# Machine Learning Engineer Nanodegree

## Capstone Project – Fantasy Football Player Predictions

Mike Dettmar
November 12th, 2019

# I. Definition

### Project Overview and Problem Statement

Fantasy Sports is a nearly 14 billion-dollar industry indulged in by 21% of the American populace. Whether for money or bragging rights, all of those 60 million players are interested in finding an edge in fantasy.

In daily and season-long formats, participants draft combinations of players with the hope that their team will perform better than others'. The core task, then, is predicting the performance of individual players. The widespread availability of sports data online enables us to apply machine learning to this problem. Indeed, many have built models attempting to beat traditional predictions, which are usually based on expert opinion.

Among the most interesting modeling approaches to fantasy performance is outlined in a 2018 article by Christopher Zita on Medium.[1] He applied different types of neural networks to the task of predicting the fantasy performance of two very different players – consistent quarterback Tom Brady and inconsistent running back Todd Gurley – during the 2018 season. Using multivariable RNNs customized to each player, Zita was able to predict Brady and Gurley's performance with mean absolute errors of 5.3 and 5.68, respectively – better than other models built in studies we reviewed and very close to or better than the best expert projections. For context, a naïve prediction using a three-day moving average of the players' fantasy scores produced considerably worse results (6.74 for Brady and 9.96 for Gurley!).[2]

Clearly it is possible to produce predictions that beat a naïve prediction. But can we easily recreate this methodology, and can we improve upon it? Zita's best predictions come from an RNN, which makes sense – the ability of RNNs to understand sequences should allow them to recognize patterns in player performance that depend on time. But Zita is does not mention much detail about his network architecture or features.

Common feature classes used in fantasy prediction concern past player performance, defensive quality, location, betting spreads, and weather (Zita directly references the first three in his write-up). We can use features derived from these classes to construct a basic RNN (presumably) similar to Zita's. A less explored but very promising feature class for this problem comes from social media sentiment. The idea is based on the 'wisdom of crowds' theory, which suggests that independent, nonexpert predictions of various phenomena tend to estimate true values with a high degree of accuracy when aggregated.[3] Furthermore, a number of studies have convincingly shown the value of Twitter volume and sentiment in predicting the

---

[1] Zita, Christopher. "Predicting Player Performance, Using Neural Networks." *Medium*, Medium, 29 July 2019, medium.com/@christopherzita/predicting-player-performance-using-neural-networks-f6142784b681.

[2] Notebook: modeling_notebooks/appendix_naive_prediction_2018_season

[3] Surowiecki, James. 2005. *The Wisdom of Crowds*. New York: Anchor Books.

outcome of NFL games.[4] Predictions for individual players have not been rigorously studied, presenting us with an opportunity

Here, then, we will attempt two things: first, to construct a baseline RNN model that can beat a three-day moving average approach with statistical significance, and second, to significantly improve upon the baseline by incorporating Twitter features. To accomplish the former, we will follow Zita's example and build an RNN for two players – Tom Brady and LeSean McCoy – using the 2010 and 2011 seasons as training data and the 2012 season for testing. We'll construct features around past player performance, betting lines, location, weather, and defensive quality. Then, to address the latter, we will add features around Twitter volume and sentiment for the player's team and opposition. We chose the 2010 - 2012 seasons because Sinha, et. al.[5] were kind enough to make their dataset of tweets publicly available. It follows that we replace Todd Gurley with LeSean McCoy, a similar inconsistent running back who was in his fourth season in 2012.

Our primary success metric will be mean absolute error (MAE, defined as the average absolute difference between predictions and actuals in the test set), mostly because this is the same metric used by Zita and will thus allow for an easy comparison. We'll secondarily track root mean squared error (RMSE) – the average difference between the squared predictions and actuals – as this is a commonly reported metric for regression problems that more heavily penalizes large errors than MAE.

# II. Analysis

### Data

To model fantasy performance in 2012 based on the preceding two seasons for Brady and McCoy, we need player fantasy points, game betting lines, defensive stats, NFL-related tweets, game locations, and game weather for the 2010 – 2012 seasons. We found all of this information by combining data from four sources:[6]

1. Fantasy stats for Tom Brady and LeSean McCoy for the 2010 – 2012 seasons. These take the form of two .csv files[7] downloaded by hand from Pro-Football-Reference.com.[8] Each .csv contains 48 games per player – 16 games per season, or 32 training games and 16 test games. Brady played in every game, but McCoy unfortunately was injured for one game in each of the 2010 and 2011 seasons, and four games in the 2012 season. We include the games where he did not play in the file, assigning a fantasy score of zero. One of the columns in the .csvs is a flag for whether or not the player played in a

[4] Yanchen Hong and Steven Skiena's 2010 study showed that the ratio of positive to negative sentiment associated with a given NFL team as expressed in the news in the week before a game correlated with game outcomes, and that a betting model based on a simple sentiment score and home field advantage could achieve a win rate of 60% between 2006 and 2008 – well above the 53% needed to profit (note that a 60% win rate does not mean 60% accuracy, because the authors were examining the point spread) (Hong, Yancheng & Skiena, Steven. 2010. "The Wisdom of Bookies? Sentiment Analysis Versus. the NFL Point Spread." ICWSM 2010 - Proceedings of the 4th International AAAI Conference on Weblogs and Social Media). Sinha, et. al. built upon this study in 2013, finding that Twitter rate (volume) features could greatly improve the performance of a model trained on traditional statistical features (though they found fairly unsophisticated Twitter unigram features, focused on the content of Tweets, less useful). Their best model, trained on the 2010 and 2011 seasons and incorporating both Twitter rate and classic statistical features, was able to choose the winner under two different betting schemes over 58% of the time in the 2012 season (Sinha, et al. "Predicting the NFL Using Twitter." *ArXiv.org*, 25 Oct. 2013, arxiv.org/abs/1310.6998). Finally, Schumaker, et. al's 2017 paper introduced a metric based on the change in the ratio of positive to negative sentiment Tweets associated with a team in in the three days before kickoff that was more strongly correlated with actual game outcomes than raw sentiment features alone (Schumaker, R.P., Brown, L.L., Labedz, C.S., & Jarmoszko, A.T. 2017. Using Financial Analysis Techniques on Twitter Sentiment to Improve NFL Predictions). This 'swing' metric, as they call it, was able to correctly call the same number of games in the 2015 season as raw odds alone, but was much more profitable because it was more successful on underdogs than the odds-based approach (incidentally, this also suggests some orthogonality in the two feature classes). Following in the authors' footsteps, we will construct raw and swing features based on tweet volume and sentiment for this study.

[5] See footnote 4.

[6] All raw data found in the data/data_raw folder.

[7] data/data_raw/players/brady.csv and mccoy.csv

[8] "Tom Brady NFL Fantasy Football Statistics". Pro-Football-Reference.com. Retrieved October 19, 2019; "LeSean McCoy NFL Fantasy Football Statistics". Pro-Football-Reference.com. Retrieved October 19, 2019.

given game. It is not necessary to predict these games, as teams typically report whether or not a player will play before the beginning of a game. Theoretically, a model should be able to learn that not playing directly predicts a fantasy score of zero; however, given the small size of our dataset, we worried that the model will not pick up on this connection. Nonetheless, we wanted to leave these games in the dataset because it could be important for a model that understands time, such as an RNN, to see that a given game was preceded by games in which the player did not play. Our solution was to keep the games where McCoy did not play to construct our input data for the RNN, but to remove any datapoint where the actual outcome is zero from the final test set.

The dataset contains the following notable columns:

a. **G# (int)**: Week a game was played in a given season. An integer from 1 to 17. Note that one number will be skipped per season because of the bye week.
b. **Date (datetime)**: The date on which the game was played. Will be used to split training/test sets.
c. **played (Boolean)**: Whether or not the player played in a given game.[9]
d. **home/away (str)**: Whether or not the game was played at home or on the road. '@' if the game was away, blank otherwise. We will have to convert this field to Boolean.
e. **Opp (str)**: The abbreviated name of the opposing team.
f. **FanPt (float)**: The number of fantasy points scored by the player in the game according to the standard fantasy football scoring scheme. This will be the target in modeling.

