Final Project: Statistical Learning

Zach Bishof

December 11, 2019

Goal

Is there a way to predict which NBA player will make the All-Star Game? The idea of the All-Star Game is that the very best NBA players from that season get to play in it. It is meant to be a high honor, and quite often players get bonuses or sponsorship deals. It is a highly coveted position for a player. The way the NBA decides if a player should get to play in the All-Star game is largely based on votes from the public and other influential people in the NBA organization.

The goal of this study is to see if a model can be built to predict whether a player makes the All-Star Game. It should be noted that the data used had covariates that would make this trivial (for instance rank), because they provide the actual information that the NBA uses to make that decision. The goal of this study was to see if there were factors that exist *before* voting occurs that can predict a player's All-Star Game acceptance.

Data

The Data used in this study was collected from kaggle (see link: https://www.kaggle.com/davra98/nba-players-20162019). It provided 1408 observations. These were players statistics for the seasons 2016-17, 2017-18, and 2018-19. There were 45 covariates, several of which were not used in this study.

A complete list of the covariates (listed in order of column number):

1) Rk - Rank calculated in part by all-star votes. This was dropped from the model, as it makes calculations trivial (see Goal section).
2) Player.x - Name of the player. Not relevant in model building.
3) Player_id - Id number assigned to each player. Not relevant in model building.
4) Pos1 - Primary posititon of a player.
5) Pos2 - Secondary position of a player. This was dropped from model building, because it had a lot of NA values, but more importantly because it almost never happens in an NBA game.
6) Age - Age of the player in years.
7) Tm - The team the player played for that season.
8) G - Number of games the player played in
9) GS - Games Started. Number of games the players started in.
10) MP - Minutes played per game.
11) FG - Field goals per game.
12) FGA - Field goal attempts per game.
13) FG. - Field goal percentage per game.
14) X3P - Three pointers made per game.
15) X3PA - Three pointers attempted per game.
16) X3P. - Three pointer percentage per game.
17) X2P - Two point field goals made per game.
18) X2PA - Two point field goals attempted per game.
19) X2P. - Two point field goal percentage per game.
20) eFG. - Effecttive field goal percentage per game (see the following link for a full explanation of this: https://en.wikipedia.org/wiki/Effective_field_goal_percentage).
21) FT - Free throws per game.
22) FTA - Free throw attempts per game.
23) FT. - Free throw percentage per game.
24) ORB - Offensive rebounds per game.

25) DRB - Defensive rebounds per game.
26) TRB - Total rebounds per game.
27) AST - Assists per game.
28) STL - Steals per game.
29) BLK - Blocks per game.
30) TOV - Turnovers per game.
31) PF - Personal fouls per game.
32) PTS - Points per game.
33) Salary - Yearly salary of the player.
34) mean_views - Daily average views on player's wikipedia page.
35) Season - season data was collected
36) Conference - Which conference the player played in (East or West)
37) Role - Role the player had on team. (Not included in model building, because source does not give explanation of this.)

38-44) Fvot, FRank, Pvot, PRank, Mvot, MRank, and Score - various ranking methods that the NBA uses for determining who gets to go to the All-Star game. This was not included for the reason described in the last paragraph of the previous section.

45) Play - Whether the player made the All-Star game. This was coded to be 1=Yes 0=No

Clean-Up comments

This data had to be cleaned. As stated, the covariates regarding rank and voting were dropped, as they did not aid in the goal. Also, the data was highly imbalanced, because there were only 73 players who made the All-Star game. This means that about 95% of the data was for players who missed the All-Star game.

This meant that the data could be cleaned without too much loss of information. After the covariates regarding voting were dropped (columns 38-44), the observations with NA values were omitted, except for Isiah Thomas. The only players with NA values seemed to be players for whom there was little information, and thus less likely to make the All-Star game. Isiah Thomas was the only exception, because for some reason he was missing mean_view, so it became necessary to manually look up how many views his wikipedia page had for the season in question. Upon doing all of this, the data set shrunk to 1135 observations, which means only 273 observations were dropped, and none of them were for players that made the All-Star Game.

Then the data was broken into 4 different sets. A training set of 900 observations, a test set of the remaining 235 observations, a training set for the cross validation process (details on that in the section regarding KNN), and a validation set. These sets were assigned randomly by R.

