UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

HIC ET UBIQUE TERRARUM

Modèle prédictif

GROUPE - J

WAGNER Octave / GAALOUL Bilel / MACHROUH SAMI / IBRAHIM Djawid / KUOY Alexandre / OUYAHIA Virgile / FRASLIN Tangi

Techniques Quantitatives de Gestion

Logiciel R

Table des matières

Introduction

Selon Sportbuzzbusiness, la NBA génère près d'une dizaine de milliards de dollars de revenus chaque année. Et selon Forbes, chaque équipe est évaluée entre 1,5 et 5,8 milliards de dollars. Nous avons donc choisi la NBA comme sujet pour construire un modèle prédictif. Nous avons aussi fait ce choix car une majorité du groupe est fan de Basket.

Comment avons-nous construit la base de données et quelles informations y avons-nous ? Nous avons suivi un tutoriel de scraping de données relatives à la NBA sur Predictivehacks. Nous avons donc récupéré les données relatives aux matchs de la saison actuelle de la NBA sur le site ESPN, ensuite nous avons transformé les données présentes sur ce site afin de faire apparaître divers ratios, en suivant le tutoriel. Nous avons donc désormais une base de données complète (voir Annexes).

Quels modèles prédictifs avons-nous utilisés ? Nous avons utilisé 6 algorithmes pour construire divers modèles prédictifs : le logit, l'arbre, la forêt, le modèle naïve bayésien avec le correcteur de Laplace, le modèle kNN (k-Nearest Neighbours) ainsi qu'un modèle que nous n'avons pas traité en cours, le réseau de neurones. L'objectif est de prédire la victoire ou la défaite de l'équipe hôte.

Pour ce faire, nous avons décomposé la base de données en deux sous-ensembles : un sous-ensemble "nbatrain" et un sous-ensembles "nbatest". Le sous-ensemble "nbatrain" est composé de 818 matchs et le sous-ensemble "nbatest" des 110 autres matchs (environ 11% comme Titanic). Le modèle va analyser les données du sous-ensemble "nbatrain" et va essayer de prédire les résultats dans "nbatest". Les résultats vont être visibles de différentes manières selon les modèles mais pour tous les modèles, nous allons pouvoir trouver un AUC. Plus un AUC est proche de 1, plus il est bon, plus il est proche de 0 et plus il donne le résultat contraire à la réalité, un AUC proche de 0,5 reflète un modèle qui ne fonctionne pas vraiment. Nous allons donc garder celui qui a le meilleur AUC afin de réaliser un robot prédictif.

En effet, nous allons donc parler de divers modèles prédictifs, et garder le meilleur afin de construire un robot permettant de prédire la victoire ou la défaite de l'équipe hôte.

I. Présentation des différents modèles

A) Le modèle LOGIT

La fonction **glm(), associée au modèle logit,** nous permet d'expliquer des modèles binaires. C'est notre cas ici puisque on cherche à savoir si l'équipe va gagner ou bien perdre son match. On a obtenu une estimation pour chacune des équipes de leurs chances de gagner selon si elles jouent à domicile ou bien à l'extérieur.

Les variables significatives sont celles avec un point ou plusieurs étoiles. Au vu du nombre important des variables significatives, nous allons nous concentrer sur celles avec une étoile et plus. Cette significativité se mesure par la p valeur avec la colonne Pr(>z). De manière générale, lorsque la p valeur est inférieure à 5%, on considère que le modèle est significatif. Plus la p valeur est élevée, plus on a de chances de faire des erreurs et moins le modèle est précis. Lorsqu'on effectue un test économétrique, il s'agit de la valeur à partir de laquelle on rejette l'hypothèse.

	Estimate	Std. Error	z value	Pr(>z)	
H_URLTeamCHI	1.76440	0.72590	2.431	0.015073	*
H_URLTeamDET	-2.74574	0.76080	-3.609	0.000307	***
H_URLTeamGS	2.60661	0.88352	2.950	0.003175	**
H_URLTeamHOU	-2.49634	0.71611	-3.486	0.000490	***
H_URLTeamMIA	1.43254	0.68373	2.095	0.036155	*
H_URLTeamMIL	1.80627	0.71780	2.516	0.011856	*
H_URLTeamOKC	-2.35689	0.66038	-3.569	0.000358	***
H_URLTeamORL	-2.57530	0.75943	-3.391	0.000696	***
H_URLTeamPHX	2.70006	0.87828	3.074	0.002110	**
H_URLTeamSA	-1.39220	0.60852	-2.288	0.022146	*
H_URLTeamUTAH	1.49139	0.71627	2.082	0.037328	*
A_URLTeamBKN	-2.66949	0.88268	-3.024	0.002492	**
A_URLTeamDET	2.08970	0.78810	2.652	0.008012	**
A_URLTeamGS	-2.26048	0.94017	-2.404	0.016202	*
A_URLTeamHOU	2.46386	0.82266	2.995	0.002745	**
A_URLTeamIND	1.84475	0.69220	2.665	0.007697	**
A_URLTeamMEM	-2.04711	0.75327	-2.718	0.006575	**
A_URLTeamMIA	-1.51072	0.73776	-2.048	0.040588	*
A_URLTeamMIL	-1.78744	0.75964	-2.353	0.018622	*
A_URLTeamPHI	-2.01029	0.75138	-2.675	0.007463	**
A_URLTeamPHX	-4.49704	1.05367	-4.268	0.0000197	***
A_URLTeamPOR	2.10277	0.75317	2.792	0.005240	**
A_URLTeamTOR	-1.93232	0.74266	-2.602	0.009271	**
A_URLTeamUTAH	-2.21784	0.88301	-2.512	0.012016	*
H_Tpct	-6.56950	2.26050	-2.906	0.003658	**
A_ARpct	11.54324	2.82418	4.087	0.0000436	***
A_ARLast10	-3.26885	1.11914	-2.921	0.003491	**

Dans la colonne « Estimate », plus la valeur est élevée, plus elle influence positivement la probabilité de gagner un match. Inversement, si la valeur est négative, elle influence positivement la probabilité de perdre un match.

On remarque que plusieurs variables sont le nom des équipes, hôte ou adversaire, cela s'explique par leur notoriété et les joueurs présents dans cette équipe. Une équipe qui perd tout le temps, aura forcément un Estimate négatif. C'est le cas pour l'équipe d'Orlando, lorsqu'elle est hôte, qui est dernière au classement et qui a un Estimate égal à -2,57, avec 3 étoiles, donc très significatif. Cela signifie que, juste le fait d'être l'équipe d'Orlando et de jouer à domicile augmente fortement les chances de perdre.

On remarque aussi qu'il n'y a que 3 ratios qui sont réellement significatifs, cela s'explique car les ratios sont corrélés au nom des équipes puisqu'elles vont ensemble.

