Capstone Project: Predicting likelihood of interception in NFL games

I. Definition

Project Overview

The National Football League (NFL) is a professional American football league that is one of the four major professional sports leagues in the United States, being the most popular among them [1] [2]. The size and importance of the league is clear when we look at its revenue: in 2017, it payed $8.2 billion to its teams [3].

In this project I attempt to build a classification model to predict the occurrence of one of the most crucial plays in the sport: the interception or pick. It happens when the team attacking attempts to pass the ball and it ends up being caught by someone from the defending team. Using play-by-play data of previous plays, plus some information that can be known about the play itself prior to it taking place, the model outputs a probability of an interception happening as a result of the play. To train and test the model, a public dataset from Kaggle is used, containing play-by-play data from 10 NFL seasons.

Problem Statement

An interception happens when a forward pass is caught by the defending team, immediately changing the ball possession to that team [4]. It is quite an impactful play. For instance, in the 2013 season, teams that intercepted the ball more times than their opponent won the game 80% of the time [5].

It can be speculated that the probability of a pass being intercepted is correlated with several factors that can be derived from play-by-play statistics, either cumulative (game score, time left to play, distance to endzone, completion percentage of the quarterback throwing the pass) or pertaining to the play itself (offensive formation, defensive formation, depth of the pass, number of current down).

This project attempts to build a classification model using the xgboost algorithm and taking in as features some of those variables. The data is obtained from a public dataset from Kaggle containing play-by-play statistics. As the target variable, we use a boolean that says whether or not an interception occurred as a result of the play. We train the model using some of the plays with their features, and then test it on other plays left unseen by the model. We then calculate how well the model is at predicting whether a play is going to result in an interception, using appropriate metrics.

Metrics

As the evaluation metrics, the project used true positive rate (or recall) and false discovery rate.

The true positive rate (or recall) shows the proportion of positive cases correctly identified as positive [6]. In other words, it is the ratio between true positives and the sum of true positives and false negatives:

$$\text{True Positive Rate} = TP / (TP + FN)$$

The false discovery rate is the proportion of cases incorrectly labeled as positive among all cases labeled as positive [7]. That is to say, it is the ratio between false positives and the sum of true positives and false positives:

$$\text{False Discovery Rate} = FP / (TP + FP)$$

A good model for the prediction of interceptions would have a high true positive rate and a low false discovery rate. That is, the model would identify as resulting in interceptions a high proportion of the plays that did result in interceptions, and, at the same time, the model would rarely say that a play will result in an interception when it actually does not.

These metrics are appropriate for classification problems, particularly those with high class imbalance, where the accuracy metric can give meaningless results. For instance, in a binary classification problem where the True class represents 2% of the total, a model that always outputs False as a prediction will have a very high accuracy of 98% while actually performing very poorly at identifying positive cases.

Moreover, the metrics are used by the benchmark model, presented later, which warranted choosing them over other options.

II. Analysis

Data Exploration

The dataset considered for this project is a CSV containing detailed play-by-play statistics for all games of the NFL seasons from 2009 to 2017, plus most of the games of the 2018 season. There are a total of 449.371 rows, each representing a play from 2526 games. Out of those plays, 4630 resulted in an interception, leading to an overall prevalence of 1%.

There are 9 play types, among which we are interested in 2: “pass” and “no_play”. The latter is relevant since some plays resulting in an interception have “no_play” as the play type. This reveals the question of how to properly filter the dataset to only have plays where there were passes, since the type “no_play” includes both passes and other events.

