Machine Learning Gate: Using Historical Game And Weather Data To Predict The Next Offensive Football Play

Stanford CS229 Project

Andrew Lipschultz, Department of Chemical Engineering, Stanford University, `andrewl5@stanford.edu`
Ishan Mehta, Department of Computer Science, Stanford University, `ishanm@stanford.edu`
Ohm Patel, Department of Computer Science, Stanford University, `ohmpatel@stanford.edu`

March 16, 2024

1 Introduction

The National Football League (NFL) is a professional sports organization which consists of 32 teams across the United States. One of the simplest and yet most important aspects of the game for defenses is predicting what the offense will do. While the notable formations and plays are finite, the variation in route direction, route distances, audibles (instant changes), and motions (players moving after being in formation), make the actual prediction task very difficult. With the rise of data science in football, we thought a project surrounding this issue could give us insight into the way NFL offenses operate and maybe even allow us to help defenses in the future. Explicitly, we are combining NFL play data with weather data to predict whether a team will pass or run in a given situation. We leveraged models of increasing complexity to accomplish this task: logistic regression, random forest, Multi-Layered Perceptron (MLP), and Long short term memory (LSTM).

2 Related Work

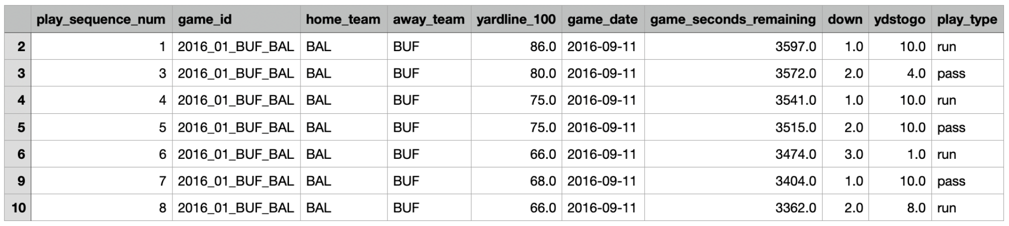
Choosing Our Algorithms
For an initial base on sports prediction and frameworks, we looked at (Horvat and Job, 2020) and (Bunker and Thabtah, 2019). The former suggests classification is advantageous compared to regression when working with a low scoring sport (football in our case). The second paper speaks to the architecture of neural networks, showing that neural networks were the most efficient and accurate approach when working with NFL data with a reported accuracy around 61 percent for predicting game results. Aside from the neural networks and general framework, the idea for logistic regression and random forest came from (Gifford and Bayrak, 2023) where they reached a 79 percent accuracy for predicting outcomes.

Pre-Snap: Defining Our Timeline
In defining when in a play-calling sequence to cut off the information flow, we relied heavily on (Goyal, 2020) and their exclusivity of only pre-snap data, which we also followed. The paper talks about the explicit difference between personnel(who is on the field) and formation(where they are on the field), defining a clear line between when a defense can act in a chronological manner. They also ran their models on two separate datasets, achieving 80% on all teams and 86% with single teams for determining run vs. pass. A similar method was taken in (Noldin, 2020) which highlighted the output buckets of play type, play length, and play direction. Although implict, their features were also all pre-snap and didn't include any formations, solidifying our belief in our timeline.

Weather
(Iskandaryan et al., 2020) demonstrates the importance of weather in sports predictive models. Although not football explicitly, it sparked the initiative to add weather data.

3 Dataset and Features

Our raw dataset came from two different sources: the nfl-data-py API and Visual Crossings. Using nfl-data-py, we extracted play data from 2016-2023 and removed all non-pass and non-run plays as well as redundant features. We also defined a pre-snap timeline as stated in our related works section, which led to us removing any features about formation, positioning, and output. The nfl-data-py API gave us play information with over 200 different features, both pre-snap and post-snap, for the desired games and teams. We used our knowledge of the domain and methodology of related works to filter out features



