When Machine Learning Meets Football

Tinhang Hong   Han Zhang   Leonardo Zhu   Kefan Jiang

Abstract

As Football being a main entertainment in American culture, it would have been nice that coaches of a team are able to mine information from the data and eventually help the team plays better with data driven techniques. One of the metric used to measure the quality of a play is how many yards a play has pushed since beginning, since football is all about pushing yards. We aim to target this as a regression problem through two different machine learning models. In the end, we find that both gradient boosting and neural networks are great methods to work with this data; however, gradient boosting is much simpler to work with since there are existing APIs.

1. Introduction

Football is incredibly popular in the U.S. It's not just a sport but a major cultural event that brings people together for game days, whether it's watching the NFL, college games, or local high school matches. Our team is interested in football too, therefore, predicting the play result is intriguing. The project is from Kaggle "NFL Big Data Bowl 2024". Our research aims to predict the net yards gained by the offense with LightGBM and Neural Network models. Some related projects focus on the players such as the player's performance, rising stars, and top performers in the play, while other projects analyze the tackling strategies. Our research focusing on the play result provides a different insight into the Kaggle project.

2. Methods and Pipe Line

This section is going to be separated into two parts: Data Processing pipeline and different stages of machine learning model development.

2.1. Data Processing pipeline

The raw dataset comes in different csv files, such as games.csv,players.csv, plays.csv and different weeks' tracking datasets. Additionally, we think it would make sense to add some aspects of individuals' quantitative performance, and thus we also introduced Madden 23 players' rating dataset from NFL website. It includes ratings for different players and we believe it would contribute to our final models. We managed to merge all of the dataset into one large dataset through pandas' different joining, merging and concatenating APIs. The details are included in our code files. It is worth to mention that we intuitively dropped some irrelavant features, such as players' descriptions. We also dropped the features that are only accessible after the plays, such as penalty yards, pass result and pass probability. Since this is real world dataset, we have also filled the null values with the average within that feature. We do not think the features that are only accessible after the play will make sense to our machine learning models. We also only kept the first frame in tracking datasets as the time after the first frame recorded in tracking datasets are only accessible after the play.

After merging all the datasets into one. We processed some data augmentation. Since we mainly focus on two models: Neural Network and light gradient boosting, we are going to have two different types of data augmentation sets for two models.

Neural Network: We used label encoder to convert all the categorical data types into integer with each of them representing corresponding meanings. Unlike linear model, We believe that arbitrarly assiging different level orders to the categorical data types will not be the issue as long as we add embedding layer in the model to catch the complexity between levels. We have also added Standard scaler to make sure all the variables stay on the same scale. We also transcribed some string variables such as height and weights into numerical data types.

Lightgbm: Since light gradient bossting can handle categorical features by itself, we only simply transformed the datasets to make sure that they are on the same scale. We also transcribed some string variables such as height and weights into numerical data types.

After some data exploration, we have found out the target variable is highly normally distributed with some degree lineates towards left. However, overall, there is not that much outliers that we need to consider with and thus should be suitable for our models.

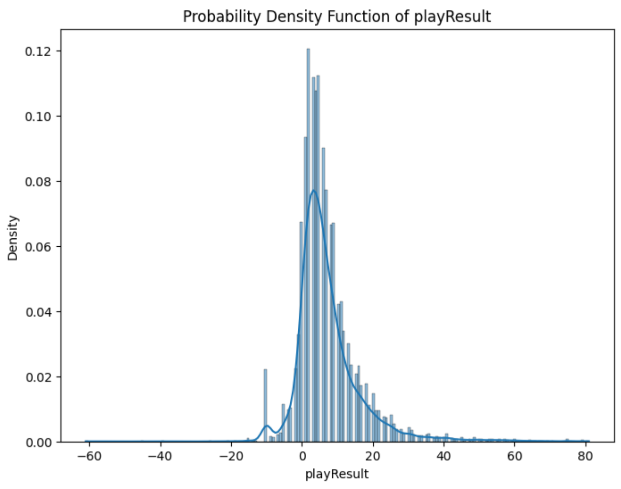

Figure 1. Distribution of our target variable.

We have also plotted out the heat map of variables that represent the linear correlation between every variables. However, it is worth to mention that this heat map only reserves linear correlation between variables and fail to extrapolate any complicated nonlinear relationship between them. The heat map helps us to see a high level relationship between them and probably provides some information of embedding layer in neural networks(lower correlation might implicate more complex embedding layer as we need more complicated model to mine the information).

