Predicting Winning Lineups for Daily Fantasy Football Leagues

Alexei Bastidas (alexeib@stanford.edu) & Ingerid Fosli (ifosli@stanford.edu)

Introduction

Over the last decade, the popularity of fantasy football has risen to the point that multiple betting sites, including FanDuel, have propped up where users can set a lineup and compete against other players with the promise of cash prizes. The goal of our project is to create optimal lineups for FanDuel by using predictions for the fantasy points earned per player. The predictions are generated as part of our project for another class (CS 229). With these lineups, the ultimate desire is to gain a competitive edge so as to be able to consistently turn a profit from betting. The biggest challenge with said model is that, unlike a sport like baseball, football is inherently dependent on pitting two 53 man teams against each other - thereby making individual predictions not as accurate, as well as the large number of possible lineups that need to be considered.

Task Definition

A lineup consists of one quarterback (QB), two running-backs (RB), three wide receivers (WR), one tight end (TE), one place kicker (PK), and one defense (Def). The main constraint is a salary constraint - each player that can fulfill a position is worth a certain salary. FanDuel sets player's salaries every week, and the total salary of your lineup can be worth at most $60000. In addition, FanDuel restricts the lineup to having at most 4 players from one team, and at least 3 teams represented. Given the dataset below, we want to predict the ideal lineups to bet on. We want to bet in tournaments where the top 50% of competitors win what they bet, which has fairly low risk and reward. Ideally, we would produce the model with the highest probability of being in the top half - instead, because of difficulties in modeling risk accurately, we try to find the lineup with the highest number of expected points. Still, we consider the model to be very useful if the lineups produced return net profit in these double-up tournaments.

Dataset

We built our own dataset by leveraging Python's BeautifulSoup and Urllib packages to scrape data from: FootballOutsiders(FO), RotoGuru(RG), NFL Statistics(NFL), and FanDuel(FD) from 2011 to the current NFL season (2016). We use our player models from our 229 project to generate point predictions, both using regression to get the predicted production for each player, as well as classification to get the probability that a player scores in each of the following

buckets: 0-5, 5-10, 10-15, 15-20, 20-25, 25-30, 30+ points. From the classification data, we can also calculate an expected production.

Our formatted dataset takes in a week and year, and, for each position group, stores a dictionary entry for each player. The key is the player name and the value is (team, expected points, salary) or (team, expected points, salary, probability of being in each point bucket), depending on if the probability distribution is desired.

Using our models, we will take this dataset and generate an optimal lineup for that week/year.

Approach

In addition to the baseline, we used two main models for generating lineups - a CSP and an MDP. For the MDP, we tried both using a greedy algorithm that didn't take our predicted points into account, and estimating rewards the probability distribution of the buckets.

Baseline and Oracle

Our baseline uses a simple greedy algorithm to make a lineup. We leverage the data provided weekly through FD, containing a player's name, position, average FPs per game(AFPG), and salary, to calculate this. Our greedy algorithm progressively finds the best, as defined by highest AFPG, QB, then the best RB1, etc. This process is done regardless of cost - once we start running into constraints, we pick the best possible player that fits into our working salary, where working salary is defined as `remaining salary - 4500 · # positions remaining` to ensure that all positions are filled. The baseline does not take into account opponents, supporting cast, etc - it predicts a lineup strictly based off averages, subject to the cost constraint.

For the oracle, we use our CSP to generate lineups, but instead of using the predicted points, we use the actual points earned to generate the best lineup within the constraints. This doesn't necessarily give the best lineup, since it only considers the top 5 players for each position, but it always gives a highly ranked lineup that will comfortably win a double-up tournament.

Constraint Satisfaction Problem

The CSP uses exact predictions of fantasy points to maximize the total expected points, given the constraints. The initial variables are the positions that need to be filled on a lineup, with the correct multiplicity. For example, the variables `('RB', 0)` and `('RB', 1)` represent the two running backs in the lineup. As input, we only take the top 5 players from each position. We found this to be a good balance of optimal predictions and the runtime; the CSP runs in about 2 minutes per week on a regular laptop during normal use, and generally produces a lineup with predicted points within one point of the maximum.