Nous avons le ratio H_Tpct, qui rappelons-le, représente le taux de victoire global de l'équipe Hôte. Celui-ci est étonnant car il a un Estimate très négatif (-6,6). Cela signifie que plus l'équipe Hôte remporte des matchs, plus elle aura un ratio de victoire élevé sur le long terme et plus elle aura de chances de perdre. On peut expliquer cela par la fatigue, des blessures, un excès de confiance, la rotation des équipes ou tout simplement si elles sont déjà qualifiées en playoff.

Nous avons également deux ratios relatifs aux taux de victoires des adversaires. Le premier, A_ARpct, représente le taux de victoire cumulé jusqu'à présent en tant qu'adversaire, c'est celui avec l'Estimate le plus élevé et a trois étoiles. Il signifie que lorsque l'équipe adverse a un taux de victoire très élevé, elle a plus de chances de perdre, et donc nous avons plus de chances de gagner, cela peut s'expliquer de la même manière que le H_Tpct.

Mais le dernier ratio est sur le court terme, contrairement aux précédents, A_ARLast10, il représente le taux de victoire sur les dix derniers matchs de l'adversaire en tant qu'adversaire. Son Estimate est positif, mais il n'a que deux étoiles, il est donc moins significatif que les autres variables que l'on a prises. Plus l'équipe adverse a un haut taux de victoires sur ce type de matchs, plus ils ont de chances de gagner, et donc l'hôte de perdre.

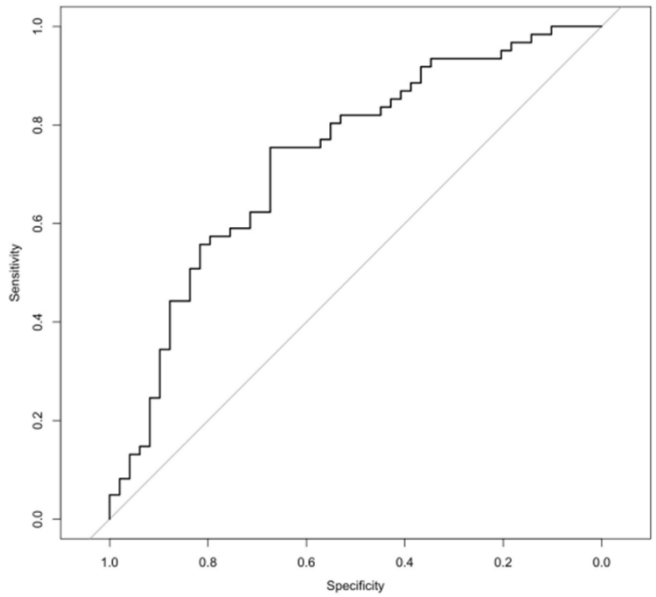
On peut expliquer cela par plusieurs choses, notamment la volonté de faire une série de victoires par exemple, ou encore le fait que ce soit à court terme prend en compte la forme des joueurs et donc cela suit une certaine continuité.

On peut également interpréter le probalogit, qui représente les chances de victoires pour chacun des matchs de la base test, selon la prédiction du modèle :

Par exemple, pour le match 928, l'équipe hôte a 75,1% de chances de gagner. Ce match est une confrontation entre l'équipe CLE (Cleveland) et DET (Détroit), qui s'est soldé par une victoire de CLE, comme le modèle la prédit.

Bien évidemment, ce modèle n'est pas parfait, pour le match 555, opposant BKN (Brooklyn) contre NY (New York), BKN avait 37,6% de chances de gagner et a finalement gagné.

```
> probalogit
      672       171       788       601        50       641
0.6664310 0.2907774 0.4886300 0.1186947 0.6761485 0.5223292
       88       534       701       923       694       373
0.3445720 0.4974624 0.1455300 0.4457388 0.2056431 0.5397015
       93       378       361       252       928       529
0.6693243 0.8520231 0.3945421 0.4921999 0.7510918 0.5951706
      503       216       329       543       158       711
0.8255867 0.6658766 0.5470198 0.2460717 0.4965421 0.5063862
       76       109       247       122       488       720
0.3997953 0.7086167 0.6967985 0.6191607 0.7717548 0.1551236
      635       468       550       535       445       446
0.4888959 0.3416439 0.6383704 0.2825815 0.8787640 0.8913500
      555       320       828       759         4       516
0.3758734 0.8867548 0.9290896 0.4665311 0.5290848 0.7339876
      577       436       327       858       321       579
0.8183289 0.8058722 0.5092063 0.7470241 0.6274582 0.4075920
       84       752       768       447       162       627
0.8949636 0.6436784 0.7395212 0.8669360 0.7459902 0.8747982
      113       723       892       840       202       194
0.8741088 0.5879868 0.9117708 0.4570497 0.2514223 0.2550818
      565       117       317       556        32       418
0.5323764 0.8752831 0.8510422 0.4721940 0.9571980 0.5617072
      649       160       911       420       776       785
0.5575736 0.3572725 0.6098419 0.6052386 0.6704284 0.5785791
      416       390       669       916       729       310
0.5417203 0.2879194 0.6781683 0.6769727 0.3329599 0.3964832
      348       493       866       861       862       914
0.2784295 0.4394763 0.3762371 0.3039707 0.2615448 0.5345451
      906       127       901        49       212       810
0.8084154 0.3291226 0.6549239 0.7545386 0.1556277 0.7527885
       69       269       753       289       722       353
0.5130616 0.4922014 0.5968125 0.4698761 0.4807060 0.6296846
      764       871       404       702       244       181
0.6839783 0.6209218 0.7341129 0.1226221 0.5473389 0.4291211
      441       615       648       813       572       155
0.4165897 0.4277840 0.3498103 0.6607078 0.7856581 0.6229370
      480
6483415
```



On obtient au final, pour le modèle logit, une AUC de 0.7327, ce qui signifie que le modèle est plutôt bon, sans être toutefois parfait. Il prédit également les résultats dans le bon sens car il est supérieur à 0,5. Il faut désormais le comparer avec les autres modèles afin de savoir lequel est le meilleur.

```
> roc(response=nbatest$H_Outcome,predictor=probalogit)
Setting levels: control = 0, case = 1
Setting direction: controls < cases

Call:
roc.default(response = nbatest$H_Outcome, predictor = probalogit)

Data: probalogit in 49 controls (nbatest$H_Outcome 0) < 61 cases (nbatest$H_Outcome 1).
Area under the curve: 0.7327
```

