Predicting Goals in the National Hockey League

Simon Bergford

Abstract

Analyzing team and player performance is crucial for any professional sports team. Professional hockey teams develop complex strategies to achieve a simple task - score more goals than their opponent. This paper analyzed shot data from the 2007-2008 to 2020-2021 NHL seasons. Several models, Gradient Boosting via XGBoost, Random Forest via Ranger, Logistic Regression, and LASSO-Penalized Logistic Regression, were used to classify shot attempts into goals and non-goals. Two sampling methods, Random Oversampling and Undersampling and Synthetic Minority Oversampling Technique (SMOTE), were examined with each model. The results show that XGBoost had the best performance of the models used, and the sampling methods improved the performance of Random Forest but not the other models.

Contents

List of Figures

List of Tables

1 Introduction

Evaluating player and team performance is crucial in any team sport. This is especially true in the National Hockey League (NHL), as the existence of a salary cap that teams cannot exceed makes every dollar spent on player salaries valuable. Common measures of player performance are goals, assists, and plus/minus, and team performance is measured with wins and losses, goal differential, shot differential, and scoring chances for and against. (A glossary of hockey terms can be found in Table 29 in Section 6.3.) These metrics work well, but there is interest from team management and fans in developing methods that work even better, as the amount of statistical information being collected by hockey leagues such as the NHL is increasing. One measure of both player performance and team performance is expected goals (xGoal), which uses many factors including shot location and shot type to evaluate the quality of each shot attempt. Several websites such as Moneypuck, Hockey-Reference, and Evolving Hockey have created their own expected goal models for player evaluation and comparison.

For goalies, a common measure of goalie performance is save percentage, which is the percent of shots on goal that a goalie prevents from scoring, not including shots that missed the net. In recent years, average save percentages in the NHL have been between 0.900 - 0.920, leading to an imbalance in the response classes, goals and non-goals. Random undersampling of the majority class, random oversampling of the minority class, or a combination of both are possible methods of addressing the imbalance. Synthetic Minority Oversampling Technique (SMOTE) is an alternative method that oversamples the minority class by generating new positive samples from the existing positive observations (Chawla *et al.*, 2002). SMOTE-NC is a variation that can handle nominal predictors, since the normal SMOTE can only handle continuous predictors.

SMOTE has been applied to other class imbalance situations. Meng *et al.* (2020) examined XGBoost's performance on credit fraud detection with three training datasets: the

original training dataset, a dataset that undersampled the majority class, and a dataset that added SMOTE observations based on the minority class. The Area under the Receiver Operating Characteristic curve (ROC) was highest with the SMOTE training dataset, and the undersampled dataset had a higher ROC than the original training dataset. Ahsan *et al.* (2018) examined SMOTE on phishing data using XGBoost, Random Forest, and SVM. Each model had improved accuracy when training on the SMOTE dataset compared to the original training data. Alghamdi *et al.* (2017) tested SMOTE and random undersampling with several models, Decision Trees, Naïve Bayes, Logistic Regression, Logistic Model Trees, and Random Forests, to predict diabetes from cardiorespiratory records. Each model had improved ROC using SMOTE, but saw no improvement using undersampling.

This paper examines several models for their ability to classify goals and determine shot quality. The effect of random oversampling, random undersampling, and the SMOTE-NC procedure on the classification models will also be studied.

2 Methods

2.1 Data

The dataset was initially downloaded from the NHL analytics site http://moneypuck.com on February 16, 2021, and then re-downloaded on July 22, 2021 following the conclusion of the 2020-2021 NHL season. Additional details about accessing the data can be found in Section 6.2. The dataset consists of 1,474,249 shot attempts in the NHL from the 2007-2008 season to the 2020-2021 season, which are classified as either saves, missed shots, or goals. Saves are shots that would have scored had the goalie not stopped the shot attempt, missed shots are shot attempts that missed the net or hit the post, and goals are shot attempts that scored. Blocked shots by either a teammate or opposing team player were not included in the data, since the location recorded for blocked shots is the block location not the shot attempt location. All data were recorded by off-ice NHL officials

during games.

2.1.1 Predictors in Dataset

A full list of predictors can be found in Table 30 in Section 6.4.

Shot location was recorded to the nearest foot and was normalized to each team's offensive and defensive zones. Each location consisted of an x location relative to the center ice red line with positive values falling in the shooting team's offensive half, and a y location relative to the middle of the ice. Shot type was also recorded and included the following types: wrist shot, snapshot, slapshot, backhand, deflection, tipped shot, and wrap around. Shot distance is the Euclidean distance from the shot coordinates to the center of the net, adjusted x = 89 and adjusted y = 0, and shot angle adjusted is the angle between the shot coordinates and a line going through the middle of the ice intersecting at the center of the net.

Figure 1 shows an example of shot locations based on the dimensions of an NHL hockey rink, 200 feet long by 85 feet wide.

Shots with a last event category not in block, faceoff, giveaway, hit, miss, shot, and takeaway were condensed into an other category as there were only 2,095 observations not in those categories. Distance from last event is the Euclidean distance from the last event to the current shot, and the time from last event is the time between those events. Speed is then calculated as last event distance divided by the time from last event. The distance and angle from the last event to the net were calculated for each shot. Shooting team goals and defending team goals were created by matching the shooting team, home or away, and home team goals and away team goals. Last event team was created in a similar manner by matching the shooting team and the team that did the last event; however, shots with an inaccurate last event team, not shooting or defending team, were combined into an other category, and there were 584 of those instances. The same procedure was used for shooting team and defending team penalty length and penalty time

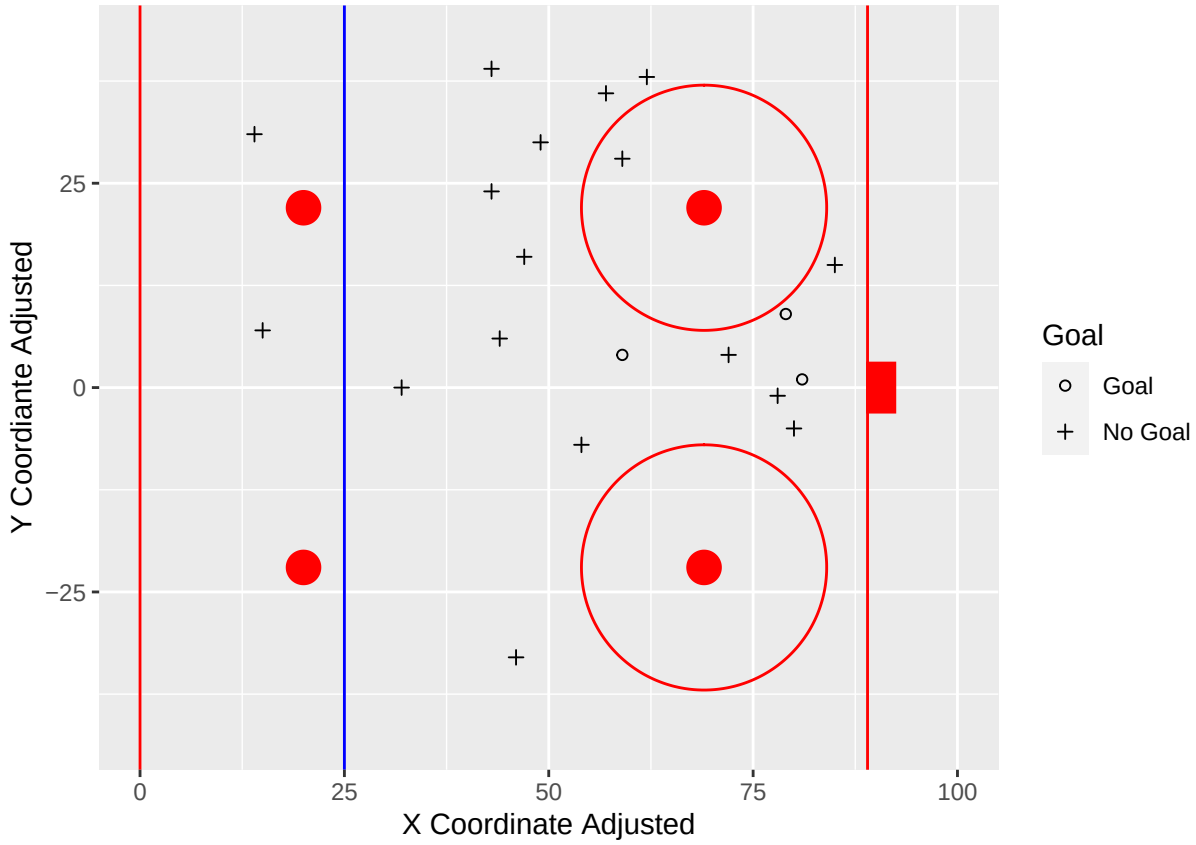

Figure 1: Coordinate Example

remaining.

Several calculated time-on-ice predictors were included for both teams: average time-on-ice for all positions, average time-on-ice by position, and maximum and minimum time-on-ice for all positions. Average rest difference was calculated as the average time-on-ice difference between the shooting team and the defending team. Total number of skaters for both the shooting team and defending team was calculated from the number of forwards and number of defensemen respectively.

2.2 Sampling Methods

Two sampling methods were examined: Random Oversampling/Random Undersampling and SMOTE.

2.2.1 Random Oversampling/Undersampling

Several oversampling and undersampling rates were examined together. Positive cases, goals, were randomly oversampled at 0%, 100%, 200%, and 300% times their actual observed values, and negative cases, saves or missed shots, were randomly undersampled at 25%, 50%, 75%, and 100% their observed values. For example, randomly oversampling goals at 100% would double the number of goals in the training set. Each original goal would be included, and goals are randomly sampled with replacement from the original set of goals. Undersampling at 50% would randomly sample half the negative cases, saves or missed shots, from the original training set. Each combination of oversampling and undersampling rates were assessed with each of the models.

2.2.2 SMOTE

The SMOTE observations were created following the procedure specified in Chawla *et al.* (2002) and listed in the SMOTE Algorithm. For each goal observation, the distance

to each other goal observation was calculated, equal to the squared difference of each continuous variable plus the square of the median standard deviation of the continuous variables for each different categorical variable. To create each synthetic observation, one of the k nearest neighbors was randomly selected and the distance vector between the neighbor and the original point was calculated. A synthetic point would be generated by multiplying the distance vector by a random value between zero and one, and then adding that vector to the current observation. The nominal features are determined by the mode of each variable. Each combination of one to five neighbors and oversampling of 100% to 300% were tested with each of the models.

The continuous SMOTE variables were x and y adjusted coordinate, time since the last faceoff, time since the last event, the adjusted x and y coordinates for the last event, the maximum and minimum time-on-ice for both shooting and defending teams, current penalty time left, the average time-on-ice of forwards and defensemen for both shooting and defending teams, period, and shooting and defending team goals. The categorical SMOTE variables were shot type, shooter handedness, last event category, the position of the player who did the last event, the team who did the last event, and the current penalty length for the shooting and defending teams. Shot distance, shot angle, distance from last event, speed from last event, total number of skaters for the shooting and defending team, average time-on-ice for all skaters for the shooting and defending team, whether the current shot is a rebound, the last event distance and angle to the net, and whether the shooter was on their offwing were calculated based on the SMOTE values.

2.2.2.1 SMOTE Algorithm

Algorithm SMOTE(T,N,K)

Input: Number of minority class samples T; Amount of SMOTE N%; Number of nearest neighbors k

Output: (N/100) $*$ T synthetic minority class samples

1. ($*$ If N is less than 100%, randomize the minority class samples as only a random

percent of them will be SMOTEd. ∗)

2. **if** N < 100

3. **then** Randomize the T minority class samples

4. T = (N/100) ∗ T

5. N = 100

6. **endif**

7. N = (int)(N/100)(∗ The amount of SMOTE is assumed to be in integral multiples of 100. ∗)

8. k = Number of nearest neighbors

9. numattrs = Number of attributes

10. Sample[ ][ ]: array for original minority class samples

11. newindex: keeps a count of number of synthetic samples generated, initialized to 0

12. Synthetic[ ][ ]: array for synthetic samples (∗ Compute k nearest neighbors for each minority class sample only. ∗)

13. **for** i ← 1 **to** T

14. Compute k nearest neighbors for i, and save the indices in the nnarray

15. Populate(N, i, nnarray)

16. **endfor**

17. Populate(N, i, nnarray) (∗ Function to generate the synthetic samples. ∗)

18. **while** N ≠ 0

19. Choose a random number between 1 and k, call it nn. This step chooses one of the k nearest neighbors of i.

20. **for** attr ← 1 **to** numattrs

21. Compute: dif = Sample[nnarray[nn]][attr] − Sample[i][attr]

22. Compute: gap = random number between 0 and 1

23. Synthetic[newindex][attr] = Sample[i][attr] + gap ∗ dif

24. **endfor**

25. newindex++
26. N = N − 1
27. **endwhile**
28. **return** (∗ End of Populate. ∗)

2.3 Models

Several models were tested for their ability to classify goals: Logistic Regression, LASSO penalized Logistic Regression, Random Forests via Ranger, and Gradient Boosting via XGBoost. R version 3.6.3 was used for data cleaning, creating the training and testing datasets, and computing the models (R Core Team, 2020).

2.3.1 Logistic Regression

Logistic Regression with a logit link was one of the models used. Logistic Regression uses a link function to transform a nonlinear relationship from binary data to a linear relationship that can be modeled linearly. The logit link function transforms the binary response to the log odds, whose domain is the entire real line. Further details can be found in Agresti (2003). The default glm function in R was used, and predictions were made using the default predict function (R Core Team, 2020).

2.3.2 LASSO Penalized Logistic Regression

LASSO penalized Logistic Regression, as introduced by Tibshirani (1996), was also used. LASSO penalized Logistic Regression computes the likelihood similarly to standard Logistic Regression but subtracts a penalized term from the likelihood. LASSO uses L_1 norm regularization penalty, $\lambda(\beta) = \lambda \sum_j |\beta_j|$, compared to ridge which uses the L_2 norm regularization penalty. This penalty can also be thought of as maximizing the likelihood subject to $\sum_j |\beta_j| \leq K$. LASSO has the added benefit of shrinking some coefficients

to zero, thereby performing variable selection, whereas ridge only shrinks the coefficients towards zero.

The glmnet package in R was used (Friedman *et al.*, 2010). Lambda values were determined with the cv.glmnet function and by raising 10 to a sequence from -1 to -6 separated by 0.25: 10^{-1}, $10^{-1.25}$, ... , 10^{-6}. Predictions were made using the predict function and the coefficients corresponding to the largest lambda value with error within one standard error of the lambda that produced the minimum cross validation error.

2.3.3 Random Forest

Random Forest is an alternative to normal decision tree bagging, introduced by Breiman (2001). Like in bagging, a bootstrap training sample is taken before growing each decision tree and then uses the CART algorithm to grow the trees until maximum size (Breiman *et al.*, 1984). Random Forest modifies this approach by restricting the number of predictors available at each node split to a random subset of predictors, with the number specified beforehand. This decorrelates the forest of decision trees since strong predictors would be chosen more often and lead to similar looking trees, and it tends to improve the generalization performance of the forest (James *et al.*, 2013). New observations, like other ensemble methods, are predicted by majority vote.

The Ranger package in R was used to create a probability forest, which uses the same splitting method as a classification forest but outputs probability estimates for each class instead of classifications (Wright and Ziegler, 2017), (Wright and Ziegler, 2015). Ranger uses Gini index as a splitting rule for classification and probability trees (Wright and Ziegler, 2015). Gini index is defined as $1-\sum_{i=0}^{c-1} p_i(t)^2$, where $p_i(t)$ is the proportion of observations in class i at node t and c is the number of classes. Variable importance is then determined by summing the total decrease in Gini for each node where a specific variable is used, weighted by the number of samples at each node (Louppe *et al.*, 2013). The default number of variables available at each split was used, $\sqrt{p} = \sqrt{80}$. The

number of trees per forest was set to the number of rows in the training dataset divided by 2,500. Predictions were made using the predict function.

2.3.4 XGBoost

Boosting is an iterative process that combines weak learners to approximate a function, $F^*(x)$, that maps x to y. Gradient boosting, as descibed in Friedman (2001), is an adaptation of boosting that uses the steepest descent of the negative gradient of a loss function to update the estimated function. The initial state is F_0, and subsequent states, $\{F_m\}_1^M$, are added. Negative binomial log-likelihood, $L(y, F) = \log(1+\exp(-2yF(x))$, is used as the loss function. A metric for variable importance in Gradient Boosting is Gain, which is the improvement in accuracy after a split has been added at a node. Gain is found by computing the change in score of the gradient statistics at each node, details can be found in Chen and Guestrin (2016)

The XGBoost package is an implementation of gradient boosting in R, (Chen *et al.*, 2021). The xgb.cv function was used to determine the number of boosting rounds for the specified training data, and the xgboost function was used to train the final model for each training instance. A max depth of 4 and a learning rate of 0.1 were used for each training situation. Predictions were made using the predict function.

2.4 Evaluation Metrics

An 80-20 training/testing split was used to create training and testing datasets. All methods were trained on the original training set, oversampled and undersampled training set, and SMOTE training set. Cross validation on the training set was used to identify the best parameters for the final XGBoost and LASSO models. Model performance was assessed on accuracy, Area under the Receiver Operating Characteristic curve (ROC), and Area under the Precision Recall Curve (PRC). ROC was calculated using the roc.area function in the Verification package, (Research Applications Laboratory, 2015), and PRC was

calculated using the pr.curve function in the PRROC package, (Keilwagen *et al.*, 2014). Details about ROC and PRC can be found in Tan *et al.* (2016).

Accuracy is the proportion of correctly classified test observations and ranges from 0 to 1. The balance of the response classes is an important factor when interpreting model performance using accuracy, as a model that only predicted the negative class in the current situation, no goal, would have an accuracy of 0.93408. Without context, this is a good score, but it doesn't reflect good model performance for the current problem.

ROC is the Area under the Receiver Operating Characteristic curve, which is the true positive rate and false positive rate of the classifier at different operating points. The true positive rate, also known as recall, is the number of true positives divided by the number of true positives and false negatives. The false positive rate is the number of false positives divided by the number of false positives and true negatives. These metrics are calculated at each threshold value. The true positive rate and false positive rate both decrease to 0 as the threshold is increased, and they both increase to 1 as the threshold is decreased. A perfect classifier would have a true positive rate of 1 and false positive rate of 0, and the ROC score would be 1. A null or a random classifier would have an ROC score of 0.5.

PRC is the Area under the Precision Recall Curve, which is the precision and recall of the classifier at different score thresholds. Precision is the number of true positives divided by the number of true positives and false positives. Precision decreases to the skew of the responses, percent of positives in the data, as the threshold is decreased since there will only be positive predictions. Recall decreases to 0 as the threshold increases since there are fewer true positives classified, but precision can have any value between 0 and 1. A perfect classifier would have a precision of 1 and recall of 1, and the PRC score would be 1. A null or a random classifier would have a PRC score equal to skew of the dataset, the number of positives over the total sample size.

