Marek Bejda (dotcz12)
Lakshmi Prakash (lprakash)

Generating a winning roster for a double-up game in Fantasy Football

##Background

In fantasy football, a user is given a starting amount ($60000 in our case) to compile a winning roster. Each roster has 9 positions that need to be filled: a Quarter back, 2 Running backs, 3 Wide Receivers, 1 Kicker, and 1 defense. We can choose any football player of the corresponding category from the 32 NFL teams for each of the above listed positions. Each player has an associated cost reflecting his performance throughout the season. The games are played every week (on Thursday, Sunday and Monday) during the football season. During each match, the statistics (the plays) for the players are recorded and fantasy points are calculated accordingly. For instance, a rushing touchdown is worth 6 points and a safety is worth 2 points. The points for all the rosters (from various users) are calculated and arranged in descending order, and a certain number of rosters from the top win. Double-up games are the ones in which about 50% of the people from the top win.

##Final Product

The application combines team and player statistics with opinions of "sports experts" from various online sources such as FantasyPros.com, National Football League (NFL.com), and FanDuel.com to generate a roster. We use acceptance-rejection Monte-Carlo to generate a roster that has a high probability of winning a double-up game as described above.

##Process

We need three important pieces of information to be able to compile a good roster:

1) The prices for each player: This gives us a constraint equation that the sum of the costs of the 9 players we choose has to be less than $60000. Since we are trying to create the best roster possible, it is reasonable to assume that we would want to use most of the available money. So, we impose another constraint that the total cost of our choice of 9 players has to be greater than $59900. We obtain this cost information from

FanDuel.com in JSON format. The nested structure of JSON files added difficulties not encountered in CSV or other basic data format. We used C++ to parse these JSON files into separate classes that transformed our data into a more C friendly format (CSV format).

2) Play-by-Play information for the whole season: We obtain the data of all the individual plays that have happened over the season from NFL.com in JSON format by using a ruby script. This serves as the past statistics of how the players have performed this season and enables us to calculate the merit of a roster. Similar to the other JSON files, these JSON files are parsed in C++ and converted into CSV format with the relevant information which is in turn read in to C data structures.
3) Sport analytics rankings: We download the list of QBs, RBs, WRs, TEs, Ks, and DSTs in order of the ranks given to them by sport experts. Minimizing over the analyst rankings adds a humane touch and expertise into a computer simulation. It accounts for variable factors such as if the game is being played at home or away which the statistics cannot take into account. This was in CSV format and thus was directly read in C.

##Putting all the data together to generate a roster

Step 1:
We go through all the players in the rankings lists provided by the sports analytics experts and create all possible rosters that satisfy our constraint equations (total cost is between $59900 and $60000 both inclusive).

Step 2:
The plays from the CSV file are read into a separate array for each team. The mean and variance for the number of plays are calculated. They come out to be _ and _ respectively.

For each team, we generate a random number obeying the Gaussian distribution. This gives us the number of plays for the run, say n for the team t. We pick n plays from team t randomly (uniform distribution). The uniform distribution ensures that the plays that are more common have a higher probability of getting picked. We add up the points for all the players according to the plays we choose. After we have done the summing up for all the plays, we

have the points corresponding to every player. Next, we use these values to calculate the total for every roster. In the total for a roster comes out to be more than 120 (this is what we statistically found to be the cut off for double-up games), we accept the roster. We repeat the above procedure for 1000 times. The acceptance probability is calculates as the number of times a particular roster gets accepted divided by the total number of runs (1000). If this probability comes out to be greater than 75%, we mark the roster as fit for consideration.

Step 3:
For all the rosters that have been marked to be fit for consideration, we sum the analyst ranks of the 9 players. The roster that has the minimum sum is our final choice; minimal sums means that the analysts ranked the players high.

##Results

Our final selection for Dec. 7 2014!!

QB: Eli Manning Salary: 7200
RB: C.J. Anderson Salary: 7200
RB: Tre Mason Salary: 5600
WR: Dez Bryant Salary: 8900
WR: Josh Gordon Salary: 7800
WR: Odell Beckham Jr. Salary: 7500
TE: Jason Witten Salary: 5700
K: Cody Parkey Salary: 5000
DST: Houston Texans Salary: 5100
Total Salary: 60000

##Optimizations

We are dealing with a lot of data in our project. The number of rosters generated initially is huge. In order to improve performance and improve run time, we did a few optimizations. We took all constant calculations (the ones independent of the loop variable) outside of the loop. We avoided checking for unnecessary conditions (the ones whose truth value can be conclude without an explicit check). We removed all duplicate generation of rosters. Also, we calculate the sum of the rankings only for the rosters that have an acceptance ratio of more than 75% instead of calculating it for all rosters at the beginning. In addition, we used optimization level –

O4 while compiling; this gave us a much needed performance boost reducing roster generation from 1 hour to only 3 minutes.

##Next Steps

Our next step would be to parallelize the code using MPI and hopefully achieve performance benefits. The parallelization is trivial as calculation for any play is independent of the other. We can parallelize it such that different threads or nodes calculate the plays for different teams.

##Relevance

The process and algorithm can be extended to help decide investment and portfolio management under a certain budgetary constraint. The allocation of money is critical to the financial industry where hedge funds and investment banks need to allocate portions of their capital into stocks and bonds. There's more than 5000 publicly traded companies that can be chosen from and tough decisions need to be made whether to invest into various levels of risk.

##References

http://www.nfl.com
https://www.fanduel.com/p/Home
http://www.fantasypros.com