Matthew Fox
CS229
Final Project Report

Beating Daily Fantasy Football

Introduction

In this project, I've applied machine learning concepts that we've covered in lecture to create a profitable strategy for daily fantasy football. I'm a huge football and fantasy football fan, and since learning more about machine learning in this class, I saw a lot of room for machine learning approaches in daily fantasy football. Daily fantasy football provides an opportunity for fans across the country to build a lineup given a set budget and costs per player that the fan believes will have a high point total calculated from the player's stats in a given weekend. Companies such as DraftKings and FanDuel provide a marketplace for fans to create these lineups and play against each other.

The premise of this project is that a machine learning algorithm can learn to create higher scoring lineups more consistently than the average fan and consistently enough to overcome the rake that either DraftKings or FanDuel takes for sponsoring the competition. In order to create these predictions, I've modeled each skill position player's expected points in a given weekend using separate linear regression models trained from a database of previous performances. From this information, I then created a constraint solving algorithm to pick many advantageous lineups that are beneath the overall budget and greater than the expected cutoff for making profit in a given competition.

For the linear regression algorithm, the inputs were different based on each skill position player. For quarterbacks, the inputs features were {passing yards, passing tds, passing attempts, completions, interceptions, and fantasy points}. For receivers (wide receivers and tight ends), the input features were {receiving yards, receiving tds, receptions, and fantasy points}. For running backs, the input features were {rushing yards, rushing tds, rushing attempts, receiving yards, receiving tds, receptions, and fantasy points}. For each of the inputs listed, there were actually three features that went into the algorithm - the stats from the last game that a player recorded fantasy points, a moving average of the last 3 games, and a moving average of the last 5 games. The output from each of these algorithms was the projected number of points that a player would get based on the Draftkings PPR scoring rules provided on Draftkings' website [1].

Related Work

From some preliminary research, a lot of work has been done on predicting football game outcomes to try to beat a Vegas set spread, but much less work has been done on daily fantasy, likely due to its recent entrance into mass popularity. From projects focusing on the Vegas spread, much more data is used about team performance rather than player performance because the outcome is based on team performance. However, I did find a couple of recent research projects that provided some insight into this problem: [2] , [3]. In the project described in [2], a former CS 229 student did a very similar project centered around fantasy basketball. The algorithms used in [2] were a linear regression much like I created for my project, and a Naive Bayes classifier. Another reason that I ended up focusing on linear regressions for my project was some of the analysis present in [2] that showed that linear regression worked considerably better than Naive Bayes or DraftKings in error rates.

In the project described in [3], the author tried to do a take a similar approach to my thesis, but used a genetic algorithm based on which players were picked in successful lineups rather than an algorithm to predict each player's points and then pick a lineup through a constraint solver. The nice aspect of predicting individual player's points is that the algorithm can generalize to many different tournament structures with slightly different optimal lineup strategies. Furthermore, the testing size for [3] was way too small to get any sort of meaningful results from the data. The author tested 25 lineups from his genetic algorithm to get the results, which could be highly skewed compared to the average results.

Dataset and Features

In order to create this algorithm, I first had to find suitable historical data that would correlate to the actual fantasy scores. There are a lot of websites that provide this historical data, but it is rare that they provide a database or a downloadable csv file for a reasonable price. I had imagined that I would use Pro Football Focus [4] as my dataset, as I had heard that they keep good historical football data. However, the files available for download were on a per player basis. The data was pretty good, but I would have to write some sort of web scraper to download the data that I was interested in. Since I don't have experience with web scrapers, I decided that that was best left learn another time.

From here, I did some research into available databases for football statistics. I ran across Pro Football API [5] and Fantasy Data [6]. Unfortunately, while these looked promising at first, Pro Football API doesn't have a fantastic API (ironically), requires PHP with which I'm not familiar, and required pings to their website for all queried data. Fantasy Data wanted $1299 for each season's historical data which was clearly out of my price range.