B) Le modèle ARBRE

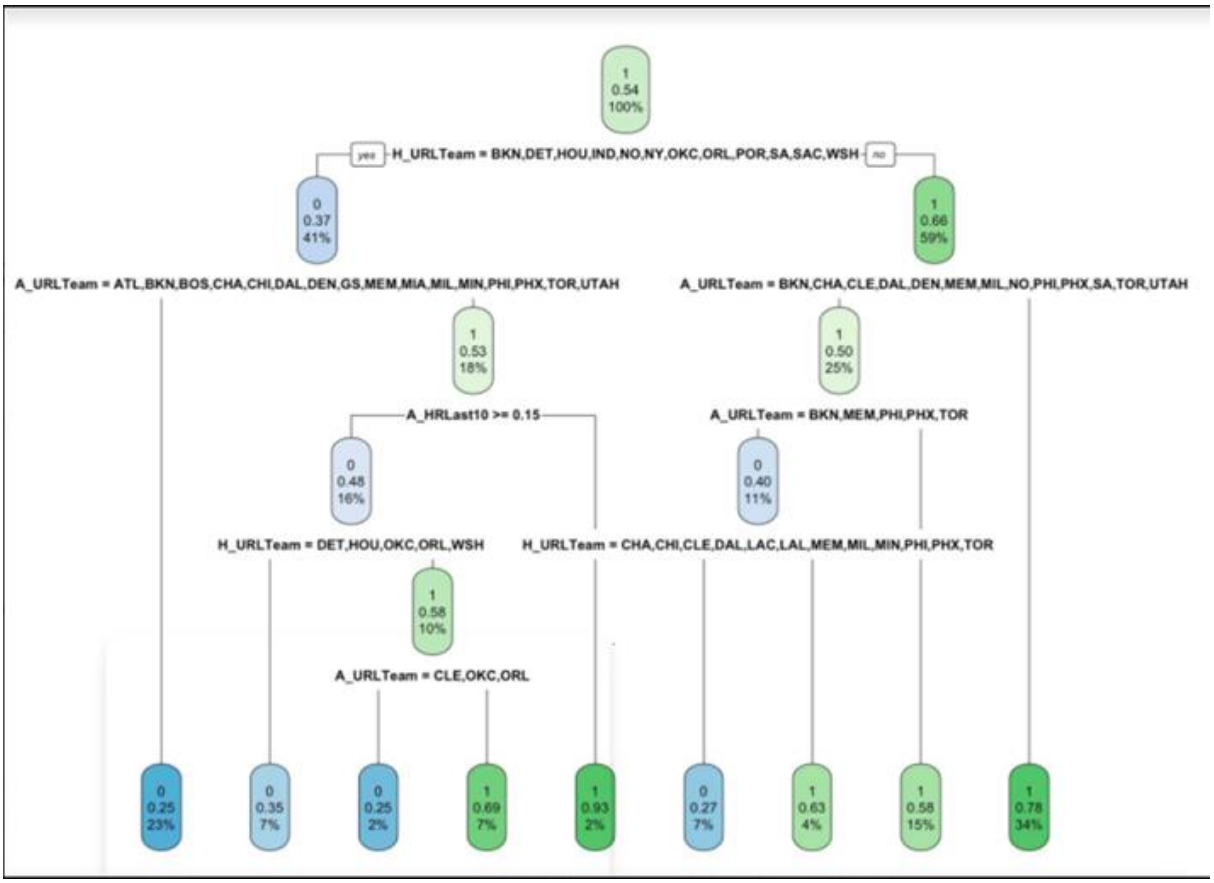
Pour réaliser l'arbre, nous avons utilisé le package "R-part". Nous commençons par prendre l'ensemble de l'échantillon dans la base de données Nba train (composée de 818 matches) et cherchons donc à prédire les résultats des matches (victoires ou défaites).
L'arbre va permettre de calculer l'indice de Gini qui nous donne la répartition d'une variable au sein de la population, ici les victoires de l'équipe hôte sur l'ensemble des matches. On cherche à garder les meilleures variables pour construire notre arbre de décision.

Après avoir élaboré et affiché le graphique de l'arbre. Nous allons à présent, l'élaguer pour éviter un phénomène d'over lifting, c'est-à-dire à partir du moment où un nouvel individu ne colle pas au modèle. L'élagage permet donc de trouver un juste milieu.

```
> printcp(arbre)

Classification tree:
rpart(formula = H_Outcome ~ ., data = nbatrain, met

Variables actually used in tree construction:
[1] A_ARLast10 A_ARpct    A_HRLast10 A_URLTeam  H_I
[6] H_URLTeam

Root node error: 374/818 = 0.45721

n= 818

        CP nsplit rel error  xerror     xstd
1 0.229947      0   1.00000 1.00000 0.038096
2 0.024064      1   0.77005 0.89305 0.037588
3 0.021390      3   0.72193 0.83422 0.037145
4 0.018717      8   0.60963 0.83690 0.037168
5 0.010695     10   0.57219 0.83690 0.037168
6 0.010000     12   0.55080 0.83422 0.037145
```

Pour ce faire, via la fonction printcp(arbre), on cherche le CP (complexity parameter) qui va être associé au taux d'erreur relatif le plus faible. Nous associons le CP à la taille de l'arbre. Ainsi, la 3ème ligne est celle qui minimise le "xerror", nous allons donc prendre son CP = 0,021390.

Par ailleurs, on remarque que les équipes hôtes présentes en ligne 1 ont 25% de chance de gagner face aux adversaires présents en ligne 2 et représentent 23% de la population. Ainsi, les équipes suivantes ont 25% de chance de gagner à domicile face à leur adversaire : Brooklyn (BKN) contre Atlanta (ATL), Détroit (DET) face à Boston (BOS), Houston (HOU) face à Chicago (CHI), etc…



D'autre part, les équipes hôtes non présentes en ligne 1 ont 78% de chance de gagner face à un adversaire ne faisant pas partie de la ligne 2 et représentent 34% de la population.

Ainsi, les équipes suivantes ont 78% de chance de gagner à domicile face à leur adversaire : Atlanta (ATL) face à Portland (POR), Philadelphie (PHI) face à Washington (WSH), Memphis (MEM) face à Sacramento (SAC)…

Enfin, les équipes hôtes présentes en ligne 1 ont 93% de chance de gagner face à un adversaire ne faisant pas partie de la ligne 2 (gauche) ayant un taux de victoire sur les 10 derniers matchs à domicile inférieur à 15%. Ces équipes représentent 2% de la population. L'équipe adverse étant dans une dynamique négative (faible taux de victoires) il est assez logique que les chances de victoire de l'équipe hôte soient élevées.

On obtient au final une AUC de 0.5953 pour le modèle de l'arbre, ce qui signifie que le modèle est plutôt mauvais. Il prédit toutefois les résultats dans le bon sens car il est supérieur à 0,5. Il faut désormais le comparer avec les autres modèles afin de savoir lequel est le meilleur, mais il est très nettement inférieur au modèle logit.

```
> roc(predictor=arbreNBA[,2],response=nbatest$H_Outcome)
Setting levels: control = 0, case = 1
Setting direction: controls < cases

Call:
roc.default(response = nbatest$H_Outcome, predictor = arbreNBA[,     2])

Data: arbreNBA[, 2] in 49 controls (nbatest$H_Outcome 0) < 61 cases (nbatest$H_Outcome 1).
Area under the curve: 0.5953
```

C) Le modèle FORÊT

Nous avons ensuite réalisé un nouvel algorithme regroupant plusieurs arbres, on parle de forêt. L'avantage de ce modèle est que la forêt va appeler plusieurs arbres. On aura une moyenne de ces arbres. Au final, le résultat sera bien plus fiable par rapport à un arbre qui peut plus facilement se tromper. En contrepartie, la forêt est moins parlante visuellement car elle ne permet pas d'afficher un arbre, mais plutôt les variables les plus significatives.

