Predicting Yards Gained on NFL Handoff

Travis McGuire
Department of Electrical Engineering
Stanford University
travis.mcguire@stanford.edu

Bradley Barnhart
Department of Electrical Engineering
Stanford University
bbarnhar@stanford.edu

Dahlia Radif
ICME
Stanford University
dradif@stanford.edu

***Abstract*—This paper demonstrates various approaches to predicting a probability distribution over possible yards gained in a NFL handoff using machine learning (ML) algorithms. Features are cleaned and engineered, before performing forward feature selection to find acceleration (`A`) and number of defenders in the box (`DefendersInTheBox`) from validation. XGBoost is found to have the best performance (as measured by a form of squared error between the true and predicted probability distribution). While the competition is not complete as of this writing, we are excited to see our final results as well as what insights led the top hundred teams to have significant improvements over the prior thousand entrants.**

***Index Terms*—Machine Learning, Statistics, Kaggle, NFL**

Code available github.com/travismcguire/cs229nflbigdata.

I. Introduction

The NFL is a huge business, with an estimated $8.1B in yearly revenue. Around 15 million people tune in for each of the 256 regular season games, with that growing to almost 100 million people for last year's Super Bowl. An important problem, tackled by coaches, analysts, and fans, is how many yards to expect when making a handoff (one of two offensive plays in the NFL). We hope to answer that question using ML. People care so much about this problem because it is an essential question, whose outcome will determine the winner of the game. We address this problem using ML, where the algorithm's input is a large number of features describing the state of the game before the play and the output is a probability distribution over gaining different amounts of yards. Several ML algorithms are evaluated for the problem, including a maximum likelihood estimation approach, logistic regression, and tree-based methods. The results are to follow.

II. Related Work

With Kaggle competitions, many people are attempting to solve the same problem and generously publish their work. Here, we note a few that we found particularly helpful or interesting notebooks for this challenge. Modeling aside, one notebook has great details for using the data module and making submissions [9]. We also found several great notebooks for exploratory data analysis which provided insights for both data cleaning and visualization [4], [8], [10]. Two of the most highly up-voted notebooks use LightGBM [5], [6]. LightGBM is a tree-based learning algorithm with gradient boosting. The two other most highly up-voted notebooks use neural networks [2], [7]. Nikita's neural network notebook is the best scoring notebook publicly available in the Kaggle competition. This past work has helped inform our data cleaning and feature selection whilst providing insight into the performance of models being used by others. Kaggle competition aside, typically this is the kind of things debated by NFL analysts and coaches. What makes a play a success or failure is something often uncertain, and at times lacking a clear answer.

III. Data and Features

As with any data-driven research project, examining and understanding the data is key to success. Feature selection is particularly important; this dataset contains a large number of features, which are considered carefully and condensed in order to prevent overfitting.

A. Data

The data is from Next Gen Stats and hosted on Kaggle, containing 23k plays (samples) with 49 features from the 2017/2018 NFL season [1]. The data is originally comprised of 500k data points representing each of the 22 players on the field, but we downsample to keep only the point corresponding to the rusher. The rationale is the features for that particular player are likely to have the most predictive power and relevance to yards gained. Given more time, considering specific details about the other 21 players might be worth investigating. We are predict a distribution over the `Yards` variable, i.e. the yardage gained on a particular play.

B. Training and Validation

Of the 23k samples, there are 16k samples in the train set and 7k in the validation set. This corresponds to the data falling around 70% training and 30% validation. The test set is brand new data, being collected after the submission deadline during the end of the 2019 NFL season. While it will be made available after the season ends, the submitted model will not be able to take it into consideration and our model's final performance will not be known until the end of December.

C. Preprocessing

1) Missing values: Several steps were taken to preprocess the data. Some data points contain missing values (e.g. `FieldPosition`), which we remedy wherever possible (e.g. by setting to the possession team value). As another example, the `DefendersInTheBox` feature contains missing values as well, which are set to the feature's median value (7).

