Supervised Conjecture Outcome Rating Estimator (SCORE)
ECE4424/CS4824 Machine Learning Final Report

Bill Norris
CMDA
Virginia Tech
Blacksburg, VA 24061
norris13@vt.edu

Alexander Ryan
Electrical Engineering
Virginia Tech
Blacksburg, VA 24061
alexryan@vt.edu

Jamahl Savage
Computer Engineering
Virginia Tech
Blacksburg, VA 24061
jamahl29@vt.edu

Ryan Stankiewicz
Computer Engineering
Virginia Tech
Blacksburg, VA 24061
ryanstan@vt.edu

Abstract

Our team used NFL historical play-by-play data to predict the number of points two teams will score against each other in a football game. Our models use a statistic called Elo rating to separately rank the offenses and defenses of all teams. In an attempt to accurately predict football games, we built four different machine learning models using these algorithms: Extreme Gradient Boost (XGBoost), Random Forests, Linear Regression, and K-Nearest Neighbor (KNN). All relevant code and documentation can be found in the project's Github repository.

1 Elo Rating

One feature which will be included in our models that does not appear in the public models created by nflfastR's (the initiative our project is based on) founders, Ben Baldwin and Mr. Caseb, is an Elo rating for each team's offense and defense. The Elo rating system, named after Arpad Elo, is a system in which each opponent in a game essentially wagers a portion of their rating based on the outcome of a game. If player A had a higher rating than player B and player A were to win the game, the increase in player A's skill rating would be almost identical to the amount that player B lost, and vice versa. One other useful feature is that the formula used to calculate the Elo ratings accounts for the fact that if the lower rated player (B) were to beat the higher rated player (A), the changes in their respective ratings should be greater than if the higher rated player were to win. This is because that outcome is less expected, so more weight should be given to its significance. One final feature of the Elo rating system which makes it extra useful for our project is that the mean rating should always remain roughly constant (in our case 1500) provided that there are not a lot of teams entering or leaving the league constantly. The only team to do so within the time period used to calculate our ratings is the Houston Texans, who were formed in 2002.

Elo ratings typically are used for games in which the outcome is restricted to a few possibilities, such as chess. In chess, you either win, lose, or draw. There does not exist a simple formula or rule to determine how badly one player beat the other. However, that is not the case in games like American football. If team A beats team B, and they both have identical ratings, the change in their respective

Preprint. Under review.

ratings should be larger if the outcome of the game was 35-0 as opposed to 35-28, because a Margin Of Victory (MOV) of 35 is much greater than 7.

To build our Elo model, we first researched how others had done it historically. Our research concluded on Nate Silver's 538 model of historic Elo rating [1] as well as Andrew Rennhack's explanation of how the Elo formula can be modified to account for MOV [2]. Utilizing these sources, we were able to develop a Python script that could assign ratings to each team's offense and defense at any given week between the 1999 and 2019 NFL seasons. To determine MOV, we found the total points scored by each offensive unit and points scored by each defensive unit, calculated their differences (HomeOffense - AwayDefense and AwayOffense - HomeDefense), and compared those numbers to the average difference between home and away teams for that given season.

In order to verify calculations, a Kernel Density plot was produced that displayed the distribution of Elo rankings for offense and defense:

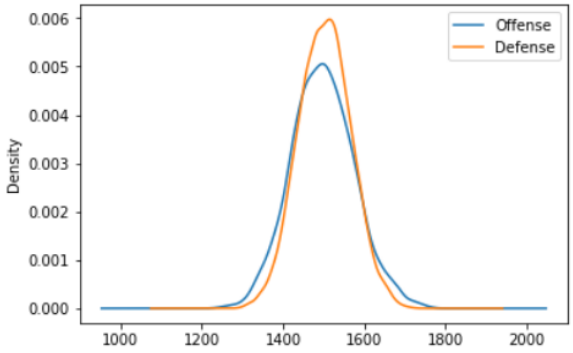

Figure 1: Elo rating density for both offenses and defenses

Figure 1: Elo rating density for both offenses and defenses

2 Algorithms

2.1 Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting is the machine learning algorithm used in the construction of the win probability model, expected points model. It builds a series of decision trees, much like random forests. Whereas random forests build a bunch of trees and either averages their results at the end, extreme gradient boosting only builds a single tree, which grows as it adds "weaker" trees to help fix the issues caused by the existing tree. As a result, one risk of using this algorithm is overfitting the training data.

