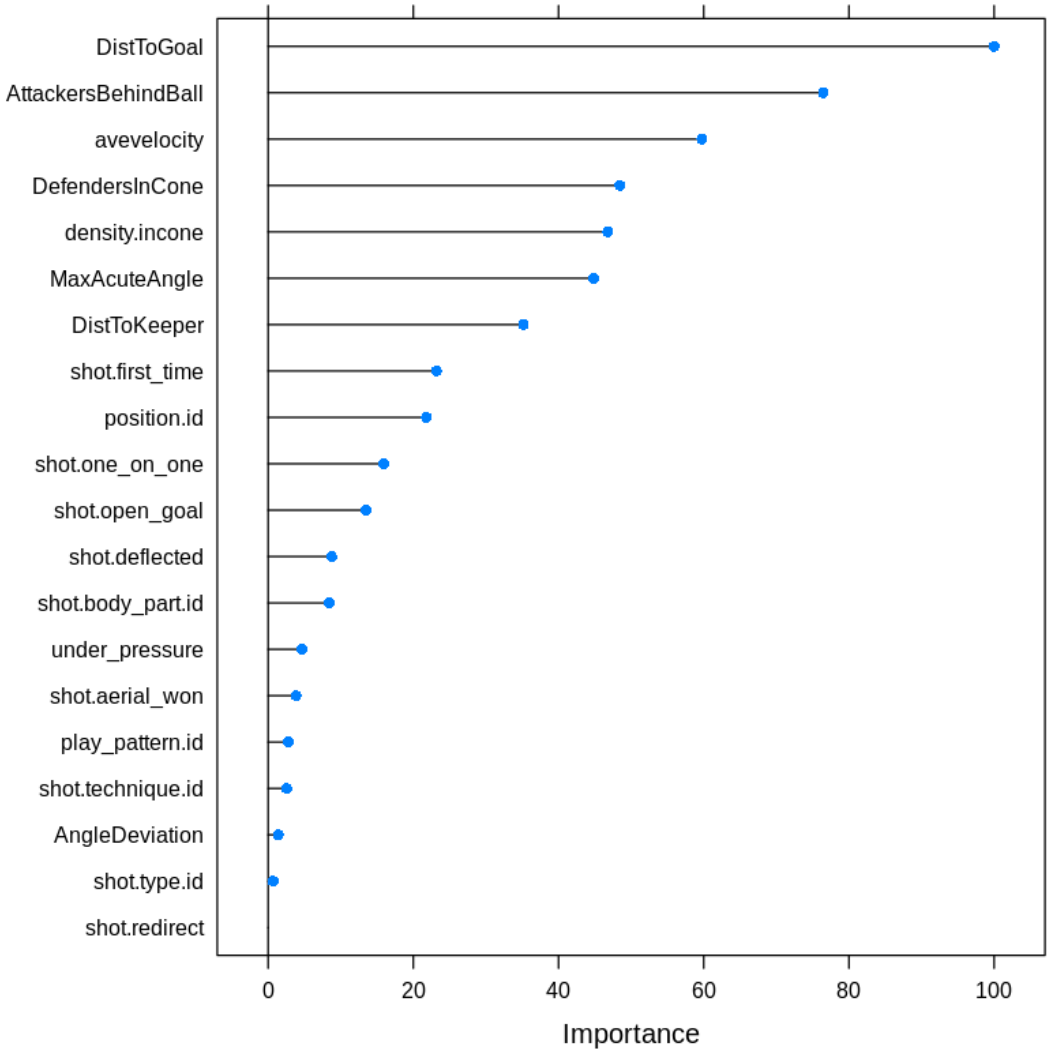
Boosted Logistic Regression:

In R:

We use the caTools package in R to fit the boosted logistic regression. We are using the Repeated Cross Validation (10 folds, 3 repetitions) method to find the hyperparameter which is the number of iterations.

```
Boosted Logistic Regression

6561 samples
  20 predictor
   2 classes: 'NG', 'G'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 5905, 5905, 5905, 5905, 5905, 5904, ...
Addtional sampling using ROSE

Resampling results across tuning parameters:

  nIter  ROC        Sens       Spec
  11     0.6441240  0.3303758  0.8842593
  21     0.6636751  0.6495784  0.5944444
  31     0.6659522  0.8172439  0.3782407

ROC was used to select the optimal model using the largest value.
The final value used for the model was nIter = 31.
```





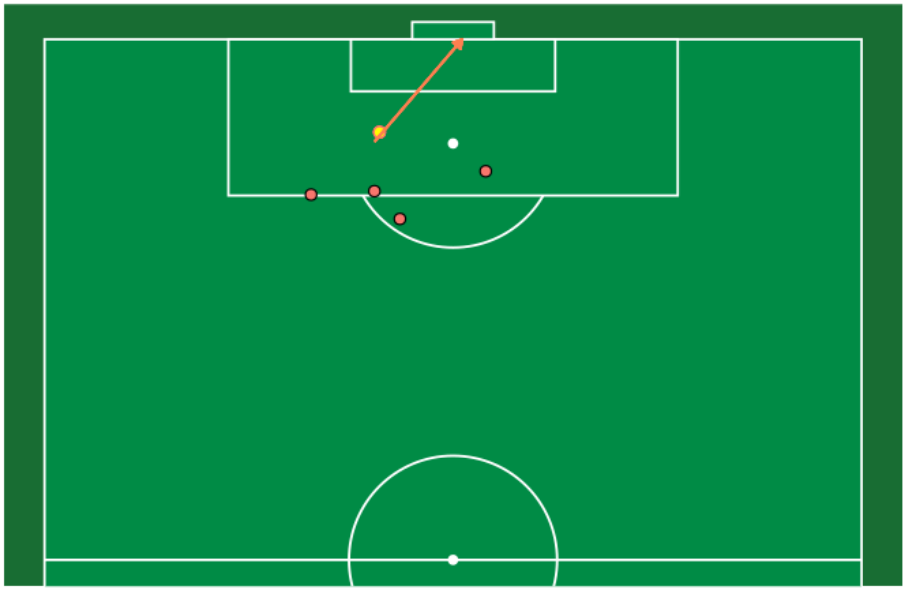
We use the fitted model to make predictions on the test data. The summary of the predictions made are as follows:

```
      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0001234 0.2689414 0.2689414 0.4268658 0.7310586 0.9933071
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.02402 0.05221 0.10672 0.13263 0.90459
0.16402765
```

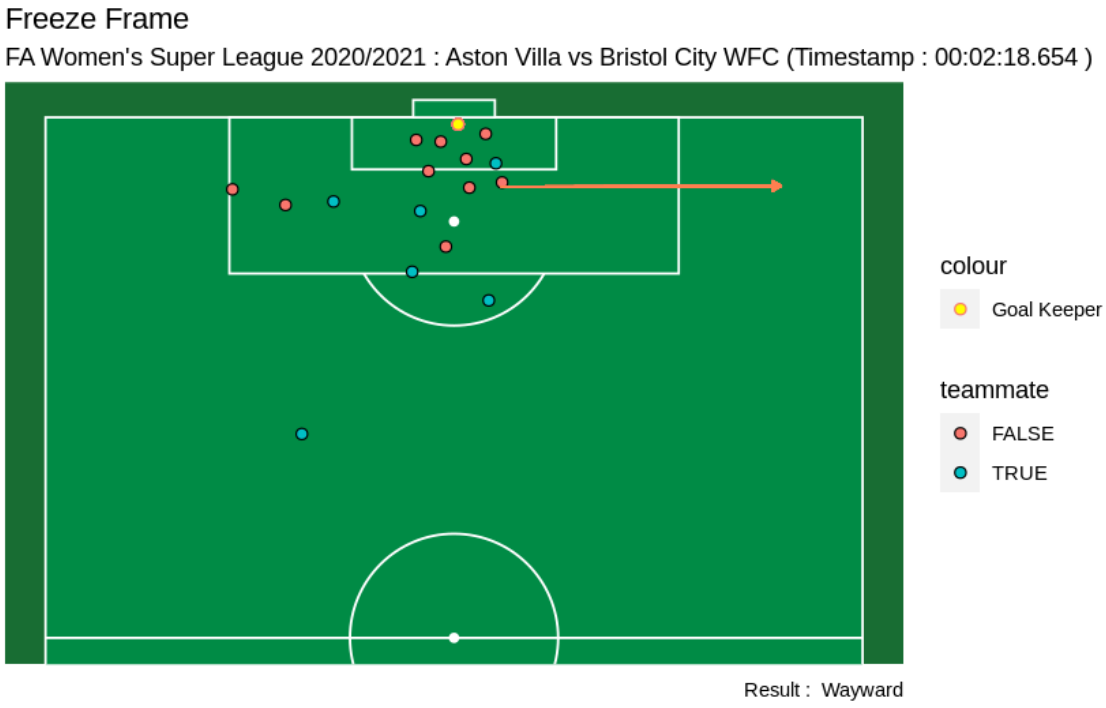
We again plot the freeze frames of the shots with the highest and lowest predicted xG's so as to compare the accuracy of our model.

Freeze Frame

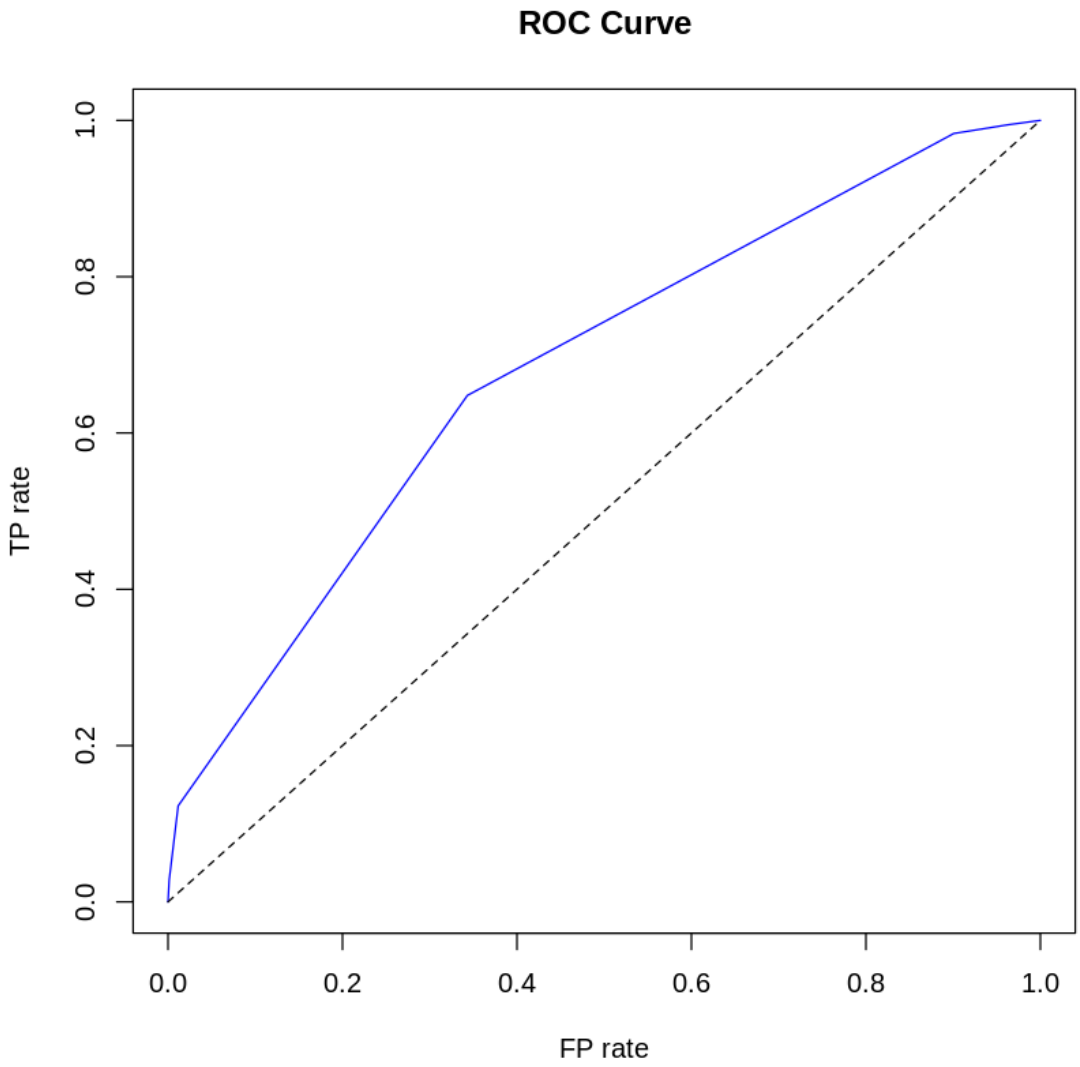
FA Women's Super League 2020/2021 : Tottenham Hotspur Women vs Everton LFC (Timestamp : 00:34