Other things that were necessary to do before breaking up the various sets was telling R to read certain variables as numeric as opposed to integer, recoding Play to be 1 and 0 instead of yes and no, and omitting columns regarding PoS1 and PoS2 because they did not aid in model building.

```
library(dplyr)
## Read original data set into R
nba<- read.csv('C:/Users/user/Documents/R/Stat Learning/Final Project/nba_final.csv')
##Set Play column as All Star=1 Not All Star=0
nba <- nba %>% mutate(Play = ifelse(Play == "No",0,1))
nba <- mutate(nba, Play=as.integer(Play))
nba <- mutate(nba, Salary=as.numeric(Salary))
nba <- mutate(nba, Age=as.numeric(Age))
nba<- mutate(nba, G=as.numeric(G))
nba <- mutate(nba, GS=as.numeric(GS))
## For some reason the Wikipedia views for Isiah Thomas were missing. I had to go to his wikipedia page
nba[159, 34]= 2398.397260
## Removing POS2 and columns regarding ranks, because they are mostly NA, and because POS2 is not very
nba <- mutate(nba[-c(4,5, 40:43)])
```

```r
## I had to omit the players with entries as NA. it turns out that this did not eliminate any players w
nba <- na.omit(nba)
###Break Data into train, validation, and test set
n <- nrow(nba)
set.seed(43855385) # You can change the seed to your favorite number
reorder = sample(1:n) # shuffle
train = nba[reorder[1:900],]
traink = nba[reorder[1:600],]
validation = nba[reorder[601:900],]
test = nba[reorder[901:1135],]
```

Methods

In order to achieve the goal of this report, we first identify the fact that we are dealing with a classification problem. As such, two potentially good statistical methods are logistic regression and KNN.

Logistic Regression

The idea of logistic regression is somewhat similar to linear regression, except instead of fitting a linear function to the data, we fit the first equation below by maximizing the second equation (for each parameter estimate), the likelihood function.

$\log \frac{P(X)}{1-P(X)} = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p$

$l(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=1} (1 - p(x_{i'}))$

Essentially, we are fitting an S-shaped curve to the data, and saying that all of the values at the high end of the curve are predicted to be Play=1, and those on the bottom are predicted to be Play=0. We do this by saying values that are predicted above 0.5 are on the high end, and below on the low end.

The assumptions that logistic regression makes are as follows:

1) The response is binary. This is true in our case, since Play is either 1 or 0.
2) Observations are independent of each other. This is true, since our data does not come from repeated measurements.
3) Little to no multicolinearity. This was something that had to be addressed later.
4) Linearity of predictors with log-odds. This appears to be a safe assumption here.
5) A large sample size. We do in fact have this.

One of the drawbacks of logistic regression is that it can be vulnerable to to overfitting. This is something that will be addressed after forward and backward selection is performed.

