DTSC 691
Machine Learning
Project Submission
Predicting NFL Game Outcomes
Nicole Gordon

Goals of the project

This project aims to combine machine learning with software development to create a web application that allows users to easily interact with a trained model. The machine learning model predicts the outcome of NFL games based on a subset of in-game statistics. The model takes in the current game conditions (eg. current scores of the home and away teams, which team has possession, quarter, etc.) as predictors and the response is the probability that the home team will win the game. This type of model would be useful for both sportsbooks and individual teams. Sportsbooks could use similar models to help set the spread, while teams and other sports analysts could use it to track their win-probability throughout the game.

In addition to creating a model with a simple user interface, the research goal of this project is to determine which features are most important for predicting the outcome of a game. Any given game could have hundreds to tens of thousands of data points, depending on how in-depth the statistics are. A recent development in the field of football analytics is the NFL's Next Gen Stats player tracking. A high fidelity tracking system captures the movement of players, which provides increasingly complicated data compared to static statistics due to the four-dimensional nature (three spatial dimensions plus time). Although this data and analysis is outside the scope of this project, it highlights the enormous scope of the data that is available to analysts and provides the motivation to identify the most predictive features.

The data for this will come from a dataset compiled by a group of statistical researchers from Carnegie Mellon University which is available on Kaggle and Github. Although NFL entities like Next Gen Stats take extensive game data, there is a lack of comprehensive, publicly available NFL data, especially compared to other major sports. This makes it more difficult for analysts not affiliated with the league to perform research in football analytics. However, a team at Carnegie Mellon created a package to scrape and clean data from the NFL website and compiled it into data sets containing information about play-by-play statistics and final game outcomes.

Using this data, a classification model was built that returns the probability of how likely the team is to win the game. The dataset has over 100 columns, so feature engineering and dimensionality reduction was performed to extract the most important features. Multiple model types were tested and grid searches were used to find the optimal hyperparameters before selecting the final model. The models were evaluated based on how accurate they are at predicting game outcomes. After the final model was chosen, trained, and evaluated, an interface was created that will make the model accessible to users.

The final product is a web application where a user can input the current game conditions and a prediction of the game outcome will be output. Although the UI is simple and easy to navigate, the goal is to create a UX design that is also presentable and professional.

Data description

The data for this project was collected by the NFL but was compiled by a team from Carnegie Mellon by using an API provided by the NFL. The team includes Maksim Horowitz, Ron Yurko, and Sam Ventura. The data was collected over the 2009-2019 NFL seasons. The full data includes regular season, preseason, and postseason. This project will only use regular season data and will only aim to make predictions on regular season games. Preseason games are often not representative of regular season games because they are often used to evaluate potential players, so the rosters vary greatly compared to the regular starters during the season. Postseason games are also not representative of regular season games because of the added stakes of the game and the single-elimination playoff structure, which may influence what decisions are made.

The data will be accessed from Github and the games_data and play_by_play_data folders will be used. Data for each year is contained in a separate csv file, so they will be read in and concatenated to form the full data sets. According to Kaggle the data has already been cleaned and parsed, so there should be only small amounts of cleaning and imputation necessary, but this will be double checked. So far I have not found a data dictionary.

The play_by_play_data has 256 columns. Many of these columns detail the result of the play or analysis by the authors. This project is only interested in the current game state, so the data will be subset and the 26 of the columns will be used. These are:

‘play_id', 'game_id', 'home_team', 'away_team', 'posteam',
'posteam_type', 'defteam', 'side_of_field', 'yardline_100',
'game_date', 'quarter_seconds_remaining', 'half_seconds_remaining',
'game_seconds_remaining', 'game_half', 'quarter_end', 'drive', 'sp',
'qtr', 'down', 'goal_to_go', 'time', 'yrdln', 'ydstogo',
'total_home_score', 'total_away_score', 'score_differential'

There are 498,393 total observations in the play_by_play data. In the initial data exploration, plays where the down number was undefined were dropped. This included beginning of the game, half time, the end of the game, two minute warnings, timeouts, and kickoffs. Plays where

there was no defined possession team were also dropped. Next, dimensionality reduction was performed. Columns that provided redundant or no information were dropped. This included ‘play_id’, 'game_date', 'side_of_field', 'yrdln', 'time', 'goal_to_go', 'sp', 'posteam', 'defteam', 'quarter_end'. A feature like ‘play_id’ is only an index so it would provide no information to the model. A feature like ‘posteam’ (which team has possession) is redundant because the column ‘posteam_type’ (if home or away has possession) conveys the same information.

The play_by_play_data contains “advanced metrics such as expected point and win probability values” (Kaggle) that were calculated by the team but these columns will not be used in my analysis. There is some legacy R code and code from a workshop they organized on Github but these will also not be used.The code from the workshop focuses on quarterback and receiver performance, which is not a part of my project.

