Predicting Fantasy Football Production Leveraging Domain Expertise

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 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 models to predict output for all fantasy football position groups - running back, quarter back, wide receiver, tight end, place kicker, and defenses - available on the FanDuel fantasy league betting site. With said models, 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 production a difficult task.

Dataset Collection

For this project, we have been building 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).
From FO we have been collecting their "revolutionary statistics that break down every single play of the NFL season"- namely Defense-adjusted Value Over Average (DVOA) statistics along with their Defense-adjusted Yards Above Replacement (DYAR) statistic. These statistics are drawn up by FO analysts after reviewing each play of each game using coaches' film and assigned grades to individual players and performances, then computed to normalize across teams so as to get a more informed sense of a team's performance relative to the entire league.

From RG we have been collecting historical FanDuel data - namely, fantasy points achieved, player cost on FanDuel, team a player was on, and their opponent.

From the NFL we have been collecting historical box score information - the number of touchdown passes a quarterback threw in a particular game, the number of yards a running back rushed for, etc.

We've collected this data and loaded it into our own custom data-structure (a series of nested Python dictionaries) in order to quickly generate training samples and prediction data with varying 'gameleads' - the number of game statistics to combine into a feature vector in order to run predictions - along with labels for both regression and classification tasks.
We have decided to use data from 2011-2015 as training and tuning data, and will be running final tests on the 2016 season data.

Approaches

Feature Selection

Currently, for both training and predictions, we have been using a gamelead of 3 games -- namely, for any player, we look at the past 3 games worth of statistics, coupled with statistics about their current team and their current opponent, in order to predict the week's fantasy production.

The features currently used for each game in the game lead are as follows:

Quarterback, Running Back, Wide Receiver, Tight End:

Team Offensive Line DVOA - Rank, Offensive Line Run Yards, Running Back Yards, Sacks Given Up, Sack Yards Given Up, Sack Rate, DVOA, DYAR
Team Offense DVOA - Rank, Yards Gained per Game, Rushing Yards, Rushing Rank, Passing Yards, Passing Rank, Strength of Schedule Rank, Team Win-Loss Percentage, DVOA, DYAR
Opponent Defensive Line DVOA - Rank, Run Yards Allowed, Running Back Yards Allowed, Sacks Made, Sack Yards Made, Sack Rate, DVOA, DYAR
Opponent Defense DVOA - Rank, Yards Given Up Per Game, Rushing Yards Given Up, Rush Defense Rank, Passing Yards Given Up, Passing Defense Rank, Strength of Schedule Rank
Individual Player Box Score Stats - Pass Attempts, Pass Completions, Completion Percentage, Interceptions, Sacks, Passing Touchdowns, Receiving Touchdowns, Rushing Touchdowns, Passing Yards, Rushing Yards, QB Rating, Home/Away Boolean
FanDuel Stats - Salary, Points Scored, Average Fantasy Points Per Game

Place Kicker:

Team Efficiency DVOA - Rank, Yards Gained per Game, Passing Offense Rank, Rushing Offense Rank, Team Total Yards Per Game, Team Win-Loss Percentage, Strength of Schedule Rank, Total DVOA, Total DYAR, Special Teams Rank, Special Teams DVOA, Special Teams DYAR
Opponent Efficiency DVOA - Rank, Yards Given per Game, Passing Defense Rank, Rushing Defense Rank, Team Total Yards Given Per Game, Team Win-Loss Percentage, Strength of Schedule Rank, Total DVOA, Total DYAR, Special Teams Rank, Special Teams DVOA, Special Teams DYAR
Individual Player Box Score Stats - Field Goals Attempted, Field Goals Made, Extra Points Attempted, Extra Points Made
FanDuel Stats - Salary, Points Scored, Average Fantasy Points Per Game

Defense:

Team Defense DVOA - Rank, Yards Given Up Per Game, Rushing Yards Given Up, Rush Defense Rank, Passing Yards Given Up, Passing Defense Rank, Strength of Schedule Rank
Opponent Offense DVOA - Rank, Yards Gained per Game, Rushing Yards, Rushing Rank, Passing Yards, Passing Rank, Strength of Schedule Rank, Team Win-Loss Percentage, DVOA, DYAR
FanDuel Stats - Salary, Points Scored, Average Fantasy Points Per Game

For the current week - the week being predicted - we include all DVOA and FanDuel statistics about the player's team and opponent, but do not include the box score statistics.

We have not yet performed feature-pruning with either automated methods or iterative self-implemented methods as we have been working on building a working system prior to optimizing. With that being said, we spent significant thought and design time picking out the listed features as we, intuitively, believe them to be good indicators.

