NBA Shot Logs Analysis

Multivariate Analysis

Sergio Llana Pérez and Pau Madrero Pardo

1-. Introduction

Information is a very valuable asset and it often contains hidden knowledge which is not always exploited. After this idea, a lot of companies emerged between the 80's and the 90's applying a process called KDD (Knowledge Discovery in Databases) on a wide variety of fields such as business and health.

Sports is a field of obvious application due to the huge amount of data that is generated in every game. The first two real cases of application were developed by IBM and A.C. Milan. The former was a system created in 1996 in order to detect statistical patterns and odd events in NBA games. On the other hand, the latter was a decision support system developed to predict football player injuries in 2002.

1.1-. Data Understanding

In order to perform a practical analysis, we will use a public dataset called "NBA shot logs" obtained from Kaggle. It contains information about over 125 thousand NBA shots, which occurred during 2014-2015 season. The dataset has been obtained by scrapping NBAs official API and it is composed by the following 21 features:

- GAME ID: numeric identifier of the game.
- MATCHUP: variable containing the date of the match and the involved teams.
- LOCATION: whether the game is played home (H) or away (A) regarding the shooter.
- W: whether the game of the shot was eventually won by the shooters team.
- FINAL MARGIN: difference in the score of both teams at the end of the game (from shooters point of view).
- SHOT NUMBER: helps to keep the order of the shots in the same game.
- PERIOD: period of the game when the shot occurred. Note that periods 5 to 7 are extra-time periods because of a draw.
- GAME CLOCK: remaining time in the period when the shot was done.
- SHOT CLOCK: remaining possession time when the shot was done (out of 24 seconds).
- DRIBBLES: count of dribbles that the shooter did before executing the shot.
- TOUCH TIME: count of seconds that the shooter held the ball before shooting.
- SHOT DIST: distance (in feet) from shooters position regarding to the basket.
- PTS TYPE: whether the shot was a 2-pointer or a 3-pointer.
- SHOT RESULT: whether the shot was made or missed.
- CLOSEST DEFENDER: closest defenders name.
- CLOSEST DEDENDER PLAYER ID: closest defender's numeric identifier.

- CLOSE DEF DIST: distance (in feet) from shooters position to the closest defender.
- FGM: whether the show was successful or not (stands for ”Field Goals Made”).
- PTS: amount of points added to shooter teams score after the shot.
- player name: shooters name.
- player id: shooters numeric identifier.

2-. Data Pre-Processing

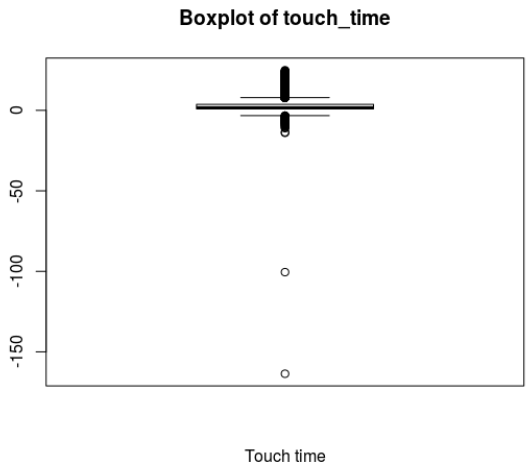
Once the raw data is read from the CSV file, we have started by formatting the “game_clock” to an understandable format for machine learning methods, we have implemented a function that turns a “MM:SS” format into the number of remaining seconds when the shot was done. In addition, the distances were transformed from feet to meters.

Finally, we have adjusted the type of some columns such as logical variables (into numeric) and ordered factors. The last transformation is to create a new data frame which filters out those unnecessary columns (e.g. “FGM” or “PTS”) and renames the existing features to have homogeneous column names.

2.1-. Data Cleansing

In this second phase, we have explored the data frame in order to look for possible wrong values, missing values and outliers.

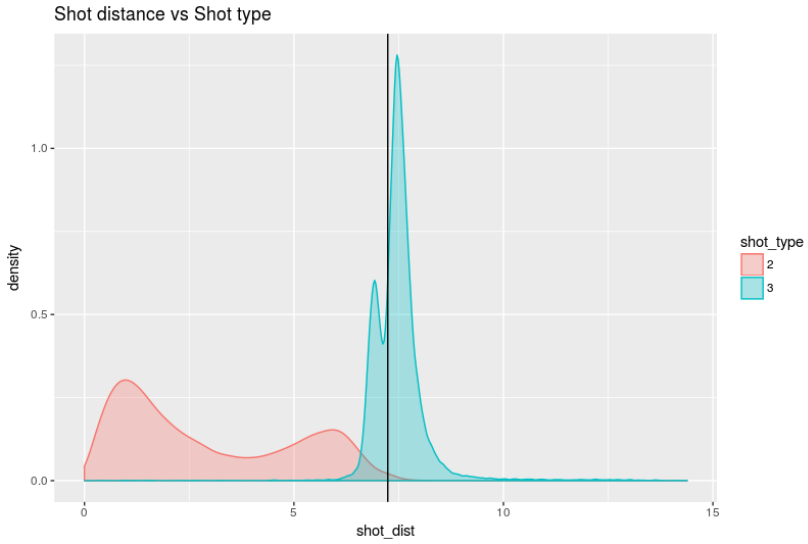
Regarding the “touch_time”, we know that their values should be between 0 and 24, as a possession in basketball cannot exceed that value. However, we can see in the following boxplot that there are values outside the range.



For handling them, there are two different cases:

- As the values that exceed the 24 seconds limit are pretty close to it (e.g. 24.3 seconds), we have decided to round them to 24.
- On the other hand, those negative values have been marked as NAs, which will be handled later.

Then, we have taken a look to the interaction between the shooter's distance and the type of shot (whether it is marked as a three or a two pointer). The following plot, created using the package "ggplot2", shows the density of both 2 and 3-pointer shots in different colours. We can appreciate how they are not separated, but we do not know the exact meaning of this distance in the data source, so we cannot make a decision.

