# NFL Big Data Bowl

Will Bekerman, Sam Fuchs, Tushar Khan
Cornell Data Science Insights: Spring 2020

---

## I. Overview

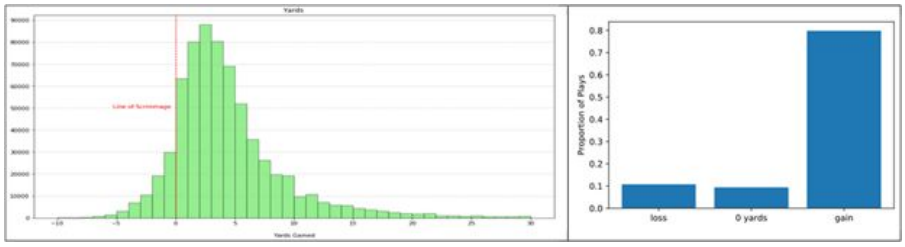
We utilized statistical, machine learning, and deep learning techniques to predict how many yards a team will gain on rushing plays using data provided by NFL's Next Gen Stats. Our goal was to gain deeper insight into rushing plays and help coaches develop effective strategies to optimize their play selections. Our final results gave us accuracy scores in the top 1% of Kaggle's NFL Big Data Bowl competition post-facto.

## II. Background

American football is a complex sport and it can be challenging to quantify the value of specific plays and actions within a play. In general, the goal of the offense is to rush or pass the ball to gain yards and score, while the goal of the defense is to prevent the offense from scoring. Thus, we sought to develop and apply models to predict how many yards a team will gain on given rushing plays. This will give us deeper insight into rushing plays and help coaches develop more efficacious in-game strategies.

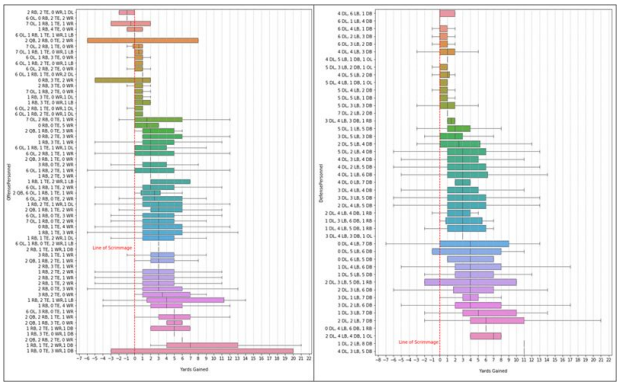
## III. Exploratory Data Analysis

We utilized game, play, and player-level data provided by the NFL's Next Gen Stats data. Our dataset contains 31,007 rushing plays from 688 games over the course of the last three NFL seasons. Out of the 2,568 total players included in the dataset, 443 are rushers, generally being halfbacks, fullbacks, and quarterbacks.



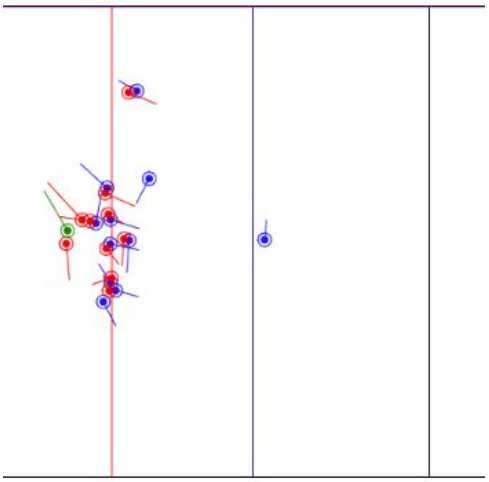

We note that our target variable 'Yards' is skewed with a median value of three yards, a mean value of just over four yards, and many outliers. The data ranges from a loss of fifteen yards to a gain of ninety-nine yards. We found that approximately eight out of every ten rushing plays resulted in a gain of yardage, while the remaining plays were split about evenly between yardage loss and no gain.

Next, we identified features that appeared to correlate with 'Yards', such as offensive formation, defensive personnel, distance to the first down, number of defenders in the box, and rusher speed and acceleration. We retained these highly correlated features and engineered others, such as player distance from the line of scrimmage and categorical features based on the typical yardage associated with offensive formation and defense personnel. We conducted separate F-tests of overall significance for these features and found they were very useful.