There are advantages and disadvantages of using each of the measurements. Accuracy is simple to calculate and understand, but is sensitive to skew, as an imbalanced

response can lead to elevated accuracy results for classifiers that perform poorly or only slightly better than a simple classifier. In simulations performed by Saito and Rehmsmeier (2015), they found that PRC was able to demonstrate performance differences between balanced and imbalanced datasets unlike ROC. They concluded that PRC was the better performance metric for imbalanced responses. ROC and PRC summarize performance at various thresholds, but different models can perform better at different operating points, so unless one model is better than another at all operating points, ROC and PRC alone should be used with caution to judge model performance.

3 Results

3.1 Original

Table 1: Original Training Data Results

Model	Accuracy	ROC	PRC
Null	0.93408	0.50000	0.06592
XGBoost	0.94039	0.80783	0.35612
Random Forest	0.93895	0.79082	0.33429
Logistic Regression	0.93601	0.77796	0.26595
LASSO LR	0.93570	0.77779	0.26011

Table 1 shows each model's performance with the original training set. Each model had better accuracy than a null model, with XGboost performing the best, Random Forest slightly lower, and Logistic Regression and LASSO performing similarly. XGBoost also had the best ROC and PRC, with Random Forest having values slightly lower for both, and Logistic Regression and LASSO performing similarly.

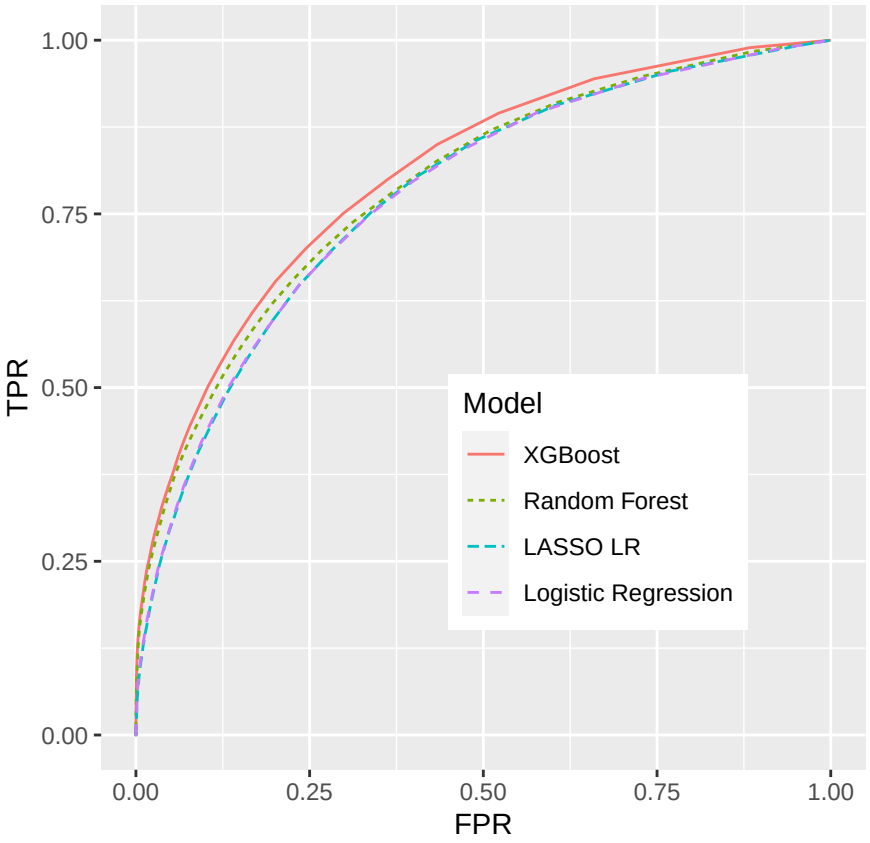

Figure 2: ROC Performance by Model

Figure 2 shows each model's ROC curve. XGBoost had the highest ROC, and performed better than all the other models at all thresholds. Random forest performed better than Logistic Regression and LASSO at higher thresholds, and Logistic Regression and LASSO performed similarly at all thresholds.

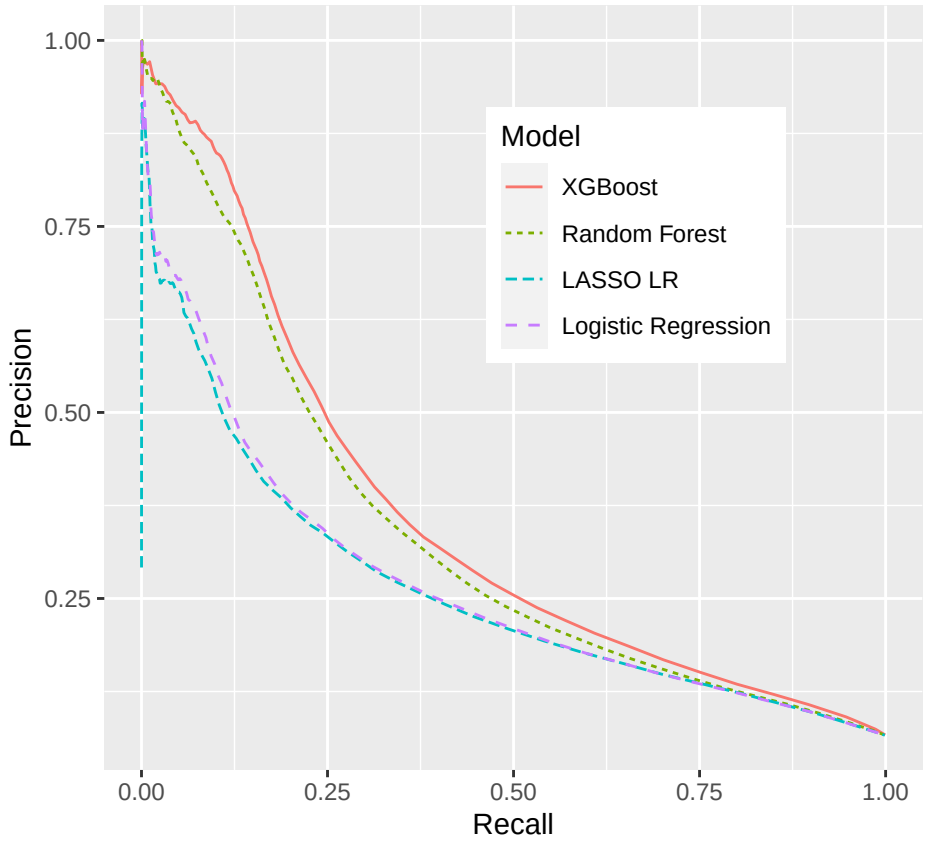

Figure 3: PRC Performance by Model

Figure 3 shows each model's PRC curve. XGBoost had the highest PRC, and performed better than all the other models at all thresholds. Random forest was slightly lower than XGBoost. Logistic Regression and LASSO performed similarly but worse than Random Forest.

Table 2: XGBoost Feature Importance, Top 10

Feature	Gain
shotDistance	0.31403
shotOnEmptyNet_0	0.09067
defendingTeamSkatersOnIce	0.08495
timeSinceLastEvent	0.08319
defendingPenalty1Length_0	0.07608
shotAngleAdjusted	0.04910
shootingTeamSkatersOnIce	0.04697
defendingPenalty1TimeLeft	0.03882
speedFromLastEvent	0.01652
shootingTeamForwardsOnIce	0.01587

Table 2 shows the relative Gain for the top 10 features in the XGBoost model. Shot distance was the most important feature, and was the only feature with a relative Gain above 0.1. 12 of the features had a relative gain above 0.01, which summed to 0.8398. 8 of the features had a relative gain above 0.03, which summed to 0.7838.

Table 3: Random Forest Feature Importance, Top 10

	Variable Importance
shotDistance	5789.99
speedFromLastEvent	4325.86
distanceFromLastEvent	3853.32
shotAngleAdjusted	3723.27
timeSinceFaceoff	3549.27

	Variable Importance
lastEventAngleFromNet	3529.68
lastEventDistFromNet	3478.32
xCordAdjusted	3474.53
averageRestDifference	3439.77
timeSinceLastEvent	3335.51

Table 3 shows the top 10 important features in the Random Forest model. Shot distance was also the most important feature using Random Forest. Several of the most important features in XGBoost are again present with Random Forest: time since the last event, adjusted shot angle, speed from the last event, adjusted y coordinate, adjusted x coordinate, time since the last faceoff, and distance from the last event.

Figure 4 shows predicted goal probability vs a shot's distance from the net and whether the shot is on an empty net. Shots closer to the net had a higher predicted goal probability than shots further from the net, and shots on an empty net were much more likely to be a goal.

Table 4: XGBoost Predicted Goal Probability by Shooting Team and Defending Team Skaters

Shooting/Defending Team Skaters	3	4	5	6
3	0.120	0.056	0.116	0.321
4	0.107	0.065	0.069	0.512
5	0.145	0.090	0.057	0.611
6	0.177	0.099	0.073	0.162

Table 4 shows predicted goal probability vs the number of skaters for the shooting

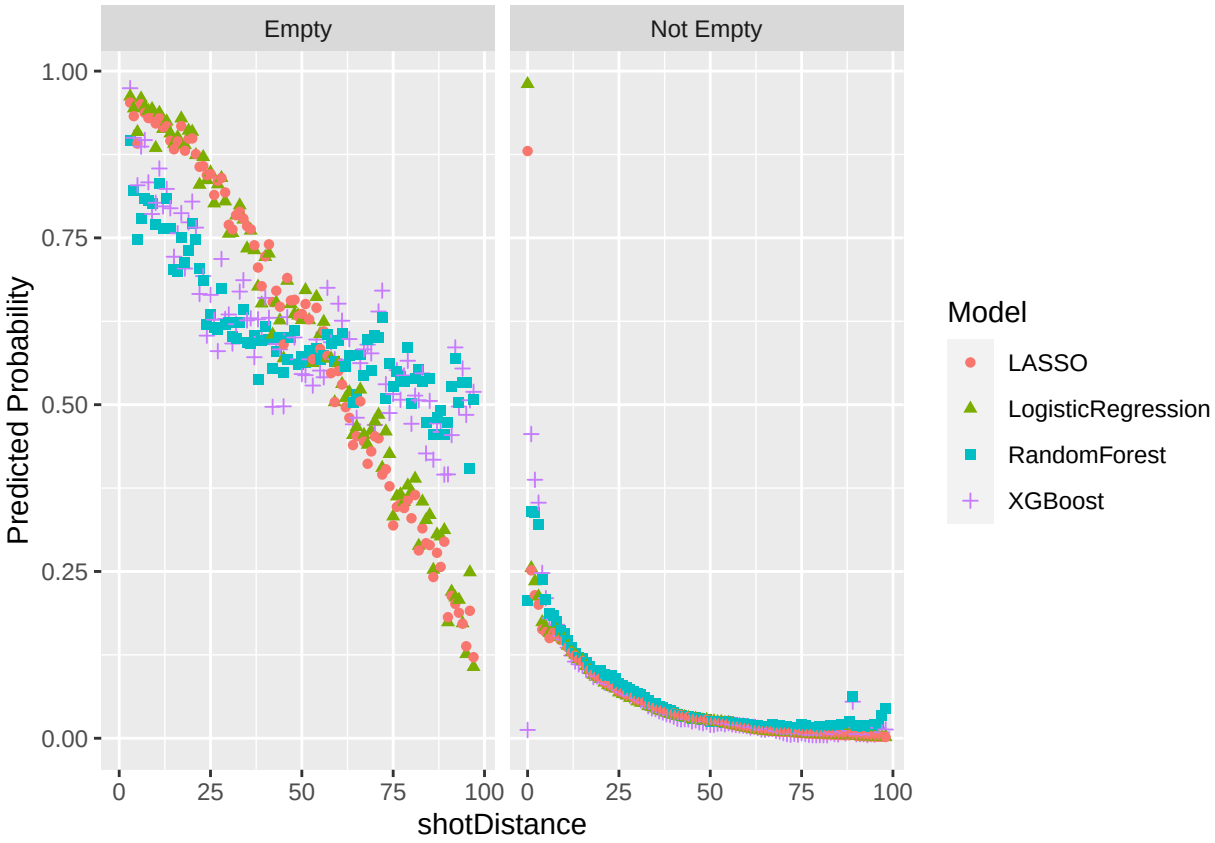

Figure 4: Predicted Goal Probability by Shot Distance and Empty Net

team and the defending team. Rows are the number of skaters for the shooting team, and columns are the number of skaters for the defending team. Shots had a higher predicted goal probability when teams had a powerplay - more skaters on the ice than the opposing team. The opposite effect can be seen when a defending team has six skaters on the ice, indicating they had pulled their goalie and had an empty net.

3.2 Over/Undersample

3.2.1 XGBoost

Tables 5, 6, and 7 show XGBoost's accuracy, ROC, and PRC performance using an oversampled and undersampled training set. Rows are the amount of oversampling, and columns are the amount of undersampling. The original training results are located at 0 oversampling and 1 undersampling.

3.2.1.1 XGBoost Oversample/Undersample Accuracy

Table 5: Over/Undersample XGBoost Accuracy

Over/Under Sample	0.25	0.5	0.75	1
0	0.92029	0.93903	0.94020	0.94039
1	0.86251	0.92083	0.93394	0.93808
2	0.80077	0.89594	0.92201	0.93196
3	0.74565	0.86784	0.90668	0.92278

Accuracy with XGBoost was lower for each of the oversampling and undersampling combinations.

3.2.1.2 XGBoost Oversample/Undersample ROC

Table 6: Over/Undersample XGBoost ROC

Over/Under Sample	0.25	0.5	0.75	1
0	0.80744	0.80779	0.80757	0.80783
1	0.79837	0.79899	0.79988	0.79927
2	0.79751	0.79825	0.79868	0.79917
3	0.79698	0.79848	0.79901	0.79895

ROC with XGBoost was lower than the original training data for all combinations of oversampling and undersampling.

3.2.1.3 XGBoost Oversample/Undersample PRC

Table 7: Over/Undersample XGBoost PRC

Over/Under Sample	0.25	0.5	0.75	1
0	0.35247	0.35610	0.35614	0.35612
1	0.34428	0.34784	0.34881	0.35029
2	0.34221	0.34595	0.34878	0.34964
3	0.34141	0.34677	0.34928	0.34881

PRC with XGBoost was lower for each level of oversampling and undersampling compared to the original training data except for no oversampling and 75% undersampling.

3.2.2 Random Forest

Tables 8, 9, and10 show Random Forest's accuracy, ROC, and PRC performance using an oversampled and undersampled training set. Rows are the amount of oversampling,

and columns are the amount of undersampling. The original training results are located at 0 oversampling and 1 undersampling.

3.2.2.1 Random Forest Oversample/Undersample Accuracy

Table 8: Over/Undersample Random Forest Accuracy

Over/Under Sample	0.25	0.5	0.75	1
0	0.92169	0.93898	0.93917	0.93895
1	0.90601	0.93626	0.93934	0.93946
2	0.89662	0.93361	0.93870	0.93942
3	0.89067	0.93232	0.93840	0.93935

Accuracy with Random Forest was higher than the original training data with no undersampling and all levels of oversampling, with undersampling of 75% and oversampling of 0% and 100%, and with no oversampling and 50% undersamapling. Accuracy with Random Forest was lower with the other oversampling and undersampling combinations.

3.2.2.2 Random Forest Oversample/Undersample ROC

Table 9: Over/Undersample Random Forest ROC

Over/Under Sample	0.25	0.5	0.75	1
0	0.79431	0.79252	0.79122	0.79082
1	0.79691	0.79515	0.79396	0.79267
2	0.79697	0.79649	0.79522	0.79462
3	0.79721	0.79752	0.79643	0.79519

ROC with Random Forest was higher for all combinations of oversampling and undersampling.

3.2.2.3 Random Forest Oversample/Undersample PRC

Table 10: Over/Undersample Random Forest PRC

Over/Under Sample	0.25	0.5	0.75	1
0	0.32123	0.33192	0.33340	0.33429
1	0.32076	0.33291	0.33568	0.33724
2	0.31745	0.33078	0.33582	0.33814
3	0.31822	0.33112	0.33566	0.33807

PRC with Random Forest was higher for no undersampling and each oversampling amount, and with undersampling of 75% and oversampling of 100%, 200%, and 300%. PRC with Random Forest was lower for the other combinations of undersampling and oversampling.

3.2.3 Logistic Regression

Tables 11, 12, and 13 show Logistic Regression's accuracy, ROC, and PRC performance using an oversampled and undersampled training set. Rows are the amount of oversampling, and columns are the amount of undersampling. The original training results are located at 0 oversampling and 1 undersampling.

3.2.3.1 Logistic Regression Oversample/Undersample Accuracy

Table 11: Over/Undersample Logistic Regerssion Accuracy

Over/Under Sample	0.25	0.5	0.75	1
0	0.91584	0.93415	0.93607	0.93601
1	0.84364	0.91637	0.92969	0.93422
2	0.74546	0.88751	0.91629	0.92649
3	0.66713	0.84317	0.89845	0.91650

Accuracy with Logistic Regression was lower than the original training data with all combinations of oversampling and undersampling except for undersampling of 75% and no oversampling.

3.2.3.2 Logistic Regression Oversample/Undersample ROC

Table 12: Over/Undersample Logistic Regerssion ROC

Over/Under Sample	0.25	0.5	0.75	1
0	0.77963	0.77883	0.77829	0.77796
1	0.77997	0.77966	0.77915	0.77888
2	0.78031	0.77986	0.77957	0.77928
3	0.78021	0.78006	0.77990	0.77960

ROC with Logistic Regression was higher than the original training data with all oversampling and undersampling combinations.

3.2.3.3 Logistic Regression Oversample/Undersample PRC

Table 13: Over/Undersample Logistic Regerssion PRC

Over/Under Sample	0.25	0.5	0.75	1
0	0.26033	0.26368	0.26511	0.26595
1	0.25790	0.26044	0.26258	0.26352
2	0.25553	0.25887	0.26081	0.26195
3	0.25417	0.25771	0.25931	0.26097

PRC with Logistic Regression was lower than the original training data with all oversampling and undersampling combinations.

3.2.4 LASSO

Tables 14, 15, and 16 show LASSO's accuracy, ROC, and PRC performance using an oversampled and undersampled training set. Rows are the amount of oversampling, and columns are the amount of undersampling. The original training results are located at 0 oversampling and 1 undersampling.

3.2.4.1 LASSO Oversample/Undersample Accuracy

Table 14: Over/Undersample LASSO Accuracy

Over/Under Sample	0.25	0.5	0.75	1
0	0.91702	0.93419	0.93588	0.93570
1	0.84484	0.91744	0.92997	0.93403
2	0.74724	0.88880	0.91736	0.92688
3	0.66852	0.84477	0.89948	0.91723

Accuracy with LASSO was lower than the original training data with all combinations of oversampling and undersampling except for undersampling of 75% and no oversampling.

3.2.4.2 LASSO Oversample/Undersample ROC

Table 15: Over/Undersample LASSO ROC

Over/Under Sample	0.25	0.5	0.75	1
0	0.77944	0.77885	0.77819	0.77779
1	0.77979	0.77949	0.77890	0.77861
2	0.78008	0.77971	0.77939	0.77928
3	0.77998	0.77989	0.77972	0.77962

ROC with LASSO was higher than the original training data with all oversampling and undersampling combinations.