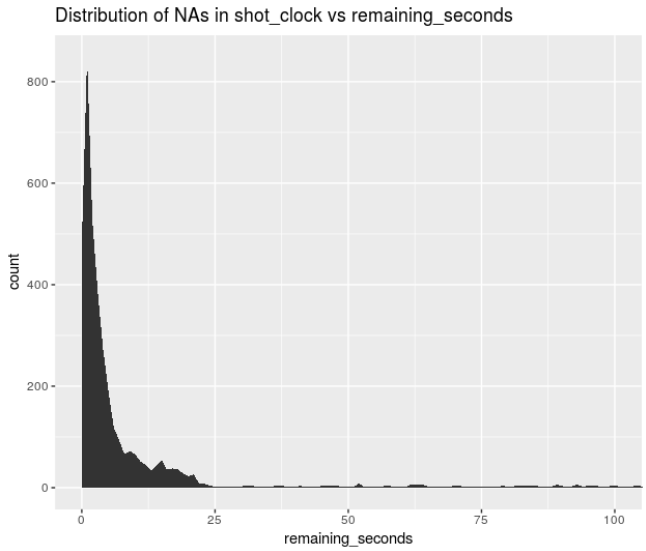


Concerning the "shot_clock" column, we have detected a spike in the histogram for values close to 24:

After doing some research online about the dataset, a basketball fan would realize these are all big men playing close to the hoop. Domain knowledge can lead to the conclusion that these are tip-ins or put-backs where a player near the hoop collects an offensive rebound after a teammate missed a shot.

```
                shooter count
                 <fctr> <int>
 1    andre drummond   177
 2    deandre jordan   131
 3    nikola vucevic    88
 4       rudy gobert    80
 5    tyson chandler    79
 6 jonas valanciunas    74
 7       enes kanter    73
 8  tristan thompson    69
 9       greg monroe    66
10       joakim noah    66
# ... with 262 more rows
```

Regarding the 5567 NAs in "shot_clock", most of them occur when "remaining_seconds" are lower than "shot_clock" and therefore the "shot_clock" is turned off, because it has no importance anymore. The following plot shows their density when compared to the "remaining_seconds".



2.2-. Feature Extraction

Finally, in order to enrich the dataset, we will create three new variables based on the existing ones. They could be very useful for future data visualization analysis or as supplementary variables in a Principal Components Analysis.

Firstly, we will start by creating a categorical feature based on the defender's distance to the shooter. It will contain four levels:

defender_dist <= 2 ft.	→	Tightly Contested
2 ft. < defender_dist <= 3.5 ft.	→	Contested
3.5 ft. < defender_dist <= 6 ft.	→	Open
defender_dist > 6 ft.	→	Wide Open

Secondly, based on the type of shots that NBA defines in https://stats.nba.com, we will create another categorical feature which depends on the number of "dribbles" and the distance of the shot.

No dribbles AND shot_dist > 4 ft.	→	Catch & Shoot
No dribbles AND shot_dist <= 4ft.	→	Cut
dribbles AND shot_dist < 4 ft.	→	Drive
dribbles > 4	→	ISO / Post up
dribbles > 20	→	Long ISO
dribbles <= 1 AND shot_type = 3	→	Spot Up Three
…	→	Other

And finally, we will create a logical variable for clutch situations. We say that a shot was done in a clutch situation when they are executed during the last minute of the potential last period (i.e. 4th period or extra time) for those games that ended up with a margin of points lower than 5 points.

2.3-. Data Enrichment

We have found a data source provided by ESPN from the same season that includes both the salary and position of every player. We thought that it would be a good idea to use the salary as a measure of the player's quality, so it could be a key feature for predicting the outcome of future shots.

As this dataset was only available in http://www.espn.com/nba/salaries/_/year/2015/, we had to use the package "rvest" in order to scrap the information from the website.

Then, when we tried to merge player's names between both datasets we realised that names didn't match correctly because of typing mistakes, abbreviations etc. We solve this challenge in two steps:

- First, we used "dplyr" to do a full join to figure out which players were not correctly mapped to the salaries dataset.
- Then, for those without a salary assigned, we used the "stringdist" package to calculate the most similar name in the other dataset using the OSA algorithm. In addition, we set a threshold in order to avoid matching wrong names.

All players were correctly fixed, except for two, which were changed manually.

2.4-. Transforming categorical variables into continuous

Our next step after the pre-processing phase is applying PCA to our dataset. Since this technique only takes into account continuous variable, some of the key features in our dataset would be ignored ("shooter", "defender", "shot_cat", "shot_difficulty" and "position").

Then, we have created a continuous variable for each of them, based on the percentage of successful shots per modality (conditioned to whether the shot is a 2-pointer or a 3-pointer). Note that for “defender”, we have not used the percentage of success, but the percentage of missed shots by the shooter.

2.5-. Handling Missing Values

shot_clock column

From the 5567 NAs in this column, we have proved before that most of them (3153) occur when the column “remaining_seconds” is lower than 24 seconds, which is the duration of a possession. The value of these observations will be imputed by a random number from a uniform distribution in the range of [0, remaining_seconds].

The remaining 1647 observations, which are a less than a 1.5% of the total observations, will be removed.

touch_time column

As we do not know the reason of the 285 values that this column contains, we will use the function “catdes” contained in “FactoMineR” with the purpose of revealing if certain values of the rest of columns are related with these missing values.

First of all, we need to create a boolean variable based on whether the “touch_time” is negative or not and then, we apply the function:

- Regarding the quantitative variables, we see in the results that shots with missing “touch_time” have a significantly lower and strangely exact ratio of success (0.3) compared to the overall mean (0.45), but most importantly, the mean of the number of dribbles before the shot is astonishingly lower than the overall mean (0.007 vs 2.05).
- On the other hand, for categorical variables, we see that the amount of shots of the categories "Cut" and "Catch&Shoot" is much higher than the average, as well as the positions "PF" and "C" and the difficulty "Tightly Contested". However, the shot categories "Drive", "ISO/Post up" and "Other" appear fewer times than average as well as the position "PG" and the difficulty "Open"

After these insights, one possible interpretation could be that shots with missing values in “touch_time” correspond mostly to quick plays where probably this time wasn't measured correctly due to the celerity (swiftness) of the game.