2) Rare values: `DefendersInTheBox` also contains extremely rare values ($\leq$ 10 samples) where only one or two defenders are in the box. We bump these up to the next closest value, three, since it is an ordinal feature. The reason is we don't want an extremely rare situation to have too much influence in the model.

3) Inconsistent values: `PossessionTeam` is another example of a feature which required cleaning, but instead because of inconsistency in the feature itself. For several teams, there were multiple categorical values used representing the same team (e.g. the Baltimore Ravens were both 'BLT' and 'BAL'). This is a problem, because this would not let the model leverage the knowledge that 'BLT' and 'BAL' are actually the same team.

D. Features

An important aspect of this project is to select the most relevant features to use for prediction. As discussed above, the dataset contains 49 different features, many of which may be spurious for predicting yards gained. These are both continuous and categorical; for example, `PlayID` is the unique play identifier, `A` is a continuous variable encoding acceleration in $\frac{yards}{s^2}$, and `PlayerHeight` is the player's height in $ft-in$. The full feature list can be found on the competition website; Figure 1 includes a few examples of both given and engineered features.

We engineered several features (e.g. `Carries`, `RusherAvgYards`, `DefenseAvgYards`) which were not selected during feature selection. The original dataset contains a variable, `YardLine`, representing the yard line of scrimmage. Using this, and information about the location of play and ball possession, we create a new variable `YardsRemaining`, which we use in our model training to eradicate infeasible probabilities for yards gained (see section IV). Another engineered feature is `Carries`, which is built by counting the number of times a player's id appears as a rusher id in the training data.

	PlayId	A	Distance	PlayerWeight	DefendersInTheBox	RusherMeanYards	YardsRemaining
0	20181014080633	3.27	3	220	7.0	3.721429	16.0
1	20170917101220	3.28	10	199	6.0	2.953488	43.0
2	20170910053220	0.93	9	223	8.0	3.337748	37.0
3	20181029002881	2.36	10	210	8.0	3.444089	35.0
4	20180930021037	1.39	10	228	6.0	4.187166	64.0
5	20171123001590	3.12	10	230	6.0	4.057471	52.0
6	20181028023641	3.58	8	224	6.0	4.169972	85.0

Fig. 1. Given and Engineered Features

E. Feature Selection

We reference Tom Bresee's notebook [10] which depicts a useful categorization of the features into Game and Field Information, Player Information, Running Play Information, and Play Formation Information. Within each category, we select what we believe to be the most relevant features to avoid overfitting. For example, `Weather` and `Humidity` might be less important (and highly correlated) features than `DefendersInTheBox` or `Down`. This is further justified by some visualizations of individual features with yards gained. For example, looking at the work done in [4], there is a relationship between yards and the offense formation, whilst weight and height are very weakly correlated with median rushing yards.

Forward feature selection is used to determine which of the few 49 features are the best predictors. The lowest cost model selects two features; these are `DefendersInTheBox` and `A`, with a cost of 0.01361 on the validation set (see Figure 2). The selected features make make sense, as intuitively they are how fast the runner goes (`A`) and how many people are in the way (`DefendersInTheBox`).

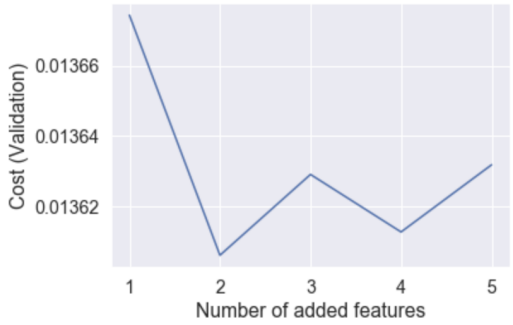

Fig. 2. Forward Feature Selection Costs on Validation Set

IV. Methods

A. Predictions

The prediction is the probability mass function (PMF) of yards gained. This is represented by $y_i \in [0, 1]$ for i = [1:199], where i = 1 corresponds to -99 yards gained and i = 199 corresponds to 99 yards gained. In other words, this is a classification problem with k = 199 classes corresponding to -99, ..., 99 yards gained. Since the desired final output is a cumulative distribution function (CDF), a cumulative sum is applied over the PMF to give the desired form of the final predicted distribution over yards gained.

