Introduction

A variety of studies have attempted to use machine learning to accurately predict the outcome of NFL games. Because there are a wide variety of player and game statistics available and increased relevance of statistics and machine learning to both predict results and make in-game decisions, this is a compelling domain. We primarily relied on a paper by Jim Werner, which outlined his attempt to select features and a machine learning model that could outperform the "Vegas Line," which is used to predict the point spread in games. *See* Jim Warner, *Predicting Margin of Victory in NFL Games: Machine Learning vs. the Vegas Line,* (December 17, 2010). This attempt used common offensive and defensive statistics and a novel "computed strength" feature with a Gaussian Naïve Bayes model to predict game outcomes. While our goal is somewhat different—to predict the winner in a matchup between any two NFL teams—this paper and supplemental research provided motivation for our overall approach. We attempted to create the most accurate model possible for predicting game winners with the data we selected. We ultimately employed a Decision Tree Classifier to successfully pick the game winner over 66.6% of the time.

As data science applications evolve, so too does their usage in gaining competitive advantage. Professional sports organizations, though no stranger to the implementation of analytics in their approach to winning, always stand to benefit from more advanced methodology. While our capstone does not exactly employ an approach that would immediately benefit a team, it taught us the foundations of what would be needed to advance the literature in the field and pursue a career in sports analytics. The goal of our project was to build a data product to accurately predict the results of NFL games. We were taught that the implementation of the data science pipeline comprised the foundational approach to solving our problem.

Hypothesis

Using data from NFL games between 2002 - 2018, we attempted to determine which features and attributes related to NFL player data best predicted a winning outcome. Using win percentage of each team as our baseline, we predict that the inclusion of features selected through careful research and analysis will lead to an increase in accuracy. Initially, 22 features were selected based on offensive and defensive game statistics we believed were most likely to be predictive of game outcomes and two novel features that were engineered were added in - “power ranking” and “win percentage.” Our initial set consisted of: 1) Game location; 2) Point differential; 3) Team passing yards; 4) Team rushing yards; 5) Team turnovers taken; 6) Team turnovers given; 7) Turnover differential; 8) Team passing yards against; 9) Team rushing yards against; 10) Team win percentage; 11) Points for rank; 12) Points against rank; 13) Passing offense rank; 14) Rushing offense rank; 15) Point differential rank; 16) Turnovers taken rank; 17) Turnovers given rank; 18) Turnover differential rank; 19) Passing defense rank; 20) Rushing defense rank; 21) Win percentage rank; and 22) Power ranking. The final set drops the game location, turnovers taken/given, and all ranks barring the power ranking (explanation in subsequent section of the paper).

Data Ingestion and Storage

With the research and hypothesis under the proverbial belt, the next step meant going out and getting as much data as possible, and storing it in a way that would be lightweight and accessible to all. After multiple failed attempts to pull at the team level from both ESPN and PFR websites, some googling led us to a script at https://github.com/zackthoutt/nfl-player-stats/blob/master/scrape-nfl-stats.py that scraped NFL player game log data from https://www.profootballreferene.com, the process could continue to the storage aspect. The script pulled 1,062,249 instances of game logs played by 25,995 players since 1950. The data was uploaded to a PostgreSQL database hosted on an Amazon RDS instance, allowing everyone to work without having to store locally. Though it sounds simple, getting the data into the database was a very cumbersome process. The project group was comprised of strong working professionals with backgrounds in analytics and project management, though there was a dearth of experience in data upload and database management. It wasn't known what worked best and why, but after reviewing documentation and discussing with instructors PostgreSQL received the nod of approval. The next step was to figure out how to interact with PostgreSQL from Jupyter, and while it was easily understood how to connect to the database, uploading one file was a head scratcher. Each JSON file pertained to one player and contained each game log as a separate dictionary in one big list. After multiple failed attempts over the course of a few days, the lightbulb came on during one working session when we found an example script that someone used with MySQL. After a bit of modification, less than an hour later the full script to upload each and every game log from the locally stored JSON files to the database was ready to go.

Unfortunately, what AWS does not make very clear is if you go over the free tier limit, they will charge you and they will not tell you that they are going to charge you until that first billing statement arrives. It can only be assumed that somewhere, deep in the fine print, this is explicitly stated. We made sure Bezos wouldn't get the last laugh by getting it expensed to the company where one of our group members works, allowing us to continue working instead of using our time to seek out alternative forms of lightweight storage. With the data stored and accessible to all, it was time to tackle what had been explained to us as "80% of the work" and that might've been understating it.

