Predicting the Betting Line in NBA Games

Bryan Cheng
Computer Science
Stanford University

Kevin Dade
Electrical Engineering
Stanford University

Michael Lipman
Symbolic Systems
Stanford University

Cody Mills
Chemical Engineering
Stanford University

***Abstract*—This report seeks to expand upon existing models for predicting the outcome of NBA games. Using a time-varying approach, the model proposed in this report couples standard machine learning techniques with weighted causal data to predict the number of points scored by each team in an attempt to beat the spread.**

I. Introduction

In the NBA, thirty teams comprise two conferences. Throughout the regular season these teams will each play 82 games, for a total of 1230 NBA games played per season. To the enterprising sports gambler, this means 1230 opportunities to beat the odds made by expert NBA analysts and cash in. To the data analyst, 1230 games provide a wealth of player and team data for modeling complex trends in the performance of individuals and franchises. This report is concerned with exploring prior research in the field of sports prediction, specifically with the goal of predicting NBA game outcomes more accurately than the NBA experts who set the betting line for each game. Before diving into our models and predictions, we want to provide an overview of the various types of bets you can make in the NBA.

a) The Spread: When betting against the spread, the gambler bets either that the favorite will win by more than the spread, or the underdog will lose by less than the spread (or win). The favorite is the team with the negative number, because they are essentially handicapped a number of points. In order to place a bet of this form, the gambler would place a bet of $110 with a payoff of $100. The amount paid for a chance to win $100 is denoted in parentheses.
e.g.
Miami Heat -8.2 (-110)
Denver Nuggets 8.2 (-110)

b) Over/Under: In this form of betting, the gambler bets whether or not the total points scored in a game will be greater than or less than a given amount. As with the spread, the amount paid for a chance to win $100 is also listed.
e.g.
Miami Heat 170 (-110)
Denver Nuggets 170 (-110)

c) Other: There are additional forms of betting, but this report is concerned only with the betting schemes discussed above.

Prior Research

A prevalent approach in the field is to use a combination of models, as opposed to a single prediction algorithm, to output a more robust prediction. For example, the TeamRankings.com website, founded by Stanford graduates Mike Greenfield and Tom Federico and featured by ESPN, combines the predictions of six different models to shape their expected outcomes of college basketball games. The intuitive idea of a combination of models is that each model can capture a different aspect of the game or correct for a poor assumption made by another model. Due to time constraints, the methods covered in this report are all singular in their approach. It is possible that some combination of our individual predictors would yield better results, and is worth further exploration in the future.

Also, Zifan Shi et al. suggested normalizing team statistics with respect to the strength of the opposing team. Though they did not propose an algorithm for doing so, this seemed a logical approach and we incorporated it into our model. In this report we first apply basic feature reduction and polynomial regression with team averages over a long period. These approaches are useful for becoming familiar with the data, and picking out any obvious trends that our more involved models might use to their advantage. We then use the classic, "tried and true" support vector machine technique with various feature sets to predict game outcome. Finally, we propose a model for estimating a team's time-varying offensive and defensive strengths in order to normalize the team statistics with respect to their opponent in each game. These normalized statistics are used to predict the number of points scored by each team in the coming game, and we then make a bet against the spread under the assumption that our predicted point spread is more accurate than the actual spread.

II. Data

For our project, we needed actual NBA game statistics, but there are no publicly available datasets. Instead, we used public box scores on the popular sports website ESPN. On ESPN, the data we wanted are organized per game in box scores. We wrote a scraper that traversed all the dates of the regular season, found if there were any games for that date, and if there were, stored the box score HTML links. With all the box score links, we issued an HTML request to those HTML pages, received the HTML response, and parsed the corresponding data from those pages. We scraped the past 10 years of data from ESPN, making small modifications year to year for HTML formatting differences. We figured 10 years would be sufficient for our purposes. As for the betting data, probably to prevent people like us from running tests like these, we were unable to find betting data from past games listed on any site. However, via an ESPN Insider account, and by changing the URL in the browser with the correct game id, we managed to get the betting data from all of last year's games. Data from prior years were not available via this method, so we were limited to testing spread and over/under results on only last year's data. After collecting this data, we organized

the data into 3 tables in a MySQL database:

TABLE I. GENERAL PER GAME DATA

Date	Time	Location	Home Team
Away Team	Home Team Score	Away Team Score	Flagrant Fouls
Technical Fouls	Officials	Attendance	Game Length
Spread	Over/Under	Home ROI	Away ROI

