Projet informatique

Prédiction du MVP NBA

Coni Soret, Nadir Tita

Mai 2020

Table des matières

Introduction

Le trophée de Most Valuable Player (MVP) est une récompense attribuée chaque année par la National Basketball Association (NBA) au meilleur joueur de la saison régulière durant laquelle toutes les équipes effectuent 82 matchs.

Jusqu'à la saison 1979-1980, le MVP est désigné par le vote de joueurs de la ligue. À partir de la saison suivante, les votants sont issus d'un panel de journalistes sportifs et de commentateurs américains et canadiens. Chacun nomme cinq joueurs, classé par ordre de préférence. Une première place vaut dix points ; une deuxième en vaut sept ; une troisième en vaut cinq ; une quatrième en vaut trois et une cinquième en vaut un seul. Depuis 2010, un bulletin représentant le vote des internautes est ajouté au scrutin. Le joueur obtenant le plus grand nombre de points remporte le titre.

Il n'y a pas de critères officiels auxquels les journalistes doivent se référer pour voter, ce qui rend l'étude de prédiction encore plus intéressante. Cependant, des tendances se dégagent très clairement lorsque l'on analyse l'attribution du tant convoité trophée. Le critère universel est qu'il faut gagner : depuis les années 80, Russell Westbrook est le seul joueur qui a été élu MVP avec moins de 50 victoires pour son équipe (47 victoires et 35 défaites) notamment grâce à sa ligne statistique en "triple-double" (plus de 10 points, 10 passes et 10 rebonds en un match). L'objectif de notre projet est le suivant : prédire le MVP chaque saison à partir des statistiques des joueurs, et dans l'idéal, prédire le bon podium.

Nous pouvons voir ce problème comme étant un problème de classification, avec par exemple, une classe pour chaque joueur du podium et une classe pour les joueurs hors podium.

Nous pouvons aussi voir ce problème comme étant un problème de régression. Comme expliqué précédemment, chaque joueur obtient des points lorsque les journalistes votent pour lui, en divisant ce nombre de points par le score maximal qu'un joueur peut avoir (ie tous les journalistes le classent premier, arrivé qu'une seule fois dans l'histoire) chaque joueur obtient alors un score entre 0 et 1 que l'on peut essayer de prédire. Il nous suffira juste de classer ces scores pour obtenir le classement que notre algorithme prédit.

Enfin, nous ajouterons que le modèle de prédiction établi peut aussi nous permettre de prendre part aux débats qu'il y a pu avoir lieu dans le monde de la balle orange. Dans la mesure où l'élection du MVP repose sur des critères personnelles, certains joueurs ont été élu à tort selon toute la communauté, du simple fan aux joueurs. L'exemple le plus connu est celui de Steve Nash en 2005 et 2006 dont les titres de MVP ont été beaucoup discutés. Suite à une documentation sur le sujet, notamment cet article qui synthétise les choix douteux qu'ont pu commettre les journalistes (https ://www.foxsports.com/nba/gallery/michael-jordan-lebron-james-kobe-bryant-biggest-mvp-snubs-of-all-time-040715) nous tenterons de prendre du recul

sur les erreurs commises par notre algorithme : il est possible que les journalistes se soient "trompés"...

I Préparation des données

Programmes associés : NBA.py ; NBA_teams_stats.py

Sans aucun doute, c'est cette partie du projet qui nous a demandé le plus de temps, tant d'un point de vue algorithmique où nous avons du créer pas mal de fonctions annexes pour construire notre base de données, que d'un point de vue réflexif où nous avons du réfléchir de quelle manière nous souhaitions donner du sens à nos données.

Pour réaliser notre projet, nous avons utilisé une base de données regroupant différentes statistiques personnelles par saison pour chacun des joueurs entre 1950 et 2017. De plus nous avions besoin des statistiques des équipes et en particulier de leur bilan, c'est-à-dire, leur classement et leur ratio de victoires. C'est pourquoi, nous avons utilisé le module NBA_api permettant d'accéder à différentes statistiques sur NBA.stats.com. Ainsi nous avons pu construire une base de données contenant les bilans annuels de chacune des équipes sur la période considérée. Ensuite nous avons regroupé les deux bases de données pour attribuer à chacun des joueurs le bilan de son équipe.

I.1 Gestion des joueurs ayant joué dans plusieurs équipes pendant la même saison

Fonctions associées : gere_tot ; delete_tot ; moy_pond dans NBA_teams_stats.py

Parfois, certains joueurs changent d'équipe au cours de la saison. Ainsi ils se retrouvent plusieurs fois dans la base de données. Par soucis de cohérence, nous avons décidé de regrouper les statistiques du joueur en une seule ligne en faisant une moyenne pondérée des bilans de chaque équipe dans lequel il a joué avec comme pondération le nombre de matchs joués dans chaque équipe.