Constraints

No repeating players

This is represented as a binary factor between every pair of two different variables to ensure that each player only occurs once on the lineup. This factor should only be needed explicitly for variables referring to the same position, as a player should only have one starting position. Currently, it's implemented for every pair of variables, since there are only 36 pairs, for simplicity in code, and just in case a player can have multiple starting positions.

Weights for expected fantasy points

We use a unary factor for each variable to represent the expected production. It returns 2^(the expected number of points earned per player). This way, the total weight of an assignment is 2^(the total expected number of points of the lineup), so the assignment with the best weight will have the highest predicted number of points.

Salary constraint

Initially, we attempted to fulfill the salary constraint by using a sum variable, built on additional variables, unary, and binary factors. However, since it generates tables for all possible values in the domain, and the domain is very large, this method is unreasonable. Instead, we calculate the sum every time we check the weight of the states. This adds some additional time, but saves a lot of memory.

Team constraint

The fanduel lineup allows at most 4 players from each team in the lineup, and at least 3 different teams must be represented. This constraint hasn't been implemented, since the CSP is already rather slow, and none of the lineups suggested have broken this constraint.

Algorithm

The algorithm used is a modified version of the backtracking algorithm from the Scheduling assignment. Instead of storing every new lineup, it only stores an assignment if its weighted to be almost as good as the best lineup so far (.9 of the weight). This saves a lot of memory, since there are a lot of valid lineups that don't come close to the highest point prediction, even when just looking at the top 5 players in each category. Notice that the salary constraint is implemented as a function, instead of in the unary and binary constraint tables.

Markov Decision Process

An MDP used the rewards associated with certain states and the probability of achieving a state with a certain action to try to maximize the reward. Our MDP representation of a lineup generator ultimately uses the probability of a player's production being in each of the point

buckets (5-10, 10-15, 15-20, 20-25, 25-30, 30+) as the probability of reaching certain states, and the expected production of each point bucket as the reward. However, doing so directly generates a lot more states than we can handle, so a reward isn't assigned to a lineup until it's complete.

Formal Definition

State

In this model, we define a state as ([Is Final State?], [Current Salary], [Current Lineup By Position Group], [Current Counts of Players in Each Team], [Final Production]). The final production is only updated if the state is an end state (ie the entire lineup is chosen) to reduce the number of possible states.

Reward

The reward is -Infinity if the state is invalid - either we go over the salary cap, or take too many players from one team. If we've reached a valid final state, the reward is the total lineup production at said final state, for the standard MDP, or just 0 for the greedy MDP. To calculate the total lineup production, we combine the probability distributions of each player in the lineup to generate the probability distribution of the lineup. We use the median of a point range (or 35 for the 30+ range) as the reward for a single player being in said range. Then we return all unique combinations of expected points and their associated probabilities. By only returning a reward after picking the lineup, we manage to both avoid bloating the intermediary states with 7 choices per player instead of just one, and we manage to avoid calculating the probability of each player being in each point range, by taking advantage of only needing to know the sum.