TABLE II. TEAM PER GAME DATA

Team ID	Game ID	1st Quarter Pts
2nd Quarter Pts	3rd Quarter Pts	4th Quarter Pts
1st OT Pts	2nd OT 2 Pts	3rd OT Pts
4th OT Pts	Field Goals Made	Field Goals Attempted
Three Pointers Made	Three Pointers Attempted	Free Throws Made
Free Throws Attempted	Offensive Rebounds	Defensive Rebounds
Rebounds	Assists	Steals
Blocks	Turnovers	Personal Fouls
Points	Fast Break Points	Points in the Paint
Total Team Turnovers	Points Off Turnovers	

TABLE III. PLAYER PER GAME DATA

Player ID	Team ID	Game ID
Minutes Played	Field Goals Made	Field Goals Attempted
Three Pointers Made	Three Pointer Attempted	Free Throws Made
Free Throws Attempted	Offensive Rebounds	Defensive Rebounds
Rebounds	Assists	Steals
Blocks	Turnovers	Personal Fouls
Plus/Minus	Points	Did Not Play Reason

With MySQL, we easily manipulated or merged the data into smaller more applicable datasets that we wanted to perform our training on. For example, we could take the averages for each team in games played in the 2012-13 NBA season, instead of analyzing the entire dataset.

III. BASIC MODELS

To start off, we looked at some basic data among teams. To determine wins, the most obvious stat is points scored or in other words, which team scores more than the other team. We wanted to know if we could make any simplifying assumptions based on how teams scored points. For one season, we plotted the histogram of points scored per game. We see that this is a

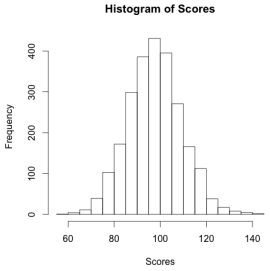

Fig. 1. Distribution of Points Scored Per Game for All Teams in NBA (2012-13)

nearly perfect Gaussian distribution. Further, by breaking this down per team per year, we can see similar results with smaller datasets.

For most of our tests, we make the assumption that the teams have fairly consistent performances (we do not take into consideration for example major injuries). Our first test was running a basic linear regression using four predictors: Home Team Points Per Game (PPG), Home Team Points Allowed Per Game (PAPG), Away Team PPG, Away Team PAPG. This model trained on season total averages for the full season, then testing on the training set predicted correctly 68%. Unfortunately, if we hold out data, the testing accuracy was only 54%.[1]

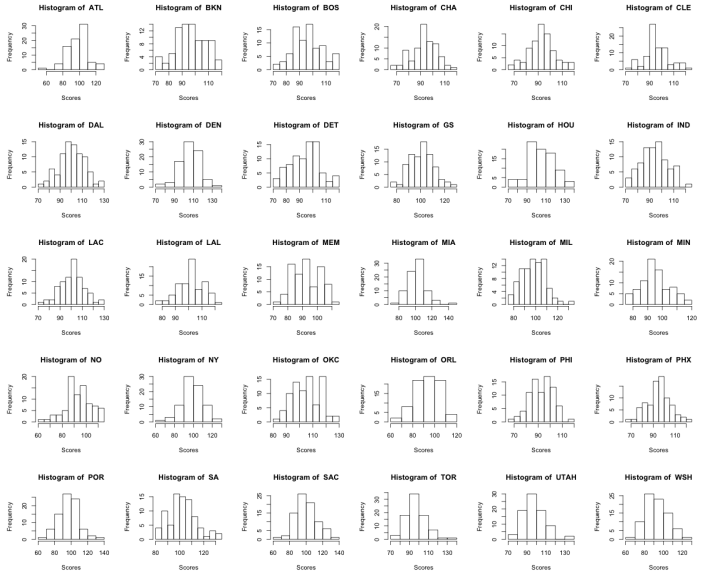

Fig. 2. Distribution of Points Scored Per Game for All Teams in NBA (2012-13)

For the next step, we incorporated more stats outside of simply points per game averages. We found the season long averages for every team stat listed in the teams stats table. Some stats exactly predicted others (2*(FGM-TPM) + 3*TPM + FTM = PTS and OREB + DREB = REB), so we had to reduce the number of predictors. We used a lasso approach to reduce the coefficients of certain predictors to 0 or effectively remove the predictors. Using cross validation, we trained on a random half of the data then tested on a held out dataset. With this, our model again predicted with 68% accuracy, but on data it had not seen before.