$$\hat{y} = P(Y \leq j) \; \forall j = [-99 : 99]$$

B. True Labels

The true CDF is the unit step function centered at the ground truth yards gained on the play (Y_m).

$$y = H(x - Y_m)$$

C. Cost Function

$$C = \frac{1}{199n} \sum_{i=1}^{n} \sum_{j=-99}^{99} (P(Y \leq j) - H(x - Y_i))^2 \qquad (1)$$

D. Maximum Likelihood

Maximum likelihood models serve as a simple baseline by which to compare the performance of later models. These models are reasonable approximations given the following assumptions:

1) Most of the values for yards gained lie in a range of small negative and small positive values, despite many values being possible (Figure 1).

2) There is a large degree of randomness in the data.

We make MLE estimates where number of yards gained is a random variable Y and j is a specific yard gained value in $-99, ..., 99$:

$$P(Y = j) = \frac{\sum_{i=1}^{n} 1\{y^{(i)} = j\}}{n} \tag{2}$$

The first model is simply the CDF of yards gained across the whole dataset, taking no input features into account. When making a prediction, the CDF is squashed and normalized appropriately to avoid assigning probabilities to impossible outcomes (e.g. gaining 2 yards when only 1 yard is remaining between the line of scrimmage and the goal line).

E. Kernelized Maximum Likelihood

The Kernelized Maximum Likelihood model is a collection of 99 Maximum Likelihood models as described in section 2 above, one fit for each value of `YardsRemaining`. Therefore, each model is unique to a particular yard line and will only be used to predict outcomes on that yard line. The predictions of these models are combined to form a bagged prediction, where the weight of each individual model is determined using the Gaussian kernel 3. This makes intuitive sense, as the model corresponding to the true value of `YardsRemaining` will have the highest weight, and the weight will decrease as the fit model's `YardsRemaining` becomes further and further from the truth.

$$K(x, z) = \exp\left(-\frac{||x - z||^2}{2\sigma^2}\right) \tag{3}$$

F. Softmax Regression

Softmax regression was fit with L2 regularization, where the unregularized form of softmax is in equation 4 [3]. Softmax is a well understood ML algorithm, which encodes the outcome as vector representing probabilities of each class for a multinomial random variable. Regularization simply encourages the softmax model's θ to take lower values.

$$P(Y = j|x; \theta) = \frac{e^{\theta_j^T x}}{\sum_i e^{\theta_i^T x}} \tag{4}$$

G. Tree-based Models

We have 49 features, many of which are categorical and are attempting to classify the true yards gained on a carry. For that reason, we believe tree-based methods may be well suited to this problem. In general, trees repeatedly divide the feature space into regions based on some loss function.

In mathematical terms, dividing the space into regions is in equation 5 [3].

$$\begin{aligned} &\mathcal{X} = \bigcup_{i=1}^{n} R_i \\ &\text{s.t} \quad R_i \cap R_j \neq \emptyset \quad \text{for} \quad i \neq j \end{aligned} \tag{5}$$

Gini index in equation 6 was used as the loss when fitting.

$$L_{Gini}(R) = 1 - \sum_j \left(\frac{\sum_{i \in R} 1\{y^{(i)} = j\}}{|R|}\right)^2 \tag{6}$$

One tree based model we employed is random forests. Random forests seem applicable because of decision trees' adaptability to classification problems with reduced risk of overfitting by using many simpler trees with reduced correlation. Random forests introduce randomness randomly select the features and bootstrap sample the data when fitting each tree.