A naïve approach would be to impute the NAs by low random variables following the previous conclusions. However, we have decided to impute them using KNN (with K = 1) and supervise that the new values are generally lower than the average.

The mean is lower in the imputed values, so our hypothesis holds:

Mean of the imputed values: 2.1253
Mean of the original values: 2.8114

2.6-. Final dataset

Once all the previous transformations were applied, we ended up with a dataset of more than 113839 rows and the following 27 features:

"home_team"	"away_team"	"is_home"	"victory"	"final_margin"
"shot_number"	"period"	"remaining_seconds"	"shot_clock"	"shot_dist"
"shooter"	"touch_time"	"defender"	"defender_dist"	"dribbles"
"success"	"shot_difficulty"	"shot_cat"	"clutch"	"position"
"salary"	"shooter_pct"	"defender_pct"	"shot_cat_pct"	"shot_difficulty_pct"
"position_pct"	"triple"			

3-. Validation protocol

We will use the holdout method to validate our final prediction models. To do the partition between the training and test set, we have contemplated different alternatives: random partition, temporal partition or partition by players.

Random partition

It consists on assigning randomly the observations to either the training set or the test set. The R's caret library allows for the partition to maintain the original balancing of the different values of the response variable. This keeps a random misbalance of these values from affecting our predictions.

Temporal partition

It consists on partitioning the data keeping the most recent observation as the test set. Obviously, this kind of partitioning can only be used with data with a feature referencing time. It tends to work really well, since it emulates predicting data from the future, given information from de past. In our dataset, however, it is probably not a good idea to use it, because a NBA season consists on the regular season and the playoffs phase. If we used it, our training set would be shots from the regular season and our test set shot from the playoffs. Since only the best teams and players play on the playoffs we would not have a good representation of observations to test a model trained with teams and players from the whole league.

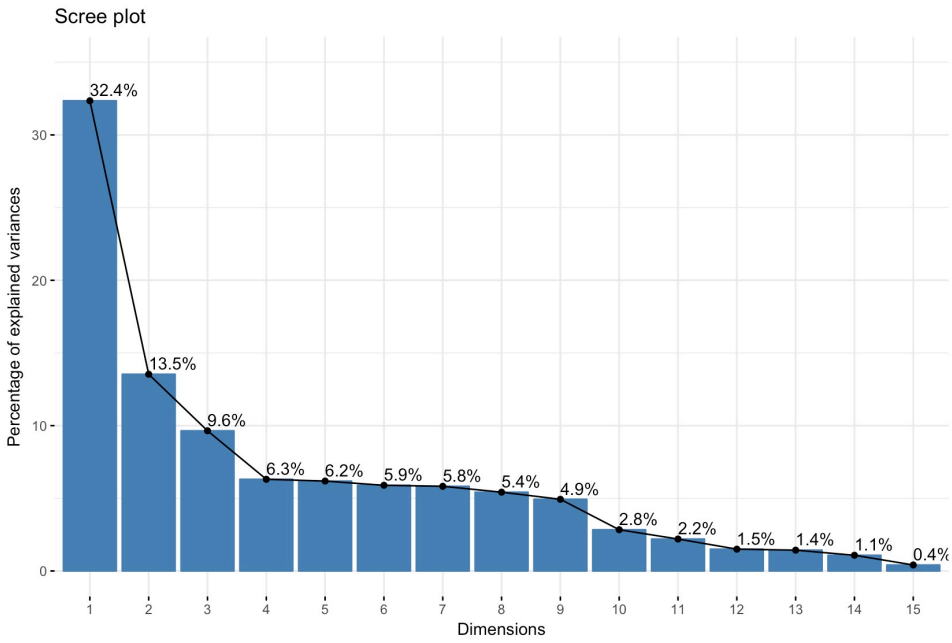
Partition by players

It consists on selecting a subset of players, and use all their shots as the test set. This method is considered very honest, since the model will not have been trained with any shots of the shooters that will appear in the test set. It could be really useful if you wanted to use the final model to predict shots from other competitions, where the players are not the same.

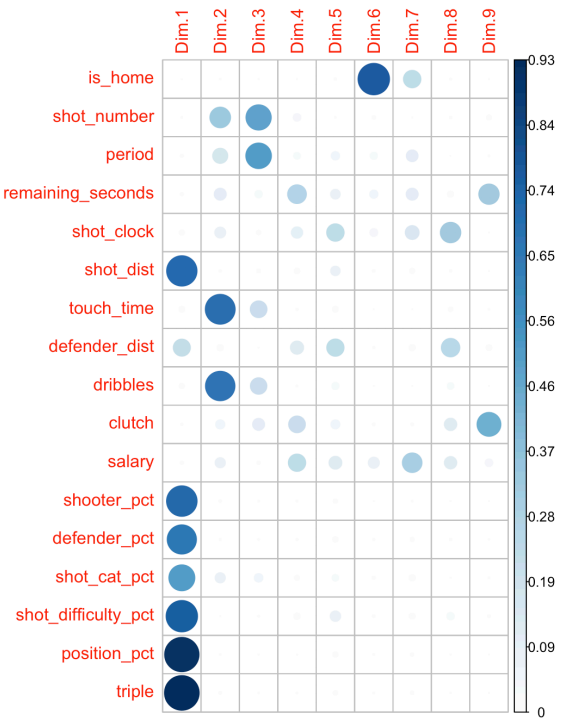
We know that the NBA is quite different from any other basketball competition, as it has the best teams and players in the world, but also some rules of the game change (e.g. 3-point line distance, basket height…). So, using a model trained with shot data only from the NBA would probably not work well predicting shots from other competitions, and not only due to the fact that the shooters are not the same as the ones used for training. We think the simplest and most adequate way of partitioning our dataset is to do it randomly. The proportions of observations will be 70% for the training set and 30% for the test set.