To play around with different predictors in the model, we tried using raw game numbers instead of the season averages to train the model, and then tested on the same data. This produced worse results at 63%. Lastly, we obviously do not have the season end averages until the end of the season. To compensate for this, we calculated the running averages (using a small python script since MySQL does not have this capability). One thing to note is that the running averages can have large changes over the course of the season. To see where they stabilized, we plotted each teams points per game average over 82 games.

From the graph, we can see some teams have very stabile and constant averages. Most of the teams seem to stabilize by 1/2 to 3/4 of the way through the season. Only a couple

[1]For the regression, the home team's PPG and away team's PAPG were more significant than the other 2 predictors. Put into the basketball context, this means an offensive team performs better at home, while a defensive team performs better on the road. Interestingly, this seems to indicate that a team is more consistent defensively on the road, while at home, their offensive production can feed of the crowd's energy.

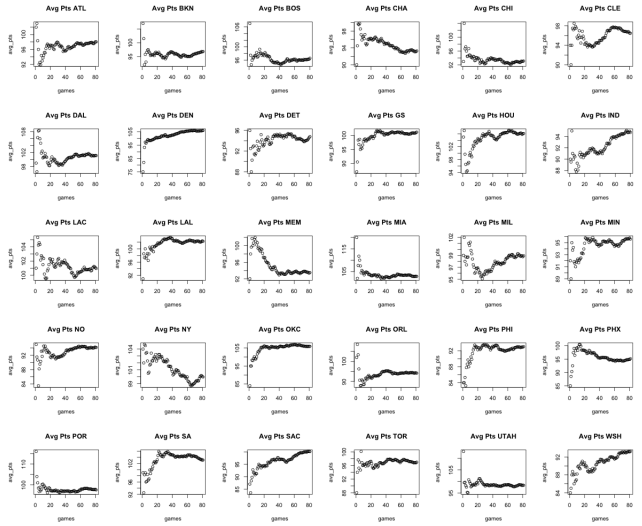

Fig. 3. Average Points Per Game Per Team Over 82 Games

kept changing until the end of the season.[2] When we trained only on the running averages in the 1st half of the season and tested on the running averages in the 2nd half of the season, we managed a get a test accuracy of 66%.

The next step we tried from this was using SVMs to classify the game as win or loss.

IV. Support Vector Machine

Using the full game data from the database, implemented a support vector machine model to classify each game sample as a binary variable 1 (win) or 0 (loss). We used the SVM in conjunction with a lasso regression and bootstrap aggregating to achieve our final prediction. Before implementing the algorithm, we pre-processed the raw data was had scaled and calculated the average of various team statistics from each season (rebounds per game, points per game, etc.). Further, for each game we created a sample point with the averages for both teams as features.

Even in this reduced form there were still 79 variables features. To infer which of the predictors were most significant we ran a lasso regression on the data. The lasso, a type of shrinkage regression, set several of our predictors coefficients to zero, which justified their exclusion from our model because they were insignificant. Our lasso model was tested over a range of shrinkage parameters and was then 10-fold cross validated. We chose the best model after cross validation to select our best subset of predictors. Though the data for our lasso model was of varied size and character throughout our experimental trials, the lasso generally left approximately 20 predictor coefficients nonzero.

Using the set of predictors that had been found to be most significant, we fit a polynomial kernel support vector classifier to the data. We implemented the SVM and tuned the result to find the optimal cost parameter and polynomial degree from a range using simple cross validation. This model was able to predict the win response for whole seasons of games with accuracy in the range of 65-69%.

To further optimize the prediction of the SVM we also implemented bootstrap aggregating (bagging) over the SVM-lasso model. We bootstrapped over our entire model so that the lasso would be fit onto each bootstrap re-sample and decide which sample were significant in that bootstrap set. Then we fit the SVM to the re-sample and predicted the results for the test set. Averaging the predictions over 20 bootstrap re-samples, we set the sample with an average of less than .5 to 0 and the rest to 1. In the best case we were able to bag over a model trained on the 2012-13 season and predict the results of the 2011-12 season with 68.4% accuracy and the results of the 2010-11 season with 65.1% accuracy.