Data Wrangling

Once we were able to get our data ingested into postgres our next step was to wrangle the data. The original format of our data was as follows:

We scraped player logs from the pro football reference website. The player logs consisted of individual player's game statistics for every game a specific player recorded a statistic in throughout the player's career. If a player played in 100 games in their career, we have 100 game logs for that player.

Each player log came in json format. Below is an example of one game log found in a player log file. The below game log is the first game Drew Brees ever recorded a statistic in:

{"player_id": 2415, "year": "2001", "game_id": "200111040sdg", "date": "2001-11-04", "game_number": "8", "age": "22.293", "team": "SDG", "game_location": "H", "opponent": "KAN", "game_won": false, "player_team_score": "20", "opponent_score": "25", "passing_attempts": 15, "passing_completions": 27, "passing_yards": 221, "passing_rating": 94.8, "passing_touchdowns": 1, "passing_interceptions": 0, "passing_sacks": 2, "passing_sacks_yards_lost": 12, "rushing_attempts": 2, "rushing_yards": 18, "rushing_touchdowns": 0, "receiving_targets": 0, "receiving_receptions": 0, "receiving_yards": 0, "receiving_touchdowns": 0, "kick_return_attempts": 0, "kick_return_yards": 0, "kick_return_touchdowns": 0, "punt_return_attempts": 0, "punt_return_yards": 0, "punt_return_touchdowns": 0, "defense_sacks": 0, "defense_tackles": 0, "defense_tackle_assists": 0, "defense_interceptions": 0, "defense_interception_yards": 0, "defense_interception_touchdowns": 0, "defense_safeties": 0, "point_after_attemps": 0, "point_after_makes": 0, "field_goal_attempts": 0, "field_goal_makes": 0, "punting_attempts": 0, "punting_yards": 0, "punting_blocked": 0}

In PostgreSQL, the first table we created was the original player logs table consisting of all the player log files uploaded from a local machine. For our modeling purposes, we focused on year 2002 forward. The reasoning is largely due to the widely accepted notion that the league early in the 2000s started to shift more towards a “passing” league, so prior data might be misleading. Our first step was to sum the individual player game logs into individual game stats for an entire team. We were able to do this by leveraging the game_id key and the game_location key. All players who recorded a statistic in a specific game for either team have the same game_id. The game_location allowed us to split the home team players and the away team players and sum the key statistics by game_id with a WHERE clause specifying home or away producing each team’s total game stats for a specific game_id. With this in mind the tables, home_game_stats and away_game_stats, were created.

The next step was creating some additional team stats that were anticipated to be important features for the model. The home_game_stats and away_game_stats tables contain the game stats for a specific team in a specific game, however, defensive yardage did not come with the original data scrape. With the idea in mind of having the defensive statistics for a team, a novel way to create the variables was sought out. Particularly, the preferred variables were:

- Opponent_passing_yards
- Opponent_rushing_yards
- Opponent_turnovers_taken
- Opponent_turnovers_given

To do this we created two additional tables, home_games and away_games. For each we did an INNER JOIN on the other table. For example, to get the Home team’s opponent_passing_yards we joined the Away team’s passing_yards statistic as opponent_passing_yards and vice versa.

At this point we have 5 tables, player_logs, home_game_stats, away_game_stats, home_games, and away_games. The home_games and away_games table contain the data needed to move forward, however, the data was required to be in one table, so we created a all_games table that was the away_games table appended to the home_games table. The all_games table contains all games statistics for the home team and the away team from 2002 through 2018 seasons.. We will use this table to generate the team's average statistics over the individual seasons.

For the next several steps in our data wrangling we moved from PostgreSQL to Jupyter. We utilized the pyscopg2, sqlalchemy, and pandas libraries to connect to our PostgreSQL database and turn the all_games table into a pandas data frame. Initially we decided to wrangle the data for one season first and once we determined the steps for one season we would apply a loop to cycle through the steps for each year in our final data set.