	play_sequence_num	game_id	home_team	away_team	yardline_100	game_date	game_seconds_remaining	down	ydstogo	play_type
2	1	2016_01_BUF_BAL	BAL	BUF	86.0	2016-09-11	3597.0	1.0	10.0	run
3	3	2016_01_BUF_BAL	BAL	BUF	80.0	2016-09-11	3572.0	2.0	4.0	pass
4	4	2016_01_BUF_BAL	BAL	BUF	75.0	2016-09-11	3541.0	1.0	10.0	run
5	5	2016_01_BUF_BAL	BAL	BUF	75.0	2016-09-11	3515.0	2.0	10.0	pass
6	6	2016_01_BUF_BAL	BAL	BUF	66.0	2016-09-11	3474.0	3.0	1.0	run
9	7	2016_01_BUF_BAL	BAL	BUF	68.0	2016-09-11	3404.0	1.0	10.0	pass
10	8	2016_01_BUF_BAL	BAL	BUF	66.0	2016-09-11	3362.0	2.0	8.0	run

Figure 1: First 10 columns and rows of data set

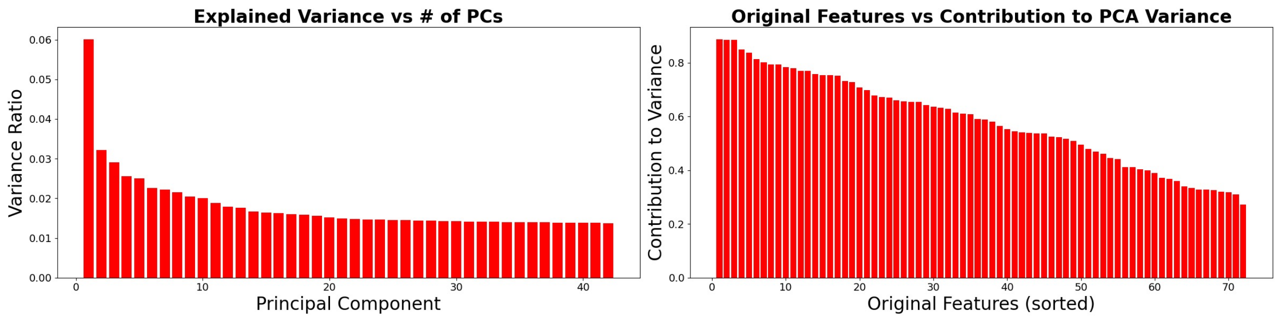


Figure 2: PCs vs variance and original features vs variance

to be only pre-snap. Then, using Visual Crossings, we downloaded weather for the 32 cities in which the teams play. The data set was extremely extensive, with over 30 features, so we reduced it to four factors with obvious implications: feelslike, humidity, conditions, and wind speed. We mapped the weather to each play by keying on the home team, giving us a large CSV file which included 239,533 plays with 28 total columns. To execute our required models, we created a 80/20 train/test split and a 64/16/20 train/val/test split. Finally, we used PCA to learn the features that most contributed to variance in the data set. We ran PCA to capture 75% of the variance in the data which left us with 43 principal components, and, to select the most important features, we took the squared contribution of each feature to the top principal components to find the 60 most important features.

4 Methods

4.1 Logistic Regression

Logistic regression is a model which outputs a binary classification based on a linear combination of independent features. We used the sigmoid function to squash the hypothesis into a range of $y \in \{0, 1\}$ before using a threshold of 0.5 to predict the output(1). We maximize log-likelihood of the independent classification probabilities to solve for the parameters(2) and accomplish this using stochastic gradient ascent(3). We used LogisticRegression from sklearn.

$$h(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \tag{1}$$

$$l(\theta) = \sum_{i=1}^{n} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)})) \tag{2}$$

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta \left(x^{(i)} \right) \right) x^{(i)} \tag{3}$$

