Optimizing Prediction Accuracy of Advanced Ensemble And Voting Classifier Methods

Ashray Pamula
Dougherty Valley High School
San Ramon, CA, USA

ABSTRACT

Machine learning (ML) models have continued to advance over the last few years, allowing for better performance on various tasks like classification, regression, and natural language processing. One relatively powerful model architecture is the Voting Classifier, which is able to combine the individual predictions of different ML models to arrive at one output prediction. Harnessing this method can produce more robust ML models that are less influenced by noise and have higher prediction accuracy. This approach can further revolutionize the future of Artificial Intelligence (AI) and Machine Learning (ML).

This paper discusses the methods of tuning various machine learning models and how they can be combined to create a complex model that uses NFL data from the past 18 years to predict the outcomes of matchups between any two competing teams to beat the established Vegas prediction accuracy of 66.23% for the specific timeframe. Through an extensive study of various existing models and key parameters that are critical to the sport of American football, a sophisticated ensemble model was developed that is highly sensitive to a handful of most important parameters in accurately predicting a given matchup at a level that consistently matches or beats Vegas predictions.

INTRODUCTION

Predictions in sports have long existed. However, the advent of analytics in the field has accelerated the prime focus on match outcomes for a variety of stakeholders. The use of complex mathematical models combined with the power of predictive modeling (with ML and AI) have set the stage for advanced predictive analytics that are otherwise challenging with millions of variables that are changing during high-level professional sports.

Advanced models that can "beat the odds" for financial gain not only help sports bettors but also help front-office personnel in sports organizations determine the statistics that are most critical to winning, allowing them to make advantageous off-season moves.

In this research project, sophisticated Artificial Intelligence and Machine Learning techniques were used to create a new-and-improved model that leveraged past statistics to predict the outcome of any NFL game over the last 18 years.

By training various machine learning models on past NFL data and then combining them in a way that consistently ensures a high level of precision and accuracy in match predictions between any two teams, the new advanced machine learning model delivers consistent and accurate predictions that are a close match to Vegas' prediction accuracy for the NFL games.

PROCEDURES

A rich dataset of tabular football data generated by compiling the boxscores and various statistics from the last 18 years of NFL games is used against multiple machine learning models to predict outcomes. A sophisticated model is then created using each of the individual models in a voting system. The results from this advanced model are compared against the statistics from Vegas' historical accuracy over the same period.

DATA ANALYSIS

Initial analysis involved training various machine learning models on a rich dataset from 18 years of NFL data to compare the classification accuracy for match-ups to determine the best performing models. Next, leveraging hyperparameter optimization, I tuned and optimized the models to achieve the highest test accuracy. Finally, I combined these models to create a complex model that can deliver consistent precision and accuracy in predicting outcomes between any two team match-ups and benchmarked it against Vegas' accuracy. Vegas gives a line for each game every week, indicating which team they believe will win in the process. Using an online dataset of these lines, I determined Vegas' total prediction accuracy over the same timespan my NFL dataset covered.

BACKGROUND

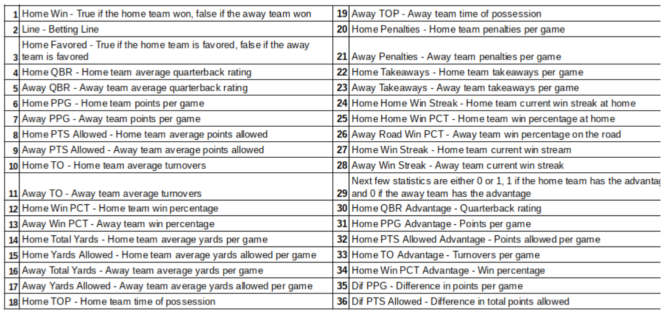
The NFL is the largest professional American Football league. The objective of the game is to earn points by advancing the ball, a process that is measured in yards. There are a variety of ways to score, including Touchdowns, Extra Points, Field Goals, and Safeties. In order to score, teams typically need to gain yards. Doing so will allow them to get closer to their targets, giving them the opportunity to get into scoring positions and score points.

Over the last 20 years, the rules around football have changed. The structure of the regular season and playoffs is no exception. For much of the last 20 years, teams played 16 regular season games in a total of 17 weeks. They were given 1 bye week in which they didn't have to play a game. However, recent rule changes have made it such that for the last 2 years, teams are playing 17 games in 18 weeks. However, this is not applicable to my study as I am only using data before this change was implemented.

Football involves many different players, positions, and statistics. In order to win a game, teams typically have to perform better in a majority of these statistics. The reason predicting football games is so difficult is because of the random variation that exists in any sport. Teams that tend to do poorly may have a good week and beat a statistically better team. In other situations, a team could be dominated in most statistics, but perform better in a select few that allows them to win the game. These intricacies are the main challenge to building a flawless model, and for this reason, Vegas isn't able to predict outcomes at an extremely high accuracy.

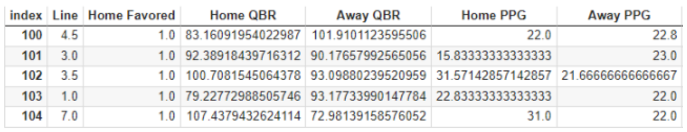
DATASET

The dataset in this project has statistics on every game played in the NFL from 2002-2019. For each game, there is information such as teams playing and status of home/away teams. There is also more in depth data such as home and away team passing yards, rushing yards, penalties, and quarterback rating. Lastly, there are binary features based on advantages in categories such as turnovers and home/away win percentage. This data was scraped from BigDataBowl and OddsShark.