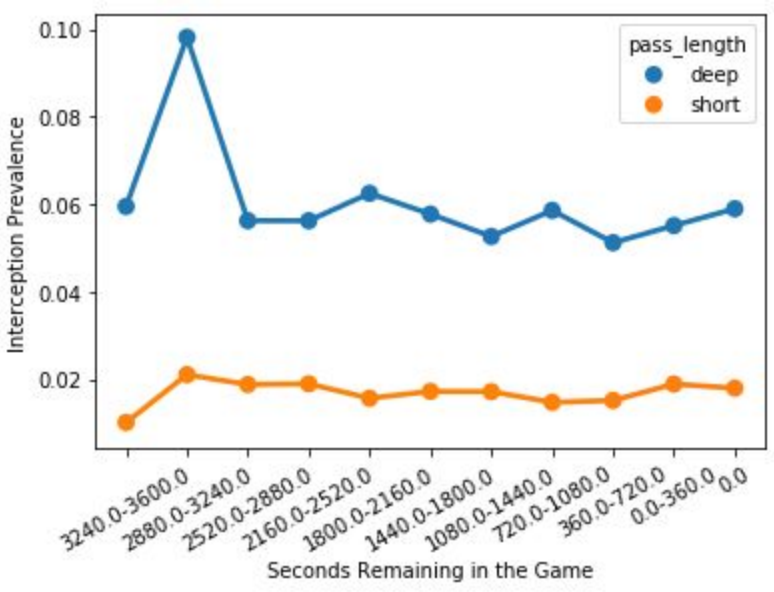
There are 11786 cases where there was a sack, which should not be included as they represent atypical plays that may interfere in the model. There are also rows that represent events that aren’t actually plays, such as the end of a quarter, and should not be included either.

Several of the features contained a significant amount of null values, which should be addressed in case we decide to include those variables. Also, there are numerical features on different scales, such as distance to goal or seconds remaining in the game, which is something that could skew our model and needs addressing.

The rows miss lagged features, that is, features regarding past rows (plays), such as whether the last play was an interception or how many yards were gained in the last play. Those will need to be created. There is also a “game_date” feature that could be useful if we consider only in which month, that is, at which stage of the season the game happened. That is another feature to be engineered. Several of the features were categorical and can’t be fed as they are to train a model, so they will have to be preprocessed. For a description of each of the 255 columns, refer to the README file. The features chosen will be detailed in a later section.

Exploratory Visualization

Intuitively, one of the most influential variables on interception prevalence could be the time remaining in the game, as less time left on the clock could push quarterbacks to try riskier passes when they are behind. We can speculate that another important feature is the length of the pass, as longer passes can lead to more yards gained but should incur in more risk of interception. We can see the interception prevalence for different levels of those variables in the graph below:



As we can see, the time remaining in the game doesn't seem to have a clear influence on the probability of passes being intercepted. The pass length, on the other hand, appears to have a strong influence, as was expected. The higher interception probability when throwing deep passes is quite possibly a strong factor that discourages teams from choosing them.

Algorithms and Techniques

The algorithm used for this project is the XGBoost classifier [8]. It is an implementation of gradient boosted decision trees designed for speed and performance, and it is indeed much faster than other implementations. A good testament to its performance is its prevalence among chosen algorithms of Kaggle competition winners [9].

Gradient boosted decision trees are an ensemble of decision trees aggregated via boosting, a technique where the predictors of the ensemble are made not independently, but sequentially. Each predictor learns from the mistakes of the predecessor. This is done by adjusting the weight of observations after each predictor is trained, giving higher weight to the ones that were incorrectly classified [10].

The model outputs a probability of pertaining to each class (either the pass will result in an interception or not). We can then choose a decision threshold and probabilities above that number will mean that the case will be assigned that class. Alternatively, the user of the model could simply look at the outputted probabilities and make decisions based on the number. In this project we do the former so we can arrive at the FDR and TPR metrics used to evaluate the model and compare it to the benchmark results.

The algorithm takes in several parameters when training. These are some of the most influential [11]:

- Number of estimators (number of trees to use as predictors)
- Learning rate (how quickly the model learns)
- Maximum depth (the maximum depth of a tree)

- Minimum child weight (prevents decision rules that apply only to a specific sample)
- Gamma (the minimum loss reduction required to split a node)
- Subsample (the fraction of observations to be randomly sampled for each tree)
- Columns sampled by tree (fraction of features to use in each tree)
- Alpha (a regularization term to handle high dimensionality)

Benchmark

Bock (2017) developed a Machine Learning model to predict whether a turnover (interception of fumble) would occur as a result of the next play. The results point in the direction of turnovers being predictable to an extent. The chosen algorithm was gradient boosting machines. The model was trained using play-by-play data obtained using nflscrapR, the same package used to generate the dataset for the present project. Data from complete seasons from 2009 through 2015 was included [12].