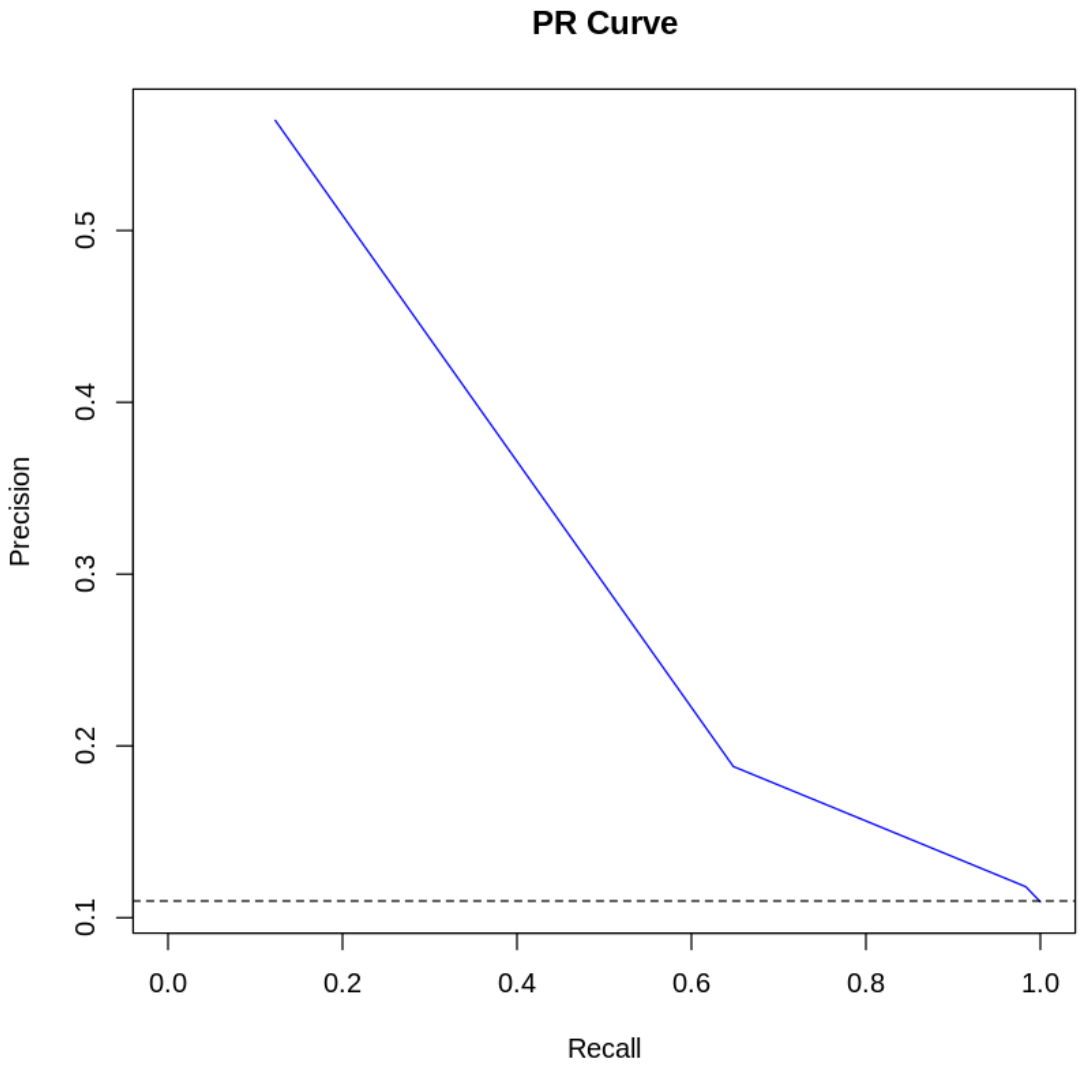


Result : Goal



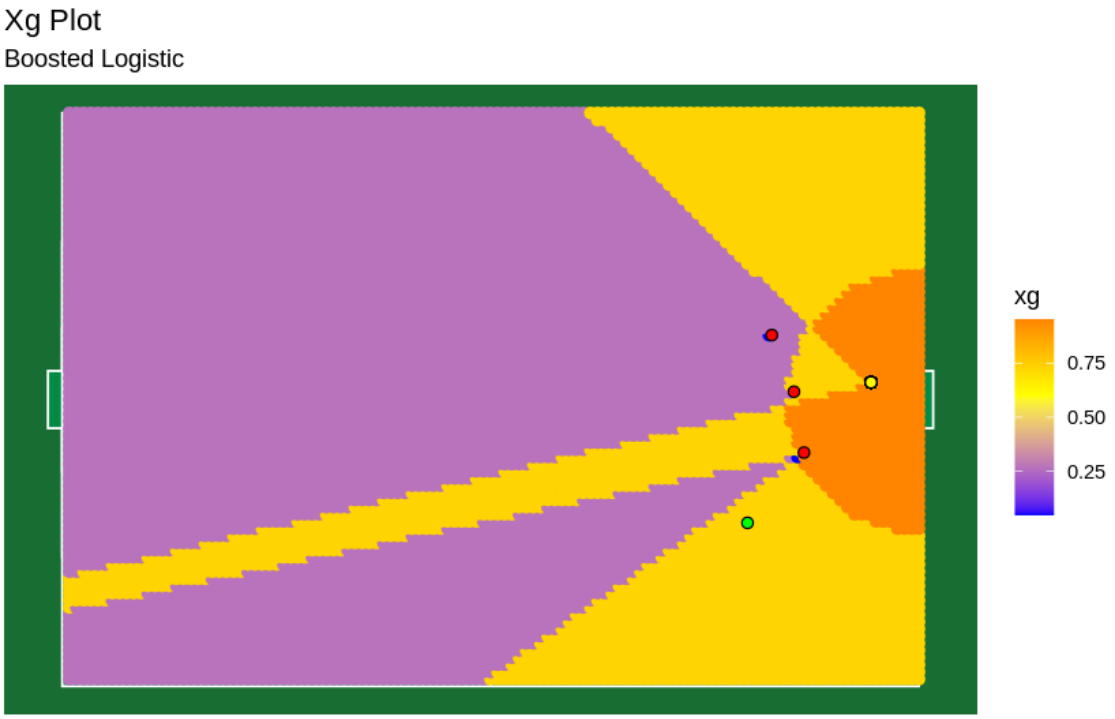
We will make use of ROC and PR curves tlo diagnose the performace of our classifier.


ROC Curve
TP rate
FP rate
0.0
0.2
0.4
0.6
0.8
1.0



We see that the ROC AUC is much higher than a classifier that would work purely based on chance, hence we can say that the classifier is a good one.

Heatmap:



Stacking:

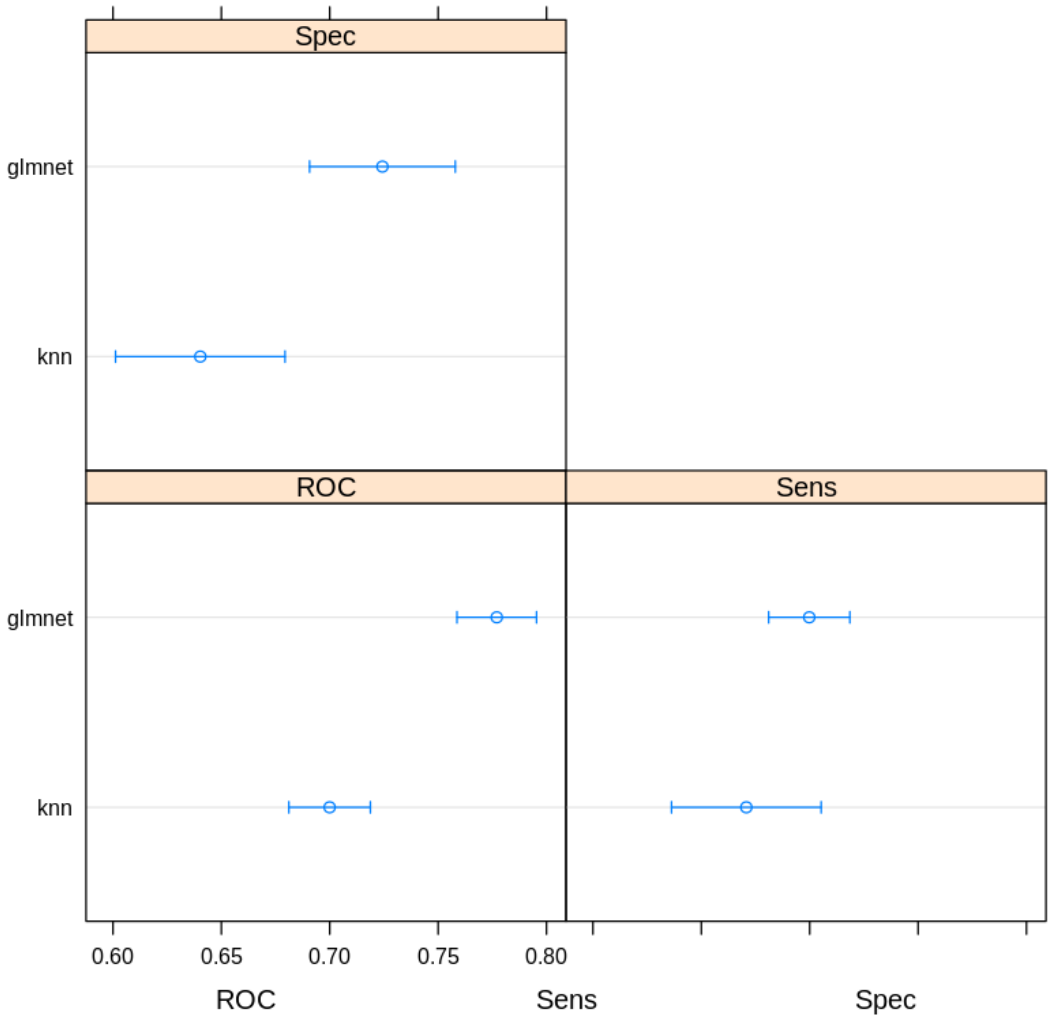
We will use the following 4 models for stacking.
1> Level 0 models: Penalized Logistic Regression, K Nearest Neighbors
Level 1 model: Logistic Regression
2> Level 0 models: Penalized Logistic Regression, K Nearest Neighbors
Level 1 model: Gradient Boost Machine (gbm)
3> Level 0 models: Penalized Logistic Regression, Decision Tree
Level 1 model: Logistic Regression
4> Level 0 models: Penalized Logistic Regression, Decision Tree
Level 1 model: Gradient Boost Machine (gbm)