The data is already in the time series format that we need for our RNN and will therefore be the base dataset to which we will join all others. Notably, we dropped the players' actual football statistics for this project, because fantasy score is directly determined by those statistics. We can therefore use it as a stand-in.

2. Data on the location, weather conditions, betting lines, and results of all NFL games played since 1966. This is a single .csv[10] of 12667 records downloaded from Kaggle user spreadspoke.[11] The .csv has the following notable columns:
    a. **schedule_date (datetime)**: The date on which the game was played. Will be used for joins to the players data.
    b. **team_home (str)**: The home team's name.
    c. **team_away (str)**: The away team's name.
    d. **spread_favorite (float)**: The point spread for the team favored to win.
    e. **over_under_line (int)**: The over/under line for number of points scored.
    f. **weather_temperature (int)**: The temperature in Fahrenheit.
    g. **weather_wind_mph (int)**: Windspeeds in miles per hour.
    h. **weather_humidity (int)**: The humidity on a scale of 1 to 100.
    i. **weather_detail (str)**: A freeform field to add additional weather. Has information on precipitation and other unusual weather events; blank if no outstanding weather information exists.

    We will use the betting and weather features in our model, as Schumaker, et. al. suggest the former is orthogonal to sentiment features while the latter likely influences player performance. Weather data is missing ~25% of the time.

[9] Highlighted fields will directly be used as features/as inputs for feature engineering.

[10] /data/data_raw/games/games_metadata.csv

[11] spreadspoke (2019, August). NFL Scores and Betting Data. Retrieved October 19, 2019.

3. Defensive statistics for each team during each game of the 2010 - 2012 NFL seasons. We also downloaded this data from Pro-Football-Reference.com, this time using the Python sportsreference package.[12] We saved the data as a single .csv file[13] which has the following notable columns:
    a. **team (str)**. The team whose defensive stats are listed.
    b. **date (datetime)**. The date the game was played.
    c. **pass_yards_allowed (int)**. The number of passing yards allowed by the defense.
    d. **interceptions (int)**. The number of interceptions made.
    e. **fumbles_forced (int)**. The number of fumbles forced.
    f. **rush_yards_allowed (int)**. The number of rushing yards allowed.
    g. **sacks (int)**. The number of sacks made.
    h. **points_allowed (int)**. The number of points allowed.

    Because both of the players we are modeling are offensive players, the quality of the defense they are up against in each game will be important. We will make features based on three day moving averages of each statistic in the dataset.

4. Tweets associated with each NFL team during the 2010 - 2012 NFL seasons. This was the same dataset used by Sinha, et. al. in their 2013 study using Twitter features to predict NFL game outcomes.[14] To briefly recap, they used Twitter's streaming API, which randomly samples ~10% of tweets per day that match given criteria, to gather tweets throughout the course of the 2010 - 2012 NFL seasons. The criteria they provided were hashtags corresponding to NFL teams. They then used these hashtags along with timestamps to associate each tweet with an NFL game and team.

    There are a few nuances to this dataset:

    a. First, the hashtag associations are not perfect – for example, some of the tweets associated with the Arizona Cardinals, upon inspection, are actually about the MLB's St. Louis Cardinals. Sinha, et. al. explicitly mention that they tried to filter out some of these edge cases, but clearly it was not possible to obtain a perfect dataset.
    b. Secondly, they divided the tweets into three categories: pregame, which were sent between 24 hours and 1 hour of a game, postgame, which occurred between 4 hours and 28 hours after the start of the previous game, and weekly, which occurred between 12 hours of the end of the previous game and 1 hour before the start of the next game. For simplicity's sake, we used only the weekly tweets for this analysis, leaving 1,708,275 tweets in the dataset.
    c. Third, the last tweets in the dataset were sent out on December 23rd, 2012. The 2012 NFL season ended on December 30th, 2012. We will drop this last point from our dataset as we will not be able to accurately build Twitter features for it.
    d. Fourth, the volume of tweets grew substantially from season to season (see figure 1). We hypothesize this is because of increasing use of Twitter as a platform in this time period (Twitter only launched in 2008). This necessitated normalizing any features based on tweet volume.
    e. Finally, and most importantly, not all tweets present in the dataset are still available (figure 1). A 404 error is thrown when attempting to retrieve the text of ~30% of the tweets, indicating the user has deleted them. This could be problematic, as we plan on using sentiment features for the players' teams and opposition as inputs to the model. If we assume tweets were deleted randomly, then we should not have to worry, but if one type of tweet is systematically deleted more often than others, then we may have a leakage problem on our hands. We suspect this

[12] The script for downloading is located at data_compilation/get_defensive_stats.py

[13] data/data_raw/teams/defensive_stats.csv

[14] Predicting the NFL using Twitter. Shiladitya Sinha, Chris Dyer, Kevin Gimpel, and Noah A. Smith. In *Proceedings of the ECML/PKDD 2013 Workshop on Machine Learning and Data Mining for Sports Analytics*, Prague, Czech Republic, September 2013.

may be the case – it seems likely that users would delete negative tweets at higher rates than positive ones.

Addressing this problem is out of scope of this project, but should be noted. To mitigate it, we will assume any deleted tweet was of neutral sentiment.

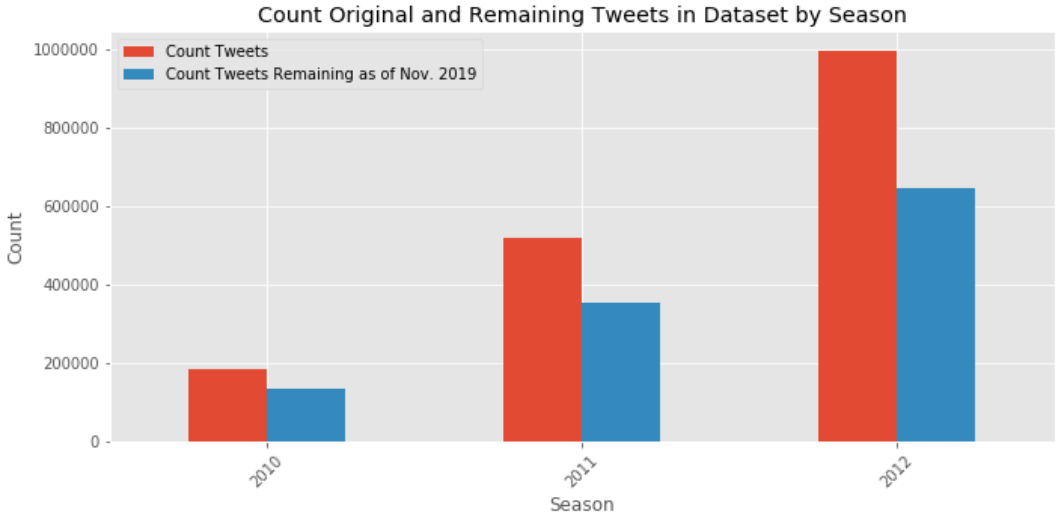



*Figure 1: count of original and remaining tweets in the dataset by season. The volume of tweets roughly doubles every season, but a greater percentage is deleted, too. For the 2012 season, roughly one third of tweets originally found in the dataset have been deleted. Chart produced in modeling_notebooks/notebook 06_visualize_twitter_data.*

The tweet data present in the repository is composed of four .csvs: a .csv containing the tweets from each season, along with one that contains column mappings.[15] Before building features with this data, we need to pull the text for all tweets (if available) and conduct sentiment analysis. Each .csv contains the following notable columns:

a. **tweet_id (int).** the ID of the tweet. Used to retrieve the text.
b. **tweet_UTCtime (datetime)**. The time the tweet was sent out, in UTC.
c. **team (str)**. The team associated with the tweet.

## Algorithms and Techniques

The primary goal of this project is to build two player-level fantasy football prediction models that can beat a baseline moving average model; a secondary goal, in service of the first, is to prove out the value of Twitter data in making these predictions.