1	Home Win - True if the home team won, false if the away team won	19	Away TOP - Away team time of possession
2	Line - Betting Line	20	Home Penalties - Home team penalties per game
3	Home Favored - True if the home team is favored, false if the away team is favored	21	Away Penalties - Away team penalties per game
4	Home QBR - Home team average quarterback rating	22	Home Takeaways - Home team takeaways per game
5	Away QBR - Away team average quarterback rating	23	Away Takeaways - Away team takeaways per game
6	Home PPG - Home team points per game	24	Home Home Win Streak - Home team current win streak at home
7	Away PPG - Away team points per game	25	Home Home Win PCT - Home team win percentage at home
8	Home PTS Allowed - Home team average points allowed	26	Away Road Win PCT - Away team win percentage on the road
9	Away PTS Allowed - Away team average points allowed	27	Home Win Streak - Home team current win stream
10	Home TO - Home team average turnovers	28	Away Win Streak - Away team current win streak
11	Away TO - Away team average turnovers	29	Next few statistics are either 0 or 1, 1 if the home team has the advantag and 0 if the away team has the advantage
12	Home Win PCT - Home team win percentage	30	Home QBR Advantage - Quarterback rating
13	Away Win PCT - Away team win percentage	31	Home PPG Advantage - Points per game
14	Home Total Yards - Home team average yards per game	32	Home PTS Allowed Advantage - Points allowed per game
15	Home Yards Allowed - Home team average yards allowed per game	33	Home TO Advantage - Turnovers per game
16	Away Total Yards - Away team average yards per game	34	Home Win PCT Advantage - Win percentage
17	Away Yards Allowed - Away team average yards allowed per game	35	Dif PPG - Difference in points per game
18	Home TOP - Home team time of possession	36	Dif PTS Allowed - Difference in total points allowed

Table 1: Football Game Statistics Used in Modeling



index	Line	Home Favored	Home QBR	Away QBR	Home PPG	Away PPG
100	4.5	1.0	83.16091954022987	101.9101123595506	22.0	22.8
101	3.0	1.0	92.38918439716312	90.17657992565056	15.83333333333333	23.0
102	3.5	1.0	100.7081545064378	93.09880239520959	31.57142857142857	21.66666666666667
103	1.0	1.0	79.22772988505746	93.17733990147784	22.83333333333333	22.0
104	7.0	1.0	107.4379432624114	72.98139158576052	31.0	22.0

Table 2: Dataset sample imported into code editor

Overall, there are 4485 samples and 35 features. Using principal component analysis, I found that the ideal number of features for this dataset was 32. I then limited my dataset to the 32 most impactful features automatically using the same method. Although 32 was the best number, I wanted to ensure that analyzing the data would still be time-friendly and practical for a computer without the power of computers accessible to professionals. Therefore, I used a variance threshold to remove features with low variance. This is helpful because it downsizes the dataset while also removing features that have a lack of variety and are not helpful to my models.

To train each model, the data was split into two parts with 75% being used to train the model and 25% being used to test the model. However, in evaluating yearly accuracies, the data was split such that the model was only tested on the data points that were from that year's NFL games. This allowed me to view how well the model performed in any given year and compare these accuracies to Vegas.

METHODOLOGY/MODELS

In order to predict these NFL games, I took the approach of finding a variety of classification models that would work to solve the problem at hand. Throughout this researching process, I learned more about models I had already known while also discovering novel machine learning models that were developed more recently.

A total of 13 models were used in this process: Logistic Regression, K-Nearest Neighbors, Decision Tree, Random Forest, Support Vector Machine (SVM), GaussianNB, BernoulliNB, AdaBoost, Gradient Boosting, CatBoost, XGBoost, LightGBM, and Multilayer Perceptron (MLP).

Each of these models used the same training, testing, and prediction process. I tried to use a variety of models to ensure that different types of prediction approaches would be represented. There were basic machine learning models, boosting models, naive-bayes models, and neural networks.

Each of these methods was found to have varying prediction accuracies from year to year. The boosting models tended to perform the best across the years, while the more basic models only occasionally had very high accuracies.

After each of these models was tested individually, I compiled them into a more advanced method known as a Voting Classifier. This method allows for multiple machine learning models to work together to classify data to a desired output.

This type of model allows for hard and soft voting. Hard voting means that each inputted model's prediction is weighed the same and the final output is determined through 'counting the votes.' Soft voting allows for the manual coding of 'weights' that can change the importance of models.

While experimenting with this model, I found that hard voting was consistently the best while soft voting often decreased accuracy. This is likely due to many of the models having similar accuracies. Weighting one model over another didn't end up benefiting the overall model as it led to some models that typically performed worse to be entirely ignored in certain cases.

Another reason soft voting was bad is that it took away from the purpose of my voting classifier. The reason I used so many different models was to prevent underfitting in prediction approaches. Different models take different steps to predict results. Therefore, I wanted to harness all of these modeling types to allow for the best cumulative accuracy. For example, a comparatively basic model like Logistic Regression may typically be worse than a boosting model like CatBoost, but in certain situations, Logistic Regression may perform better. Hard voting allowed for such discrepancies to matter in end predictions.

