Predictive Modeling of NFL Play Calls Using Machine Learning and AI Techniques

Liam Shen, Dan Chioffi

* Northeastern University, Boston, MA

ABSTRACT

This study looks at how NFL players guess what the play calls will be, which is a very difficult task that requires them to make decisions all the time. The study uses play-by-play data from many NFL seasons and machine learning and artificial intelligence to build a complicated model. We consider things like difference in scores, the distance needed for a first down, where the teams are on the field, how much time is left, and the last set of plays. Statistics like the plays of the other team and the chance of not scoring make these parts stronger. The model uses Decision Tree Classifiers to set standards and value traits, LSTM Neural Networks to handle consecutive plays, and Q-learning to find the best strategies for each stage of the game. The precision, F1 score, and ROC-AUC of the model are compared to those of background models. This study shows a new way to use AI to analyze and predict NFL play selection, which makes sports analytics better. The approach and how engaged fans are may be affected.

INTRODUCTION

It's hard to predict NFL play calls because the game is so tactical and random. To make NFL play calls better, this project uses study and the latest machine learning and AI tools.

In sports analytics, predicting NFL play calls is a challenge but a chance to improve team tactics and fan engagement. So, this work is at the convergence of sports science and advanced analytics. The complexity of NFL play calls, impacted by many tactical and strategic factors, makes them ideal for machine learning. Coaches and analysts might improve game-day and pregame decision-making by extracting relevant patterns and forecasts from play-by-play data.

Because the NFL is dynamic and unexpected, forecasting play calls is difficult. Each NFL play is the result of a complicated decision-making process that influences future plays, generating a cascade effect that affects the whole game. This research uses a variety of machine learning algorithms to solve these challenges. Our technique combines analytical insights into a prediction model to capture play sequences and analyze opposing team strategies. We want to solve a long-standing sports analytics problem.

The Oxford Academic study on the unpredictable nature of NFL play calls makes it clear how important it is to look at the factors that affect a coach's real-time decisions. After learning this, we added more game scenario measures to our model to better show how the sport works in real life. Northwestern University suggested looking at basic types of play and past play trends. Our consecutive data analysis is based on this point of view. IOS Press discovered that real-time play prediction systems need to be able to respond quickly to changes in the game. We changed how we use LSTM Neural Networks because of this finding. These networks are great at dealing with sequential and time-sensitive data.

Through our study, we look at how hard it is to accurately show how NFL play callers make decisions. Coaches look at the play, the field position, the time left, the score difference, and the distance to the next first down. Because players have to guess the other team's plans, play calling is fluid and depends on the situation.

Strategic communication between teams is very unexpected because choices are made based on how the game is changing. Many traditional prediction models struggle to capture how complicated and interconnected strategy decisions are made in real time. What we do to fix these issues is use Q-learning, LSTM Neural Networks, and Decision Tree Classifiers. Q-learning changes based on what happens in the game, LSTM sorts and understands linear data, and Decision Trees accurately figure out which features are the most important.

By tackling NFL play prediction, our research analyzes AI's wider sports ramifications. In addition to technological achievements, our work advances sports analytics technology. Our methodologies and insights may boost sports data analysis research. Machine learning and NFL strategy research is more than academic; it's redefining sports analytics. We want to improve NFL play calling understanding by using results from past studies and AI techniques. This study makes sports data better and shows how machine learning can be used to make hard decisions, which is good for coaches, experts, and fans.

BACKGROUND

In this section, we will discuss some of the algorithms and structures used to build our models for the project.

Q-Learning

Q-learning is a model-free reinforcement learning approach that figures out the best way to choose an action for a limited Markov decision process. The system learns about an action-value function that tells it what the best action is based on how useful it is. Q-learning depends on the Q-function, also called the Q-table, to change based on how the agent interacts with its surroundings. The Q-function shows how much the state is expected to gain from taking action and following policy. Here is the main equation for the Q-learning update rule:

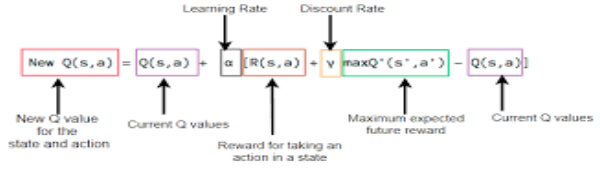

Figure 1: Q-Learning formula

We can use deep Q-learning for our study because it uses a deep neural network to get close to the Q-function. This helps guess what the NFL will do when the state or action spaces are too big for a normal Q-table, like when there are a lot of possible game events and actions.