We have seen that the penalized logistic model works great as a linear model to estimate Xg of shot. Hence, we observe the falloff of Xg in its heat map as the DistanceToGoal increases. On the other hand, Decision Trees and k-nn are good non linear classifiers that upon testing had marginal correlation with the penalized logistic model. We also observed that decision trees like to overfit on the test data and tend to give very extreme results for class probabilities (as seen in heat map). We can solve both of these issues by stacking these models with the glmnet model as base models. Further, for the meta-model we can test if using a non-linear model gives better performance than a linear model by considering Logistic Regression and GBM in layer 1.

Penalized Logistic Regression, kNN as Base models:

We again use ROSE for resampling. To train the models, we use k-fold repeated CV method where the number of folds is taken to be 5, with 2 repetitions.
Hence for each base model, we get 10 Sensitivity, Specificity and ROC values. We find the summary of these, and they are plotted on the same graph as a means of comparison between the two base models.

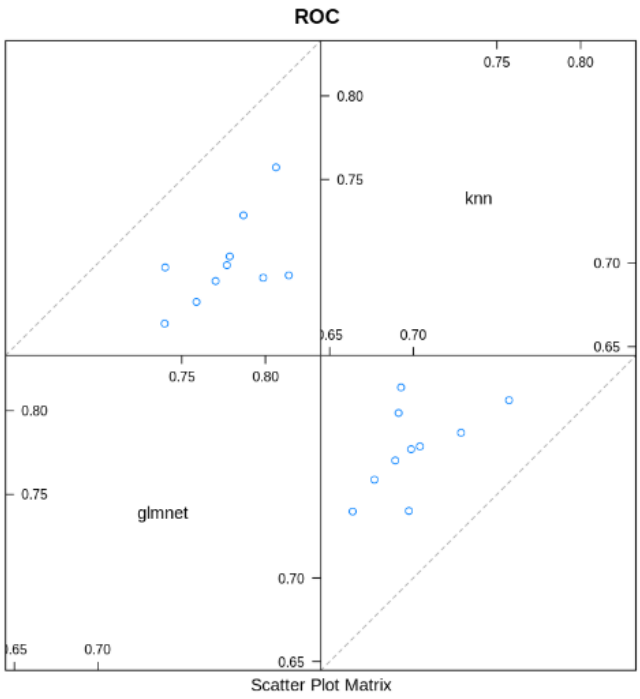
```
Call:
summary.resamples(object = stacking_results)

Models: glmnet, knn
Number of resamples: 10

ROC
            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
glmnet 0.7397736 0.7616519 0.7778491 0.7770525 0.7957404 0.8139983    0
knn    0.6637682 0.6897251 0.6950184 0.6999463 0.7026538 0.7572388    0

Sens
            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
glmnet 0.6618151 0.6810713 0.7003425 0.6997991 0.7206764 0.7371575    0
knn    0.5813356 0.6508990 0.6709472 0.6707671 0.6785103 0.7733105    0

Spec
            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
glmnet 0.6319444 0.6961806 0.7326389 0.7243056 0.7621528 0.7777778    0
knn    0.5277778 0.6145833 0.6458333 0.6402778 0.6701389 0.7222222    0
```



To ensure the effectiveness of stacking algorithm, we further check the correlation of prediction outcomes of the underlying models. If we see that the correlation is less than 0.75, we can say that the stacking can be effective.

A matrix: 2 × 2 of type dbl

	glmnet	knn
glmnet	1.0000000	0.5678742
knn	0.5678742	1.0000000



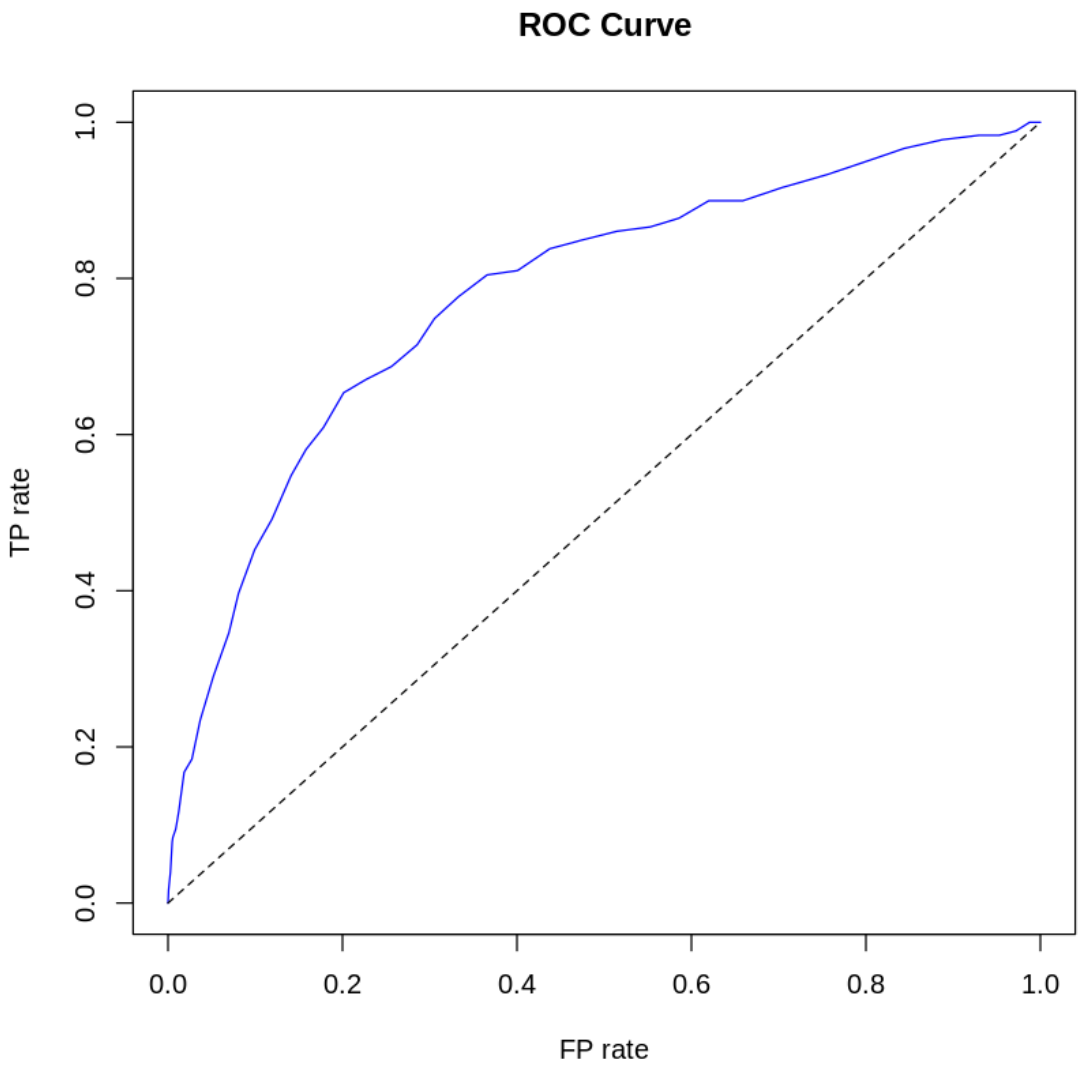
We see that the value of correlation <0.75, implying that we can move along with our stacking.
Now we will use 2 meta models- the logistic regression and the gradient boost machine.

Using Logistic Regression:

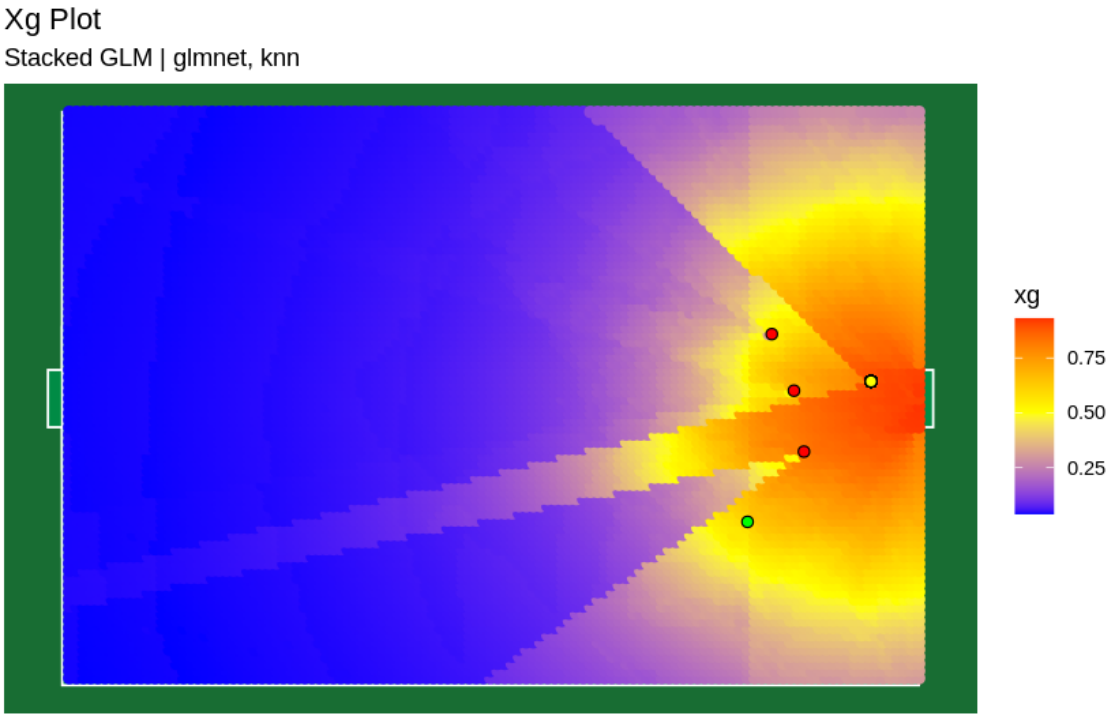
```
A glm ensemble of 2 base models: glmnet, knn

Ensemble results:
Generalized Linear Model

13122 samples
    2 predictor
    2 classes: 'NG', 'G'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 2 times)
Summary of sample sizes: 10498, 10498, 10497, 10497, 10498, 10498, ...
Addtional sampling using ROSE

Resampling results:

  ROC        Sens       Spec
  0.7753607  0.7025768  0.71875

Confusion Matrix and Statistics

          Reference
Prediction   NG    G
        NG 1030   46
        G   430  133

               Accuracy : 0.7096
                 95% CI : (0.6869, 0.7315)
    No Information Rate : 0.8908
    P-Value [Acc > NIR] : 1
```

```
                 Kappa : 0.2311

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.74302
            Specificity : 0.70548
         Pos Pred Value : 0.23623
         Neg Pred Value : 0.95725
             Prevalence : 0.10921
         Detection Rate : 0.08115
   Detection Prevalence : 0.34350
      Balanced Accuracy : 0.72425

       'Positive' Class : G
```

From the confusion matrix, we again see that we have high entries on the diagonals, implying the classification model is a good one. Also, we see that the model has classified the test set with an accuracy of 0.7096.
We will use other diagnostic measures like ROC AUC.



