# Prediction of winner in NBA basketball games

## Abstract

I apply three machine learning approaches: Support Vector Machine(SVM), Linear Regression, MultipleLayer Perceptron to predict the winner of a National Basketball Association(NBA) game. The data is selected from the historical statistics of regular season games and the features are defined and computed from it. The accuracy of all algorithms is beyond 64% by the naive majority voting mechanism. And the MLP does the best job.

## 1. Introduction

The National Basketball Association is the premier men's professional basketball league in the world.The winner team within the NBA will gain popularity and get more income, and is beneficial to the league and its players. So predicting the winner in a game based on the previous performance is very valuable to business managers, coaches, players, fans, gamblers, and statisticians alike. I apply approaches to improve this prediction.
So far some people used the statistics of the teams like the average score, the shot efficiency and so on, while some people focused on the past history of the team like the winning percentage of the team, the overall behavior in the past ten games and the points differential. I'm trying to combining the features together to take a deeper look at both the behaviors and the historical data of the team. I've gathered the regular seasons game box scores from 2010 to 2016 and tried to compute the feature vectors. After that I'll start to get the teams' behaviors and apply PCA(Principal Component Analysis) to analyze the most important features. At last, I'll use three approaches for machine learning process and compare the result with that generated by the naive majority voting mechanism based on the winning percentage of a team.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

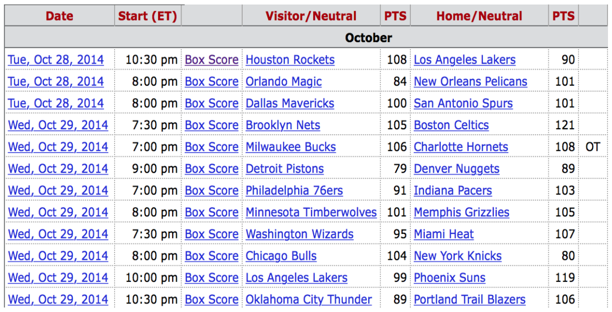
## 2. Model

I'm trying to combine the detailed behaviors statistics with the comprehensive situation of teams together and then use three approaches for training and prediction. The first big problem is where to gather the data. I searched the Internet and read the paper and finally decided to fetch data from http://www.basketball-reference.com. I can also gather the detailed behaviors of teams in each game, but it'll take more time. The accuracy of this learning system mainly depends on whether your features are well defined or not. Intuitively, we will consider winning percentage of both teams. Also point differential per game is a good feature to measure the strength of a team. In addition to these, we should also look at recent behaviors of both teams so we define the win percentage and point differential per game for the latest N games. Last but not least, To define the features, there're ten features to take into consideration:

- Winning percentage of team A
- Winning percentage of team B
- Point differential per game of team A
- Point differential per game of team B
- Winning percentage of team A in latest N games
- Winning percentage of team B in latest N games
- Point differential per game of team A in latest N games
- Point differential per game of team B in latest N games
- As a visiting team winning percentage of team A
- As a home team winning percentage of team B

So the aim of the following job is to collect data and generate the feature vectors for the overall behaviors of both teams in a match. And then I'll implement three machine learning processes with ten features to show that the accuracy is better. From the website, I can collect the box scores of regular season game from 2011-2012 to 2015-2016. What I did to extract the data was to copy the box scores into a spreadsheet in $Excel$, and then I wrote a $macro$ to replace all the team names with numbers ranging from 1 to 30. After that, I converted .xlsx file to .txt

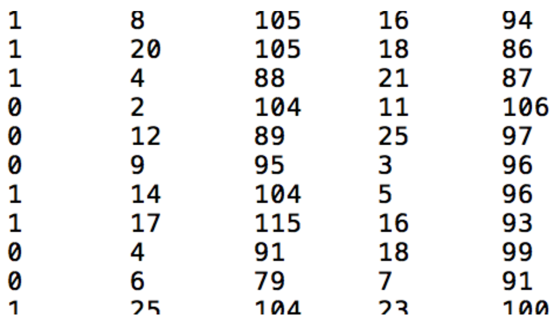
with space splitting each instance. Note that some teams changed their name in 2013 or 2014, so I had to find it out and replace it manually. Since the data was generated in the order of ascending time. So I deleted the date and detailed time of the games and added an column for labeling whether team A was the winner or not. If so, the label is 1. Otherwise it's 0.
The original data was like the illustration shows:



| Date | Start (ET) | | Visitor/Neutral | PTS | Home/Neutral | PTS | | |
|---|---|---|---|---|---|---|---|---|
| | | | October | | | | | |
| Tue, Oct 28, 2014 | 10:30 pm | Box Score | Houston Rockets | 108 | Los Angeles Lakers | 90 | | |
| Tue, Oct 28, 2014 | 8:00 pm | Box Score | Orlando Magic | 84 | New Orleans Pelicans | 101 | | |
| Tue, Oct 28, 2014 | 8:00 pm | Box Score | Dallas Mavericks | 100 | San Antonio Spurs | 101 | | |
| Wed, Oct 29, 2014 | 7:30 pm | Box Score | Brooklyn Nets | 105 | Boston Celtics | 121 | | |
| Wed, Oct 29, 2014 | 7:00 pm | Box Score | Milwaukee Bucks | 106 | Charlotte Hornets | 108 | OT | |
| Wed, Oct 29, 2014 | 9:00 pm | Box Score | Detroit Pistons | 79 | Denver Nuggets | 89 | | |
| Wed, Oct 29, 2014 | 7:00 pm | Box Score | Philadelphia 76ers | 91 | Indiana Pacers | 103 | | |
| Wed, Oct 29, 2014 | 8:00 pm | Box Score | Minnesota Timberwolves | 101 | Memphis Grizzlies | 105 | | |
| Wed, Oct 29, 2014 | 7:30 pm | Box Score | Washington Wizards | 95 | Miami Heat | 107 | | |
| Wed, Oct 29, 2014 | 8:00 pm | Box Score | Chicago Bulls | 104 | New York Knicks | 80 | | |
| Wed, Oct 29, 2014 | 10:00 pm | Box Score | Los Angeles Lakers | 99 | Phoenix Suns | 119 | | |
| Wed, Oct 29, 2014 | 10:30 pm | Box Score | Oklahoma City Thunder | 89 | Portland Trail Blazers | 106 | | |

*Figure 1.* original data

and then I converted it to the .txt file by macro to map the name of each team to an integer. And note that some teams changed their name so I have to manually do that. The generated data was as follows:



```
1    8     105    16    94
1    20    105    18    86
1    4     88     21    87
0    2     104    11    106
0    12    89     25    97
0    9     95     3     96
1    14    104    5     96
1    17    115    16    93
0    4     91     18    99
0    6     79     7     91
1    25    104    23    100
```

*Figure 2.* converted data

After extracting the box scores, I did some data analysis for features.

I first implemented the naive majority classifier based on the winning percentage of two teams in a game.

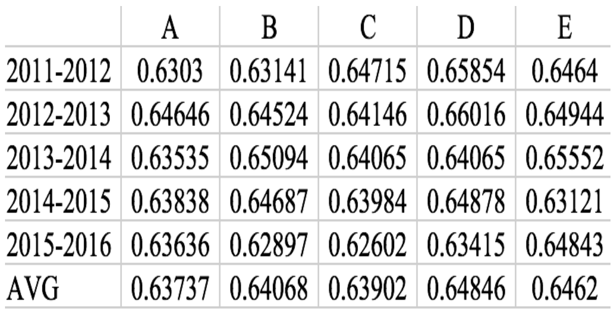
| Params | Meaning |
|---|---|
| pr | Probability of winning |
| win-count | Times of winning |
| count | Total times of games of teams |
| predict-correct | Times of correct prediction |
| total-instance | Number of instances |

Each time I read one instance and predict the result based on winning probability of the visitor team and the home team. And I check whether the prediction is correct or not, if it's correct, I'll increase predict-correct and win-count of the visitor team by one, otherwise increase win-count of the home team by one and compute the pr for both teams. After all instances processed, I can get the ratio of predict-correct over total-instance to get the accuracy.