Neural networks are used in deep Q-learning to predict Q-value functions. The network is given the state and makes Q-values for each move. The measured rewards and projected stage Q-values are used to figure out these goal Q-values.

Our work uses a modified version of Deep Q-learning to consider how NFL play calls are made in terms of time and order. This could mean changing the reward system to account for the strategic complexity of football play calling or adding repeated layers to the neural network to keep track of past actions and states. This plan helps the Q-learning program learn how to call NFL plays, which leads to predictions that are accurate for the present situation and for how the game's strategy will change in the future.

Recurrent Neural Network

Recurrent Neural Networks (RNNs) gather data in a certain order. RNNs are different from feedforward neural networks because they have memory, having loops to help them remember things. RNNs are designed to recognize a series of input features and guess what the next feature will be.

At each time step, an RNN takes in an input along with a hidden state from the previous time step, which contains learned information about the sequence thus far. The output from the current step becomes part of the input for the next step, creating a chain of information that flows through the sequence. This process is defined by the equation:

$$h_t = f(W[h_{t-1}, x_1] + b)$$

where h_t is the current hidden state, h_{t-1} is the previous hidden state, x_t is the input time, W are the weights, b is the bias, and f is a non-linear activation function, such as tanh or ReLU [8].

RNNs are great at jobs that depend on the situation, like language understanding and time series analysis. Because of the disappearing gradient problem, it's hard for regular RNNs to keep track of context over long runs. This issue shows up when an input value drops or rises dramatically on the network output and secret layers. Advanced RNNs, such as LSTMs, work well in this situation. Because they can keep and lose information over long amounts of time, LSTMs are good for jobs that require long-term understanding.

LSTM Neural Network

LSTMs have changed the way sequence prediction is done by keeping big temporal links.

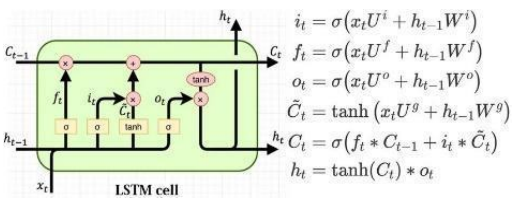

Figure 2: An LSTM cell

The LSTM unit's architecture, consisting of an input layer, multiple gating layers, and an output layer, facilitates this capability. Each gate within the LSTM cell—namely the forget gate (f_t), input gate (i_t), and output gate (o_t)—applies non-linear activation functions that allow the network to learn which data in a sequence is important to keep or discard [9].

The forget gate (f_t) uses a sigmoid function σ to decide which information to remove from the cell state. This function outputs a number between 0 and 1, which is then multiplied by the previous cell state (c_{t-1}) to apply the forgetting mechanism. The equation is as follows:

$$f_t = \sigma(x_t U^f + h_{t-1} W^f)$$

where x_t is the current input vector, h_{t-1} is the previous hidden state, U^f and W^f are the weights applied to the input and hidden state respectively for the forget gate.

The input gate (i_t) and the candidate layer ($\tilde{c}_t$) determine which new information will be stored in the cell state. The input gate, like the forget gate, uses the sigmoid function to decide which values to update, and the candidate layer uses the tanh function to create a vector of new candidate values. The updated cell state (c_t) is the sum of the previous cell state multiplied by the forget gate's output and the candidate values scaled by the input gate's output.

The output gate (o_t) controls what the next hidden state (h_t) will be, which is the output of the LSTM unit that is passed to the next time step or to an output layer. The hidden state is the product of the output gate's sigmoid function and the tanh of the updated cell state.