## IV. Visualization and Simulation

Since the dataset contains features taken at the time of handoff, we only have one snapshot in time for each play. To better understand the data we have for each play, we built a simple visualization that plots each player and draws a ray from their position to indicate the direction they are facing. We color each player according to their team and role (red is offense, blue is defense, green is offensive rusher), and indicate special yards of the play with colored lines across the field (red is line of scrimmage, blue is where the rusher gets tackled, black is the end zone). This gave us valuable insight into the chaotic data with which we were working. We see in the figure to the right that there exists a general lack of structure between the players since they have already started moving and their relative orientations do not seem indicative of the general direction they travel (notice how the rusher is facing backwards but actually advances forward quite far down the field).



While this visualization was effective in displaying the static attributes of each player at the time of handoff, our data also contained dynamic attributes for each player like their velocity and acceleration. We decided that the best way to capture these important features was to animate our visualization by extrapolating the position of each player using our dynamic data. However, doing this naively simply would result in each player moving away from the center and off the screen in some random direction within a few seconds. What we needed was a simulation engine.

We implemented a physics engine backend in Javascript, loosely based on the Box2D physics engine. We represent objects in the physics engine as *bodies* and hitboxes as *shapes*. Every

body has a shape to detect collisions and calculate intersection manifolds. There are three general types of bodies: static bodies which do not move because they have infinite mass; kinematic bodies which move according to their velocity and do not collide with static bodies or other kinematic bodies; and dynamic bodies which move according to their velocity, update their velocity according to their acceleration, update their acceleration according to the forces and impulses acting upon it, and collide with all types of bodies. The trajectory of a dynamic object is modelled by a differential equation which we solved using a simple Euler integrator. This is only accurate for small updates, but is generally sufficient for most physics engines.

We used impulse-resolution to resolve collisions between objects by using their intersection manifold to compute a collision vector, scaling the vector by the intersection depth, and applying the collision vector as an impulse to the colliding objects after scaling it by the masses of each object. This type of collision resolution causes objects to sink into each other if forces constantly push them together due to floating-point error accumulation. We corrected this using Baumgarte Stabilization, which introduces a constant factor *slop* that defines a maximum penetration between colliding objects and prevents the objects from sinking beyond this penetration depth.

The bodies are simulated together in an environment called the *world*. The world tracks all the bodies, quickly detects possible pairs of colliding objects using broad-phase collision detection, then detects which pairs are actually colliding using narrow-phase collision detection. Finally, these collisions are resolved using impulse resolution. Narrow-phase collision detection is necessarily implemented using brute-force since there is no other way to detect colliding objects. Thus, broad-phase collision detection is used to efficiently eliminate pairs of objects that are guaranteed not to be colliding using simple and quick heuristics that take advantage of the size of body shapes and their relative locations in order to speed up the narrow-phase.

We update the bodies in the world at a constant rate in the world regardless of computation speed by defining a global *computations per second (cps)* variable. For each world update, we track how much time has elapsed since the previous update and add it to an accumulator. Then, we iteratively decrement the accumulator by the inverse of the cps and apply updates to the bodies within a single world update. This has two advantages: first, each update is now a function of the global time instead of computation time, which means the simulation always proceeds at the same rate on any machine. The only advantage to having a faster machine is being able to define a higher cps, which results in a smoother and more accurate simulation. Any time left-over in the accumulator is used in the next world update, which means the simulation stays consistent and doesn't speed up or slow down due to rounding errors. Also, we can detect when the simulation is under load by detecting that the accumulator is above a certain threshold. If the simulation is under load, each world update will take progressively more and more time, resulting in a lagging simulation. Thus, we clamp the upper value of the accumulator, which effectively throttles the simulation when computations get intense.

With our physics engine backend, we modeled each player as a dynamic object with its position, direction, velocity, acceleration, and mass initialized from the data. We used static bodies to create the boundary of the field. Through experimentation, we found a lower collision elasticity to better model real-world collisions between football players. At this point, we had a simulation that could accurately show us what the play looked like at the *time of handoff*, before the players scattered and collided in different directions.