For the second goal, we will begin by building an XGBoost model using all of our features. The reason for this is twofold:

1. As a tree-based model, XGBoost makes it easy to understand feature importance. We are particularly enamored of Shapley values, which, at a high level, iteratively remove individual features from the model in a Monte-Carlo-like process to see how each feature affects the output score. By building an XGBoost model for Brady and McCoy, we will be able to use Shapley values to understand how influential Twitter features are compared to traditional features.

[15] All present in the ~/data/data_raw/tweets_meta folder

2. XGBoost is an algorithm known for its 'unreasonable effectiveness' and ease of implementation. We therefore expect it to perform considerably better than our baseline moving average prediction. If this is the case, the XGBoost model can serve as a second benchmark model against which to compare our more complicated RNN.

The additional preprocessing required to stand up and XGBoost will be minimal because all of our features are numerical or Boolean, and XGBoost can handle missing values by default.

Because our dataset is very limited (16 games per season means we have only 32 training samples and 16 test samples; less for McCoy, who sat out some games because of injuries), we are wary of overfitting. We will therefore begin with very conservative hyperparamters for XGBoost.

From there, we'll spend a bit of time tweaking the parameters to see if we can achieve better performance on the test set, noting that this may be somewhat dangerous without a validation set. On the best model, we will examine the Shapley values, hoping that a few of the Twitter features will be among the top performers.

Finally, to mimic a real-world fantasy season, we will build a series of models through online training. In other words, we'll train a model on the 2010 and 2011 seasons, predict the first game of the 2012 season, add that game to the training set, retrain, predict the second game of the 2012 season, and so on until we have predicted every game in the 2012 season. This should improve performance of the model by inflating the size of the training set over time.

For our final model, we will build a Recurrent Neural Network, or more specifically, a Long-Short Term Memory Recurrent Neural Network. We believe an LSTM-RNN to be the best choice for this problem for a couple of reasons:

1. As neural network variants, LSTMs have the ability to learn very complex, nonlinear relationships that might exist in a dataset. We can imagine these existing in fantasy sports, where player performance is high-variance and there is no guarantee that, say, a better opposing defense will lead directly to a worse performance for our quarterback.
2. LSTMs, like all RNNs, are designed for time series because they take previous outputs as inputs. LSTMs in particular have memory cells that allow them to retain information from long ago in the series, thus letting them discover long-term linkages. It seems clear that a player's performance in fantasy sports will have some sort of relationship to their past performance. Moreover, LSTMs solve the 'vanishing gradient' problem common to RNNs.

We have a bit more data wrangling to do before implementing an RNN: we will need to scale our features to be between -1 and 1 to help the network converge, and we will also need to handle missing values. For the latter issue, we will take a Naïve Bayes (that is, the mean) estimation for every feature in the training set and impute that value as each missing value in the train and test sets.

We'll roughly follow the same workflow as we did with the XGBoost models to build the LSTM models: begin with conservative hyperparameters (specifically, a low number of epochs and neurons; overfitting will be a special concern here because our small dataset will be passed through an algorithm capable of finding very complex patterns, like noise), tune the hyperparameters, and finally build a model that iteratively trains on each new datapoint. We will do this twice: once to construct a 'baseline' LSTM model without any Twitter features (this will roughly mimic Zita's approach), and once for our final model with Twitter features.

Finally, we will need a model to perform sentiment analysis on our Twitter text so we can transform tweets into features. Because the performance of any NLP model can vary greatly depending on the similarity of the data it was trained on to a given dataset, we would ideally build our own sentiment analysis model for NFL tweets. However, this would be difficult and time consuming, especially because we do not have any sentiment labels for our dataset. A quick scan online did not yield any pre-built models specific to Tweets, so we decided to test two different prepackaged solutions for generalized sentiment analysis: the sentiment analyzer in the Python package TextBlob, and that in Stanford's Core NLP library.

We chose to test TextBlob's analyzer mostly because it is available via an easy-to-install Python package. The analyzer was built using a Naïve Bayes model on a movie reviews dataset. Stanford Core NLP was attractive because it is an industry-standard sentiment analyzer that is trained on a series of hand-labeled word combinations (albeit, these also come from a movie review dataset). The model is trained using a Recurrent Neural Tensor Network. A full overview of the methodology is out of this paper's scope, but suffice it to say that this is a robust model whose sentence-level training and predictions promise strong performance on short and oft-fragmented tweets.[16]

We first considered testing using the Twitter Sentiment140 dataset – a set of 1.6 million tweets labelled with sentiment based on the presence of emojis, available on Kaggle.[17] However, a cursory glance over the data showed the content to be quite different from our NFL tweet dataset. Instead, we hand-labelled 100 tweets from the 2010 season to test for accuracy.

TextBlob directly labels a full block of text passed to it as positive, negative, or neutral; Core NLP, on the other hand, labels each sentence in a passed text block on a scale of -2 to 2 (where -2 is strongly negative, 2 is strongly positive, and 0 is neutral). To score the sentiment of a full tweet, then, we simply took the sum of scores returned for each sentence in a tweet and classified each result as positive, negative, or neutral (for example, a tweet with one strongly positive and one neutral sentence in it would be classified as positive; 2 + 0 = 2, which, being greater than zero, is positive).

This methodology on the test set did not produce promising results. TextBlob was only able to attain 25% accuracy (versus 47% if predicting all positive), while Core NLP only had an accuracy of 5%![18] However, we did not consider this a strong knock against either model, because we were not highly confident in our own hand-labelling. Many of the tweets in the hand-labeled set were ambiguous, and it was difficult even for humans to discern their sentiment without context.

Despite its low accuracy on our 'test set,' we decided to move forward with Stanford Core NLP because of the robustness of its methodology and its speed. It took TextBlob about 2 seconds to analyze the sentiment of one tweet, which means it would have taken nearly 24 days to fully classify every tweet in our dataset. Stanford Core NLP was much faster, classifying the entire dataset in 18 hours.

---

[16] A full overview of the methodology behind Stanford Core NLP can be found at https://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf

[17] Go, A., Bhayani, R. and Huang, L., 2009. Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford, 1(2009), p.12.

[18] Notebooks can be found in sentiment_analysis_validation/notebooks

This is obviously the part of the project with the most opportunity for improvement. Spending more time either building a custom NLP model or improving the existing ones has potential to greatly improve the accuracy of our sentiment classifications, and therefore of our fantasy prediction models.

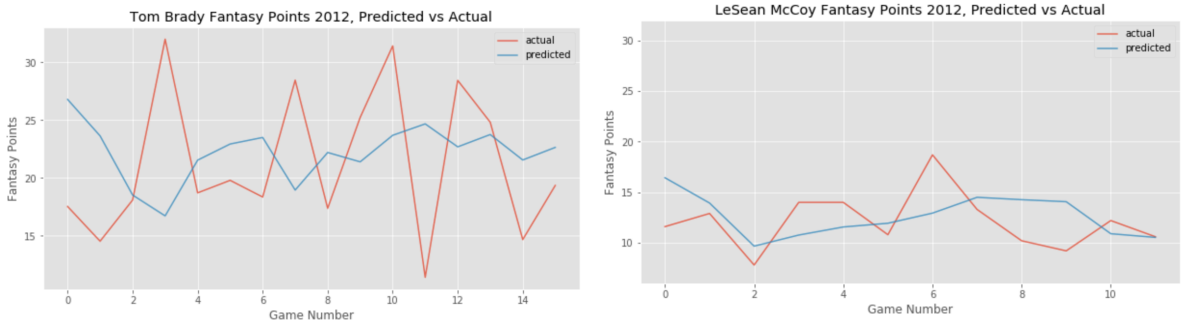
## Benchmark

We had two benchmarks to beat: a simple prediction based on a three-day moving average and an LSTM without Twitter features.