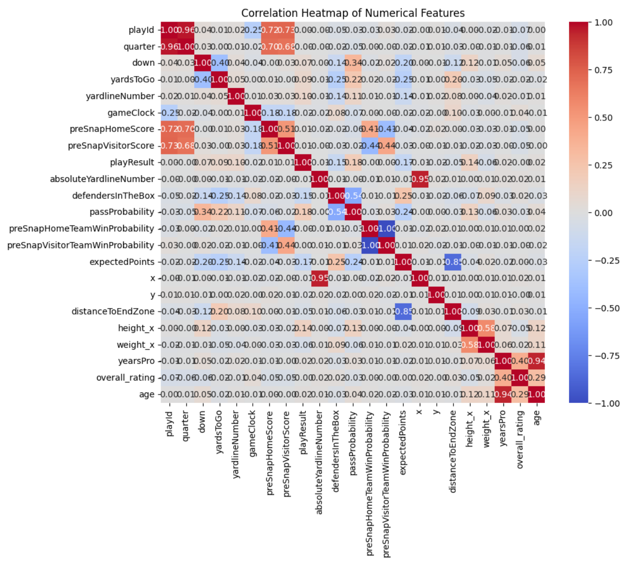

Figure 2. Distribution of our target variable.

2.2. Machine Learning Models

We choose to work with two models: Neural Networks and light gradient boosting models. For light gradient boosting, we selected it because it provides simple apis and many hyper parameters to tune with. It is also highly parallelizable by using k fold cross validation and its computing speed could potentially be really fast. Additionally, unlike neural networks, it provides simple APIs to interpret the model by looking at the feature importance. The gradient boosting methods are usually just the state of art for tabular data problems and really easy to work with. The objective function that we are trying to minimize is the mean squared error for the target variable, where it is defined as MSE $= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

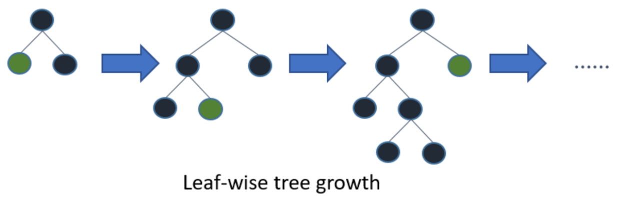

Figure 3. A simple picture that captures the broad idea of ensemble and gradient boosting techniques within lightgbm. src: Medium

We also choose to work with neural networks because it is usually complex enough to capture the non-linearity in a model by back propagation and different activation functions. It is fun to stack up different layers together to group a final neural networks to work with the dataset. We implemented Batch normalization layers, activation layers, embedding layers and different number of hidden layers to come up with a complexed neural networks. Note that embedding layer is extremely important because it captures the non linear information of categorical variables and map the information to higher space. In order to maintain the complexity of our model, we only implemented embedding layer on categorical features. We did not use any LSTM or tansformer to deal with information from previous inputs since we believe our tabular dataset does not include any sequenced information like text. We provide a skeleton structure of our neural network below. Note that we have not defined the hyper parameters yet(such as regularizations, dropout between layers etc). We implemented auxillary loss where we define the objective function of a single data entry to be $l(y_i, f(x_i)) = \sum_{j=1}^{k} w_j(y_{i,j} - \hat{y}_{i,j})^2$, where $j \in \{1, \ldots, k\}$ with k being the number of target variable we choose to work with and w_j to be the weights for that target variable's loss.

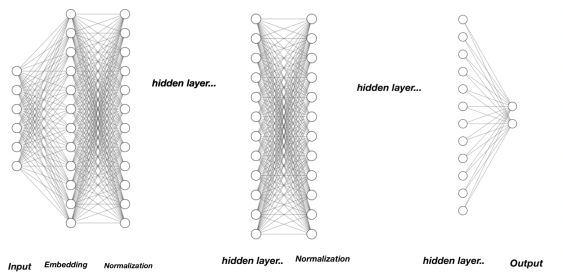

Figure 4. A skeleton structure of neural networks.