The first thing done in Jupyter was the creation of a dictionary that contained the team name as the key and a data frame of only that team's game stats for the specific season. The end result here was a dictionary with 32 data frames. Now that there were individual data frames for each team, we needed to add a week_count column and a win_percent column to each of the team data frames for the different teams. The week count column was critical in order to determine averages. The win_percent column was an additional statistic / potential feature we wanted to incorporate in our model. Functions were created that accepted a data frame as a parameter and then looped through the dictionary and generated these two columns for each of the data frames for the different teams.

Our next step was to convert the following columns from strings to numeric values:

- Game_location
- Outcome
- Game_date

Once we had all columns as numeric values, we could begin converting our game stats into averages. Our approach for the averages was to begin with the 5th game of each team's season and modify that row in the team's data frame to be the average of the first four games of the season. For the next game we took the average of the first five games of the season and converted those averages to be the team's statistics for the 6th game of the season. We continued this process through the rest of the team's games for that season. The intent here is when a game is looked at as an instance for our model the features will be the averages of one team through that point in the season compared to the averages of another through that point in the season. We did not include the first four games as instances in our final data frame. The creation of this function wound up being the most difficult task with regards to wranging, however, with the help of Adam and multiple different stack overflow examples we were able to push through each wall that popped up in front of us and achieve the result we were looking for. Seek and you shall find.

After finalizing steps to obtain a data frame for a single season, a loop was needed to apply the function for each of the seasons. This also posed a challenge for us. On our first run we were able to get through seasons 2002 – 2010 but hit a roadblock in season 2011. Within our win_percent function, we created a list of week numbers by calling for the unique values from the week_number column. For a reason still not quite understood, a period was being included with the value in the list as well as a NaN value for some teams. Again, we scoured the internet for a solution and were able to come across a post that included 6 lines of code that resolved our issue. The data frame containing averages for each team and subsequent game had been created for each season.

Our final steps in wrangling took us back to PostgreSQL where we used psycopg2 to convert our final data frame, final_wable_before_join, to a table in PostgreSQL. In PostgreSQL we used the SQL rank function to assign each team a rank between one and thirty-two for each statistical category for each week number of each season. Each week's ranks were based on the averages of the previous games played for that season. We then took the average rank for each team across all statistical categories and made a rank of averages to create our novel feature "power_ranking." In addition to the power ranking we also used SQL to create a "point_differential" and "turnover_differential" statistic. This provided the complete set of statistical categories that we wanted to take into the exploratory data analysis phase. Our potential features consisted of 22 statistical categories for each team however we needed each team that was involved in a game stats together in one row to complete our instance. To do this we performed steps similar to the beginning of our wrangling. We split the final_table_before_join table into separate home and away games tables and then joined the tables together by game_id creating final_table_joined. We transferred our data back to a pandas data frame to begin Exploratory Analysis.

Exploratory Data Analysis

We successfully uploaded our data frame into a separate Jupyter Notebook for exploratory data analysis and feature engineering. After running general descriptive statistics on our original grouping of 22 features we visualized the data with histograms and the RadViz visualizer. The results of this analysis with the histograms we ran appear to confirm our assumptions about the significance of several of our key features. For example, the distributions for team passing and rushing yards, appeared to have a close to normal and un-skewed distribution. Unfortunately, the RadVis visualization was less useful with respect to which values were predictive of win outcome.

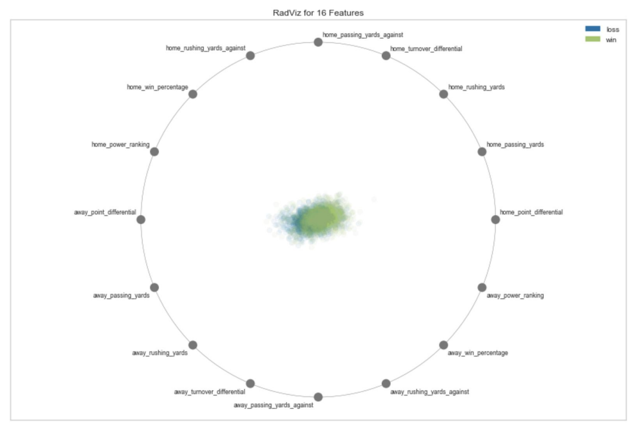


Fig 1.1: RadViz visualization from yellowbrick

We then assessed each of the 22 features using a feature importance analysis and a Rank2D visualization in order to determine the collinearity between features in order to determine which were most predictive of winning outcomes. Based on this initial analysis, we engineered 2 new features: point differential and turnover differential. This resulted in 8 core features for each team: (1) Rushing Yards For; (2) Rushing Yards Against; (3) Passing Yards For; (4) Passing Yards Against; (5) Point Differential; (6) Turnover Differential; (7) Power Ranking; (8) Winning Percentage. Further feature importance analysis revealed that point differential, the power ranking and win percentage were critical variables.