Again, we see that the classifier performs much better than a classifier that would perform just based on chance.
Heatmap:



Using gbm:

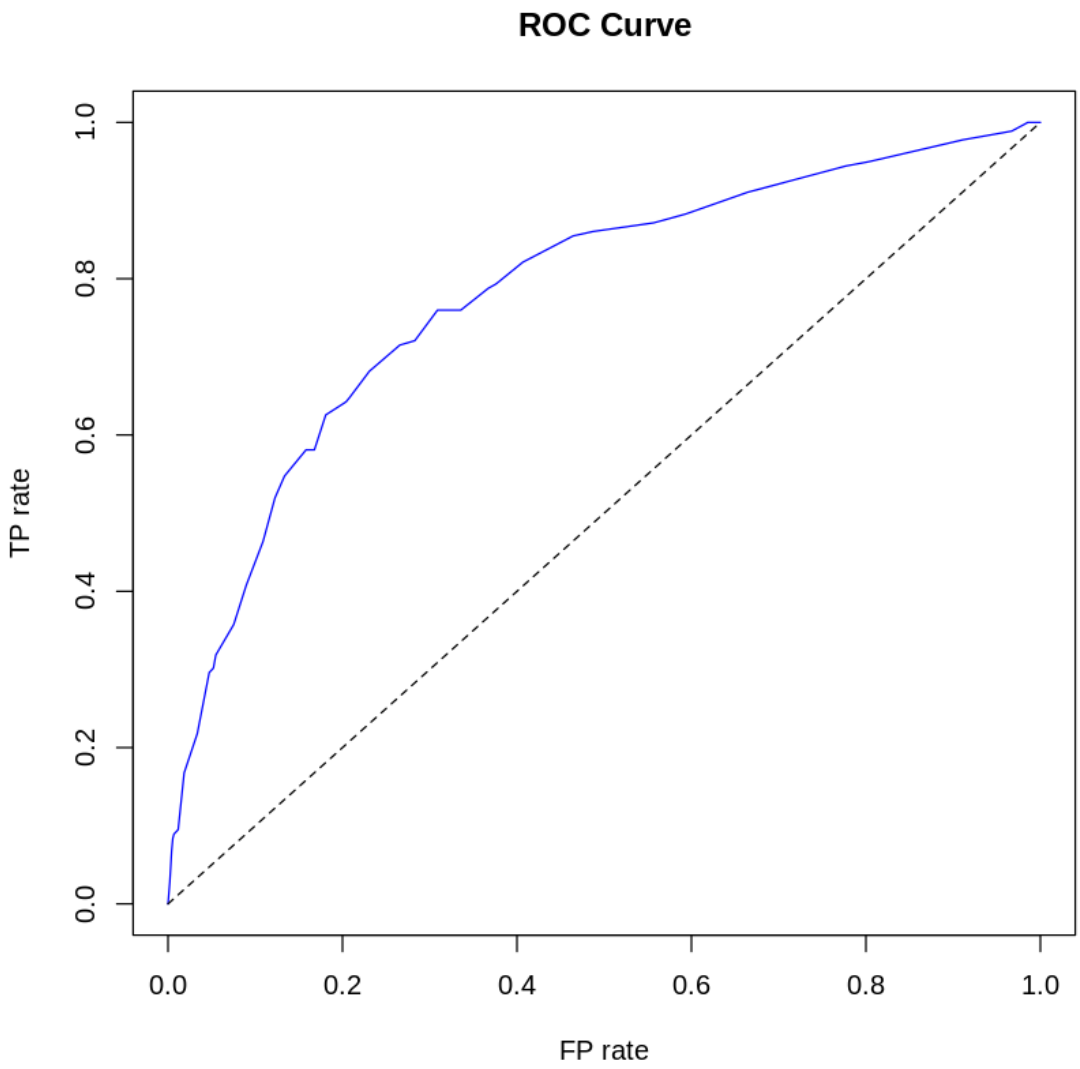
Now we use gbm as the meta model.

```
Ensemble results:
Stochastic Gradient Boosting

13122 samples
    2 predictor
    2 classes: 'NG', 'G'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 2 times)
Summary of sample sizes: 10497, 10498, 10498, 10498, 10497, 10498, ...
Addtional sampling using ROSE

Resampling results across tuning parameters:

  interaction.depth  n.trees  ROC        Sens       Spec
  1                   50      0.7748452  0.7319812  0.6875000
  1                  100      0.7753141  0.7315964  0.6871528
  1                  150      0.7754448  0.7379306  0.6809028
  2                   50      0.7746848  0.7399864  0.6781250
  2                  100      0.7732374  0.7414843  0.6756944
  2                  150      0.7730901  0.7405851  0.6763889
  3                   50      0.7751767  0.7395566  0.6815972
  3                  100      0.7736703  0.7384445  0.6822917
  3                  150      0.7729436  0.7377601  0.6812500

Tuning parameter 'shrinkage' was held constant at a value of 0.1
```

```
          Reference
Prediction   NG    G
        NG 1116   56
        G   344  123

               Accuracy : 0.7559
                 95% CI : (0.7344, 0.7766)
    No Information Rate : 0.8908
    P-Value [Acc > NIR] : 1

                  Kappa : 0.2647

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.68715
```

```
            Specificity : 0.76438
         Pos Pred Value : 0.26338
         Neg Pred Value : 0.95222
             Prevalence : 0.10921
         Detection Rate : 0.07505
   Detection Prevalence : 0.28493
      Balanced Accuracy : 0.72577

       'Positive' Class : G
```

We attain an accuracy of 0.7559. From the confusion matrix, we see that we have high values for the diagonal entries, which points to this being a good model.
We will also use other diagnostic measures like ROC AUC.



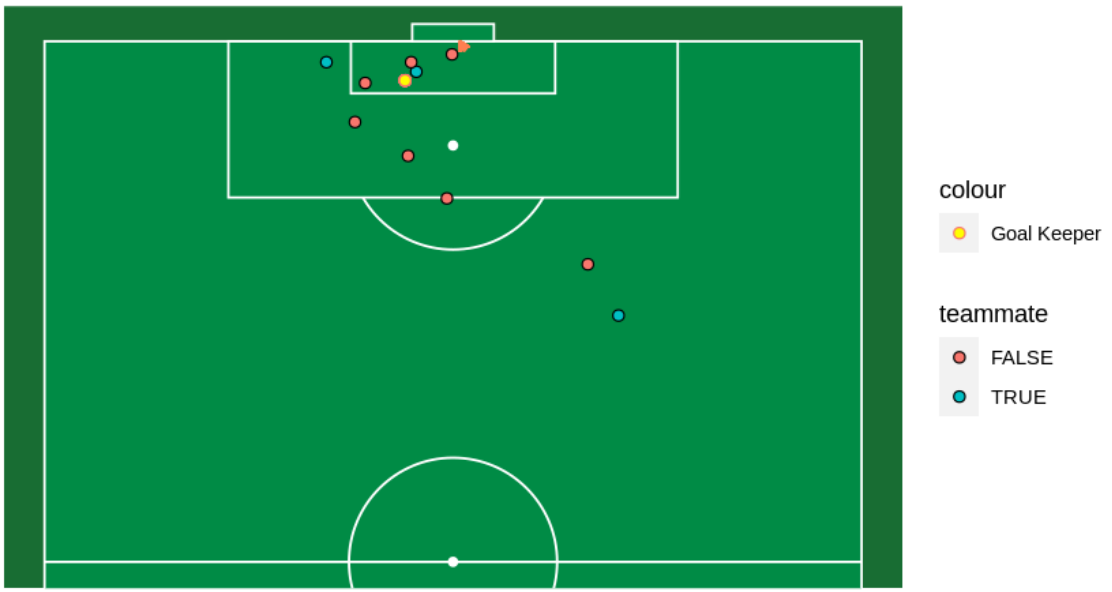
Again we see for this model, the ROC AUC is much higher than what would be expected of a model purey based on chance.
Now we will use this model to make predictions on the test data.