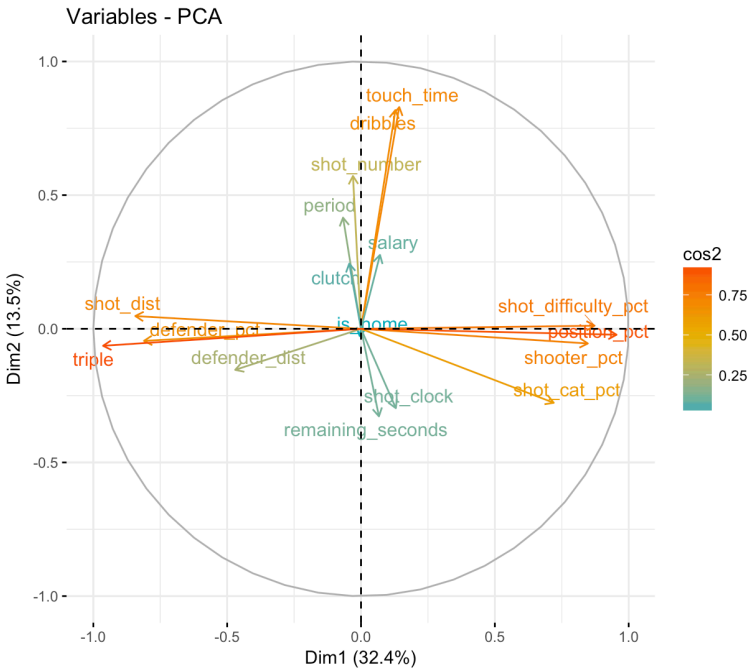
4-. Principal Components Analysis

Once our data is pre-processed and partitioned, we will proceed to perform a Principal Component Analysis. We will use all the continuous variables as active, and all the categorical ones –response variable included– as supplementary. The active individuals will be those contained in the training set, whereas the supplementary will be those in the test set. We will standardise the data before performing the PCA.



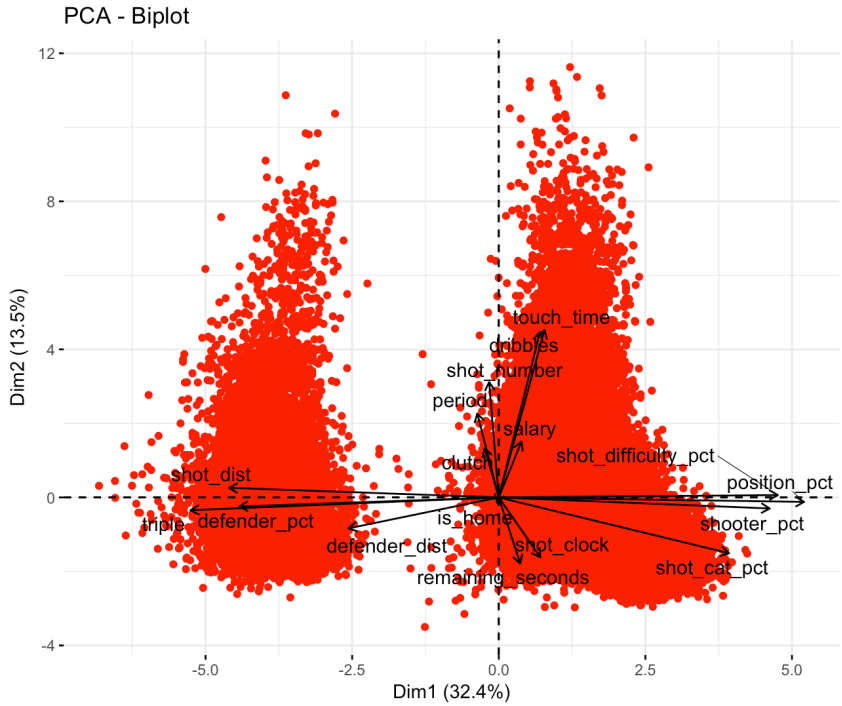
We will use the Last Elbow Method to select the number of significant components on the scree plot above. In this case, we will choose the first 9 components. The following step is to interpret the first principal components. To do so we will plot the correlations between the features and the components, as well as the variable factor map of the first factorial plane.





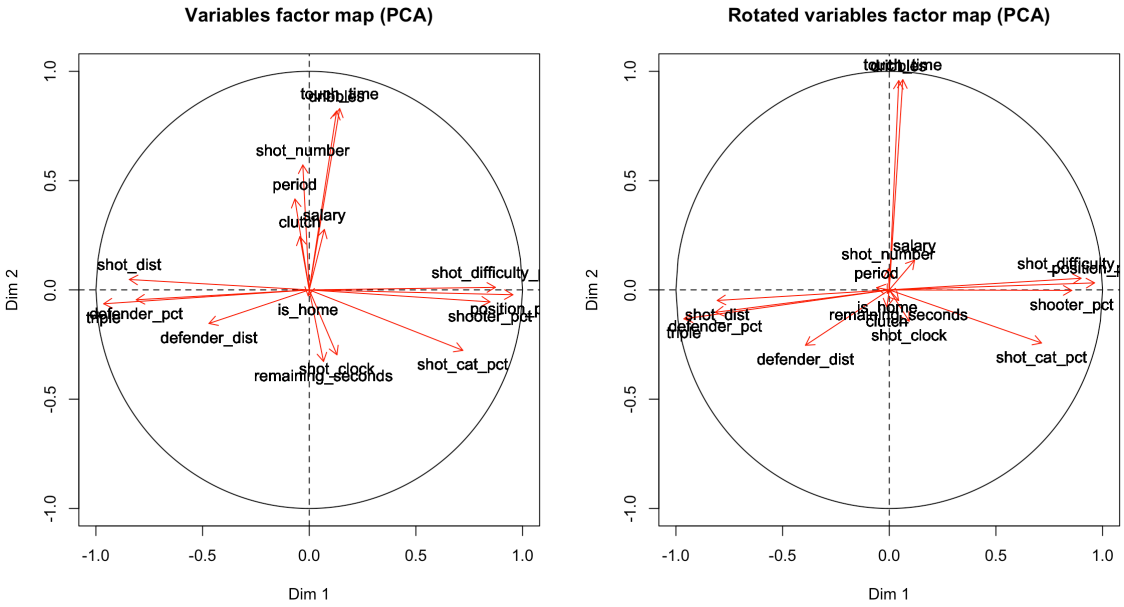
On both plots, we can clearly see that all the percentages we computed from the categorical variables, as well as the shot distance and whether is it a triple, are highly correlated with the first principal components. The second one is mainly correlated with the touch time and the number of dribbles, whereas the rest of components don't have that clearly correlated features.