The baseline mean prediction yielded an MAE of 6.25 for Tom Brady and 2.88 for LeSean McCoy during the 2012 season (table 1).[19] The prediction for Brady is similar to what we saw in the 2018 season – unsurprising, as he is a consistent player. On the other hand, McCoy's baseline prediction is already extremely accurate, especially when compared to the Todd Gurley's baseline MAE of 9.96 in the 2018 season. Apparently, McCoy's 2012 season wasn't so inconsistent after all! It could be quite difficult to beat this performance, but we will try.

| Player | RMSE | MAE |
|---|---|---|
| **Brady** | 7.51 | 6.33 |
| **McCoy** | 3.19 | 2.65 |

*Table 1: RMSE and MAE from 3-day moving average prediction in 2012 season.*



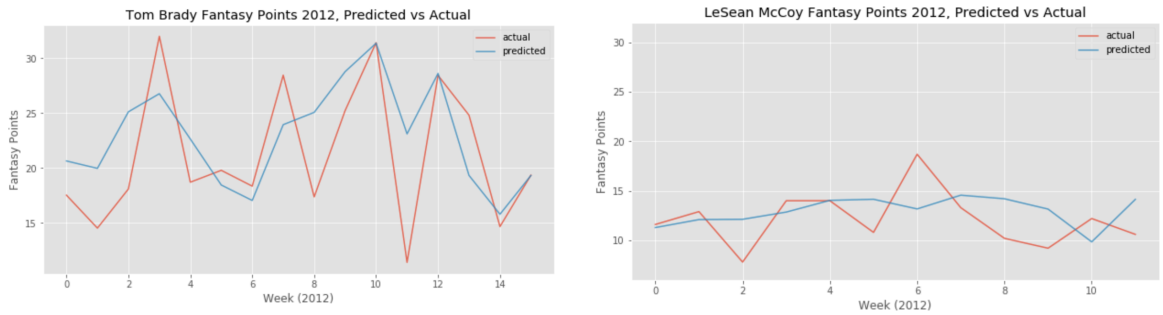

*Figures 2 and 3: 3-day moving average test set predictions, Brady (left) and McCoy (right)*

To build the baseline LSTM, we followed the same development process that will be outlined in detail below for the final LSTM model. At a high level, we built a simple sequential model in Keras with a single LSTM layer with 20 neurons and 50 epochs using MAE as the loss function and the Adam optimizer for each player. We then toyed with the number of epochs, the number of neurons, adding additional LSTM layers, adding dropout, changing the window size, changing the batch size, and adding regularization to improve the MAE attained on the test set. We additionally attempted a model using online training. To address random initialization of the network weights, we trained each model 25 times and averaged the MAEs attained. The best models that will be used as baselines for the final LSTM are outlined in table 1.

| Player | LSTM Layers | Dropout | Regularization | Epochs | Lookback Window | Batch Size | Online? | Avg RMSE | Avg MAE |
|---|---|---|---|---|---|---|---|---|---|
| **Brady** | Two layers; 100 + 20 neurons | 0.3 | 0.01 L1; 0.01 L2 | 250 | 5 | 5 | N | 5.65 | 4.43 |
| **McCoy** | Two layers; 30 + 20 neurons | 0.3 | None | 20 | 3 | 1 | N | 4.18 | 3.62 |

*Table 2: Average RMSE and MAE from baseline LSTM (no Twitter features) prediction in 2012 season.*

[19] Notebook: modeling_notebooks/09_baseline_mean_prediction




*Figures 4 and 5: Best baseline LSTM test set predictions, Brady (left) and McCoy(right)*

We note that the baseline LSTM model for Tom Brady is better than the naïve prediction, and that the chart of predictions versus actuals seems to show the network accurately learning some real patterns in his fantasy output. We have therefore in a sense already accomplished our first goal. However, the LSTM performs worse on McCoy and appears to systematically overestimate his performance. This may be indicative of an overfit model.

# III. Methodology

## Data Preprocessing

*Data Cleaning*

We began the data preprocessing step by cleaning each of our individual data sources. In modeling notebook no. 2,[20] we cleaned the player and game data. Specific steps we performed were:

1. For both player and game data, drop columns that will not be used for feature engineering.
2. For both player and game data, rename confusing columns (for example, change 'G#' to 'week').
3. Filter out game data that does not concern one of the players' teams.
4. Join player data and game data on date and team.
5. Transform 'home/away' and 'team_favorite _id,' two string columns respectively containing a flag of whether or not the game was played at home and a code indicating the favored team, into Booleans called 'home' and 'favorite' to indicate whether or not the player's team played at home and was favored.

After performing these steps, we were left with .csv files for each player that contained betting and weather information.

In notebooks 3 and 4,[21] we corrected a common issue in two of our base datasets. An important key for joins between our player, defensive, and Twitter datasets is the team abbreviation. Unfortunately, all of these datasets use slightly different abbreviations. In notebook three, we compared the player data abbreviations to the defensive abbreviations and used our NFL domain knowledge to align the two. In notebook 4, we compared player abbreviations to Twitter abbreviations and aligned those, too.

*Sentiment Analysis*

[20] modeling_notebooks/02_clean_game_data
[21] modeling_notebooks/03_clean_defensive_data and modeling_notebooks/03_clean_twitter_data

Next, we had to retrieve our Twitter text and conduct sentiment analysis. We retrieved the text from the tweet IDs in notebook 1, relying on a script provided by Martijn Pieters on Stack Overflow.[22] Pulling the text took about 5 hours and required Twitter API credentials.

We left-joined the text to the original Tweet data to preserve tweets that had been deleted, then moved on to the sentiment analysis (notebook 5).[23] First, we cleaned all the tweet text using Python's tweet-preprocessor package. The preprocessor's 'clean' function is designed to convert messy tweets into an NLP-ready format by removing URLs, hashtags, mentions, emojis, reserved words, and smileys.[24] Additionally, we stripped punctuation from the tweets, as this can often be arbitrary.

A cursory glance at our tweet text dataset revealed quite a few Spanish language tweets. We only downloaded the English version of Stanford Core NLP, so we wanted to exclude these from the sentiment analysis. In order to do this, we used Python's langdetect library to label the language of each tweet (a process that took another hour and a half).[25] Langdetect is not known for being the most accurate library, and indeed it classified a number of English tweets as other Germanic languages. We therefore ignored any output by langdetect other than Spanish, for which it seems pretty accurate. In the sentiment analysis, we just assigned a sentiment of 'neutral' to all Spanish-language tweets. We note that this might have biased some of our features that look at net sentiment or percent of neutral tweets if some teams have a higher proportion of Spanish-speaking fans than others. This is worth flagging for further research.

Finally, we downloaded and installed the English version of Stanford Core NLP[26] and booted up the server on our local machine. We used pycorenlp,[27] a Python package that serves as a wrapper for Stanford Core NLP, for sentiment analysis. Using the sentiment aggregation methodology outlined in the 'Algorithms and Techniques' section, we assigned a sentiment score to each tweet, assigning a score of 0 (neutral) to tweets that had been deleted. This process took almost 19 hours.

*Feature Engineering*

With our data preprocessed, we defined a series of functions to build features. We have five feature classes, so we built one aggregation function per feature class (notebook 7).[28] We'll outline the features built for each class below:

1. **Defensive Features**
    a. 1, 3, and 5 week moving average, minimum, and maximum of each defensive stat (fumbles, interceptions, passing yards allowed, rushing yards allowed, sacks, points allowed).

    Here, we simply took moving statistics for each defensive measure available to us in the weeks leading up to the game of interest. Logically, it makes sense that we would want to look at both long and short windows of defensive performance – we want the model to see both short- and long-term trends in defensive performance. The choice of windows and aggregation mechanisms to use was arbitrary. We took the approach of throwing more features at the model, as XGBoost and RNNs are algorithms that should be able to weed out inconsequential features fairly easily. This would nonetheless be a good candidate for future research.

---

[22] Martijn Pieters, Twitter API – get tweets with specific id, URL (version: 2019-11-11): https://stackoverflow.com/questions/28384588/twitter-api-get-tweets-with-specific-id

[23] modeling_notebooks/05_sentiment_analysis

[24] https://pypi.org/project/tweet-preprocessor/

[25] https://pypi.org/project/langdetect/

[26] https://stanfordnlp.github.io/CoreNLP/

[27] https://pypi.org/project/pycorenlp/

[28] modeling_notebooks/07_build_features

To build these features, we just calculated the moving statistics on the team defensive stats DataFrame and joined it to our base players DataFrame on opposing team and game date.