We decided to go even deeper with the simulation and use it to engineer new features in our data. A few ideas we considered were tracking the number of times the rusher collided with the opposing team, the distance the rusher traveled before his first collision, and the total number of collisions between the offense and the defense. This would require a simulation that was much more accurate at extrapolating the eventual positions of the players. We naively achieved this by constructing a very simple set of rules that dictate how each player moves as follows:

1. The offense, including the rusher, always accelerates towards the opposite end zone.
2. The defense always accelerates towards the closest offensive player.

These rules were implemented by applying the appropriate force vectors to the dynamic bodies in the physics engine. The resulting simulation showed defensive players changing their direction to move towards the offense, the rusher sometimes making a wide initial turn to cut towards the end zone, all with the occasional collisions stopping players in their tracks. In general, the simulation was more accurate *over time* than without enforcing game logic. For example, the defensive rule resulted in complex behavior like the defense tracking the offense and eventually catching them, sticking to them, and opposing their forward movement.

However, we found that the features we tracked did not significantly enhance our model accuracies, likely because the extrapolation is extremely naive. It makes very broad, general assumptions and is drastically affected by small changes in the input. As a result, we left the simulation as just a visualization of the data. In the future, we would like to keep tackling feature engineering using our simulation. With some time and experimentation, we believe that we could come up with an enhanced rule set that could capture a greater essence of the game.

## V. Modeling: Direct yardage prediction

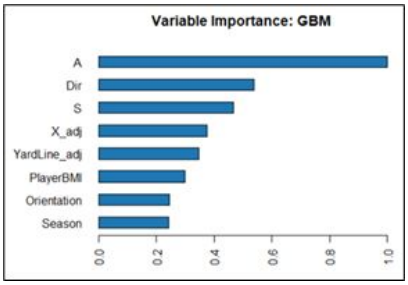
Continuous rank probability score (CRPS) is used as the scoring metric in the competition. It is a forecast accuracy measure that takes values between zero and one and compares the empirical cumulative of our predictions with that of the actual data. Notably, CRPS is directly related to the mean absolute error (MAE) of a predictive model (i.e., CRPS = MAE/number of predicted values = MAE/199). Our first approach to modeling was to directly predict yardage values since this is a very interpretable way of addressing our problem and is an important question to investigate.

We started by creating a baseline linear regression model with all of the features in our data. We evaluated the prediction accuracy of our statistical models by using three metrics: AIC, min-max accuracy, and correlation between the actual and predicted yardage values. The first criterion is an estimator of out-of-sample prediction error and is commonly used to assess the relative quality of statistical models. Min-max accuracy is a common measure of prediction accuracy and is the average of min(actual, predicted)/max(actual, predicted.). Finally, we evaluate the correlation between actual and predicted yardage since it is important for these values to have similar directional movement, i.e., when the actual values increase, the predicted values also increase and vice-versa. For our baseline linear regression model, we recorded an AIC of approximately 147162, min-max accuracy of approximately 0.35, and correlation between actual and predicted yardage values of approximately 0.18. In examining the difference in the summary statistics between our actual and predicted values, we note that the baseline model does a good job capturing the center of the data, but fares poorly identifying large gains or losses of yardage.

Next, we developed a more comprehensive statistical model using feature selection and adding polynomial and interaction terms between features. First, we ran backward selection using BIC criterion (similar to AIC, but limits the number of features used in the model). This procedure selected an eight-predictor model including player-level features, such as location, speed, acceleration, and position played, along with game-level features including yard-line, side of field, yards until first down, and number of defenders in the box. Then, we added some appropriate polynomial and interaction terms to the features in our model. We evaluated this model using the previous criteria and found that AIC was lowered by nearly four orders of magnitude to approximately 19.41, demonstrating that the overall quality of our predictive model had significantly improved. Notably, the correlation between actual and predicted yardage values improved to approximately 0.25, but the min-max accuracy did not experience much improvement. Noting the relatively poor min-max accuracy and examining the difference in the summary statistics between our actual and predicted values, we found that this model also does a good job capturing the center of the data, but cannot identify big yardage gains or losses. We hypothesized that this was due to our assumption that yardage gain is approximately Gaussian in shape: in reality, the distribution of yards appears to have a higher peak and heavier tails compared to a Normal distribution.