Finally, I stumbled upon nflgame, an open-source python library which supports both real-time data gathering and historical game data dating back to 2009 [7]. This package comes with all data up until the current week downloaded alongside the package, so each query is much shorter, only accessing local data unless you are depending on real-time results. The downside to nflgame is that some of the features that I had hoped to use are not available or easy to impute from historical data. Some of these features included opposing team's defensive statistics, weather projections, and injury reports. Furthermore, gameflow specific statistics (such as yards in the 4th quarter) are not easily accessible and prove expensive to calculate for every data point.

After settling on nflgame for my data source and getting comfortable with the API, the next design decision to tackle became how to preprocess the data to get a standard (X, Y) dataset as we have seen in class to provide to the feature selection algorithms and to the final regression algorithm. The dataset that I gathered included data from 2009-2016. I decided that I would leave 2016 as a test set since I have some daily fantasy results from 2016. Furthermore, I decided to leave out weeks 1-6 and week 17 from the dataset because we either will not have enough up-to-date data (weeks 1-6) or can't assume valuable results (week 17 due to players resting for the playoffs).

Next, I had to decide which features would be valuable for each position I would like to predict. Currently I have preprocessed 3 datasets: QB_data, WR/TE_data, and RB_data. For each of these, I've extracted the major 3 stat categories per position (yards, tds, attempts/receptions) as well as interceptions for quarterback. I've also constructed sliding windows of 1, 3, and 5 games to collect these stats to have some ability for recency bias and some ability for normalization across recent weeks. Finally, I wrote a script to take the current week's results and output the actual fantasy points earned according to DraftKings' rules for fantasy scoring in order to get the Y result for a player's features.

The last design decision was which players from a given week to include as data points. It didn't make sense to include all players who had any statistics in a given week, since this balloons the number of points to gather, skewing towards very low statistical totals. Since the intent of this project is to predict very high-scoring lineups, I also didn't want to skew the dataset towards lower outputs. Therefore, for each week I decided to grab the top performers from each category (receiving, rushing, and passing) in order to learn which statistics create top performers. However, in order to not imbalance my dataset only to high performers such that my algorithm will consistently overpredict, I tailored the window of top statistical performers to the position (I chose the top 60 WR/TEs in a given week, the top 25 RBs, and the top 25 QBs). These ranges are consistent with the range of players which are reasonable to evaluate in a given week for a DraftKings lineup.

In order to create the data for future use, I wrote a python script to choose data points, build the feature sets, calculate the point totals, and output a CSV file of the results. I currently have approximately 1500 QB data points, 1500 Rb data points, and 3500 WR/TE data points. This dataset size proved sufficient for learning and provided a reasonable dataset with regards to compute time.

Methods

As I mentioned above, I used three separately trained linear regression models to create my predictions. The linear regression model is based off of a least-squared error minimization optimization problem that solves for coefficients to multiply all of the features in order to get as close as possible to the output. The really great thing about linear regression is that there is a closed form to solve for these theta values (from the lecture notes):

$$\theta = (X^T X)^{-1} X^T \vec{y}.$$

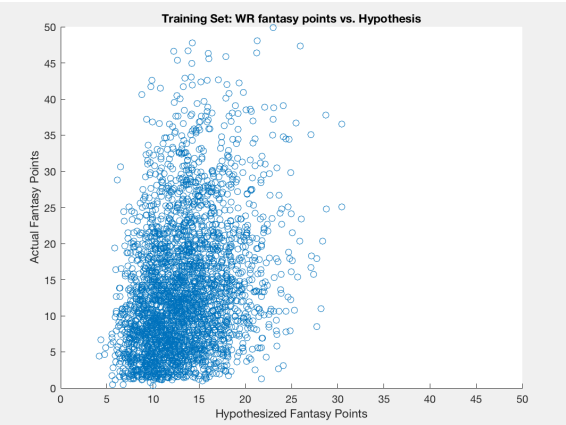
Generally in linear regression, you also add in one constant term for each feature vector in order to capture some underlying y intercept term for the data. I'll speak a bit more to the reason that I didn't include this constant term in the results section, as I originally started with this term, but I eventually decided to remove this term from the standard linear regression algorithm.