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1364  0.2555  0.3602  0.4117  0.5639  0.9479
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.02402 0.05221 0.10672 0.13263 0.90459
0.9045875
```

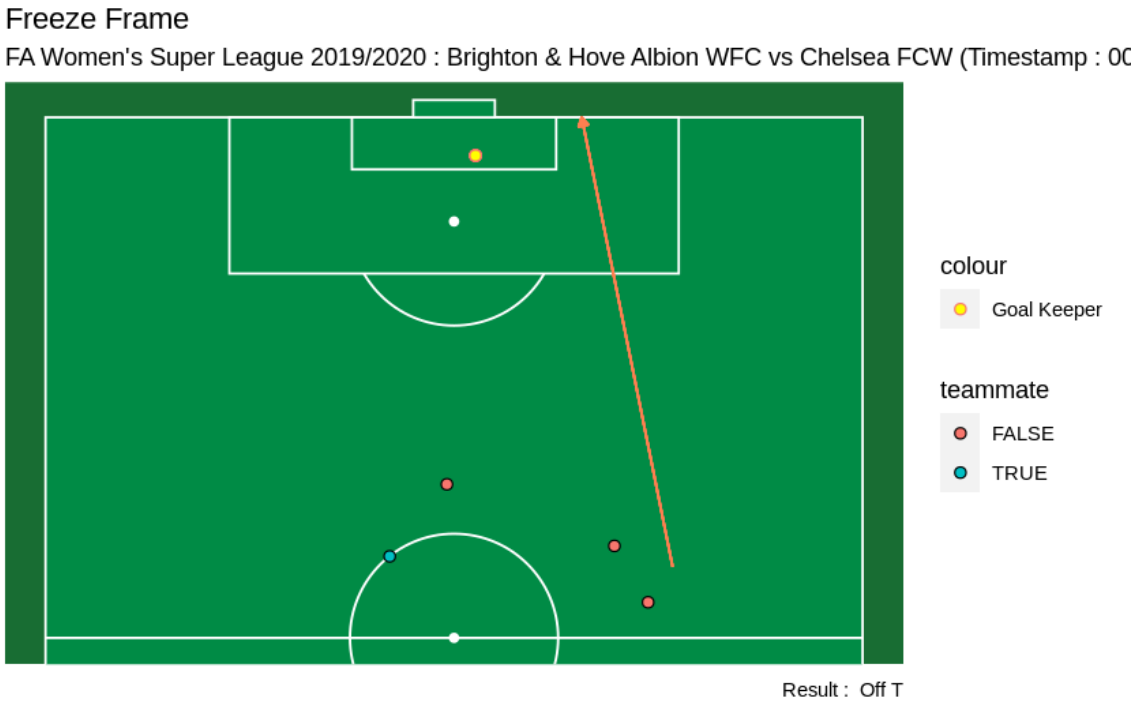
We plot the freeze frames of the shots with the highest and lowest predicted xG's.

Freeze Frame

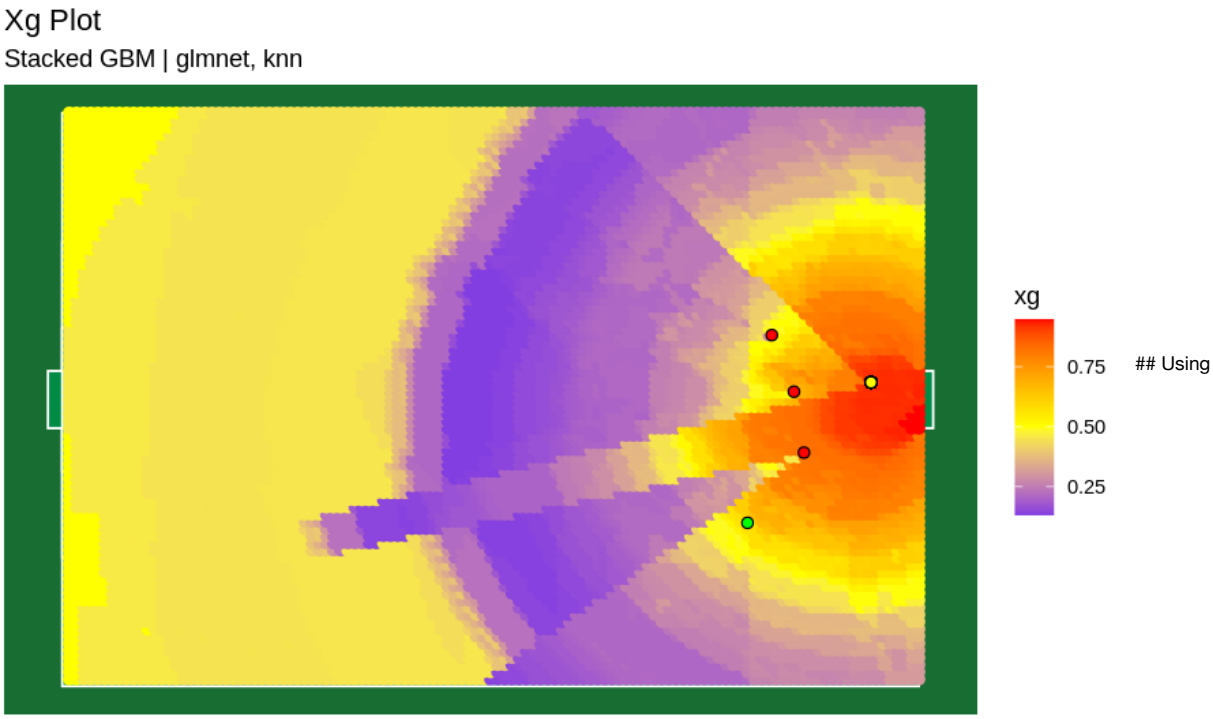
FA Women's Super League 2020/2021 : Bristol City WFC vs Arsenal WFC (Timestamp : 00:19:07.903)



Result : Goal

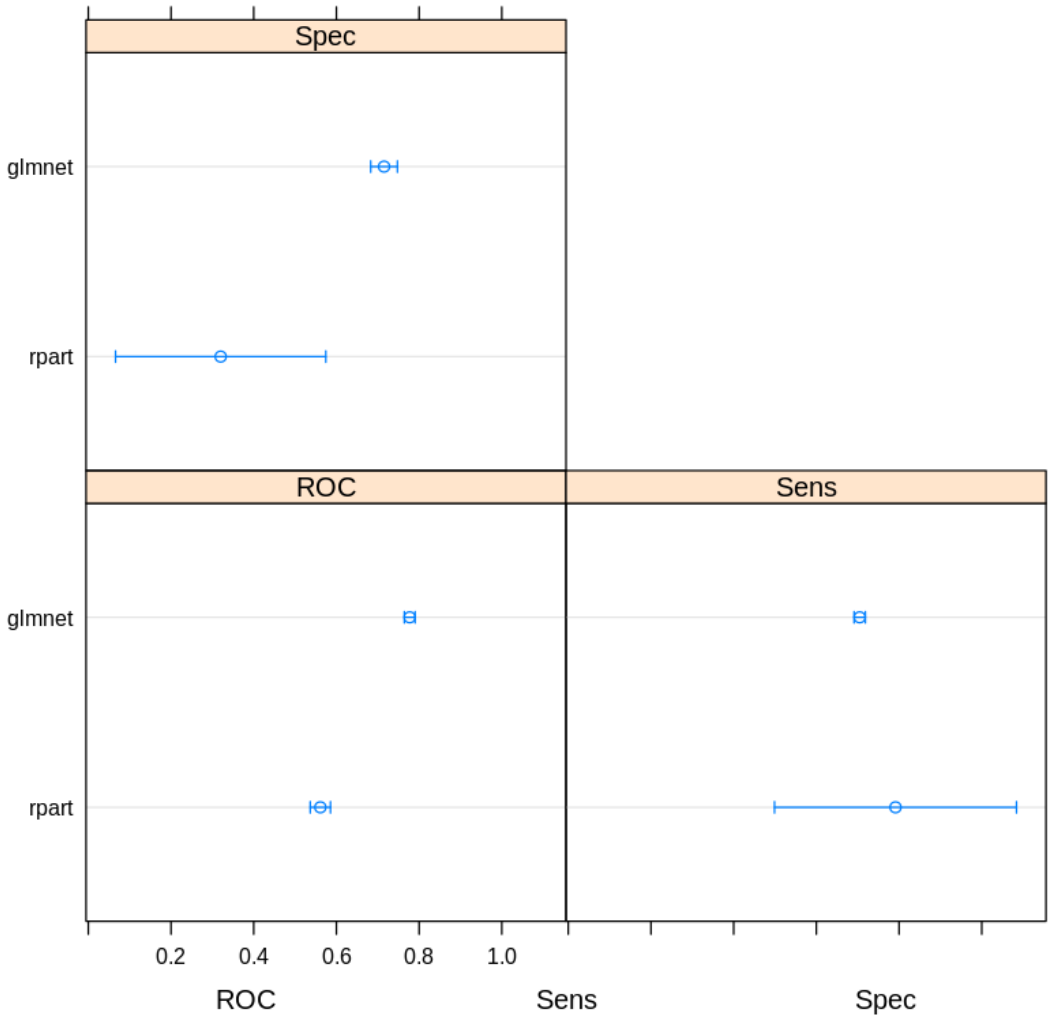

Freeze Frame
FA Women's Super League 2019/2020 : Brighton & Hove Albion WFC vs Chelsea FCW (Timestamp : 0(
colour
Goal Keeper
teammate
FALSE
TRUE
Result : Off T

Heatmap:



Penalised Logistic Regression and Decision Trees as the base models: We again use ROSE for resampling. To train the models, we use k-fold repeated CV method where the number of folds is taken to be 5, with 2 repetitions.
Hence for each base model, we get 10 Sensitivity, Specificity and ROC values. We find the summary of these, and they are plotted on the same graph as a means of comparison between the two base models.

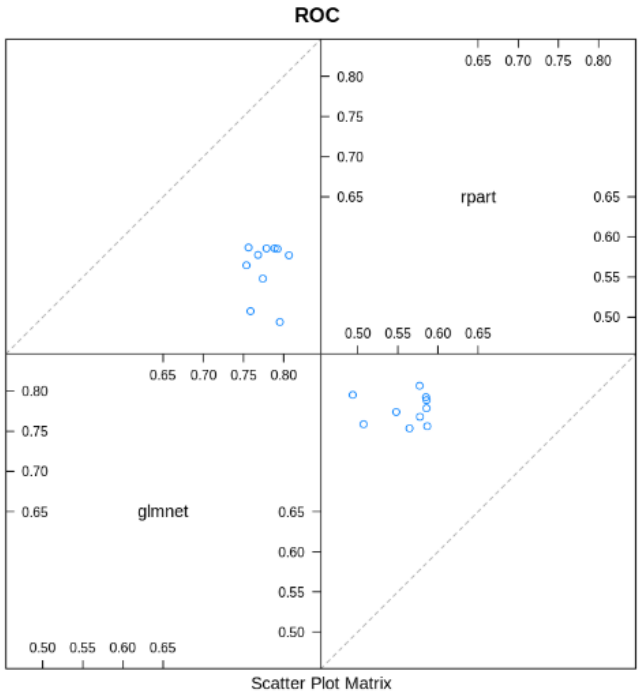
```
Call:
summary.resamples(object = stacking_results_dt)

Models: glmnet, rpart
Number of resamples: 10

ROC
            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
glmnet 0.7536863 0.7611171 0.7763895 0.7772973 0.7915596 0.8066852    0
rpart  0.4939117 0.5521495 0.5773461 0.5611830 0.5855280 0.5867016    0

Sens
              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
glmnet 0.661815068 0.6971318 0.7120223 0.7048423 0.7172517 0.7262618    0
rpart  0.001712329 0.9800942 0.9858791 0.7914370 0.9871575 0.9871575    0