Checking Some simple regression models for predicting Salary

The first thing that was done was running a simple logistic regression model for every relevant predictor, so as to determine which model would be best suited for starting the forward selection process. The following R chunk demonstrates this, but the output is omitted, as it would take up a lot of space. Essentially, we are looking for the lowest AIC value, which turns out to be the model using PTS as the predictor.

```r
Age.Play <- glm(Play~Age, train, family = "binomial")
summary(Age.Play)
wiki.Play <- glm(Play~mean_views, train, family="binomial")
summary(wiki.Play)
FG.Play <- glm(Play~FG, train, family="binomial")
summary(FG.Play)
Pos1.Play <- glm(Play~Pos1, train, family="binomial")
```

```
summary(Pos1.Play)
Pos2.Play <- glm(Play~Pos2, train, family="binomial")
summary(Pos2.Play)
Tm.Play <- glm(Play~Tm, train, family="binomial")
summary(Tm.Play)
G.Play <- glm(Play~G, train, family="binomial")
summary(G.Play)
GS.Play <- glm(Play~GS, train, family="binomial")
summary(GS.Play)
MP.Play <- glm(Play~MP, train, family="binomial")
summary(MP.Play)
FGA.Play <- glm(Play~FGA, train, family="binomial")
summary(FGA.Play)
FG..Play <- glm(Play~FG., train, family="binomial")
summary(FG..Play)
X3P.Play <- glm(Play~X3P, train, family="binomial")
summary(X3P.Play)
X3PA.Play <- glm(Play~X3PA, train, family="binomial")
summary(X3PA.Play)
X3P..Play <- glm(Play~X3P., train, family="binomial")
summary(X3P..Play)
X2P.Play <- glm(Play~X2P, train, family="binomial")
summary(X2P.Play)
X2PA.Play <- glm(Play~X2PA, train, family="binomial")
summary(X2PA.Play)
X2P..Play <- glm(Play~X2P., train, family="binomial")
summary(X2P..Play)
eFG..Play <- glm(Play~eFG., train, family="binomial")
summary(eFG..Play)
FT.Play <- glm(Play~FT, train, family="binomial")
summary(FT.Play)
FTA.Play <- glm(Play~FTA, train, family="binomial")
summary(FTA.Play)
FT..Play <- glm(Play~FT., train, family="binomial")
summary(FT..Play)
ORB.Play <- glm(Play~ORB, train, family="binomial")
summary(ORB.Play)
DRB.Play <- glm(Play~DRB, train, family="binomial")
summary(DRB.Play)
TRB.Play <- glm(Play~TRB, train, family="binomial")
summary(TRB.Play)
AST.Play <- glm(Play~AST, train, family="binomial")
summary(AST.Play)
STL.Play <- glm(Play~STL, train, family="binomial")
summary(STL.Play)
BLK.Play <- glm(Play~BLK, train, family="binomial")
summary(BLK.Play)
TOV.Play <- glm(Play~TOV, train, family="binomial")
summary(TOV.Play)
PF.Play <- glm(Play~PF, train, family="binomial")
summary(PF.Play)
PTS.Play <- glm(Play~PTS, train, family="binomial")
summary(PTS.Play)
```

```
Salary.Play <- glm(Play~Salary, train, family="binomial")
summary(Salary.Play)
mean_views.Play <- glm(Play~mean_views, train, family="binomial")
summary(mean_views.Play)
```

Forward and Backward Selection

Forward and backward selection was done in order to find the best multiple logistic regression model.

The idea for forward selection is to start with a simple regression model (the one involving PTS here), and then running each model involving two predictors that include PTS. The model with the lowest AIC is kept, and then the process is repeated for three predictors. The process terminates when the model without adding a new predictor has a lower AIC value than all of those which would add a new predictor.

Backward regression is a similar idea, except this time we start with the full model (i.e. the model containting *every* potential predictor), and then test each model where a predictor has been removed, comparing AIC values along the way. Here the process is terminated when the model that does not have a predictor removed has a lower AIC value than all of the models with a predictor removed.

This would take a very long time to do manually, but fortunatelly it is possible to make R do it for us!

```
## Model found from previous R chunk to have lowest AIC
PTS.Play <- glm(Play~PTS, train, family="binomial")


## Forward Regression
library(MASS)
initialfit <- PTS.Play
finalfit<- glm(Play~. , data=train[-c(1:3,5, 33:38)], family = "binomial")
forward <- stepAIC(initialfit, direction="forward",scope=list(lower=initialfit,upper=finalfit))
forward$anova # display results
```

```
# Backward Regression
library(MASS)
fit <- glm(Play~. , data=train[-c(1:3,5, 33:38)], family = "binomial")
backward <- stepAIC(fit, direction="backward")
backward$anova # display results
```

The model found from foward selection was significantly less flexible (i.e. fewer predictors). It also had fewer issues with multicolinearity, which is something we cannot have in a logisitc regression model. The model from forward selection was chosen. This model was