Another tree based model we used is XGBoost. This is a fast and highly performant implementation of gradient boosted decision trees, and has been popular in Kaggle competitions. With gradient boosting, the weak learners are fit to minimize the error (residual) of the previously fit weak learner model.

In the same vein, another tree based model we utilized is LightGBM. Like XGBoost, LightGBM is an implementation of gradient boosted decision trees. However, instead of growing trees a level at a time, the algorithm grows a single leaf node at a time. In problems with a lot of data, this smaller search space can help further improve the speed the algorithm learns.

V. Experiments

A. Results

We used sklearn to implement the multinomial model and random forest classifier. For softmax regression, we use a Newton solver with $L2$ regularization. For the random forest classifier, the maximum depth of the tree is set to 4, minimum number of samples for a split is set to 20, and minimum number of samples in a leaf node is set to 10. Default parameters are used for the XGBoost and LightGBM models. Below are plots of the various models we fit.

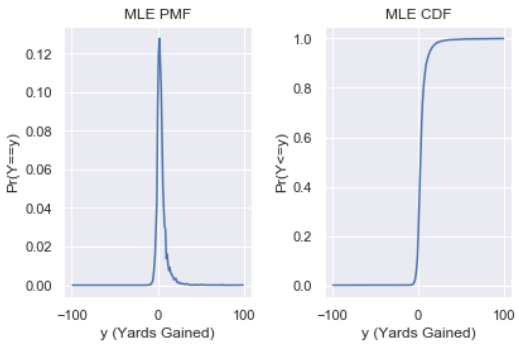

Fig. 3. MLE Model

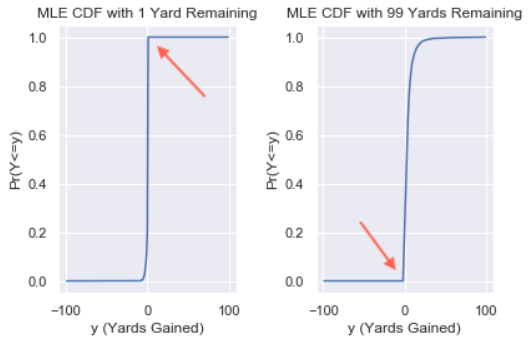

Fig. 4. Squashed MLE Model

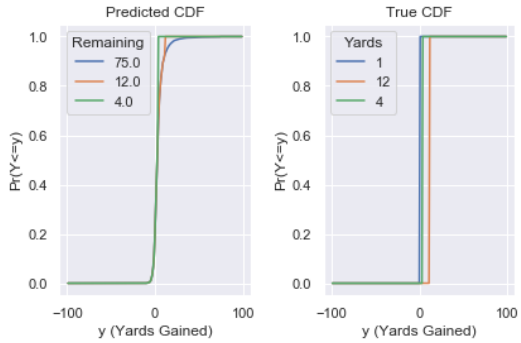

Fig. 5. Squashed MLE Model Predictions vs Truth

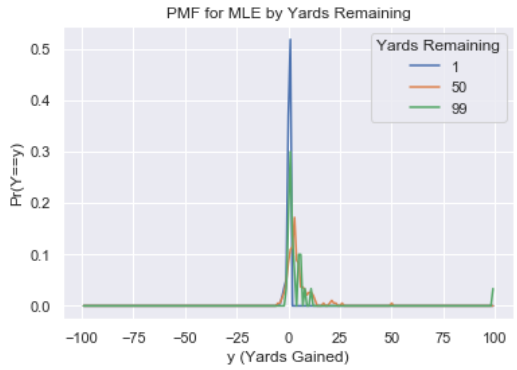

Fig. 6. MLE by Yards Remaining Model PMF

The results from the various models used to fit the data are summarized in Table I.