3.2.4.3 LASSO Oversample/Undersample PRC

Table 16: Over/Undersample LASSO PRC

Over/Under Sample	0.25	0.5	0.75	1
0	0.25776	0.26163	0.26231	0.26011
1	0.25579	0.25779	0.25909	0.25956
2	0.25368	0.25679	0.25801	0.26029
3	0.25212	0.25564	0.25703	0.25939

PRC with LASSO was lower than the original training data with all oversampling and undersampling combinations except for no oversampling and undersampling of 50% and 75% and 200% oversampling and no undersampling.

3.3 SMOTE

3.3.1 XGBoost

Tables 17, 18, and 19 show XGBoost's accuracy, ROC, and PRC performance using a SMOTE training set. Rows are the amount of SMOTE, and columns are the number of neighbors. A SMOTE % of 1 creates synthetic observations equal to the number of goals in the original training data.

3.3.1.1 XGBoost SMOTE Accuracy

```
## [1] "Original XGBoost Accuracy: 0.94039"
```

Table 17: SMOTE XGBoost Accuracy

SMOTE %/Neighbors	1	2	3	4	5
1	0.94035	0.94035	0.94027	0.94038	0.94030
2	0.94043	0.94036	0.94033	0.94035	0.94034
3	0.94051	0.94036	0.94052	0.94044	0.94038

Accuracy with XGBoost was lower than the original training data for all combinations of SMOTE oversampling and number of neighbors except for 200% SMOTE oversampling and 1 nearest neighbor and with 300% SMOTE oversampling and 1, 3, and 4 nearest neighbors.

3.3.1.2 XGBoost SMOTE ROC

```
## [1] "Original XGBoost ROC: 0.80783"
```

Table 18: SMOTE XGBoost ROC

SMOTE %/Neighbors	1	2	3	4	5
1	0.80581	0.80558	0.80576	0.80598	0.80569
2	0.80495	0.80567	0.80579	0.80523	0.80478
3	0.80473	0.80522	0.80501	0.80504	0.80444

ROC with XGBoost was lower than the original training data for all combinations of SMOTE oversampling and number of neighbors.

3.3.1.3 XGBoost SMOTE PRC

```
## [1] "Original XGBoost PRC: 0.35612"
```

Table 19: SMOTE XGBoost PRC

SMOTE %/Neighbors	1	2	3	4	5
1	0.35576	0.35528	0.35527	0.35587	0.35560
2	0.35563	0.35559	0.35588	0.35542	0.35445
3	0.35565	0.35529	0.35568	0.35596	0.35415

PRC with XGBoost was lower than the original training dataset with all combinations of SMOTE oversampling and all number of neighbors.

3.3.2 Random Forest

Tables 20, 21, and 22 show Random Forest's accuracy, ROC, and PRC performance using a SMOTE training set. Rows are the amount of SMOTE, and columns are the number of neighbors.

3.3.2.1 Random Forest SMOTE Accuracy

```
## [1] "Original Random Forest Accuracy: 0.93895"
```

Table 20: SMOTE Random Forest Accuracy

SMOTE %/Neighbors	1	2	3	4	5
1	0.93935	0.93934	0.93926	0.93904	0.93898
2	0.93888	0.93870	0.93811	0.93789	0.93746
3	0.93815	0.93748	0.93628	0.93574	0.93475

Accuracy with Random Forest was higher than the original training dataset for 100% SMOTE oversampling and all number of neighbors. Accuracy was lower for all other combinations.

3.3.2.2 Random Forest SMOTE ROC

```
## [1] "Original Random Forest ROC: 0.79082"
```

Table 21: SMOTE Random Forest ROC

SMOTE %/Neighbors	1	2	3	4	5
1	0.79460	0.79504	0.79430	0.79353	0.79317
2	0.79443	0.79405	0.79269	0.79212	0.79065
3	0.79398	0.79416	0.79111	0.79032	0.78881

ROC with Random Forest was higher than the original training data for all combinations of SMOTE oversampling and number of neighbors, except for 200% SMOTE oversampling and 5 neighbors and 300% SMOTE oversampling and 4 and 5 neighbors.

3.3.2.3 Random Forest SMOTE PRC

```
## [1] "Original Random Forest PRC: 0.33429"
```

Table 22: SMOTE Random Forest PRC

SMOTE %/Neighbors	1	2	3	4	5
1	0.33495	0.33511	0.33283	0.33045	0.32866
2	0.33060	0.33030	0.32447	0.32162	0.31757
3	0.32657	0.32693	0.31830	0.31407	0.30921

PRC with Random Forest was higher than the original training data for 100% SMOTE oversampling and 1 and 2 neighbors, but was lower for all other combinations of SMOTE oversampling and number of neighbors.

3.3.3 Logistic Regression

Tables 23, 24, and 25 show Logistic Regression's accuracy, ROC, and PRC performance using a SMOTE training set. Rows are the amount of SMOTE, and columns are the number of neighbors.

3.3.3.1 Logistic Regression SMOTE Accuracy

```
## [1] "Original Logistic Regression Accuracy: 0.93601"
```

Table 23: SMOTE Logistic Regression Accuracy

SMOTE %/Neighbors	1	2	3	4	5
1	0.93356	0.93357	0.93297	0.93266	0.93240

SMOTE %/Neighbors	1	2	3	4	5
2	0.92545	0.92558	0.92403	0.92284	0.92203
3	0.91304	0.91300	0.90997	0.90817	0.90681

Accuracy with Logistic Regression was lower than the original training data for all combinations of SMOTE oversampling and number of neighbors.

3.3.3.2 Logistic Regression SMOTE ROC

```
## [1] "Original Logistic Regression ROC: 0.77796"
```

Table 24: SMOTE Logistic Regression ROC

SMOTE %/Neighbors	1	2	3	4	5
1	0.77629	0.77623	0.77252	0.77098	0.76903
2	0.77452	0.77425	0.76742	0.76474	0.76119
3	0.77341	0.77293	0.76389	0.76050	0.75579

ROC with Logistic Regression was lower than the original training data for all combinations of SMOTE oversampling and number of neighbors.

3.3.3.3 Logistic Regression SMOTE PRC

```
## [1] "Original Logistic Regression PRC: 0.26595"
```

Table 25: SMOTE Logistic Regression PRC

SMOTE %/Neighbors	1	2	3	4	5
1	0.25955	0.25841	0.25439	0.25206	0.24976
2	0.25449	0.25298	0.24574	0.24171	0.23834
3	0.25104	0.24938	0.24009	0.23534	0.23142

PRC with Logistic Regression was lower than the original training data for all combinations of SMOTE oversampling and number of neighbors.

3.3.4 LASSO

Tables 26, 27, and 28 show LASSO's accuracy, ROC, andPRC performance using a SMOTE training set. Rows are the amount of SMOTE, and columns are the number of neighbors.

3.3.4.1 LASSO SMOTE Accuracy

```
## [1] "Original LASSO Accuracy: 0.9357"
```

Table 26: SMOTE LASSO Accuracy

SMOTE %/Neighbors	1	2	3	4	5
1	0.93230	0.93237	0.93205	0.93161	0.93165
2	0.92335	0.92313	0.92245	0.92211	0.92165
3	0.90925	0.90892	0.90746	0.90628	0.90621

Accuracy with LASSO was lower than the original training data for all combinations of SMOTE oversampling and number of neighbors.

3.3.4.2 LASSO SMOTE ROC

```
## [1] "Original LASSO ROC: 0.77779"
```

Table 27: SMOTE LASSO ROC

SMOTE %/Neighbors	1	2	3	4	5
1	0.76399	0.76391	0.75980	0.75711	0.75525
2	0.76218	0.76113	0.75419	0.75159	0.74754
3	0.76081	0.76025	0.75098	0.74657	0.74206

ROC with LASSO was lower than the original training data for all combinations of SMOTE oversampling and number of neighbors.

3.3.4.3 LASSO SMOTE PRC

```
## [1] "Original LASSO PRC: 0.26011"
```

Table 28: SMOTE LASSO PRC

SMOTE %/Neighbors	1	2	3	4	5
1	0.24416	0.24300	0.24080	0.23776	0.23818
2	0.24025	0.23951	0.23463	0.23145	0.22844
3	0.23729	0.23654	0.22942	0.22642	0.22300

PRC with LASSO was lower than the original training data for all combinations of SMOTE oversampling and number of neighbors.

4 Conclusion

4.1 Discussion

The main goal of this project was to determine a good model for classifying goals and non-goals. Due to the imbalance in the response, two sampling methods, random oversampling and undersampling and SMOTE, were examined to determine if they could improve prediction performance.

For the original training data, XGBoost and Random Forest performed better than Logistic Regression and LASSO across all the evaluation metrics. Oversampling and undersampling worked best with Random Forest, as its ROC and PRC performance improved. XGBoost's performance with oversampling and undersampling was similar or decreased compared to the original training data for all of the metrics. Logistic Regression and LASSO's PRC accuracy and PRC with oversampling and undersampling were similar or lower compared to their results with the original training data, but their ROC performance was similar or better than the original training data. SMOTE decreased model performance for all models and all evaluation metrics, except for Random Forest with certain combinations of SMOTE oversampling and number of neighbors and XGBoost's accuracy with certain combinations of SMOTE oversamping and number of neighbors.

Several of the features in the top 75% of relative gain in the XGBoost model are features that Moneypuck uses in their gradient boosted expected goals model. Details about Moneypucks's model can be found in the **Shot Prediction Expected Goals Model** section at http://moneypuck.com/about.htm. The shared features are shot distance, time since the last event, whether a shot was on an empty net, shot angle, and man advantage (shooting/defending skaters on ice). Speed from the last event and x coordinate adjusted are also used in Moneypuck's model, and they have a relative gain above 0.01 in the XGBoost model. The XGBoost model had similar results to Moneypuck's model, as Moneypuck's model had an ROC score of 0.8108 compared to XGBoost's 0.8078, and a PRC score of

0.3392 compared to XGBoost's 0.3561.

The results found for SMOTE and oversampling and undersampling differ from what was found in the literature referenced in the introduction. Random Forest saw an improvement in accuracy for certain combinations of SMOTE oversampling and number of neighbors, similar to the results found in Ahsan *et al.* (2018) and Alghamdi *et al.* (2017). Random Forest also saw improvements in PRC scores for several combinations of oversampling and undersampling, which were not found in Alghamdi *et al.* (2017). XGBoost did not see an improvement in ROC for both oversampling and undersampling and SMOTE, unlike the improvements seen in Meng *et al.* (2020) and Ahsan *et al.* (2018). Logistic Regression and LASSO also did not see any improvement in ROC or PRC for SMOTE, unlike with Alghamdi *et al.* (2017). Logistic Regression and LASSO did have improvements in ROC with oversampling and undersampling, but not with PRC. These discrepancies point to the importance of using the correct evaluation metric with responses that are imbalanced, pointed out in Saito and Rehmsmeier (2015). The different results found also make sense for logistic regression and LASSO penalized logistic regression; since they are likelihood-based models, changing the training data via SMOTE or oversampling and undersampling should lead to worse performance.

4.2 Limitations

There are several limitations of data used in this project, which are generally due to inaccuracies in data recording. After examining the data, out of 1.4 million shot attempts, 228 had fewer skaters than the minimum required of 3 or more skaters than the maximum of 6, indicating there were mistakes made by the NHL officials recording the game data. There were also instances when a team had a penalty but the number of players did not match up with the expected number of skaters on the ice. Other examples include missing shot type and missing shooter handedness, or having an inaccurate last event category or last event team. Accurate shot location is also a question, and there have been some attempts

to adjust for arena bias by comparing the distribution of shots at each arena, (Schuckers and Curro, 2013). In addition, 3,369 were likely missing shot location and were assigned 0 x coordinate and 0 y coordinate. Event category can also be a subjective determination, such as the difference between a giveaway and a takeaway, which both result in turning the puck over, also noted by Schuckers and Curro (2013).

There are other variables that are being recorded but are currently unavailable to the public. For example, the NHL started using sensors in pucks and on players' jerseys at the beginning of the 2020-2021 NHL season. Metrics such as shot speed, player location, and player speed are among those being recorded. Access to these metrics would likely improve model performance, and the sensors would improve data accuracy, but the data are currently only used by NHL front offices and broadcasters.

Future research could improve on the models in this paper. Using the currently inaccessible data listed above would likely improve most models, but other predictors such as historical metrics of shooter or goalie quality might also help with model performance. Methods similar to what Schuckers and Curro (2013) proposed could be developed to help with any data inaccuracies regarding event location and event type. SMOTE's effectiveness can also be examined in regards to a dataset's class imbalance, correlation between continuous predictors, and the proportion of data types for SMOTE-NC.

5 Bibliography

Agresti, A. (2003) *Categorical Data Analysis*. John Wiley & Sons.

Ahsan, M., Gomes, R. and Denton, A. (2018) Smote implementation on phishing data to enhance cybersecurity. In: *2018 IEEE international conference on electro/information technology (EIT)*, 2018, pp. 0531–0536. IEEE.

Alghamdi, M., Al-Mallah, M., Keteyian, S., et al. (2017) Predicting diabetes mellitus using SMOTE and ensemble machine learning approach: The henry ford ExercIse testing

(FIT) project. *PloS one*, **12**, e0179805. Public Library of Science San Francisco, CA USA.

Breiman, L. (2001) Random forests. *Machine learning*, **45**, 5–32. Springer.

Breiman, L., Friedman, J., Stone, C. J., et al. (1984) *Classification and Regression Trees*. CRC press.

Chapelle, O., Shivaswamy, P., Vadrevu, S., et al. (2011) Boosted multi-task learning. *Machine learning*, **85**, 149–173. Springer.

Chawla, N. V., Bowyer, K. W., Hall, L. O., et al. (2002) SMOTE: Synthetic minority over-sampling technique. *Journal of artificial intelligence research*, **16**, 321–357.

Chen, T. and Guestrin, C. (2016) Xgboost: A scalable tree boosting system. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

Chen, T., He, T., Benesty, M., et al. (2021) *Xgboost: Extreme Gradient Boosting*. Available at: https://CRAN.R-project.org/package=xgboost.

Corporation, M. and Weston, S. (2020) *doParallel: Foreach Parallel Adaptor for the 'Parallel' Package*. Available at: https://CRAN.R-project.org/package=doParallel.

Dowle, M. and Srinivasan, A. (2021) *Data.table: Extension of 'Data.frame'*. Available at: https://CRAN.R-project.org/package=data.table.

Friedman, J., Hastie, T. and Tibshirani, R. (2010) Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, **33**, 1–22. Available at: https://www.jstatsoft.org/v33/i01/.

Friedman, J. H. (2001) Greedy function approximation: A gradient boosting machine. *Annals of statistics*, 1189–1232. JSTOR.

Friedmann, J., Hastie, T. and Tibshirani, R. (2000) Additive logistic regression: A statistical view of boosting. *Ann. Statist*, **28**, 337–407.

Gorman, B. (2018) *Mltools: Machine Learning Tools*. Available at: https://CRAN.R-project.org/package=mltools.

Grau, J., Grosse, I. and Keilwagen, J. (2015) PRROC: Computing and visualizing precision-recall and receiver operating characteristic curves in r. *Bioinformatics*, **31**, 2595–2597.

James, G., Witten, D., Hastie, T., et al. (2013) *An Introduction to Statistical Learning*. Springer.

Keilwagen, J., Grosse, I. and Grau, J. (2014) Area under precision-recall curves for weighted and unweighted data. *PLOS ONE*, **9**.

Louppe, G., Wehenkel, L., Sutera, A., et al. (2013) Understanding variable importances in forests of randomized trees. *Advances in neural information processing systems 26*.

Mahmud, S. H., Chen, W., Jahan, H., et al. (2019) iDTi-CSsmoteB: Identification of drug–target interaction based on drug chemical structure and protein sequence using XGBoost with over-sampling technique SMOTE. *IEEE Access*, **7**, 48699–48714. IEEE.

Meng, C., Zhou, L. and Liu, B. (2020) A case study in credit fraud detection with SMOTE and XGBoost. In: *Journal of physics: Conference series*, 2020, p. 052016. 5. IOP Publishing.

Microsoft and Weston, S. (2020) *Foreach: Provides Foreach Looping Construct*. Available at: https://CRAN.R-project.org/package=foreach.

Moneypuck.com -download datasets (2021) http://moneypuck.com/data.htm.

Mukherjee, M. and Khushi, M. (2021) SMOTE-ENC: A novel SMOTE-based method to generate synthetic data for nominal and continuous features. *Applied System Innovation*, **4**, 18. Multidisciplinary Digital Publishing Institute.

R Core Team (2020) *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. Available at: https://www.R-project.org/.

Razali, R. M., Chui, P. L., Hong, C. L., et al. (2015) Harmful microalgae assemblage in the aquaculture area of aman island, northen strait of malacca. *Malaysian Journal of Science*, **34**, 20–32.

Research Applications Laboratory, N. - (2015) *Verification: Weather Forecast Verification Utilities*. Available at: https://CRAN.R-project.org/package=verification.

Saito, T. and Rehmsmeier, M. (2015) The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, **10**, e0118432. Public Library of Science.

Schuckers, M. and Curro, J. (2013) Total hockey rating (THoR): A comprehensive statistical rating of national hockey league forwards and defensemen based upon all on-ice events. In: *7th annual MIT sloan sports analytics conference*, 2013.

Tan, P.-N., Steinbach, M. and Kumar, V. (2016) *Introduction to Data Mining*. Pearson Education India.

Tibshirani, R. (1996) Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, **58**, 267–288. Wiley Online Library.

Wickham, H. (2016) *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. Available at: https://ggplot2.tidyverse.org.

Wickham, H., Averick, M., Bryan, J., et al. (2019) Welcome to the tidyverse. *Journal of Open Source Software*, **4**, 1686. DOI: 10.21105/joss.01686.

Wright, M. N. and Ziegler, A. (2015) Ranger: A fast implementation of random forests for high dimensional data in c++ and r. *arXiv preprint arXiv:1508.04409*.

Wright, M. N. and Ziegler, A. (2017) ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, **77**, 1–17. DOI: 10.18637/jss.v077.i01.

6 Appendix

6.1 Reproduction

The data and code used for the analysis, as well as the resources for the paper can be found at: https://github.com/SimonBergford/UMN-PLAN-B.

6.2 Downloading Moneypuck Data

The data used in this analysis be downloaded from the GitHub repository listed above. Updated data from Moneypuck can be downloaded by going to http://moneypuck.com/data.htm and scrolling to the **Download Shot Data** section. All past seasons, 2007-2008 to 2019-2020, or recent seasons, 2013-2014 to 2019-2020, can be downloaded together. Individual seasons can also be downloaded. The current season, 2020-2021 can be downloaded in the same manner, and will be updated until the conclusion of the 2020-2021 season. The data will be downloaded in a zipped file.

6.3 Glossary of Hockey Terms

Table 29: Hockey Glossary

Term	Definition
Assist	Up to the last two players to touch the puck for the offensive team before the goal scorer. Resets when an opposing player gains possession of the puck.
Backhand	A shot using the convex side of the blade. Similar to both a wrist shot and snap shot but usually slower.
Blocked Shot	A shot attempt blocked by a player before reaching the net.