Play ~ PTS + Age + GS + mean_views + MP + DRB + PF + AST + G

However, a few of these variables had p-values above .1 associated with their parameter estimates. This essentially means that these predictors were not significant in the model, thus they were removed one at a time. This was done one at a time, because it was possible that a p-value would drop for one predictor after removing the other. This was not the case. The predictors that were removed were PF and GS. The final model, built on the training set, can be seen below.

```
## Checking for Multicolinearity. A VIF score above 5 is problematic. None seem to have this issue.

library(car)
 vif(glm(Play ~ PTS + Age + mean_views + DRB + AST + G, data=train, family="binomial"))
```

```
##        PTS        Age mean_views        DRB        AST          G
##   1.520583   1.749373   1.169710   1.183456   1.159936   1.450780
```

```
## Summary of the final model built on the training data


summary(glm(Play ~ PTS + Age + mean_views + DRB + AST + G , data=train, family="binomial"))
```

```
##
## Call:
## glm(formula = Play ~ PTS + Age + mean_views + DRB + AST + G,
##     family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2010  -0.0786  -0.0148  -0.0010   3.6773
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.092e+01  4.984e+00  -6.205 5.47e-10 ***
## PTS          3.864e-01  6.176e-02   6.257 3.93e-10 ***
## Age          2.920e-01  7.313e-02   3.993 6.53e-05 ***
## mean_views   3.404e-04  1.274e-04   2.672  0.00754 **
## DRB          3.660e-01  1.205e-01   3.039  0.00238 **
## AST          3.172e-01  1.215e-01   2.611  0.00902 **
## G            1.456e-01  3.645e-02   3.995 6.47e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 440.87  on 899  degrees of freedom
## Residual deviance: 123.82  on 893  degrees of freedom
## AIC: 137.82
##
## Number of Fisher Scoring iterations: 9
```

Recall that each of these parameter estimates tell us the change in the log-odds for Play with a change in value by 1 on each predictor.

Confusion Matrix for Prediction on Training Set

```
## Have R make predictions on train season set
pred_LR1 <- predict(glm(Play ~ PTS + Age + mean_views + DRB + AST + G , train, family="binomial"), trai
LR.pred1 <- ifelse (pred_LR1 > 0.5, 1,0)
## Have R make a confusion matrix on train set
lr.table1 <- table(train$Play, LR.pred1)
lr.table1
```

```
##    LR.pred1
##       0   1
##   0 831   9
##   1  14  46
```

Misclassification, Sensitivity, and Specificity

Using the above confusion matrix, misclassfication, sensitivity, and specificity was calculated in R for the training set.

```
LR1.TER <- 1- (sum(diag(lr.table1))/sum(lr.table1))
LR1.TER # test error rate
```

```
## [1] 0.02555556
```

```
LR1.sens <- lr.table1["1","1"]/sum(lr.table1["1",])
LR1.sens # sensitivity
```

```
## [1] 0.7666667
```

```
LR1.spec <- lr.table1["0","0"]/sum(lr.table1["0",])
LR1.spec # specificity
```

```
## [1] 0.9892857
```

Confusion Matrix for Prediction on Test Set

```
## Have R make predictions on train season set
pred_LR2 <- predict(glm(Play ~ PTS + Age + mean_views + DRB + AST + G , train, family="binomial"), test
LR.pred2 <- ifelse (pred_LR2 > 0.5, 1,0)
## Have R make a confusion matrix on train set
lr.table2 <- table(test$Play, LR.pred2)
lr.table2
```

```
##    LR.pred2
##       0   1
##   0 220   2
##   1   4   9
```

Misclassification, Sensitivity, and Specificity

Using the above confusion matrix, misclassfication, sensitivity, and specificity was calculated in R for the test set.

```
LR2.TER <- 1- (sum(diag(lr.table2))/sum(lr.table2))
LR2.TER # test error rate
```

```
## [1] 0.02553191
```

```
LR2.sens <- lr.table2["1","1"]/sum(lr.table2["1",])
LR2.sens # sensitivity
```

```
## [1] 0.6923077
```