HYPERPARAMETER TUNING

In order to optimize each model before combining them into the voting classifier, I used a process known as hyperparameter tuning. In this process, I chose a few parameters and found which value for the parameters led to the greatest prediction accuracy for each model. After completing this process, I was able to tune every model to have the highest possible prediction accuracy, allowing for the final ensemble voting classifier to also have its highest prediction accuracy possible.

For every applicable model, I set the random_state parameter to 0 to ensure that all runs of the code would be reproducible as possible. This helped me tune the other parameters more effectively as it prevented chance processes from disrupting my attempts to increase each model's prediction accuracy.

Model	Parameter	Parameter	Parameter	Parameter	Parameter
Logistic Regression	C = 0.14	random_state = 0			
K-Nearest Neighbors	n_neighbors = 58				
Decision Tree	max_depth = 1	min_samples_split = 12	random_state = 0		
Random Forest	n_estimators = 417	max_depth = 3	min_samples_split = 2	random_state = 0	
Support Vector Classifier	C = 173	kernel = "poly"	degree = 2	probability = True	random_state = 0
Gradient Boosting	learning_rate = 0.03	n_estimators = 46	min_samples_split = 34	max_depth = 5	random_state = 0
Extreme Gradient Boosting	learning_rate = 0.081	n_estimators = 100	max_depth = 1	silent = True	
Light Gradient Boosting	learning_rate = 0.166	n_estimators = 90	max_depth = 1		
CatBoost	learning_rate = 0.04	iterations = 1000	depth = 1		
AdaBoost	learning_rate = 0.26	n_estimators = 8	random_state = 0		
GaussianNB					
BernoulliNB	alpha = 194	fit_prior = False			
Multilayer Perceptron	alpha = 60	max_iter = 48	random_state = 0		

Table 3: Parameters and hyperparameters used in each machine learning model

SIMPLE MODELS

First, I tuned the more 'simple' models that I had used in the past in other AIML projects. These models were Logistic Regression, KNN, Decision Tree, Random Forest, and SVC.

For Logistic Regression, I used only one hyperparameter: C. C, which is known as a Regularization Parameter, determines how much to weigh training vs testing error when making the model. In this case, C = 0.14 ensured that prediction accuracy was as high as possible.

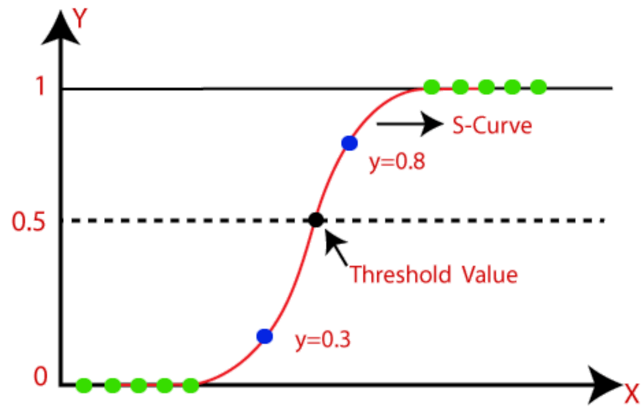


Figure 1: Logistic Regression

For KNN, I used the parameter of n_neighbors. This parameter looks at data points that are closest to a given input and classifies based on what the majority of the surrounding data points output to. For my model, n_neighbors = 58 was most effective.

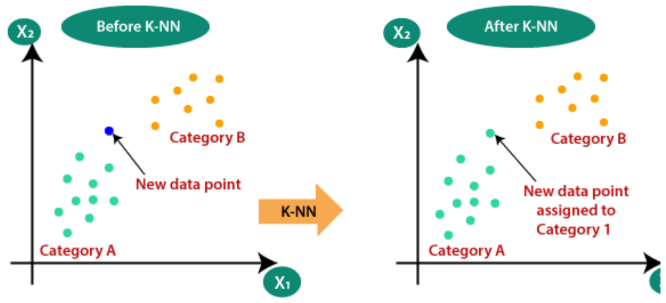


Figure 2: KNN

The next model I used was Decision Tree. I used the parameters of max_depth and min_samples_split. Max_depth represents the longest distance from the start of the tree to the furthest node. In this case, max_depth = 1 was best. Min_samples_split represents the minimum number of samples with the same attribute to validate a split to be created. The best value was min_samples_split = 12.

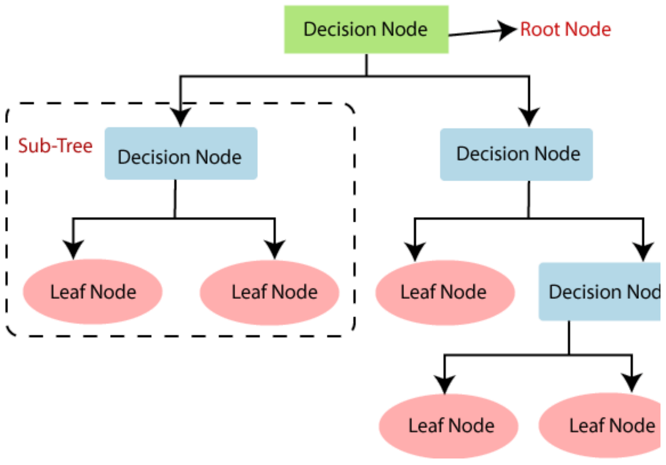


Figure 3: Decision Tree

I then used Random Forest. This model is essentially a group of many decision trees. I used the same max_depth and min_samples_split parameters, but I also used the hyperparameter of n_estimators which represents the number of decision trees to use in your forest. The best values were as follows: n_estimators = 417; max_depth = 3; min_samples_split = 2.