We also explored fitting the model on a larger training set, which led too a small improvement in test accuracy. We fitted and bagged a model trained on the 2010-11 and 2011-12 seasons and predicted the 2012-13 results with 68.9% accuracy. This was our best non-training error from the lasso/SVM model. This is a strong result in context; ESPNs Accuscore algorithm for NBA odds (win/loss) bets had an accuracy of 70.3% last season. Though a consistent expert picks panel does not formally exist at ESPN for basketball, in NFL football the Accuscore football equivalent has been more accurate than any ESPN expert for the past 3 years.

To examine the real effect of our model we calculated the real-time average of a teams statistics for the 2012-13 season and ran our bagged lasso/SVM model on a training set that used the previous two seasons as well as an arbitrary number of elapsed games in the 2012-13 season. When we predicted the remain games of the season we consistently achieved accuracies of 63.5% (70% of games remaining) or greater with the accuracy increasing to 65.6% toward the end of the season (30% of games remaining). We also ran the predictor on the entire running averages set and got an accuracy of 65%. Though this test of the true use-case yielded weaker results than the full-season retrospective classifications, we believe that given more time we could improve the model. First, we could calculate running averages for each season and use these samples to train the data. We could also go a level deeper in detail and try to use a construction of player statistic contributions (in real season time) to construct the team average statistics and then make predictions.

A. Boosting

One of the methods cited by TeamRankings.com as an element of their prediction formula was a decision tree. Because of his success we also pursued a decision tree to try to classify our data. However, instead of simply fitting a decision tree regression to the data we implemented a Boosting method to combine a myriad of weak learner decision trees to form a strong learner tree at the end. We optimized by simple cross validation over three different tuning parameters: the depth of the tree, the number of trees, and the shrinkage parameter lambda. Unfortunately, the accuracy of the model plateaued at 64.5%. Boosting is a particularly slow and computationally heavy method so it was difficult to run a cross validations/optimizations over many combinations of depth, tree count, and shrinkage parameter.

[2]If you look at Indiana's steady increase in average points per game, it can be attributed to Paul George's meteoric growth as a player. Our models never account for player or team growth.

B. Spread and Over/Under Analysis

We ended up developing full models to predict the spread, but to give us an initial idea on how spread and over/under lines are set, we ran a simple regression using teams averages as predictors and the following graphs are the result.

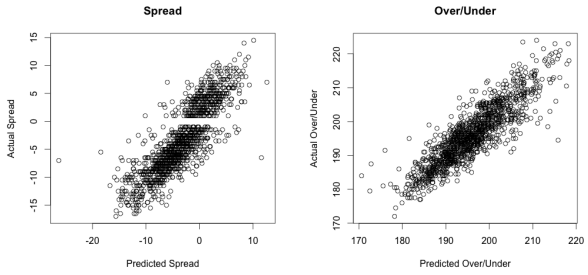

Fig. 4. Prediction of Spread and Over/Under using Basic Linear Regression

From the graphs, we can see a clear linear relationship between what we predicted and what the final betting lines were set at. The variance could come from betting houses adjusting to how the public is betting or variation in the models that our linear model did not account for (such as injured players, fatigue, etc). We then looked at how the Vegas spread and over/under lines compared to actual game results.

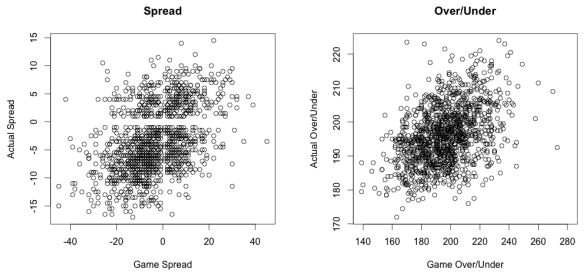

Fig. 5. Spread and Over/Under vs. Actual Game Results

From these graphs, it is clear that from the cloud shape that there is a lot of noise in the actual games compared to the predictions. This could be the underlying nature of sports that sports are inherently very noisy and there is nothing we can do. On the other hand, the betting lines do a good job splitting their predictions in half, essentially meaning that they do a good job themselves over the course of the season setting the lines such that they are guaranteed to win.

