Simar Mangat
Keyur Mehta
Maneesh Apte

CS221 Project Progress Report

Project Overview

As a brief project overview, our project focuses on using machine learning to predict NBA games, both from the perspective of win-loss and additionally the point spread. We plan to build on existing systems by incorporating both macro and micro game features, the latter being less explored by prediction models we saw during our research.

Model

The model for NBA game prediction is contingent upon the representation of teams' given information during the season and how that information is understood. We think of this information as particular to each season, and for this progress report implementation used the data from 2014-2015. The reason for this is that we believe there is too much variability between seasons with trades, injuries, etc, and hypothesized that we have sufficient data to predict within a season anyways. Since our intention is to use machine learning, **we chose to represent an NBA game via a variety of quantifiable features representing a multidimensional point space**. In summary, the feature vector for each game consists of the desired statistics about the considered team and corresponding statistics for the opposing team. For precise features, see table below:

Considered Team Feature	Opponent Team Feature	Feature Full Name
3P%	OPP-3P%	3 Point Field Goal Percentage
3PA	OPP-3PA	3 Point Field Goals Attempted
3PM	OPP-3PM	3 Point Field Goals Made
AST	OPP-AST	Assists
BLK	OPP-BLK	Blocks
DREB	OPP-DREB	Defensive Rebounds
FG%	OPP-FG%	Field Goal Percentage
FGA	OPP-FGA	Field Goals Attempted
FGM	OPP-FGM	Field Goals Made
FT%	OPP-FT%	Free Throw Percentage
FTA	OPP-FTA	Free Throws Attempted
FTM	OPP-FTM	Free Throws Made
MIN	OPP-MIN	Minutes
OREB	OPP-OREB	Offensive Rebounds
PF	OPP-PF	Personal Fouls
PTS	OPP-PTS	Points

REB	OPP-REB	Rebounds
REST		Days rested
STL	OPP-STL	Steals
TOV	OPP-TOV	Turnovers
W%		Win %

There are a few interesting aspects to note, particularly the handling of features for the current team being considered versus its opponent and how that relates to the overall model. **To model the NBA game, we consider the perspective of every NBA team and its corresponding opponents within the particular season.** Then, for the day they play a game against an opponent team, we transform the concept the NBA teams involved into the features in the table above, with features like field goal attempts, rebounds, win percentage, etc, designated for the both the team being considered and the same features regarding the team's opponent on that day. A label is associated with that game, either a 'W' or 'L'.

Note that this means that while there 1230 total games possible in a season, our model actually considers double that, because we take each team against all others and don't factor out repeats. We felt this modeling was advantageous because within the context of machine learning the "double counting" leads to different feature sets for each team, thus giving a richer view on prediction.

To make this concrete with an example, consider the New Orleans Pelican game versus the Spurs which took place on April 15, 2015. We have a vector of different values that represent the feature values. In that vector are also the corresponding statistics for the Spurs. The game is labeled as a 'W'. This vector and label are then used for training in the algorithm.

It is important to note that we are still in the process of adding more features to our model, particularly regarding the micro indicators. Some of these could be player or position specific, e.g. how many points each point guard has scored, how many rebound the center has received, number of drives made by particular players, etc.

Another important aspect of the modeling step was the partitioning of the training data and test data. Since we only were considering data specific to a season, we had to decide on an appropriate way to partition the data such that we could maximize our prediction accuracy but somewhat minimize the amount of training data needed. We decided to model this by setting a threshold of games, with the games leading up to the

threshold being considered as training data and the games for the rest of the season being the test data.

Algorithm

Based on the feature representation of matched up teams, we decided to use multiple different algorithms to learn the weights of those features and then predict based on those weights. We tried three different algorithms: SVMs, Naive Bayes, and Random Forest.

Support vector machines are supervised learning models used to produce classifications and regressions. They are regarded as some of the best machine learning algorithms for prediction. SVMs envision the data as a point space where multidimensional planes are drawn to separate the data into categorizations. We believe that SVMs will be a powerful algorithm to harness the depth of our feature set while avoiding overfitting because of the algorithm ability to take features with the highest predictability power.

Naive bayes is a probabilistic supervised algorithm based on Bayes theorem. It is said to be naive because it makes the assumption that all pairs of features are conditionally independent when conditioned on the possible outcome (So in our case ‘W’ or ‘L’). Naive bayes is a powerful prediction algorithm that proves to be very fast. We believe naive bayes has the potential to do well on our model, however, since the pairs of features of the NBA game are in some cases inextricably linked (e.g. field goal attempts and field goal percentage). This could potentially mean the algorithm could be assuming too simplified of a situation.

Random forest is a supervised algorithm that can be used for classification and for regression, with common application in biocomputation. It operates by constructing many decision trees during training time, and then during test time outputs the class that is the mode of the classes (which is classification) or mean prediction (which is regression) of the individual trees. Random forest is a useful algorithm because it is robust to inclusion of potentially irrelevant features, and the randomization of the underlying decision trees allows the algorithm to reduce overfitting. We know from prior experience that the random forest algorithm can be successful in certain circumstances, and thus we decided to incorporate it as an algorithm.

Considering our previous example with the Pelican versus Spurs game, each of these algorithms would take the feature data for the all the Pelican games in season 2014-15 and train on the the games up to a certain threshold. In SVMs case, the training would

consist of associating a weights vector with the features and then minimizing an objective function. Prediction would then follow from these weights. In naive bayes, probabilities are computed are stored during the training and then during prediction the probabilities given a ‘W’ and given a ‘L’ are compared and the larger is returned as the classification. Finally, with random forest, the training would consist of partitioning the data into decision trees, with a certain element of randomization.

Initial Results

We had promising results from the multiple algorithms we ran. Results are plotted in graphs below. We tried to fine tune the best value for the game threshold, running the algorithms by trying all possible threshold values and plotting the results against the prediction accuracy. Our results are plotted in the accuracy versus threshold graph, which shows naive bayes as the clear winner in terms of prediction as the threshold increases. However, we believe naive bayes was able to achieve in part by overfitting the data. We will continue to investigate it though. **We also had promising results from SVM algorithm, which gave a peak prediction of nearly 65% accuracy** without needing to ingest much of the season as training data (about 20 games per team). Random forest did not perform as well as we hoped. We also tried to get rid of the threshold value completely and moved towards randomly sampling a certain number of games as the training data, and then using the rest of the data for testing. The results for each algorithm are shown in the graph of accuracy versus number of random games sampled. Setting a threshold was clearly the better approach, with the algorithms in the random sampling case being quite brittle and fluctuating.

Out results using the threshold compare favorably to our original baseline, which was 53%, and entailed choosing home teams as winners. Our oracle was the Accuscore algorithm for NBA (win/loss) bets, and for the 2014-2015 season, their model had an accuracy of 70.3%. Considering that we have not incorporated all micro features into our model yet, we believe there are exciting possibilities for prediction in the next stages of our project.