4.2 Random Forest

A decision tree operates as a classification tool, where it predicts a given class based on the input variables. It has a graphical representation where the root node encompasses the entire data set, each node as we progress partitions the tree based on a binary or continuous input variable, and a leaf corresponds to the predicted class. The creation of an optimal decision tree is NP complete, so we use a greedy algorithm to construct a tree where the best split is chosen on each node based on minimizing a loss function. To create our decision trees, we used Gini loss to define out best splits(4). However, decision trees have a propensity to be overfit, so we used a random forest to inform our final prediction. A random forest creates multiple decision trees during training and outputs the majority vote of all the trees during testing; multiple trees are created by only considering a subset of the data and features at each split. We used RandomForestClassifier from sklearn.

$$G(p_{mk}) = \sum p_{mk}(1 - p_{mk}) \tag{4}$$

4.3 Neural Networks

4.3.1 MLP

A neural network is used to introduce non-linearity in both parameters and inputs for a classification task. Specifically, our MLP is a fully-connected two layer neural network (each with 100 nodes) with a non-linear activation function between layers; we used ReLU. The MLP is trained using stochastic gradient descent in backpropogation. We used MLPClassifier from sklearn. The formula for our two-layered MLP is

$$\forall j \in [1, \ldots, m], \quad z_j = \mathbf{w}^{[1]^\top} \mathbf{x} + b^{[1]} \quad where \quad \mathbf{w}^{[1]} \in \mathbb{R}^d, \quad b^{[1]} \in \mathbb{R} \tag{5}$$

$$a_j = ReLU(z_j) \tag{6}$$

$$\mathbf{a} = [a_1, \ldots, a_m]^\top \in \mathbb{R}^m \tag{7}$$

$$\bar{h}_\theta(\mathbf{x}) = \mathbf{w}^{[2]^\top} \mathbf{a} + b^{[2]} \quad where \quad \mathbf{w}^{[2]} \in \mathbb{R}^m, \quad b^{[2]} \in \mathbb{R} \tag{8}$$

4.3.2 MLP With Dropout

We added dropout to avoid overfitting. Dropout works by randomly deactivating a given neuron during each training epoch with a given probability p and then the remaining neurons are scaled by $1 - p$ to compensate for the reduced number of neurons and ensure consistency during training and testing. We utilized tensorflow and a dropout rate of 0.1. Our first layer had 128 neurons, the second had 64, and ReLU was our activator. We used batch gradient descent with size 20 in backpropogation.

4.3.3 MLP With Convolution Layer

A convolution layer works by passing a kernel, a matrix with learnable weight values, over the input data and summing the element-wise multiplication over the kernal area to assign a single value. Convolution is typically used to capture spatial data, but convolution, especially 1D, can also be used capture time series data–in our case, plays ran during a similar time period. We used the same neural net architecture as the one with dropout with a 1D convolution with 64 filters and a kernel size of 3 inserted as the first layer. We then flattened the data to pass onto the MLP. Defining $w \in \mathcal{R}^k$, and $k = 2\ell + 1$, we then pad the z values with $z_{1-\ell} = z_{1-\ell+1} = ...z_0 = 0$ and $z_{m+2} = z_{m+\ell} + ... = 0$. We then define the output(9) of the layer from (Kiranyaz et al., 2021)as

$$Conv1D = \sum_{j=1}^{2l+1} w_j z_{i-l+(j-1)}. \tag{9}$$

4.4 LSTM Model

An LSTM model is a variant of recurrent neural networks designed to process sequential data more effectively. Unlike standard feedforward neural networks, LSTM allows the output of a node to influence future inputs to itself and other nodes within the same layer, making it adept at handling time-series data or sequences of variable lengths. They are also able to handle the vanishing gradient problem, enabling them to remember information over extended sequences. This is helpful in our model as an LSTM can utilize data across an entire season. An LSTM uses: $i(t)$(10) the input gate, $f(t)$(11) the forget gate, $o(t)$(12) the output gate, $\tilde{c}(t)$(13) as the new memory cell, $c(t)$(14) as the final memory cell, and $h(t)$(15) as the hidden state. We define the formulas as:

$$i(t) = \sigma(W_{ix}x(t) + W_{ih}h(t-1) + b_i) \tag{10}$$

$$f(t) = \sigma(W_{fx}x(t) + W_{fh}h(t-1) + b_f) \tag{11}$$

$$o(t) = \sigma(W_{ox}x(t) + W_{oh}h(t-1) + b_o) \tag{12}$$

$$\tilde{c}(t) = \sigma(W_{cx}x(t) + W_{ch}h(t-1) + b_c) \tag{13}$$

$$c(t) = f(t) \odot c(t-1) + i(t) \odot \tilde{c}(t) \tag{14}$$

$$h(t) = o(t) \odot \sigma(c(t)) \tag{15}$$

The input gate determines what information is stored in a given cell, and the forget gate decides what information is to be removed from the cell. The output gate determines what is given to the next hidden state. The new memory cell generates possible new values for the cell based on the input, and the final memory cell computes new memory through combining the input and forget gates. Finally, the hidden state is the output of the LSTM. We used tensorflow to create an LSTM, and the architecture was two fully connected layer with 128 and 64 nodes respectively each with dropout with $p = 0.5$. We considered sequence lengths of 6(a drive), 60(a game), and 200(part of a season) and trained using batch gradient descent with size 32 in backpropogation.

5 Experiments / Results / Discussion

5.1 Greedy

In the data set, passes represent 57.5% of the data and runs represent 42.5%, so a greedy algorithm would predict all passes at .575 accuracy.

5.2 Logistic Regression

To evaluate logistic regression, we used accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$ and average log loss (equation 2). When we first conducted logistic regression on the unrefined feature set (before bucketing personnel and running PCA), we had hundreds of features in the one-hot representation and the model did not converge. However, after narrowing to 60 features, we got an accuracy of .724 and a log loss of .549 (table 1). The confusion matrix suggests we are better at classifying passes, which often have more distinct personnel and in-game situations.

5.3 Random Forest

After logistic regression, we deployed a random forest, and also used accuracy to evaluate performance. We tried to create a random forest with $k \in \{10, 50, 100, 200\}$ learners. It performed slightly better than logistic regression with an accuracy of .732, and 200 learners showed the best accuracy (table 1).

5.4 MLP

In designing the MLP, we tried to create $k \in \{2, 3, 4\}$ layers, but 3 showed no improvement over 2, and 4 performed significantly worse. We also tried $i \in \{100, 175, 250\}$ number of nodes, and increasing the number of nodes also showed no improvement, leaving us with out final structure of 2 fully connected layers of 100 nodes. To ensure we didn't overfit, we also ran the model on our training data and got .7275 accuracy, similar to our test. Additionally, we discovered plentiful data was essential for a well performing neural network. When we applied the basic MLP model to a single team(roughly 6,000 training examples compared to 250000 when considering all teams), the average accuracy was .645. The more complex MLP with dropout and a convolutional layers improved the accuracy significantly to .731, but neither topped the accuracy of a dataset composed of all the teams .740 accuracy (table 1).

5.5 LSTM

As oppose to the regular MLP, the LSTM performed best on single teams with a median accuracy of .740 and worse when trained on all of the data with an accuracy of .736. Regarding single team data, our best performer was the Pittsburgh Steelers with the sequence defined to be 6 plays with .766 accuracy(figure 6). Since this is the length of an average drive, it suggests the Steelers repeat plays within a drive or across sequential drives often. On the other hand, we struggled with the Seattle Seahawks at all timesteps which indicates they are more volatile and previous plays do not contribute much to future play calling. The LSTM was better able to find patterns in individual teams, which we hypothosize is because the previous plays run by team A are relevant to themselves but not to team B.

5.6 Overall Discussion

After seeing all results fall within a couple of percentange points across every model (except single team MLP and greedy), we looked to see whether there were noticeable patterns in what was being misclassified. We took the frequency-based average across all features for both correctly/incorrectly classified examples. We then ran a Kolmogorov-Smirnov p-test as described in (Berger and Zhou, 2014) to ascertain whether features were represented differently across the two populations. The results suggest that for all features there is a statistical insignificance across populations, representative features plotted below(figure 4). Thus, we hypothesize that the data is inadequate to predict much higher than .75 accuracy as the two populations are too statistically similar. In terms of misclassifications, figure 3 shows we misclassified 6644 runs as passes and 8829 passes as runs. We attribute the former to play-action passes, a play in which the quarterback fakes giving the ball to the running back before passing. The latter can be attributed to draws, which are plays in which there are no additional blockers and the quarterback stands in a shotgun passing formation before turning it into a run play.