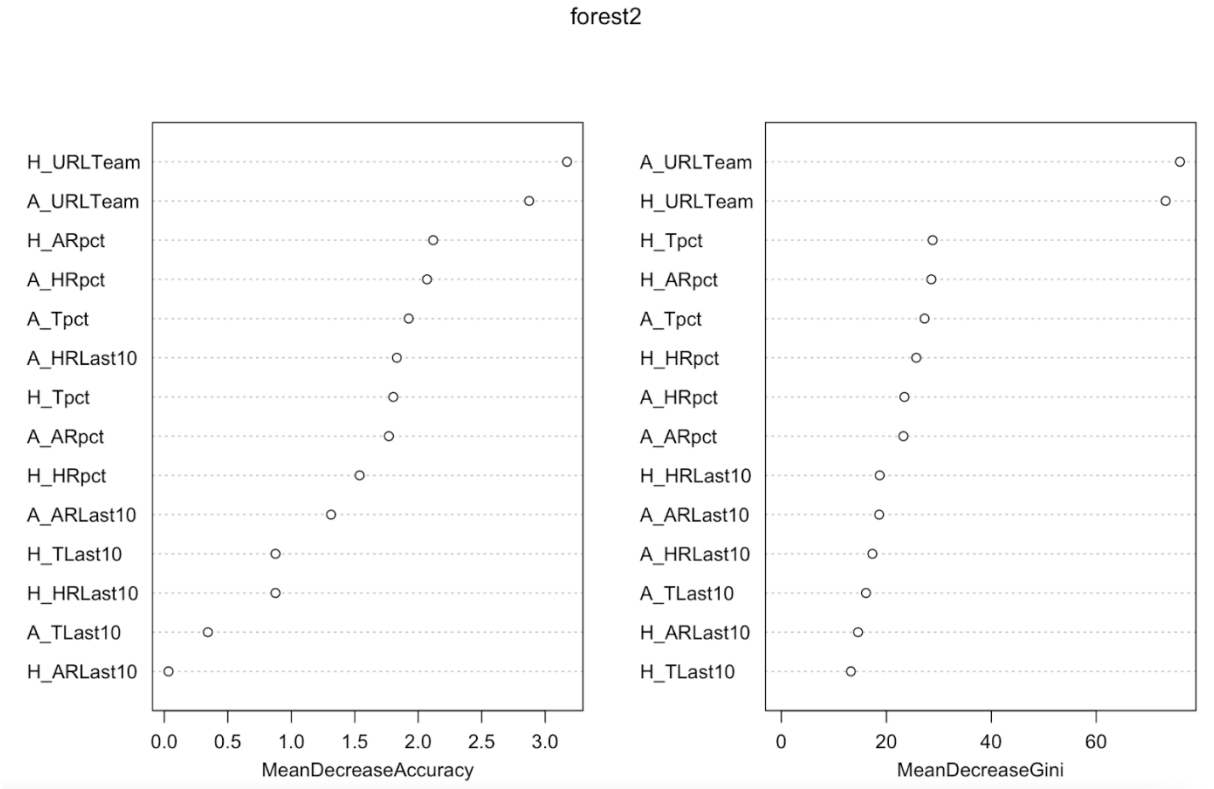
Pour créer la forêt, nous utilisons le package "randomForest" en indiquant tout d'abord, le nombre d'arbres à utiliser, puis le nombre de variables que nous prenons en compte. Après, il convient de trouver le nombre d'arbres et de variables, qui permettent de minimiser l'erreur de la forêt.

Ainsi, à l'aide d'un graphique, nous avons cherché dans un premier temps, le nombre d'arbres qui minimise le taux d'erreur. Puis, après trouvé le nombre de variables qui minimise le taux d'erreur. Finalement, nous trouvons comme solution un nombre de 80 arbres pour 3 attributs.

On obtient également une estimation du taux d'erreur ou mauvaise prédiction du résultat du match à 36,31%. Ce taux, qui peut paraître élevé, résulte du fait que les variables interagissent entre elles, prédire un match de basketball est et restera complexe.

```
> forest=randomForest(H_Outcome~.,data=nbatrain,mtry=3,ntree=80);forest

Call:
 randomForest(formula = H_Outcome ~ ., data = nbatrain, mtry = 3,      ntree
               Type of random forest: classification
                     Number of trees: 80
No. of variables tried at each split: 3

        OOB estimate of  error rate: 36.31%
Confusion matrix:
    0   1 class.error
0 169 205   0.5481283
1  92 352   0.2072072
```

De plus, on explique le mieux les victoires ou défaites des équipes NBA par l'équipe host ou adversaire, ceux sont les critères avec les indices de Gini les plus élevés. Ici, les variables à prendre en considération, vont être surtout l'équipe à domicile, puis l'équipe à l'extérieur. Après nous retrouvons les différents ratios. Ce modèle ne permet pas de savoir si l'influence est positive ou négative vis-à-vis de la victoire de l'hôte.



On obtient au final, pour le modèle de la forêt une AUC de 0.6209, ce qui signifie que le modèle est moyennement bon. Il prédit également les résultats dans le bon sens car il est supérieur à 0,5. Il faut désormais le comparer avec les autres modèles afin de savoir lequel est le meilleur, mais tout comme l'arbre, il est nettement inférieur au modèle du logit, et supérieur à celui de l'arbre.

```
> roc(predictor=forestprev[,2],response=nbatest$H_Outcome)
Setting levels: control = 0, case = 1
Setting direction: controls < cases

Call:
roc.default(response = nbatest$H_Outcome, predictor = forestprev[,     2])

Data: forestprev[, 2] in 49 controls (nbatest$H_Outcome 0) < 61 cases (nbatest$H_Outcome 1).
Area under the curve: 0.6209
```

D) Le modèle NAÏVE BAYESIEN

A présent, nous allons vous parler de la méthode du naïf bayésien. C'est une méthode utilisée à la base pour détecter les mails et les avis frauduleux. Cependant, cette méthode peut aussi être utilisée pour d'autres choses comme la prédiction des victoires de l'équipe hôte en NBA comme dans notre cas. Ce modèle est très simple mais très faux. Les mails frauduleux se basent donc sur une fausse probabilité pour une combinaison de mot avec le correcteur de Laplace.

Ainsi, le même principe va être appliqué à notre base NBA, en prenant en compte les différentes variables. En clair, la probabilité qu'une combinaison de variables apparaisse, va être associée par l'algorithme soit à une victoire ou soit à une défaite, toujours avec le correcteur de Laplace.