3. Results and Discussion

	MAE on val set	MAE on test set
Neural Network	4.970	4.738
LightGBM	4.875	4.812

Table 1. Mean Absolute Error (MAE) of models on the validation set and test set

The prediction output of the NN model and the LightGBM model we constructed is displayed in Table. 1, where the mean absolute errors on the validation set are also included for comparison. There is no apparent difference in the performance of the two models.

To quantitatively evaluate the overall performance of our two models, we figured out that the total range of the 'playResult' value in our test set is 96 yards so that the MAE of the predicted value derived from our models is non-significant (only $4.8/96 \approx 5\%$ of the total range), which means the errors produced by our models can be assumed as noise to some extent. The breakdown of the absolute error distribution will be discussed in the subsection of error analysis.

Furthermore, to indicate the feasibility of our two models, we also compared the prediction results with the MAE of values randomly drawn from a Gaussian distribution that is centered on the mean value of all the input `playResult` with the same variance as that of the corresponding distribution in the input dataset. It turns out that the values from the Gaussian distribution render a MAE of 9.889 that is almost double the MAE of our prediction models, which means both the NN and the LighGBM models having some meaningful subtlety other than random noise can perform well in our regression task.

We provide more details about the model training process below that lead to a robust conclusion about the practical correctness of our models.

3.1. Hyperparamter Tuning

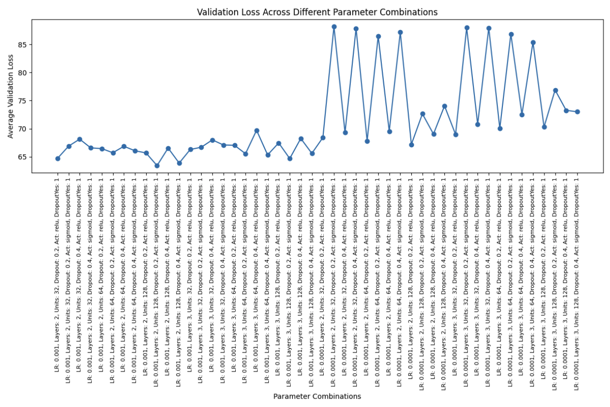

Figure 5. Validation losses for different combinations of the NN model's hyperparameters fine tuned with K-Fold cross validation

Given that we accomplished training the LightGBM model together with tuning the hyperparameters by simply calling the function `GridSearchCV` from `sklearn.model_selection`, we instead focus on the neural network model here to present more interesting details in the fine-tuning process.

For the Neural Network (NN) model, we conducted K-Fold cross validation upon explicitly iterating over and evaluating all possible combinations of the hyperparameters that are respectively learning rate, number of fully connected (FC) layers, number of neurons in each layer, dropout rate, and the selection of activation functions. According to the validation loss averaged out over the K-folds for each set of hyperparameters, shown in Fig. 5, the best NN model structure exists when the hyperparameters are set to the values in Table. 2. Eventually, we created the "best" NN model based on that optimal combination of hyperparameters, for which the training process is plotted over epochs in Fig. 6.

NN: Hypterparameter Name	Best Value
Learning rate	0.001
Number of layers	4
Number of neurons (per layer)	128
Dropout rate	0.2
Activation function	`'sigmoid'`

Table 2. Optimal combination of hyperparameters for the NN model derived from the K-Fold cross validation

One thing to note is that while K-Fold cross validation for the NN model typically took more than one hour to complete, the time spent in searching for the best set of hyperparameters for the LightGBM model is much shorter. In terms of the cost of training time, the LightGBM reasonably defeat the NN model in our task as their MAE performances are very close to each other.

3.2. Regularization

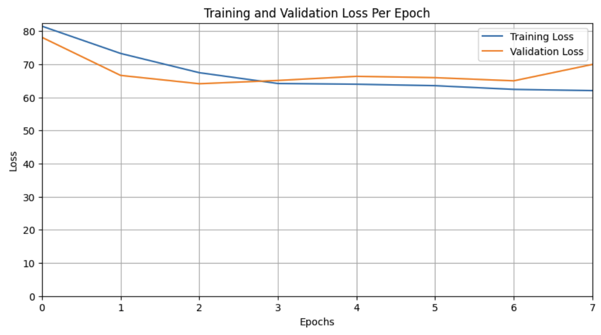

Figure 6. Training and validation losses over epochs in the training process of the NN model