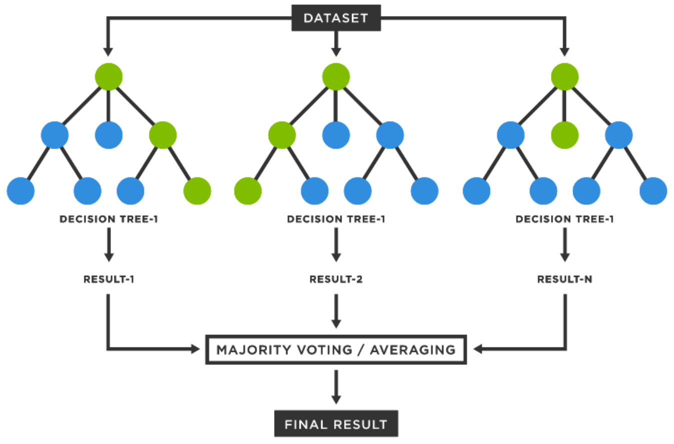


Figure 4: Random Forest

The last of the 'simple' models I used was a Support Vector Classifier. I used the parameters of C, kernel, degree, and probability. Like its use in Logistic Regression, C represents the importance given to train vs test error. Kernel is used to transform data into a required processing format. Degree is used to set the degree for the polynomial kernel function. It can only be used with the kernel type 'poly.' Probability allows for the enabling of probability estimates on the model. I used the following values: C = 173, kernel = "poly", degree = 2, probability = True.

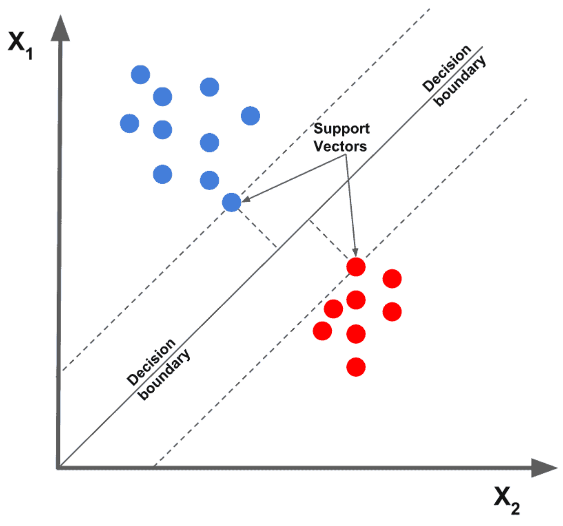


Figure 5: Support Vector Machine

BOOSTING MODELS

The next set of models I looked at were Boosting models. These models take a base model, typically something like Decision Tree, and boosts it in a variety of ways. The goal of boosting is to turn a weak learner, which has low prediction accuracy, into a strong learner, which has higher prediction accuracy. It does so by sequentially training a system of models into a single model that is able to use the highly accurate parts of each individual model to predict more accurately. In doing so, such models are able to boost the performance of a more basic type of model.

The first boosting model I used was Gradient Boosting, with the parameters learning_rate, n_estimators, min_samples_split, and max_depth. For boosting models, n_estimators represents the number of boosting stages to use when training the model. Learning_rate is a metric of how fast the model learns. Given that boosting models are based on trees, max_depth and min_samples_split represent the same values that they did for the Decision Tree and Random Forest modes that I used earlier. The parameter values that gave the highest prediction accuracy were; learning_rate = 0.03, n_estimators = 46, min_samples_split = 34, and max_depth = 5.

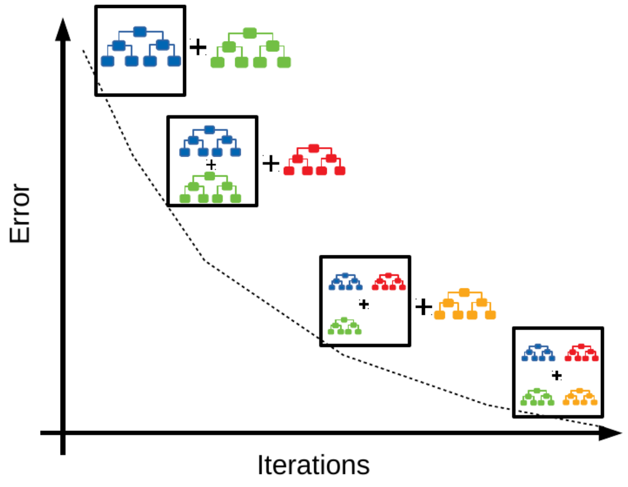


Figure 6: Gradient Boosting Machine

I then used three other models that are based on or considered forms of gradient boosting: Extreme Gradient Boosting (XGBoost), Light Gradient Boosting (Light GBM), and CatBoost. Each of these model types use Gradient Boosting but apply it in different ways. They differ most in terms of splitting methods and how they treat categorical variables. XGBoost was the first of the three to be available for public use, followed by LightGBM and then CatBoost. These three models are considered to be of the most novel machine learning algorithms that are now available and will likely become more and more popular in the near future.