Since our data set contains so many observations, it doesn't make much sense to plot the factor map of individuals. We will show though the biplot, which will allow us to see the shape of its cloud of points as well as the direction of growth of the variables.



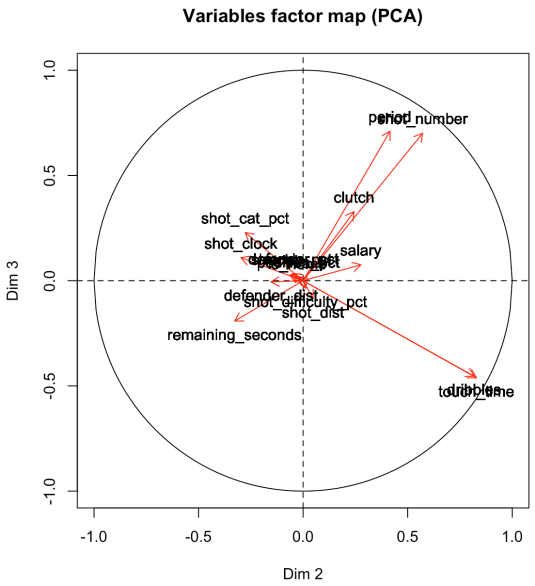
It's clear to see the separation in two main blocks of individuals in the first factorial plane, due to a gap of individuals in a part of the first factorial plane. That is probably caused by the correlation of Boolean variables with the component (e.g. triple). We can confirm that by looking at the variables whose directions of growth goes towards these groups. The left group have higher values of the shot distance, as well as triple (which means that they are triples). On the other hand, the variables that grow towards the group on the right are those whose values represent the rate success on their categories. Therefore, they will be "easier" shots. The observations with high values on the coordinates of the second component, will correspond to longer individual plays, since the values of the number of dribbles and touch time will be above the average.

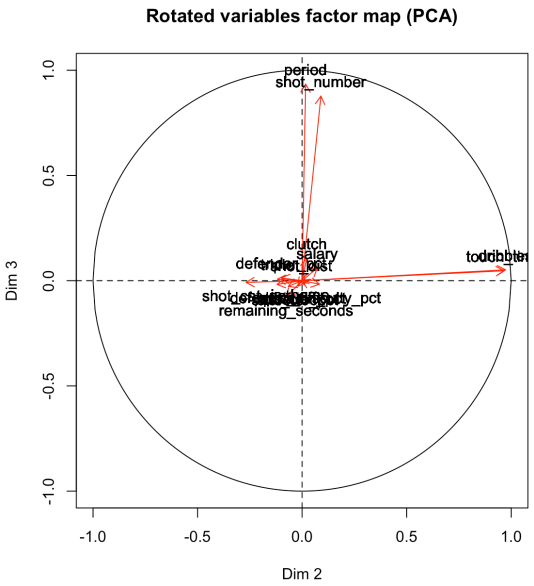
Despite some variables already being highly correlated with the main principal components, we will apply the varimax rotation in order to refine our found latent factors and align other latent factors with the rest of dimensions.



We can see above the plot on the first factorial plane of the factor map of variables. The one on the left corresponds to the original one, and the one on the right to the one after the varimax rotation. We can see that we have lost significance of the variables which were already less significant, but we have improved both the significance and correlation with the first components of the variables which were already more significant.

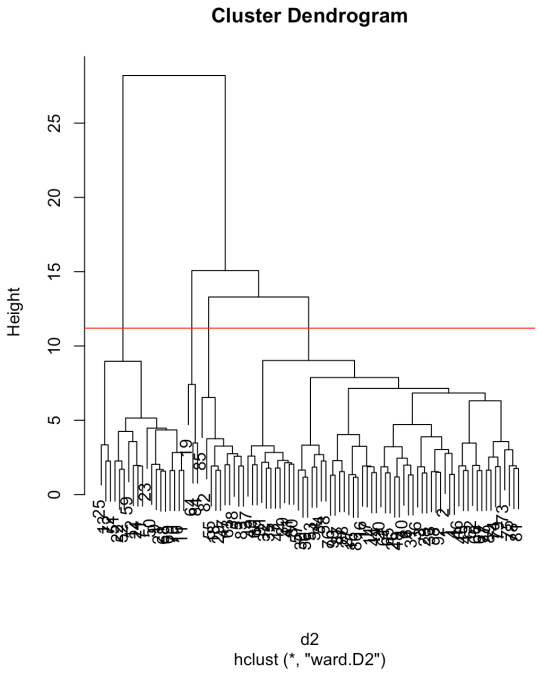
In the plot below, we can see the comparison we have just seen, but using the second and third components instead of the first and second. In this case the rotation clearly helps to align the latent factors with the components. We can conclude that the latent factors for the first three components are (briefly summarised): shot difficulty, play duration and moment of the game.