We applied various regularization techniques when training our model to reduce the possibility of overfitting the data: for the NN model, we utilized the l2-norm regularizer in FC layers, added dropout layers, limited the number of FC layers, and defined early stopping with patience of 5; for the LightGBM model, besides using l2-norm, we fulfilled the purpose of constraining the complexity of model by controlling a series of hyperparameters embedded in random forests (Table. 3), which were fine tuned in cross validation implicitly implemented by the function `GridSearchCV` mentioned above.
More specifically for the NN model, as shown in Fig. 6, the generalization gap initially shrinks but eventually ends up with a relatively larger value exactly before the training process is halted. It means the overfitting behavior is successfully detected and suppressed by the early stopping mechanism. Besides, despite the fluctuation in the training and validation losses over epochs, the generalization gap overall remains within a very small range because we chose mean squared error in yards as the loss, implying the global success of our regularization approaches.

LightGBM: Hypterparameter Name
Total number of leaves
Minimum data size in each leaf
Maximum depth of the forest
`'lambda_l2'`
Learning rate
Feature fraction

Table 3. List of hyperparameters for the LightGBM model that was fine tuned by the `GridSearchCV` function

3.3. Feature Importance and Error Analysis

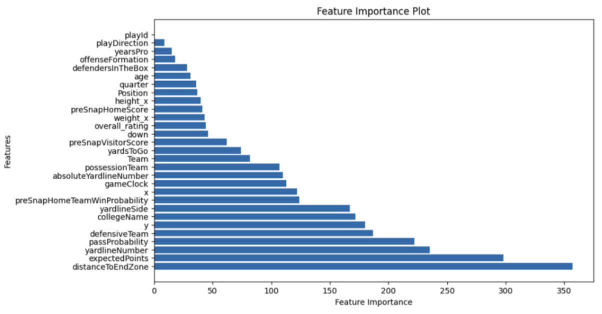

Figure 7. Feature importance scores calculated by the built-in function of the LightGBM model

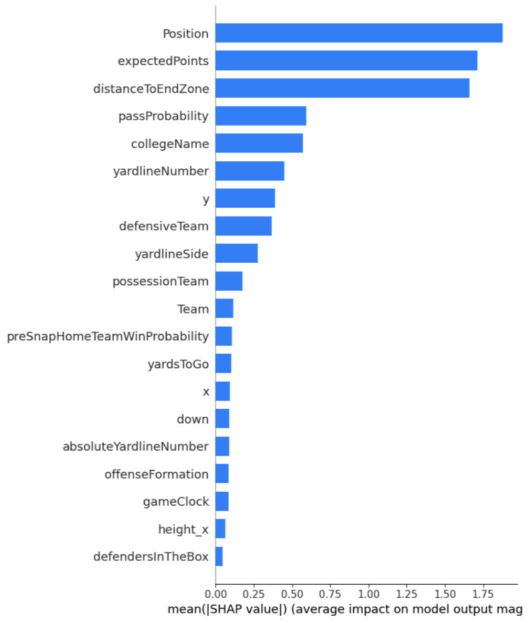

Figure 8. Feature importance represented by overall SHAP values for the LightGBM model

The graphs of feature importance reveals which features used in our model take the most important effect on the results of prediction. From another point of view, those most important features have the greatest power to “mislead” our model in error cases. Therefore, having a better understanding of the feature importance is equivalent to seeing the drawbacks of our models, which can provide some hints to improve the models.
Since the random forest used in LightGBM evolves by picking the locally best feature to split data, we relied on that interpretable property to analyze the importance of features, and therefore we merely created feature importance histograms for our LightGBM model (Fig. 7 & Fig. 8).

While the built-in methods in LightGBM measures feature importance mainly based on gain and frequency of each feature, the SHAP method takes into account interactions between features, the scale and impact of each feature, and both the local and global contributions of features. Due to the overall advantages of SHAP values that are not possessed by the built-in feature analysis, we intend to discuss the rationality of the first three most important features in the SHAP method to demonstrate the feasibility of our model. The top three most important features are 'Position', 'expectedPoints', and 'distanceToEndZone' respectively.

The 'Position' in Fig. 8 represents the official position of ball carriers in their teams. Since different positions feature distinct capabilities among players, if the players at one specific position are typically good at rushing instead of passing, then a ball carrier of that position will be more likely to rush with the ball at the beginning of each play. This can significantly affect the final play result so it makes sense for our models to assign more weight to the input 'Position'.