The study segmented the predictions into three categories of impending play: “run”, “pass” and “run or pass”. The results for the “pass” segment can be used as a benchmark model for this project, as we are going to consider only pass plays in order to predict whether they will result in an interception. The model obtained a false discovery rate of 8.3% and a true positive rate of 38.1%. Those metrics can be used as a benchmark.

III. Methodology

Data Preprocessing

The first step in data preprocessing was to engineer lagged features, that is, features that consist of characteristics of previous data points. Those were: the play type and amount of yards gained in the previous play, whether the last play incurred in a completed pass or if it resulted in an incomplete pass, and what was the location of the pass if one was attempted in the previous play. Another engineered feature was the month when the game took place, extracted from the game date, which shows how far into the season the game was.

After engineering the lagged features we were free to filter for the plays that were relevant to our problem. As discussed, some interceptions happened in plays tagged with the ‘no_play’ play type, which means that filtering for plays with ‘pass’ as their play type wouldn’t include all relevant cases. The correct column to use was ‘pass_attempt’, which showed whether there was a pass attempt at the play. Rows where ‘pass_attempt’ equals 0 were removed. From those, plays that ended in a sack were also excluded, as they aren’t representative of a typical pass play.

Features with the potential to be important were pre-selected. Then, null values for those features were handled, with the appropriate approach for each case. Some null rows were

deleted, some were assigned to a new category, some were addressed with imputation of values according to the proportion for the existing categories, and others were bugs that were corrected, according to each column's case.

Numeric features were then standardized by removing the mean and scaling to unit variance. Categorical features were prepared via one-hot encoding, resulting in one binary column for each category in the original column. Additionally, the algorithm BoostARoota, which is designed specifically for feature selection within xgboost [13], was applied during training.

Implementation

A function was created to include all steps involved in training and testing the model. It was based on the template presented in the Analytics Vidhya website [14], and was modified to fit the project's case and include extra steps.

The first step in the modeling pipeline was to split the dataset in training and testing subsets. The split was stratified to make sure each subset would have a representative proportion of positive and negative labels. The training set would then be used for the subsequent steps of model training and also for later refinement through grid-search cross-validation, while the testing set was used to assess out-of-sample model performance.

In order to address the heavy imbalance between labels (close to 98% of labels were negative), the dataset was then resampled. More about the process of arriving at a final method of resampling as well as a comparison with the performance when using the original training sample can be found at the Refinement section. Only the training set was resampled.

The next step was to perform feature selection with the library BoostARoota. It makes use of duplicated, shuffled features known as "shadow features" and several runs of xgboost on the dataset (the training set) to arrive at a final feature set [15].

Cross-validation was then used to arrive at an optimal number of estimators to train the model and to set as a fixed parameter when later performing grid-search cross-validation for other parameters. A metric was developed to arrive at a better FDR-TPR trade-off instead of the typical AUC metric, which focuses on the relationship between FPR and TPR. However, later it was concluded that the area under the curve of precision and recall ("aucpr") would accomplish the same thing, as the FDR is simply equal to 1 - precision.

The model was then fit using training data and predicted labels were derived using testing data (with the default decision threshold of 50%). Several functions were constructed to allow for appropriate model evaluation. One was coded to produce a confusion matrix with labels and improved readability. Another was written to get a dataframe with predicted probabilities instead of labels, along with the real labels, which was fed to another function that returned the FDR and TPR metrics for a specified list of decision thresholds.

The model fit function also displayed accuracy and area under the curve of FPR and TPR as additional metrics, and produced a graph with final feature importances in the model. A function was also defined to log into a text file the model parameters and the figures returned by the other functions involved.