Spec
             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
glmnet 0.65972222 0.6770833 0.7118056 0.7152778 0.7500000 0.7916667    0
rpart  0.09722222 0.1545139 0.1666667 0.3201389 0.1736111 1.0000000    0
```



Again we check for correlation between the models.

A matrix: 2 × 2 of type dbl

	glmnet	rpart
glmnet	1.00000000	0.03645198
rpart	0.03645198	1.00000000



Scatter Plot Matrix

The correlation between the models comes out as very low, implying we can go for stacking. Again we use both Logistic Regression and gbm as the meta models.

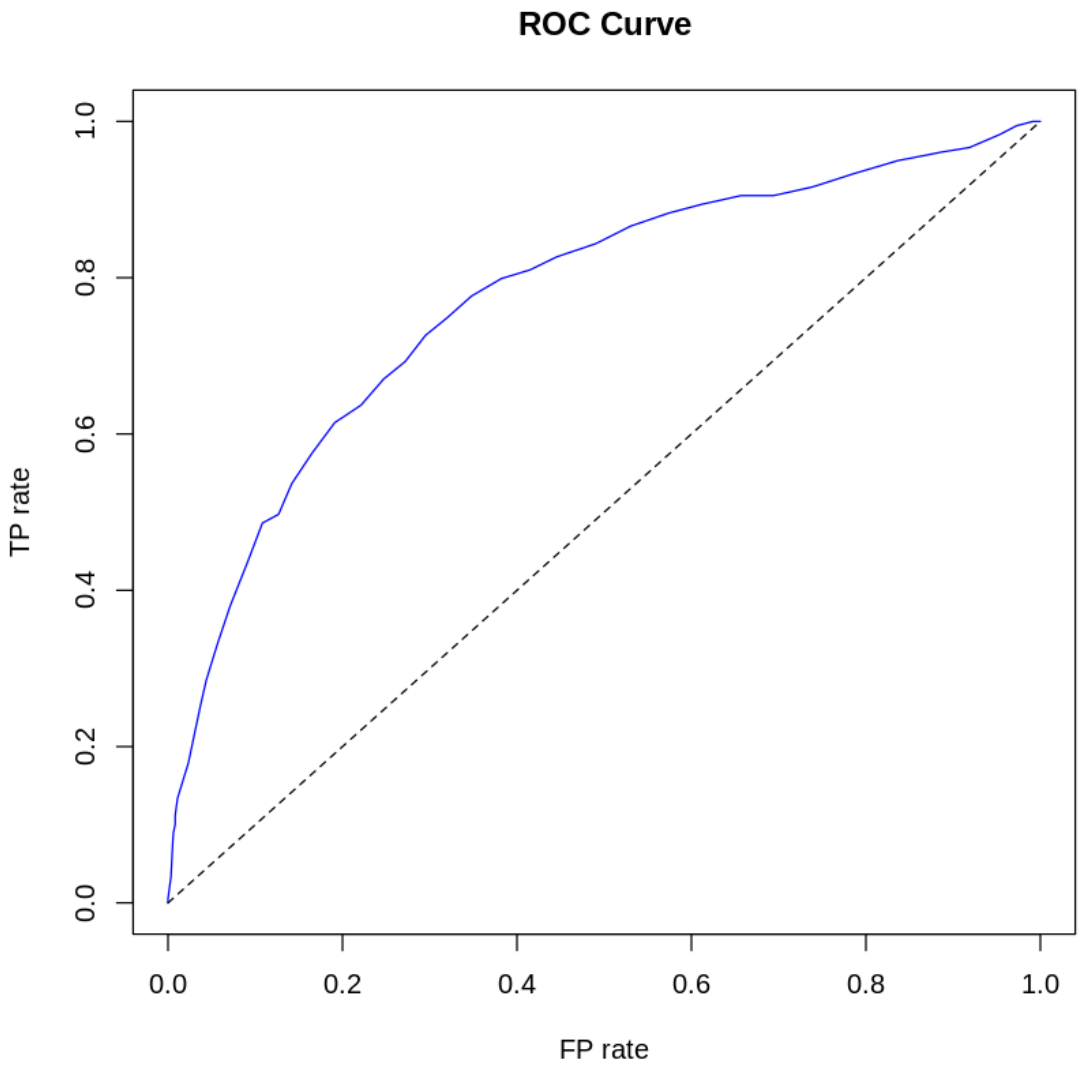
Using Logistic Regression:

```
A glm ensemble of 2 base models: glmnet, rpart

Ensemble results:
Generalized Linear Model

13122 samples
    2 predictor
    2 classes: 'NG', 'G'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 2 times)
Summary of sample sizes: 10497, 10498, 10498, 10497, 10498, 10498, ...
Addtional sampling using ROSE

Resampling results:

  ROC        Sens       Spec
  0.7760804  0.6986396  0.7204861
```

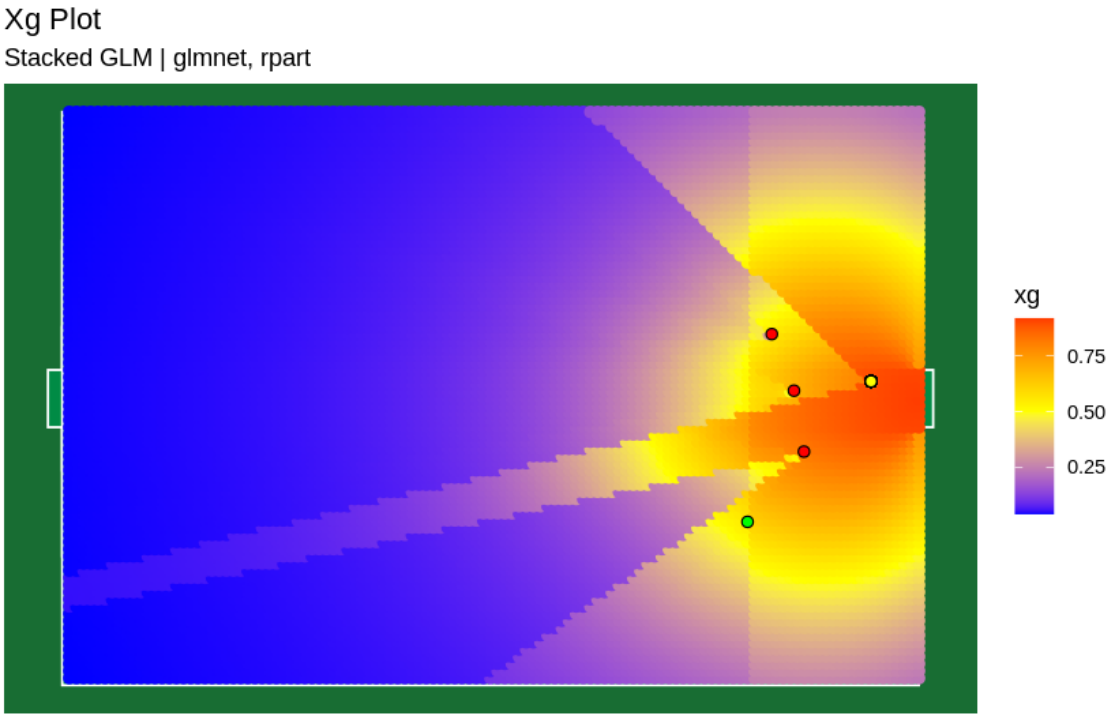
```
Confusion Matrix and Statistics

          Reference
Prediction   NG    G
        NG 1043   53
        G   417  126

               Accuracy : 0.7132
                 95% CI : (0.6907, 0.735)
    No Information Rate : 0.8908
    P-Value [Acc > NIR] : 1
```

```
                 Kappa : 0.2211

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.70391
            Specificity : 0.71438
         Pos Pred Value : 0.23204
         Neg Pred Value : 0.95164
             Prevalence : 0.10921
         Detection Rate : 0.07688
   Detection Prevalence : 0.33130
      Balanced Accuracy : 0.70915

       'Positive' Class : G
```

From the confusion matrix, we again see that we have high entries on the diagonals, implying the classification model is a good one. Also, we see that the model has classified the test set with an accuracy of 0.7132.
We will use other diagnostic measures like ROC AUC.



Again the classifier performs much better than a classifier that is just based on chance.
Heatmap:

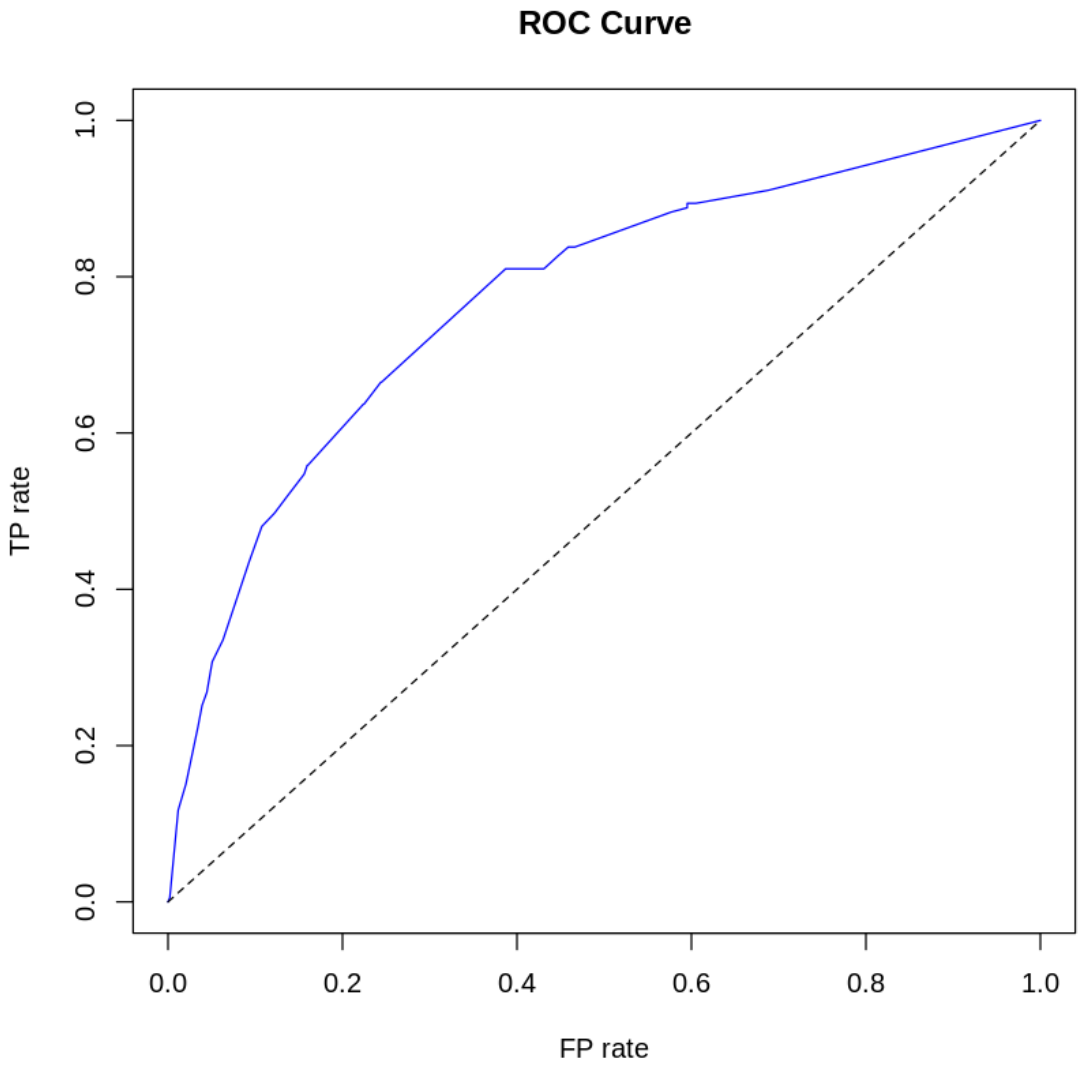


Using gbm:

Using gbm as the meta model,.