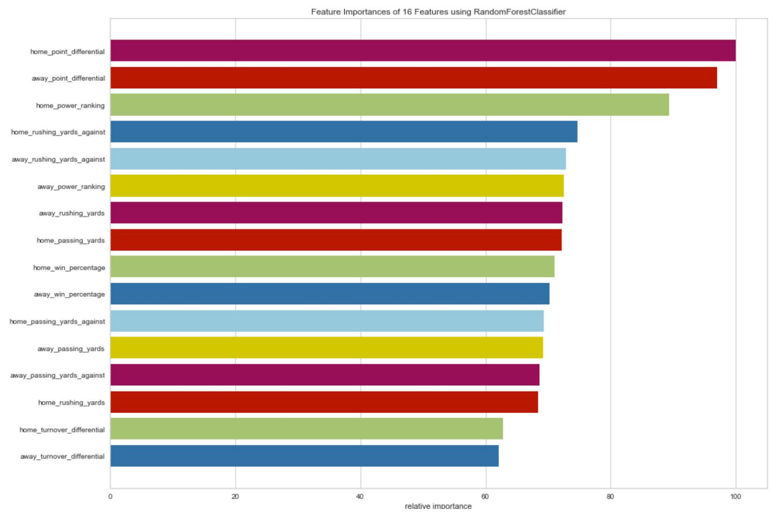


Fig 1.2: Feature Importance visualization from yellowbrick ranking newly selected features

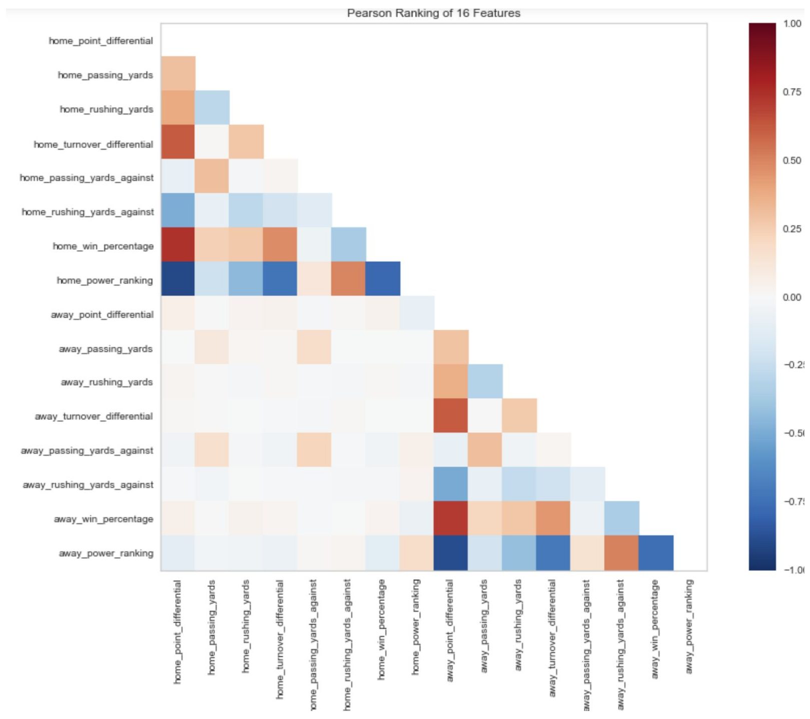


Fig 1.3: Rank2D visualization from yellowbrick assessing collinearity. Expected collinearity among win percentage and average point differential per game.

Our exploratory phase led us to the creation of some and the selection of our final features. With this in hand, and time running down, the modeling framework had already been drawn up to expedite the process.

<u>Modeling</u>

The paper by Jim Warner initially employed a Gaussian approach with success, so naturally we started with that one plus the decision tree which we had learned about in class. Further research led us to try a wide array of different classification models including: Gaussian, Decision Tree, K-Nearest Neighbors, Random Forest, Extra Trees, Gradient Boosting, Voting, Bagging, AdaBoost, and Stacking. Initial results at the baseline never got us past a combined f1 score of 0.585 (Gaussian).