2. **Weather Features**
    a. Inclement weather flag
    b. Wind
    c. Temperature
    d. Humidity

Three of these features (wind, temperature, and humidity) were already available in the games data. The inclement weather feature was engineered as a Boolean with a value '1' if the 'weather_detail' was not blank and contained anything other than 'DOME' (indicating the game was played in a domed stadium). Given the small amount of data, it seemed sensible to have a single all-encompassing flag for inclement weather rather than trying to tease out the effects of rain versus snow versus what-have-you.

3. **Twitter Features**
    a. Number of tweets about player's team in the week before the game as a percentage of all tweets sent that week
    b. Number of tweets about opponent's team in the week before the game as a percentage of all tweets sent that week
    c. Percent change in percentage of tweets about player's team between three and one days before kickoff
    d. Percent change in percentage of tweets about opponent's team between three and one days before kickoff
    e. Net sentiment for player's team in the week before the game
    f. Net sentiment for opponent's team in the week before the game
    g. Percent of neutral tweets about player's team in the week before the game
    h. Percent of neutral tweets about opponent's team in the week before the game
    i. Percent change in net player team sentiment between three and one days before kickoff
    j. Percent change in net opponent team sentiment between three and one days before kickoff
    k. Percent change in percent of neutral tweets about player's team between one and three days before kickoff
    l. Percent change in percent of neutral tweets about opponent's team between one and three days before kickoff

Evidently, we took the 'kitchen sink' approach to feature development for the Twitter class, for reasons explained under the 'defensive features' section. The type of Twitter features we developed are defined by two dimensions: volume versus sentiment features, and total versus swing features.

As one might guess, volume features primarily concern the number of tweets sent about a player or opponent's team. Sinha, et. al.'s paper investigates volume features and found them useful for predicting NFL game outcomes. We had to normalize all count features by dividing them by the total number of tweets sent in a given period to control for the overall increasing volume of tweets. Sentiment features, explored in Schumaker, et. al.'s paper, focus on the net sentiment (or percentage of a certain sentiment) for a player or opponent's tweets.

Total features are those that look at all tweets for a player or opponent's team in a given week before a game. Swing features look at the change in tweets in the 24-hour period beginning three days before the start of a game and the period beginning one day before the start of a game. The suggestion of the usefulness of these features also comes from Schumaker, et. al.

To build the features, we start by filtering the whole Twitter dataset to include only tweets concerning one of the players' teams. We then group by the year-week combination that the tweet was sent and

perform the aggregation. Finally, we join the features back to the base player DataFrame on team and year-week.

4. **Player Features**
    a. 1, 3, and 5 week moving averages, mins, and maxes of fantasy points scored by player.

    We replicated the approach used to build defensive features here for points scored by our player. This is entirely in service of the XGBoost model; we did not include these features in the RNNs because they should be able to figure out the time-series of player points scored on their own.

5. **Betting Features**
    a. Spread
    b. Over/under line

    These features, too, required minimal engineering. Over/under line was already present in the games dataset. We transformed 'favorite_spread' to just 'spread' relative to the players team by making the spread positive if the player's team was not the favorite. This way, we could incorporate information about the relative quality of the teams and whether or not the player's team was favored into a single feature.

    Betting features are clearly important because they reflect the opinion of a wide variety of professionals on the outcome of the game. Like the Twitter features, they have potential to harness the 'wisdom of crowds.'

Finally, it must be noted that we saved all of these features without any additional transformations or missing value handling. This is fine for XGBoost, but we perform a bit of additional preprocessing in the LSTM notebooks: namely, scaling the features using scikit-learns MinMaxScaler and replacing missing values with the training set average.

## Implementation and Refinement

*XGBoost*[29]

The first models for us to implement were the XGBoost models. The primary purpose of these models was to understand feature importance, but given XGBoost's strength as an algorithm, we will also note performance.

First, we had to install the xgboost and shap packages – a nontrivial task as installation of xgboost on macOS (which the author runs) is notoriously finicky.[30] Having accomplished this, we read in our data, dropped games where the player did not play, and split the data into train and test sets based on a date (we chose May 1st 2012, as any games after this date would have been played in the 2012 season). We then transformed the training and test sets into xgboost's DMatrix objects for more efficient training.

We next instantiated an XGBRegressor object with the following parameters:

- Objective: 'reg:squarederror'
- colsample_bytree: 0.3
- learning_rate: 0.1
- max_depth: 3
- alpha: 10
- n_estimators: 30

[29] All modeling in notebook modeling_notebooks/10_model1_xgboost
[30] https://xgboost.readthedocs.io/en/latest/build.html; I actually was not able to get this working and decided instead to build the model on SageMaker!

For the objective function, we chose squared error. Of course, our primary success metric for the project was mean absolute error, not mean squared error. However, mean squared error is an objective that comes prepackaged with XGBoost – using mean absolute error would have required coding up our own function, which we did not want to spend time doing given that the primary purpose of this model is not to achieve the best performance but rather to better understand feature importance.

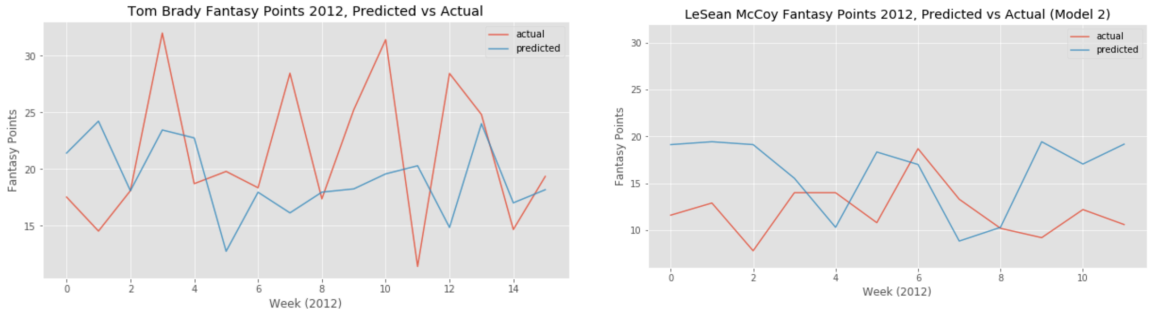
Given the tiny size of our training data (only 32 records for Brady and 30 for McCoy), we considered overfitting the principal risk of all our modeling efforts. The rest of our hyperparameters were therefore selected to be very conservative. colsample_bytree of 0.3 means that each tree will use only 30% of available features; max_depth of 3 means that any given tree will make at most 3 splits; n_estimators of 30 means we will build only 30 trees; and alpha of 10 means we will apply a heavy dose of L1 regularization (which tends to shrink the coefficients of unimportant features to 0 by penalizing the inclusion of additional features in the loss function).

Next, we fit the model for each player, made predictions on the test set, and computed MAE and RMSE. For Brady, the first model attained an MAE of 5.76 – a slight improvement over the 6.33 attained by the moving average predictions. For McCoy, however, the model only produced an MAE of 5.71, which was considerably worse than the moving average result of 2.65.