The games_data has 10 columns. The data will be subset based on the columns that identify the game details and the outcome of the game. These six columns are:

‘game_id’, 'home_team', 'away_team', 'week', 'season', 'home_score', 'away_score'.

There are 2,816 total observations in the games_data. The only feature that was dropped was ‘season’.

Both the play_by_play_data and games_data share the game_id primary key, so an inner join was performed on these two tables after the dimensionality reduction. Next, the data was manipulated so that instead of each row containing information about both teams during that play, each row only contained information about either the home or the away team on each play. This process also included scrubbing the data of specific team names and instead only using labels of ‘home’ and ‘away’. Features were renamed to make them more intuitive. For example, ‘total_home_score’ from the play-by-play data reflects the home team’s total score on that play. This was renamed to ‘cur_home_score’ because it is the home team’s current score at that moment in the game. From the games data, the feature ‘home_score’ was renamed to ‘final_home_score’ because it reflects what the home team’s score was at the end of the game. After the data was split and stacked, the final dataset had twice and many rows. Feature engineering was also performed during this part of the project. A feature called ‘home_won’, which is a boolean variable with a value of 1 if the home team won and 0 if the home team lost, was added by determining if ‘final_home_score’ was greater than ‘final_away_score’. A feature called ‘home_possession’, which is a boolean variable with a value of 1 if the home team has possession and 0 if the home team does not have possession, was added by determining if ‘posteam_type’ was equal to ‘home’. Finally, one-hot encoding was used to transform the categorical feature ‘game_half’, which possessed values of ‘Half1’, ‘Half2’, and ‘Overtime’. The data was exported to a csv to use for preprocessing, training and testing.

In the preprocessing stage, the dimensionality was further reduced by dropping the features 'game_id', 'final_home_score', 'final_away_score'. The final scores were initially kept because they were going to be the labels for a regression model to predict the final score, but then I decided to create a classification model that classifies the home team as winning or losing, so

these variables were no longer needed. Finally, features that were highly correlated with each other or that had extremely low correlation with the target variable were dropped.

Model Specifications

The six features in the data were 'qtr', 'cur_home_score', 'cur_away_score', 'score_differential', 'week', 'home_possession'. The final target variable was ‘home_won’. There were 421,910 observations in the final data set. The goal is to create a classification model that can accurately predict if the home team will win based on the current game statistics.

Multiple classification models were developed, tuned, trained and tested in Python in order to find the most optimal features and model to predict game outcomes. The types of models are logistic regression, random forest, SVM, KNN, and neural network. The data was split into a training and a test set and saved as CSVs prior to the model creation so that the same training and test sets could be used for each model. Grid searches with cross-validation were performed for each model type. Accuracy is used as the performance metric for all of the models. The accuracy reflects how often the model correctly predicts that the home team will win. For reference, predicting the home team won each time would yield an accuracy of 0.5630. All of the models improved significantly on this accuracy.

For the logistic regression a grid search was performed for the C value and the solver. Three-fold cross validation was used.The final hyperparameters were a C value of 0.15 and the newton-cg solver. Newton-cg was tested because the number of samples was much greater than the number of features, and the model was a binary classifier. The accuracy of the final logistic regression on the test set was 0.7471.

For the random forest a grid search was performed for the max depth, number of estimators, and minimum number of samples per split. Five-fold cross validation was used. The final hyperparameters were a max depth of 38, 110 estimators, and a minimum of 10 samples per split. The accuracy of the final random forest on the test set was 0.8206.

For the SVM, the training data was scaled using skelarn’s StandardScaler prior to model creation. A grid search was attempted to find the best kernel and C value. However, since the memory required to store the kernel scales quadratically with the number of data points, this dataset was too large for a SVM to be feasible and the training time required would have been too long. Due to this, SVM models were not successfully created.

For the KNN a grid search was performed for the number of neighbors. Five-fold cross validation was used. The final hyperparameter was 7 neighbors. The accuracy of the final random forest on the test set was 0.7986.

For the neural network the training data was scaled using skelarn’s StandardScaler prior to model creation. A grid search was performed for the number of hidden layers, the number of neurons per layer, and the learning rate. Three-fold cross validation was used with 3 iterations

and 100 epochs. The model had an input layer which flattened the input data, and a dense output layer with one neuron and a sigmoid activation function since it is a binary classifier and the output should be a probability between 0 and 1. A binary cross-entropy loss function was used and the metric used was accuracy. There was early stopping used with a patience of 10 in order to speed up training, but it was never hit. The final hyperparameters were 2 hidden layers, 30 neurons, and a learning rate of about 0.0066. The accuracy of the final neural network on the test set was 0.7490.

The final model chosen was the random forest. It was chosen because it provided the best accuracy on the test set. The other benefit of the random forest is that it doesn't require feature scaling, so the current game data that is provided by the user can be directly input into the model.