Final Project Report

Submitted by

Joshua Woods

Sagar Desai

Nathaniel Werner

Abstract

This project focuses on developing a predictive model that determines whether an NFL offense will call a run or a pass on a given play. Our model was trained on 4 seasons of NFL play by play data supplemented with the overall of the main player from the given play. The overall was obtained from the launch ratings of the Madden video games for each corresponding season. Feature engineering turned the raw game data into useful features that were utilized to effectively train multiple classification models. The best performing model is K-nearest neighbors with an accuracy score of 99%. With further development and more data this model could be utilized to gain insight into live game play calling and could be used as a defensive tactic to help understand what the offense is going to call next. This project demonstrates that this is a feasible task, and with some modifications, there is potential to be able to predict other characteristics or outcomes of the play.

Introduction

NFL Savant is a website that contains many different statistics that are all relevant to the NFL. Trends and other interesting statistics are highlighted across their website. We gathered the "overall" of the players through collecting the launch ratings from the Madden video games from the relevant year. Our goal with the use of this data, is to create a successful binary classifier of whether any given play in an NFL game, is going to be a run or a pass play based on information about the play such as field position, down, score, player ability, and more. Without finding any similar projects online, we were unsure what to expect in terms of accuracy and correctness when developing our models. However our models seemed to appropriately capture and recognize the patterns that are statistically significant.

Data Description

- **pbp19-22.csv:** initially consisted of 161,776 rows with 43 columns of data. This was all of the play by play data from the 2019, 2020, 2021, and 2022 seasons of the NFL. Data was collected year by year and aggregated all into one document.
- **playerStats.csv:** data about all of the players was collected from the launch ratings of the Madden video games. 4 separate files were combined into one file, the one file contained the player name, the overall of the player, and the season year of that player to account for overall changes year after year. Names of the player were edited to match the description column of our pbp19-22.csv dataset. The format is as follows F.LASTNAME where F is the initial of the players first name.
- **train/test sets:** all of the training/testing data was split through an 80%/20% split. After preprocessing and other data cleaning, this leaves around 30,000 samples in our testing set.

With these datasets we should be able to create successful models that can accurately predict whether a play is going to be a run or a pass play based on prior plays and other in-game metrics.

Preprocessing

In our play by play data, multiple steps to clean and set up the data appropriately had to be taken to effectively use it. We had to extract text data from the 'Description' column in our data set. We had to extract this to get the main player from the play, for example, a sample of the description column is as follows:

(1:32) (SHOTGUN) 12-T.BRADY PASS SHORT RIGHT TO 28-J.WHITE TO PIT 36 FOR 4 YARDS (26-M.BARRON).

Most of this information is available elsewhere in the dataset, the time of play, the formation, the yardage, and the amount of yards gained on the play. However the name of the player is not, and we believed that if a team has a star quarterback, this will influence the play calling so it is a relevant stat that needs to be included. Regex is used to pull the name of the first player from the above string so the result looks like this:

T.BRADY

This allows us to correlate the overall of the player to each row which will give our model more information to work with. After the addition of this column we had to remove 25 unimportant columns from the dataset as they did not impact the performance of the model. This left us with a total of 18 columns. Out of these 18 columns 4 were categorical, which led us to perform one hot encoding on those 4 columns. 2 of the columns had 32 values, while the others had two different categories. After one hot encoding, we were left with 87 columns. We chose one hot encoding for multiple reasons, the compatibility with different models, to avoid ordinal rankings where there is no linear scale representing the values. This is especially important when two of the 4 columns were about the team that was on offense and defense, we do not want the model to think that team 1 is better than team 32 simply since it is ranked higher. To fill in missing values in our dataset we used the simple imputer with the method of median which is the average value which is 77. We determined that this was a fair average to fill in, as players that did not have an overall had to have been called up throughout the season. The mode of the column was 75 which was not too far off which confirmed our belief that 77 was a fair filler value. Lastly we scaled the data so that the mean was 0 and the standard deviation was 1. We did this so that no one feature has undue influence on the model due to the scale of the values. This also makes the data more robust to outliers in the data which provides more strength to the model. The formula for StandardScalar is as follows

$$X = \frac{X - mean(X)}{std(X)}$$

X is the original feature, mean(X) is the mean of the column containing X, and std(X) is standard deviation of the column containing X. After all preprocessing of the data, our dataset left to work contained 151,788 rows with 87 columns.

Model Development

During this project we developed both linear and non-linear models. We did this to test the differences between models and to see which models would perform optimally. We developed 5 models, Logistic Regression, K-Nearest Neighbors, Random Forest, XGBoost, and MLP Classification. The first two being linear models and the other three being the non-linear models.

KNN and Logistic Regression were chosen due to their methods of handling classification problems. KNN is a non-parametric method, which means it does not make any underlying assumptions about the distribution of the data. This is useful because it allows the model to adapt to the minute patterns of the dataset. Logistic Regression, on the other hand, is a parametric approach, meaning it makes certain assumptions about the data distribution and employing a logistic function to model the probability of a particular outcome.

KNN is an algorithm used for classification and regression tasks. It operates by predicting the label of a new point based on the labels of its closest neighbors in the training set. It finds the closest neighbor by calculating the distance (Euclidean, Manhattan, Minkowski) between data points. For our model, Recursive Feature Elimination (RFE) was used to determine the most significant features for predicting NFL plays. This technique streamlined the model by retaining only the most impactful features. We optimized the parameters using GridSearchCV, focusing on the number of neighbors to find the optimal count of nearest neighbors, weights to decide the influence of neighbors based on their distance, and distance metrics to choose the most effective method for computing distances between points. A 5-fold cross-validation approach was used to ensure the model's ability to generalize. Recursive Feature Elimination (RFE) was used in conjunction with Logistic Regression to select the most significant features. We narrowed down to the top 10 features, ensuring a focus on the most impactful variables for our model. For parameter optimization, we use GridSearchCV to test a range from 1 to 30 for number of nearest neighbors, considered uniform and distance for weights, and considered Euclidean, Manhattan and Minkowski distance metrics. The GridSearchCV found that the optimal parameters were Manhattan for the distance metric, 1 for the number of nearest neighbors, and uniform for weight.

Logistic Regression is a popular statistical method used for binary classification problems. It is particularly well-suited for situations where the relationship between the predictor variables and the probability of a particular outcome is not linear but can be modeled using a logistic function. Two logistic regression models were implemented, each with a different regularization technique: logistic regression with lasso, or L1 regularization and logistic regression with ridge,

or L2 regularization. Lasso regularization aims at feature reduction and model simplicity while ridge regularization focuses on reducing multicollinearity and model robustness. Recursive Feature Elimination was used to identify the ten best features that contributed most to the model's predictive capability to simplify the model while retaining the best information for accurate predictions. To normalize feature values, we used StandardScaler. To deal with missing data, we used a SimpleImputer, using mean imputation. For parameter optimization, we used GridSearchCV to find the best 'C' parameter for both lasso and ridge regression. The 'C' parameter controls the controls the strength of regularization, with smaller values indicating stronger regularization. For both the lasso and ridge models, we found the best C score to be 100.

The random forest and xgboost models both share the fact that they are tree based models, and both rely heavily on feature importance statistics. They both are good at handling non-linearity which is important for our use case of varying game statistics. There are differences between the two models however, the random forest builds its trees independently of one another where the XGBoost builds its models based on one another. The XGBoost uses a gradient boosting feature in which each tree corrects the error from the combined ensemble. Whereas our MLP Classifier takes a different approach, it is a neural network that offers the ability to handle complex relationships within datasets. We hope these different model choices will highlight which ones perform optimally on our dataset.

Our random forest model was trained on an 80%/20% split on for the training and testing data which leaves roughly 120,000 samples for training and 30,000 for testing. The random forest models are known for high accuracy and a low chance of overfitting, which we believe will lead to high success during model training. During training of the random forest model, we tested multiple N_estimators hyper parameters. The values tried ranged from 1 - 200, with varying increments. The N_estimators parameter determines the amount of trees that are used for the Random forest model, and during our training, we found the more trees the better, getting a higher accuracy with each increase in the amount of trees. Random State in all models is equal to forty two for reproducibility.

During development of the XGBoost model, we performed a GridCV search to determine the best parameters for the model. We tested variants of the Learning Rate, N_estimators, and the Max Depth. For the learning rate, 0.1, 0.01, 0.001, for the N_estimators, 50, 100, 200, and for the max depth, 3, 5, and 7 were tested. A cross validation of three was used during the training to find the best parameters. The training/testing split is the same as the random forest and for the rest of the models. During training it turned out that 0.1 learning max depth of 7 and N_estimators of 200. It seems that the more complex the tree models are the better they perform, as this is the case in both of the tree models trained.

For the MLP Classification model, we had 3 hidden layers with sizes 100, 50, and 25. The model had a max iterations of 1000. This provides the model with ample training time to get the optimal

results. As this model is a neural network, we had high hopes for the complexities of the dataset to be picked up. The neural network can learn hierarchical features from the raw input data which makes it an ideal choice when working with large complex datasets. This model is also highly configurable which makes it appealing to ourselves as it can handle many types of datasets.

Model Evaluation

Our KNN model performed excellently for our dataset, with the accuracy of the test set being 99%, classifying every value in the test set correctly, except for one false positive. All other evaluation statistics support how well this model performs as well. A precision score of 99% shows that this model is very effective at avoiding false positives. The recall score is 100%, meaning our model correctly predicts every true positive value in the test set. The F1 score is 99%, backing up our high precision and recall. This model seems to perform too well, and overfitting the dataset is possible. Our dataset only has the plays from three NFL seasons, and as more data from NFL seasons becomes available, we could prove or disprove this theory. One thing that definitely supports this theory is the perfect scores the model produced on the training set, as shown in *Figure 1*. Despite the possible overfitting, the model still performed very well on the test set, with all scores being within 1% of the perfect test score, showing that this could be extremely useful in a real-life situation. The confusion matrix in *Figure 2* shows just how well this model performed

	Accuracy	Precision	Recall	F1 Score	AUC
Training	1	1	1	1	1
Testing	0.99	0.99	1.0	0.99	0.99

Figure 1

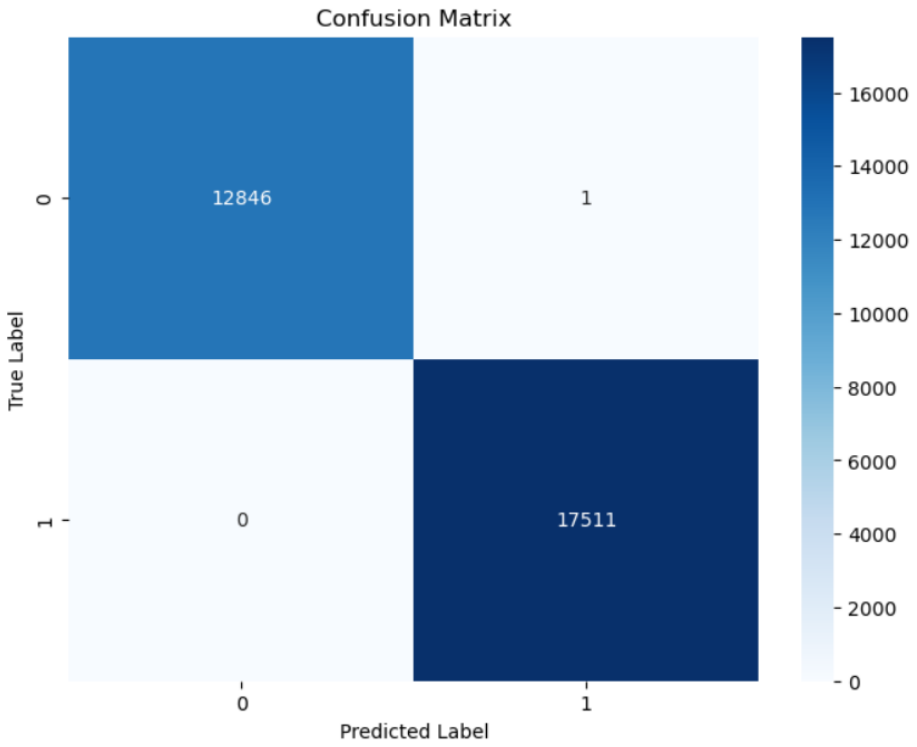

Figure 2

Unlike our KNN model, both of our logistic regression models were the worst performers out of all the models we tested, with a testing accuracy of 73%. The other evaluation metrics were similarly disappointing. The models produced a precision of 75%, a recall of 81%, and an F1 score of 78%. *Figure 3 and Figure 4* visualize these disappointing scores, showing the relatively high number of both false positives and negatives. The fact that all the models produced the same evaluation scores on both the training and testing data is strange but can be attributed to the high value of the 'C' hyperparameter of 100 that was found to be best by GridSearchCV for lasso, and a value of 1000 was found to be best for ridge. The C hyperparameter is the inverse of regularization strength, so a score of 100 implies a low regularization level. Normally, this would raise the risk of overfitting, but the training and test scores shown in *Figure 3*, show otherwise, as they are within 1% of each other. The C value being this high, in conjunction with the lower evaluation scores on training and testing sets, shows that a more complex model may be better suited for this dataset than logistic regression.

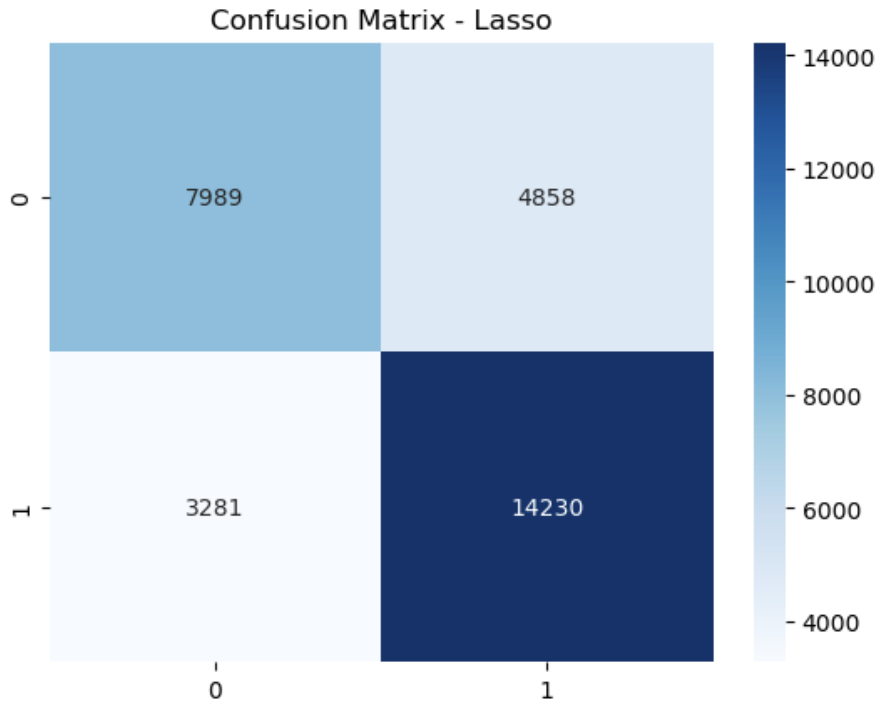

Figure 3

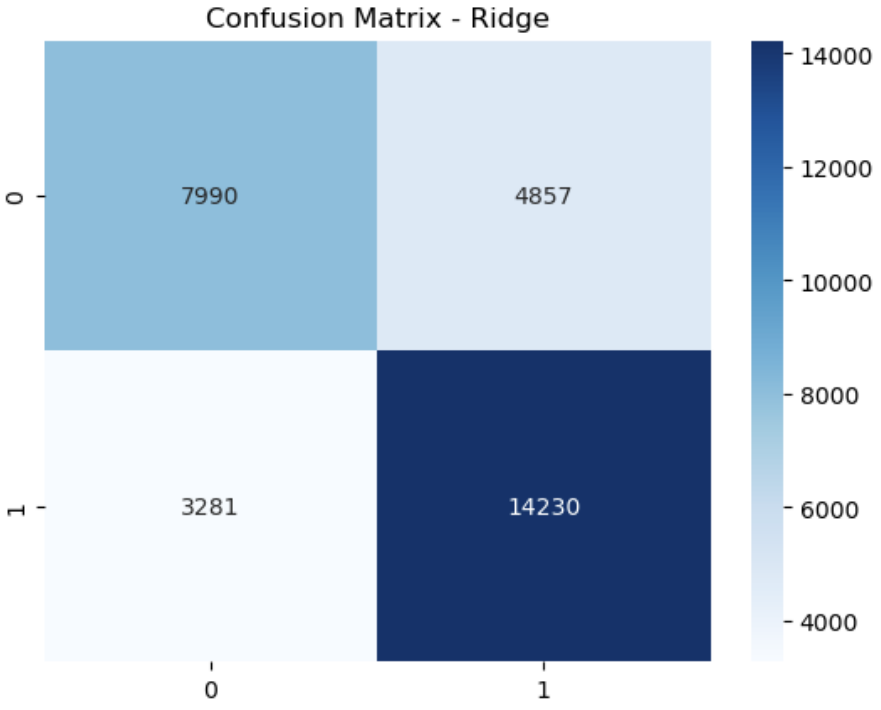

Figure 4

	Accuracy	Precision	Recall	F1 Score	AUC
Training	0.74	0.75	0.82	0.78	0.80
Testing	0.73	0.75	0.81	0.78	0.80

Figure 5 (Ridge and Lasso Scores)

Our random forest model performed rather well for our dataset, with the test set accuracy being a commendable 88%, showing that the model correctly classifies 88% of the values in the test set which comes out to around 26,000 of the 30,000 samples in the test set. Other statistics also support the fact that this model performs well, such as the recall which is at 91% which shows the percentage of correctly predicted positives to actual positives. Precision, which shows the percentage of accurate positive predictions, is an impressive 89% which shows the ability to limit false positives within the predictions. The F1 Score stands at a satisfactory 90% showing the balance between recall and precision. However, while this model seems good when looking at the testing scores, I believe that the model is overfitting compared to the training data, as all of the aforementioned stats stand at a consistent one for the training dataset. All statistics are shown in the table in *Figure 6.* This is possible due to the large number of trees in the model, and it could be memorizing the training data even though we have over 120,000 samples in the training set. The difference between the training and testing set across most metrics is no more than 12% in the worst case, and as low as 5% difference in the best case, so this model does still show potential to be useful in a real scenario. The ROC curve for the testing data shows great promise as well as shown below in *Figure 7.*

	Accuracy	Precision	Recall	F1 Score	AUC
Training	1	1	1	1	1
Testing	0.88	0.89	0.91	0.90	0.95

Figure 6

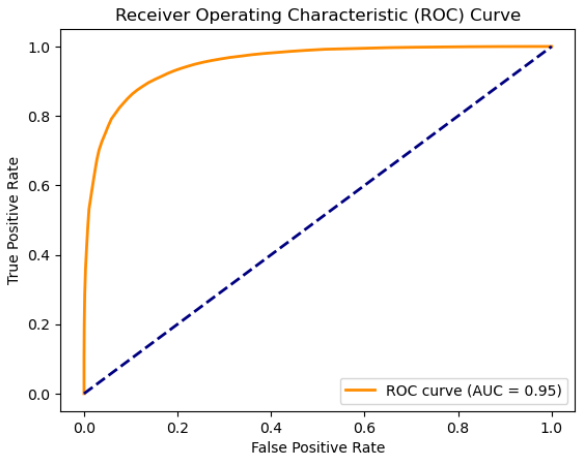

Figure 7

The XGBoost model showed great promise as well with the performance being similar and better than the random forest model. For our accuracy, the XGBoost model achieved an astounding 93% accuracy which was very good for our dataset where we were unsure if the models we were training would be able to effectively perform. The recall stands at a 95% on the testing data indicating a high percentage of correct positive predictions. Same goes for the precision, showing a 92% indicating only 8% of false positives. The rest of the relevant performance stats are shown in *Figure 8,* the table below. A key difference between this table and the one shown in *Figure 6*, is that the training and testing scores are much closer than they were with the random forest model. This leads me to have greater confidence that this model is not overfit compared to the random forest. This performance of this model compared to random forest shows that while they are both ensemble learning tree models, they are not equal as the XGBoost model performed with a better accuracy score and the rest of the scores. This shows that for our data set, the gradient boosting optimization featured in the XGBoost model performs with greater success on our dataset. The regularization techniques likely play a factor in this as well, which helps the model in preventing overfitting. Along with the supporting scores, the feature importance tree makes sense with our predictions. The columns that the model selected with high feature importance, make sense to us as to why they would be chosen. For example, the feature with the highest importance is Overall, which makes sense, as if your team has a star player in either the running back or quarterback position, it makes sense that the play would end up with the ball in their hands. The second most important column, is yards, which also makes sense as that is a large part of the play, most running plays typically yield less than 10 yards while passing plays are more likely to go for a greater amount of yards. Some of the less important columns, out of the top fifteen pictured in *Figure 9* below, are columns like OffenseTeam_NYG, which signifies a relationship between that team and either having tendency to run or pass the ball more consistently than other teams. This could be due to the star running back that the giants have, or due to the game plan of the coaching staff.

	Accuracy	Precision	Recall	F1 Score	AUC
Training	0.94	0.93	0.96	0.95	0.99
Testing	0.93	0.92	0.95	0.94	0.98

Figure 8

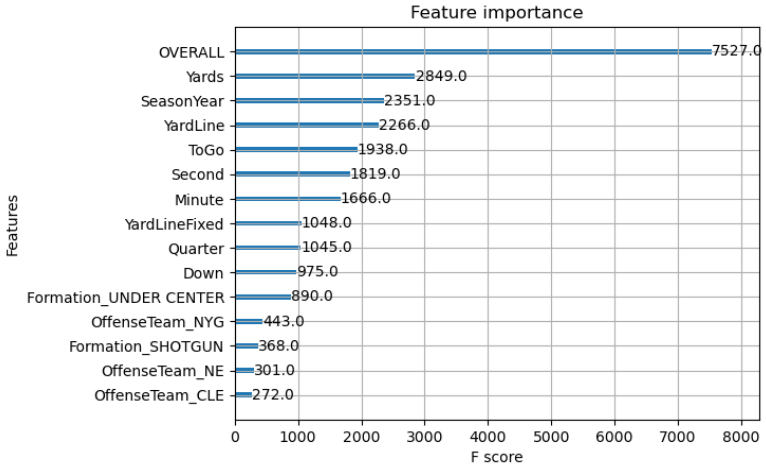

Figure 9

For the last model trained, a different approach from the ensemble tree learning was taken. The last model is a Neural Network MLP Classification model. This model performed the second best of the above, with an accuracy of 92%. The precision for this model was a very strong 93% which is the best of the above models. The recall for this model is 94% which is another strong metric. The rest of the relevant metrics are shown below in *Figure 10.* When looking at the training stats compared to the testing stats, there is no reason to believe that there is overfitting going on within the model, as all stats are within a 6% variation from training to testing, with all of the training scores being better than the testing which is to be expected. The MLP Classifier performing worse than the XGBoost but better than the random forest models could be due to a few things. The dataset could still be relatively small for the neural network to appropriately make all of the connections that it needs too. However, this shows the robustness and strength of the XGBoost and tree models when compared to neural networks. Even though the MLP classification model did not prove to be quite as accurate as the XGBoost model, it could still prove to be a useful model especially if the dataset were to be expanded upon. The confusion matrix for this model proves that it can still be useful even in its current state, as shown in *Figure 11*.