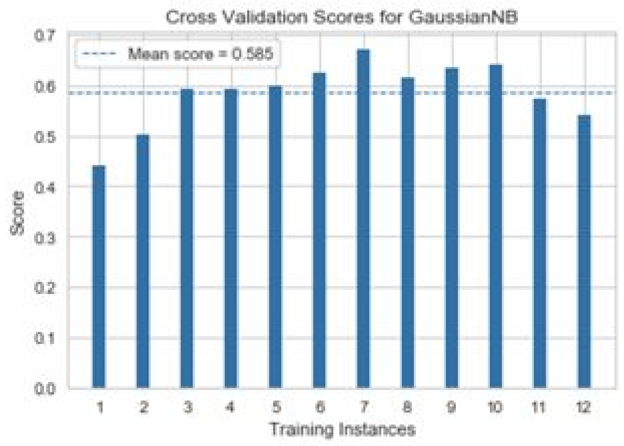


Fig 1.4: Cross Validation visualization from yellowbrick looking at the baseline score for the Gaussian model. Initial features used were the win percentages for each team.

Initial observations showed that while win precision was typically around 64%, the loss prediction tended to be imprecise for all approaches, though the Gaussian approach did have almost 59% loss precision at one point. This is likely due to a class imbalance, since on average the home team wins more often than the away team.

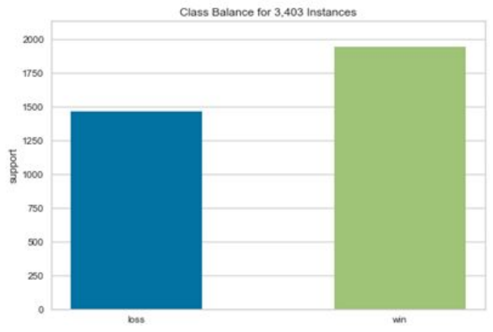
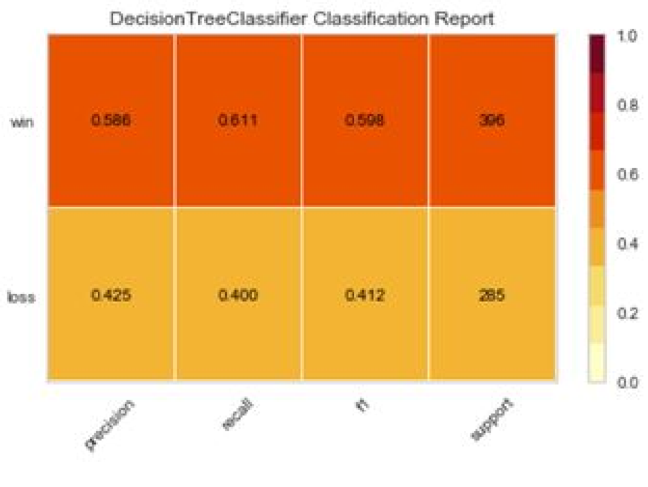


Fig 1.6: Class Balance visualization from yellowbrick showing more wins than losses in the dataset

Due to the relatively poor performance, the K-Nearest Neighbors and Extra Tree models were dropped. With the new features, the 8 remaining models were tested and the top 4 to move on to hyperparameter tuning / testing on 2018 data based on cross validation and further analysis were the Decision Tree, Gaussian, Bagging Classifier (did initial hyperparameter tuning to find best classifier) and Stacking Classifier. Since the Gaussian and Bagging Classifiers are special cases, hyperparameter tuning was conducted on the Random Forest and Gradient Boosting to see if the scores could become comparable to the other high performers. The Decision Tree classifier saw remarkable improvement from its baseline jumping +0.114 from 0.525 to 0.639.



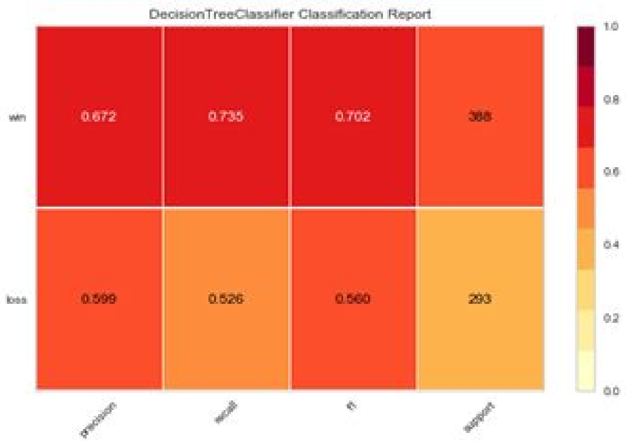


Fig 1.5: Classification Report visualization from yellowbrick looking at the baseline score for the Decision Tree (left) and comparing to the scores after including new features and hyperparameter tuning (right).