```
          Reference
Prediction   NG    G
        NG 1104   60
        G   356  119

               Accuracy : 0.7462
                 95% CI : (0.7244, 0.7671)
    No Information Rate : 0.8908
    P-Value [Acc > NIR] : 1

                  Kappa : 0.244

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.66480
            Specificity : 0.75616
         Pos Pred Value : 0.25053
         Neg Pred Value : 0.94845
             Prevalence : 0.10921
         Detection Rate : 0.07261
   Detection Prevalence : 0.28981
      Balanced Accuracy : 0.71048

       'Positive' Class : G
```

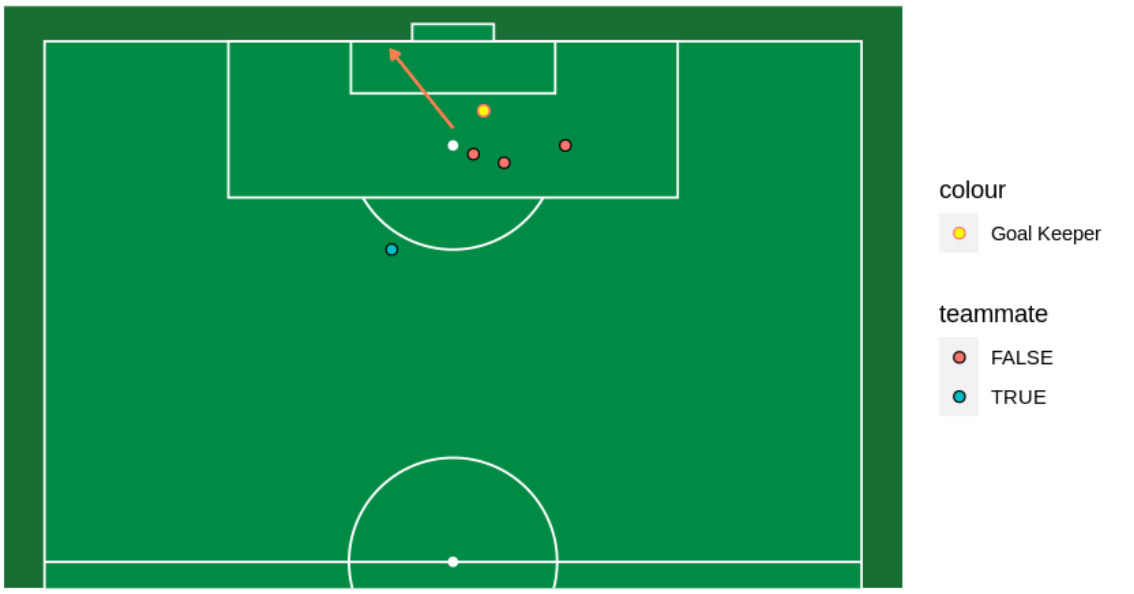
We get an accuracy of 0.7462. From the confusion matrix, we get high entries on the diagonals implying this is a good classifier.
We get the following ROC.



The ROC AUC is much higher than what we expect from a classifier purely based on chance.
We also plot the freeze frames of the shots with the highest and the lowest predicted XG's.

Freeze Frame

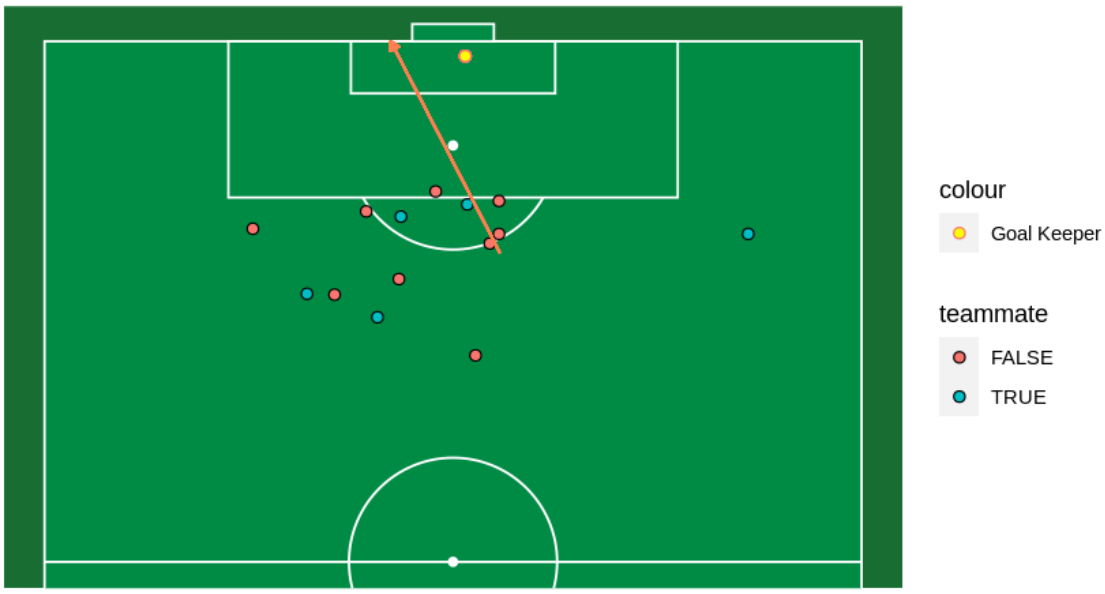
FA Women's Super League 2018/2019 : Birmingham City WFC vs West Ham United LFC (Timestamp : (



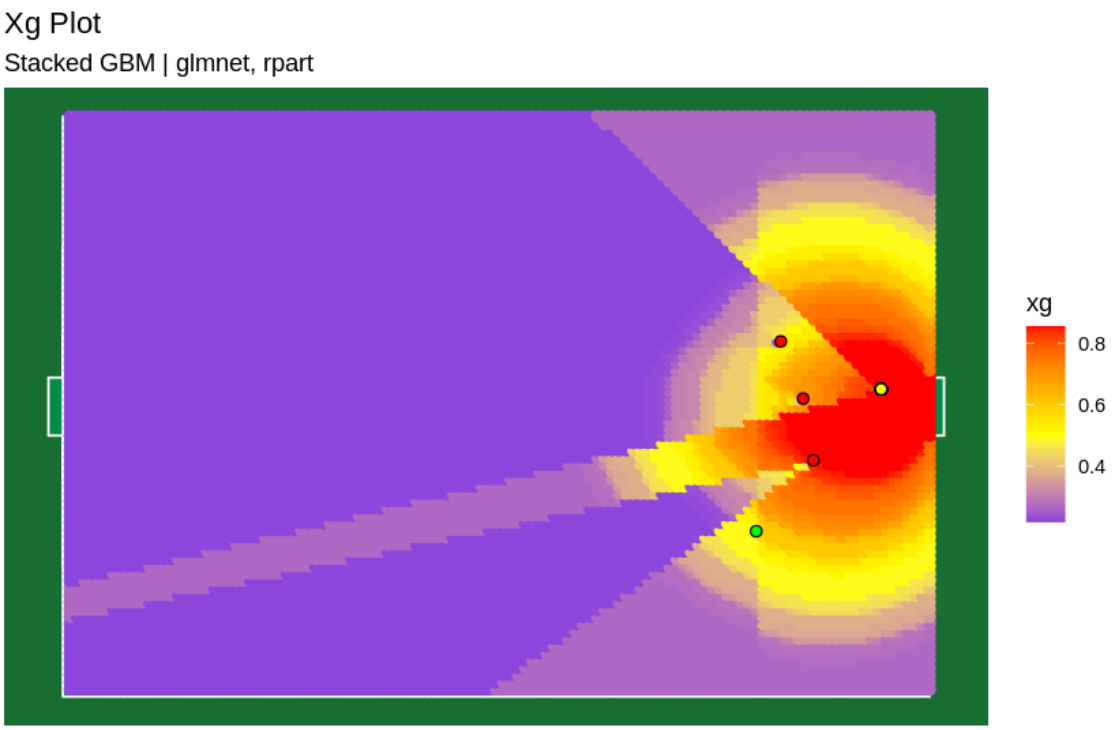
Result : Off T

Freeze Frame

FA Women's Super League 2020/2021 : Aston Villa vs Arsenal WFC (Timestamp : 00:10:06.492)



Result : Off T



Comparison of Stacked Models : We have observed upon considering *glm* as the meta model, the accuracies were 0.70 and 0.71 for knn and rpart base models respecitvely. While upon considering *gbm* as the meta model, the accuracies were 0.75 and 0.74. Hence, we can conclude that in our case a non linear meta model is required. However, in terms of accuracy there is no significant difference b/w k-nn and rpart base models.

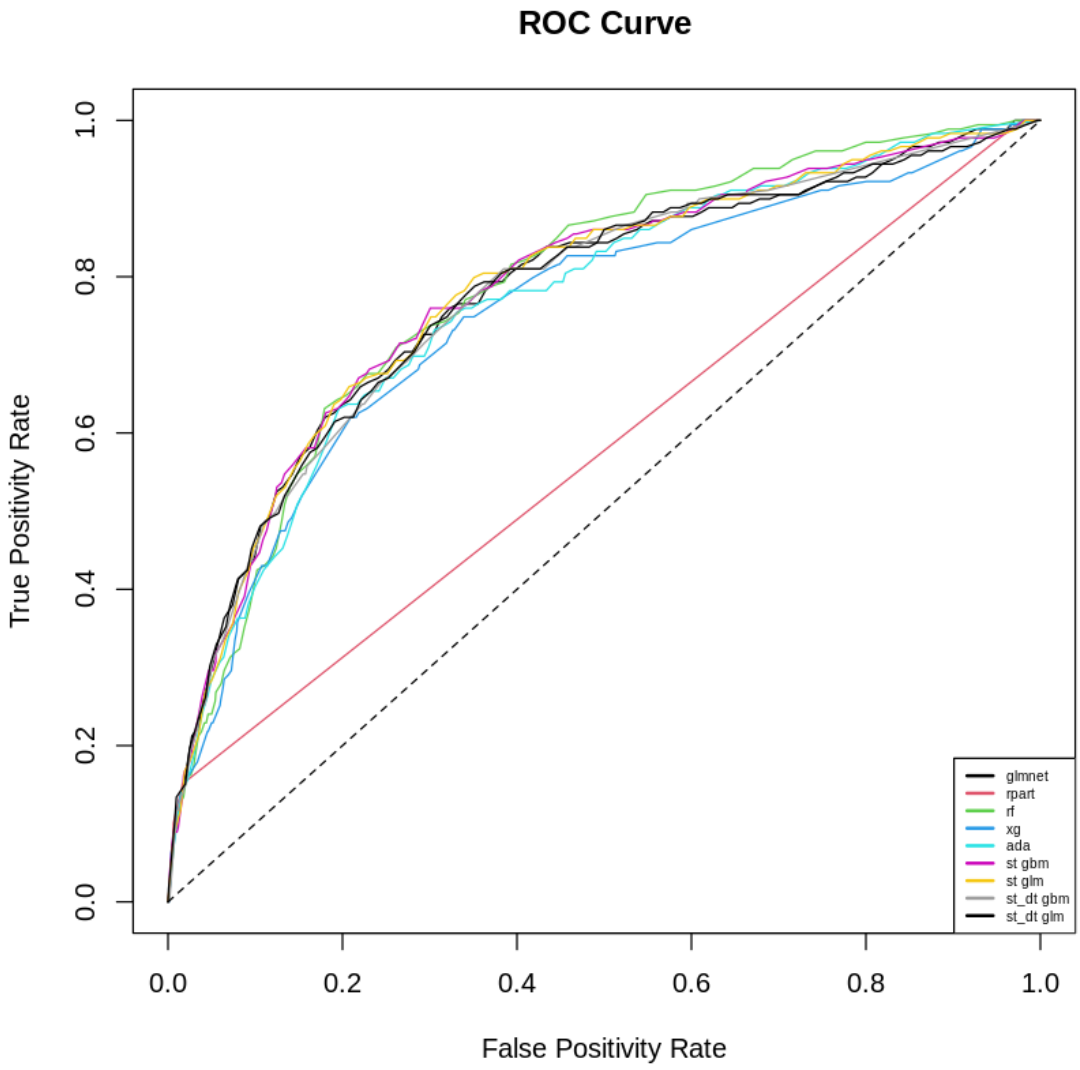
Comparison of models:

We have throughout used ROC as a measure to check for the effectiveness of a model.

Using ROC AUC

Now we will compare all the models we have used to find out which model provides the highest ROC AUC. Also the ROC of all the classification models are overlaid on the same plot to facilitate comparison.