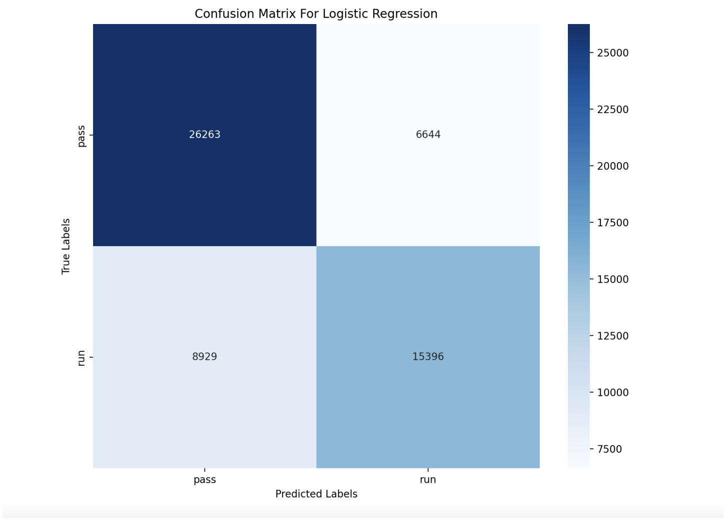


Figure 3: Logistic Regression Confusion Matrix

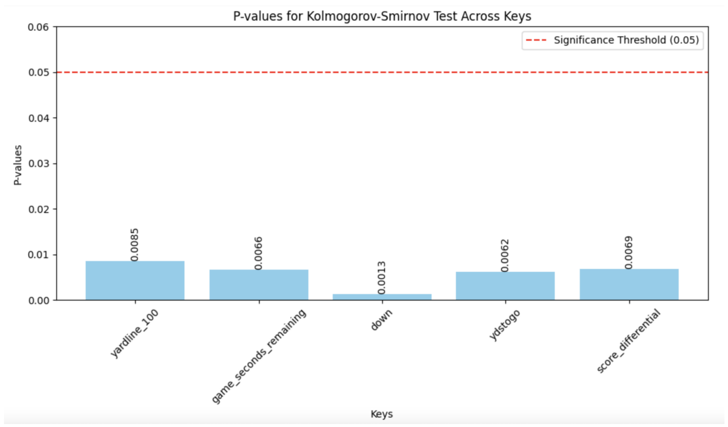


Figure 4: Kolmogorov-Smirnov Test

Algorithm	Accuracy	Loss	Dataset
Greedy	57.5%	N/A	All Teams
Logistic Regression	72.79%	0.549	All Teams
Random Forest	73.20%	0.542	All Teams
Neural Network	73.91%	0.535	All Teams
Neural Network w/ Dropout	73.91%	0.524	All Teams
Neural Network w/ Convolution	73.99%	0.525	All Teams
Neural Network	Max: 74.2% Min: 49.5% Median: 64.6% Average: 64.5%	Max: 3.93 Min: 0.565 Median: 0.819 Average: 1.193	Single Team
Neural Network w/ Dropout	Max: 75.0% Min: 70.3% Median: 73.0% Average: 72.8%	Max: 0.575 Min: 0.523 Median: 0.55 Average: 0.548	Single Team
Neural Network w/ Convolution	Max: 75.5% Min: 71.0% Median: 73.3% Average: 73.1%	Max: 0.566 Min: 0.515 Median: 0.542 Average: 0.543	Single Team
LSTM	73.60%	0.529	All Teams
LSTM	Max: 76.6% Min: 71.1% Median: 74.0% Average: 73.9%	Max: 0.562 Min: 0.483 Median: 0.525 Average: 0.523	Single Team
Training	72.75%	.561	All Teams