V. Causal Weighted Regression

In our final approach, we attempted to create a model that captures both the natural fluctuations in a team's performance throughout the season, and also adjust their statistics to more accurately reflect their performance in a game. For instance, a team that puts up a lot of points against the best team in the league should potentially have their rating increased, even if they lose. We introduce the causal limitation somewhat artificially to this method, although our reasoning stems from the fact that we have a time-varying model, and it would not make sense to incorporate future data. One thing to note is that this is a hybrid approach, and once we obtain the causally normalized statistics, we use polynomial regression along with SVM and Naive Bayes techniques to further enhance the model's predictions. The following steps describe the full method we devised. Running this algorithm over a season

Algorithm 1 Calculate Time-Varying Defensive and Offensive Strengths By Win Propagation

Require: λ, K_O, K_D
  $OS^{(0)} = 1$
  $DS^{(0)} = 1$
  for all $game \in allGames$ **do**
    $m \leftarrow numberOfGamesWinnerHasPlayed$
    $p \leftarrow numberOfGamesLoserHasPlayed$
    $OS_{winner}^{(m+1)} \leftarrow \lambda OS_{winner}^{(m)} + (1-\lambda)K_O \frac{DS_{loser}^{(p)}}{OS_{winner}^{(m)}}$
    $DS_{winner}^{(m+1)} \leftarrow \lambda DS_{winner}^{(m)} + (1-\lambda)K_D \frac{OS_{loser}^{(p)}}{DS_{winner}^{(m)}}$
    $OS_{loser}^{(p+1)} \leftarrow \lambda OS_{loser}^{(p)} + (1-\lambda)K_D^{-1} \frac{DS_{winner}^{(m)}}{OS_{loser}^{(p)}}$
    $DS_{loser}^{(p+1)} \leftarrow \lambda DS_{loser}^{(p)} + (1-\lambda)K_O^{-1} \frac{OS_{winner}^{(m)}}{DS_{loser}^{(p)}}$
  end for

will produce an estimated offensive strength ($OS^(m)$) and a defensive strength($DS^(m)$) where $m \in Mgamesperseason$ is the number of games played for each team. The constants K_O and K_D control the relative weighting of defense vs. offense, and λ is the forgetting factor - i.e. how much past game results should affect the current strengths. These values are then smoothed with a polynomial interpolator to get an estimate of a team's offensive and defensive strengths relative to the other teams at any point in the season. One important thing to note is that we do withhold the test set from the training set when we apply the smoothing, as failure to withhold this data would cause the test data to leak into to the training data. The estimations of this method provide a fairly decent relative representation of all the teams' capabilities, at least when compared retrospectively to their performances last year. The defensive strengths of a team are then used in

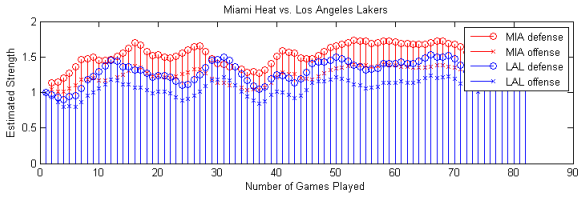

Fig. 6. In the 2012 season, the Miami Heat were regarded as the best defensive team in the league. The Los Angeles Lakers were more or less in the middle of the league both offensively and defensively.

every game to normalize their stats to get an estimation of their effective stats relative to the league at that point. These values are similarly smoothed to reduce the high variance that is inherent in sports data.

These normalized statistics are then used in a linear regression over the training set to map a given statistic to normalized points scored in a game. We found that a derived statistic called "effective field goal percentage" (eFGP) was the most highly correlated with points scored, and so our model uses only the regression fitted to this statistic in predicting the normalized number of points that will be scored. Once

a normalized point prediction is made based on a teams smoothed normalized eFGP going into a game, the prediction is de-normalized with the opposing teams estimated defensive strength ($DS_{opponent}^{(numGamesOpponentHasPlayed)}$) to get a real score prediction. We can then compare the predicted scores for each team to get a predicted spread, and by comparing this to the given spread for the game, we are able to bet one way or another.

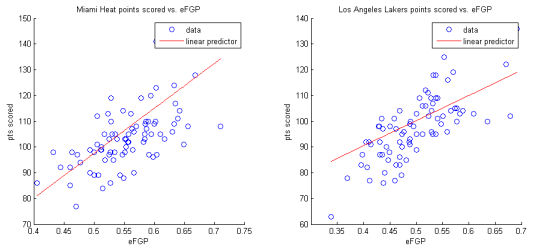

Fig. 7. This regression is fitted for each team in the hopes of capturing any differing offensive paradigms between teams.