Thus, we abandoned our assumption of normality and moved on to random forest regression. We found that the correlation between actual and predicted yardage values improved to approximately 0.30 and the min-max accuracy increased to approximately 0.41. These are improvements of approximately five percent compared to our polynomial regression model using the same data including the same features. Our random forest regression model attained an MAE (average difference between yards predicted and the actual yards) of approximately 3.41 yards, a decrease of around 0.08 yards from our previous model. The root mean squared error (RMSE) also decreased by approximately 0.33 yards. Since errors are squared before being averaged, RMSE gives a relatively high weight to large errors, so RMSE is a useful metric when large errors are particularly undesirable. Thus, we can conclude that our random forest

regression model does a good job minimizing large errors, likely due to the lack of a Gaussian assumption.




Our best results came when using random forest with a log-transformed response, feedforward neural network, and gradient boosting machine. These models decreased MAE to approximately 3.30 yards and demonstrated similar overall performance. This means that we could, on average, predict yards gained plus or minus 3.3 yards for a given play. Notably, these models had a CRPS of approximately 0.017, while the top Kaggle submission had a CRPS of approximately 0.012. While we had some success directly predicting yardage values, it was a surprisingly difficult problem. In order to improve our CRPS, we decided to re-evaluate the problem.

## VI. Modeling: Probability distribution prediction

When we calculate the CRPS loss of a regression prediction, we assume a Heaviside step function for our prediction, which is compared to the Heaviside step function which is the goal of the CRPS loss. However, allowing our model to predict a more complex probability distribution will give it more degrees of freedom to improve its score. In particular, a more complex probability distribution prediction will allow the model to score better on long yardage gains and losses, since the gradient is more favorable, i.e., the model can learn to predict improbable outcomes without being overly penalized for incorrect predictions. This is the CRPS loss algorithm,

$$C = \frac{1}{199N} \sum_{m=1}^{N} \sum_{n=-99}^{99} (P(y \leq n) - H(n - Y_m))^2,$$

where H is the Heaviside step function and P is the input CDF.

*Feedforward CDF Classifier*

The first model we built for this purpose was a simple, densely connected classifier model with 40 input nodes, 3 densely connected hidden layers with 20 nodes, and 199 outputs, each corresponding to an amount of yardage gained or lost on the play. This model was trained to predict the Heaviside distribution, and the loss function against that distribution was the mean-squared-error, equivalent to the CRPS score. The one-dimensional input shape constrained the data we could use to train the model, as the data given for each play was essentially two dimensional (players x features). In order to train the model, we used only the data pertaining to the rusher. Nonetheless, our model was able to score a CRPS loss of approximately 0.0124 on a test set. At this point, it became clear that CRPS scores would be much more difficult to interpret than the regression errors we analyzed using previous models. As we looked at model predictions for various plays, we noticed that almost half of our CDFs

were decreasing at some point along the function. Since this invalidated our predictions, we reformulated our model to ameliorate this issue.