| season | accuracy |
|---|---|
| 2011-2012 | 0.6363636363636364 |
| 2012-2013 | 0.6403580146460537 |
| 2013-2014 | 0.6455284552845528 |
| 2014-2015 | 0.647479674796748 |
| 2015-2016 | 0.6304559270516718 |

This mechanism determines the winner by looking at whose current winning percentage is greater in this season. Of course at the beginning they may be both 0 as no matches between them, and in this case I'll random a real number between 0 to 1 and I predict team A is the winner if this number is less than 0.5. I tested it on dataset of 2013-2014, 2014-2015 and 2015-2016 and the accuracy is around 64%. To deal with the data, I'm still using Java because I don't have too much experience with Matlab, which takes me more time. This is the first progress when comparing with other people's idea. After gathering behavior's data for both teams in each game, I'll use PCA(Principal Component Analysis) to increase the convergence and reduce the dimension. When all feature vectors are ready, I'll go ahead to train the datasets with Linear Regression, Maximum Likelihood Classifier and SVM. Our goal is that the implemented algorithm should give better prediction result than the Naive majority classifier with around 64%.

For each group of features listed at the beginning of the Model section, I used them to make online predictions for the games and computed the accuracy based on the current value of features:



| | A | B | C | D | E |
|---|---|---|---|---|---|
| 2011-2012 | 0.6303 | 0.63141 | 0.64715 | 0.65854 | 0.6464 |
| 2012-2013 | 0.64646 | 0.64524 | 0.64146 | 0.66016 | 0.64944 |
| 2013-2014 | 0.63535 | 0.65094 | 0.64065 | 0.64065 | 0.65552 |
| 2014-2015 | 0.63838 | 0.64687 | 0.63984 | 0.64878 | 0.63121 |
| 2015-2016 | 0.63636 | 0.62897 | 0.62602 | 0.63415 | 0.64843 |
| AVG | 0.63737 | 0.64068 | 0.63902 | 0.64846 | 0.6462 |

*Figure 3.* The accuracy of prediction over all features

We can see that the accuracy of prediction for 2015-2016 seasonal games is comparatively below the average. And that's because we only have 900 more games in this season while 1200 more in the others.

Also here I am taking $N$ equals to 10 because I did the following analysis:

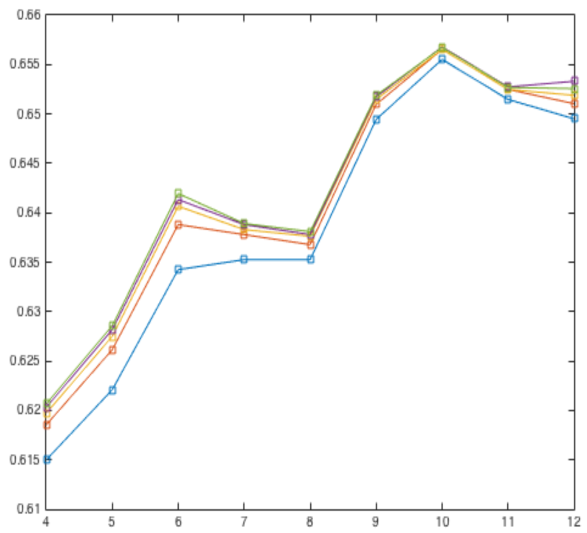


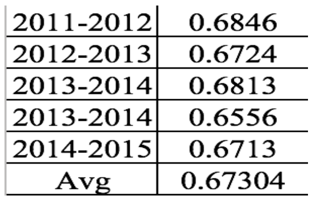
*Figure 4.* The accuracy of prediction by win percentage with different N

As we can see in the figure, the accuracy climbs up quickly with the increasing number of recent games we are looking at. And it reaches the top when $N$ is equal to 10. After that, it starts to decrease. So here I decided to look at the recent 10 games for the win percentage. The same idea can be used for looking at the recent 11 games for the points differential after analysis.

## 3. Algorithm and Evaluation

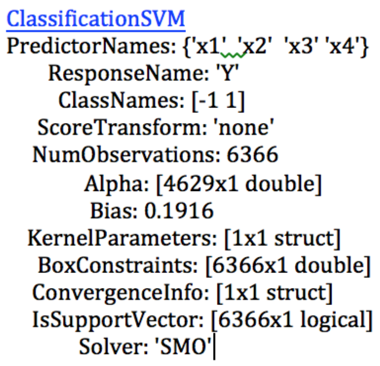
### Support Vector Machine

To train and predict the result, what I'm using at first is SVM(Support Vector Machine), which constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space. Since my data was stored into $Matlab$, and there're some existing calls about SVM, I decided to try SVM at first.
I started with the function call $fitcsvm$ with linear kernel function to train a svm model from the training data. After getting the model, I test it on the following year seasonal games, and found the accuracy is pretty good:
The running result of generating the prediction model is like:
We can see that the SVM with linear kernel works pretty well for this model, so we can say that it's more likely that



| | |
|---|---|
| 2011-2012 | 0.6846 |
| 2012-2013 | 0.6724 |
| 2013-2014 | 0.6813 |
| 2013-2014 | 0.6556 |
| 2014-2015 | 0.6713 |
| Avg | 0.67304 |

*Figure 5.* SVM accuracy



```
ClassificationSVM
PredictorNames: {'x1' 'x2'  'x3' 'x4'}
      ResponseName: 'Y'
        ClassNames: [-1 1]
    ScoreTransform: 'none'
   NumObservations: 6366
             Alpha: [4629x1 double]
              Bias: 0.1916
  KernelParameters: [1x1 struct]
    BoxConstraints: [6366x1 double]
   ConvergenceInfo: [1x1 struct]
   IsSupportVector: [6366x1 logical]
            Solver: 'SMO'
```

*Figure 6.* SVM Model

it's linear separable. If the accuracy is pretty low, then I'll assume that it may be not linear separable or there're lots of noise. And in that case, I may move on and try Gaussian Kernel or Polynomial kernel. Since it's more likely to be linear separable, I'll use linear regression in next section.

### Linear Regression

The second method implemented is the linear regression. The method consists of multiplication of each feature by a weight, making a summation of all values obtained and a bias, and uses this final value for the classification:

$$Y = \omega_0 + \sum_{i=1}^{n} \omega_i * x_i \tag{1}$$

I computed $Y$, and if $Y$ is greater than 0, the visitor team is considered the winner and otherwise the home team is considered the winner.

The important part here is to determine the value of vector $w$. I tried to train on all feature vectors with stochastic gradient descent to obtain the convergence for $w$. However, I tried several settings of step size and iteration numbers, yet the experiment showed that even after $3*10^5$ iterations some of variables still didn't converge.
The figure below shows one of the variables:

So to boost the convergence, the method I used was to apply PCA(Principle Component Analysis). With PCA,

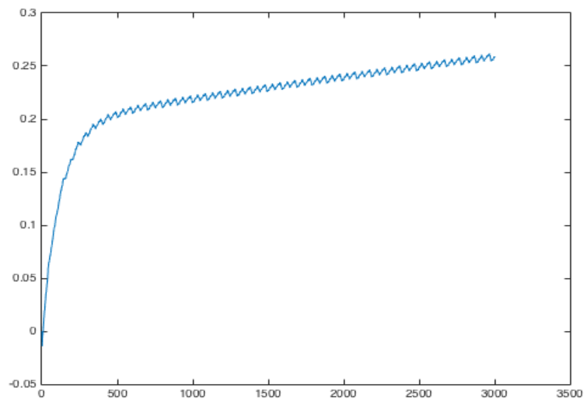
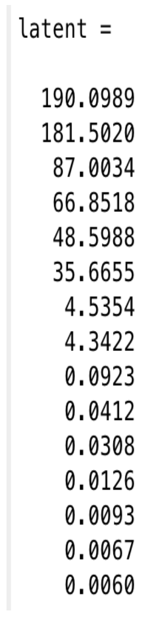


*Figure 7.* Not converged

we can decrease the number of features and thus the variables will be more likely to converge more quickly. By loading feature vectors into $Matlab$, we can use predefined functions to compute the eigenvalues of covariance matrix:



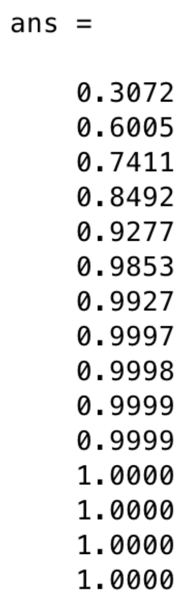
```
latent =

  190.0989
  181.5020
   87.0034
   66.8518
   48.5988
   35.6655
    4.5354
    4.3422
    0.0923
    0.0412
    0.0308
    0.0126
    0.0093
    0.0067
    0.0060
```

*Figure 8.* Eigenvalues of covariance matrix

And then the contribution percentage can be derived by the command $cumsum(latent)./sum(latent)$.
Based on the contribution, we can set a threshold to 80%. According to the figure below, we can say that the first four features can be used :

And the PCA feature vectors can be computed with the reduced features projected into the new space. Note that



```
ans =

    0.3072
    0.6005
    0.7411
    0.8492
    0.9277
    0.9853
    0.9927
    0.9997
    0.9998
    0.9999
    0.9999
    1.0000
    1.0000
    1.0000
    1.0000
```

*Figure 9.* Eigenvalues of covariance matrix

I preprocessed the value of features with -9999 as there's no data for each team at the beginning. So I removed the unpredicted instances which will make the training and learning process become more accurate and convincing. And the weight vectors can converge as shown:

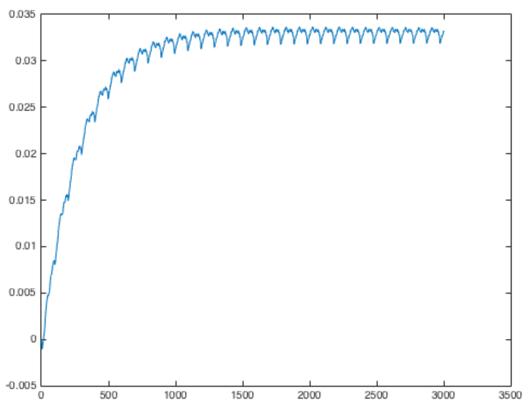


*Figure 10.* Convergence of variables

Weight vectors can be derived: -0.0762 0.1008 0.0771 0.0027 0.0563 And the accuracy of prediction is shown as follows:

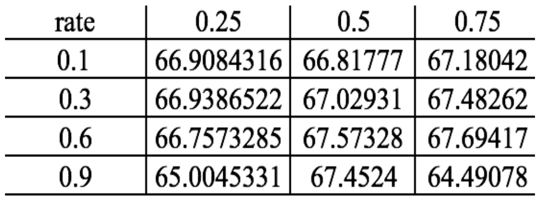
| season | accuracy |
|---|---|
| 2011-2012 | 0.6789 |
| 2012-2013 | 0.6942 |
| 2013-2014 | 0.6391 |
| 2014-2015 | 0.6718 |
| 2015-2016 | 0.6522 |
| Avg | 0.66924 |

From the result, we can see that the average accuracy of linear regression is approaching 67%, which is higher than naive majority voting classifiers.

**Multilayer Perceptron**

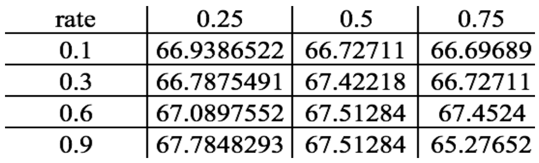
A multilayer perceptron (MLP) is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs. One characteristic of this algorithm is backpropagation. Here I used MLP for the prediction of the result of the game. The basic implementation is from Yu Hen Hu, and the mechanism involves randomly picking samples to train, feed-forwarding phase to compute sum of square errors, doing the error back-propagation phase to compute error and updating the weight matrix using gradient and some other parameters.