	Accuracy	Precision	Recall	F1 Score	AUC

Training	0.98	0.98	0.99	0.98	1
Testing	0.92	0.93	0.94	0.93	0.97

Figure 10

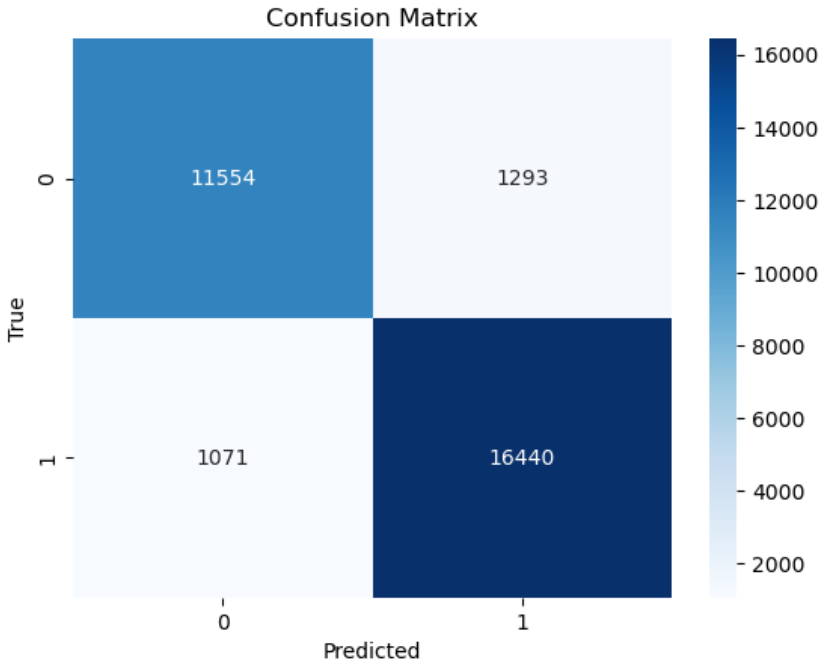

Figure 11

Between all models, the KNN model performed the best with near-perfect accuracy, precision, recall, and F1 scores in both the training and test data. However, concerns about overfitting are present due to the too-good-to-be-true performance. Both logistic regression models, while being functional, showed comparatively lower performance than our other models with an accuracy of 73%, suggesting more complex models might be suitable for this dataset. Random Forest and XGBoost, the tree-based models, both performed very well. The Random Forest model showed high accuracy and other evaluation metrics, but these metrics also pointed to possible overfitting. On the other hand, the XGBoost model proved it was better suited for this dataset, outperforming the Random Forest model and showing high accuracy without significant overfitting. Its feature importance analysis provided insight into play-calling decisions by highlighting the role of player ratings and game situations. The Neural Network MLP Classification model, while it didn't reach the high scores of the XGBoost, still performed well with high accuracy and precision, showcasing its potential usefulness, particularly as the dataset is expanded as more NFL seasons are played. The performance difference between the XGBoost and MLP Classifier could be attributed to the dataset size or the inherent strengths of tree-based models in handling such data.

Conclusion

In conclusion, this project has successfully demonstrated the viability of using machine learning models to predict NFL play calls with a high degree of accuracy. Among the models developed, the XGBoost model emerged as the most effective, balancing high accuracy with strong generalization. Its ability to both handle complex data structures and provide insights into feature importance through feature analysis makes it particularly suitable for this application. The KNN model, while achieving near-perfect scores, raised concerns about overfitting, highlighting the importance of not only model accuracy but also the ability to generalize to unseen data. The Logistic Regression models, while functional, were outperformed by more complex models, suggesting the need for advanced modeling techniques. The Random Forest model showed commendable performance but hinted at overfitting, which was less pronounced in the XGBoost model. The MLP Classifier with its neural network approach, also performed admirably, indicating potential for further improvement, especially with an expanded dataset.

Looking forward, the robustness and adaptability of models like XGBoost and MLP Classifier suggest a promising direction for predictive analytics in the NFL. With the continuing evolution of machine learning techniques and the growing availability of sports data, there is potential for even more accurate predictions in the future. Future research could explore the integration of additional data sources, more sophisticated feature engineering, and the application of these models in real-time scenarios. The success of this project lays a strong foundation for further exploration and innovation in the field of sports analytics.