Actions

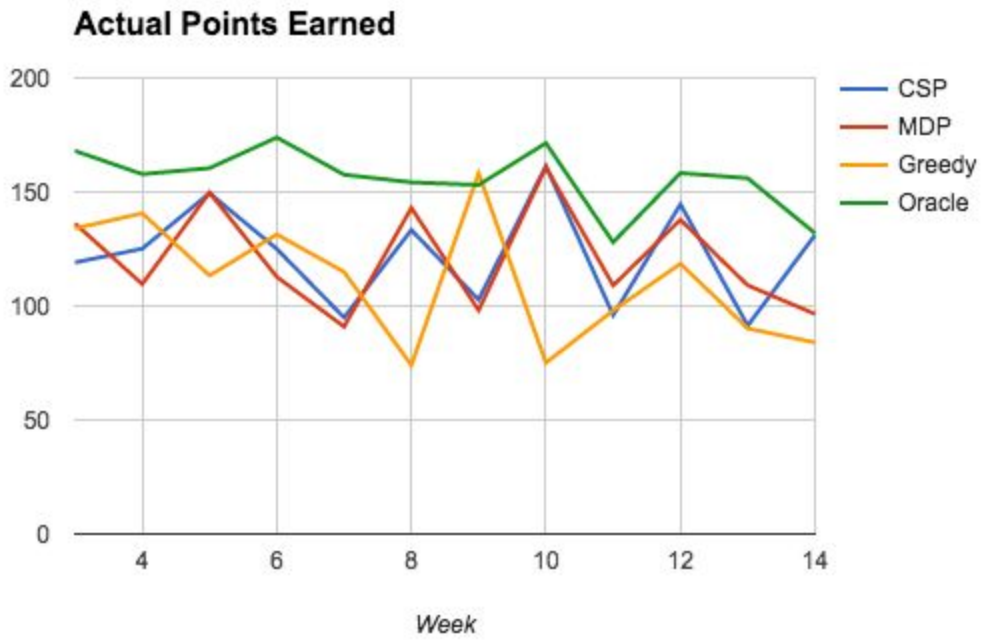
Until the lineup is set, an action involves choosing a player. We choose players in order of position to reduce the number of possible incomplete lineups. The reward is always zero unless we reach an end state. An end state will either have a complete lineup, or an invalid lineup, and always only the action 'end'. The rewards of the end states are described in the rewards section.

Algorithm

We use value iteration to determine the optimal policy. We found this algorithm to be sufficient since the number of states is fixed for each week, and the actions taken (the players chosen) also have different potential rewards from week to week, so an algorithm like QLearning would give us no added value, as the weights learned would not necessarily carry over from week to week.

Results

Overall, both our approaches generated fairly good lineups. They certainly aren't the best, but they have pretty good chances in the doubleup tournaments. The graph below gives a summary of results:



The CSP and MDP are fairly close, and follow the same pattern most of the time. The greedy algorithm does a lot better on some weeks, but seems to rely too much on luck - overall it doesn't do as well. All of the algorithms give very oscillating point values. Ideally we want something more stable, but since the oracle varies quite a bit as well, this variation doesn't come as a surprise.

CSP Experiments

We made two main cuts to make the CSP run for a reasonable time: the number of players to consider for each position, and adding a minimum salary for a lineup.

The following table shows the prediction data for week 12, varying the number of players considered for each position.

Players	predicted points	actual points	num operations	time
3	112.0907528	110.8	44	

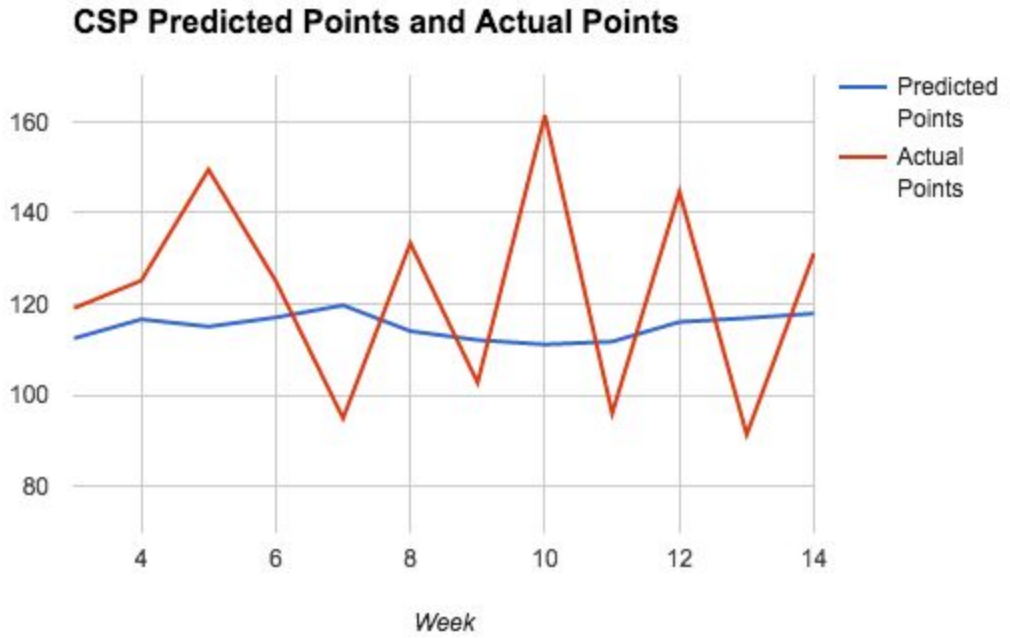
4	116.0599206	144.6	5000	
5	116.0599206	144.6	19104	10 sec
6	116.86175	115.44	301779	1.5 min
7	116.86175	115.44	1049791	5 min