Although the Gaussian Naive Bayes approach does not require any hyperparameter tuning, its scores after adding in the new features also jumped more

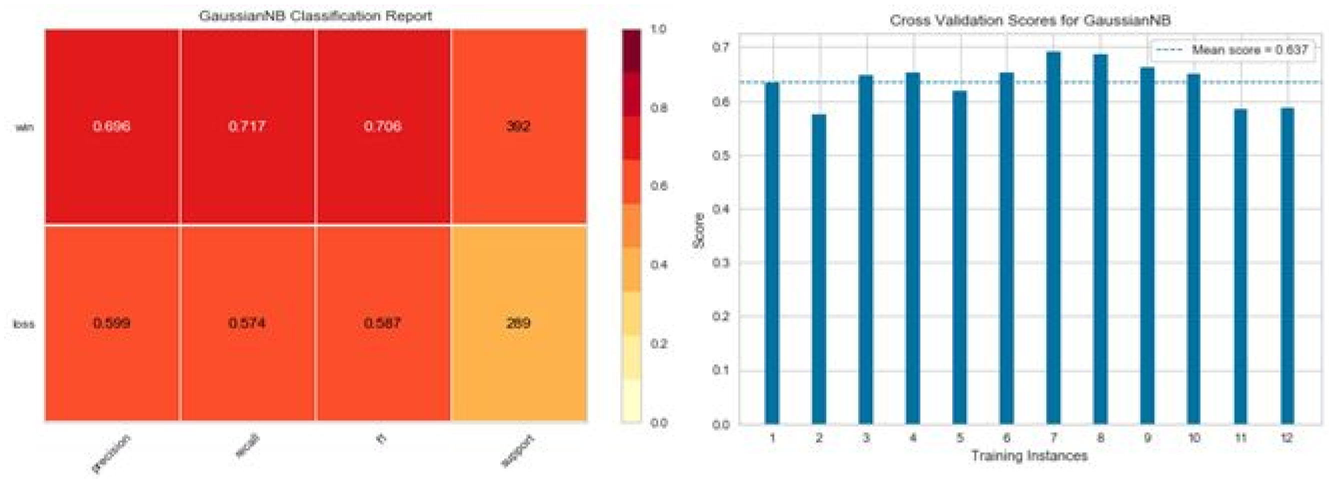
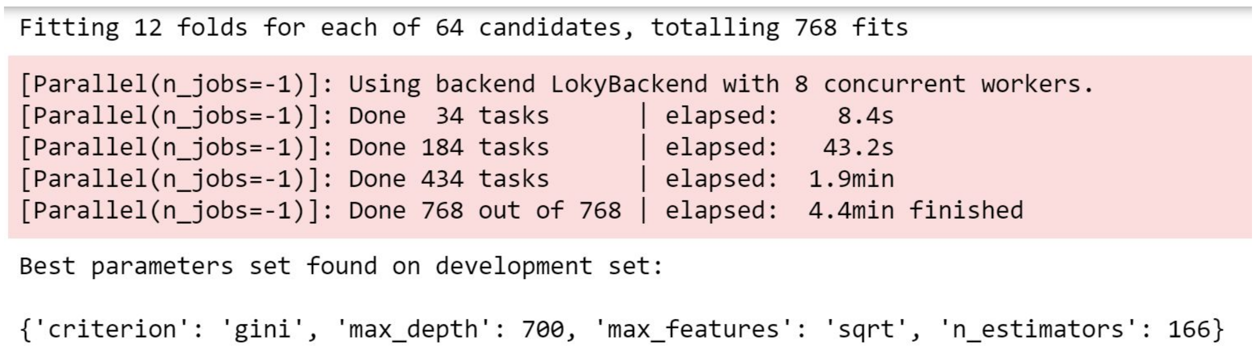


Fig 1.6: Classification Report and Cross Validation visualizations for Gaussian approach after adding in new feature



```
Fitting 12 folds for each of 64 candidates, totalling 768 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    8.4s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   43.2s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done 768 out of 768 | elapsed:  4.4min finished

Best parameters set found on development set:

{'criterion': 'gini', 'max_depth': 700, 'max_features': 'sqrt', 'n_estimators': 166}
```

Fig 1.7: Hyperparameter tuning script output for random forest classifier.