*Feedforward PDF Classifier*

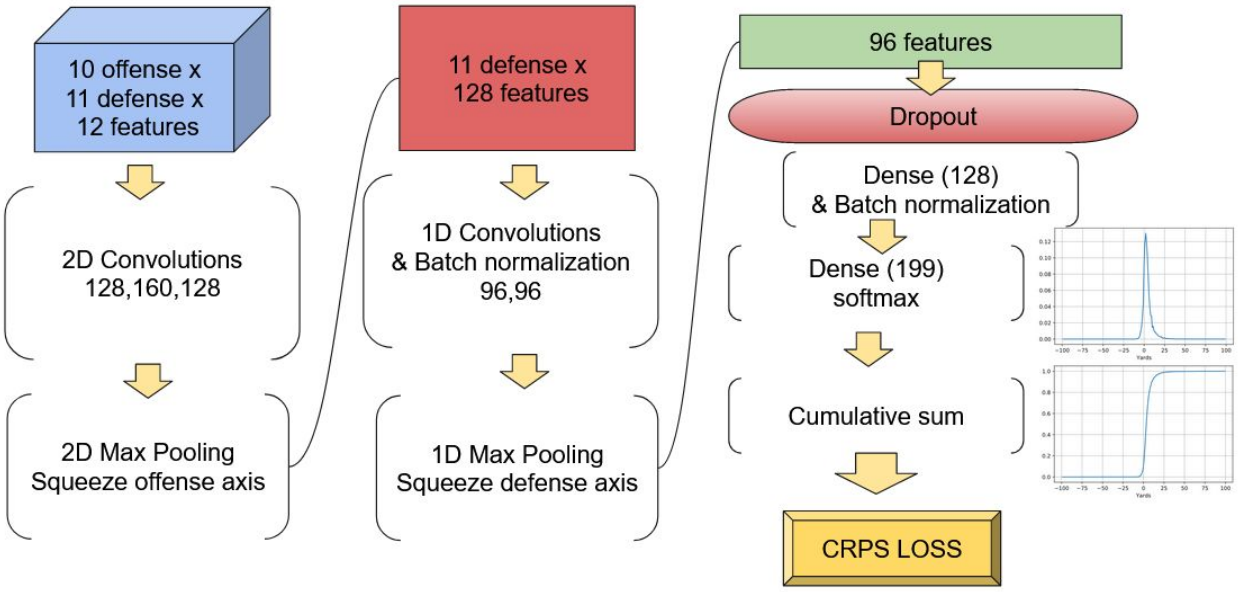
As we sought out ways to prevent a decreasing CDF, we found that any solution in the CDF space would necessarily complicate our model, because there was no straightforward way to enforce increasing sequential outputs from the model. Thus, we built a model to predict the PDF, i.e., the derivative of the CDF, and enforce that all outputs be between 0 and 1 by adding a softmax activation to the final layer. With some tuning, this new model performed as well as the original CDF model, but no longer produced decreasing CDF outputs. Since this model only used data for the rusher (only one of the 22 players on the field) and any attempt to increase our accuracy tended to induce overfitting, we sought to develop a new method to incorporate data for all of the players on the field.

*Convolutional PDF Classifier*

In order to use data for all of the 22 players, we had to restructure our network. Since the most important aspect of the plays we wanted to model was the interactions between opposing players, for each play, we formulated our input as a matrix of 10 offensive players (excluding the rusher) against 11 defensive players. Each layer of this matrix represented a different relationship between each pair of players. We started with the difference in the X and Y components of position and velocity, and then included the euclidean distance between each pair of players as well. As the model developed, we added in more features. These were the final 12 layers of our input matrix for each play:

- Defender velocity (X and Y)
- Offense-Defense relative position (X and Y)
- Offense-Defense euclidean distance
- Offense-Defense relative velocity (X and Y)
- Defense-Rusher relative position (X and Y)
- Defense-Rusher euclidean distance
- Defense-Rusher relative velocity (X and Y)

These features were convoluted in two dimensions, then along the defense axis, then passed through a dense layer before calculating the CRPS loss. This is the architecture for our final convolutional model:




This model far outperformed our previous models, likely due to the availability of more information about each play. This model was also more resilient to overfitting, which allowed it to be tuned more aggressively. Ultimately with the convolutional net, we were able to reach CRPS scores of approximately 0.012 on the test set. With respect to the Kaggle competition, these scores would have placed our submission in the top three on the leaderboard. However, it is important to note that our model was tested differently than the Kaggle submissions, which were scored throughout the season on data from ongoing games. This testing setup would be more challenging than our setup, which randomly selects plays to reserve for the test set. Ultimately, we cannot determine exactly how well our model would have scored in the competition, but we can confidently conclude that our model is able to make accurate predictions for the outcomes of individual plays.

## VII. Future Work

In the future, we hope to apply our work and create a tool for estimating how many yards a given play will gain. We seek to create an interface similar to a playbook that a user can interact with, select players and conditions, and see how models play out under given scenarios. Since our convolutional model only uses the positions and velocities of the players, it would be reasonable for a user (and perhaps a coach) to draw out a play and gauge how effective it might be by assessing the prediction made by our model. Additionally, we hope to extend our work to scrape game data from live NFL games and display real-time analysis and visualizations.