We did not spend much time attempting to refine these models, as performance was not the primary objective; we therefore did not attempt to perform any feature selection. Nonetheless, we took a moment to iterate through a series of different hyperparameter combinations, retraining the model each time to see if we could achieve better performance. For Brady, we were unable to do so, but we did eke out slight gains for McCoy. The hyperparameters and performance for the best XGBoost models is located in Table 3.

| Player | colsample_bytree | learning_rate | max_depth | alpha | RMSE | MAE |
|---|---|---|---|---|---|---|
| **Brady** | 0.3 | 0.1 | 3 | 10 | 7.33 | 5.76 |
| **McCoy** | 0.25 | 0.6 | 2 | 50 | 5.68 | 6.62 |

*Table 3: RMSE and MAE for XGBoost in 2012 season.*

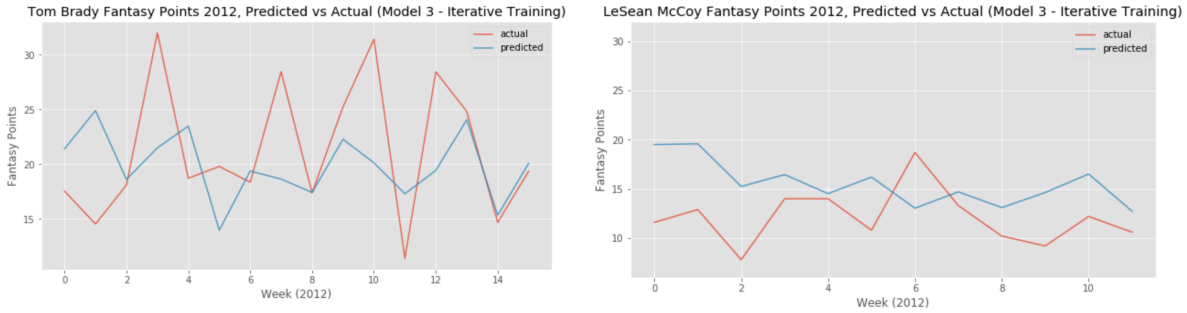



*Figures 6 and 7: XGBoost test predictions Brady (left) and McCoy (right)*

Finally, using the hyperparameters from the best models, we attempted online training. By increasing available training data, this approach significantly improved results for both players (though performance for McCoy was still well below the baseline). These results are displayed in Table 4.

| Player | RMSE | MAE |
|---|---|---|
| **Brady** | 6.33 | 4.88 |
| **McCoy** | 4.94 | 4.35 |

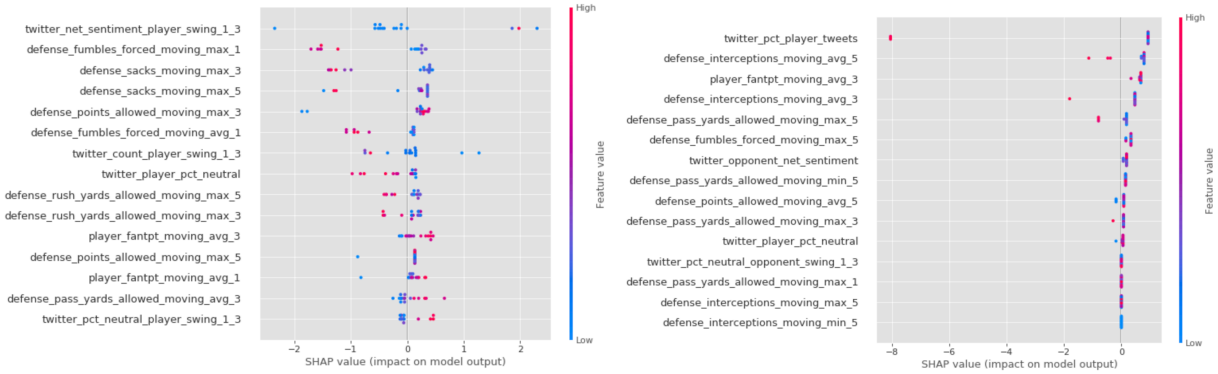
*Table 3: RMSE and MAE for XGBoost trained online in 2012 season.*




*Figures 8 and 9: Test predictions for best XGBoost model with online training, Tom Brady (left) and LeSean McCoy (right)*

It is worth noting here how seemingly off-base the predictions are for LeSean McCoy. Test set MAE is significantly worse compared to the moving average baseline, and looking at the chart of predictions, the model does not seem to be picking up on any real patterns (though, as one would expect, the online training model starts to get it near the end of the season). This makes us think that we may simply have a data problem – it could be that something about McCoy's performance fundamentally changed between 2011 and 2012, such that a model trained on earlier seasons does not generalize to 2012. More on this when we discuss final results.

We used the shap package to generate Shapley values on our test set for the best model trained without online training. Shapley values are calculated by removing features one at a time from a model, retraining, and observing how the model's output is impacted. We can therefore observe how severely the presence/absence of a feature with certain values impacts a model's prediction. Figures 10 and 11 display the test set Shapley values for both McCoy and Brady.




*Figures 10 and 11: Top 15 Shapley values for best XGBoost models for Tom Brady (left) and LeSean McCoy (right). The y-axis indicates the feature name, order by importance. The x-axis shows the feature's impact on model output. Red points indicate a higher feature value, while blue points are lower. For example, we can see that the second most important feature for Brady, defense_fumbles_forced_moving_max_1, causes scores to be more negative when the value is higher.*

Encouragingly, a few of the Twitter features appear in the top 15 features, with the swing in player net sentiment and the normalized count of tweets about a player being the top features for Brady and McCoy, respectively (of course, we should not pay too much attention the values for McCoy because of the model's poor performance). Three Twitter features appeared in the top 15 for Brady, and four for McCoy. The other strongest features, unsurprisingly, concerned the quality of the defense and the player's own performance in the past few games.

We have effectively confirmed that our Twitter features do add value, at least for Tom Brady. We will now move on to the LSTM model, which will hopefully lead to improvements in performance.

*Long-Short Term Memory Recurrent Neural Network*[31]

[31] Notebooks 11 – 13: modeling_notebooks/11_model2_lstm_baseline, 12_model3_lstm_all_features, 13_model4_lstm_all_features_pared

We implemented the LSTM RNN using Keras, which allows one to very easily define neural networks by stacking layers on top of one another.

After importing required packages and reading in the dataset, we defined the feature set to use for modeling. To build the 'baseline' LSTM (whose performance is reported in the 'Benchmark' section), we selected all betting features, all weather features, and all moving average defensive features (we left out the min and max features to reduce dimensionality). We then replaced missing values for each feature with the training set's mean and scale the features using scikit-learn's MinMaxScaler.

LSTMs expect to receive data as a 3-dimensional ndarray where the first dimension is the number of samples, the second is the lookback window size, and the third is the number of features. A lookback window is a hyperparameter that determines how many previous data points (or timesteps) are seen for a given prediction. For example, a lookback window of 5 would mean that we append the 5 previous records to each datapoint and then pass that group of 5 to the model for consideration. Note that a consequence of this is the larger our lookback window, the smaller our training set – if we have 32 data points in our training set but we have a lookback window of 3, we will end up with only 29 datapoint in our training set because the first three points do not have enough data before them to construct a lookback window.

In any case, we made a function that accepts a DataFrame and a lookback window and returns a new ndarray in the proper format. Only after constructing this new ndarray did we remove records in which the player did not play (because, as discussed above, we want the network to see games that the player did not play in, but not to make predictions on them) and split the data into training and test sets.

We began with the 'baseline' LSTM model that did not include any Twitter features. For both players, we started with a simple network architecture: a sequential model with a single LSTM layer with 50 neurons, trained for 50 epochs, with a lookback window of 1 and a batch size of 16 (the batch size is the number of samples considered before a step of gradient descent is performed; 16 seemed like a reasonable starting point because there are 16 games in an NFL season), and using MAE as the loss function and the efficient Adam optimizer.

Because of the small size our dataset, we were able to test many network configurations in a relatively short amount of time. So, to optimize the models, we built several networks for each player, each time toggling the number of neurons, the number of layers, the level of dropout, the amount of regularization, the number of epochs, the batch size, and the lookback window. After building each model, we considered two indicators: a chart of the training and validation loss (we used the test set for validation given our data constraints – see figure 12) and the MAE/RMSE. The former allowed us to identify the optimal number of epochs by finding the point at which test loss ceased to decrease; it also allowed us to detect if our model was overfit. The latter simply told us whether or not the model was performant.