In LSTM output layers, the SoftMax activation function is often used to sort things into groups. SoftMax changes the secret state into a probability distribution with many output classes. This makes it ideal for classifying things into more than one group. Here is the equation for SoftMax function:

$$softmax(h_t W + b)$$

where W represents the weights connecting the hidden state to the output layer, and b denotes the bias. This function ensures that the outputs sum to one and are thus interpretable as probabilities.

To guess what the NFL teams will do, the LSTM model must look at data from a series of plays and make a probability distribution for what plays are likely to happen next. Using the SoftMax activation function in the output layer, the model may be able to correctly guess the chances of a play call. This function lets you correctly guess play calls and make smart decisions in real-time games.

Decision Tree Classifiers

Machine learning can't work without Decision Tree Classifiers, which are simple to understand and use. They are experts at classification jobs that need them to guess what will happen by using data attribute decision rules. To make a hierarchical decision model, these classifiers divide the information into groups based on trait values. The data is split up by decision nodes, each of which represents a feature of the information. The parts of this node show the rules for making decisions that change results and choices.

A Decision Tree Classifier is made by starting with the root node, which is the dataset, and then branching out by feature values [2]. Recursive splitting keeps going until it hits a stopping condition, which is usually when all the subgroups at a node have the same goal value or when splitting them further is not helpful. Root nodes start the process of making a choice, and leaf nodes show outcomes or groupings. Based on the values of the attributes, the decision nodes in between them split the information into different parts. Division is caused by Information Gain and Gini Index.

Information Gain is central to decision trees, particularly in the ID3 algorithm [3]. It's a measure based on entropy from information theory, representing the unpredictability or impurity in a dataset. In a binary classification context, entropy

$$H(S) = -p_+ log_2(p_+) - p_- log_2(p_-)$$

where $p_+$and p_- are the proportions of positive and negative examples in S). Information Gain, then, is the reduction in entropy achieved by partitioning the dataset based on a specific attribute, calculated as

$$IG(S,A) = H(S) - \sum_{v \epsilon Values(A)} \frac{|S_v|}{|S|} H(S_v) \quad ,$$

indicating the effectiveness of an attribute in classifying the dataset. The Gini Index, another key measure, is employed to evaluate the purity of a dataset and is particularly used in the CART (Classification and Regression Tree) algorithm. The Gini Index for a subset S is given by $Gini(S) = 1 - \sum_{i=1}^{n} {p_i}^2$, where p_i is the proportion of examples belonging to class i in S. A Gini Index of zero denotes perfect purity, with higher values indicating increased levels of impurity or mixture within the subset.

Decision Tree Classifiers help NFL play calling forecasters research and pick key attributes. They evaluate down, distance, and field position, which most affect play calling. Features' contributions to decision tree development are measured by feature significance. Classifiers are used to evaluate sophisticated models like LSTM Neural Networks and Q-learning. They let scientists compare different methods for understanding NFL play calling's intricate dynamics.

Related Works

Different academic views have contributed to NFL play prediction research, each with distinct insights and methodologies for advancing knowledge. Building on past research's foundation, we seek to enhance NFL play calling predictive modeling.

Our analysis is based on Oxford Academic's "Unpredictability of NFL Play Calls" study. Their study of NFL game volatility shows that play calls are difficult to predict, which our research will solve using machine learning algorithms and AI techniques [6].

Northwestern University's "Tendency Analysis and Basic Play Type Prediction," examines past play patterns and basic play types. Their study has influenced our sequential data analysis approaches, allowing us to spot trends in NFL complicated plays [7].

IOS Press's "Real-time Play Prediction" study emphasizes the need for adaptive models that can handle NFL game dynamics. We employed LSTM Neural Networks, which are great at processing time-sensitive data and adapting to changing game circumstances [7].

In their study "Predicting Play Calls with Hidden Markov Models," arxiv.org shows how probabilistic models may capture play calling's time-dependent properties. Our model's capacity to create and alter play-calling methods increased once Q-learning was added [4].

The topic in Towards Data Science's "Predicting NFL Play Calls" highlights data-driven model building. Their practical applications and insights into enormous datasets have influenced the collecting and curation of play-by-play data, which underpins our prediction models [1].

Each of these works has been important to NFL play prediction. These insights and our AI-based approaches improve sports analytics debates and widen the use of advanced computational tools in strategic sports decision-making.

Method & Implementation

Data

Our data preparation step included NFL play-by-play data from 2009–2018, as well as from 2023. The 2009-2018 set was used on the LSTM to tailor more towards remembering team play history, while the 2023 dataset was used for the Deep Q-Learning and Decision Tree Classifier.

First, we removed mistakes and filled in missing numbers to create a complete data set for analysis. We identified and converted NFL play calling features after data purification. 'Current down,' 'yards to go,' 'field position,' 'time remaining,' and 'team' were examples. These crucial elements were transformed into machine learning model-friendly formats to grasp each play's context. Category variables were encoded, and numerical values adjusted to standardize data across measurements.

LSTM model preparation modified the 'time' feature. The game clock string was converted to a continuous numerical variable named 'time_elapsed'. This adjustment was crucial for the LSTM model, which uses temporal data to understand play sequence and timing. Measurement of game time gives the LSTM network essential temporal context.

Using the Python library ecosystem, including Pandas for data manipulation and NumPy for numerical computations, and preprocessing, the final dataset was structured as CSV files for machine learning compatibility and simplicity of usage.

LSTM Neural Network

Choosing the model's most significant qualities requires strategic research into which play-by-play data components affect game results most. 'down,' 'ydstogo,' and 'yardline_100' were picked because they represent game strategy.

These qualities provide the LSTM model context to anticipate future movements accurately, which is important because play prediction requires understanding game factors like play significance and team placement.

Additionally, standardizing features with StandardScaler is crucial to preprocessing. This normalization prevents a huge feature from dominating the model's learning process by ensuring that other features contribute equally. Normalization is crucial in neural network models, since input data scales can significantly affect weight changes during training. We also normalized

features to zero mean and one standard deviation to improve training stability and efficiency.

Our model architecture records and learns from play-by-play data's temporal sequences using 100-unit LSTM layers. LSTM layers excel in sequential data processing, making them suitable for this application. The major temporal feature extractor, the first LSTM layer, understands play sequence patterns. This foundation allows subsequent LSTM layers to extract increasingly complicated temporal correlations and patterns. Batch normalization stabilizes and accelerates learning after each LSTM layer. Normalizing layer inputs reduces internal covariate shift and improves training efficiency. To prevent overfitting, each LSTM and dense layer has a 50% dropout. Dropout is a regularization approach that randomly deactivates a portion of neurons during training to make the network more resilient and independent of any single group.

Furthermore, a layer of 64 neurons with a ReLU activation function follows the LSTM layers. LSTM layers' temporal features are interpreted and mapped onto an abstract representation by this layer. A thick final layer with a SoftMax activation function provides a probability distribution across potential play calls. The model's classification job relies on this layer to turn the network's learnt representations into predictions.

We picked the Adam optimizer because it can handle sparse gradients and change learning rate. At 0.0005, the optimizer balances fast convergence with the risk of exceeding the loss function's minimum. The learning rate of LSTM networks controls how quickly the model learns from data. A high learning rate may cause the model to converge too quickly on a bad answer, whereas a low rate might impede training.

We used early halting mechanism with three-epoch patience was used in our training, along with monitor validation loss and stop training improvements to prevent overfitting. Early stopping in deep neural network training ensures that the model does not learn from the training data until it loses generalizability to unknown input.

The model is trained using 64 batches across 10 epochs. In addition to accuracy and AUC, a proprietary F1 score metric is employed to assess models. In class imbalanced situations, this statistic provides a more balanced assessment of the model's performance. The harmonic means of precision and recall, the F1 score, evaluates the model's accuracy and resilience, including false positives and negatives.

Q Learning

After preprocessing, we created an NFL play prediction Python environment. Important game elements were added to this setting to simulate NFL games. It imported crucial play-by-play and label mapping data for play type classification. The environment defines status and action spaces. State space contained current down, distance to first down, field position, game time remaining, and team identities. Action space, which represents a game's options, was matched to dataset plays.

To improve model learning, state transition and sampling functions were added [5]. These functions extracted game scenarios and results from the dataset for model training and testing. The environment's reset and step routines allowed the model to start new game situations and advance through game states.

To assess the model's play-calling accuracy, a reward calculation system was included. The reinforcement learning architecture relied on this technique to help the model understand its actions and adjust its strategy.

After creating the NFL environment, we developed the NFL Play Prediction Model using an LSTM Neural Network. This neural network had LSTM and Dense layers. For the model to store knowledge over time and relate current judgments to previous play calls, the LSTM layers were essential. Our dataset had many game scenarios and play sequences, making this characteristic crucial. The network had Dense layers to abstract and process LSTM layers' retrieved information. Dense layers helped predict play calls at the end of the procedure.

The model was improved by using multiple activation functions. The 'ReLU' (Rectified Linear Unit) function added non-linearity to intermediate layers, helping the model learn more complex data patterns. Our multi-class categorization problem suited the output layer's 'SoftMax' activation function.

The next thing we did was the Q-Network. This method used a neural network to approximate Q-function. This neural network had many layers to analyze input data and approximate Q-values [10]. By forecasting these Q-values, the network could assess the reward of each action in the current state and guide the model to maximize the predicted outcome.

The model needed training after Q-Network construction. The Bellman equation was crucial for model iteration to Improve forecasting. The model used this equation to update its Q-values during training to predict the utility of actions in different game stages.

The training process also tuned critical hyperparameters. These included the learning rate, which controlled learning speed and efficiency, and the discount factor, which balanced present rewards vs. future benefits. The epsilon-greedy method was also used for investigation. This method allowed the model to explore and learn about its surroundings while balancing known knowledge with new activities.

Decision Tree Classifier

Python's scikit-learn Decision Tree Classifier was used for benchmarking and feature analysis. Pandas loads our preprocessed NFL play-by-play dataset. Label Encoding converted category variables to numbers for analysis.

The dataset had features and the goal variable, 'PlayType'. The dataset was split into training and testing sets for model training and validation. We trained the Decision Tree Classifier on the training set with scikit-learn.

After training, we predicted the test set and calculated F1 score, accuracy, log loss, and AUC score. These measures illuminate the model's NFL play call prediction. We used Matplotlib to show the model's most important features.

RESULT & ANALYSIS

We used multiple criteria to evaluate Deep Q-Learning, LSTMs, and Decision Tree Classifier models for NFL play calling prediction.

Decision Tree Classifier

The Decision Tree Classifier was assessed using F1 score, ROC-AUC, test loss, and accuracy. F1 was 0.502, indicating the

model's accuracy and recall balance. ROC-AUC, which measures the model's class differentiation, was 0.608. The model predicted correctly with a test loss of 18 and an accuracy of 0.5.

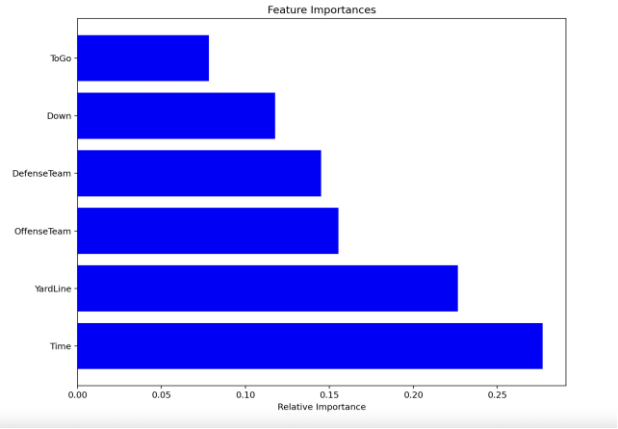

Figure 3

A feature significance analysis in the Figure 2 Decision Tree Classifier showed each variable's prediction importance. 'Time' was the most significant attribute with a relative value greater than 0.25, indicating its importance in NFL game strategy. Field position is crucial to play calling, so 'YardLine' was the second most important feature at 0.23. 'OffenseTeam' and 'DefenseTeam' also influenced the decision, but less than 'Down'.

LSTM Neural Network

The LSTM model's last epoch performance measurements and test evaluation reveal its NFL play call prediction capabilities. The model's growing F1 score (0.7122 in training and 0.7166 in testing) suggests a balanced precision-recall trade-off, which is crucial when false positives cost as much as false negatives.

Epoch #	F1 Score	Training Accuracy	AUC	Training Loss	Test Accuracy
1	0.5611	0.5862	0.8716	0.7605	-
2	0.6796	0.6803	0.9196	0.5821	-
3	0.6922	0.6930	0.9255	0.5627	-
4	0.6989	0.6997	0.9283	0.5527	-
5	0.7025	0.7030	0.9298	0.5477	-
6	0.7060	0.7065	0.9307	0.5448	-
7	0.7075	0.7080	0.9313	0.5420	-
8	0.7093	0.7099	0.9319	0.5403	-
9	0.7106	0.7112	0.9326	0.5373	-
10 (Final)	0.7122	0.7126	0.9329	0.5223	-
Test Set	0.7166	-	0.9359	-	0.7167

Table 1: Results after training on whole dataset

The training accuracy of 71.26%, together with a close test accuracy of 71.67%, illustrates that the model's predictions are often correct when compared to the ground truth. An accuracy of more than 70% in the high-stakes context of NFL games, where strategic choices are made in real-time, suggests that the model can be a beneficial tool for coaches and commentators. It can aid with game preparation and opponent analysis by offering insights into predicted play calls based on prior play trends.

The drop in training loss from 0.7605 to 0.5365, as well as the comparably low validation loss of 0.5223, indicate the model's potential to learn successfully while not overfitting to the training data unnecessarily. This decrease in loss suggests that as training proceeds, the model's predictions grow more matched with the actual findings. However, the modest rise in validation loss in the latest epoch implies that the model is overfitting or has achieved its learning capacity for the present feature set.

The AUC score of 0.9359 on the test set is quite notable. This measure shows a high level of separability; the model can discriminate between various forms of play calls. In fact, this indicates that the model can appropriately forecast whether a play will be a pass, run, or other form of play, affording teams an edge in predicting their opponents' actions.

The F1 score is especially critical in unbalanced datasets like those used in NFL play prediction, where certain play patterns may be more prevalent than others. The high F1 scores suggest that the model catches not simply the dominant play types, but also the less common ones, which are typically significant times in a game.

The model's generalizability is evidenced by the tight alignment of training and test accuracy, as well as F1 scores. This suggests that, once deployed, the model will likely maintain equal levels of accuracy when exposed to new, previously undisclosed NFL game data, making it a formidable tool for live game prediction.

Q Learning

The Deep Q-Learning model was assessed over a span of 10 episodes, focusing on total cumulative reward per episode, average loss per episode, and epsilon values over the episodes.

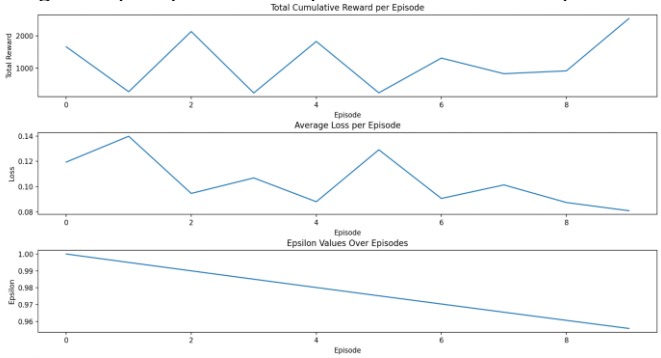

Figure 4

The total cumulative reward per episode, a statistic that represents the model's decision-making success, exhibited a moving trend with big dips and rises over the first episodes and a considerable gain in the last episode. The average loss each episode, which measures the model's learning progress, varied over episodes, with an apparent decline towards the later episodes, showing that the model's performance increased with time.

The epsilon values declined steadily over episodes, indicating a progressive shift in the model's learning strategy from exploration to exploitation, which is consistent with the expected behavior of epsilon decay in reinforcement learning. This trend in epsilon values implies that as the model gets experience, it grows greater confidence in the taught policy and relies less on random exploration, preferring to make decisions based on existing information.

CONCLUSION

The outcome of this research is a huge step forward in the application of machine learning to the hard challenge of projecting NFL play calls. Our huge dataset, systematically generated from

NFL play-by-play data from 2009 to 2023, serves as a fertile basis for our analytical ambitions.