The goal of the model was to predict the difference between how many more points a given offensive unit would score than a given defensive unit. Finding that difference for each team in a game would allow you to make better predictions on the outcome of the game. In order to make that estimation, the differential mentioned above was calculated and set that as a target, and the following features were passed to the algorithm:

- **home:** a binary representation of whether or not the offensive unit in question is the home team.
- **offElo:** the Elo rating for the offensive unit, as described in the above section dedicated to the Elo rating system.
- **defElo:** the Elo rating for the defensive unit, as described in the above section dedicated to the Elo rating system.
- **era:** an integer from 1-4 which represented what era the game took place in. While "era" isn't something that can explicitly be defined in this particular case, offensive production

has increased over the years due to a variety of reasons (more rules benefitting the offense, coaches making smarter decisions on offense, etc) and this field attempts to capture that as best as possible.

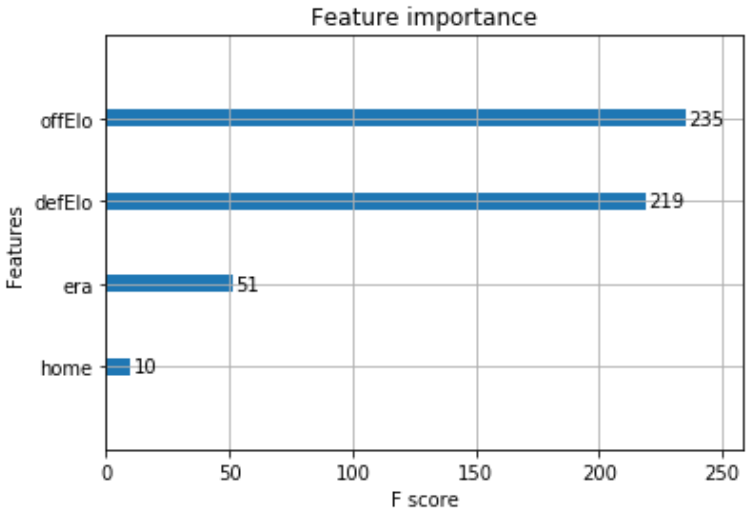

Figure 2: Feature Importance Plot

As seen in the above figure, some fields are more important than others. As expected, the Elo ratings are most predicative, with the offensive ranting being slightly more influential than the defensive.

Overall, as shown in the following figure, XGBoost has relatively low error metrics, which suggests that XGBoost is a reliable way to predict point differential in an NFL game.

```
Mean Absolute Error: 9.290301354317348
Mean Squared Error: 143.16043559025763
Root Mean Squared Error: 11.964967011666085
```

Figure 3: XGBoost Metrics

2.2 Random Forest

The Random Forests algorithm has been used by several past projects on predicting the win probability before each play of an NFL game. The technique is based on the principle of ensembling, in which multiple models are produced and from these models a consensus result is then predicted. This also reduces the chances of overfitting the data due to the diversity of many trees. The purpose behind using random forests for this problem is to be able to effectively predict the amount of points a team will score in a game if their opponent's score and both teams' defensive and offensive elos are given and used as features for training the model.

The Random Forests algorithm worked similar to the XGBoost algorithm. Key features from 1999 to 2019 are extracted and exported from the play by play database into a comma separated values (.csv) file by our data scraper. Before training the model, the code uses the home and away team's names, the season the game was played, and the week the game was played to get the offensive and defensive elos of the home and away teams for each week a game was played. The total number of points scored by the one of teams along with the calculated Elos are split into training and test sets used as the features for the Random Forests model to train and test on.The total points scored by the team whose points are not used in the features list are split into the training and test labels. During the training process, the Random Forests model utilizes 1000 decision trees to make predictions and then takes the average of those values as its output. It is also important to note that the model has its random state set for reproducible results. After the training is completed, the trained Random Forests model is run on the test set and the metrics are output to the console.

Due to how large the dataset being used for the training and testing of the model, only the first 50 actual and predicted scores for the team in question are displayed in Figure 4. Looking at Figure 4 may suggest that Random Forests is not a reliable model at times for predicting a team's total score based on theirs and the opponent team's elos. However, the metrics of the model shown in Figure 5 would show that error is relatively low based on all the predictions made by the model. A way to decrease the calculated error would be to increase the amount of decision trees used in the Random Forests model. It is important to note that this model already requires a large amount of computational resources, owing to the large number of decision trees joined together. Consequently, by adding more trees to the model, the complexity of random forests is increased and will require more time to train than other comparable models.

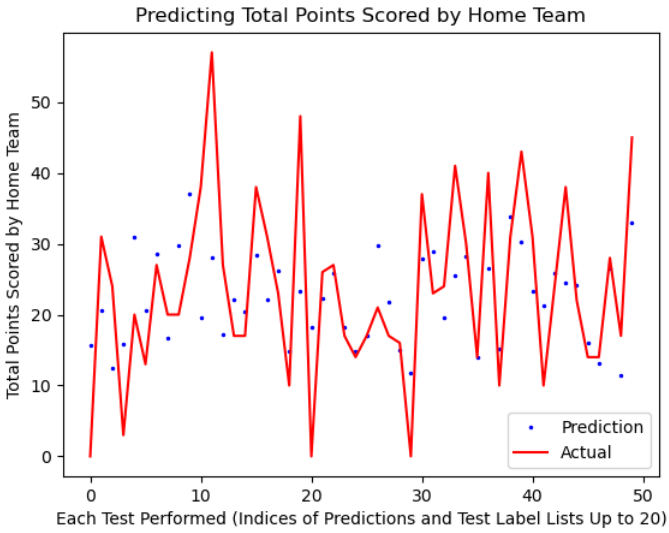

Figure 4: Random Forests Predictions Versus Actual Labels

```
Mean Absolute Error: 6.386546337817639
Mean Squared Error: 66.2144416464873
Root Mean Squared Error: 8.137225672579525
```

Figure 5: Random Forests Metrics

2.3 Linear Regression

The Linear Regression algorithm functioned by comparing the Elo rankings of team A versus team B in the selected label space and then drawing a regression of those scatter plots to form a basis against which to make predictions. The end user could choose which label space they wanted to see: home offensive points, away offensive points, home defensive points, and away defensive points. Thus, this accounts for regular scoring drives, such as touchdowns and field goals, as well as defensive scoring, such as fumble recoveries and interceptions that result in scores.A regression function was drawn for each potential label space, which can be seen Figure 6 below.

As seen in Figure 7 below, Linear Regression makes predictions with relatively low error. It is interesting to note that while on average the mean absolute error of offensive scoring is around 6%, the mean absolute error for defensive scoring is around 2%.

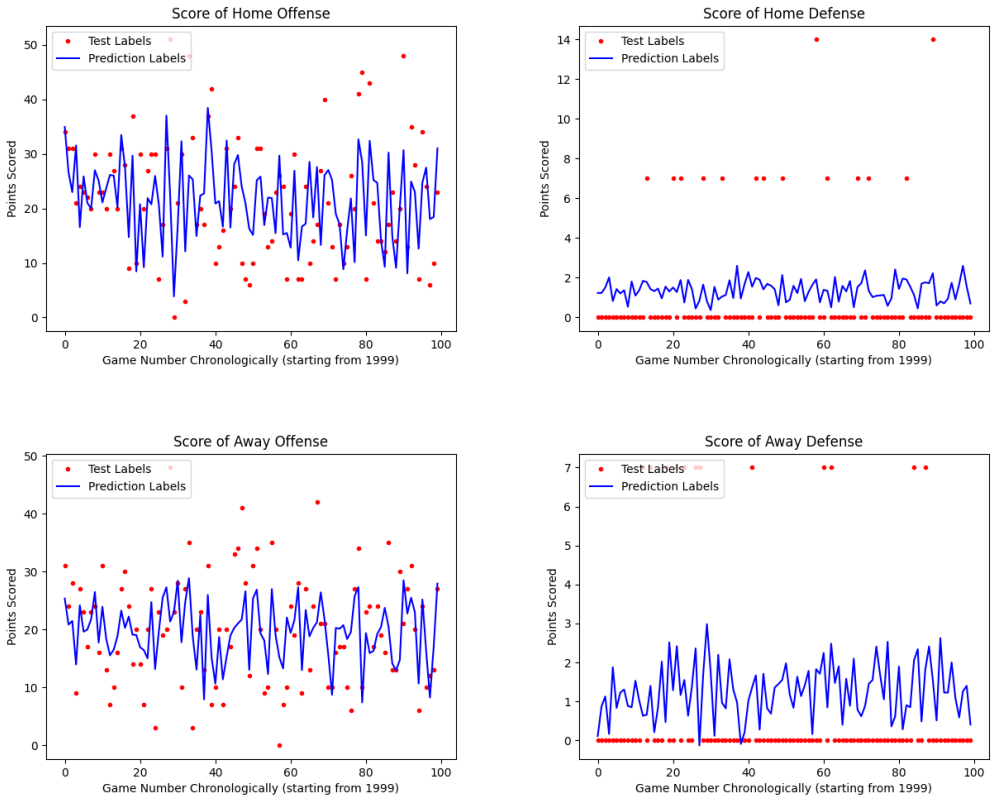

Figure 6: Linear Regressions Predictions Versus Actual Labels

```
Enter what data you want to see (HO for home offense, AO for away offense, HD for home defense, AD for away defense): HO
Home Offense
--------------------------------------------------------------------------------------------------------------------------
LR Mean Absolute Error: 6.161278816730447
LR Mean Squared Error: 62.12990691597503
LR Root Mean Squared Error: 7.882252654918835
```

Figure 7: Linear Regression Metrics

2.4 K-Nearest-Neighbor (KNN)

The KNN model was created with a different goal from the previous algorithms. Instead of labeling the final scores of games as the other algorithms do, the KNN algorithm attempts to classify whether a given play will result in a touchdown or not. The following features were used to define each play vector: home offensive Elo rating, home defensive Elo rating, away offensive Elo rating, away defensive Elo rating, season type, quarter seconds remaining, half seconds remaining, game seconds remaining, game half, drive, special play, quarter, down, yards to goal, yards to first down, play type, rush attempt, pass attempt, and touchdown probability. The labels used were binary values, 0 or 1, that indicated whether a given drive was a touchdown or not.

The data was split 80:20 to create training data and testing data subsets for the model. Initially, the model was trained to work with 5 nearest neighbors. This resulted in 96% accuracy over the test set. However, after further investigation, it was realized that the model was predicting touchdowns 0% of the time. Since there were so few plays that resulted in touchdowns in the dataset, choosing 5 nearest neighbors always resulted in predicting a play as not a touchdown. After forcing the algorithm to work with 1 nearest neighbor, the total accuracy dropped to 94%. However, the model was now accurately predicting 4% of touchdowns as opposed to 0% previously. Nevertheless, these are unfavorable results. In conclusion, even with the elo ratings added as features to our training and test sets, KNN

is not a reliable algorithm for this task. This is primarily due to the fact that there is a lack of data for plays that result in touchdowns.

3 Conclusions

Using the XGBoost algorithm as a baseline, since it is the original prediction method used by nflfastR, the Random Forests and Linear Regression algorithms produced slightly lower error percentages overall, which would result in better predictions. However, this is hard to compare directly as these XGBoost predicted different metrics than Random Forests or Linear Regression. Comparing Random Forests to Linear Regression would be more apt, but analysis of prediction error results in a minute difference slightly favoring Linear Regression. Overall, either model would produce satisfactory prediction results.

4 Contributions

4.1 Bill Norris

Developed Elo rating and XGBoost algorithms.

4.2 Alexander Ryan

Developed Linear Regression algorithm and formatted team midterm and final reports.

4.3 Jamahl Savage

Developed Random Forests algorithm.

4.4 Ryan Stankiewicz

Developed K-Nearest-Neighbor algorithm.

References

[1] Fischer-Baum, Reuben, and Nate Silver. “The Complete History Of The NFL.” FiveThirtyEight, 16 Sept. 2015, projects.fivethirtyeight.com/complete-history-of-the-nfl/.

[2] Rennhack, Andrew. “Elo Ratings Part 2 – Margin of Victory Adjustments.” andr3w321 RSS, 21 Apr. 2016, andr3w321.com/elo-ratings-part-2-margin-of-victory-adjustments/.