I first implemented XGBoost. This model uses a pre-sorted algorithm to find split values. It only accepts numerical values as features, so I had to transform my data before using this model. I used the parameters of learning_rate, n_estimators, and max_depth, all of which had the same purpose that they did in Gradient Boosting. They had the following values: learning_rate = 0.081, n_estimators = 100, max_depth = 1.

I then implemented LightGBM. This model uses Gradient-based One-Side Sampling, a novel method to filter through data to find the best split values. This process keeps all instances with large gradients (which are important as gradient is a measure of error, so large error means that sample is likely influential for the model) while randomly selecting some instances with smaller gradients to increase the overall efficiency of the model. LightGBM is also able to handle categorical features and uses its own algorithm to figure out how to split them. I used learning_rate = 0.166, n_estimators = 90, and max_depth = 1.

The last boosting model I used was CatBoost. CatBoost is also able to have categorical features passed into the model. Unlike LightGBM, CatBoost uses one-hot-encoding, a process by which categorical features are converted to numbers of different values while still representing their category. The parameters I used for my CatBoost model were learing_rate, iterations, and depth. Learning_rate is the same as it was in previous models. Iterations represent the number of trees that are built and depth represents the actual depth of each of these trees. The parameters that allowed for highest predictive accuracy were learning_rate = 0.04, iterations = 1000, and depth = 1. I also used the parameter of silent = True to prevent any text output from cluttering the editor.

The other type of boosting is Adaptive Boosting, which is known as AdaBoost. This type of boosting differs from Gradient Boosting as it uses a sequential learning technique that adds predictors gradually rather than all at once. Therefore, it is able to minimize error adaptively after adding in each feature. This is a huge benefit of Adaptive Boosting, but also serves as a hindrance as the model can only be trained in the specific order that the predictors are added in.

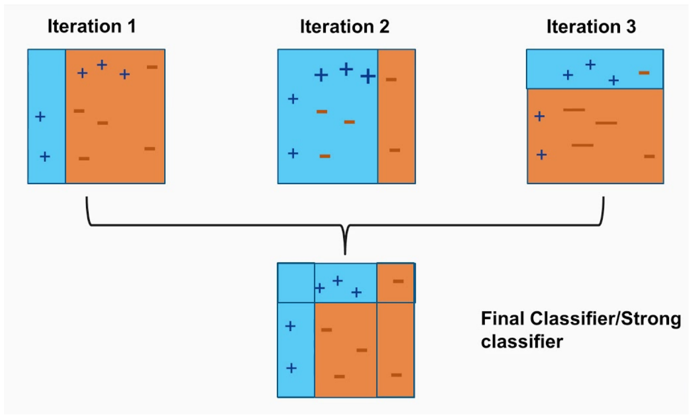


Figure 7: AdaBoost

The parameters I used for this model are n_estimators and learning_rate. These represent the same idea that they represented in each of the gradient boosting models. In my model, I used learning_rate = 0.26 and n_estimators = 8.

OTHER ADVANCED MODELS

After implementing these boosting models, I wanted to incorporate other types of models into my Voting Classifier. To do this, I used two different Naive Bayes models and a neural network.

I first implemented the Naive Bayes models. These models are based on a mathematical concept that was first introduced many centuries ago. They utilize conditional probability to determine the chance of different outcomes occurring. This is helpful in a machine learning application as we can determine what the most likely outcome is for a given testing data sample by seeing how its different feature values align with other data samples after training the model.

$$\overset{\text{Posterior}}{P(A|B)} = \frac{\overset{\text{Likelihood}}{P(B|A)}\overset{\text{Prior}}{P(A)}}{\underset{\text{Normalizing constant}}{P(B)}}$$

$$P(B) = \sum_{Y} P(B|A)P(A)$$

Figure 8: Naive Bayes

The first Naive Bayes model I used was GaussianNB. This model is specifically used for continuous data. I didn't specify any parameter values when creating this model and instead let the model use its default values.

I also used BernoulliNB as a part of my ensemble classifier. This form of Naive Bayes is typically used for features with binary outcomes. Although some of my data was continuous, I was able to transform it to fit the requirements of this model. Given that some features were boolean, I wanted to use this model to see if these features held any more importance in predicting which team would win. I used two parameters; alpha and fit_prior. Alpha is a smoothing hyperparameter that removes extreme variation/noise from the dataset. Fit_prior is a boolean value that determines if the model should learn existing probability distributions of features. If this value is false, the model will default to assuming each feature has a uniform distribution. I used the values alpha = 194 and fit_prior = False.

The neural network I used was a Multilayer Perceptron (MLP). Like other neural networks, MLPs have an input layer, hidden layer, and output layer. The data is inputted and each feature is taken in as a separate neuron. Each neuron has an associated weight, which is picked randomly at the start, resulting in an end output number after going through a series of hidden layers. This number is used to determine the categorical output of the sample as samples that are of the same category tend to have output numbers that are close to one another.