Center	A forward who has more responsibility in the defensive zone. They will also be the primary faceoff takers.
Check/Hit	A defensive player hits the offensive player with the puck and causes them to lose possession.
Defensemen	A player who is more focused on defense. They play further from the net in the offensive zone and closer to the net in the defensive zone.
Deflection	A shot attempt that changes direction unintentionally.
Empty Net/Extra Attacker	When a team pulls their goalie off the ice so an extra skater can join the play. Usually happens when a team is behind late in the game or when there is a delayed penalty on the opposing team.
Faceoff	A restart in play as an official drops the puck between two players, one from each team.
Forward	At 5-on-5, a player who is more focused on offense. They play closer to the net in the offensive zone. Usually consists of 2 wings and 1 center.
Giveaway	An offensive player turns the puck over on their own - mishandling the puck, throwing the puck away, etc.
Goal	A shot attempt that goes into the net.
Goal Differential	The number of goals scored minus the number of goals allowed.
Missed Shot	A shot attempt that would not have scored regardless of whether the goalie had saved it. Shots that hit the post and do not deflect into the net are missed shots.
Off wing	A player playing on the opposite side of their handedness - a left handed player playing or shooting from the right side of the ice.

Plus/Minus	The goal differential for a player when a goal is scored at even strength - 5-on-5, 4-on-4, 6-on-5, etc. Each offensive player on the ice is credited +1 and each defensive player -1.
Salary Cap	A limit on how much a team can spend on player salaries. The limit as of the 2020-2021 season is 81.5 million dollars.
Save	A shot attempt that would have scored had the goalie not blocked it.
Save Percentage	The percent of shots on goal a goalie successfully saves.
Scoring Chance	A shot from a dangerous area of the ice, usually between the faceoff dots and lower than the tops of the circles.
Shot Attempt	A player shoots the puck towards the net in an attempt to score.
Shot Differential	The number of shots for minus the number of shots allowed.
Shot on Goal	A shot attempt that scores a goal or would have scored a goal had the goalie not stopped the puck. Shots that hit the post or would have missed the net are not included.
Slap Shot	A shooter will raise their stick behind them then bring it back down as they transfer their weight from their back leg to their front leg. The fastest shot speed but takes longer than a wrist shot or a snap shot.
Snap Shot	Similar to a wrist shot, a shooter will lean on their stick and quickly snap their wrists as they shoot. Their weight will be on the same leg as their shooting handedness - left leg for a left handed shooter
Takeaway	A defensive player takes the puck from the offensive team - intercepting a pass, stealing the puck from a player, checking a player with the puck, etc.

Tipped Shot	A shot attempt that is purposefully deflected by an offensive player, usually with their stick.
Wing	A forward who has less responsibility in the defensive zone.
Wrap Around	A shot attempt where the offensive player attempts to score by stuffing the puck into the net after being behind the net.
Wrist Shot	A shooter will bring the puck transfer their weight from their back leg to their front leg

6.4 Table of Predictors Used

Table 30: Predictors Used in Models

Predictor	Mean, IQR / Unique Levels
averageRestDifference	Mean: -2.042, +/- 1.5*IQR: (-26.642, 22.558)
defendingPenalty1Length	0: 243656, 120: 49346, 240: 1042, 300: 452, 600: 354
defendingPenalty1TimeLeft	Mean: 10.393, +/- 1.5*IQR: (10.393, 10.393)
defendingTeamAverageTimeOnIce	Mean: 33.088, +/- 1.5*IQR: (0.463, 65.713)
defendingTeamAverageTimeOnIceOfDefencemen	Mean: 36.142, +/- 1.5*IQR: (-5.108, 77.392)
defendingTeamAverageTimeOnIceOfForwards	Mean: 31.09, +/- 1.5*IQR: (-3.41, 65.59)
defendingTeamDefencemenOnIce	0: 110, 1: 6804, 2: 284137, 3: 3759, 4: 40
defendingTeamForwardsOnIce	0: 24, 1: 2819, 2: 55648, 3: 230740, 4: 5167, 5: 442, 6: 10
defendingTeamGoals	0: 103451, 1: 79991, 2: 55266, 3: 32248, 4: 15094, 5: 5975, 6: 1970, 7: 657, 8: 150, 9: 41, 10: 7
defendingTeamMaxTimeOnIce	Mean: 45.501, +/- 1.5*IQR: (-2.499, 93.501)

defendingTeamMinTimeOnIce	Mean: 22.51, +/- 1.5*IQR: (-10.49, 55.51)
defendingTeamSkatersOnIce	0: 18, 1: 7, 2: 7, 3: 4357, 4: 50862, 5: 238349, 6: 1248, 7: 1, 8: 1
distanceFromLastEvent	Mean: 60.831, +/- 1.5*IQR: (-27.523, 149.186)
isPlayoffGame	0: 273200, 1: 21650
lastEventAngleFromNet	Mean: -0.317, +/- 1.5*IQR: (-63.271, 62.636)
lastEventCategory	BLOCK: 33135, FAC: 75532, GIVE: 26363, HIT: 55693, MISS: 28283, OTHER: 430, SHOT: 55541, TAKE: 19873
lastEventDistFromNet	Mean: 71.227, +/- 1.5*IQR: (-50.972, 193.426)
lastEventTeam	DEFENDING: 114714, OTHER: 127, SHOOTING: 180009
lastEventxCord_adjusted	Mean: 24.843, +/- 1.5*IQR: (-108.657, 158.343)
lastEventyCord_adjusted	Mean: 0.014, +/- 1.5*IQR: (-65.986, 66.014)
offWing	0: 172614, 1: 122236
period	1: 95288, 2: 100366, 3: 93579, 4: 5429, 5: 153, 6: 28, 7: 5, 8: 2

playerPositionThatDidEvent	C: 98960, D: 85584, G: 11, L: 55414, MISSING: 281, R: 54600
shooterLeftRight	L: 179008, MISSING: 23, R: 115819
shootingPenalty1Length	0: 280124, 120: 14049, 240: 260, 300: 156, 600: 261
shootingPenalty1TimeLeft	Mean: 2.949, +/- 1.5*IQR: (2.949, 2.949)
shootingTeamAverageTimeOnIce	Mean: 31.046, +/- 1.5*IQR: (0.746, 61.346)
shootingTeamAverageTimeOnIceOfDefencemen	Mean: 32.298, +/- 1.5*IQR: (-5.202, 69.798)
shootingTeamAverageTimeOnIceOfForwards	Mean: 31.68, +/- 1.5*IQR: (-2.82, 66.18)
shootingTeamDefencemenOnIce	0: 651, 1: 28079, 2: 264259, 3: 1842, 4: 18, 5: 1
shootingTeamForwardsOnIce	0: 19, 1: 195, 2: 17288, 3: 247085, 4: 28509, 5: 1710, 6: 44
shootingTeamGoals	0: 109677, 1: 83663, 2: 53908, 3: 28519, 4: 12340, 5: 4664, 6: 1489, 7: 437, 8: 118, 9: 29, 10: 6
shootingTeamMaxTimeOnIce	Mean: 43.851, +/- 1.5*IQR: (-1.149, 88.851)
shootingTeamMinTimeOnIce	Mean: 20.045, +/- 1.5*IQR: (-12.955, 53.045)

shootingTeamSkatersOnIce	0: 14, 1: 6, 2: 2, 3: 1573, 4: 14988, 5: 273213, 6: 5049, 7: 4, 8: 1
shotAngleAdjusted	Mean: 30.02, +/- 1.5*IQR: (-9.732, 69.772)
shotDistance	Mean: 34.837, +/- 1.5*IQR: (-11.953, 81.627)
shotOnEmptyNet	0: 293548, 1: 1302
shotRebound	0: 279558, 1: 15292
shotType	BACK: 22103, DEFL: 6042, MISSING: 38, SLAP: 61609, SNAP: 41266, TIP: 17951, WRAP: 2991, WRIST: 142850
speedFromLastEvent	Mean: 7.931, +/- 1.5*IQR: (-3.499, 19.36)
timeSinceFaceoff	Mean: 62.164, +/- 1.5*IQR: (-32.336, 156.664)
timeSinceLastEvent	Mean: 15.75, +/- 1.5*IQR: (-8.25, 39.75)
xCordAdjusted	Mean: 59.635, +/- 1.5*IQR: (13.135, 106.135)
yCordAdjusted	Mean: -0.179, +/- 1.5*IQR: (-42.179, 41.821)

6.5 Original Training Data Confusion Matrices

Tables 31, 32, 33, 34 show the confusion matrix for each model with the original training data. Rows are model predictions and columns are non-goals and goals. Each model had more false negatives than false positives.

Table 31: XGBoost Confusion Matrix

XGBoost/Goal	0	1
0	274561	16723
1	852	2714

Table 32: Random Forest Confusion Matrix

Random Forest/Goal	0	1
0	274827	17416
1	586	2021

Table 33: Logistic Regression Confusion Matrix

Logistic Regression/Goal	0	1
0	274836	18290
1	577	1147

Table 34: LASSO Confusion Matrix

LASSO/Goal	0	1
0	274914	18461
1	499	976

6.6 Original Training Data Predictor Effects

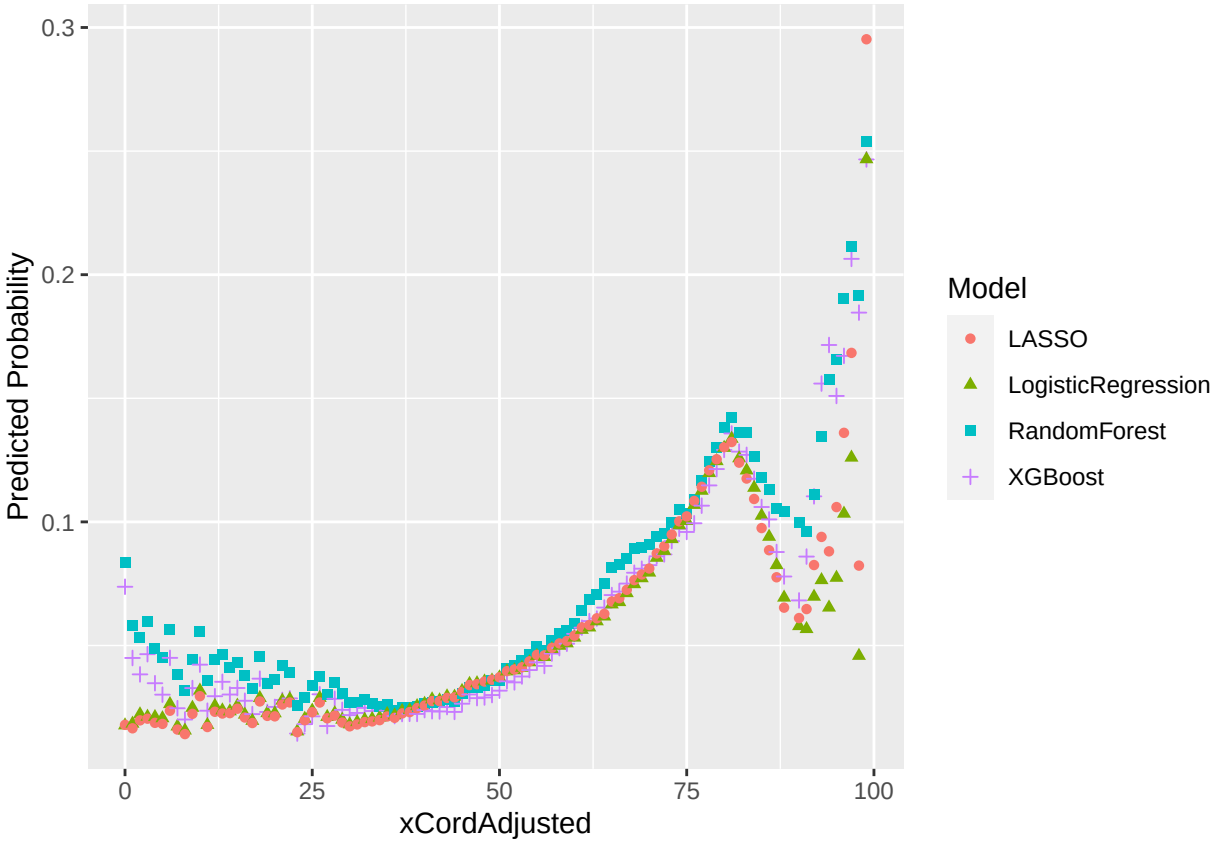

Figure 5: Predicted Goal Probability by X-Coordinate

Table 35: Random Forest Predicted Goal Probability by Shooting Team and Defending Team Skaters

Shooting/Defending Team Skaters	3	4	5	6
3	0.142	0.119	0.130	0.381
4	0.121	0.086	0.085	0.499
5	0.138	0.093	0.064	0.610
6	0.181	0.115	0.097	0.228

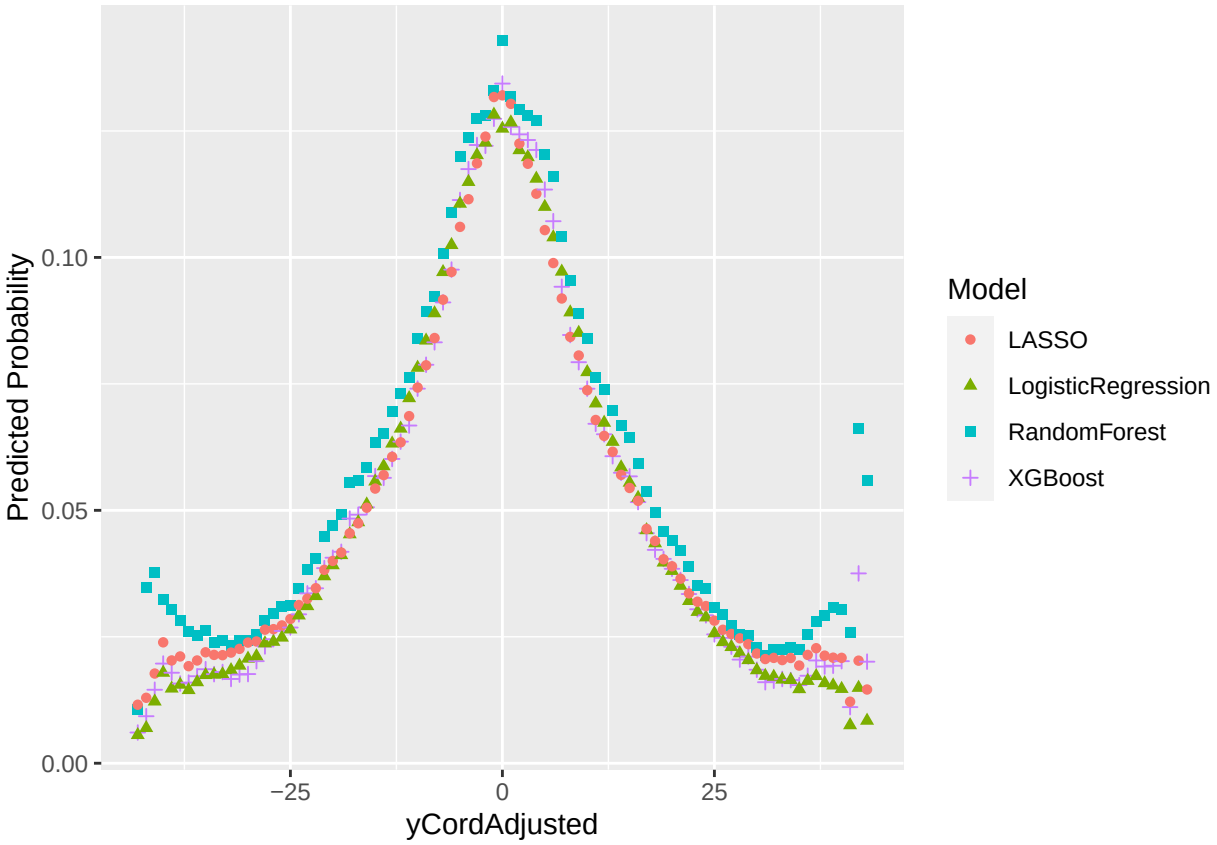

Figure 6: Predicted Goal Probability by Y-Coordinate

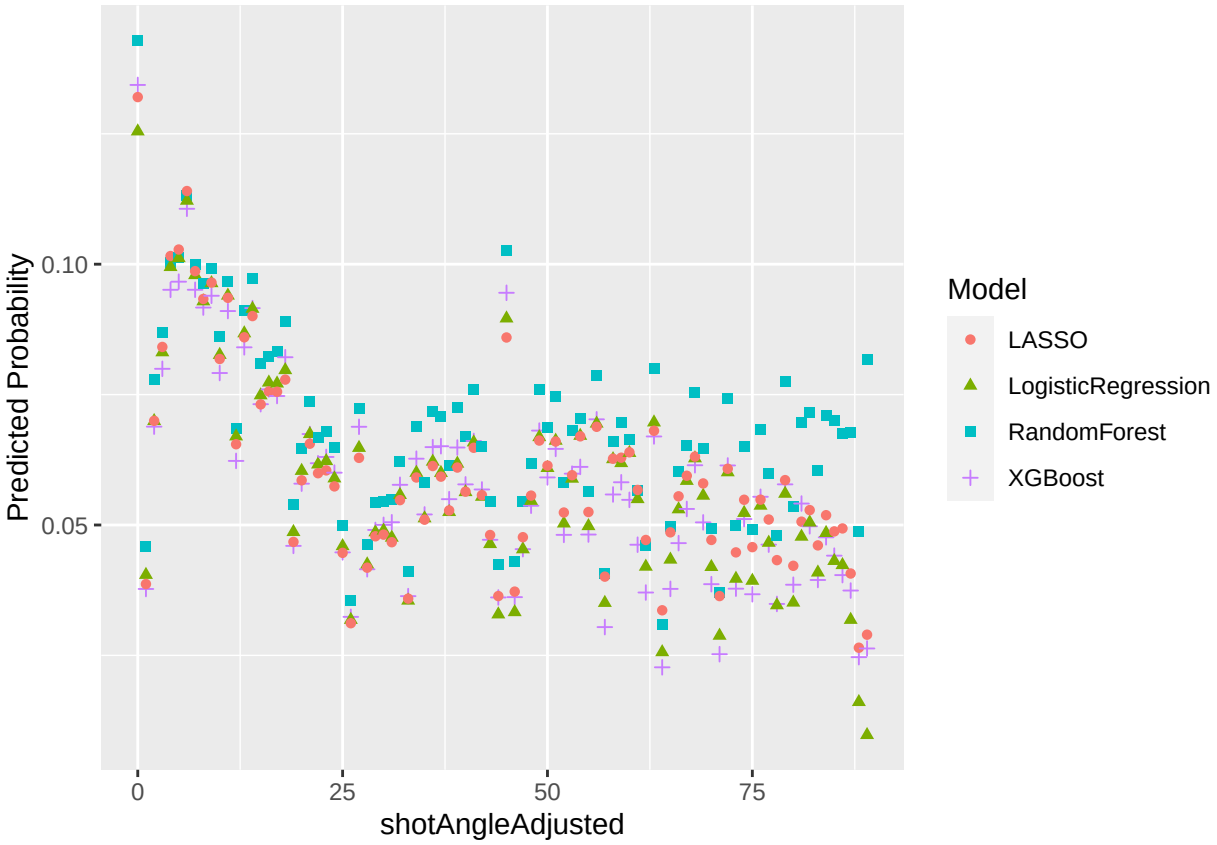

Figure 7: Predicted Goal Probability by Shot Angle

Table 36: Logistic Regression Predicted Goal Probability by Shooting Team and Defending Team Skaters