The Decision Tree Classifier originated as a helpful baseline model, stressing the importance of key game features on play-calling decisions. Time elapsed and yard line placement were notably key indicators, directing strategic decisions in play execution. While the Decision Tree established a standard with an F1 score of 0.502 and a ROC-AUC of 0.608, it underlined the need for more complicated methodologies to capture the dynamic nature of in-game decision-making.

In our research to increase predictive acuity, we employed Long Short-Term Memory (LSTM) Neural Networks, which are well-known for their capacity to interpret sequential and time-series data. The LSTM model effectively detected the temporal links and strategic evolutions inherent in play sequences, delivering a complex insight that basic classifiers like as Decision Trees could not. The model's progressive learning was indicated by a final F1 score of 0.7166, a considerable increase over the Decision Tree's baseline, and an AUC of 0.9359, suggesting stronger discriminatory power.

While the LSTM network gave a good basis for sequence prediction, it was only when combined with Deep Q-Learning approaches that its complete potential was displayed. This convergence of methodologies enables a more in-depth analysis of the strategic intricacy of NFL play calling, as demonstrated by the positive trend in cumulative rewards over episodes and the effective convergence of loss values.

Based on the findings of this inquiry, it is evident that there is potential for improvement. Future study can concentrate on enhancing the granularity of our models in order to integrate player-specific strategies as well as environmental elements like as weather conditions, which have a substantial effect in determining play-calling choices. Extending the dataset to contain more extensive data, such as player locations and in-play actions, may greatly improve our systems' predicting ability.

Exploration of ensemble techniques, which may mix the properties of numerous models to achieve improved accuracy and robustness, is another promising route. Real-time prediction skills may alter in-game strategies and produce extraordinary levels of fan involvement.

In conclusion, the machine learning models created in this study demonstrated considerable potential in appropriately forecasting NFL play calls, providing important contributions to sports analytics. The outcomes of this study not only enhance our understanding of machine learning in complicated decision-making situations, but also create the groundwork for future developments.

GITHUB LINK

Github: https://github.com/liamshen10/CS5100_Final_Project

REFERENCES

[1] Atkinson, B. (n.d.). *Predicting NFL Play Calls. Towards Data Science. Retrieved from https://towardsdatascience.com/predicting-nfl-play-calls-during-a-game-6238b295e6f0*

[2] Charbuty, B., & Abdulazeez, A. (2021). *Classification based on decision tree algorithm for machine learning. Journal of Applied Science and Technology Trends, 2(01), 20-28.*

[3] Du, W., & Zhan, Z. (2002). *Building decision tree classifier on private data.*

[4] Dutta, R., Yurko, R., & Ventura, S. L. (n.d.). *Unsupervised Methods for Identifying Pass Coverage Among Defensive Backs with NFL Player Tracking Data. Retrieved from https://arxiv.org/pdf/1906.11373.pdf*

[5] Fan, J., Wang, Z., Xie, Y., & Yang, Z. (2020, July). *A theoretical analysis of deep Q-learning. In Learning for Dynamics and Control (pp. 486-489). PMLR.*

[6] Ötting, M. (n.d.). *Unpredictability of NFL Play Calls. Oxford Academic. Retrieved from https://academic.oup.com/imaman/article/32/4/535/6211379*

[7] Patel, P. (2020, January 31). *NFL Tendency Analysis and Basic Play Type Prediction. Northwestern University. Retrieve dfrom https://sites.northwestern.edu/msia/2020/01/31/nfl-tendency-analysis-and-basic-play-type-prediction/*

[8] Sherstinsky, A. (2020). *Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Physica D: Nonlinear Phenomena, 404, 132306.*

[9] Staudemeyer, R. C., & Morris, E. R. (2019). *Understanding LSTM--a tutorial into long short-term memory recurrent neural networks. arXiv preprint arXiv:1909.09586.*

[10] Tan, F., Yan, P., & Guan, X. (2017). *Deep reinforcement learning: from Q-learning to deep Q-learning. In Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14–18, 2017, Proceedings, Part IV 24 (pp. 475-483). Springer International Publishing.*