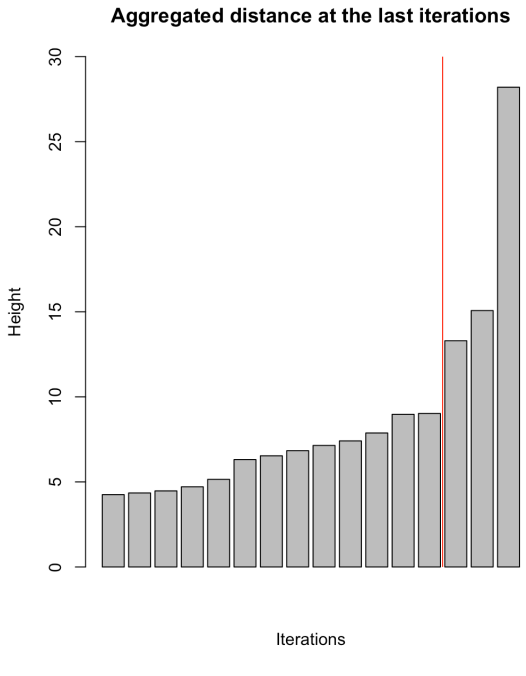




5-. Clustering

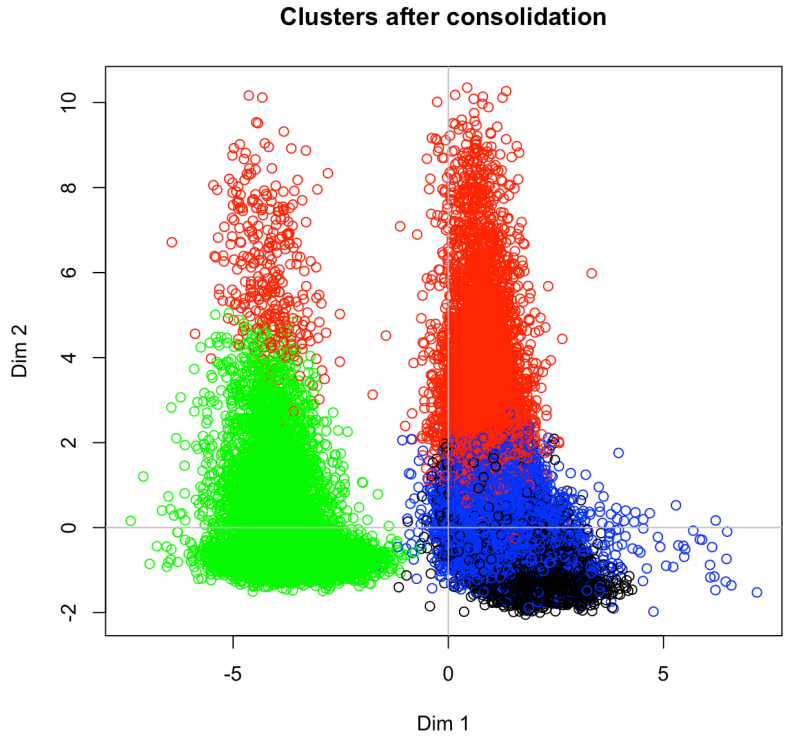
We will now perform clustering in order to find groups of shots that are homogeneous and distinct among them, so that we can obtain specific information about each group. We will do it in the factorial space because it helps reducing the dimensionality, and it takes into account the structural component of the data, discarding the noise. Given that we are working with a large dataset, we will perform a commonly used technique that requires much less computing time with quite good results. This technique consists on performing twice the k-means algorithm with a random initialization and *k* equals to 14. Then, we form the cross-table of both partitions and compute the centroids of its non-empty cells. Finally, we perform hierarchical clustering of the computed centroids weighted by the number of individuals per cell.





The plots above show the aggregated distance at each iteration of the hierarchical clustering algorithm. We have decided to cut at the second highest jump in height – starting by the end– as the first one would only leave us with two clusters, and we want to find more.

After the hierarchical clustering, we will perform consolidation with the k-means algorithm, using the results of the previous clustering as the initialization. The plot below shows the distribution of the final clusters in the first factorial plane.