```
ROC AUC of all models :
glmnet : 0.775216193464452
rpart : 0.576503788168669
rf : 0.784652177240377
xg : 0.750497436289891
ada : 0.765908395194
st gbm : 0.780919491849698
st glm : 0.78036274584832
st_dt gbm : 0.773102089232418
st_dt glm : 0.774192622637178

Maximum ROC AUC is given by model : rf
```



We see that the maximum ROC AUC is given by Random Forest. However stacking using Penalized Logistic Regression and kNN (using both Logistic Regression and gbm as meta models) gives comparable ROC AUC.

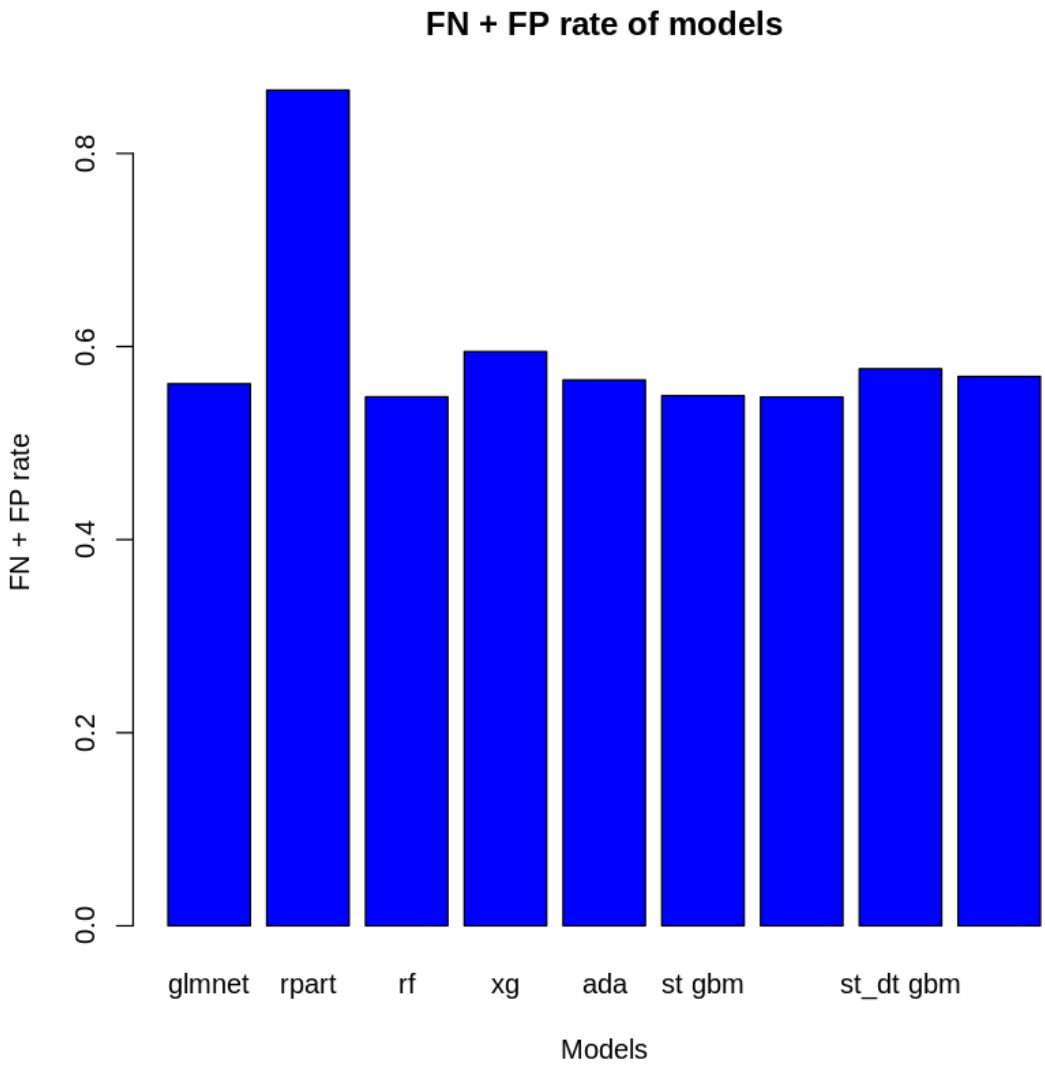
Using Accuracy Measure

Accuracy of all models :
glmnet : 0.687614399023795
rpart : 0.89261744966443
rf : 0.787065283709579
xg : 0.758389261744966
ada : 0.784624771201952
st gbm : 0.698596705308115
st glm : 0.782794386821232
st_dt gbm : 0.634533251982916
st_dt glm : 0.707138499084808

Using FN+FP rates

The model with the lowest FN+FP rates is the best model.

```
Lowest FN + FP rate of all models :
glmnet : 0.56145634039948
rpart : 0.865600367337568
rf : 0.547845718221474
xg : 0.594742481059157
ada : 0.565290426264636
st gbm : 0.549127573276192
st glm : 0.547738578097497
st_dt gbm : 0.576930435448075
st_dt glm : 0.568948496211831

Lowest FN + FP rate is given by model : st glm
```



The stacking model where we use Penalized Logistic Regression and kNN as the base models and Logistic Regression as the meta model, provides the lowest FN+FP rates. However, random forest and the stacking model where we use Penalized Logistic Regression and kNN as the base models and gbm as the meta model provide very comparable FN+FP rates.

Conclusions of Comparison

Stacked Models : - Using the ROC AUC metric, we notice in the stacked models, the models with **glmnet** and **k-nn** as base models consistently performed better.

- Using Accuracy metric, models with **gbm** as meta model, performed better.

Boosting vs Bagging : - Using all the metrics, here random forest performs marginally better than XgBoost and AdaBoost. But the difference is not very significant.

Ensembled vs non-Ensembled Models : - All the ensembled models perform significantly better than a single decision tree, but do perform similar to the glmnet model, in all measures we considered.

- A possible reason for similarity with glmnet model is presence of very significant variables that are almost linearly related to the Xg of shot. There are some non-linear relations in the data, but they have less impact than the linear relations.

Future Work:

xG model improvements:
1) Right now, our existing xG takes into consideration the location of off-ball players freeze frames at the time shots are taken. However, some of these features are discrete in nature. The number of defenders blocking the goal is an integer and can only increase/decrease in steps of 1. This problem can be overcome by replacing non-continuous features where we expect the relationship between that feature and goalscoring rates to be smooth. This applies to any features that rely on the location of players or are derived from them e.g. the open-goal feature. We achieve this by representing defenders as 2D bell-shaped surfaces (2D Gaussian distributions) instead of fixed-radius circles. As a result, we can now measure partial blocking if the blocker is near the shot-taker-goalposts triangle i.e. the blocker could still block the goal, but it's less likely than if the blocker were firmly in the shot-taker-goalposts triangle. A consequence of this is that the extent to which a player blocks the goal becomes a continuous number that gradually transitions from 1 to 0 as a blocker moves away from the shooter-goalface triangle.

The concept of xG is one of the oldest in football analytics. Despite that, there are still many improvements that can be made to these models to make them behave more intuitively, make them more resilient to small changes in the location of players, and unlock more insights into players' and teams' strengths and weaknesses in shooting situations.

Match score and league table position predictions:
2) Simulating the league table for a season by calculating the expected points for each team. We can do this by assuming a Poisson distribution for the number of goals scored by each team during a match. This may be improved by appropriately weighting the xG of shots taken during a match based on the current scoreline and the time at which the shot is taken during a match.

Acknowledgement and Bibliography

We have taken the help of the following references for our project work:
References: Umami, Gutama, Hatta, "Implementing the Expected Goal (xG) Model to Predict Scores in Soccer Matches", International Journal of Informatics and Information Systems Vol. 4, No. 1, March 2021, pp. 38-54
Corentin Herbinet, "Predicting football results using machine learning techniques"
Karan Bhowmick and Vivek Sarvaiya, (2021), "A comparative study of the different classification algorithms on football analytics", http://dx.doi.org/10.21474/IJAR01/13280 (http://dx.doi.org/10.21474/IJAR01/13280)
Konstantinos Apostolou and Christos Tjortjis (2019), Sports Analytics algorithms for performance prediction, 2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)

https://topepo.github.io/caret/model-training-and-tuning.html#model-training-and-parameter-tuning\ (https://topepo.github.io/caret/model-training-and-tuning.html#model-training-and-parameter-tuning\) https://glmnet.stanford.edu/articles/glmnet.html\ (https://glmnet.stanford.edu/articles/glmnet.html\) https://dzone.com/articles/build-custom-ensemble-models-using-caret-in-r (https://dzone.com/articles/build-custom-ensemble-models-using-caret-in-r)
https://topepo.github.io/caret/subsampling-for-class-imbalances.html (https://topepo.github.io/caret/subsampling-for-class-imbalances.html)
https://cran.r-project.org/web/packages/ROSE/ROSE.pdf (https://cran.r-project.org/web/packages/ROSE/ROSE.pdf)
https://hastie.su.domains/Papers/AdditiveLogisticRegression/alr.pdf (https://hastie.su.domains/Papers/AdditiveLogisticRegression/alr.pdf)
https://blog.paperspace.com/gradient-boosting-for-classification/ (https://blog.paperspace.com/gradient-boosting-for-classification/)
https://www.javatpoint.com/stacking-in-machine-learning# (https://www.javatpoint.com/stacking-in-machine-learning#):~:text=Stacking%20is%20one%20of%20the,new%20model%20with%20improved%20performance

Packages Used:
StatsBombR, SDMTools, caret, caretEnsemble, ggplot2, ggsoccer, gganimate, glmnet, ROSE, rpart, randomForest, caTools, xgboost, plyr, fastAdaboost

We would also like to thank Professor Deepayan Sarkar for his guidance without which this project would not have been a success.