Shooting/Defending Team Skaters	3	4	5	6
3	0.255	0.042	0.018	0.298
4	0.141	0.063	0.043	0.546
5	0.244	0.069	0.059	0.616
6	0.379	0.130	0.123	0.200

Table 37: LASSO Predicted Goal Probability by Shooting Team and Defending Team Skaters

Shooting/Defending Team Skaters	3	4	5	6
3	0.248	0.045	0.022	0.339
4	0.136	0.064	0.042	0.537
5	0.221	0.070	0.059	0.610
6	0.314	0.120	0.111	0.181

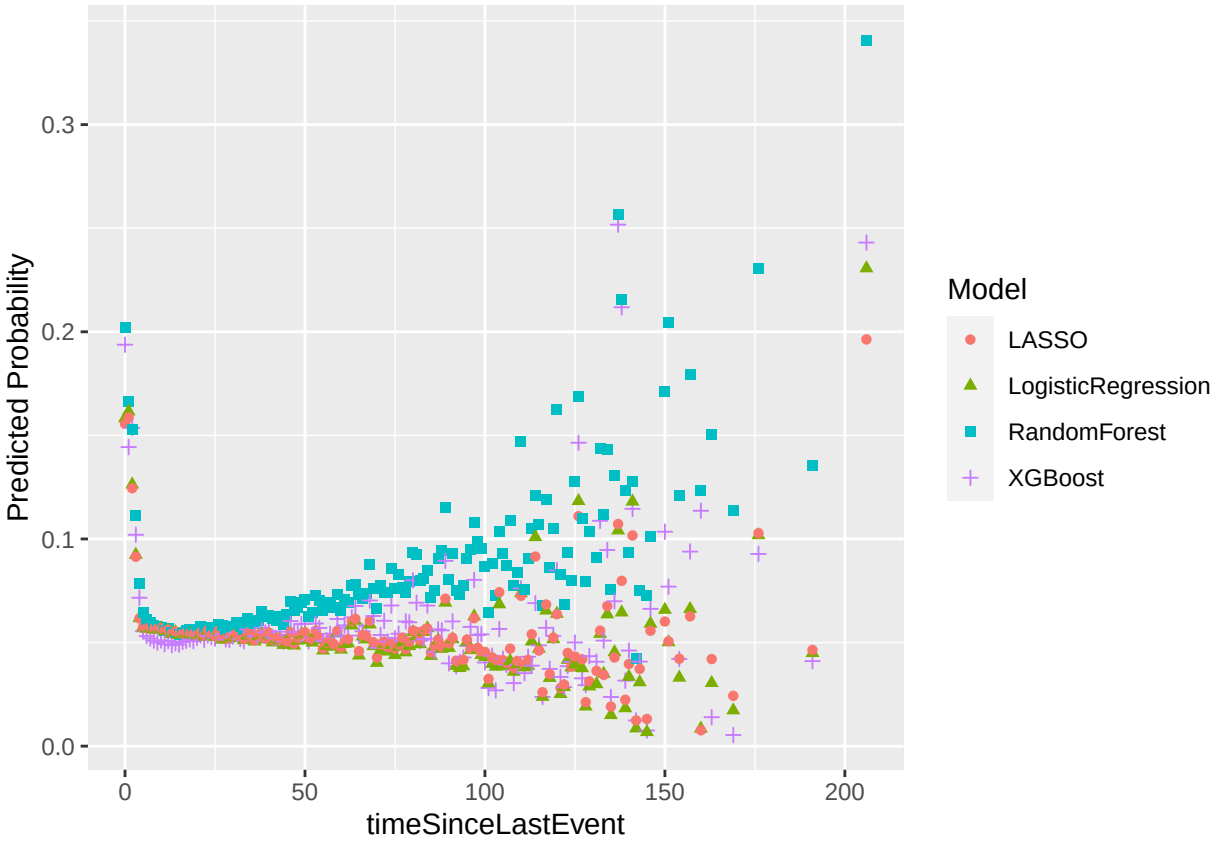

Figure 8: Predicted Goal Probability by Time since Last Event

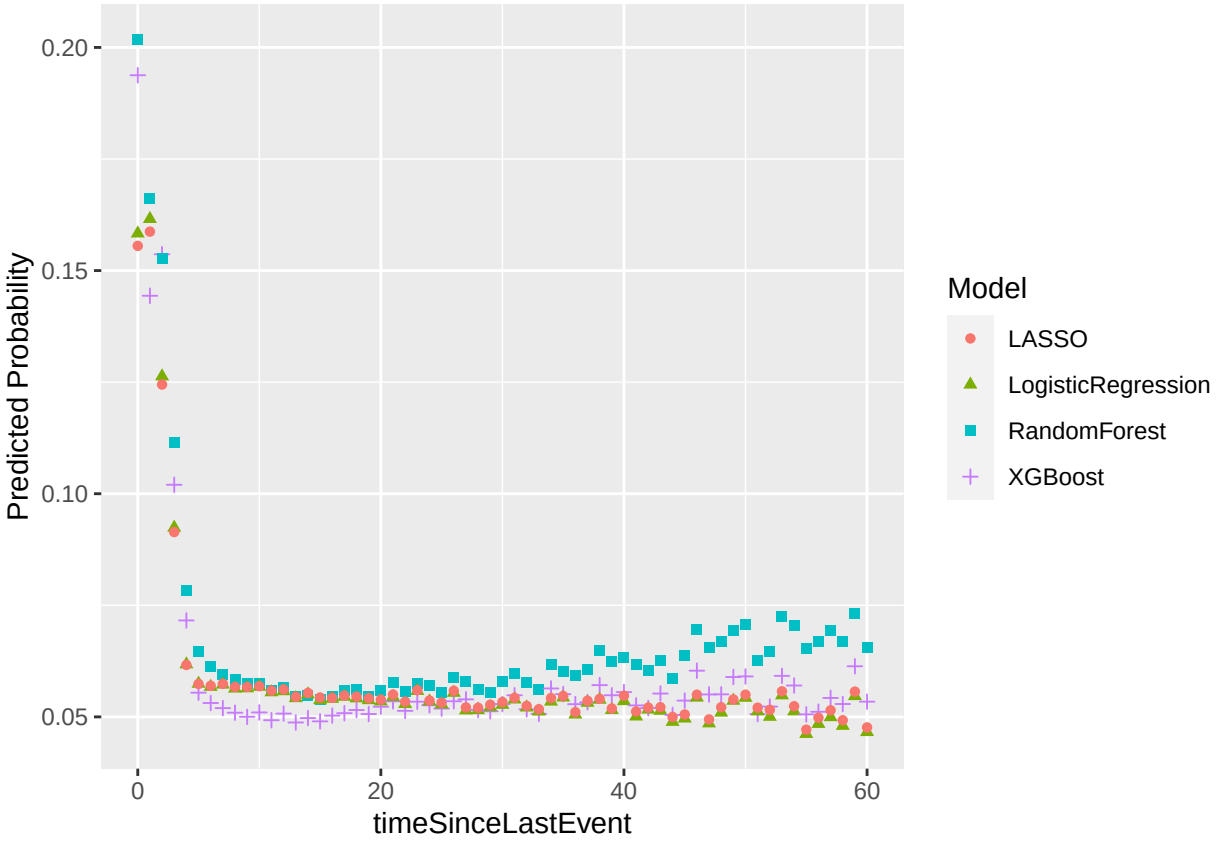

Figure 9: Predicted Goal Probability by Time since Last Event, Max Gap Between Events 60 Seconds