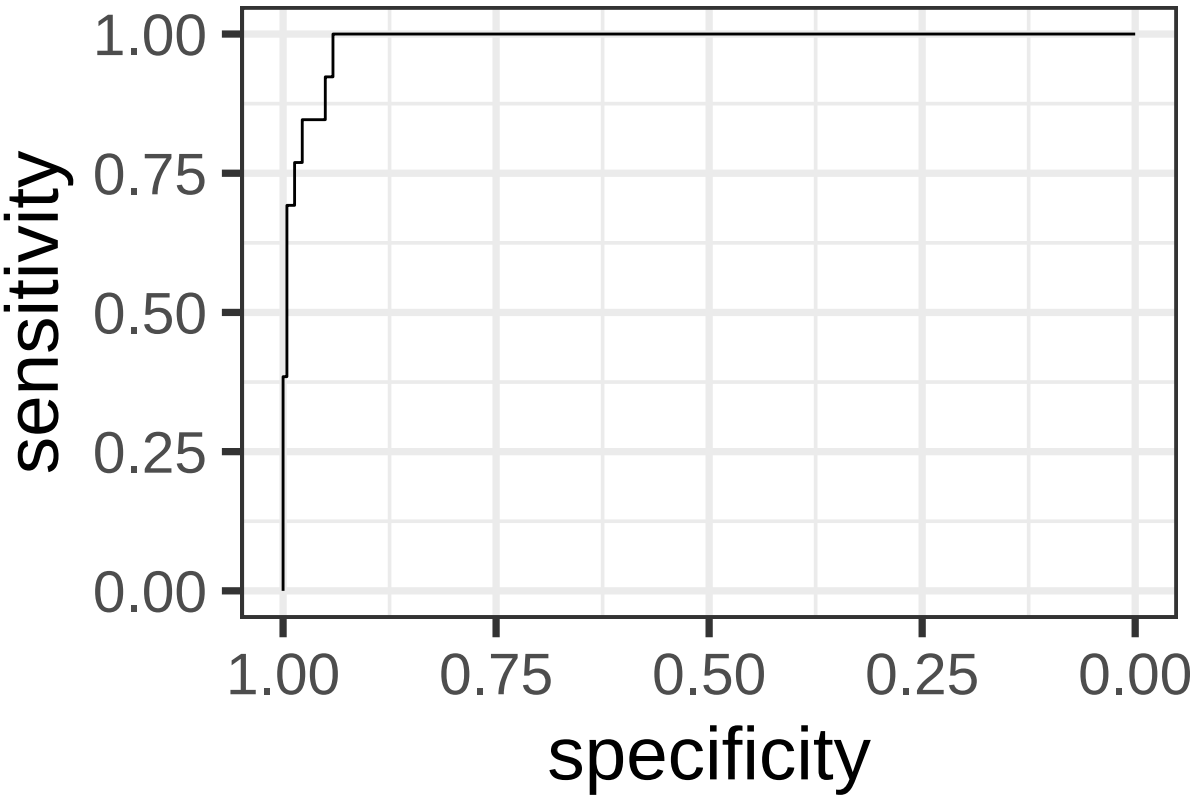
```
LR2.spec <- lr.table2["0","0"]/sum(lr.table2["0",])
LR2.spec # specificity
```

```
## [1] 0.990991
```

Area Under Curve For ROC

One way to check the fit of the logistic regression model is to use AUC of ROC. If this value is close to 1, then the model is considered to be a good fit. This appears to be the case, since the AUC is 0.9875, as can be seen below.

```
library(pROC)
library(ggplot2)
LR.roc <- roc(test$Play, pred_LR2,legacy.axes=TRUE)
ggroc(LR.roc)+theme_bw(28)
```




```
auc(LR.roc)
```

```
## Area under the curve: 0.9875
```

KNN

The process of K-Nearest Neightbor is to essentially categorize each observation in accordance with the other observations nearest them. Essentially, given some observation for which we wish to make a prediction, the Euclidean distance to that observation is calculated for each observation in the training set. Then, the tuning parameter N tells us how many of the nearest observations we look at, and use majority rule to determine which classification is assigned.

Determining N is the difficult part. To do this, cross-validation was used. The idea of this is to first split the training set into two parts; a smaller training set and a validation set (see the data section for how this was done). Then, a model is built on the smaller training set, and validated on the validation set. Each time a model is built and then validated, the process is repeated using a different value for N. Essentially, the value for N that gives the lowest missclassification rate on the validation set is kept.

The disadvantages to using this method is that the algorithim does not learn from the training data, but rather uses the training data to make classifications. What this means is that KNN is not great at dealing with noisy data. Also, if the data set is very large, then it can take a long time to process.

Cross-Validation

We use the validation set to tune the parameter (k), in order to obtain the most optimal choice for minimizing misclassification rate.

```
library(class)

KNN.play   <- knn(traink[-c(1:3,5, 33:38)], validation[-c(1:3,5, 33:38)], cl=traink$Play, k=6,prob=TRUE

##KNN.play## Gives predictions on validatin set. Commented out for ease of use. Feel free to put this b
```

Producing a confusion matrix for validation set

```
knn.table <- table(validation$Play,KNN.play)


knn.table
```

```
##    KNN.play
##       0   1
##   0 280   2
##   1  13   5
```

After doing cross-validation, it seems that k=6 is the sweet spot. Not all of the code used is included, since basically the same r chunks were ran over and over, changing only k until the model had the lowest misclassification rate on the validation set. This seemed to happen for k=5 or k=6. k=6 was chosen, because it would provide a less flexible model, and this would be better when making predictions on the test set.