Dans R, on utilise la library "e1071" qui permet d'importer la fonction bayes. Suite à ça, nous implémentons l'algorithme du naïf bayésien, avec la fonction naiveBayes. Enfin, nous indiquons la base utilisée qui est évidemment celle de la nba, en fonction de H_Outcome. Nous rajoutons en plus, ici le paramètre de Laplace avec sa valeur, en l'occurrence nous avons choisi Laplace = 1, comme dans le cours. Cependant, nous aurions pu très bien utiliser Laplace = 2.

Nous avons commencé par essayer d'imager nos résultats à travers un boxplot (boite à moustache) mais ce dernier n'a pas pu être généré pour des raisons que nous ignorons, et ce malgré plusieurs essais.

```
> boxplot(probabayes[,1]~nbatest$H_Outcome)
Erreur dans plot.new() : figure margins too large
```

```
> probabayes
                  0           1
[1,] 8.775629e-02 0.9122437126
[2,] 8.360763e-01 0.1639236806
[3,] 9.541220e-01 0.0458780145
[4,] 9.935058e-01 0.0064942044
[5,] 4.681470e-01 0.5318529621
[6,] 6.924594e-01 0.3075406017
[7,] 3.270303e-01 0.6729697315
```

Nous avons également 110 matchs. Pour le match 1 de la base test : l'équipe hôte a 8,78% de chance de perdre et 91,2% de chances de gagner, tout comme pour le logit. Cependant, on parle cette fois-ci de l'équipe en première position dans la base test. Nous pouvons donc passer au plus important : l'AUC.

On obtient au final pour le modèle Naïve Bayésien une AUC de 0.717, ce qui signifie que le modèle est, comme le logit plutôt bon, sans être toutefois parfait. Il prédit également les résultats dans le bon sens car il est supérieur à 0,5. Il est, tout comme le logit encore une fois, meilleur que l'arbre et la forêt.

```
> auc(predictor=probabayes[,2],response=nbatest$H_Outcome)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
Area under the curve: 0.717
```

E) Le modèle KNN

A présent nous allons présenter l'algorithme des k plus proches voisins, en anglais k-NN (« k Nearest Neighbors"). Cet algorithme va chercher les K personnes d'une base de données qui se ressemblent le plus pour ensuite prévoir une variable de sortie.

Dans notre cas, il aura pour objectif de prévoir les victoires de l'équipe hôte en NBA. Le k dans la formule, représente le nombre de voisins que nous prenons pour la prévision.

Pour appliquer cette méthode dans R avec la library “class”, il est nécessaire de transformer toutes les variables en numérique. Il fallait utiliser la fonction : “model.matrix”, et cela va nous donner “nbanumtrain” qui est composée de 818 matchs et “nbanumtest” des 110 autres matchs.

L’étape suivante va être de les normaliser, pour cela il est nécessaire de calculer la moyenne et l’écart type de “train” afin de pouvoir les appliquer sur train et test. Ensuite, on utilise le modèle prédictif KNN, avec je le rappelle k = x, le nombre de voisins. Il faut trouver celui qui va minimiser l’erreur dans la table. Tout d’abord pour prendre k = x, nous prenons la racine carré du nombre de lignes dans le train qui est de (818), ce qui nous donne k=28, puis nous cherchons autour de cette valeur.

En cherchant autour de cette valeur, on trouve un k permettant d’avoir un meilleur modèle que les autres, k = 39.

```
> table(nbanumtest$H_Outcome,voisin)
   voisin
     0  1
  0 30 19
  1 11 50
```

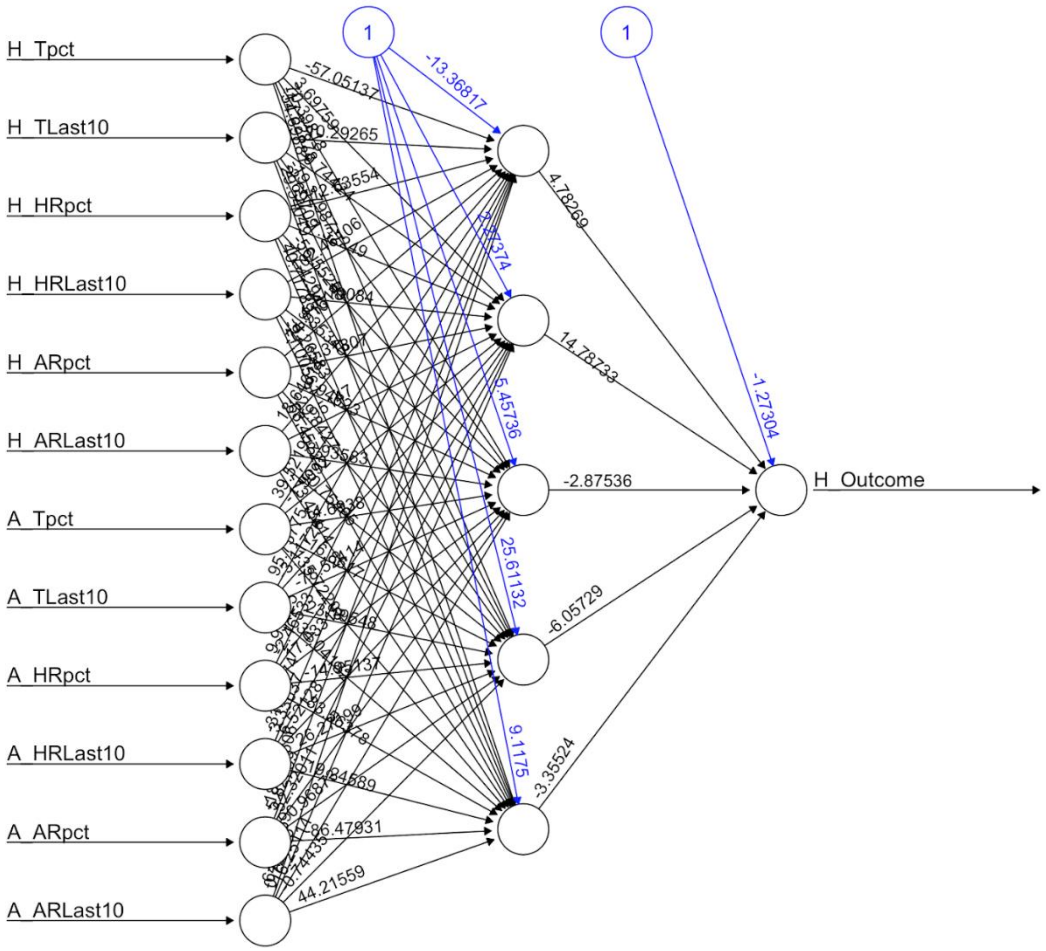
On observe 19 + 11 erreurs sur 110. Ce qui nous permet de calculer le pourcentage d'erreurs. On a donc un pourcentage d'erreurs qui est égal à 30/110 = 27,27% d’erreurs, ce qui est plus faible de 9,04 points de pourcentage que le modèle de la forêt.