To further improve our model's ability to beat the spread, we incorporate both SVM and Naive Bayes classifiers trained on a feature set of the game statistics in addition to our predictions. The classifiers are used both to directly predict whether or not a team will beat the spread in a given game, and also to predict whether or not the prediction given by the model will succeed or not. The first case is referred to as Method 1, and the second as Method 2. Here are the results of the model under various test schemes: K-fold Cross Validation with $K = 10$, and Random Holdout Cross Validation with 30% of the data withheld as the test set. The RHCV values are obtained as the mean of 30 iterations. Though the above figures

TABLE IV. SPREAD PREDICTION MODEL RESULTS

Method	*10-fold CV i*	*10-fold CV ii*	30× *RHCV i*	30× *RHCV ii*
Normal Predictor	52.53%	51.67%	51.39%	50.71%
SVM Method 1	51.69%	50.84%	49.88%	50.23%
SVM Method 2	52.78%	50.84%	50.33%	49.59%
NB Method 1	51.42%	50.52%	50.07%	49.74%
NB Method 2	51.10%	51.22%	50.42%	50.33%

seem to imply that this would be a reasonable approach, our results show that it was not any more successful at predicting game outcomes than the brute-force bulk approaches discussed earlier (win/loss prediction results not shown in table as they have been discussed at length in prior sections). However, since this model was designed specifically with the goal of predicting the point spread, it is not surprising that it would perform worse with regard to winner prediction. Beating the spread is not necessarily the same problem as predicting the winning team. We also tested this model under more stringent conditions imposed due its causal nature, so this probably accounts for the lower performance in win/loss prediction accuracy. With our results, we are reluctant to believe that our model achieves significantly better than 50% prediction accuracy vs. the spread, and although multiple iterations of the two cross-validation schemes did show a slight favorable edge towards 51% and 52%, we are certainly not beating the requisite 52.4% prediction accuracy required to enact a financially rewarding betting strategy.

This modeling approach makes more rigid assumptions about the time-varying nature of team performance, but also allows for better estimation of a team's true value at any given time. We think that expanding upon the existing causal weighting algorithm to include more variables would significantly improve the strength estimation aspect of the model. It was disappointing that incorporating the extra classification step did not significantly improve the model, but given that our predictions are hovering at around 50% anyway, it is not surprising that the predictions have only a very small correlation with the outcomes (if any), and no amount of classification can fix that.

VI. CONCLUSION

In this report we have shown that machine learning techniques can be successfully applied to NBA games to predict the winner of any given game with around 68% accuracy. This level of accuracy rivals that of professional analysts and basketball experts. However, in our endeavor to predict the spread outcomes with an accuracy greater than 52.4%, the model we developed fails to meet this goal under testing.[3] There are several contributing factors to this.

Once our model was trained, it resulted in a simple deterministic predictor. Ideally, since we are trying to model the interactions of complex entities, we would like to add more layers to our model and incorporate an element of stochasticity. This in conjunction with batch simulation would probably converge on a more accurate estimate for game outcomes than our train-once, predict-once method. Successful sports betting companies such as Accuscore claim to use this batch simulation approach. Implementation of a more complex stochastic model would require greater access to specialized data, and also more computational power to run many simulations. These are limitations that an individual wishing to beat the spread will always face, and it is no surprise that the predictors with more resources at their disposal are able to perform better. We conclude that in order to make predictions about such a complex interaction, the quality and complexity of the data used is perhaps the most important factor in determining the success of the model.

ACKNOWLEDGMENT

The members of this project team would like to thank Andrew Ng and his teaching staff for sharing their wealth of knowledge on the related subject material. We should also like to thank Yuki Yanase for helping manually collect betting data from the ESPN website.

REFERENCES

[1] Adams M, Ryan P., George E. Dahl, and Iain Murray. "Incorporating Side Information in Probabilistic Matrix Factorization with Gaussian Processes." 25 Mar, 2010. arxiv.org.

[2] "Four Factors." Basketball Reference. n.d. basketball-reference.com.

[3] Hess, David. "Under the TeamRankings Hood, Part 4: Models, Models, Everywhere." TeamRankings. 12 Mar, 2011. teamrankings.com.

[4] Shi, Zifan et al. "Predicting NCAAB match outcomes using ML techniques - some results and lessons learned." 14 Oct, 2013. arxiv.org.

[5] Walsh, PJ. "How to bet over/unders." ESPN Insider. 4 Dec, 2013. insider.espn.go.com.

[3] For the poster presentation, we made a prediction for that night that Orlando would beat the spread in their game against Charlotte. We just wanted to say, that our prediction was right: ORL 92 CHA 83 - http://espn.go.com/nba/recap?gameId=400489191