Build Model Using K=6

```
library(class)

KNN.play1   <- knn(train[-c(1:3,5, 33:38)], test[-c(1:3,5, 33:38)], cl=train$Play, k=6,prob=TRUE)


##KNN.play## Gives predictions on validatin set. Commented out for ease of use. Feel free to put this b
```

Confuison Matrix for Predicitons on Test Set

```
knn.table1 <- table(test$Play,KNN.play1)


knn.table1
```

```
##    KNN.play1
##       0   1
##   0 217   5
##   1  10   3
```

Misclassification, Sensitivity, and Specificity

Using the above confusion matrix, misclassfication, sensitivity, and specificity was calculated in R for the test set.

```
knn.TER <- 1- (sum(diag(knn.table1))/sum(knn.table1))
knn.TER # test error rate
```

```
## [1] 0.06382979
```

```r
knn.sens <- knn.table1["1","1"]/sum(knn.table1["1",])
knn.sens # sensitivity
```

```
## [1] 0.2307692
```

```r
LR2.spec <- knn.table1["0","0"]/sum(knn.table1["0",])
LR2.spec # specificity
```

```
## [1] 0.9774775
```

Results

Now that both methods are complete, it is important to take a look at the test error (i.e. misclassification, sensitivity, and specificity on test set). Upon doing this it is clear that the model built by logisitc regression did a better job. The values for those for logistic regression were 0.02553191, 0.6923077, and 0.990991 respectively. These values for KNN were 0.06382979, 0.2307692, and 0.9774775 respectively. Logistic regression did better for each of these.

Also, logistic regression had a fairly decent AUC value for the ROC curve. This is an indicator the the logistic regression model was a good fit for the data.

As such, it would make sense to proceed with the logistic regression model that was chosen, however now built on the entire data set.

Conclusion

In the end neither of these methods produced perfect results. If this was in a professional setting, then it would be wise to ask for more time to build a better model, and try using some different types of methods.

The logisitc regression model produced a decent test misclassification rate of less than 3 percent, however the sensitivy rate was only about 69 percent. A likely reason this happened was due to the imbalance in the data. It is difficult to make predictions for a classification that is so rare. That being said, the specificity was very high, achieving a score of about 99 percent. What this means is that this model is very good at identifying players that will not make the All-Star game, but it is less good at identifying players that will make the All-Star game.

The final model can be seen below. Recall that each parameter estimate represents the change in the log-odds for each change in the given predictor by 1.

This implies that the largest effect on the log-odds, of the predictors included in this model was the number of games a player has played in that season. This assertion is made, because the parameter estimate for the predictor G is the larges of those listed below. That being said, none of them were very far apart (with the exception of mean_views), since they mostly all had the same order of magnitude.

```r
## Summary of the final model built on the training data


summary(glm(Play ~ PTS + Age + mean_views + DRB + AST + G , data=nba, family="binomial"))
```

```
##
## Call:
## glm(formula = Play ~ PTS + Age + mean_views + DRB + AST + G,
##     family = "binomial", data = nba)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2029  -0.0726  -0.0146  -0.0013   3.7252
```

```
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.932e+01  4.227e+00  -6.936 4.04e-12 ***
## PTS          3.876e-01  5.562e-02   6.969 3.19e-12 ***
## Age          2.603e-01  6.120e-02   4.253 2.11e-05 ***
## mean_views   3.590e-04  1.086e-04   3.306 0.000948 ***
## DRB          3.851e-01  1.065e-01   3.617 0.000298 ***
## AST          3.272e-01  1.098e-01   2.979 0.002892 **
## G            1.323e-01  3.117e-02   4.246 2.18e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 541.81  on 1134  degrees of freedom
## Residual deviance: 151.49  on 1128  degrees of freedom
## AIC: 165.49
##
## Number of Fisher Scoring iterations: 9
```