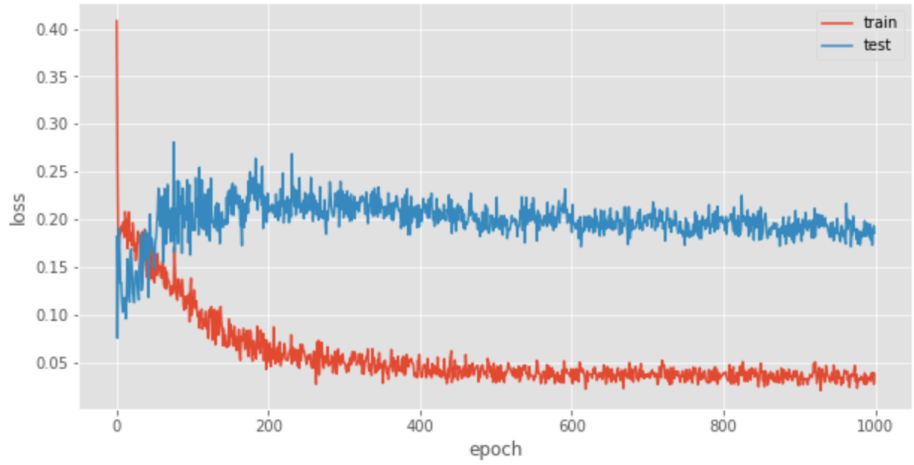


*Figure 12: Example loss history curve. Here, we see that test loss begins increasing after only a small number of epochs, indicating the model is overfit. We used this tool to help us determine the optimal number of epochs while training our networks, among other hyperparameters.*

The weights of neural networks are initialized randomly, meaning that two networks trained on the same data with the same architecture and parameters can have substantially different performance. In order to deal with this, whenever we found a promising model (modeled by MAE), we would train it 25 times and make a boxplot of the MAEs and RMSEs. We stored both the best and average MAEs returned; we used the average as the primary success metric.

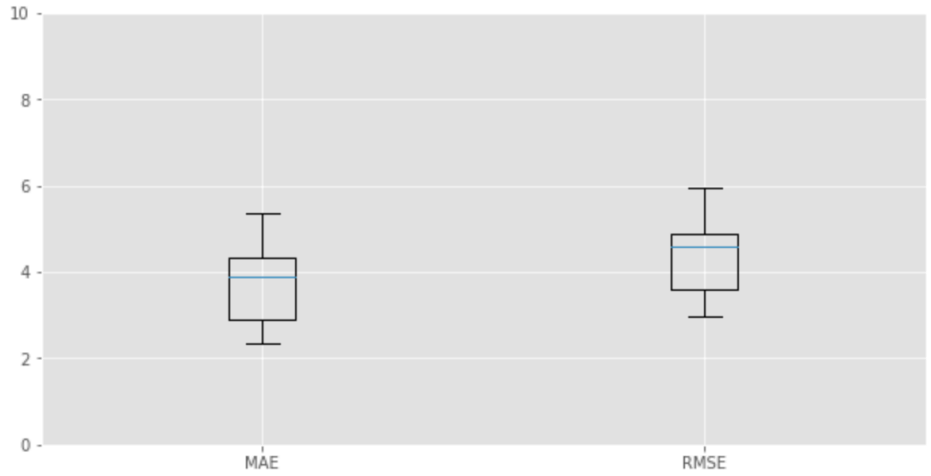



*Figure 13: Example boxplot for MAE and RMSE of neural network after 25 runs. We will use the average MAE as our primary success metric, but it is worth noting that depending on the weight initializations, performance can be significantly better or worse.*

After a while, we landed on the architecture outlined in the 'Benchmark' section. Using these parameters, we also attempted online training. The process of online training for a neural network is a bit different than for an XGBoost model: rather than rebuilding the entire model after adding each new datapoint, we train the existing model for a few more epochs after adding each data point, thus updating the weights. To do this, we need to make a few changes to the network architecture: namely, batch size must be one (because each additional data point we add will be treated as a batch), and the model must be 'stateful' so we can control when the cell states are reset. Default neural networks in Keras are stateless, meaning the state of their gradients will automatically be reset after every batch. For online training, we want to reset cell states after each epoch, but not after each batch (remember, a batch is just one record!).A stateful model allows us to do this manually.

The models trained online did not perform as well as the offline models for either player. We will therefore focus on the LSTMs trained offline for the remainder of the paper.

To build the final model, we repeated the exact same process outlined above but added all of the Twitter features. We began iterating on model parameters from the best parameters discovered in building the baseline model. In other words, instead of reinventing the wheel, we built off what we had discovered in training the baseline model. Ultimately, we found that the same architecture was most effective for each player.

Unfortunately, we found that the model with Twitter features was actually less effective than the baseline LSTM for Tom Brady (Table 4; it was slightly better for McCoy)! Because of our concerns about overfitting, we began experimenting with feature selection. Returning to the feature importance gleaned from our XGBoost model, we attempted forward selection, but found that results generally worsened if we removed too many features. As a compromise, we simply selected the top Twitter feature for each of Brady and McCoy – player sentiment swing and normalized count of player team tweets – and added them to the baseline feature set for both players. We found that this led to a modest boost in performance, and kept the resultant models as our final solutions.

| Player | RMSE | MAE |
|---|---|---|
| **Brady** | 5.80 | 4.91 |
| **McCoy** | 4.17 | 3.66 |

*Table 4: RMSE and MAE from LSTM model with baseline features + all Twitter features.*

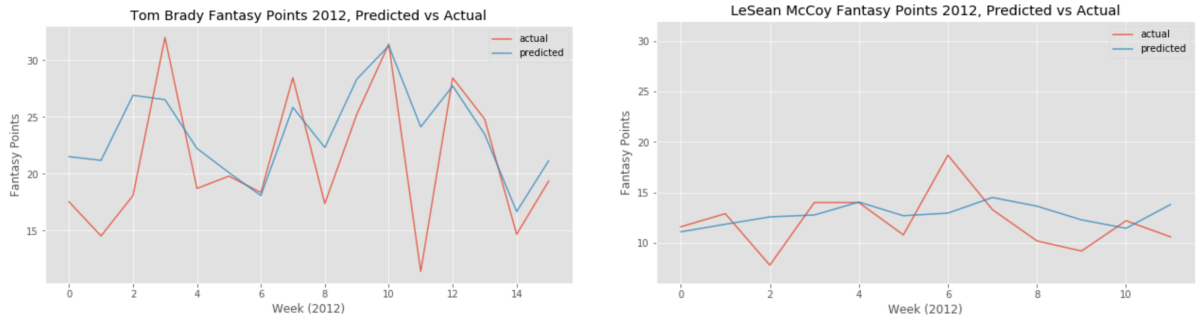
# IV. Results

## Model Evaluation, Validation, and Justification

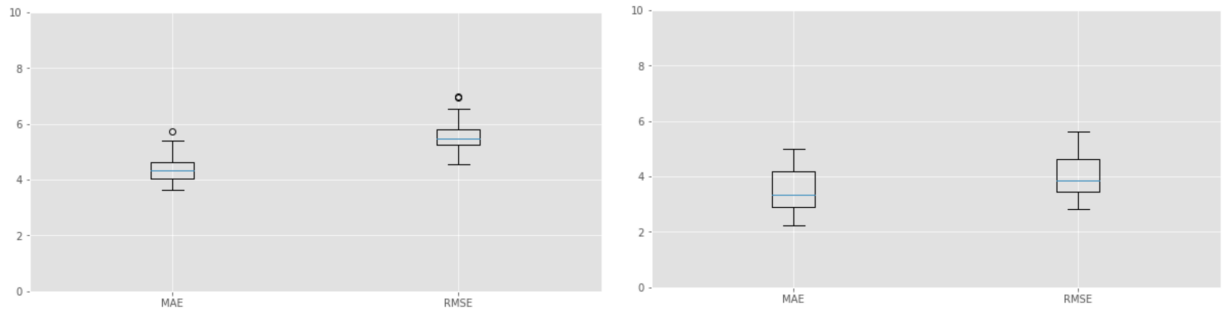
The final model architecture and results are presented below:

| Player | LSTM Layers | Dropout | Regularization | Epochs | Lookback Window | Batch Size | Online? | Avg RMSE | Avg MAE |
|---|---|---|---|---|---|---|---|---|---|
| **Brady** | Two layers; 100 + 20 neurons | 0.3 | 0.01 L1; 0.01 L2 | 250 | 5 | 5 | N | 5.56 | 4.41 |
| **McCoy** | Two layers; 30 + 20 neurons | 0.3 | None | 20 | 3 | 1 | N | 4.03 | 3.44 |