Now we will use the *catdes* function from R to find some characteristics about each one of the obtained clusters. We will start by the cluster shown in the plot as black.

```
> catdes.res$category$black[1:6,]
                                  Cla/Mod  Mod/Cla    Global       p.value   v.test
position=C                       45.10756 36.13344 19.134625  0.000000e+00      Inf
shot_cat=Drive                   69.41424 10.39664  3.577703  0.000000e+00      Inf
shot_cat=Cut                     97.70788 61.36065 15.001004  0.000000e+00      Inf
shot_difficulty=Tightly Contested 51.60576 44.06619 20.397048  0.000000e+00      Inf
success=1                        32.31523 61.30812 45.317990  0.000000e+00      Inf
shot_difficulty=Contested        31.98059 37.05280 27.675434 7.962972e-232 32.50897
> rbind(head(catdes.res$quanti$black, 4), tail(catdes.res$quanti$black, 3))
               v.test Mean in category Overall mean sd in category Overall sd p.value
shot_cat_pct   215.92372        0.5929878    0.4531240     0.10016108 0.10243518       0
shot_clock     113.06306       16.4401313   12.1766289     5.83554975 5.96334320       0
shooter_pct    105.00273        0.5046568    0.4530911     0.05471912 0.07766126       0
position_pct   104.91672        0.4950375    0.4530459     0.01953342 0.06329395       0
defender_dist  -93.94926        0.7528008    1.2529515     0.48791858 0.84188260       0
triple         -94.13868        0.0000000    0.2616454     0.00000000 0.43953054       0
shot_dist     -179.23118        1.0471381    4.1195008     0.70264621 2.71083423       0
```

```
> catdes.res$category$red[1:5,]
                        Cla/Mod   Mod/Cla     Global      p.value   v.test
position=PG           25.83744 66.266770 24.9498042 0.000000e+00      Inf
shot_cat=Long ISO    100.00000  4.411765  0.4291738 0.000000e+00      Inf
shot_cat=ISO/Post up  62.11427 90.879773 14.2330087 0.000000e+00      Inf
shooter=chris paul    39.54984  3.173375  0.7805441 1.640412e-87 19.83000
shooter=john wall     34.36989  2.708978  0.7667403 2.541664e-62 16.66044
> rbind(head(catdes.res$quanti$red, 3), tail(catdes.res$quanti$red, 1))
              v.test Mean in category Overall mean sd in category Overall sd p.value
dribbles    220.03505       10.38209494    2.0452640       4.6037551  3.5110467       0
touch_time  208.75630        9.62011094    2.8050020       3.7810823  3.0252460       0
shot_number  51.13485        9.28237874    6.6474124       5.8458094  4.7751342       0
triple      -46.37876        0.04166667    0.2616454       0.1998263  0.4395305       0
```

```
> catdes.res$category$green[1:6,]
                          Cla/Mod  Mod/Cla   Global       p.value   v.test
shot_cat=Spot Up Three    99.99424 84.57304 21.78873  0.000000e+00      Inf
shot_difficulty=Wide Open 57.94072 37.80019 16.80680  0.000000e+00      Inf
shot_difficulty=Open      35.94169 48.99898 35.12072  0.000000e+00      Inf
position=SG               37.96757 28.62779 19.42451 4.693531e-308 37.51796
success=0                 30.70109 65.16635 54.68201 1.108797e-272 35.28200
position=SF               36.94296 24.22914 16.89589 8.841445e-219 31.57238
> rbind(head(catdes.res$quanti$green, 4), tail(catdes.res$quanti$green, 4))
                     v.test Mean in category Overall mean sd in category Overall sd p.value
triple              279.2912        0.9998539    0.2616454     0.01208774 0.43953054       0
defender_pct        218.5616        0.6494385    0.5469902     0.05933349 0.07794678       0
shot_dist           206.8167        7.4909860    4.1195008     0.67211217 2.71083423       0
defender_dist       115.5804        1.8381033    1.2529515     0.88845177 0.84188260       0
shot_cat_pct       -165.5477        0.3511463    0.4531240     0.02253797 0.10243518       0
shooter_pct        -219.4851        0.3505867    0.4530911     0.05031778 0.07766126       0
shot_difficulty_pct -257.6304       0.3512727    0.4531461     0.03185423 0.06575537       0
position_pct        -268.2899       0.3509286    0.4530459     0.01138428 0.06329395       0
```

```
> catdes.res$category$blue[1:4,]
                       Cla/Mod  Mod/Cla   Global       p.value   v.test
shot_cat=Other        78.42067 33.74521 17.48067  0.000000e+00      Inf
shot_cat=Catch&Shoot  81.38410 55.07228 27.48971  0.000000e+00      Inf
position=PF           50.84330 24.30495 19.41949 2.376632e-180 28.63627
position=C            49.47534 23.30409 19.13463 1.593360e-133 24.59057
> rbind(head(catdes.res$quanti$blue, 2), tail(catdes.res$quanti$blue, 2))
                     v.test Mean in category Overall mean sd in category Overall sd p.value
shot_difficulty_pct  149.6067     4.952777e-01    0.4531461    0.022178966 0.06575537       0
position_pct         134.5294     4.895133e-01    0.4530459    0.018887200 0.06329395       0
defender_pct        -106.5903     5.114073e-01    0.5469902    0.044231982 0.07794678       0
triple              -138.9784     3.089089e-05    0.2616454    0.005557872 0.43953054       0
```

6-. Prediction Model

Now that we have done the analysis of the data set, we will fit a model in order to be able to predict future observations. For instance, it could be used by coaches in order to

estimate the likelihood of the success of a specific play. Note that we are clearly in a binary classification problem which will be address by building a decision tree and then, comparing its predictive performance with a random forest.

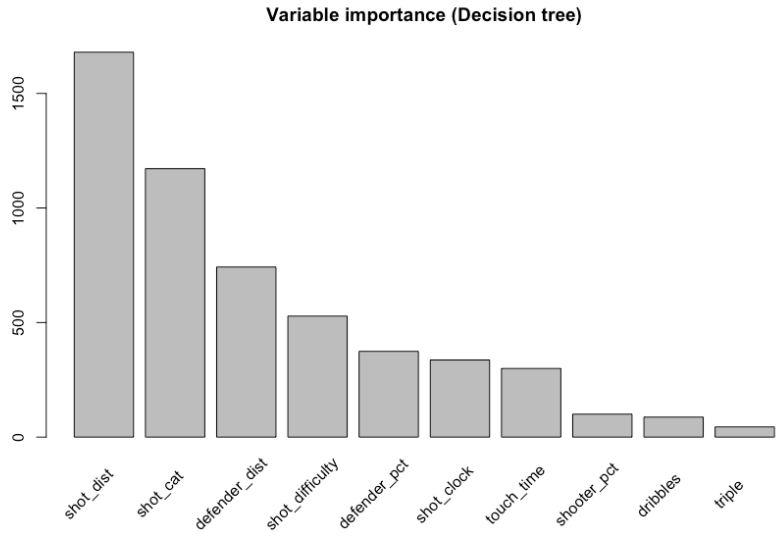
Before entering into practical details of the performance of those models with our dataset, we want to state the main difference between them. Decision trees are built on the entire training set, using all features. Although they are post-pruned, they tend to over-fit as they have high variance. A small change in the data may drastically change the model. However, they are quite informative and interpretable.

On the other hand, random forests are an ensemble model of several decision trees which "randomizes" the process. First, they use a different sample to build each tree (i.e. bagging) and then, at each node of each tree they use a random subset of the available features. This allows to reduce variance.

6.2-. Decision tree

We have used R's package *rpart* to fit the decision tree. The validation protocol used was 10-fold cross validation to obtain an honest estimate of the generalization error of the tree on new unseen data. In addition, we have used this error to compare the possible values of the complexity parameter *cp*, which specifies how the cost of the tree is penalized by the number of leaves.