As we increase the number of players to consider for each position, the predicted points increases. But the increase is most significant from three to four players, and the number of operations required in the backtracking search also increases significantly. Additionally, notice that the actual points earned by the lineup is larger for 4-5 points than for 6-7 points. This varies by week, but suggests that the extra gain of using 6-7 players instead of 4-5 players is insignificant. We ended up choosing to use 5 players for each position, as it runs acceptably fast and performs relatively well from week to week.

min salary	num operations
60000	4810
59500	13933
59000	19104
58000	22351
57000	22717

Restricting the minimum salary as well as the number of players to consider for each position group didn't have the effect we expected. For example, in week 12, when restricting the number of players to consider to 5, we get the table to the left. Each generates the same lineup (with salary 60000). The surprising result is how little the minimum salary affected the number of operations backtracking search ran. Minimum salary was a useful feature when we considered more players for each positions, but since we consider so few players for each position, it's no longer relevant.

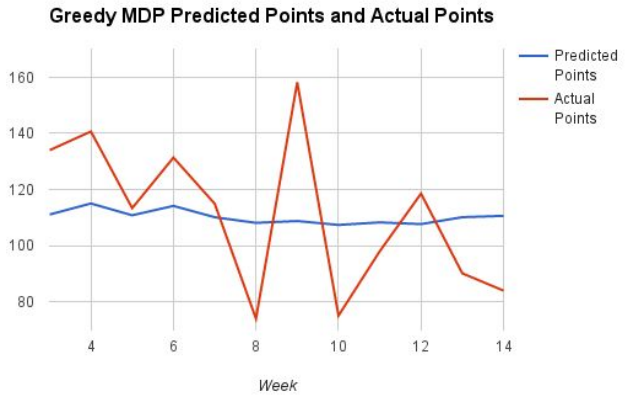
The resulting CSP predictions vs actual points are depicted below:

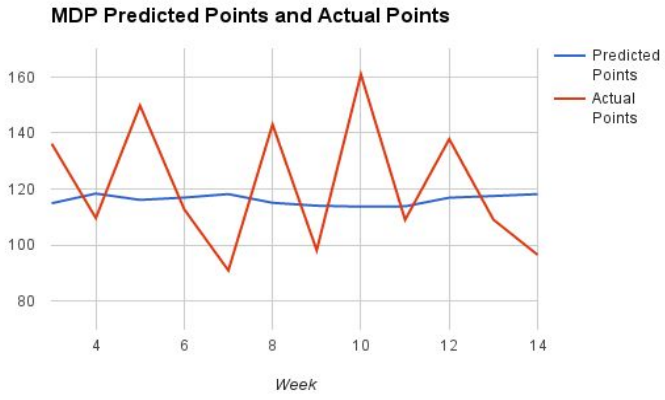


MDP Experiments

Our main work with the MDP involved redesigning the state representation to be able to model the fantasy lineup. Initially, we included the current value of an incomplete lineup in the state, as the sum of the expectation of each player, given by the bucket they'd been assigned. This gives `O((players in position group · probability buckets)^(players in lineup))` states, or about `(5 · 7)^9 ≈ 10^14` states, if we consider 5 players for each position group. With the states as they currently are, we instead have `O((players in position group)^(players in lineup) · (players in position group · probability buckets))` states. With the same numbers as before, this gives about `10^8` states. To further reduce the number of states, we varied the number of players considered for each position group, depending on how important the position seemed. In particular, we looked at only the top 5 QBs, 10 WRs, 10 RBs, 5 TEs, 5 PKs, and 5 Defenses. The reason we chose a larger number for WRs and RBs is not only because the MDP has to choose more than just one of these positions, but also because these positions yield the most value -- RBs and WRs tend to be the highest earners and as such we wanted to pick line-ups that properly sampled from these positions. Similarly, we note only 5 QBs tend to go over 20 points a week, and similarly, only 5 defenses tend to go over 8 points a week -- as such we wanted to pre-filter these selections.

We also ran the MDP without the probability buckets, as a greedy algorithm that picks the first valid lineup it finds. This worked surprisingly well:





The greedy MDP doesn't do quite as well as the MDP with 7 probability buckets, but the difference is smaller than expected. The similarities in results might be because the player predictions are already ordered and pre-filtered, so the first lineup the MDP sees is fairly likely to be a good one.

Literature Review

There has been a lot of work done on fantasy sports, including multiple projects done in this class.[1] However, most of the work available focuses on predicting fantasy points per player,

instead of generating the lineups. Perhaps generating the point predictions is more interesting, or perhaps people prefer to keep their lineups more private. Still, our models for generating lineups is only as good as the prediction they're fed, so it makes sense to take a look at the state of player predictions. There are multiple methods used; Bayesian inference is used to make predictions for the current season at bayesff.com, initially based on the previous season, but updated throughout. Other lineup generators include fantasyomatic, which uses regression algorithms,[2] and fantasy football analytics, who wrote their own open source R package.[3] We haven't found any lineup generators that are built on classification, however.

The regression algorithms used by fantasyomatic is based on a concept they call "Strength of Schedule." The concept can be summarized as the "belief that a player will do better than average against a defense that is weak against their position and worse than average against a defense that is strong against their position." In other words, to estimate how well a player will do, they focus a lot on the defense of the team they're playing against. We do include the defense of the opponent as a feature regression; however, we don't focus as much on the season history of the opponent. They have a lot of information about what features they focus on for the regression, but they don't seem to generate lineups from the individual player data.

We haven't found other lineup generators that have code released, or that describe their methodology. Instead, they seem to be largely based on expert opinion, using predicted points as tools.

Conclusion

The ultimate goal of the project is to generate lineups that are profitable in double-up tournaments. We haven't taken this into account; instead of generating the lineup that's most likely to make a profit, we generated the lineup with the highest expected points. Based on our results, we expect this lineup to make a net profit, but not to earn money every week. In the future, we would like to make our lineups produce profit more consistently, perhaps by attempting to take into account what lineups other people might generate, or by better modeling the relationship between players.

A potential way of better modeling this relationship is to generate covariance matrices to express the relationship between players, and use convex optimization to generate the lineups. We didn't have time to attempt to implement this, but it would be interesting to see if it could give us better and more stable results.

---

[1] http://cs229.stanford.edu/proj2015/104_report.pdf,
http://web.stanford.edu/class/cs221/restricted/projects/jblock93/final.pdf,
http://web.stanford.edu/class/cs221/restricted/projects/dbeam/final.pdf
http://web.stanford.edu/class/cs221/restricted/projects/hughec/final.pdf

[2] http://www.fantasyomatic.com/?page_id=4312

[3] http://fantasyfootballanalytics.net/2016/06/ffanalytics-r-package-fantasy-football-data-analysis.html