The 'expectedPoints' are the predicted points on each play, officially provided before the beginning of that play. In other words, the 'expectedPoints' can be assumed as an output of a professional prediction taking into account many factors relevant to each play. Consequently, it is theoretically important for our models to take 'expectedPoints' as a prior condition when making predictions about the posterior result.

The 'distanceToEndZone' refers to the distance remaining for a team to advance the football and reach the opponent's end zone to score points. It is a crucial metric for teams and players to consider when strategizing their plays and determining their next moves on offense. Therefore, the 'distanceToEndZone' should be heavily used by our models to predict play results as this value is practically important for each offending team.

According to the above analysis, our two models, or at least the LightGBM model, are conceptually correct regarding the feature importance. Correspondingly, when there are some remarkable errors in prediction, the most likely reason is that our models make mistakes under the influence of the most important features associated with outliers, which has been proven to be true for our test set since we manually checked some samples for which the model went wrong.

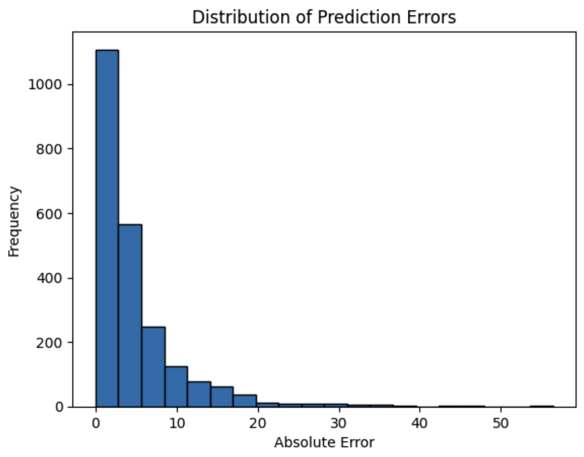

Figure 9. Distribution of absolute individual test errors generated by the NN model

As for error analysis, the numerical distribution of absolute errors on our test set is shown in Fig. 9, where the majority of errors are smaller than 10 yards. Since the error distribution is overall right-skewed without any outlying spikes, given the fact that both our models output predictions with very small MAEs, the models we constructed for this regression project show similar high-quality performance.

Combining all the above analyses, we can draw a conclusion that both the neural network model and the LightGBM model we trained based on the datasets of NFL Big Data Bowl 2024 are trustworthy to be practically used to predict the play results of NFL games.

4. Process

In the course of any research or project, the path to discovery often includes a series of experiments and approaches that do not always lead to successful outcomes but are essential in shaping the final results. This section is dedicated to documenting those efforts—both fruitful and unfruitful—that were significant in our project, yet did not find a place in the main results section. Here, we discuss various strategies we employed, obstacles we encountered, and the pivots we made in our project direction. By sharing these experiences, we aim to provide a comprehensive view of our process, offering insights that may be valuable for future endeavors and acknowledging the entirety of our scientific exploration.

4.1. Data Processing Phase

In Data Processing part, one big challenge we faced was to decide which features to keep and which features to drop because each data point has quite a few features that are not contributing to the final play results we are going to predict. Some of the features we just dropped it manually because intuitively, they are not correlated to the play result, for example the player's name, gameId and so on. Some features we dropped them due to the correlation matrix between all other features and play result. We found that the feature frameId has no correlation with play result at all. We go back to the original dataset and found that frameId has the same value for all plays. Therefore we decided to delete this feature.

Another problem we faced during the data processing stage was how we encode the categorical features in the dateset. We actually did several attempts to determine the best way of handling categorical features. The algorithm used by lightGBM model can take in categorical features without the need to convert them into numerical values manually. Therefore, we did not preprocess the categorical features in lightGBM model.

For Neural Network, by contrast, we have to process the non-numerical features in advance. We applied two different ways: we tried both one-hot encoder and label encoder, which are two common ways to do encode categorical features. We are not sure which one to use because they both have advantages and disadvantages. Label encoding implicitly assigns an order to the categories based on their encoded values. This numerical representation can lead the machine learning algorithms to assume an ordinal relationship where the higher number could be interpreted as ”greater” or ”more important” than a lower number. However, our categorical features are not ordinal. One-hot encoder, on the other hand, does not assume any ordinality or ordering between the categories. However, it converts each category value into a new binary column, and thus increases the dataset's dimensionality and encourages sparsity.