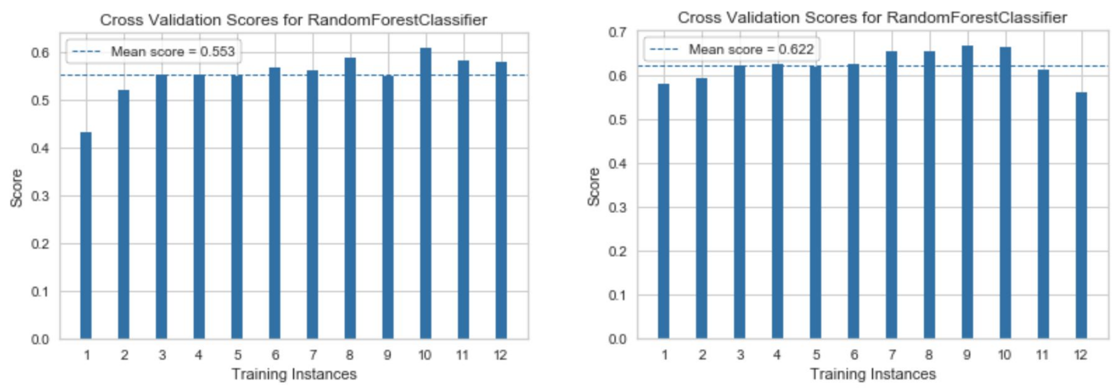


Fig 1.8: Cross Validation scores for baseline (left) vs, Random Forest with new features and hyperparameter tuning.

After hyperparameter tuning, we decided to test our data on 2018 NFL games. The results were as follows:

- Decision Tree = 66.6% total precision
- Gaussian = 63.2% total precision
- Bagging Classifier = 63.7%
- Stacking Classifier = 63.7%

All of the models performed well, however, the Random Forest and Gradient Boosting (not shown) had roughly 60% total precision each. The loss precision saw the greatest increases in each of the models, though win precision for most still made slight gains.

<u>**Future Direction and Limitations**</u>

The conclusion of our modeling led us to the selection of the Gaussian approach due to its consistently high performance. Although the Decision Tree led us to the best scores, recent runs have left us perplexed as to why the consistency has dropped. Our next step would have been to create a flask application that would allow a user to select a matchup between two teams and predict the result of the game. The hope was to move towards using regression modeling to predict scorelines, however, due to time and resource constraints this was unable to be done, though it will be the plan moving forward.

Our larger conclusion was that we needed more data and of a higher quality / granularity. The desired data is the distinction between what kind of stadium a particular team plays in (dome vs.

outdoors), the weather on the day of the outdoor games, play-by-play data to calculate the all-telling DVOA (Defense-adjusted Value over Average; see https://www.footballoutsiders.com/info/methods) metric. One big limitation also came in the form of a lack of fumble data for turnovers. Our script was having trouble scraping it so we took it out altogether in order to expedite the process. This leads to misrepresentation of teams that thrive throughout the year on defense, such as the 2016 Super Bowl champion Denver Broncos. The model performances would also likely improve factoring in player attributes such as the average age of the team, the amount of previous pro-bowlers and all-pros a team has, and a ranking among current players at each position.

All-in-all, we learned a great deal about data science and its applications in the real world. The goal is to find a career in sports analytics as a data scientist, and this was a monumental and invaluable first step forward.

References

Jim Warner, *Predicting Margin of Victory in NFL Games: Machine Learning vs. the Vegas Line,* (December 17, 2010).

Footballoutsiders.com, *Methods to our Madness*

https://www.footballoutsiders.com/info/methods

Paul Joos, *NFL Predictions Using Machine Learning,* (November 11, 2015)

http://pjoos.github.io/2015/11/11/nfl_ml.html

Predicting the Winner of NFL Games – An Attempt to Reach 80% Accuracy,

https://samstatsheets.com/predicting-the-winner-of-nfl-games-an-attempt-to-reach-80-accuracy/