Pour le modèle KNN, on obtient au final une AUC de 0.716, quasiment égal au modèle naïf Bayésien, ce qui signifie que le modèle est plutôt bon, sans être toutefois parfait encore une fois. Il prédit également les résultats dans le bon sens car son AUC est supérieur à 0,5.

```
> probaknn=as.numeric(probaknn)
> roc(response=nbanumtest$H_Outcome,predictor=probaknn)
Setting levels: control = 0, case = 1
Setting direction: controls < cases

Call:
roc.default(response = nbanumtest$H_Outcome, predictor = probaknn)

Data: probaknn in 49 controls (nbanumtest$H_Outcome 0) < 61 cases (nbanumtest$H_Outcome
 1).
Area under the curve: 0.716
```

F) Le modèle RESEAU DE NEURONES

Pour finir, nous avons voulu tester un algorithme supplémentaire pour essayer de dépasser le modèle du logit, qui est le réseau de neurones. Comme son nom l'indique, c'est un algorithme qui se base sur le fonctionnement des neurones se trouvant dans le corps humain. Les neurones sont connectés entre eux et vont se transmettre de l'information sous impulsion électrique. C'est ce principe que nous retrouvons dans le réseau de neurones.
D'ailleurs, le réseau est très complexe. En effet, à chaque utilisation, il n'affichera pas les mêmes informations. Chaque neurone ne reçoit pas l'information de la même manière à chaque fois.

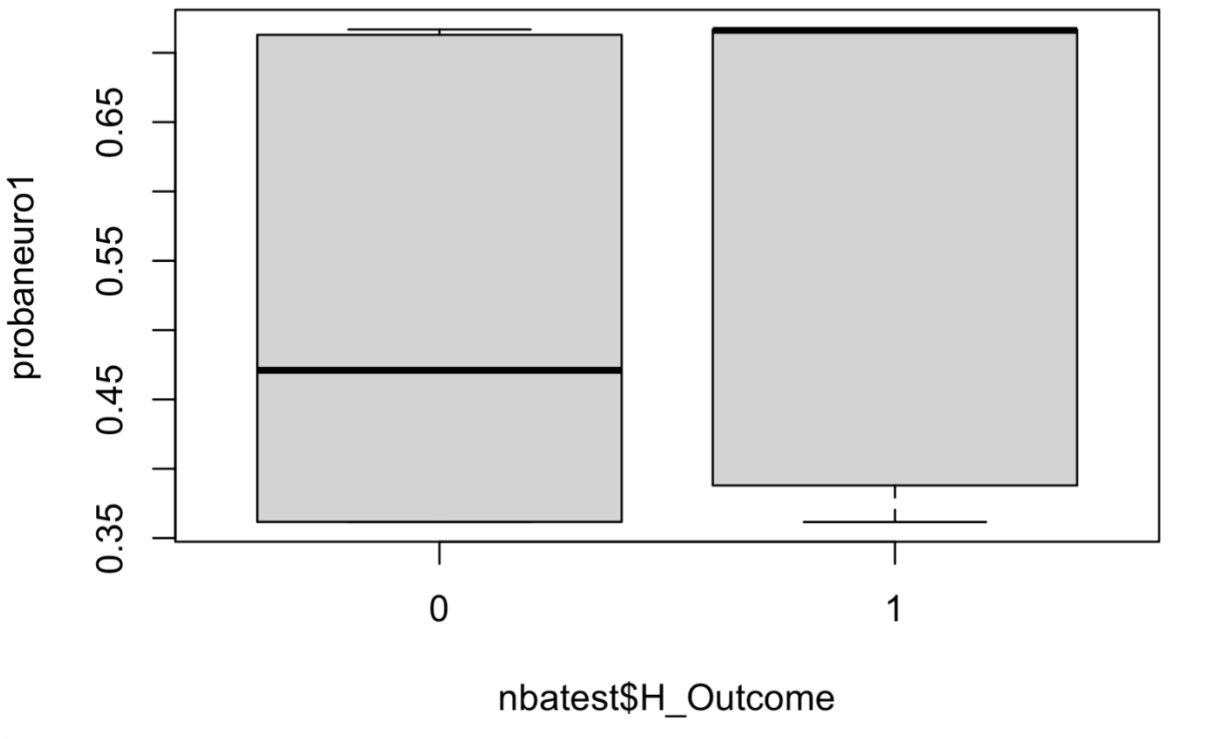


Pour le code R, il fallait importer la library "neuralnet" permettant d'importer la fonction : neuralnet, nécessaire à cet algorithme. Ensuite, nous avons élaboré le réseau, en indiquant différents paramètres, comme le nombre de variables que nous prenons dans ce qu'on appelle la couche cachée et en créant plusieurs couches cachées. Par la suite, nous

pouvons indiquer le nombre d'apprentissages, dans la même logique que les forêts, plus on va faire d'apprentissages et plus on aura un résultat fiable. Et on indique, linear.output = F, pour dire que nous ne voulions pas que l'entrée soit proportionnelle à la sortie.

Nous avons gardé seulement les ratios, car il était plus facile d'afficher le réseau de neurones sans les différentes équipes. Dans le graphique, le réseau présente plusieurs familles. Tout d'abord les inputs se retrouvent à gauche, Ici, il s'agit des différents ratios puis la famille des couches cachées et enfin les outputs à droite, en l'occurrence ici la variable que l'on cherche à prédire : H Outcome. Comme expliqué précédemment, tous les neurones sont connectés entre eux, c'est pour cela que nous voyons pas mal de lignes noires. Chaque ligne va avoir une valeur, que nous pouvons interpréter de la même façon que dans le logit2.

Nous pouvons voir rapidement la boite à moustache, le probaneuro1 en fonction du nbatest, la probabilité de perdre est autour de 0,45 et la probabilité de gagner, autour de 0,70, ce qui est plutôt pas mal.



Ensuite, nous pouvons afficher une table permettant de savoir le nombre d'erreurs dans la diagonale. Pour cela, nous considérons que lorsque la probabilité du probaneuro est supérieur à 0,5 alors la valeur dans le “previsionneuro” va valoir 1, et dans le cas contraire 0. Voici donc la table :

```
> table(previsioneuro1,nbatest$H_Outcome)

previsioneuro1  0  1
         loose 25 18
         win   24 43
```

Il y a donc 18+24, donc 42 erreurs dans la diagonale, soit environ 38% d’erreurs.

Donc comme tous les autres algorithmes, on va essayer de trouver le meilleur AUC, aire sous la courbe possible. Pour ce faire, nous allons jouer sur les paramètres expliqués précédemment.

Tout d’abord, et en général, le nombre d’apprentissage va augmenter naturellement la valeur de l’AUC vers 1. Mais lorsque nous augmentons le nombre d’apprentissage, l’AUC ne semblait pas augmenter. Nous avons donc attendu le temps que R fasse le nombre d'apprentissage.

Après, nous avons joué sur la famille des couches cachées. On a tout d’abord changé le nombre de couches, en en mettant 3 puis 2 par exemple. Mais là, l’AUC semblait diminuer à chaque fois, donc nous n’étions pas satisfaits. Finalement, nous n'avons gardé qu’une seule couche, et même qu’une seule variable dans la couche car c’est avec ce paramètre que nous trouvions l’AUC le plus élevé, à savoir : 0,6671.

```
> auc(predictor=probaneuro1,response=nbanumtest$H_Outcome)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
Area under the curve: 0.6671
```

En conclusion, pour construire notre modèle prédictif, nous allons garder la fonction logit qui présentait le meilleur AUC : 0,7327, bien que d'autres modèles ont été très proches de cette AUC. Nous allons retrouver dès à présent ce modèle prédictif dans un robot, où lorsque nous rentrons deux équipes, nous avons une probabilité associée à la victoire de l'équipe hôte.

II. Le fameux robot

On va à présent vous présenter le robot, qu'on a pu construire et qui nous permet d'insérer à la fois l'équipe à domicile et l'équipe extérieur afin de calculer la probabilité de gain du match pour l'hôte. Nous pouvons facilement en déduire la probabilité de défaite de l'adversaire, en faisant 1-x ; x étant la probabilité de gain pour l'hôte.

Pour construire le robot, on a pris la fonction A) Le modèle LOGIT comme expliqué précédemment dans le dossier, ce qui nous a permis de créer une fonction que l'on a appelée « Capitaine ». Pour ce faire, nous avons créé deux nouvelles bases de données, une première avec le dernier match de chacune des équipes, en tant qu'hôte, et une seconde avec également le dernier match de chacune des équipes, mais en tant qu'adversaire cette fois-ci. Ces bases de données nous ont permis de récupérer les ratios et les associer à leurs équipes. La fonction s'écrit comme suit :

```
Capitaine=function(){
  print("Bonjour je vais prédire le résultat de votre match")
  H_URLTeam=as.factor(readline("Équipe à domicile :"));
  A_URLTeam=as.factor(readline("Équipe à l'extérieur :"));
  ratioH=NBAH[H_URLTeam==H_URLTeam,c(5:10)];
  ratioA=NBAA[A_URLTeam==A_URLTeam,c(11:16)];
  basket=data.frame(H_URLTeam=H_URLTeam,A_URLTeam=A_URLTeam,ratioH,ratioA);
  print("proba de gagner:");
  pr=mean(predict(logit2,newdata=basket,type="response"));
  print(pr)
}
```

La fonction print permet d'afficher à l'utilisateur un texte. Les fonctions readline permettent à l'utilisateur de renseigner le nom des équipes du match qu'il souhaite prévoir.

Pour exécuter ce robot, il suffit d'écrire le nom de notre fonction suivi de parenthèses, dans notre cas, « Capitaine() ». Le robot demande alors de saisir le nom de l'équipe à domicile, puis celle à l'extérieur, ensuite il affichera la probabilité de gain de l'équipe à domicile. Voici quelques exemples :

```
> Capitaine()
[1] "Bonjour je vais prédire le résultat de votre match"
Équipe à domicile :BOS
Équipe à l'extérieur :CLE
[1] "proba de gagner:"
[1] 0.5268213
```

On a mis comme équipe hôte, BOS (Celtics de Boston), et comme adversaire CLE (Cavaliers de Cleveland) . Selon le robot, et donc selon le modèle logit, Boston à 52,68% de chances de gagner contre Cleveland. Le match semble serré.

```
> Capitaine()
[1] "Bonjour je vais prédire le résultat de votre match"
Équipe à domicile :SA
Équipe à l'extérieur :HOU
[1] "proba de gagner:"
[1] 0.8055146
```

Dans le second cas, on a mis comme équipe hôte SA (Spurs de San Antonio) et comme adversaire HOU (Rockets de Houston). Selon le modèle encore une fois, San Antonio à 80,55% de chance de gagner contre Houston.

Conclusion :

Pour conclure, oui le modèle n'est pas parfait, l'AUC est de 0,7327. Il faut toutefois prendre en compte qu'il est naturellement difficile de prédire le résultat d'un match de Basket. Il y a néanmoins d'autres algorithmes dont nous n'avons ni parlé ni traité en cours et qui auraient pu nous donner un modèle plus efficace. Nous avons tout de même ajouté l'algorithme de réseau de neurones parce qu'il nous intéressait personnellement. Il faut également prendre en compte que certains éléments qui ne sont pas dans la base de données influencent grandement le résultat d'un match, tels que la fatigue des joueurs, relatif au temps de repos entre deux matchs, l'humeur, les blessures ou encore la présence ou non de supporters. Cependant, il est toujours nécessaire d'avoir un principe de parcimonie, un modèle compliqué mais pas trop non plus.

Il y a également quelque chose que nous souhaitons préciser. Ce modèle n'est pas destiné à faire gagner des paris sportifs relatifs à la NBA. Les paris sportifs peuvent réellement être dangereux, notamment pour les mineurs, d'autant plus que les applications de paris sportifs ont une communication et un marketing ultra agressifs. Soyez comme Kylian Mbappé, bien qu'il soit joueur de foot et pas de basket, dîtes non aux paris sportifs !

Merci pour votre attention, en espérant que ceci a suscité votre curiosité.

Annexes

Lexique des variables

H_DATE	Date du Match
H_URLTeam	Nom Equipe Host
A_URLTeam	Nom Equipe Adversaire
H_Tpct	Host : Taux de victoires cumulé jusqu'à présent
H_TLast10	Host : Taux de victoires sur les 10 derniers matchs
H_HRpct	Host : Taux de victoires cumulé jusqu'à présent en tant qu'Hôte
H_HRLast10	Host : Taux de victoires sur les 10 derniers matchs en tant qu'Hôte
H_ARpct	Host : Taux de victoire cumulé jusqu'à présent en tant qu'Adv
H_ARLast10	Host : Taux de victoires sur les 10 derniers matchs en tant qu'Adv
A_Tpct	Adversaire : Taux de victoires cumulé jusqu'à présent
A_TLast10	Adversaire : Taux de victoires sur les 10 derniers matchs
A_HRpct	Adversaire : Taux de victoire cumulé jusqu'à présent en tant qu'Hôte
A_HRLast10	Adversaire : Taux de victoires sur les 10 derniers en tant qu'Hôte
A_ARpct	Adversaire : Taux de victoire cumulé jusqu'à présent en tant qu'Adv
A_ARLast10	Adversaire : Taux de victoires sur les 10 derniers matchs en tant qu'Adv
H_Outcome	1 = Victoire de l'Hôte ; 0 = Défaite de l'Hôte