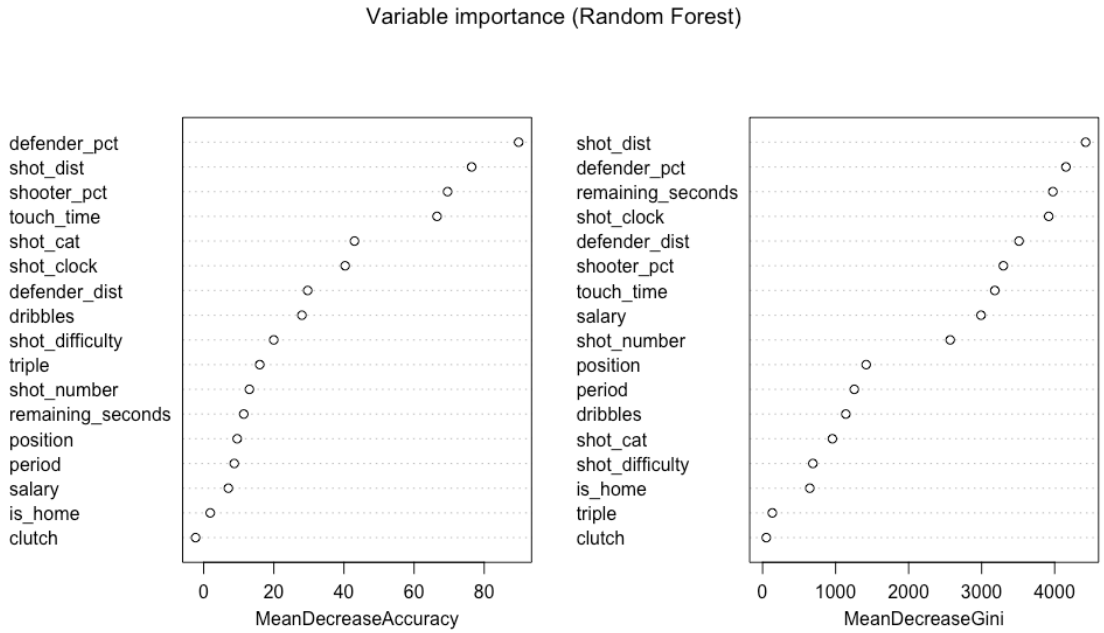
Once obtained the best *cp* with value 0.00028, we have pruned the tree and use it to predict the observations of the test set and to obtain the importance of the variables, displayed in the following plot.



6.3-. Random forest

With random forests there is no need of cross validation, as the out-of-bag error yields an honest estimate of the generalization error. Then, we have defined a grid of candidate values for both the number of trees in the ensemble (*ntrees* parameter) and the number of variables taken into account at each node (*mtry* parameter) and we have chosen the combination that minimizes the OOB error.

The best random forest model (*mtry* = 3 and *ntree* = 1000) was used to predict the observations of the test set and the variable importance too.



6.4-. Comparison

After predicting the test set observations with both models, we have obtained the following confusion matrices:

	NO	**YES**
NO	15302	9636
YES	3372	5841

	NO	**YES**
NO	14958	9274
YES	3716	6203

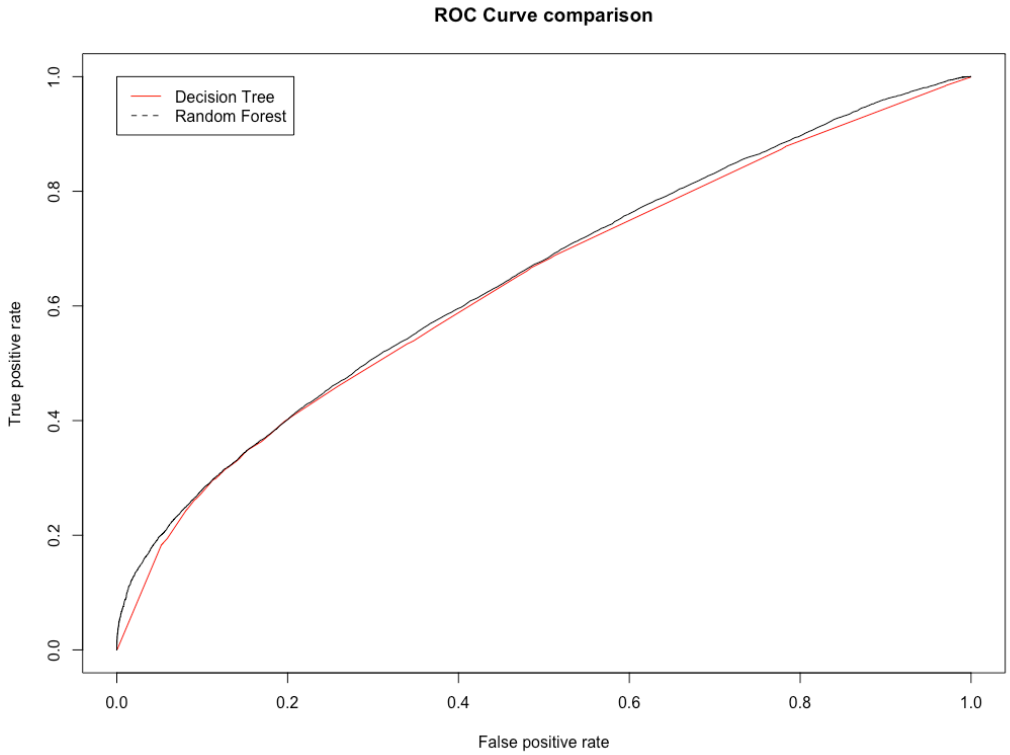
Table 1. Left: Decision tree. Right: Random forest.

From the previous matrices, we can compute the accuracy of the models, which is pretty similar. However, the random forest has a higher precision. As a team coach would be interested in predicting the successful plays, precision is a more significative measure in this case.

	Accuracy	Prediction (positive)
Decision Tree	61.91 %	37.77 %
Random Forest	61.96 %	40.07%

Table 1. Comparison of models with test set.

Finally, we have plotted the ROC curves of both models in the same plot. Both are pretty similar, being the random forest a bit better.



7-. Conclusions