Model	Train Error	Validation Error
MLE	0.01393	0.01404
Kernelized MLE	0.01380	0.01404
Softmax Regression	0.01355	0.01366
Random Forests	0.01346	0.01361
LightGBM	0.02108	0.02123
XGBoost	0.01326	0.01359

TABLE I
COST ON TRAIN AND DEVELOPMENT SET

This metric quantifies how different our predicted CDF is from the true CDF. We can see that for all six models, the validation cost is slightly higher than the training cost, and the difference between the two costs are very small. As a result, we can be confident that we are not overfitting to the training set. The best performing model is the XGBoost model with a cost of 0.02108 for the train set, and 0.02123 for the validation set.

In addition to the model costs, we include a visualization of our performance in the competition. We note that this performance is associated with the random forest model as opposed to the lowest cost XGBoost model. This is because this model was fit after the competition deadline, thus we would expect higher competition performance with XGBoost.

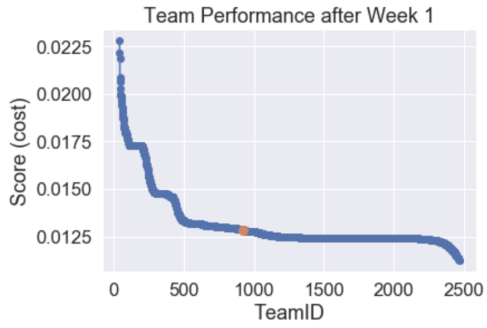

Fig. 7. Visualization of Performance in Kaggle Competition

B. Discussion

After fitting and evaluating the different models on the data, it is apparent that the tree based methods outperformed the naive baselines with which we initially began. Table I shows the costs calculated for each model on both the train and validation sets using Equation 1. This metric is used, as opposed to accuracy, because we are ultimately predicting a PMF, from which we obtain a CDF, and measuring its deviation from the true CDF which is a step function. The overall best model is the tree-based model XGBoost, with the lowest cost on both train and validation sets; 0.01326 and 0.01359 respectively. However, it is useful to note that, despite the fact that this model is more complex than the baseline MLE, the improvement in cost is not substantially large, on the order of magnitude of 10^{-4}. This is an interesting observation; simply using observed yards gained in order to calculate the PMF and CDF, and squashing these probabilities using `YardsRemaining`, provides reasonable predictions. This could be due to the particular features and combinations of features we used, as well as the hyper-parameters chosen.

Figure 3 shows the baseline model using an MLE estimate. We calculate the PMF for yards gained by looking at how many times y yards were gained divided by the total number of plays. Then using a cumulative sum, we calculate the CDF for yards gained from the PMF. Essentially we are treating each yards gained as a Bernoulli random variable, and importantly, we do not use any other features to inform our prediction.

Figure 4 shows how we "squash" the yards gained CDF. By squash, this means that depending on the yard line you are on (alternatively, how many yards you have remaining for a touch down) there are many infeasible yards gained values. On the left, one can see how the CDF becomes sharp since $Pr(Y \leq 1) = 1$ with 1 yard remaining. On the right, one can see how the CDF becomes sharp since $Pr(Y \leq -1) = 0$ with 99 yards remaining. Of course, this squashing can and should be done for any number of yards remaining. Indeed, we apply this squashing function to every model in order to eradicate probabilities we know to be infeasible.

Figure 5 shows three example predicted CDFs for real data, and their corresponding "true" CDF (step function from ground truth data). As can be seen, with 75 yards remaining the true result was 1 yard gained , with 12 yards remaining the true result was 12 yards gained, and with 4 yards remaining the true result was 4 yards gained. This supports our intuition that infeasible values should be squashed, as the most true yards one can gain with 4 yards remaining is 4 yards.

Figure 6 shows 3 of the 99 models fit by yards remaining with the intuition that the distribution of yards gained may vary by yards remaining. While this model performed the worst, whenever we boosted the 99 predictions using weights from a Gaussian kernel with high variance, the performance approached the best performance seen thus far doing MLE with all the data. From this, we concluded that segmenting our MLE models by yards remaining offers less room for improvement than focusing on feature-based methods, which allowed us to utilize attributes of the dataset.