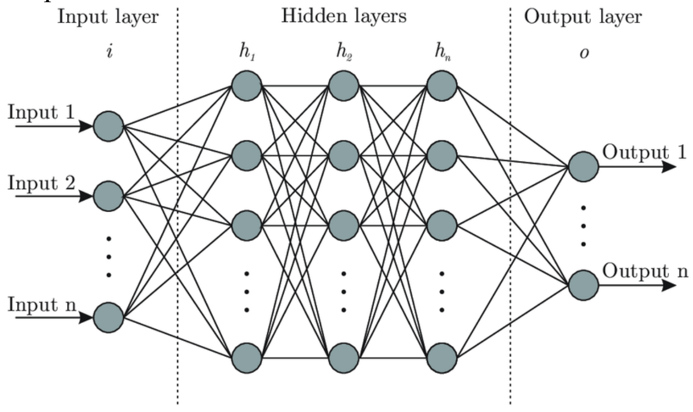


Figure 9: Neural Networks in MLP

MLPs differ from other neural networks as they can work with non-linear data as well. They also use a process known as 'backpropagation' to minimize error through adjusting each neuron's weights. All perceptrons also use the optimization function of Stochastic Gradient Descent to minimize error, allowing for the highest prediction accuracy in the end. For my MLP, I specified the parameters of alpha and max_iter. Alpha is a parameter that can counter overfitting by constraining the values that weights can take on. This ensures that the MLP will not perfectly fit training data while sacrificing accuracy of testing data. Max_iter represents the maximum number of iterations that the neural network can go through in order to optimize its weights. The parameters I found to result in the highest accuracy were: alpha = 60 and max_iter = 48.

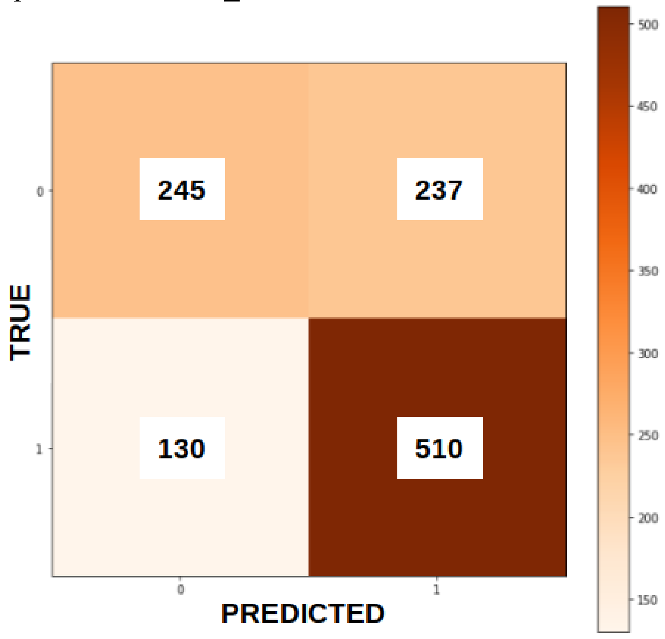
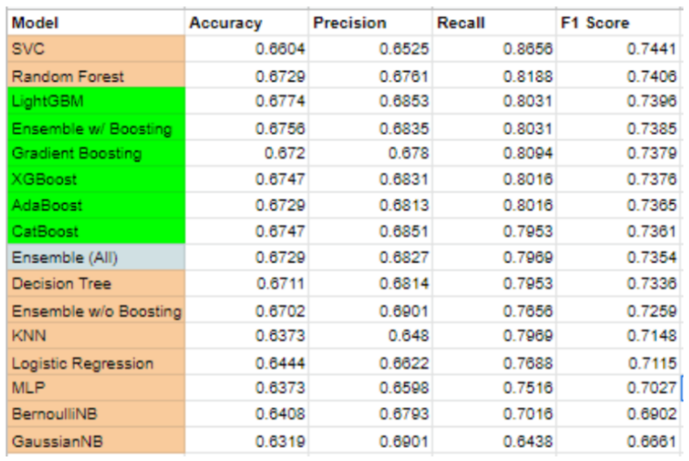


Figure 10: Confusion Matrix with Predictions from the Ensemble Voting Classifier

RESULTS AND DISCUSSION

Overall, my model beat the Vegas model by achieving higher prediction accuracy of 67.29% vs 66.23% for the same timeframe. It achieved superior performance in ~59% of the seasons (10 out of 18). Best performance was in 2005 with a 72.73% prediction accuracy, while both models had their worst performances in 2006.

After creating my ensemble model, I created two other voting classifier models, one based solely on the boosting models and one based on the non-boosting models. I did this in order to see how the voting classifier would compare when different models are passed through. Due to boosting models being a type of ensemble model, I wanted to make sure that the models were tested separately as well. The resulting models made it easier to compare accuracy, precision, recall, and f1 score among the different model types to see just how beneficial the voting classifier was.