Refinement

Interceptions are very rare. When considering plays with pass attempts from the dataset, only 2.64% resulted in an interception. This represents a highly imbalanced dataset, one that can lead to fitting models that will almost always predict negative cases. It also means that the algorithm will have comparatively very few lines to learn about the positive cases (the ones when an interception happens).

The initial results, obtained by fitting a xgboost classifier on that highly imbalanced training set were as expected. Only 3 out of 35036 samples were labeled as positive with the default decision threshold of 50%. Using a threshold of 80%, both the FDR and the TPR were equal to 0%.

One of the techniques used to handle cases of imbalanced data is to rebalance the dataset. For this purpose, the project made use of the imbalanced-learn library [16]. It offers two major types of rebalancing: undersampling and oversampling. In undersampling, the majority class is reduced in order to have a number of samples closer to that of the minority class. In oversampling, additional samples are engineered using the examples from the minority class in order to arrive at a similar size to that of the majority class.

Attempts were made with different methods for both types of rebalancing. More complex methods tried at first did not offer satisfactory improvements over using the dataset without rebalancing. For instance, when using the SMOTE oversampling technique, 15 of 35036 samples were marked as interceptions with a 50% threshold, and with a threshold of 80% the FDR was 33% and the TPR was 0%.

The methods that offered the best results were the simplest: random undersampling and oversampling. These methods pick samples at random from the majority class to remove them, in the case of undersampling, and from the minority class to duplicate them, in the case of oversampling. Since results were similar for the two types of random rebalancing, but the oversampling type incurred in much larger training times, especially when considering the prospect of doing grid-search cross-validation on the resulting dataset, random undersampling was picked. When using the undersampled dataset, the model predicted that 9799 of the 35036 samples would be interceptions under a 50% decision threshold, and using a 80% threshold the FDR and TPR were of 83.8% and 22.2%, respectively.

To build upon those results, the BoostARoota xgboost feature selection algorithm was applied. This resulted in a reduction of the amount of features used, from 59 features before feature selection, to 51 features afterwards. An interception was predicted for 9931 of the 35036 samples under a threshold of 50%, while using a decision threshold of 80% led to an FDR of 85.8% and a TPR of 26.1%.

To improve those metrics, grid-search cross-validation was used. Several parameters had their values chosen from a range, training the model on part of the training set, with the rest serving to validate the performance of the model with those parameters. The cross-validation was 5-fold, and the parameters tuned were maximum depth, minimum child weight, gamma , subsample, columns sampled by tree and alpha. The best selection of parameters was decided

according to which led to a higher precision. This metric was chosen with the goal of reducing FDR.

When using the parameters of the best resulting model to redo all of the steps involved in training and then predict against the testing set, the metrics were not better: 8622 positive labels out of 35036 with the 50% threshold and a FDR and TPR of 83.8% and 17.3% with a 80% decision threshold. However, due to the fact that the model is validated many more times on unseen data, it can be expected to be more robust, to generalize better, and produce a better average result when applied to new data. For that reason, that is the final model.

Another attempt at improvement was using other modelling options such as Balanced Bagging, Random Forest, Easy Ensemble, RUS Boost, and using bagging on top of xgboost. They did not produce improved results. A complete list of model runs can be found on the log file. For convenience, the results described in this section were put at the beginning of the file along with a title explaining which iteration they refer to.

IV. Results

Model Evaluation and Validation

The parameters selected by the grid-search cross-validation process were set as the final ones. For a final list of parameters of the model, please refer to the log file under “Final results, after doing grid-search cross-validation”. Here is a summary of the final characteristics of the model:

- It consisted of a xgboost classifier
- The objective was logistic regression for binary classification
- 60 decision trees were used
- The learning rate was of 10%
- The trees had a maximum depth of 3
- The minimum child weight was of 1
- The gamma was 0.1
- 70% of observations were randomly sampled for each tree
- 90% of the features were used in each tree
- The regularization alpha was 1

The first metric evaluating the final model was derived from grid-search cross-validation, which led to a model with a precision of 72.6%. However, the most appropriate way to evaluate the model is to use data not included in that process.