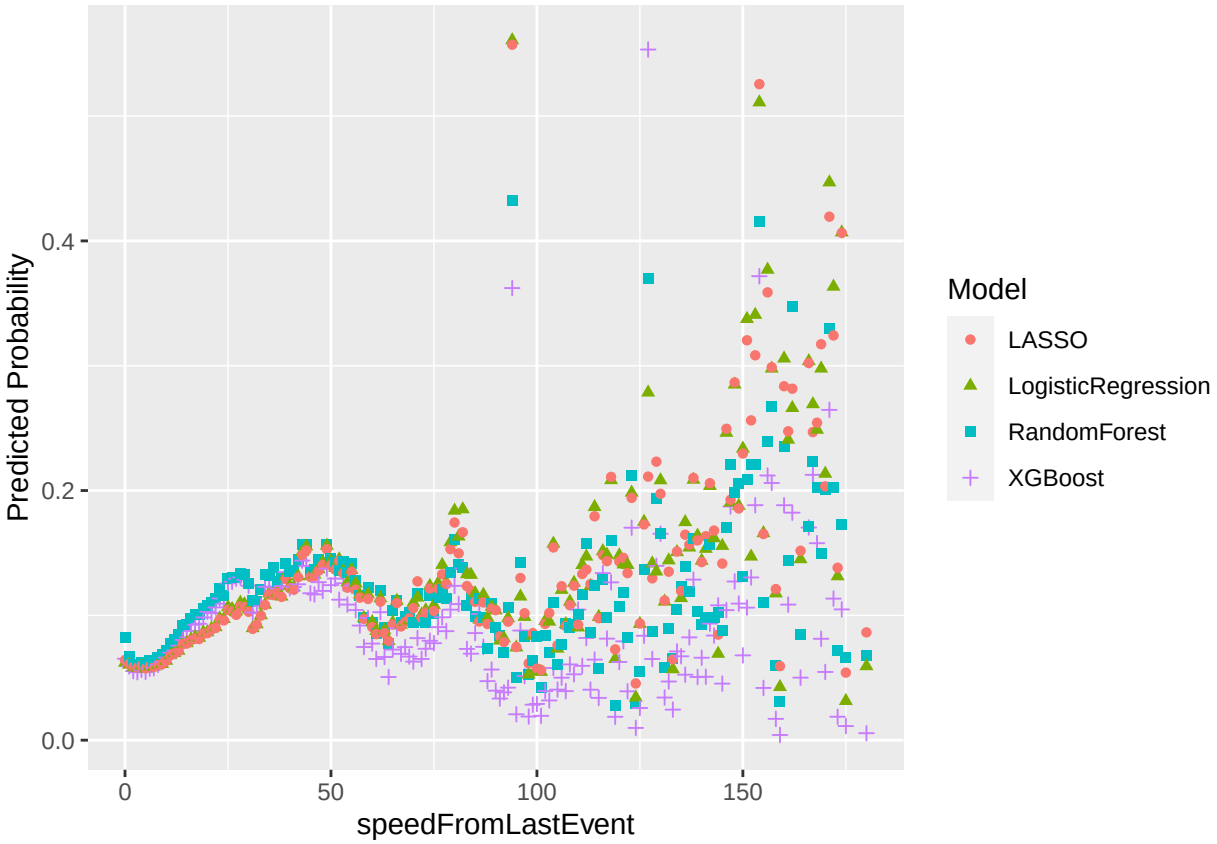

Figure 10: Predicted Goal Probability by Speed from Last Event

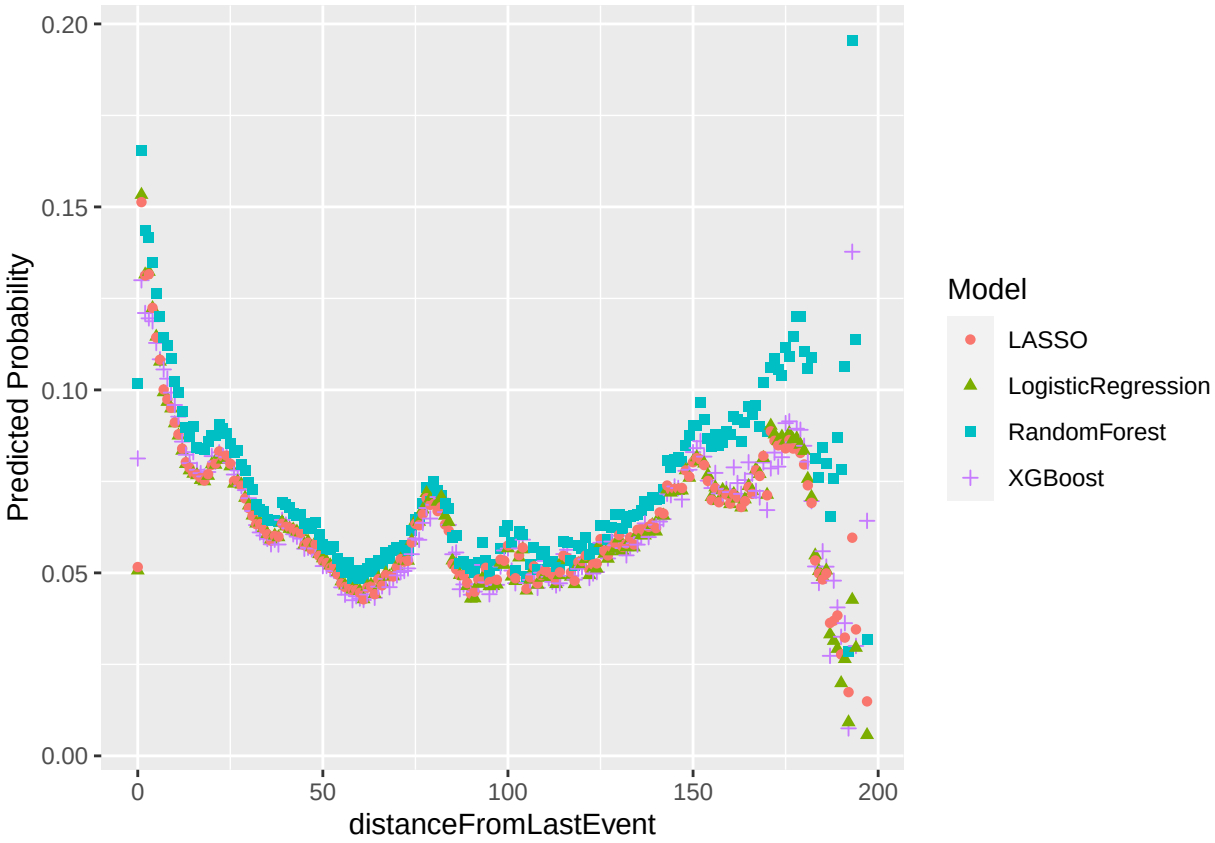

Figure 11: Predicted Goal Probability by Distance from Last Event

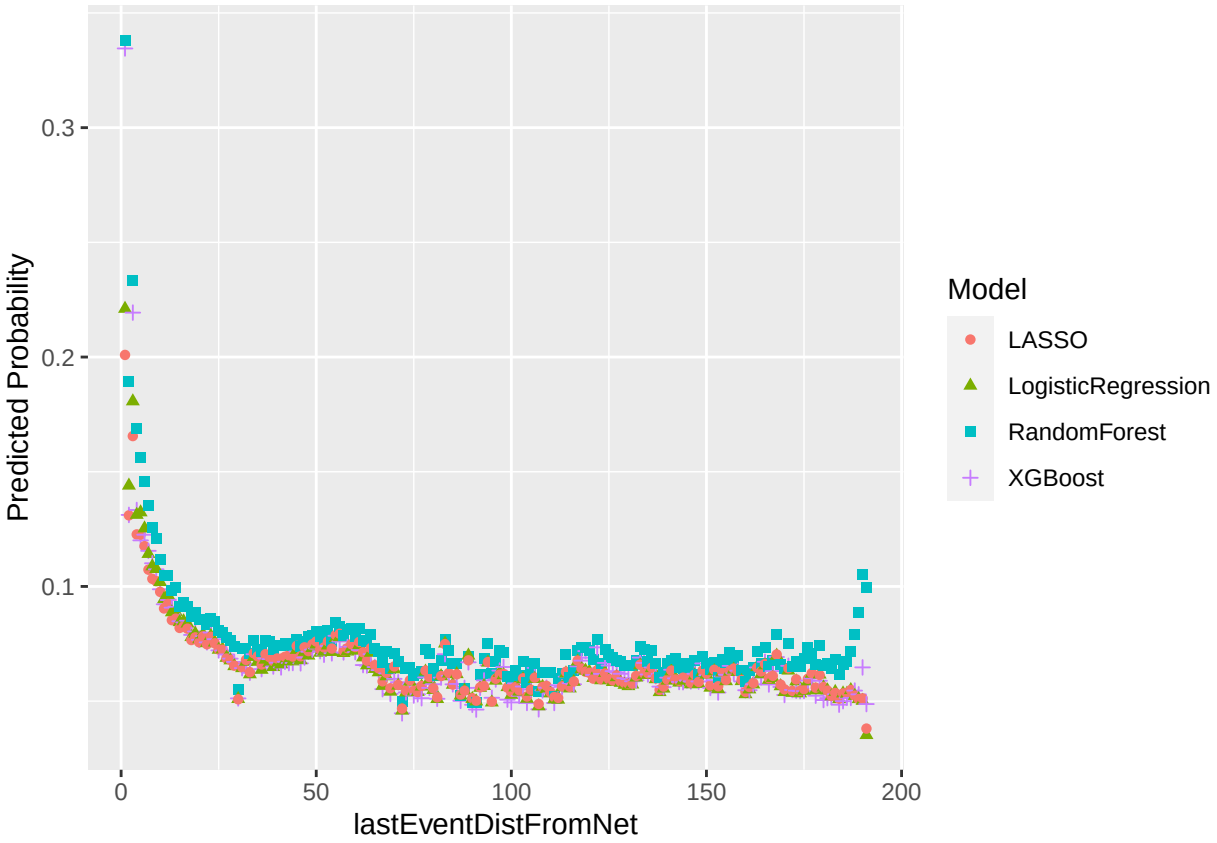

Figure 12: Predicted Goal Probability by Last Event Distance from Net

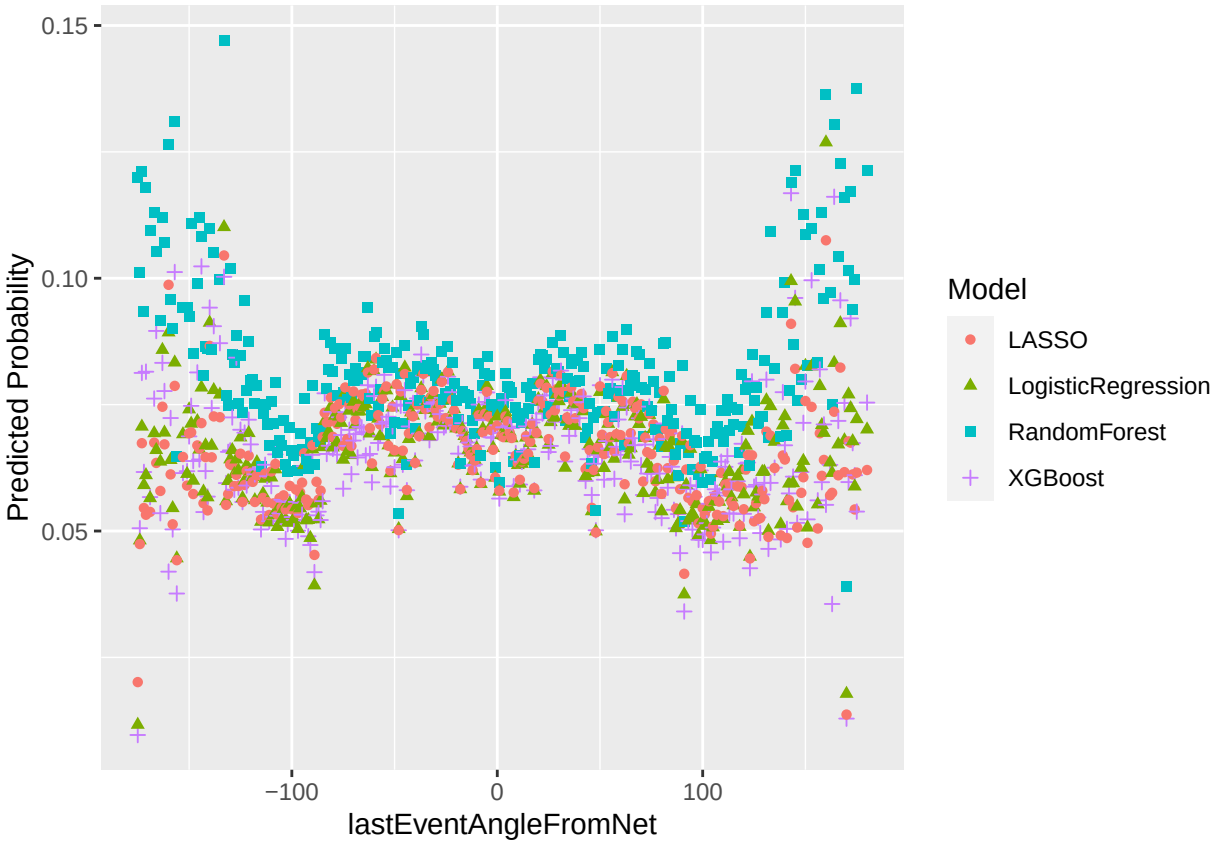

Figure 13: Predicted Goal Probability by Last Event Angle from Net

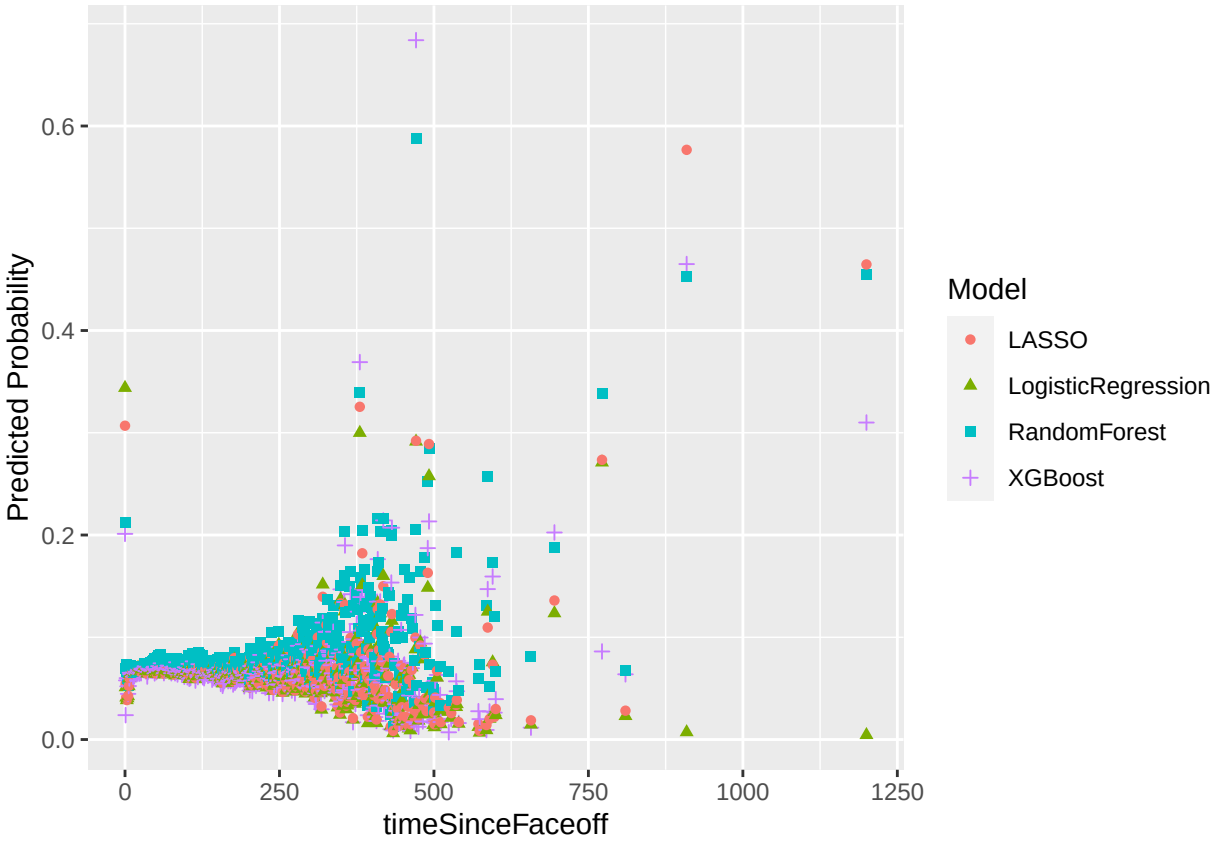

Figure 14: Predicted Goal Probability by Time Since Faceoff

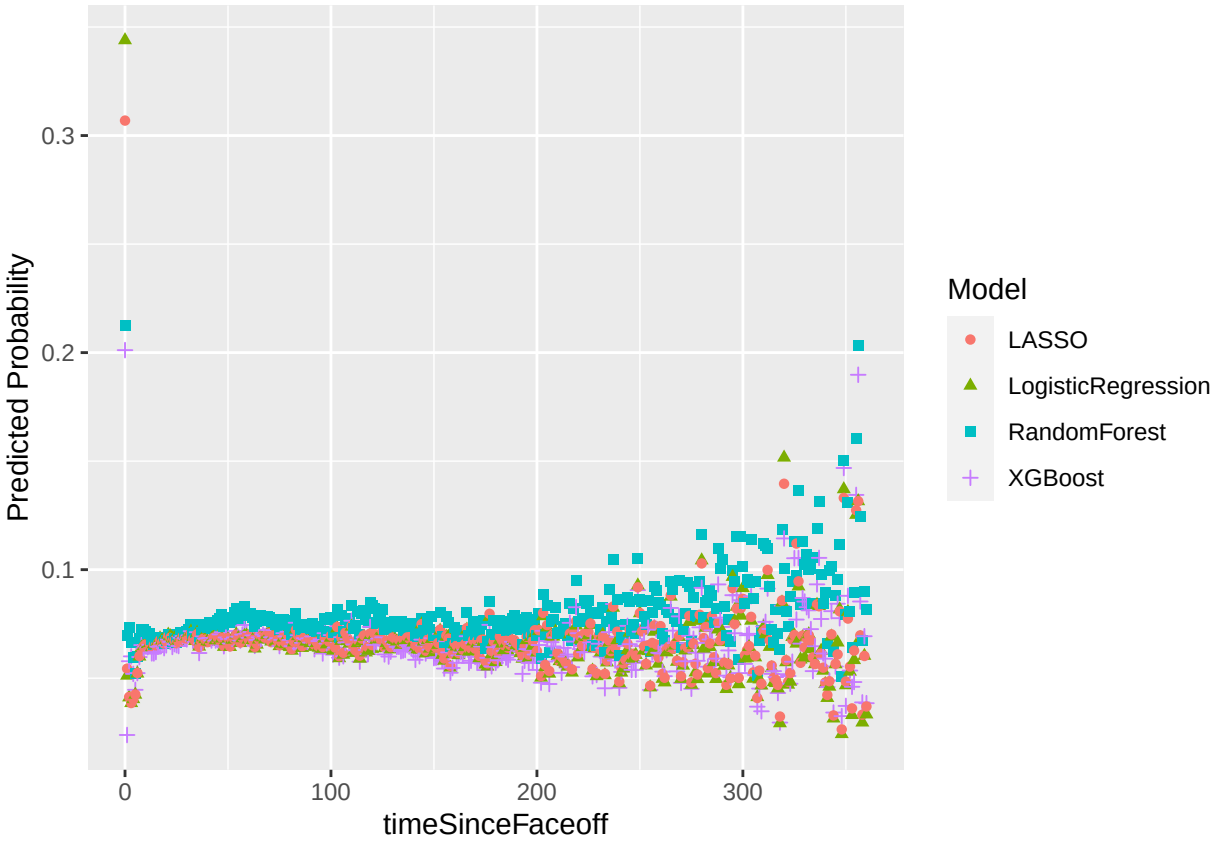

Figure 15: Predicted Goal Probability by Time Since Faceoff, Max Time 360 seconds

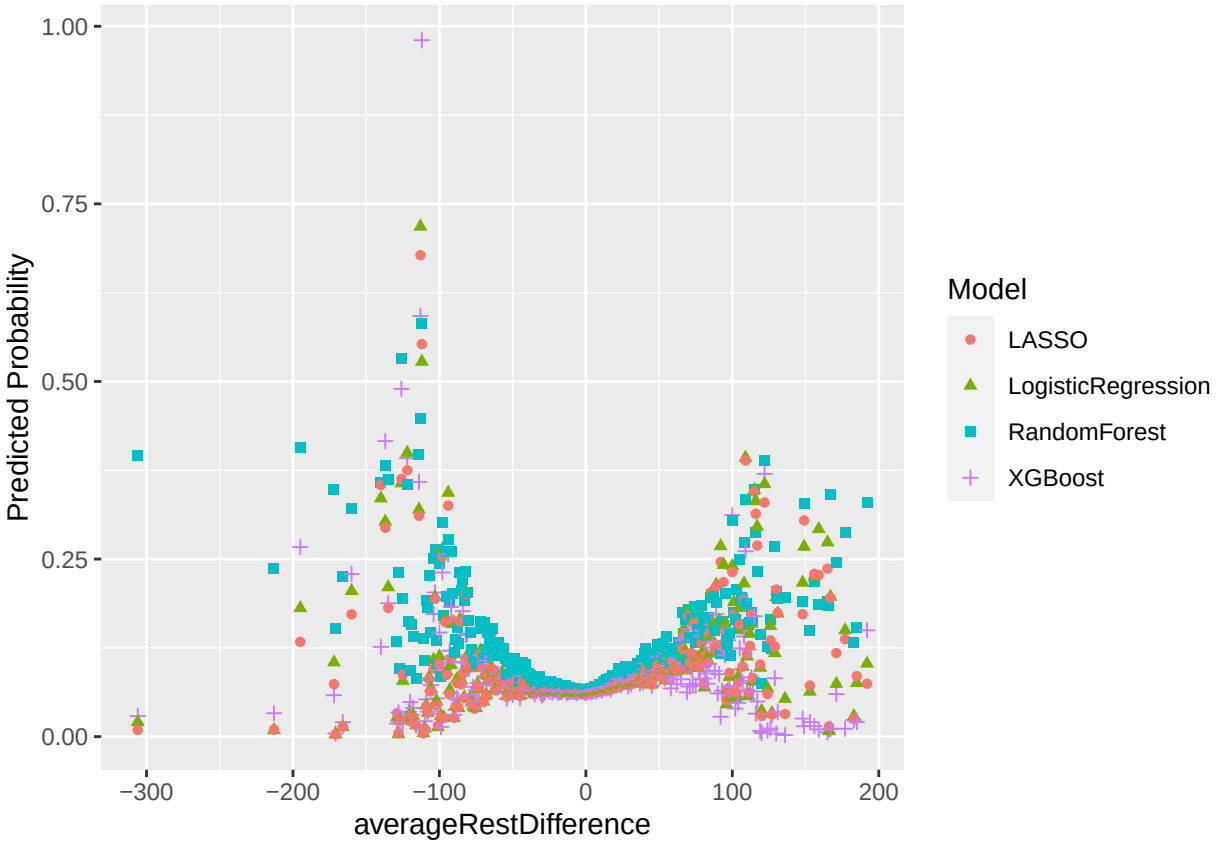

Figure 16: Predicted Goal Probability by Average Rest Difference

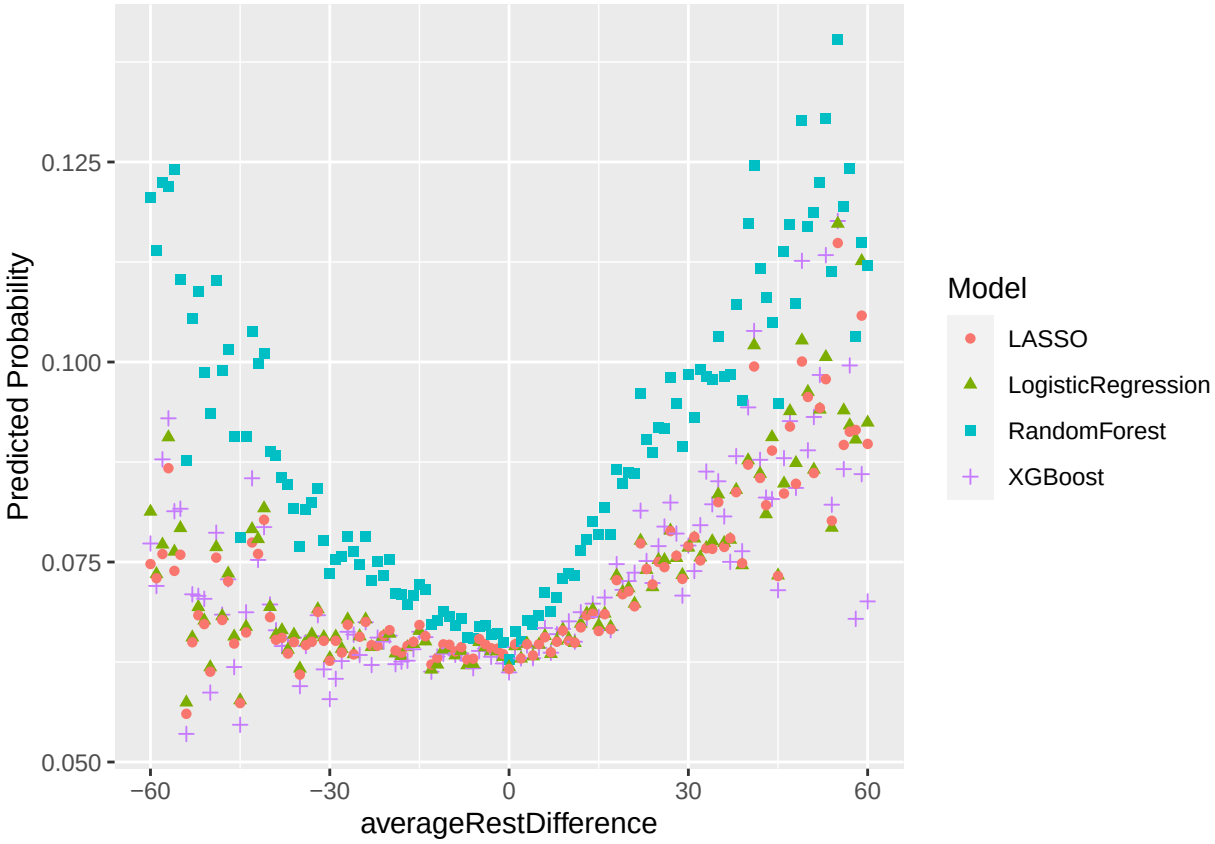

Figure 17: Predicted Goal Probability by Average Rest Difference, Max Difference 60 Seconds

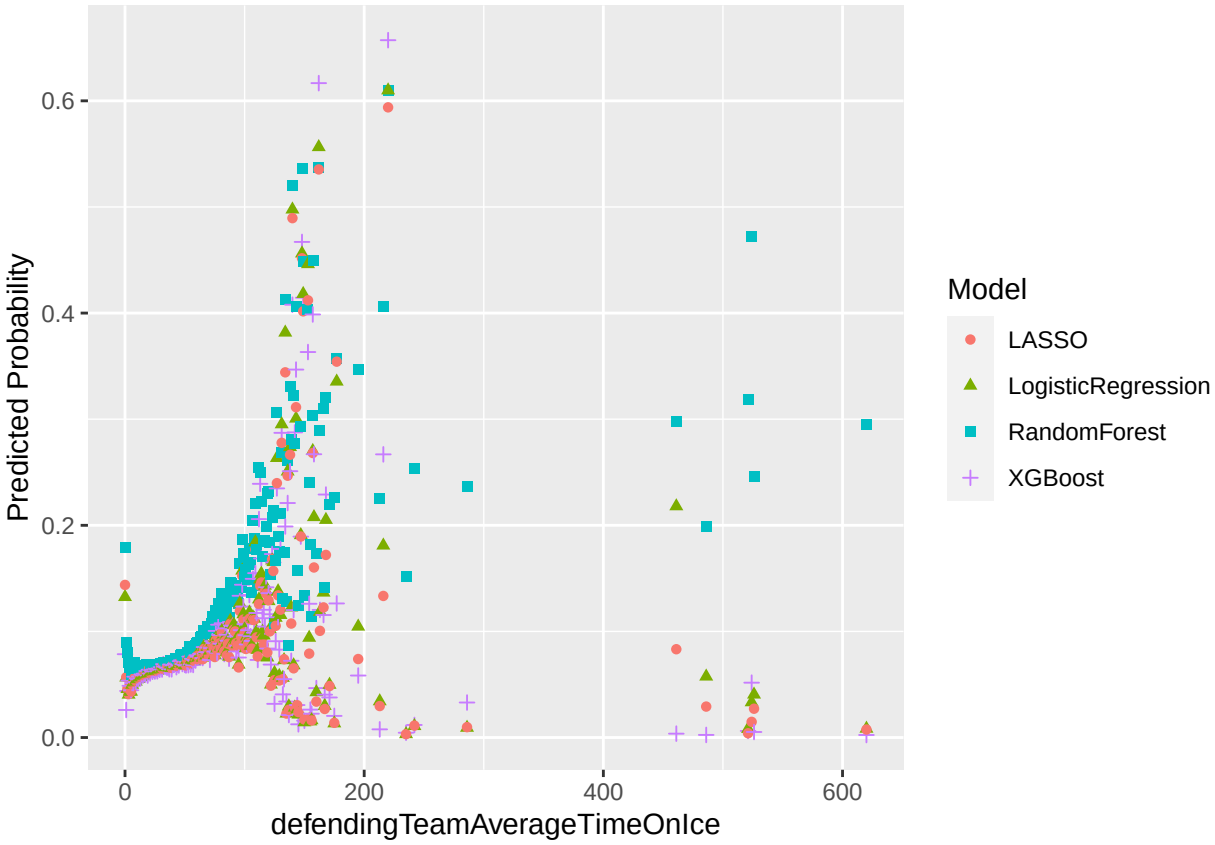

Figure 18: Predicted Goal Probability by Defending Team Average Time-On-Ice

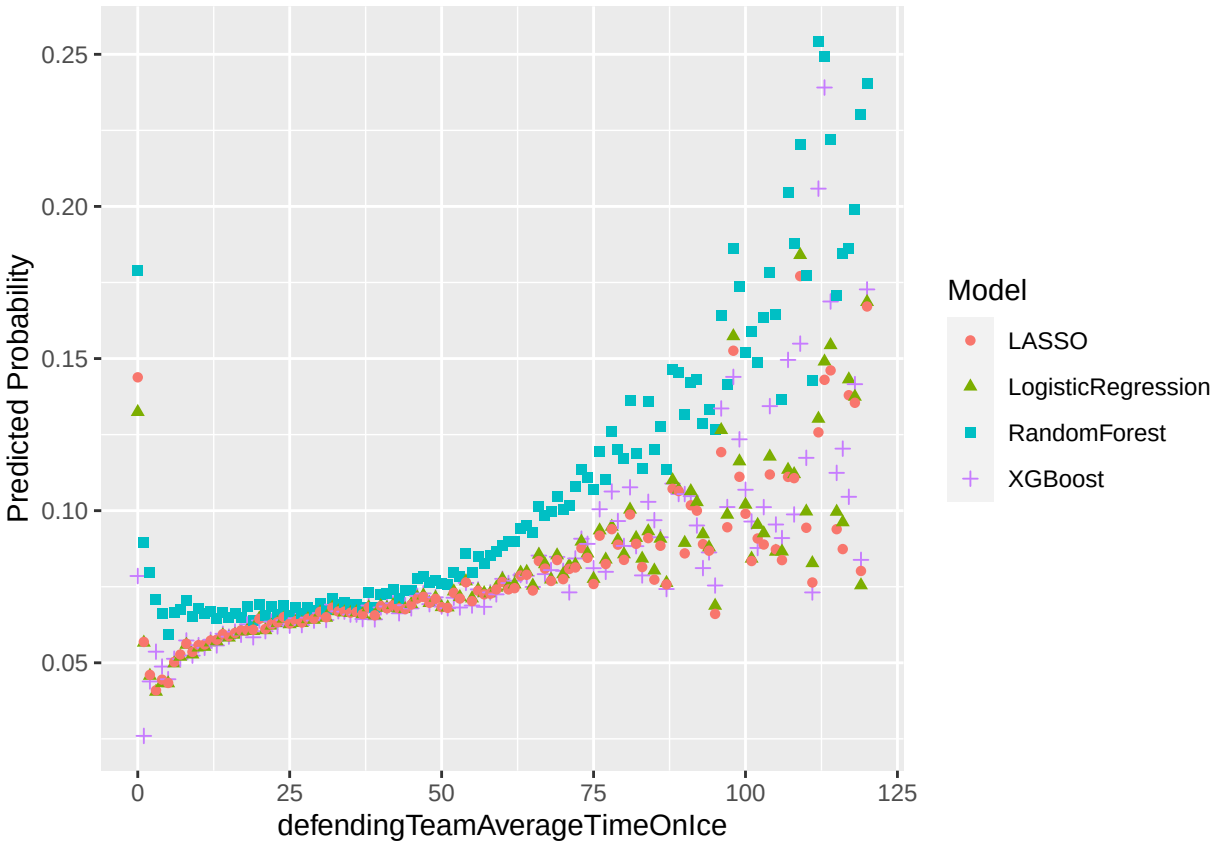

Figure 19: Predicted Goal Probability by Defending Team Average Time-On-Ice, Max 120 Seconds

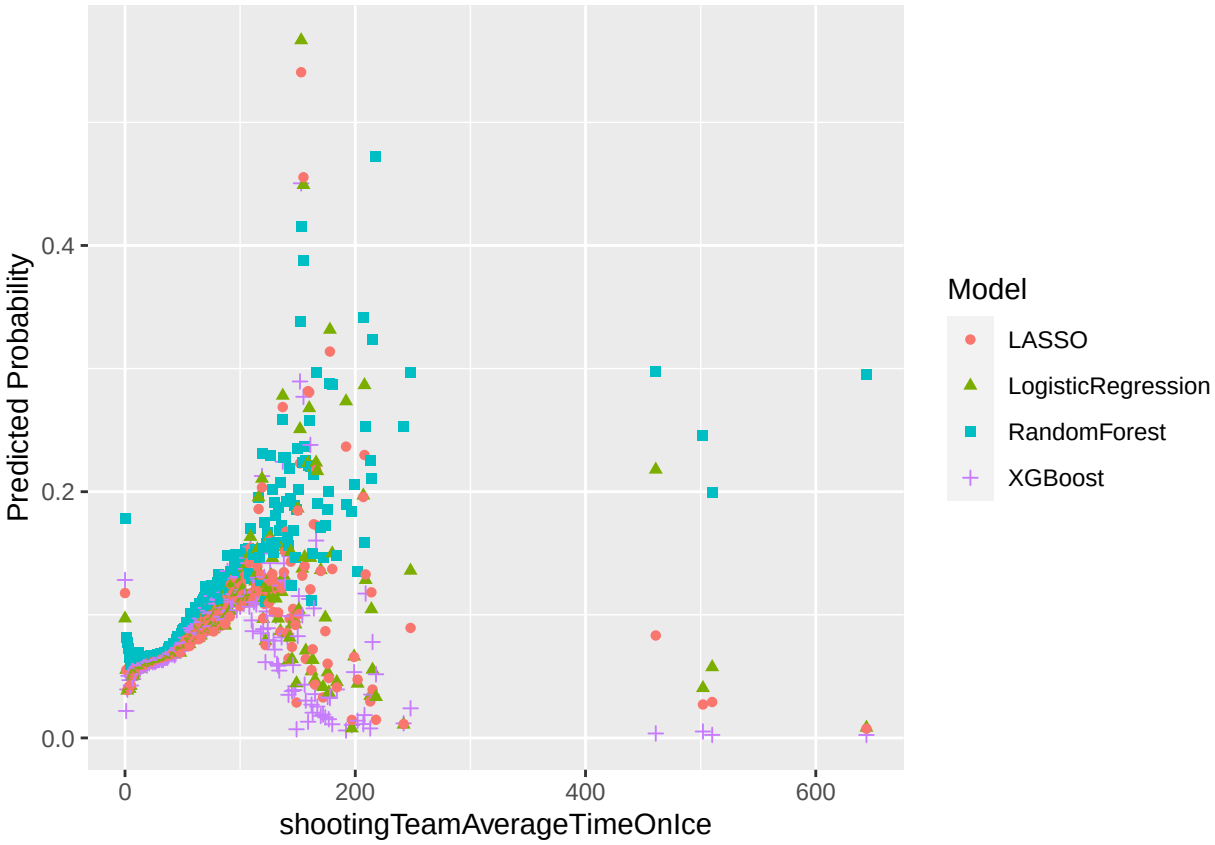

Figure 20: Predicted Goal Probability by Shooting Team Average Time-On-Ice

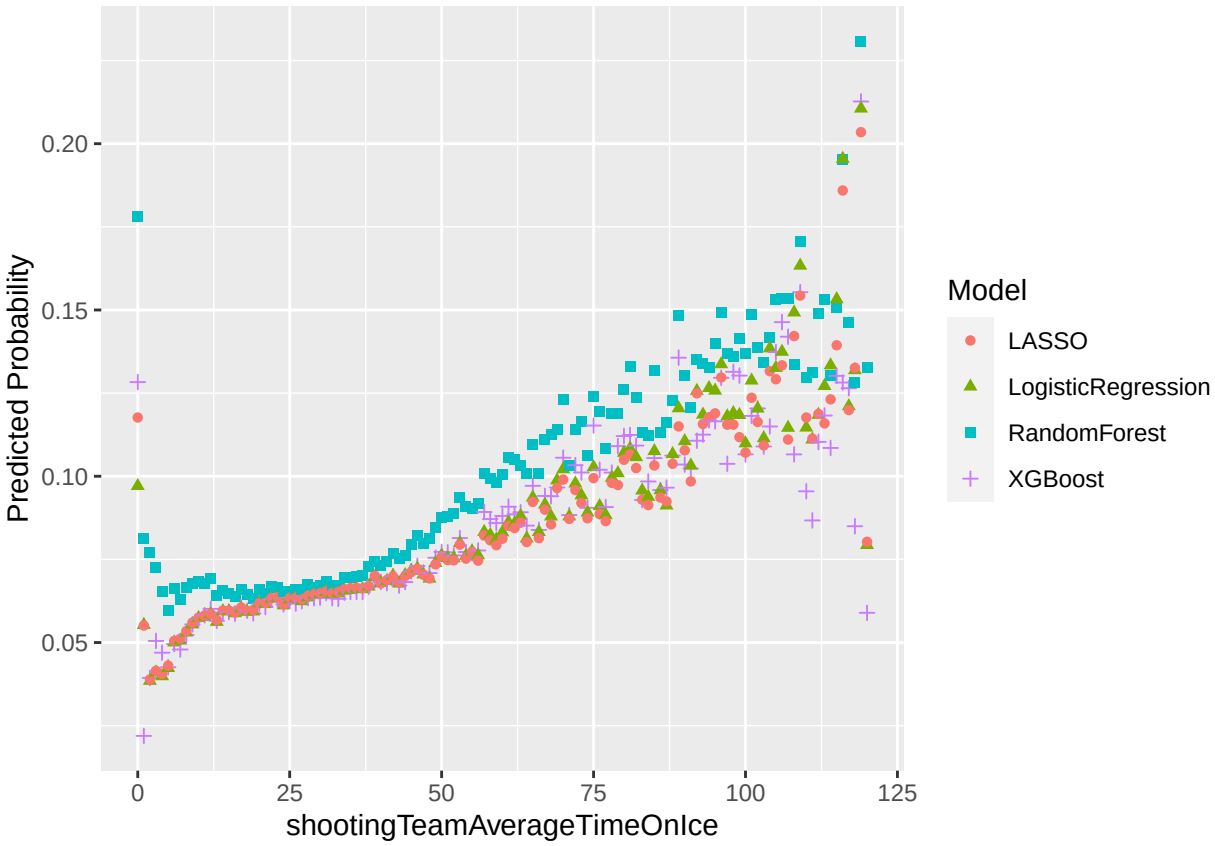

Figure 21: Predicted Goal Probability by Shooting Team Average Time-On-Ice, Max 120 Seconds

6.7 Original Training Data Variable Importance / GLM Coefficients

Table 38: XGBoost Feature Importance

Feature	Gain	Cover	Frequency
shotDistance	0.31403	0.10927	0.07476
shotOnEmptyNet_0	0.09067	0.03836	0.01450
defendingTeamSkatersOnIce	0.08495	0.04252	0.02685
timeSinceLastEvent	0.08319	0.06707	0.05282
defendingPenalty1Length_0	0.07608	0.00433	0.00681
shotAngleAdjusted	0.04910	0.04252	0.05093
shootingTeamSkatersOnIce	0.04697	0.01101	0.01046
defendingPenalty1TimeLeft	0.03882	0.01686	0.02206
speedFromLastEvent	0.01652	0.03018	0.04652
shootingTeamForwardsOnIce	0.01587	0.00479	0.00807
yCordAdjusted	0.01251	0.03368	0.03656
shotRebound_0	0.01110	0.00803	0.00895
shootingTeamMinTimeOnIce	0.00957	0.03561	0.02963
xCordAdjusted	0.00953	0.02434	0.03580
lastEventCategory_HIT	0.00943	0.00298	0.00454
shotType_BACK	0.00768	0.01305	0.00807
lastEventDistFromNet	0.00680	0.02685	0.03139
timeSinceFaceoff	0.00678	0.02008	0.03517
distanceFromLastEvent	0.00592	0.01979	0.03480
shotType_SNAP	0.00583	0.00862	0.00530
shootingPenalty1TimeLeft	0.00567	0.01019	0.01210
shotType_TIP	0.00559	0.00877	0.01147

Feature	Gain	Cover	Frequency
shotType_WRAP	0.00541	0.01137	0.00517
defendingTeamForwardsOnIce	0.00505	0.01511	0.01236
period	0.00474	0.02189	0.01336
lastEventAngleFromNet	0.00468	0.01581	0.02811
offWing_0	0.00465	0.00374	0.00983
defendingTeamMinTimeOnIce	0.00457	0.02587	0.02421
shotType_WRIST	0.00434	0.00217	0.00340
shotType_SLAP	0.00356	0.01401	0.00706
shootingPenalty1Length_0	0.00350	0.00503	0.00403
shootingTeamAverageTimeOnIceOfDefencemen	0.00348	0.01671	0.02131
shotType_DEFL	0.00332	0.00974	0.00983
shootingTeamAverageTimeOnIce	0.00265	0.01363	0.01853
shootingTeamAverageTimeOnIceOfForwards	0.00252	0.02573	0.02610
playerPositionThatDidEvent_MISSING	0.00251	0.00414	0.00681
defendingTeamAverageTimeOnIceOfDefencemen	0.00237	0.01985	0.02295
averageRestDifference	0.00228	0.01953	0.02421
shootingTeamMaxTimeOnIce	0.00228	0.01259	0.02131
lastEventyCord_adjusted	0.00226	0.01530	0.02005
lastEventxCord_adjusted	0.00226	0.01634	0.01614
defendingTeamAverageTimeOnIceOfForwards	0.00221	0.01603	0.01778
defendingTeamAverageTimeOnIce	0.00208	0.01820	0.02093
lastEventCategory_SHOT	0.00206	0.00416	0.00668
shootingTeamGoals	0.00172	0.01073	0.01009
defendingTeamMaxTimeOnIce	0.00151	0.01050	0.01778
lastEventCategory_GIVE	0.00140	0.00575	0.00315

Feature	Gain	Cover	Frequency
shootingPenalty1Length_120	0.00110	0.00294	0.00164
playerPositionThatDidEvent_D	0.00106	0.00262	0.00630
defendingTeamGoals	0.00103	0.01287	0.01046
isPlayoffGame_0	0.00103	0.00469	0.00454
defendingPenalty1Length_120	0.00099	0.00746	0.00391
defendingTeamDefencemenOnIce	0.00077	0.00701	0.00277
lastEventCategory_TAKE	0.00055	0.00190	0.00252
lastEventTeam_DEFENDING	0.00054	0.00013	0.00240
shotType_MISSING	0.00047	0.00826	0.00265
defendingPenalty1Length_240	0.00027	0.00935	0.00303
lastEventCategory_BLOCK	0.00027	0.00050	0.00151
playerPositionThatDidEvent_C	0.00022	0.00042	0.00177
lastEventCategory_FAC	0.00020	0.00004	0.00101
playerPositionThatDidEvent_G	0.00020	0.00937	0.00315
lastEventCategory_MISS	0.00018	0.00048	0.00126
shootingTeamDefencemenOnIce	0.00018	0.00193	0.00164
defendingPenalty1Length_300	0.00013	0.00049	0.00063
lastEventTeam_SHOOTING	0.00012	0.00309	0.00113
playerPositionThatDidEvent_R	0.00010	0.00048	0.00101
lastEventTeam_OTHER	0.00009	0.00105	0.00088
playerPositionThatDidEvent_L	0.00009	0.00002	0.00088
shooterLeftRight_L	0.00009	0.00295	0.00139
shootingPenalty1Length_600	0.00008	0.00305	0.00113
shootingPenalty1Length_240	0.00007	0.00363	0.00126
defendingPenalty1Length_600	0.00006	0.00003	0.00050

Feature	Gain	Cover	Frequency
lastEventCategory_OTHER	0.00005	0.00083	0.00063
shooterLeftRight_R	0.00005	0.00000	0.00076
shooterLeftRight_MISSING	0.00001	0.00155	0.00050

Table 39: Random Forest Feature Importance

	Variable Importance
shotDistance	5789.99289
speedFromLastEvent	4325.86393
distanceFromLastEvent	3853.31978
shotAngleAdjusted	3723.26556
timeSinceFaceoff	3549.27458
lastEventAngleFromNet	3529.68420
lastEventDistFromNet	3478.32164
xCordAdjusted	3474.53101
averageRestDifference	3439.77116
timeSinceLastEvent	3335.51494
shootingTeamAverageTimeOnIce	3270.22510
shootingTeamMaxTimeOnIce	3172.82131
defendingTeamAverageTimeOnIce	3154.49383
yCordAdjusted	3150.07960
shootingTeamAverageTimeOnIceOfForwards	3137.48009
shootingTeamAverageTimeOnIceOfDefencemen	3135.67445
defendingTeamAverageTimeOnIceOfForwards	3105.13444
defendingTeamMaxTimeOnIce	3091.37951

	Variable Importance
defendingTeamAverageTimeOnIceOfDefencemen	3077.15686
defendingTeamMinTimeOnIce	3011.28432
lastEventyCord_adjusted	2962.33310
lastEventxCord_adjusted	2880.34428
shootingTeamMinTimeOnIce	2849.18329
defendingTeamSkatersOnIce	2120.72646
defendingPenalty1TimeLeft	1552.75420
shootingTeamGoals	1517.46256
defendingTeamGoals	1464.99020
period	1216.09160
shotOnEmptyNet_1	1175.32555
defendingTeamForwardsOnIce	1171.49214
shotOnEmptyNet_0	1082.50501
defendingPenalty1Length_0	1011.98322
shootingTeamForwardsOnIce	898.28795
shotRebound_0	868.87984
shootingTeamSkatersOnIce	857.78317
shotRebound_1	812.08856
defendingPenalty1Length_120	659.66368
defendingTeamDefencemenOnIce	553.45614
shootingTeamDefencemenOnIce	473.51048
playerPositionThatDidEvent_D	433.85354
shootingPenalty1TimeLeft	377.91410
shotType_SNAP	373.66888
playerPositionThatDidEvent_C	372.60125

	Variable Importance
playerPositionThatDidEvent_L	370.28411
playerPositionThatDidEvent_R	364.99140
shotType_WRIST	363.25314
lastEventCategory_SHOT	347.08123
shooterLeftRight_R	345.64928
shooterLeftRight_L	343.36662
offWing_0	318.83710
offWing_1	318.39999
shotType_SLAP	304.33517
shotType_TIP	302.23999
lastEventCategory_MISS	293.98499
lastEventCategory_GIVE	282.13693
isPlayoffGame_0	281.02188
isPlayoffGame_1	280.38797
lastEventCategory_BLOCK	274.26997
lastEventTeam_SHOOTING	270.26095
lastEventTeam_DEFENDING	269.78706
lastEventCategory_HIT	267.87126
shotType_BACK	247.59879
lastEventCategory_TAKE	242.98619
shotType_DEFL	218.17585
lastEventCategory_FAC	210.47228
shootingPenalty1Length_0	171.22087
shootingPenalty1Length_120	139.95155
defendingPenalty1Length_240	95.43800