Each experiment will be executed for three loops and get the mean of the rate. The $\alpha$ vector is defined as [0.1 0.3 0.6 0.9] and $momentum$ is defined as [0.25 0.5 0.75]. With several configurations of different number of hidden layers, different number of neurons, the learning rate $\alpha$, and the value of momentum, the prediction rate can be obtained as follows:



| rate | 0.25 | 0.5 | 0.75 |
|---|---|---|---|
| 0.1 | 66.9084316 | 66.81777 | 67.18042 |
| 0.3 | 66.9386522 | 67.02931 | 67.48262 |
| 0.6 | 66.7573285 | 67.57328 | 67.69417 |
| 0.9 | 65.0045331 | 67.4524 | 64.49078 |

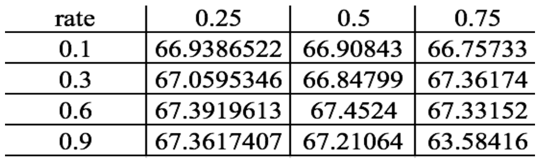
*Figure 11.* One hidden layer and four neurons



| rate | 0.25 | 0.5 | 0.75 |
|---|---|---|---|
| 0.1 | 66.9386522 | 66.72711 | 66.69689 |
| 0.3 | 66.7875491 | 67.42218 | 66.72711 |
| 0.6 | 67.0897552 | 67.51284 | 67.4524 |
| 0.9 | 67.7848293 | 67.51284 | 65.27652 |

*Figure 12.* One hidden layer and eight neurons

We can observe that adding more neurons do increase the performance to some degree. Note that too few hidden units will generally leave high training and generalization



| rate | 0.25 | 0.5 | 0.75 |
|---|---|---|---|
| 0.1 | 66.9386522 | 66.90843 | 66.75733 |
| 0.3 | 67.0595346 | 66.84799 | 67.36174 |
| 0.6 | 67.3919613 | 67.4524 | 67.33152 |
| 0.9 | 67.3617407 | 67.21064 | 63.58416 |

*Figure 13.* One hidden layer and twelve neurons

errors due to under-fitting. Too many hidden units will result in low training errors, but will make the training unnecessarily slow, and will often result in poor generalization. After training the data with one configuration, I test it on the test data and evaluate it. Each time I recorded the best accuracy and found one good model with the configuration of two hidden layers with eight and twelve neurons in each layer respectively. The accuracy on the models is computed:

| season | accuracy |
|---|---|
| 2011-2012 | 0.6917 |
| 2012-2013 | 0.6872 |
| 2013-2014 | 0.6915 |
| 2014-2015 | 0.6899 |
| 2015-2016 | 0.6877 |
| Avg | 0.6896 |

As we can see, multilayer perceptron algorithm works the best of the algorithms implemented with almost 69%.

## 4. Related work

There's a project from `http://www.mbeckler.org/coursework/2008-2009/10701_report.pdf` by Matthew and Michael. What they did was to use SQL to store all players' statistics to obtain the statistics of each team. And then they used algorithms for training and prediction with the assistance of K-means and Approximate Value. They did good job in prediction outcome of games with regards to the team's statistics. And their work can achieve about 73% accuracy, which is really good because of solid data preparation. But my work is based on only the box scores, and I also takes the recent win percentage and point differentials into consideration to measure the recent performance of teams and I explore several algorithms for the best classification method.

## 5. Conclusion, Limitations, and Future Work

I collected box scores for each NBA seasonal game and converted to the dataset we need. And I performed data analysis for the features I defined, also the accuracy of naive majority voting was computed as our initial goal. After that I implemented several algorithms, PCA was used to boost convergence in linear regression. For MLP, I tested many different configurations on the parameters. And the result is that MLP is the best algorithm with around 69% accuracy and both SVM and linear regression yield about 67% accuracy. All of the implemented algorithm achieved the goal of 64% by the naive majority voting mechanism.

And the limitation of my work is that I don't import the statistics of players or teams. So in the future, I should follow Matthew and Michael's work to build SQL and grasp all player's performance to compute the statistics for all teams. With the comprehensive performance by analyzing the box scores and the the generated detailed teams' statistics, the prediction can be expected to be more accurate.