Classification

For our classification model, we have bucketed the output of the model into five categories: 0-5, 5-10, 10-15, 15-20, 20+ points. We are currently using the Random Forest Classification Algorithm as implemented in the Python SKLearn Library. We use Random Forest as, in our experience and per our TA's recommendation, ensemble methods have great performance modeling non-linearity in features, and leveraging large feature spaces (currently each example has around 200-300 features) for accurate predictions. We have different model

parameters for each position, but have found that a large number (100-500) of shallow trees (depth 5-10) have performed best for us.

Below are our current evaluations for QB and RB models using cross-validation on our training data (from years 2011-2015). For WR our accuracy is 0.5369. For TE it is 0.7126. For PK it is 0.4267, and for Defenses it is 0.3367.

```
----------------------Training RB Model---------------------
----------------------Evaluation----------------------------
             precision    recall  f1-score   support

        0.0       0.72      0.93      0.81      1013
        1.0       0.29      0.28      0.29       299
        2.0       0.31      0.08      0.13       191
        3.0       0.18      0.05      0.08       104
        4.0       0.38      0.15      0.21       103

avg / total       0.55      0.62      0.56      1710

The accuracy score is 62.05%
Confusion matrix, without normalization
[[941  64   5   2   1]
 [188  84  14   7   6]
 [ 88  69  16   6  12]
 [ 44  40   9   5   6]
 [ 39  33   8   8  15]]
Normalized confusion matrix
[[ 0.93  0.06  0.    0.    0.  ]
 [ 0.63  0.28  0.05  0.02  0.02]
 [ 0.46  0.36  0.08  0.03  0.06]
 [ 0.42  0.38  0.09  0.05  0.06]
 [ 0.38  0.32  0.08  0.08  0.15]]
```

```
----------------------Training QB Model---------------------
----------------------Evaluation----------------------------
             precision    recall  f1-score   support

        0.0       0.52      0.50      0.51        88
        1.0       0.23      0.15      0.18        72
        2.0       0.28      0.19      0.23       120
        3.0       0.36      0.19      0.25       141
        4.0       0.35      0.67      0.46       141

avg / total       0.35      0.36      0.33       562

The accuracy score is 35.59%
Confusion matrix, without normalization
[[44 11 11  8 14]
 [14 11 18  7 22]
 [ 4  8 23 24 61]
 [ 6 12 16 27 80]
 [17  5 14 10 95]]
Normalized confusion matrix
[[ 0.5   0.12  0.12  0.09  0.16]
 [ 0.19  0.15  0.25  0.1   0.31]
 [ 0.03  0.07  0.19  0.2   0.51]
 [ 0.04  0.09  0.11  0.19  0.57]
 [ 0.12  0.04  0.1   0.07  0.67]]
```

Regression

Similarly to our classification model, we have been using the Random Forest Regression algorithm as implemented in the Python SKLearn library. Our output with regression is actual fantasy points, as opposed to buckets. For evaluating our regression results we have been using mean absolute error, mean squared error, and explained variance score (normalized R^2 coefficients). Our current results can be seen to the right.

```
----------------------Training RB Model---------------------
----------------------Evaluation----------------------------
mean_squared_error : 26.424743
explained_variance_score : 0.426388
mean_absolute_error : 3.739487
----------------------Training WR Model---------------------
----------------------Evaluation----------------------------
mean_squared_error : 35.715733
explained_variance_score : 0.288390
mean_absolute_error : 4.418305
----------------------Training TE Model---------------------
----------------------Evaluation----------------------------
mean_squared_error : 19.611079
explained_variance_score : 0.375631
mean_absolute_error : 2.965428
----------------------Training QB Model---------------------
----------------------Evaluation----------------------------
mean_squared_error : 45.140645
explained_variance_score : 0.405118
mean_absolute_error : 5.382778
----------------------Training PK Model---------------------
----------------------Evaluation----------------------------
mean_squared_error : 18.859041
explained_variance_score : 0.031127
mean_absolute_error : 3.479688
----------------------Training Def Model--------------------
----------------------Evaluation----------------------------
mean_squared_error : 34.598624
explained_variance_score : 0.136904
mean_absolute_error : 4.568676
```

Next Steps

Moving forward, we want to continue to tune both our regression and classification models by performing fine-grained feature selection with our current features. Particularly we intend to leverage both iterative methods, and pre-implemented methods in SKLearn such as SelectKBest with f_regression and f_classification (we cannot use Chi^2 on account of having negative input values due to DVOA).

Moreover, we want to continue testing different learning algorithms such as AdaBoost, Gradient Boosted Trees (both the SKLearn and XGDBoost implementations), and simpler methods such as Logistic Regression and Naive Bayes. The advantage of having taking the time to build a modular data structure to house our training data along with an object-oriented approach to our model means performing these experiments comes quite quickly.