Model	Accuracy	Precision	Recall	F1 Score
SVC	0.6604	0.6525	0.8656	0.7441
Random Forest	0.6729	0.6761	0.8188	0.7406
LightGBM	0.6774	0.6853	0.8031	0.7396
Ensemble w/ Boosting	0.6756	0.6835	0.8031	0.7385
Gradient Boosting	0.672	0.678	0.8094	0.7379
XGBoost	0.6747	0.6831	0.8016	0.7376
AdaBoost	0.6729	0.6813	0.8016	0.7365
CatBoost	0.6747	0.6851	0.7953	0.7361
Ensemble (All)	0.6729	0.6827	0.7969	0.7354
Decision Tree	0.6711	0.6814	0.7953	0.7336
Ensemble w/o Boosting	0.6702	0.6901	0.7656	0.7259
KNN	0.6373	0.648	0.7969	0.7148
Logistic Regression	0.6444	0.6622	0.7688	0.7115
MLP	0.6373	0.6598	0.7516	0.7027
BernoulliNB	0.6408	0.6793	0.7016	0.6902
GaussianNB	0.6319	0.6901	0.6438	0.6661

Table 4: Accuracy, Precision, Recall, F1 Score Results

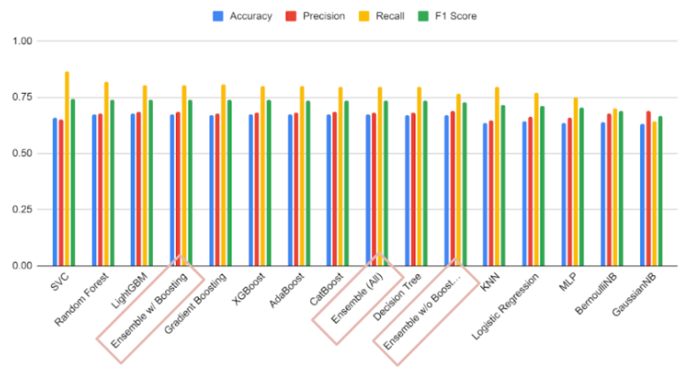


Figure 11: Accuracy, Precision, Recall, and F1 Score Results

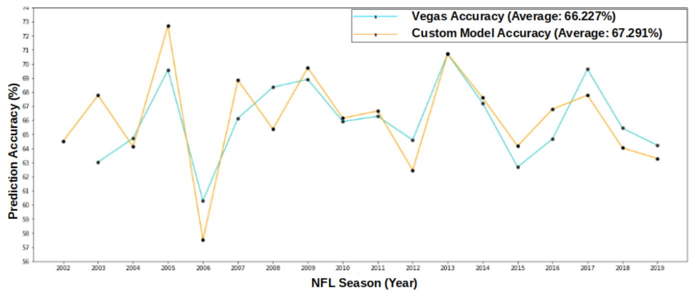
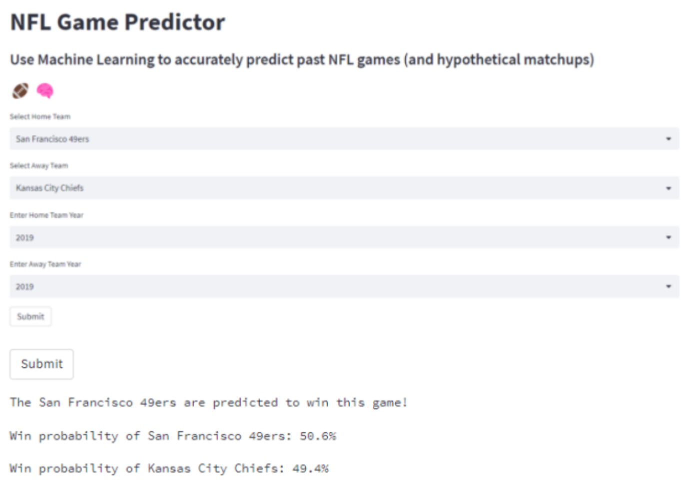


Figure 12: NFL Game Prediction of Vegas Oddsmaker vs Homemade Model

DEMONSTRATION

In order to demonstrate the model in action, I developed a web application using Streamlit, an online platform that allows for backend and frontend customization. The web application allows anyone to get a model prediction between any two NFL teams from any year in the project's timeframe. The application can be accessed online by following https://avp123-nflstreamlit-nflapp-z925xp.streamlit.app/

Once on the site, simply select the two NFL teams and the two years of interest. You will see the predicted winner along with the exact probability of that team winning.



NFL Game Predictor

Use Machine Learning to accurately predict past NFL games (and hypothetical matchups)

Select Home Team: San Francisco 49ers

Select Away Team: Kansas City Chiefs

Enter Home Team Year: 2019

Enter Away Team Year: 2019

```
The San Francisco 49ers are predicted to win this game!

Win probability of San Francisco 49ers: 50.6%

Win probability of Kansas City Chiefs: 49.4%
```

Figure 13: NFL Game Predictor Streamlit App

CONCLUSIONS

Despite technological constraints such as access to sophisticated datasets, large computing power, or advanced algorithms that professionals might have, developers like myself are now able to predict at accuracies comparable to those of large corporations due to open-source ML/AI packages like scikit-learn. The accessibility of modeling techniques have bridged the gap that has long existed between the ‘experienced team of professional programmers’ and the ‘high school solo programmer.’

Such Voting Classifier methods are extremely beneficial in solving ambiguous problems that require multiple, complex classification models. By harnessing the abilities of different models, novel approaches can be taken to maximize prediction accuracy of any classification task. This is especially apparent when looking at predictions spanning a large amount of time. In my project, any single model fluctuated greatly in its accuracy over different years. Due to the constantly changing nature of the game of football, each model reacted differently to new statistics from each year. However, the Voting Classifier was able to minimize such errors by taking a more holistic view of predictions and deciding upon the most agreed-upon result. Using such methods at less granularity can allow for an increase in accuracy and more consistent results for those in any field wanting to utilize Artificial Intelligence and Machine Learning.