Acronyme et Nom des équipes

ATL	Hawks d'Atlanta	**HOU**	Rockets de Houston	**OKC**	Thunder d'Oklahoma City
BKN	Nets de Brooklyn	**IND**	Pacers de l'Indiana	**ORL**	Magic d'Orlando
BOS	Celtics de Boston	**LAC**	Clippers de Los Angeles	**PHI**	76ers de Philadelphie
CHA	Hornets de Charlotte	**LAL**	Lakers de Los Angeles	**PHX**	Suns de Phoenix
CHI	Bulls de Chicago	**MEM**	Grizzlies de Memphis	**POR**	Trail Blazers de Portland
CLE	Cavaliers de Cleveland	**MIA**	Heat de Miami	**SA**	Spurs de San Antonio
DAL	Mavericks de Dallas	**MIL**	Bucks de Milwaukee	**SAC**	Kings de Sacramento
DEN	Nuggets de Denver	**MIN**	Timberwolves du Minnesota	**TOR**	Raptors de Toronto
DET	Detroit Pistons	**NO**	Pelicans de la Nouvelle-Orléans	**UTAH**	Jazz de l'Utah
GS	Warriors de Golden State	**NY**	Knicks de New York	**WSH**	Wizards de Washington

Etats lorsque ce n'est pas précisé dans le nom :

CHA : Caroline du Nord

CLE : Ohio

GS : San Francisco

MIL : Wisconsin

Listing R :

Logit

```
nrow(nba)
set.seed(216)
index=sample(928,110)
str(nba)
nbatest=nba[index,-1]
nbatrain=nba[-index,-1]
str(nbatest)
logit=glm(H_Outcome~.,data=nbatrain,family="binomial")
logit
summary(logit)
logit=step(logit,direction="both")
summary(logit2)
str(nba)
probalogit=predict(logit2,newdata=nbatest,type="response")
probalogit
nba[32,]
boxplot(probalogit~nbatest$H_Outcome)
library(pROC)
roc(response=nbatest$H_Outcome,predictor=probalogit)
plot(roc(response=nbatest$H_Outcome,predictor=probalogit))
```

Arbre

```
arbre=rpart(H_Outcome~.,data=nbatrain,method="class")

printcp(arbre)

arbre=rpart(H_Outcome~.,data=nbatrain,method="class",cp=0.021390 )

rpart.plot(arbre)

arbre=rpart(H_Outcome~.,data=nbatrain,method="class",cp=0.021390)

rpart.plot(arbre)

arbreNBA=predict(arbre,newdata=nbatest,method="prob")

roc(predictor=arbreNBA[,2],response=nbatest$H_Outcome)
```

Forêt

```
forest=randomForest(H_Outcome~.,data=nbatrain,mtry=3,ntree=100)

forest

plot(forest$err.rate[,1])

forest=randomForest(H_Outcome~.,data=nbatrain,mtry=3,ntree=80)

forest=randomForest(H_Outcome~.,data=nbatrain,mtry=3,ntree=80);forest

forest=randomForest(H_Outcome~.,data=nbatrain,mtry=3,ntree=80);forest

forest=randomForest(H_Outcome~.,data=nbatrain,mtry=3,ntree=80);forest

forest=randomForest(H_Outcome~.,data=nbatrain,mtry=3,ntree=80);forest

forest=randomForest(H_Outcome~.,data=nbatrain,mtry=3,ntree=80);forest

forest=randomForest(H_Outcome~.,data=nbatrain,mtry=3,ntree=80);forest

forest=randomForest(H_Outcome~.,data=nbatrain,mtry=3,ntree=80);forest

forest=randomForest(H_Outcome~.,data=nbatrain,mtry=3,ntree=80);forest

forest=randomForest(H_Outcome~.,data=nbatrain,mtry=3,ntree=80);forest

forest=randomForest(H_Outcome~.,data=nbatrain,mtry=3,ntree=80);forest

forestprev=predict(forest,newdata=nbatest,type="prob")

roc(predictor=forestprev[,2],response=nbatest$H_Outcome)

forest2=randomForest(H_Outcome~.,data=nbatrain,mtry=3,ntree=80,importance=T)

varImpPlot(foret2)

varImpPlot(forest2)
```

Baysien

```
library(e1071)
nba2=nba
str(nba2)
nba2=nba2[,-1]
str(nba2)
bayes=naiveBayes(H_Outcome~.-H_DATE,data=nba,laplace=1)
probabayes=predict(bayes,newdata=nbatest,type="raw")
library(pROC)
auc(predictor=probabayes[,2],response=nbatest$H_Outcome)
probabayes
boxplot(probabayes[,1]~nbatest$H_Outcome)
```

KNN

```
library(class)
nbanum=nba
nbanum$H_Outcome=as.numeric(as.character(nbanum$H_Outcome))
str(nbanum)
str(nba)
toto=data.frame(model.matrix(~H_URLTeam+A_URLTeam,data=nbanum))
str(toto)
ncol(nba)
nbanum=cbind(toto[,-1],nbanum[,c(4:16)])
str(nbanum)
set.seed(216)
index=sample(928,110)
nbanumtest=nbanum[index,]
nbanumtrain=nbanum[-index,]
m=apply(nbanumtrain,2,FUN=mean)
s=apply(nbanumtrain,2,FUN=sd)
ncol(nbanum)
nnbanumtest=scale(nbanumtest[,-71],center=m[-71],scale=s[-71])
sqrt(928)
nrow(nbanumtrain)
sqrt(818)
nnbanumtrain=scale(nbanumtrain[,-71])
voisin=knn(train=nnbanumtrain,test=nnbanumtest,cl=factor(nbanumtrain$H_Outcome),k=38)
table(nbanumtest$H_Outcome,voisin)
voisin=knn(train=nnbanumtrain,test=nnbanumtest,cl=factor(nbanumtrain$H_Outcome),k=39)
table(nbanumtest$H_Outcome,voisin)
library(pROC)
probaknn=voisin
probaknn=as.numeric(probaknn)
roc(response=nbanumtest$H_Outcome,predictor=probaknn)
```

Réseau de neurones

```
library(neuralnet)

titi=glm(H_Outcome~H_Tpct+H_TLast10+H_HRpct+H_HRLast10+H_ARpct+H_ARLast10+A_Tpct+A_TLast10+A_HRpct+A_HRLast10+A_ARpct+A_ARLast10,data=nbanumtrain)

tuto=formula(titi)

tuto

set.seed(216)

neuro1=neuralnet(tuto,data=nbanumtrain,hidden=1,linear.output=F,rep=1)

probaneuro1=compute(neuro1,covariate=nbanumtest[,-71])$net.result

library(pROC)

auc(predictor=probaneuro1,response=nbanumtest$H_Outcome)

previsioneuro1=ifelse(probaneuro1>=0.5,"win","loose")

table(previsioneuro1,nbatest$H_Outcome)
```

Robot

```
Capitaine=function(){
  print("Bonjour je vais prédire le résultat de votre match")
  H_URLTeam=as.factor(readline("Équipe à domicile :"));
  A_URLTeam=as.factor(readline("Équipe à l'extérieur :"));
  ratioH=NBAH[H_URLTeam==H_URLTeam,c(5:10)];
  ratioA=NBAA[A_URLTeam==A_URLTeam,c(11:16)];
  basket=data.frame(H_URLTeam=H_URLTeam,A_URLTeam=A_URLTeam,ratioH,ratioA);
  print("proba de gagner:");
  pr=mean(predict(logit2,newdata=basket,type="response"));
  print(pr)
}
```