*Table 5: Average RMSE and MAE from final LSTM model predictions in 2012 season.*




*Figures 14 and 15: Test predictions best final LSTM models Tom Brady (left) and LeSean McCoy (right)*




*Figures 16 and 17: Test predictions best final LSTM models Tom Brady (left) and LeSean McCoy (right)*

The final feature list was:

- player_played
- player_home
- betting_over_under_line
- betting_spread

- weather_temperature
- weather_wind_mph
- weather_humidity
- weather_inclement
- defense_fumbles_forced_moving_avg_1
- defense_fumbles_forced_moving_avg_3
- defense_fumbles_forced_moving_avg_5
- defense_interceptions_moving_avg_1
- defense_interceptions_moving_avg_3
- defense_interceptions_moving_avg_5
- defense_pass_yards_allowed_moving_avg_1
- defense_pass_yards_allowed_moving_avg_3
- defense_pass_yards_allowed_moving_avg_5
- defense_rush_yards_allowed_moving_avg_1
- defense_rush_yards_allowed_moving_avg_3
- defense_rush_yards_allowed_moving_avg_5
- defense_sacks_moving_avg_1
- defense_sacks_moving_avg_3
- defense_sacks_moving_avg_5
- defense_points_allowed_moving_avg_1
- defense_points_allowed_moving_avg_3
- defense_points_allowed_moving_avg_5
- twitter_pct_player_tweets
- twitter_net_sentiment_player_swing_1_3

We chose these models because they were the best performing models that included any of our Twitter features.

In order to ensure robustness of the models, we shifted the entire dataset to the left (in other words, we dropped the first training observation and added the first test observation to the training set) and retrained. The results obtained were very close to the models presented above, with average MAEs of 4.36 for Tom Brady 3.48 for LeSean McCoy.

The results are quite interesting. For Brady, the LSTM solution both with and without Twitter features worked excellently. Even with a very aggressive network (100 neurons in the first layer and 250 epochs!), the model did not overfit to the training data. Figure 14 comparing predictions to actuals in the test set shows that the model was clearly able to pick up on some real patterns in Brady's fantasy output.

To test the statistical significance of our results against the moving average baseline, we conducted a one sample t-test with the distribution of MAEs attained by the 25 networks trained for the final model as the sample and the moving average MAE as the population mean.[32] The p-value produced was well below zero, indicating a statistically significant improvement over the moving average model. To test against the baseline LSTM, we conducted a dependent two-sample t-test where the two samples were the 25 MAEs produced by each model. Here, we found the difference to be insignificant – unsurprising given the fact that the difference in average MAEs was only 0.02 points. Ultimately, though the final LSTM model was very good, it does not appear that the Twitter features were able to significantly improve model performance.

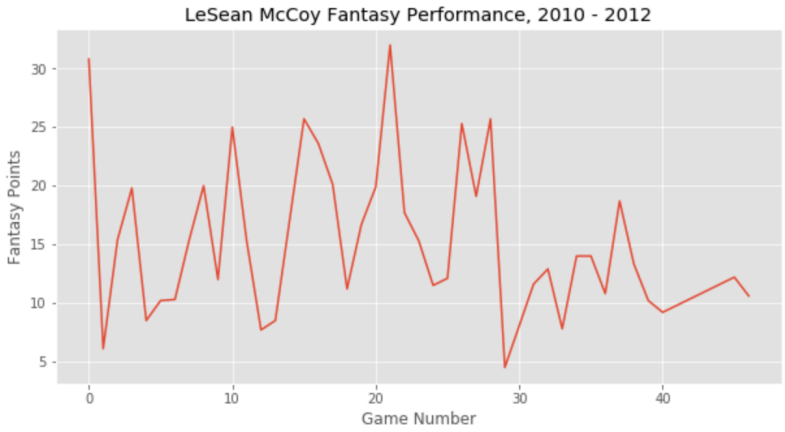
For LeSean McCoy, we have quite a different story. We saw a greater improvement in the LSTM model compared to the baseline when we added Twitter features than for Brady – 0.18 points. This difference was also not statistically significant, but it should not be ignored, either, because very small improvements in performance can make big differences in things like betting and stock prediction where the slightest edges, when amplified by leverage, can lead to large returns. The LSTM model also beat the best XGBoost model by almost a whole point, validating the approach.

That said, none of our models came close to beating the simple moving average approach for McCoy. We were worried this might be the case when we saw how strong that prediction was for McCoy – recall the

[32] Notebook 14: modeling_notebooks/14_statistical_comparison

same approach produced an MAE of 9.96 for Todd Gurley, the running back from Chris Zita's analysis that we substituted with LeSean McCoy, versus McCoy's 2.65.

Zita's original impetus for choosing Brady and Gurley was that the former was a consistent player, while the latter was inconsistent. We assumed at the outset that he meant consistent/inconsistent *within a single season*, and indeed that seemed to be the case for the seasons he examined. We attempted to replicate this setup by choosing earlier seasons for Brady and Gurley analogue LeSean McCoy. However, what we got were three seasons where Brady was *inconsistent within seasons, but consistent across them*, and where McCoy was the opposite. It is clear from figure 15 that Brady's performance in the 2012 season was quite varied; however, our model was able to understand that variance because it was present in the 2010 and 2011 seasons. McCoy, on the other hand, was relatively consistent in 2012, but his 2012 season did not at all look like his 2010 or 2011 seasons (figure 18).




*Figures 18: LeSean McCoy fantasy points scored, 2010 - 2012*

Ultimately, we cannot expect a machine learning model to perform well on a dataset where the test set fundamentally does not resemble the training set. While selecting a candidate model for McCoy, we noticed that every attempt to perform deeper learning by increasing the number of neurons or epochs quickly led to severe overfitting. The reader will notice that the final model's predictions are a fairly vanilla straight line – this is because the patterns found in the training set predicted much better fantasy performance than was actually had in the test set, and thus the best a model could do is get close to an averaging approach.

# V. Conclusion

All in all, our results are a bit of a mixed bag. We list our primary takeaways below:

1. **LSTMs are a good approach for fantasy football player prediction, provided one has adequate training data.** The LSTM approach was very impressive compared to the moving average and XGBoost approaches for Tom Brady. Moreover, it was able to find real, non-obvious patterns in his performance despite being trained on only 27 data points! This is important – the average football career is only 3 seasons, so we need to be able to make good predictions with little data. On the other hand, the LSTM did not perform well on LeSean McCoy because his performance in 2010 and 2011 did not predict his performance in 2012. As the adage says – garbage in, garbage out.

2. **We do not have conclusive evidence that Twitter data can improve the performance of a model built on more traditional football features.** Unfortunately, the Twitter features did not improve the performance of our LSTM model, even for Tom Brady. When we added all of the Twitter features, they actually degraded model performance, suggesting that some of the Twitter

patterns present in the training set did not generalize to the test set. Nonetheless, we remain optimistic about the potential of Twitter features. For one thing, they appeared as top features in both player XGBoost models, suggesting they were strongly predictive in the training set. Second, caveats abounded with the type of Twitter features we used. Between the missing data, normalization, imperfect sentiment analysis, and reliance on team-level tweets rather than player-level tweets, our Twitter features were far from perfect and we feel that more research in this area is needed.

Our two goals in this project were to construct a baseline RNN that significantly beat a moving average approach for individual player prediction in fantasy football, and to improve upon that baseline using Twitter features. Despite the hiccup with McCoy, we consider the first goal accomplished, and we have additionally learned that in a production environment, we would need to constantly monitor our models in case there is a significant change in our player's performance patterns. We did not accomplish the second goal, but we believe that the door remains wide open.

In a future study, we would suggest the following next steps:

1. Collect more recent tweet data using the streaming API and save all text locally to preserve a complete dataset.
2. Try building a custom sentiment analysis model just for NFL tweets.
3. Collect player-level tweets in addition to team-level tweets. We recommend continuing to collect team-level tweets as well to construct Twitter features for the defense.

Fantasy sports are a fun and potentially lucrative pastime. Even with the mixed results attained here, we can see clearly that machine learning has a lot of potential to augment our understanding of and skill at the game.