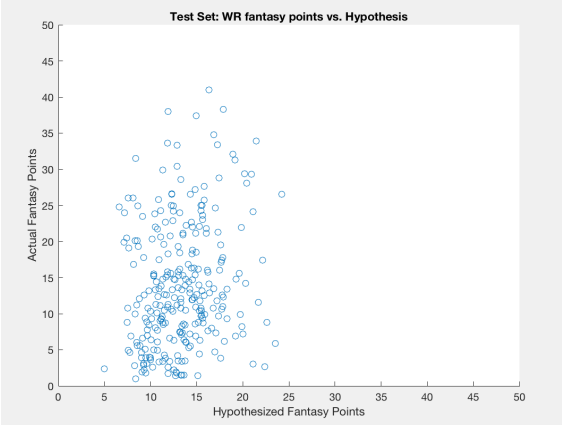
The other major algorithm that I used was a constraint solving algorithm to pick high performing lineups from all of the possible lineups I could produce. The search space for this problem is much too large to brute force (I began to calculate the total lineups I could predict, but stopped when it reached the billions), so I had to be a bit more clever. I ended up using a randomized algorithm which would select a random grouping of players according to DraftKings positions, and then compare this lineup against the cost constraint. If it were a legal lineup, I would see my total predicted scores and see if it was above some threshold that I tuned to attempt to make between 150 and 250 lineups given a reasonable running time.

Experiments/Results/Discussion

The results of this project are classified in two ways: the average error of the three linear regressions and the percent of risked money won by betting on these lineups from preliminary results. It's important to note here that I would not expect this algorithm to achieve any sort of low average error because there are inherently way more variables to a football player's performance than I am modelling with just past performance.

I initially ran linear regression on all three datasets, and plotted the results. The graphs looked very skewed to players with low actual fantasy scores. Furthermore, it seemed from the theta values that the biggest contributor towards the predicted score was the y-intercept, meaning that the algorithm didn't really learn good indicators for correlating previous stats with scores. Also, some statistics that should clearly have a positive correlation to fantasy score (like passing yards in the last game) had negative theta values, which did not make intuitive sense. After thinking about this some, I decided to remove the y-intercept term to force the algorithm to use the features to come as close as possible to the actual fantasy scores. I also realized a big mistake in my data pre-processing: I hadn't considered bye weeks in the nfl where players never will score any points. Given this, I decided to throw out any training data point where the player gained 0 fantasy points in a week (because the nflgame library didn't provide bye weeks). After this step, I got the following results (just showing WR results for space reasons):

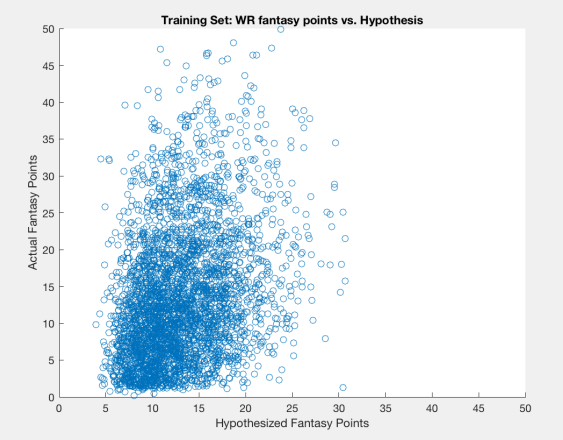


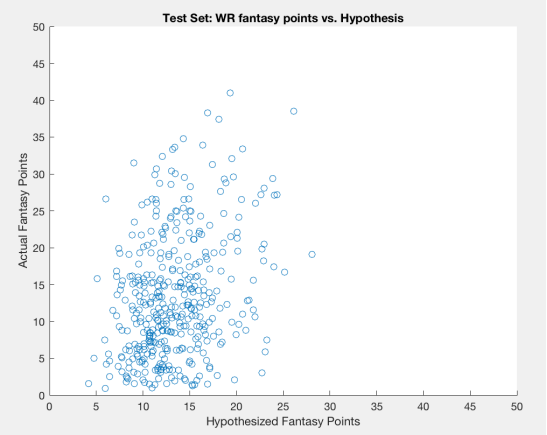


The ideal version of these graphs would be a line looking like x=y. The average error for the training set and test sets were:

	WR/TE	RB	QB
Training Set Error	6.8455	6.8234	6.5059
Test Set Error	6.8133	7.5226	6.1430

I then realized that I also needed to pre-process out bye weeks from the sliding windows of games. This meant that I would throw out the most recent week where any player scored 0 fantasy points and calculate the sliding windows without this week. At this point, I also added in previous week's fantasy scores to try to improve the predictions. These improvement produced the following analogous graphs and errors:





	WR/TE	RB	QB
Training Set Error	6.8081	6.9385	6.4598
Test Set Error	6.5100	7.3394	6.1655

With the new theta values for my three linear regressions, I grabbed the proper data from last week's draftkings results and player salaries. I ran 2 million possible lineups through my constraint solving algorithm and picked the 130 lineups which produced the highest predicted scores. I wrote a function to convert the actual score of a lineup into the amount of money it would have made in the most popular progressive tournament that DraftKings provides (approximately 500,000 lineups are entered). In total, the algorithm risked $390 and would have won back $431, a winning percentage of 10.5%. Unfortunately, since DraftKings tournament data from previous weeks is notoriously difficult to come by, I wasn't able to run the monetary part of this algorithm across more than this past weekend. I plan to continue a study on the effectiveness of the suggested lineups once I accumulate more DraftKings results.

Future Work

I plan to continue to develop on the ideas presented in this project in a few ways. Firstly, I would like to analyze entering different types of competitions (where the prize distributions vary in possibly beneficial ways). Unfortunately, data for previous draftkings tournaments is very hard to come by. In order to analyse entry into different types of competitions, I'll need to generate a dataset by scraping the DraftKings results links weekly before the results are taken down. Secondly, I was unable to find a suitable dataset that included as much information as I would have liked to build up a complete feature set that could result in a more accurate algorithm. I plan to continue searching for a more suitable dataset as well as begin working on a web scraper to either grab data from Pro Football Focus or from ESPN. Next, I would like to add a model to predict scores for defense and special teams so that I can also optimize for a high scoring pick in this field. Finally, I would like to improve the constraint solving algorithm to use some sort of alpha-beta tree in order to sort the lineups that I believe to be the most promising and then pick the highest predicted lineups.

Overall, I believe that his project has shown the promise that I had hoped it would when I proposed this area of research. Although the results are far from conclusive to show that this algorithm can consistently beat daily fantasy, they certainly motivate future research into results and into iterations to improve the algorithm.

References/Bibliography

[1]"Fantasy Football Contest Rules & Scoring | DraftKings." DraftKings - Daily Fantasy Sports for Cash. N.p., n.d. Web. 16 Dec. 2016.
[2] Hermann, Eric, and Adebia Ntoso. "Machine Learning Applications in Fantasy Basketball." Stanford University, 20 Dec. 2015. Web. 5 Dec. 2016.
[3] Ruths, Troy. "Double Yo Money." N.p., 24 Sept. 2015. Web. 5 Dec. 2016.
[4] Pro Football Reference. N.p., n.d. Web. 5 Dec. 2016.
[5] "ProFootballAPI.com." Live NFL Stats API - ProFootballAPI.com. N.p., n.d. Web. 16 Dec. 2016.
[6] "We Collect and Distribute Sports Data in Real Time to Power Your Website and Mobile App ." NFL Fantasy Football Data Feed API | FantasyData.com. N.p., n.d. Web. 16 Dec. 2016.
[7] "Nflgame 1.2.20 : Python Package Index." Nflgame 1.2.20 : Python Package Index. N.p., n.d. Web. 16 Dec. 2016.