Overall, we expect random forest classifiers to outperform our MLE estimate due to the ability of uncorrelated trees to make more accurate predictions. The random forest classifier makes use of the two features selected via forward selection, `Acceleration` and `DefendersInTheBox`. We thus attribute this improved performance to the use of important features in the dataset, chosen for the explanatory power they have when predicting yards gained.

VI. Conclusion

In conclusion, we are pleased with the overall performance of our model. We reiterate that improvements to the random forest method using XGBoost resulted in the lowest cost model. Due to the high degree of randomness in yards gained and the limited amount of training data, it is not reasonable to expect the most well-designed model to necessarily top the leaderboard (topping the leaderboard means performing the best on a hidden test set, but the model will eventually be evaluated on totally new data from upcoming games). By utilizing features in the dataset, we were able to reduce the cost obtained from the baseline MLE estimates.

Further work would involve exploring the features in this dataset more; we narrowed our search to seven features, from which only two were selected. By trying different features and combinations of features, we can analyse the changes, if any, to our model. Granted more time, speaking to expert football coaches could further inform our understanding of which features ultimately hold more predictive power in relation to yards gained.

It is interesting to consider the distribution of the performance of the models submitted to the competition 7. Roughly 80% of models perform very similarly, even though the architectures and hyperparameters are likely to vary significantly among these models. Yet, the cost decreases noticeably near the upper end of the distribution. How did these models perform so well? It is possible they are better tuned, but it seems more likely to us that these models involved unique insights that others lacked. Possible explanations include strategically combining features based on deep football knowledge, discarding outliers in the data, and finding ways to rule out impossible or highly unlikely outcomes. Squashing the CDF and differentiating MLE models based on *YardsRemaining* are two of our unique insights. Once submissions become public, it will be exciting to analyze these top submissions.

Finally, we reiterate that we were unable to submit our final model to the competition in time. It would therefore be useful to use our model to predict on the actual test dataset, once it is revealed to the public, and observe improvements, if any, to the overall cost.

Contributions

Travis: Problem formulation, Python implementation, analysis of results, experimentation, paper/poster writing

Brad: Problem formulation, Python implementation, analysis of results, experimentation, paper/poster writing

Dahlia: Problem formulation, Python implementation, analysis of results, experimentation, paper/poster writing

References

[1] NFL Big Data Bowl. https://www.kaggle.com/c/nfl-big-data-bowl-2020/data. Accessed: 2019-010-30.

[2] Nikita Ageev. Neural networks, multiple output + stadium clean. https://www.kaggle.com/truenikita/neural-networks-multiple-output-stadium-clean, 2019.

[3] Various Authors. Stanford cs229 notes. cs229.stanford.edu/syllabus.html, 2019.

[4] Tom Bresee. Next gen eda. https://www.kaggle.com/tombresee/next-gen-eda, 2019.

[5] hukuda222. Nfl simple model using lightgbm. https://www.kaggle.com/hukuda222/nfl-simple-model-using-lightgbm, 2019.

[6] Prashant Kikani. Nfl starter. lgb + feature engg. https://www.kaggle.com/prashantkikani/nfl-starter-lgb-feature-engg, 2019.

[7] Prashant Kikani. Nfl starter. mlp + feature engg. https://www.kaggle.com/prashantkikani/nfl-starter-mlp-feature-engg, 2019.

[8] Ethan Schacht. Nfl 2018 eda and feature engineering. https://www.kaggle.com/etsc9287/nfl-2018-eda-and-feature-engineering, 2019.

[9] DJ Sterling. Nfl big data bowl official starter notebook. https://www.kaggle.com/dster/nfl-big-data-bowl-official-starter-notebook, 2019.

[10] Jason Zivkovic. Comprehensive cleaning, engineering and eda. https://www.kaggle.com/jaseziv83/comprehensive-cleaning-and-eda-of-all-variables, 2019.