We tried both encoders and applied them on our preliminary model. And according to the mean squared error we got, we found that label encoder does a better job than one-hot encoder in predicting the play result. We discussed in out group why this is the case. We thought that label encoder keeps the simpler dimensionality of original feature space. In neural networks, especially simpler or less deep networks, having fewer input features can reduce the complexity of the model, leading to faster convergence during training. Also, one-hot encoder transforms categorical variables into a format where most of the elements are zeros, which increases the sparsity of the dataset. Sparse data can sometimes adversely affect the performance of neural networks, as these models are generally better at learning from dense data.

Therefore, considering the smaller mean squared error and the whole reasons stated above, we finally decided to use label encoder to deal with categorical features in neural network.

4.2. Model Training Phase

Continuing with the discussion of how to deal with categorical data, we also considered to use embedding layers to better handle categorical features. Instead of using sparse representations that grow with the number of categories as with one-hot encoding, embeddings maintain a constant and typically much smaller dimensionality. Moreover, the embedding vectors are learned based on the data, allowing the model to develop nuanced representations of the categories that reflect their relationships and impact on the response variable. This can lead to a more meaningful and contextually relevant understanding of the data without introducing ordinality implicitly as label encoder does.

We also explore the concepts of auxiliary loss when building the Neural Network thanks to the guidance of Robby Costales. PassLength is an inaccessible feature before each play while it have a strong correlation with the final play result as the correlation heatmap suggest. This strong correlation makes sense intuitively since how long the ball is passed significantly contributes to the final points the play result in. Since we have this feature on our hand while we are not able to use it in the model, it would be meaningful to use it as an auxiliary output before we get the final play result output.

Auxiliary loss do contribute a lot to our neural network model because it acts as a form of regularization in the sense that it asks the neural to learn additional tasks, and thus generate a more generalized representation of data instead of overfitting the main task in the training data. It also enhances the gradient signal in networks because the gradients form the loss function can diminish as they propagate back through the layers during training. The auxiliary outputs in the intermediate layer can provide addtional gradient signals in backpropagation, and thus improve the performance of the neural network.

The initial performance of NN with a larger number of layers was not very satisfactory compared to that of LightGBM, one probable reason is that the size of our training set is not sufficiently large and diverse to properly represent the true distribution of data. Therefore, we tried to augment the data further with perturbation techniques and random noise added but they were later proven to take very limited effect on the tabular data in our case.

5. Contributions

In this section, we will discuss some unmentioned methods or datasets that contribute to the final prediction of play result. We will also cite the libraries we use for building the model.

In terms of the efforts we make to improve our datasets, we introduced Madden 23 players' rating dataset from NFL website and add the ratings for each ball carrier as an extra feature to help our model generate better predictions. We think that the ball carrier's ability might be a contributor to the final play result prediction.

We used some open libraries in python to do the data processing as well as model building.
The first library is **LightGBM**, which is used for building and training gradient boosting models. To train Neural Network models, we also used **Keras, TensorFlow** libraries. We used **Sklearn** library to encoder the categorical feature, split the data into training, validation and test sets. Sklearn is also used to implement k-fold cross validation. We applied the commonly used **Numpy, Pandas, Matplotlib, Seaborn** to draw graphs and pictures. We also used **SHAP** library to calculate the most important features in lightGBM model to better interpret and explain our model. Finally we also called **Warnings** library to control warning messages.

Please refer to kaggle to get to know the details of this competition(Lopezv et al., 2023)

The github link is also attched in reference (Hong et al., 2024)

Few of the feature engineering idea is enlightened by other authors on kaggle (Mansoor, 2024)

References

Hong, T., Zhang, H., Zhu, L., and Jiang, K. Git hub repo for this project. Technical report, USC, https://github.com/davidhong013/csci567_final_project, 2024.

Lopezv, M., Bliss, T., Blake, A., Patton, A., McWilliams, J., Howard, A., and Cukierski, W. Nfl big data bowl 2024. Technical report, Kaggle, https://kaggle.com/competitions/nfl-big-data-bowl-2024, 2023.

Mansoor. Classification model for nfl big data bowl 2024. Technical report, Kaggle, https://www.kaggle.com/code/mansooralam559/classification-model-for-nfl-big-data-bowl-2024, 2024.