Figure 5: Results Table

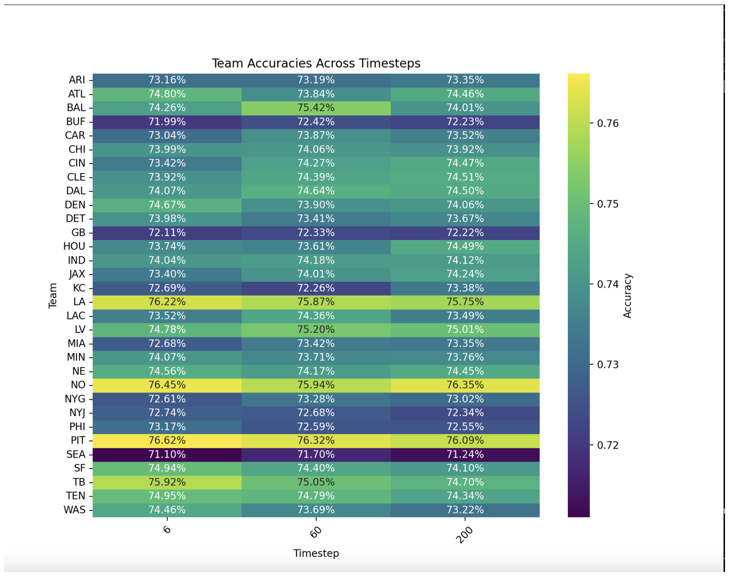


Figure 6: LSTM Heatmap

6 Conclusion / Future Work

In conclusion, we sought out to utilize various features within a football game including personnel, weather, score, field position, and more to be able to predict an offense's next play. After finding and preprocessing the data, we implemented a greedy, logistic regression, random forest, neural network (with convolution and dropout), and LSTM model across both a combined and individual team dataset. LSTM when trained on individual teams performed the best; we think this is due to the large influence previous plays have on future decisions. In the future, with more computation power, we would try to run an LSTM with larger sequence lengths and more hidden layers. We also would want to dive deeper as to why 77 percent seems to be an asymptote. We also would try to find a dataset that would reveals information about ambiguous situations(play action/draw).

7 Contributions

We felt the work was shared evenly and that we made a great team.
Specifcally:
Andrew Lipschultz:

1. Found, downloaded, and preprocessed all weather data
2. Wrote code for, ran, and reported all neural net algorithms including dropout and convolution layers
3. Wrote Sections 3 and 4 of the final report

Ishan Mehta:

1. Processed team data to include relevant features
2. Wrote code for, ran, and reported all LSTM algorithms
3. Wrote sections 1, 5, and 6 of the final report
4. Created Results Table

Ohm Patel:

1. Found and downloaded team data
2. Created and executed data preprocessing framework
3. Wrote code for, ran, and reported Logistic Regression and Random Forest
4. In charge of sections 2 and 7 of the final report

References

Vance W Berger and YanYan Zhou. 2014. Kolmogorov–smirnov test: Overview. *Wiley statsref: Statistics reference online.*

Rory P Bunker and Fadi Thabtah. 2019. A machine learning framework for sport result prediction. *Applied computing and informatics*, 15(1):27–33.

Matt Gifford and Tuncay Bayrak. 2023. A predictive analytics model for forecasting outcomes in the national football league games using decision tree and logistic regression. *Decision Analytics Journal*, 8:100296.

Udgam Goyal. 2020. *Leveraging machine learning to predict playcalling tendencies in the NFL.* Ph.D. thesis, Massachusetts Institute of Technology.

Tomislav Horvat and Josip Job. 2020. The use of machine learning in sport outcome prediction: A review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(5):e1380.

Ditsuhi Iskandaryan, Francisco Ramos, Denny Asarias Palinggi, and Sergio Trilles. 2020. The effect of weather in soccer results: an approach using machine learning techniques. *Applied Sciences*, 10(19):6750.

Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J Inman. 2021. 1d convolutional neural networks and applications: A survey. *Mechanical systems and signal processing*, 151:107398.

Niklas Clemens Noldin. 2020. Predicting play types in american football using machine learning. *Signature.*