	Variable Importance
shotType_WRAP	93.16498
defendingPenalty1Length_300	50.34512
lastEventCategory_OTHER	41.66050
playerPositionThatDidEvent_MISSING	40.91123
defendingPenalty1Length_600	28.48639
shootingPenalty1Length_600	25.03123
shotType_MISSING	20.77978
lastEventTeam_OTHER	17.54439
shootingPenalty1Length_240	14.83504
shootingPenalty1Length_300	13.56381
shooterLeftRight_MISSING	3.14597
playerPositionThatDidEvent_G	1.77689

```
##
## Call:
## glm(formula = goal ~ ., family = "binomial", data = mp[tr, ])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1072  -0.3887  -0.2616  -0.1799   3.9334
##
## Coefficients: (3 not defined because of singularities)
##                                          Estimate Std. Error z value
## (Intercept)                             7.472e+00  1.831e-01  40.796
## xCordAdjusted                          -4.139e-02  1.810e-03 -22.860
## yCordAdjusted                          -2.821e-04  3.093e-04  -0.912
```

```
## shotTypeBACK                          -3.640e-01  1.417e-02 -25.686
## shotTypeSNAP                           2.803e-01  1.180e-02  23.756
## shotTypeSLAP                           3.627e-01  1.384e-02  26.212
## shotTypeTIP                           -2.157e-01  1.448e-02 -14.895
## shotTypeWRAP                          -9.475e-01  4.275e-02 -22.163
## shotTypeDEFL                          -2.454e-01  2.351e-02 -10.438
## shotTypeMISSING                        1.465e+00  2.296e-01   6.380
## offWing1                               3.940e-02  8.178e-03   4.817
## shooterLeftRightL                     -3.627e-02  8.846e-03  -4.101
## shooterLeftRightMISSING               -4.542e+00  9.605e-01  -4.729
## period                                -8.676e-02  6.270e-03 -13.838
## timeSinceFaceoff                      -2.645e-04  8.359e-05  -3.164
## timeSinceLastEvent                    -1.453e-03  3.441e-04  -4.222
## lastEventCategoryBLOCK                 1.102e-01  1.602e-02   6.876
## lastEventCategoryHIT                  -5.456e-02  1.631e-02  -3.345
## lastEventCategoryGIVE                  3.394e-01  1.677e-02  20.237
## lastEventCategorySHOT                 -1.235e-01  1.510e-02  -8.174
## lastEventCategoryMISS                 -1.276e-02  1.686e-02  -0.757
## lastEventCategoryTAKE                  2.390e-01  1.818e-02  13.148
## lastEventCategoryOTHER                -5.058e-03  1.036e-01  -0.049
## playerPositionThatDidEventL            1.216e-02  1.057e-02   1.150
## playerPositionThatDidEventD           -3.919e-02  1.318e-02  -2.972
## playerPositionThatDidEventR            1.321e-03  1.102e-02   0.120
## playerPositionThatDidEventMISSING      5.075e-01  1.097e-01   4.626
## playerPositionThatDidEventG           -3.217e+00  7.422e-01  -4.335
## lastEventTeamSHOOTING                 -7.406e-02  9.855e-03  -7.515
## lastEventTeamOTHER                     3.097e-01  2.018e-01   1.535
```

```
## lastEventxCord_adjusted                -5.764e-03  6.535e-04  -8.821
## lastEventyCord_adjusted                -3.580e-04  2.588e-04  -1.384
## shotDistance                           -8.801e-02  1.645e-03 -53.513
## shotAngleAdjusted                      -7.648e-03  3.831e-04 -19.964
## distanceFromLastEvent                   3.715e-03  3.024e-04  12.287
## speedFromLastEvent                      8.848e-03  3.632e-04  24.363
## shotOnEmptyNet1                         6.388e+00  4.186e-02 152.580
## lastEventDistFromNet                   -9.707e-03  8.044e-04 -12.067
## lastEventAngleFromNet                   1.605e-04  1.270e-04   1.263
## isPlayoffGame1                         -1.282e-02  1.538e-02  -0.834
## shootingTeamGoals                       5.174e-02  3.494e-03  14.807
## defendingTeamGoals                     -2.175e-02  3.504e-03  -6.206
## averageRestDifference                  -4.863e-03  7.101e-04  -6.849
## shotRebound1                            9.293e-01  1.599e-02  58.116
## shootingTeamMaxTimeOnIce                1.354e-03  3.349e-04   4.042
## shootingTeamMinTimeOnIce                2.859e-03  4.328e-04   6.605
## shootingTeamAverageTimeOnIce            8.383e-03  1.149e-03   7.294
## shootingTeamSkatersOnIce                6.731e-01  2.314e-02  29.086
## shootingPenalty1Length120               4.340e-01  3.325e-02  13.050
## shootingPenalty1Length300               5.348e-01  1.960e-01   2.729
## shootingPenalty1Length240               1.955e-01  1.716e-01   1.139
## shootingPenalty1Length600              -9.104e-02  1.623e-01  -0.561
## shootingPenalty1TimeLeft                1.195e-03  4.604e-04   2.594
## shootingTeamForwardsOnIce               1.474e-01  1.566e-02   9.414
## shootingTeamAverageTimeOnIceOfForwards -4.258e-03  5.443e-04  -7.824
## shootingTeamDefencemenOnIce                    NA         NA      NA
## shootingTeamAverageTimeOnIceOfDefencemen -4.494e-04  7.503e-05  -5.990
```

```
## defendingTeamMaxTimeOnIce                 -8.505e-04  3.322e-04  -2.560
## defendingTeamMinTimeOnIce                 -3.188e-03  3.888e-04  -8.199
## defendingTeamAverageTimeOnIce                     NA         NA      NA
## defendingTeamSkatersOnIce                 -1.438e+00  2.467e-02 -58.286
## defendingPenalty1Length120                -2.041e+00  2.521e-02 -80.965
## defendingPenalty1Length240                -2.087e+00  7.971e-02 -26.178
## defendingPenalty1Length600                -7.497e-01  1.337e-01  -5.608
## defendingPenalty1Length300                -1.485e+00  9.236e-02 -16.081
## defendingPenalty1TimeLeft                 -1.138e-03  2.597e-04  -4.382
## defendingTeamForwardsOnIce                -2.763e-01  2.082e-02 -13.269
## defendingTeamAverageTimeOnIceOfForwards   -2.666e-03  4.027e-04  -6.620
## defendingTeamDefencemenOnIce                      NA         NA      NA
## defendingTeamAverageTimeOnIceOfDefencemen -9.207e-04  2.057e-04  -4.476
##                                           Pr(>|z|)
## (Intercept)                                < 2e-16 ***
## xCordAdjusted                              < 2e-16 ***
## yCordAdjusted                             0.361700
## shotTypeBACK                               < 2e-16 ***
## shotTypeSNAP                               < 2e-16 ***
## shotTypeSLAP                               < 2e-16 ***
## shotTypeTIP                                < 2e-16 ***
## shotTypeWRAP                               < 2e-16 ***
## shotTypeDEFL                               < 2e-16 ***
## shotTypeMISSING                           1.77e-10 ***
## offWing1                                  1.46e-06 ***
## shooterLeftRightL                         4.12e-05 ***
## shooterLeftRightMISSING                   2.26e-06 ***
```

```
## period                                  < 2e-16 ***
## timeSinceFaceoff                       0.001554 **
## timeSinceLastEvent                     2.43e-05 ***
## lastEventCategoryBLOCK                 6.14e-12 ***
## lastEventCategoryHIT                   0.000824 ***
## lastEventCategoryGIVE                   < 2e-16 ***
## lastEventCategorySHOT                  2.97e-16 ***
## lastEventCategoryMISS                  0.449225
## lastEventCategoryTAKE                   < 2e-16 ***
## lastEventCategoryOTHER                 0.961063
## playerPositionThatDidEventL            0.250005
## playerPositionThatDidEventD            0.002955 **
## playerPositionThatDidEventR            0.904589
## playerPositionThatDidEventMISSING      3.73e-06 ***
## playerPositionThatDidEventG            1.46e-05 ***
## lastEventTeamSHOOTING                  5.69e-14 ***
## lastEventTeamOTHER                     0.124828
## lastEventxCord_adjusted                 < 2e-16 ***
## lastEventyCord_adjusted                0.166476
## shotDistance                            < 2e-16 ***
## shotAngleAdjusted                       < 2e-16 ***
## distanceFromLastEvent                   < 2e-16 ***
## speedFromLastEvent                      < 2e-16 ***
## shotOnEmptyNet1                         < 2e-16 ***
## lastEventDistFromNet                    < 2e-16 ***
## lastEventAngleFromNet                  0.206539
## isPlayoffGame1                         0.404526
```

```
## shootingTeamGoals                           < 2e-16 ***
## defendingTeamGoals                         5.42e-10 ***
## averageRestDifference                      7.45e-12 ***
## shotRebound1                                < 2e-16 ***
## shootingTeamMaxTimeOnIce                   5.31e-05 ***
## shootingTeamMinTimeOnIce                   3.98e-11 ***
## shootingTeamAverageTimeOnIce               3.00e-13 ***
## shootingTeamSkatersOnIce                    < 2e-16 ***
## shootingPenalty1Length120                   < 2e-16 ***
## shootingPenalty1Length300                  0.006360 **
## shootingPenalty1Length240                  0.254667
## shootingPenalty1Length600                  0.574854
## shootingPenalty1TimeLeft                   0.009476 **
## shootingTeamForwardsOnIce                   < 2e-16 ***
## shootingTeamAverageTimeOnIceOfForwards     5.11e-15 ***
## shootingTeamDefencemenOnIce                      NA
## shootingTeamAverageTimeOnIceOfDefencemen   2.10e-09 ***
## defendingTeamMaxTimeOnIce                  0.010463 *
## defendingTeamMinTimeOnIce                  2.41e-16 ***
## defendingTeamAverageTimeOnIce                    NA
## defendingTeamSkatersOnIce                   < 2e-16 ***
## defendingPenalty1Length120                  < 2e-16 ***
## defendingPenalty1Length240                  < 2e-16 ***
## defendingPenalty1Length600                 2.05e-08 ***
## defendingPenalty1Length300                  < 2e-16 ***
## defendingPenalty1TimeLeft                  1.17e-05 ***
## defendingTeamForwardsOnIce                  < 2e-16 ***
```

```
## defendingTeamAverageTimeOnIceOfForwards   3.59e-11 ***
## defendingTeamDefencemenOnIce                     NA
## defendingTeamAverageTimeOnIceOfDefencemen 7.59e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 572695  on 1179398  degrees of freedom
## Residual deviance: 487138  on 1179332  degrees of freedom
## AIC: 487272
##
## Number of Fisher Scoring iterations: 6

## 71 x 1 sparse Matrix of class "dgCMatrix"
##                                                      1
## (Intercept)                                 2.984241e+00
## xCordAdjusted                               .
## yCordAdjusted                               .
## shotTypeWRIST                               .
## shotTypeBACK                               -3.063102e-01
## shotTypeSNAP                                2.450457e-01
## shotTypeSLAP                                3.177909e-01
## shotTypeTIP                                -1.741876e-01
## shotTypeWRAP                               -6.939037e-01
## shotTypeDEFL                               -1.867452e-01
## shotTypeMISSING                             9.733835e-01
## offWing1                                    3.758469e-02
```

```
## shooterLeftRightL                    -1.246203e-02
## shooterLeftRightMISSING              -1.078364e-01
## period                               -5.843748e-02
## timeSinceFaceoff                     -1.838463e-04
## timeSinceLastEvent                   -1.231561e-03
## lastEventCategoryBLOCK                9.767186e-02
## lastEventCategoryHIT                 -7.541639e-02
## lastEventCategoryGIVE                 3.065950e-01
## lastEventCategorySHOT                -8.003962e-02
## lastEventCategoryMISS                 .
## lastEventCategoryTAKE                 1.754971e-01
## lastEventCategoryOTHER                .
## playerPositionThatDidEventL           .
## playerPositionThatDidEventD           .
## playerPositionThatDidEventR           .
## playerPositionThatDidEventMISSING     1.140913e-01
## playerPositionThatDidEventG          -1.058776e+00
## lastEventTeamSHOOTING                -2.738307e-02
## lastEventTeamOTHER                    .
## lastEventxCord_adjusted               .
## lastEventyCord_adjusted               .
## shotDistance                         -4.965536e-02
## shotAngleAdjusted                    -1.407811e-02
## distanceFromLastEvent                 .
## speedFromLastEvent                    8.717404e-03
## shotOnEmptyNet1                       6.015186e+00
## lastEventDistFromNet                 -2.599633e-04
```

```
## lastEventAngleFromNet                          .
## isPlayoffGame1                                 .
## shootingTeamGoals                              3.508695e-02
## defendingTeamGoals                            -1.172553e-02
## averageRestDifference                          .
## shotRebound1                                   8.952531e-01
## shootingTeamMaxTimeOnIce                       2.715626e-04
## shootingTeamMinTimeOnIce                       3.793748e-04
## shootingTeamAverageTimeOnIce                   2.287568e-03
## shootingTeamSkatersOnIce                       4.850297e-01
## shootingPenalty1Length120                      2.627291e-01
## shootingPenalty1Length300                      .
## shootingPenalty1Length240                      .
## shootingPenalty1Length600                      .
## shootingPenalty1TimeLeft                       5.620176e-04
## shootingTeamForwardsOnIce                      1.567658e-01
## shootingTeamAverageTimeOnIceOfForwards        -2.922688e-04
## shootingTeamDefencemenOnIce                    .
## shootingTeamAverageTimeOnIceOfDefencemen      -5.193422e-05
## defendingTeamMaxTimeOnIce                      .
## defendingTeamMinTimeOnIce                     -2.269676e-03
## defendingTeamAverageTimeOnIce                  .
## defendingTeamSkatersOnIce                     -1.222951e+00
## defendingPenalty1Length120                    -1.683284e+00
## defendingPenalty1Length240                    -1.527387e+00
## defendingPenalty1Length600                    -7.073485e-02
## defendingPenalty1Length300                    -9.000312e-01
```

```
## defendingPenalty1TimeLeft                  -1.879934e-03
## defendingTeamForwardsOnIce                 -2.518409e-01
## defendingTeamAverageTimeOnIceOfForwards    -5.913912e-04
## defendingTeamDefencemenOnIce                .
## defendingTeamAverageTimeOnIceOfDefencemen -5.622371e-05
```

6.8 Goals vs Predicted Goals

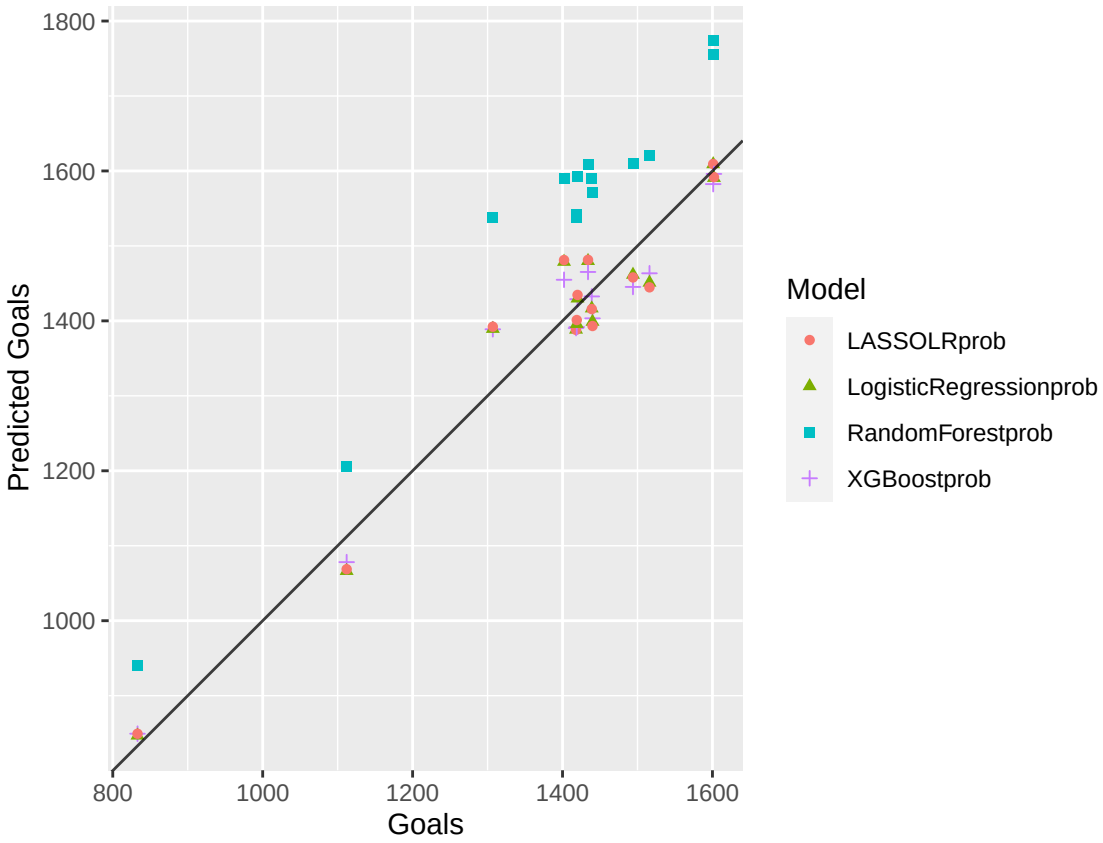

Figure 22: Predicted Goal vs Goals, Season

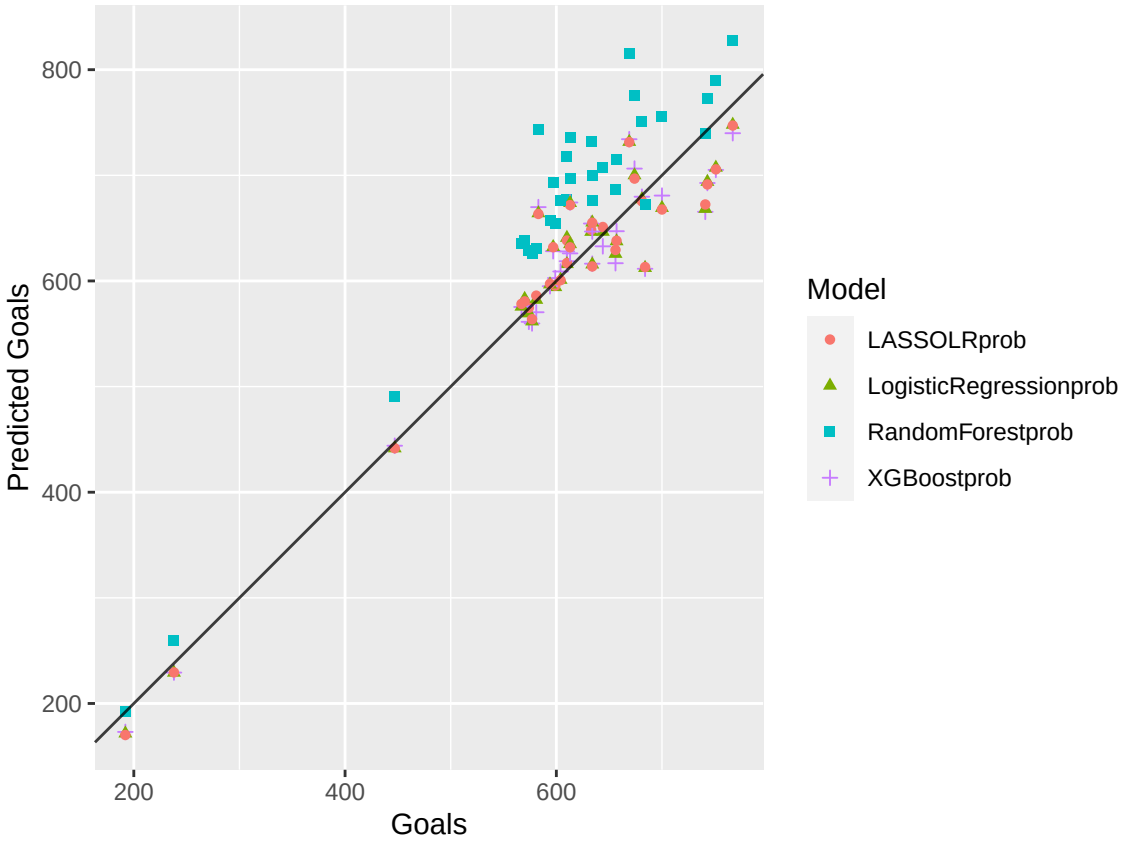

Figure 23: Predicted Goal vs Goals, Team