Given the efficacy of this Voting Classifier, such methods should also become more common in other industries, such as public health and education. High predictive accuracy is necessary when creating life-saving solutions and educating future generations, so applying this ensemble model to other use cases is a logical extension of this research. I plan to further my research on this innovative model combining classifiers through additional datasets in the aforementioned industries.

ACKNOWLEDGMENTS

I would like to thank my mentor, Christopher Mauck, for guiding me on this research and inspiring me to continue my journey with Artificial Intelligence and Machine Learning. He helped me collect the data and brainstorm ways to predict the outcomes of NFL games. His knowledge of machine learning combined with his experience as the quarterback for MIT's football team have been immensely valuable in delving deeper into the technical concepts and applicability to NFL sports data.

References

Brown, Christophe. "Simple Modeling of NBA Positions Using the K-Nearest Neighbors Machine Learning Algorithm." *Medium*, Towards Data Science, 6 Nov. 2021, https://towardsdatascience.com/simple-modeling-of-nba-positions-using-the-k-nearest-neighbors-machine-learning-algorithm-223b8addb08f.

Mir, Saadan. "NBA Draft Analysis: Using Machine Learning to Project NBA Success." *Medium*, Towards Data Science, 10 Feb. 2022, https://towardsdatascience.com/nba-draft-analysis-using-machine-learning-to-project-nba-success-a1c6bf576d19.

"NFL Odds History." *NFL Football Odds & Line History on TeamRankings.com*, https://www.teamrankings.com/nfl/odds-history/results/?season-filter=2021.

"1.11. Ensemble Methods." Scikit, https://scikit-learn.org/stable/modules/ensemble.html#voting-classifier.

Staff, Odds Shark, and Benjamin Eckstein. "Odds Shark." *Odds Shark*, https://www.oddsshark.com/.

"NFL Big Data Bowl." *NFL Football Operations*, https://operations.nfl.com/gameday/analytics/big-data-bowl/.

"NFL Odds History." *NFL Football Odds & Line History on TeamRankings.com*, https://www.teamrankings.com/nfl/odds-history/results/.

panelFerathKherifAdeliyaLatypova, Author links open overlay, et al. "Principal Component Analysis." *Machine Learning*, Academic Press, 15 Nov. 2019, https://www.sciencedirect.com/science/article/pii/B9780128157398000122.

"1.13. Feature Selection." *Scikit*, https://scikit-learn.org/stable/modules/feature_selection.html#feature-selection.

Gera, Divya. "Boosting Algorithms: AdaBoost, Gradient Boosting, XGB, Light GBM and CatBoost." *Medium*, Medium, 9 Sept. 2020, https://medium.com/@divyagera2402/boosting-algorithms-adaboost-gradient-boosting-xgb-light-gbm-and-catboost-e7d2dbc4e4ca.

How SVM Works, https://www.ibm.com/docs/en/spss-modeler/saas?topic=models-how-svm-works.

"Multilayer Perceptron." *How Do Multilayer Perceptrons Help Solve Complex Problems?*, https://h2o.ai/wiki/multilayer-perceptron/.

"1.9. Naive Bayes." *Scikit*, https://scikit-learn.org/stable/modules/naive_bayes.html.

Alpaydin, Ethem. "Machine Learning." *Amazon*, The MIT Press, 2021, https://docs.aws.amazon.com/machine-learning/latest/dg/training-parameters.html.

Huilgol, Purva. "Accuracy vs. F1-Score." *Medium*, Analytics Vidhya, 24 Aug. 2019, https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2.

"Designing Your Neural Networks." KDnuggets, https://www.kdnuggets.com/2019/11/designing-neural-networks.html.

Mallick, Satya. "Support Vector Machines (SVM): Learnopencv #." *LearnOpenCV*, 4 May 2021, https://learnopencv.com/support-vector-machines-svm/.

Pal, Aratrika. "Gradient Boosting Trees for Classification: A Beginner's Guide." *Medium*, The Startup, 2 Oct. 2020, https://medium.com/swlh/gradient-boosting-trees-for-classification-a-beginners-guide-596b594a14ea.

packt1000. "Extending Machine Learning Algorithms – Adaboost Classifier | Packtpub.com." *YouTube*, YouTube, 7 Dec. 2017, https://www.youtube.com/watch?v=BoGNyWW9-mE.

"Decision Tree Algorithm in Machine Learning - Javatpoint." *Www.javatpoint.com*, https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm.

“What Is a Random Forest?” *TIBCO Software*, https://www.tibco.com/reference-center/what-is-a-random-forest.

Stanina, Iuliia. “Implementing Naive Bayes Algorithm from Scratch - Python.” *Medium*, Towards Data Science, 11 Dec. 2021, https://towardsdatascience.com/implementing-naive-bayes-algorithm-from-scratch-python-c6880cfc9c41.

“Logistic Regression in Machine Learning - Javatpoint.” *Www.javatpoint.com*, https://www.javatpoint.com/logistic-regression-in-machine-learning.

“K-Nearest Neighbor(KNN) Algorithm for Machine Learning - Javatpoint.” *Www.javatpoint.com*, https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning.