Predicting NFL Plays with Boosted-Algorithms

COMP 4442 – Fall 2021

Troy Jennings, Adi Bose, Romith Challa

1. Executive Summary

In this study, we explored tree-based ensemble methods to assess the performance and efficiency of these algorithms when applied to a classification problem. Specifically, we targeted three techniques with varying levels of complexities: a basic decision-tree, a general gradient-boosting algorithm, and an extreme-gradient boosting algorithm (XG-Boost). The objective for each method was to accurately predict NFL plays (*run*, *pass*, *field-goal*, *punt*) based on categorical variables such as down, as well as continuous variables such as time remaining, yardage information, score-differential, and time-outs remaining.

To ensure a fair comparison, each model was initially setup with default parameters, but was manually tuned for shared parameters. From our analysis, we observed that a simple decision tree was the easiest to execute but was the least accurate (63%). A generic boosting algorithm improved the classification accuracy (69%) but was slower to run (37 sec). Lastly, XG-Boost resulted in the highest accuracy (70%) while still retaining efficiency (3 sec). Among all three methods, we discovered that fourth-down was the most important feature used, while yardage information also showed impressive predictive power.

2. Data Source & Definitions

A comprehensive play-by-play NFL dataset was retrieved from Kaggle (Reference 1). The source data was originally scraped through the *nflscrapR* library and compiled by Carnegie Mellon University. Although the set contained over 250 attributes and 400,000 observations, encompassing a total of nine seasons, we were most interested in the attributes defined in Figure 1.

The dependent variable for this study was **play_type:** a multi-level categorical variable which describes the resulting type of play for each observation in the dataset. For the purposes of our analysis, the values for this variable take on “field_goal”, “punt”, “pass”, or “run”.

3. Exploratory Data Analysis

The extensive feature set was culled down to the six key attributes outlined in Figure 1 and a cursory look into the data helped us filter out edge-cases such as timeouts, penalties, and extra-point-

attempts. Focusing on the four most recently available regular seasons (2015-2018) further reduced the dataset to approximately 100,000 observations.

In order to develop a feature-engineering plan, we investigated the data even further by first generating a frequency plot of play type by down, as shown in Figure 2. This plot visually demonstrates some known characteristics of football: (a) the field-goal and punt play types occur primarily on 4th down, and (b) the distribution of play types is similar between passing and running plays, albeit with the total number of passes slightly outnumbering the runs. This first point supports a hypothesis that we will be able to classify field-goal and punt plays easier, and with higher accuracy. We then moved to exploring the distributions for potential predictors such as *ydstogo*, *yardline_100*, and *score_differential*, as shown in Figure 3 through Figure 5 in the **Appendix**.

From the "Distribution of Yards-Till-First Down" plot (Figure 3), we see a right-tailed distribution and most of the values at 10. This makes intuitive sense for NFL data since any play resulting in a 1st down will have 10 yards to go and downs with less than 10 yards to go increase as teams move the ball towards the goal line. The mode being such an outlier suggests that *ydstogo* is not a good candidate for binning into categories, so we maintain it as a numeric feature for our study.

For the "Distribution of Yards-Till-Endzone" (Figure 4), we see a left-tailed distribution, with high count values at 75 and 80. Again, this makes sense for our data, given domain knowledge, since kickoffs that resulted in a touchback used to be placed at the 20-yard line (I.e., 80 yards to go until the goal line) and after recent rule changes, are now placed at the 25-yard line (I.e., 75 yards to go until the goal line). As before, given the non-normal distribution, we elected to keep *ydstogo* as a numeric feature.

For the "Distribution of Score-Differential for all Plays" (Figure 5), the data does exhibit a normal distribution, however, we elected to not bin the data to maintain consistency across all our numeric variables of interest.

Finally, from the boxplots of each feature variable against play type (Figure 6) we noted that the *yardline_100* feature exhibited the most variation in data between play types. From this, we again hypothesized that the *yardline_100* feature would have higher importance in our classification models.

4. Research Question & Methodology

This project seeks to address the following research question *"Does an increase in sophistication of tree-based boosting algorithms impact the accuracy & efficiency of NFL play predictions?"*

To prepare for modeling, the data was separated by season, with observations from 2015 to 2017 encompassing the training set and 2018 season serving as the test set. This partitioning strategy made the

most sense in the context of a sports domain and prevented the need for further stratification that would have been required for random splitting. Then, a basic decision-tree model was created, using the default parameters, to generate a baseline tree and classification performance benchmark. A standard gradient-boosting algorithm and an extreme-gradient boosting algorithm (XG-Boost) models were also created to generate additional benchmarks to evaluate ensemble methods. Performance was measured across all three models based on (a) classification accuracy rate of predicting play type on the test set and (b) relative time to train the models.

5. Primary Method

In order to fully understand advanced boosting models, a basic understanding of a decision tree is imperative. A decision tree aims to classify an observation by starting at the root node, which consists of all possibilities, and then goes down the binary-split branches based on satisfying an attribute requirement at that branch. Eventually, this observation will reach a leaf node where it is classified. There are a diverse set of methods that determine how a branch can get split and it usually involves choosing a feature and corresponding value that best partitions the pool of possibilities into a "pure" set of conditions that can accurately classify an observation.

Using this as a basis, boosting models create multiple tree models, one after another in a sequential process, by learning from the misclassification errors of the previous model. Although there are several ways to "boost" a tree model, our study focused primarily on gradient-boosting methods which uses the gradient-descent algorithm to minimize the loss function at each iteration of the tree-building. In a multi-level classification problem such as this, the probabilities of the outcomes are first predicted and each subsequent model is fitted based on the gradients of the previous model. Various learning and validation parameters help control the model from over-fitting and this leads to a more reliable classifier than standard decision trees.

For even more extensive parameter tuning and efficient model training, a variation of gradient-boosting can be implemented. Extreme gradient-boosting models rely on second-order gradients and are innately designed to optimize the training process and offer more customizable parameters. While the underlying process remains the same as in standard gradient-boosting, XG-Boost offers in-built cross validation while training the model, along with L1 and L2 regularization to ensure we avoid over-fitting.

6. Implementation

In order to maintain consistency in the training and test datasets for each implementation, we standardized the data preparation process to meet the requirements across all models built. Categorical predictor variables were first converted into factor-types and then were dummy-vectorized. We chose not to bin any numeric variables into categories as per our exploratory analysis.

Decision-Tree:

Before building the boosted models, we first constructed a basic decision tree through the *tree* package. This function grows a tree through recursive binary partitioning, choosing splits based on either the deviances or the Gini index. As the purpose of this model was to serve as a baseline for the study, no further parameters were tuned. The constructed model was then applied to the test set to evaluate accuracy of play classification. Since there were many packages that offer similar functions to construct a tree, we also experimented with *RPart* to generate a cleaner visual of a decision tree (Figure 7).

Gradient-Boosting:

After creating a baseline tree model, the next step was to implement a tree-based ensemble boosting method. To that extent, we used *gbm* package to build a standard gradient-boosting classifier. This model was fitted onto the training dataset with a multinomial distribution, with fifty trees as recommended by the *gbm.perf* estimator. This model was also evaluated on the 2018 season data and the corresponding accuracy scores were noted through *confusionMatrix,* along with the runtime.

XG-Boost:

Next, we explored extreme gradient-boosting through *xgboost* library. To fulfill the formatting requirements of this model, the outcome factor was encoded into numeric form, starting with zero and the data was fed into the model in the form of a sparse matrix through the *xgb.DMatrix* function. Then each parameter was defined in a list, to be passed alongside into the model function.

For general parameter tuning, the **boosters** that are applicable for classification problems are “gbtree” and “gbdart”. As our goal was to compare tree-based models, we chose to use “gbtree”. The **nthread** parameter defaults to using the system’s maximum number of cores to train model as quickly as possible. For the learning task parameters, we chose “mlogloss” as **eval_metric** and “multi:softprob” for **objective** as this was a multiclass problem. Lastly, this model offers numerous tree-booster parameters to customize. We chose to iterate 50 times (**nrounds**) for the purpose of comparing the outcome with

standard boosting which was iterated 50 times. The learning rate (**eta**), which shrinks the feature weights at each round, was chosen to be 0.1 to avoid overfitting. The **max_depth** was tuned to 5 as the maximum depth of each tree to also keep it fairly consistent for comparing with gradient-boosting. As our specific training set seemed to not show signs of overfitting based on an initial run and plotting of the evaluation metric, we defaulted on other regularization parameters such as **lambda**, **alpha, and gamma.** Although a basic grid-search was also conducted through *expand.grid* functionality, to exhaustively run through possible parameters to determine best combination, we saw only marginal improvement and decided to keep original parameters for comparison.

After the model had been trained, we retrieved the logloss values at each iteration through the **evaluation_log** attribute of the model and plotted across the iterations to visually assess any divergence that would indicate over-fitting. Then *xgb.importance* functionality was used to retrieve and plot the most important features determined by the model based on information gain. Lastly, as the model outputted probability of each class over the iterations, rather than a single prediction, we wrangled the data and mutated numeric classifications back into factor-format to be able to compare the actual plays in the test set and output corresponding accuracies through *confusionMatrix* functionality. As with the other methods, the training step was timed through the built-in *Sys.time* tool.

7. Analyses

In order to address our research question, classification accuracy scores were captured and model-training runtimes were logged after each implementation. The baseline decision tree model classified play types with an accuracy of 62.6% in 1.1 seconds. The gradient-boosted model improved accuracy to 69.1% but trained slower at 36.8 seconds (using 50 iterations). Finally, the XG-Boost model classified plays with an accuracy of 70.3%, while taking only 2.9 seconds to train (50 iterations). These results can be visualized in Figure 11.

The observed improvement in classification accuracies, across the three models, are indicative of the benefits that come from using ensemble techniques, which combine multiple weak learners to improve predictive performance. As gradient-boosting fits each subsequent tree model based on the residuals of the previous, this bump in accuracy from using a single tree was expected. Although XG-Boost did not offer much improvement in accuracy compared to its standard counterpart, it only took a fraction of the time to train the model. Despite not taking full advantage of the parallel-computing capabilities, we still saw a significant bump in efficiency even from other optimizations such as maximum system core usage and the utilization of sparse matrices for memory reduction.

The evaluation metric for XG-Boost, "mlogloss", was plotted across all 50 iterations in Figure 8. This log-loss metric is the negative average log of corrected predicted probabilities, and the objective at each iteration is to lower this value. In Figure 8, we visualize this for our training model, indicated by the blue circles. By the end, we can observe only a marginal improvement for log-loss for each additional iteration, indicating an optimal stopping point to prevent over-fitting the data. The red curve representing the test-set log-loss values offers a visual indicator to ensure this, as a divergence would indicate that we are unable to generalize the model to the test set any longer.

Another key interpretation to be made is from Figure 9 which ranks the features that proved to be the most useful for XG-Boost to classify play types. Similar to the decision tree, we observe that 4th down was the predictor that offered the most "information gain", the purest branch split that enabled easy classification. Yardline to the endzone was the second most important feature, which interestingly coincides with being the attribute with highest variability across play types according to our exploratory analysis. This proved to be a more useful characteristic for the classification model.

In Figure 10, the accuracies for XG-Boost were further broken down for each play type, and we observed that field-goals and punts were easiest to predict, as also hypothesized due to their frequency in 4th down situations. More interestingly, however, Figure 12 shows a similar plot of individual play type accuracies, but for all the models instead. Here, we observe a stark drop in accuracy for decision tree in running plays. This can be attributed to the fact that our baseline tree model wasn't as robust for the more complicated classifications. While punts and field goals were easy to predict even with a single branch, passing and running plays needed more complex and deeper branches than we implemented, to lead to a more accurate leaf node. While tree-based methods are prone to over-fitting, insufficient tree splits can also lead to misclassifications due to impure branches defaulting to the more frequently appearing class in the dataset, rather than an informed decision.

8. Conclusions

In conclusion, our study showed that an increase in sophistication of tree-based boosting algorithms does impact the runtime and accuracy of classifying NFL plays. Gradient-boosting offered an improvement in accuracy, at the expense of efficiency, while XG-Boost optimized both. A future study that compares XG-Boost models with variations in hyper-parameter tuning can help shed light on which parameters are most important. Comparing boosted models to results of bagging techniques could provide insight into the advantages and pitfalls of various ensemble classification methods.

References

1. Horowitz, M. (2018, December 22). *Detailed NFL play-by-play data 2009-2018*. Kaggle. Retrieved November 19, 2021, from https://www.kaggle.com/maxhorowitz/nflplaybyplay2009to2016.
2. *Beginners tutorial on XGBoost and parameter tuning in R tutorials & notes: Machine learning*. HackerEarth. (n.d.). Retrieved November 19, 2021, from https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/.
3. Brownlee, J. (2020, August 14). *A gentle introduction to the gradient boosting algorithm for machine learning*. Machine Learning Mastery. Retrieved November 19, 2021, from https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/.
4. Comprehensive R Archive Network (CRAN). (2021, November 8). *Extreme gradient boosting [R package xgboost version 1.5.0.1]*. The Comprehensive R Archive Network. Retrieved November 19, 2021, from https://cran.r-project.org/web/packages/xgboost/.
5. *Decision trees in R: Examples & code in R for Regression & Classification*. DataCamp Community. (n.d.). Retrieved November 19, 2021, from https://www.datacamp.com/community/tutorials/decision-trees-R.
6. *Gradient boosting machine: Gradient boosting machine for data science*. Analytics Vidhya. (2021, March 25). Retrieved November 19, 2021, from https://www.analyticsvidhya.com/blog/2021/03/gradient-boosting-machine-for-data-scientists/.
7. Grover, P. (2019, August 1). *Gradient boosting from scratch*. Medium. Retrieved November 19, 2021, from https://blog.mlreview.com/gradient-boosting-from-scratch-1e317ae4587d.
8. Gupta, A. (2021, May 4). *XGBoost hyperparameters - explained*. Medium. Retrieved November 19, 2021, from https://amangupta16.medium.com/xgboost-hyperparameters-explained-bb6ce580501d.
9. Kurama, V. (2021, April 9). *Gradient boosting for classification*. Paperspace Blog. Retrieved November 19, 2021, from https://blog.paperspace.com/gradient-boosting-for-classification/.
10. Morde, V. (2019, April 8). *XGBoost algorithm: Long may she reign!* Medium. Retrieved November 19, 2021, from https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d.

Appendix

Predictor	Description	Values
yardline_100	Yards until the goal line	Integer from [0, 100]
half_seconds_remaining	Seconds remaining in the half	Integer from [0, 1800]
down	The current play down	Integer from [1, 4]
ydstogo	Yards until first down	Integer from [0, 100]
posteam_timeouts	Offensive team timeouts remaining	Integer from [0, 3]
score_differential	The difference in score between teams	Integer from (-∞, ∞)

Figure 1 – Description of predictor variables.

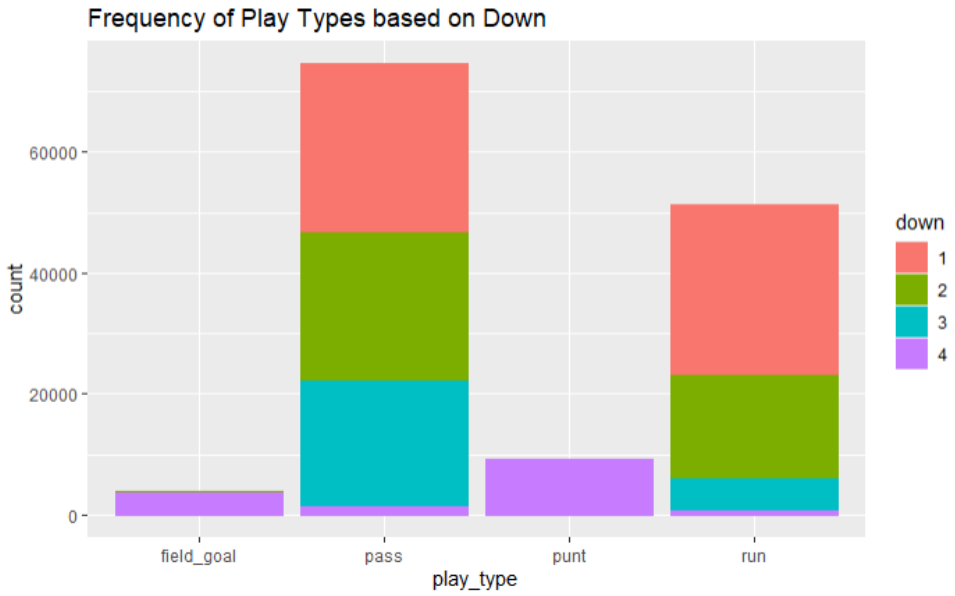

Figure 2 – Frequency plot of play type on specific down.

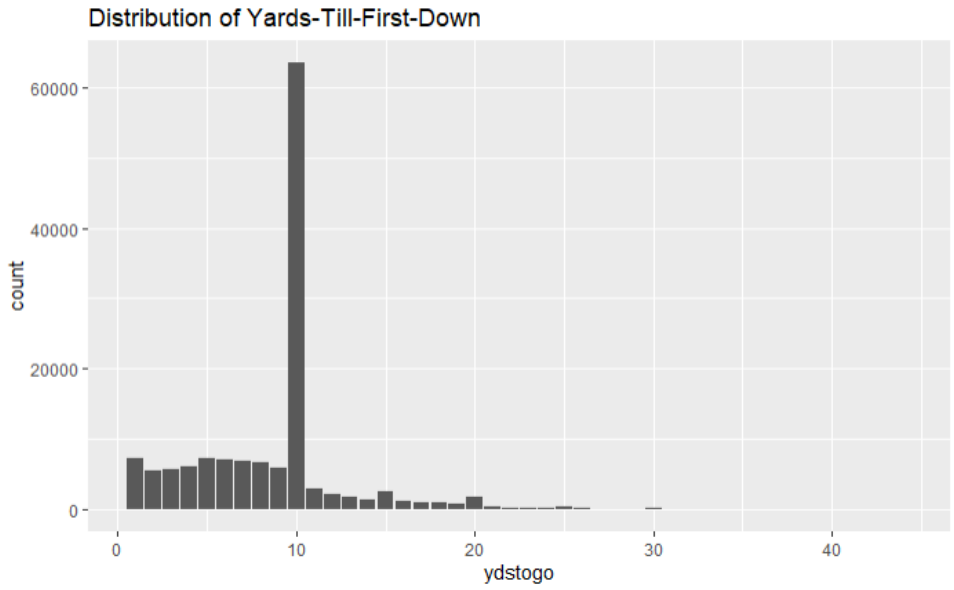

Figure 3 – Histogram of yards-till-first down for all plays.

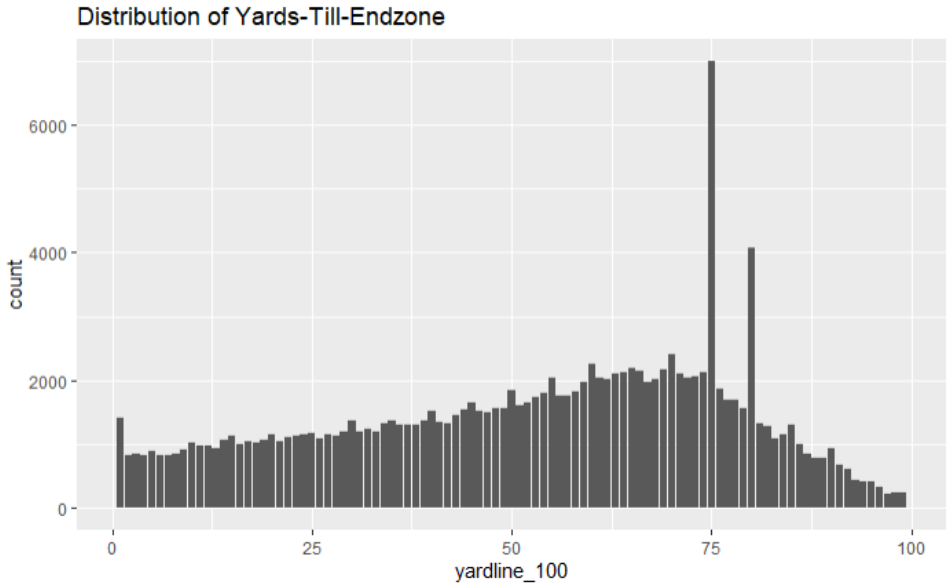

Figure 4 – Histogram of yards-till-endzone for all plays.

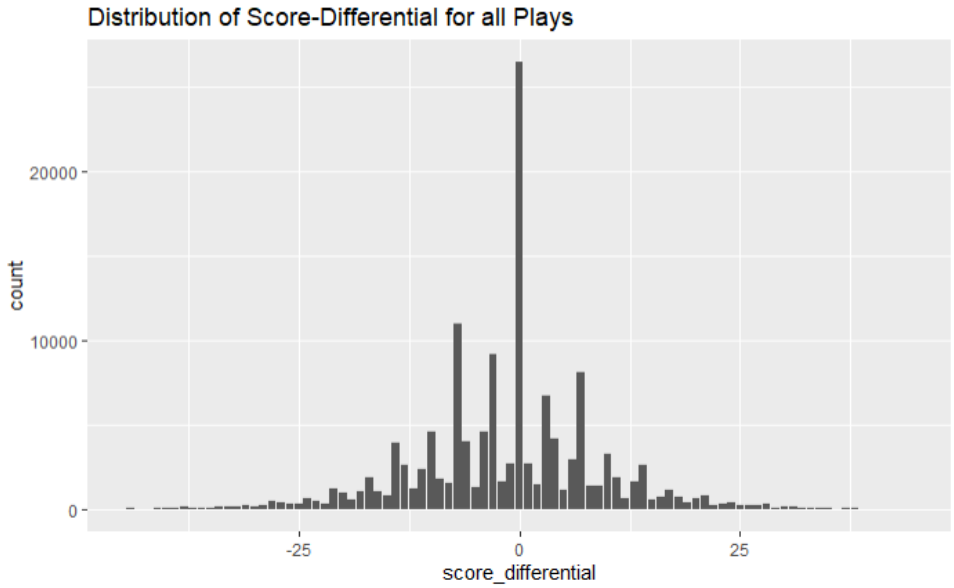

Figure 5 – Histogram of score-differential for all plays.

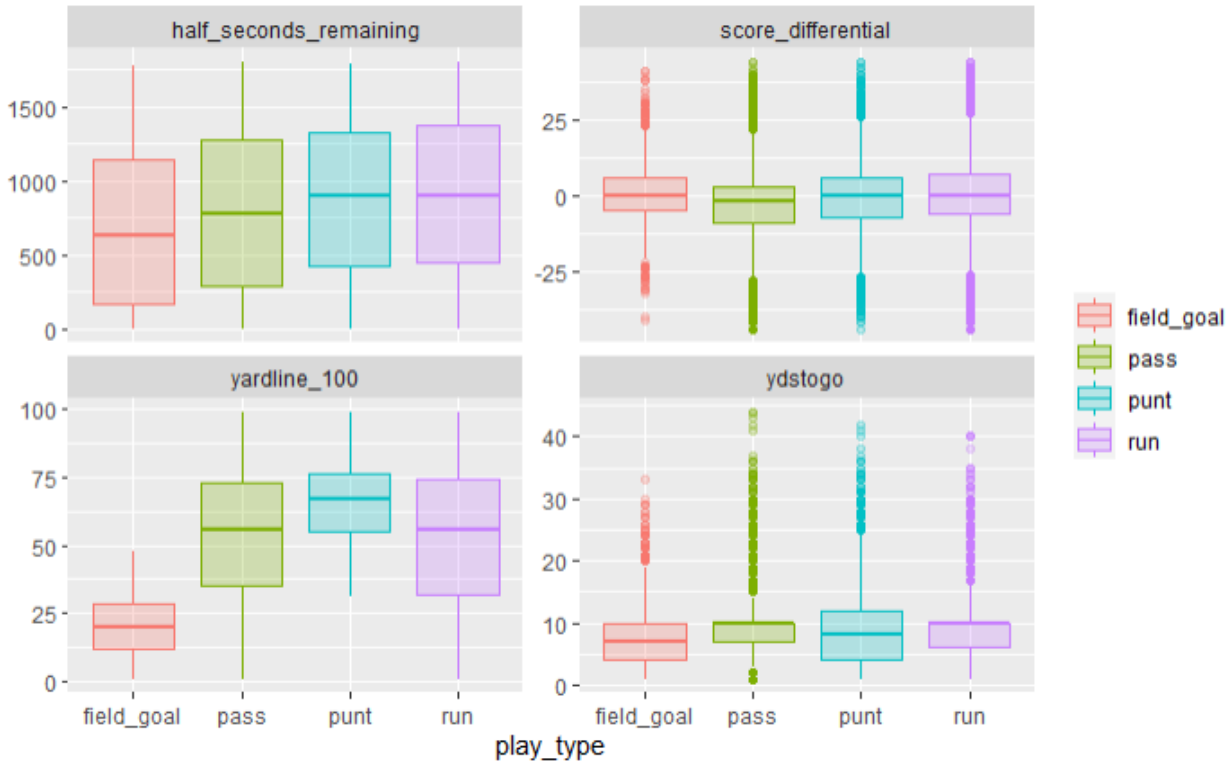

Figure 6 – Boxplots for primary predictors by play type.

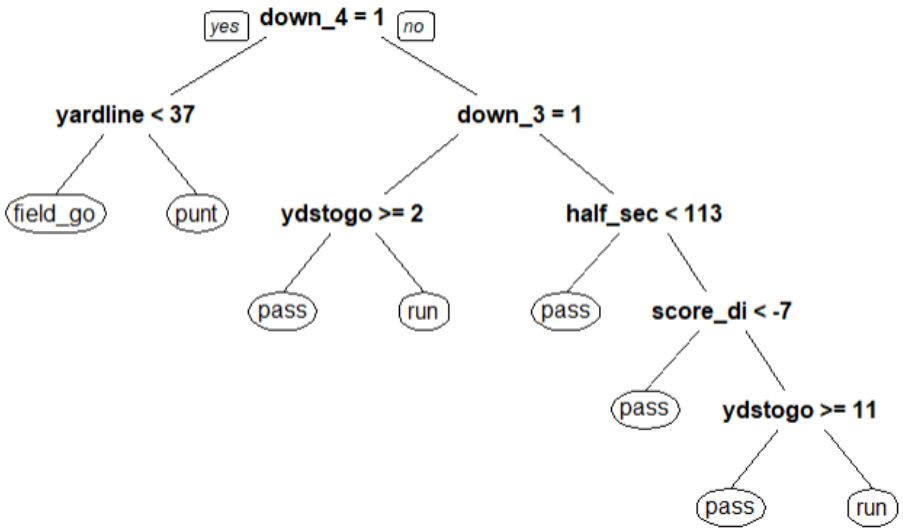

Figure 7 – Baseline decision tree.

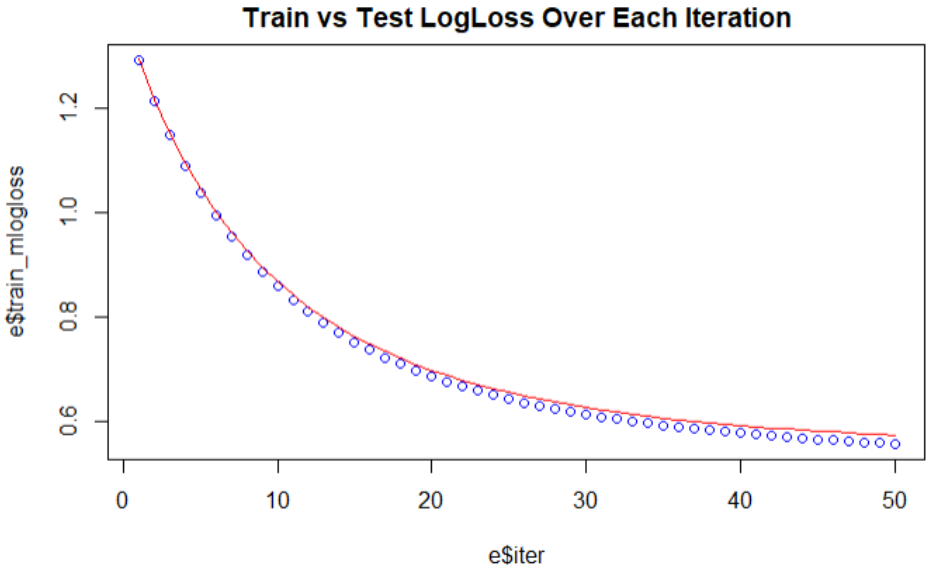

Figure 8 – Training vs. Test data logloss over increasing iterations.

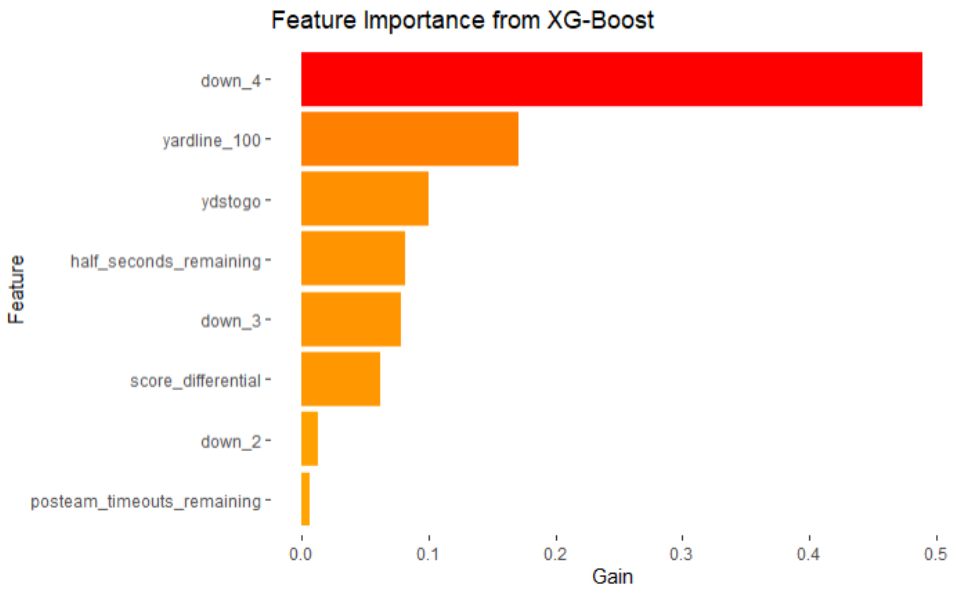

Figure 9 – Feature importance information gain for XGBoost.

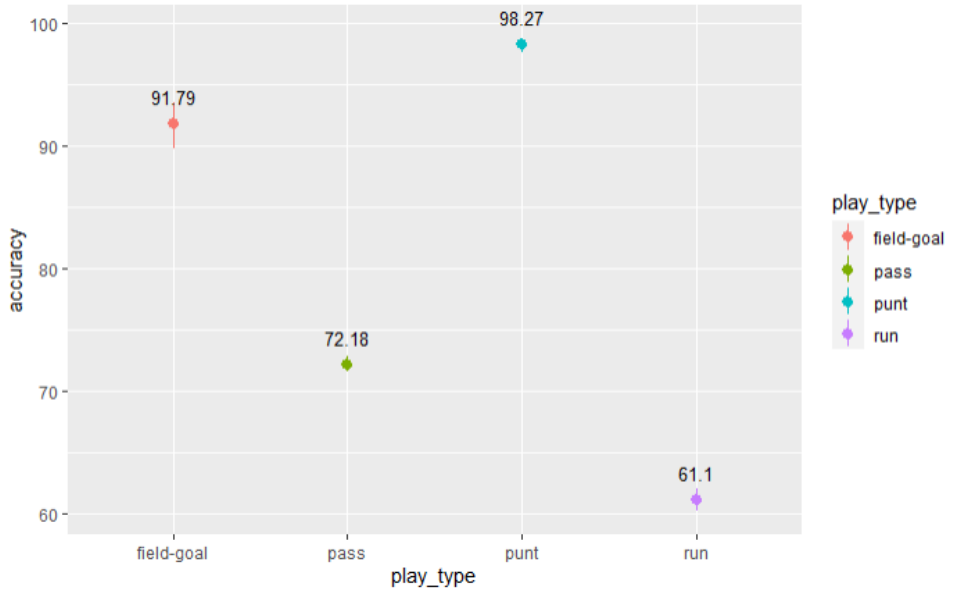

Figure 10 – XGBoost final model classification accuracies.

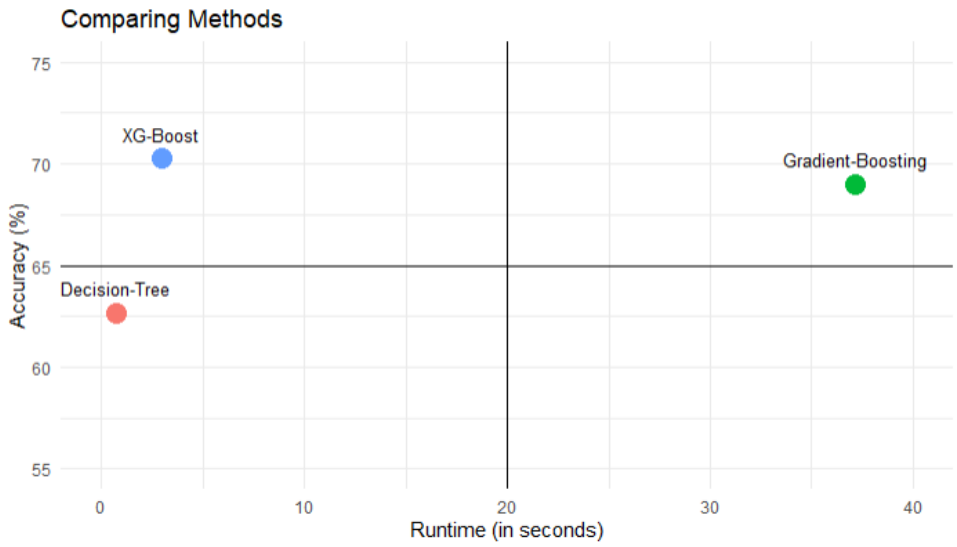

Figure 11 – Model comparison across accuracy and runtime.

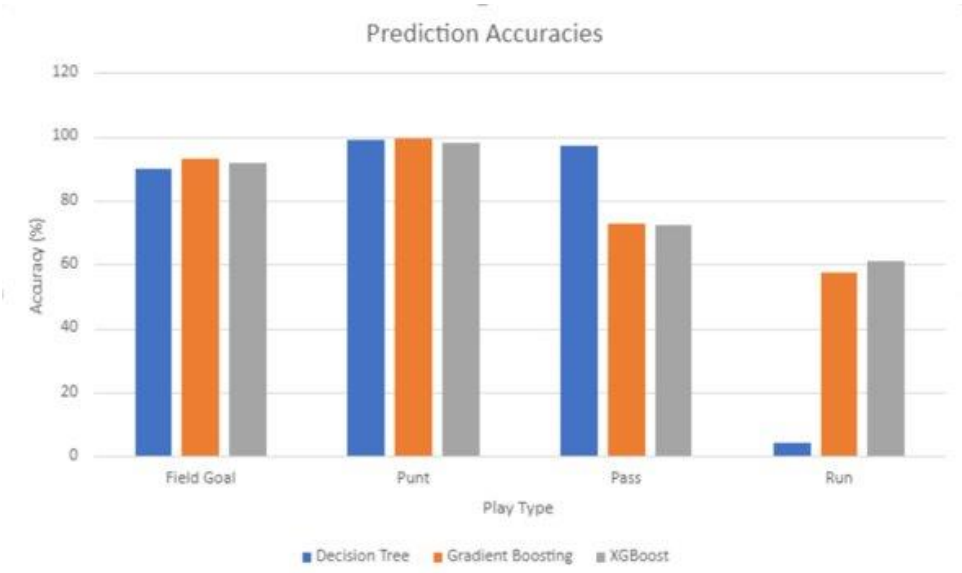

Figure 12 – Classification accuracy across play types for each model.