To better evaluate the model, a stratified subsample of 20% of the dataset was held out from the cross-validation process. After that was done, the resulting model was then used to predict against that subsample. This test led to the final metrics described on the previous section. Those numbers are definitely better than naive predictions (based on random chance), considering how rare the positive cases are. However, it is unlikely that a model with that

performance can have significant real-world use, as the FDR is too high for a given level of TPR (and vice versa).

Justification

The benchmark study reported a model for the Pass play segment that can be compared to this project's final model. The final metrics of the benchmark model were a FDR of 8.3% and a TPR of 38.1%. In turn, the ones regarding this project's final model were 83.7% and 17.2%. That means that both metrics were inferior to the ones of the benchmark model, especially the FDR which was ten times worse.

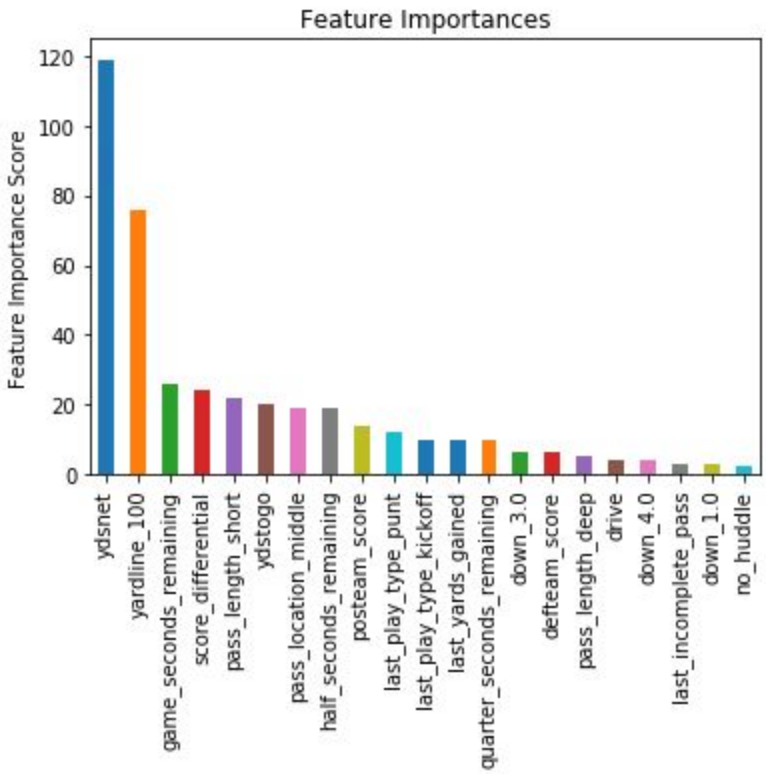
To verify what could be responsible for this difference in performance, thorough review of the modelling steps was done, as well as attempts at different approaches to engineering features and filtering the dataset before training. Other features not included in the benchmark model were added and engineered, as well as different evaluation metrics for cross-validation tried. Feature scaling was included, for instance, even though theoretically it is believed to be unlikely to improve xgboost classification models when they are based on decision trees [17], because it had been implemented in the benchmark study. However, none of those attempts led to improvements in score, and some led even to a vastly inferior performance.

In summary, this project's model did not improve upon the benchmark model and it could have only limited practical use. A predicted probability of interception higher than 80% would mean that there is approximately a 16% chance of there being an interception, which *is* much higher than the base probability of close to 2%, so the model could still serve as an auxiliary, limited information source for coaches, sports bettors, or anyone else interested in anticipating interceptions.

V. Conclusion

Free-Form Visualization

The graph below shows the ranking of features included in the final model according to their feature importance score, which in the case of xgboost's get_fscore method represents the number of times each feature was used for a split.



Most of the top features in importance were numeric features. The 'ydsnet' feature represents how many yards were gained/lost on the drive so far; 'yardline_100' shows how many yards are left to cover to reach the goal; 'game_seconds_remaining' is the time remaining in the game; 'score_differential' is the difference in score between the team in possession of the ball and the defending team. Completing the top 5 features is a feature resulting from one-hot encoding the categorical variable 'pass_length', and it shows whether the attempted pass was short.

Reflection

To summarize, these were the steps included in building the model:

- The problem to solve was defined by browsing public datasets in Kaggle
- The relevant dataset was obtained in CSV format
- A benchmark model was found to evaluate results
- The dataset was loaded and preprocessed
- Classifiers were trained on the resulting dataset iteratively, adding steps to improve performance (rebalancing, feature selection, cross-validation)
- The best model was then evaluated, with comparison to the benchmark model using the same metrics (FDR and TPR at a chosen decision threshold)

At first, my intention was to follow the steps done in the benchmark study, achieving a comparable performance and then working on improving upon it. However, it proved very

challenging to determine from the paper exactly what the steps were, with a lot of ambiguity in some descriptions and definitions.

The most interesting aspect of working on the project was to see how influential rebalancing can be when dealing with highly imbalanced datasets. Results would vary greatly not only when comparing performance using the imbalanced dataset versus with an imbalanced one, but also when comparing different techniques of rebalancing. At some point, I had by mistake rebalanced both the training and testing sets, resulting in misleading good scores which in hindsight are clearly wrong. It is the step where switching choices had the most significant impact in performance.

Improvement

A potential way to improve the project's model would be to change the definition of some of the lagged features. For instance, the last_pass_complete and last_pass_incomplete features were the most important ones in the benchmark model, but didn't get nearly as much importance in the present model. In the case of the project, the feature showed whether the last play resulted in a complete pass and in an incomplete pass, respectively, but perhaps the results could be different if more information was taken into account and it instead showed whether the last pass by the team currently in possession of the ball was a complete one or an incomplete one. This, in fact, may have been the definition employed in the benchmark model.

The benchmark model also applied an algorithm that involved bagging on top of gradient boosted machines, and choosing an optimal decision threshold to minimize FDR before training the model again using that threshold. I could not figure out how to reproduce that algorithm, and while my attempts at bagging or optimizing FDR did not produce significant improvements in scoring, those are areas that may offer potential for bettering performance.

Lastly, it is very possible that a much more robust model can be made using the NFL's "Next Gen Stats", which capture "real time location data, speed and acceleration for every player, every play on every inch of the field" [18]. That information goes well beyond typical play-by-play data, which aims simply to inform the audience about how the game is unfolding. Those more detailed stats, while in the present not readily available programmatically to the public, offer a lot of potential for feature engineering to capture a bigger chunk of the explanation regarding whether or not a pass will result in an interception.

References

[1] https://en.wikipedia.org/wiki/National_Football_League
[2] Jozsa, Frank P. (2004). Sports Capitalism: The Foreign Business of American Professional Leagues. Ashgate Publishing. p. 270. ISBN 978-0-7546-4185-8. Since 1922, [the NFL] has been the top professional sports league in the world with respect to American football
[3] Harris, Nick (January 31, 2010). "Elite clubs on Uefa gravy train as Super Bowl knocked off perch". The Independent. London. Retrieved November 28, 2012.
[4] https://operations.nfl.com/the-rules/2018-nfl-rulebook/
[5] https://thepowerrank.com/2014/01/31/how-to-predict-interceptions-in-the-nfl-backed-by-surprising-science/
[6] https://en.wikipedia.org/wiki/Precision_and_recall#Recall
[7] https://stats.stackexchange.com/questions/336455/fpr-false-positive-rate-vs-fdr-false-discovery-rate
[8] https://github.com/dmlc/xgboost
[9] https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/
[10] https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d
[11] https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
[12] Bock, J.: Empirical Prediction of Turnovers in NFL Football (2017)
[13] https://github.com/udacity/machine-learning/blob/master/projects/capstone/capstone_report_template.md
[14] https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
[15] https://github.com/chasedehan/BoostARoota
[16] https://pypi.org/project/imblearn/
[17] https://github.com/dmlc/xgboost/issues/2621
[18] https://nextgenstats.nfl.com/