I.2 Ajout des scores MVP dans la base

Fonctions associées : add_all_scores ; add_score dans NBA.py

Ensuite nous avons ajouté deux nouveaux attributs : classé MVP et score MVP. Ces deux attributs sont initialisés à 0 pour tous les joueurs. Puis pour chaque année, nous avons manuellement ajouté le score obtenu par chacun des joueurs classés au MVP dans 'score MVP' et un 1 dans 'classé MVP'. Nous n'avons pas pu trouver une solution automatisée qui serait moins longue à coder pour ajouter ces attributs. Ainsi nous avons au total 592 MVP pour un

total de 14949 joueurs. De plus, nous appliquons une fonction à ce score MVP. Au préalable les moins bons au MVP ont un score proche de 0,001 alors que les meilleurs ont un score proche de 1. Nous pensons que cela donne une bien trop faible importance à ceux classés dans les dernières places aux MVP alors qu'ils ont certainement réalisé une très belle saison. En effet ils ont quasiment le même score que le plus mauvais joueur de la saison. Pour pallier à cela nous appliquons :

$$f(x) = x^{0.2} \tag{1}$$

I.3 Normalisation des données

Fonctions associées : normalise_by_year dans fonctions_utiles.py

Étant donné que le MVP est élu sur une saison donnée, chaque performance est à mettre en perspective avec celles réalisées par les autres joueurs cette année là. C'est pourquoi nous avons choisi de normaliser les statistiques par saison. Par exemple, en 2012 en raison d'une grève des joueurs la saison a été réduite à 66 matchs ce qui a par exemple réduit le nombre de points marqués des joueurs. Lors de son apprentissage nous ne voulons pas que notre algorithme pense qu'il est possible d'être MVP sur une saison quelconque en jouant un nombre si faible de matchs ou en marquant si peu de points.

I.4 Choix des features

Fonctions utilisées : SelectKBest, chi2 de la librairie sickit-learn

La base construite comprenant une quarantaine de features, nous allons vite rencontrer un problème de complexité si nous ne réduisons par leur nombre. Pour le choix de ces derniers, nous nous sommes appuyés d'une part sur notre connaissance de ce sport qui nous permet d'éliminer facilement certaines caractéristiques inutiles et d'autre part des test statistiques de base qui nous ont permis de garder des features moins intuitifs et plus complexes comme le 'VORP' (Value Over Remplacement Player) qui est une statistique caractérisant la contribution globale d'un joueur sur son équipe. Les features que nous avons gardés sont généralement des statistiques avancées dans le sens où elles sont issues de calculs faits à partir de statistiques brutes. Ainsi ces statistiques ont généralement une grande signification et contiennent beaucoup d'informations.

I.5 Division de la base de données

Fonction associée : coupe_by_year

On ne peut pas utiliser les fonctions pour couper le dataset en train/test set de la librairie sickit-learn car on souhaite conserver les données par année pour l'entraînement.

II Régression linéaire "artisanale"

L'objectif de ce projet étant avant tout de nous faire coder, nous avons décidé de coder nous-même certaines fonctions classiques comme la normalisation des données ou encore la descente de gradient afin de ne pas rendre la partie algorithmique trop succincte.

II.1 Approche directe : régression linéaire

Fichier associé : reg_lin.py

II.1.1 Commentaire d'un code : la descente de gradient

Inutile de faire des rappels mathématiques sur quoi consiste cette méthode d'optimisation. On code d'abord la fonction "hypothesis" qui prend en argument le paramètre θ que l'on souhaite optimiser, la base de données X et n qui correspond au nombre de features. On retourne un vecteur h dont la ligne i correspond au produit scalaire entre notre paramètre θ et la ligne i de notre base de données. Cette valeur correspond alors au score prédit par notre régression linéaire du joueur situé à la ligne i. Ensuite on met en place la descente de gradient dans la fonction "BGD" qui prend comme arguments supplémentaires la variable cible y, un coefficient d'apprentissage α et le nombre d'itération n_iters. On retourne le coût 'cost' qui est une liste qui contient la "mean squarred error" entre la prédiction contenu dans h et la cible y avant chaque correction. À chaque étape, le paramètre θ est corrigé.

Complexité : on note n_{iter} le nombre d'itérations de descente que nous voulons effectuer ; n le nombre de features ; m le nombre de lignes dans notre base.
On fait n_{iter} boucles dans lesquelles on met à jour chacun des n θ_i. Chaque mise à jour demande $O(m)$ opérations. De plus à chaque boucle on calcul la fonction $hypothesis$ en $O(m*n)$
Finalement la complexité est de $O(n_{iter} * m * n)$

II.1.2 Résultats et commentaires

Comme on pouvait s'en douter, les résultats ne sont pas excellents. Notre méthode pour savoir si le modèle est bon ou pas est d'entraîner le modèle sur 35 années puis de le tester sur une année. Nous faisons cela car ce qui nous intéresse avant tout est le classement final des MVP. Ainsi nous ne pouvons tester qu'une année à la fois. Nous obtenons seulement 15 bonnes prédictions de MVP et uniquement un seul podium prédit dans le bon ordre.
En étudiant les prédictions nous remarquons un phénomène : les scores prédits pour ceux classés au MVP sont à chaque fois trop faibles. Alors même si ce n'est pas directement les scores qui nous intéressent mais l'ordre des scores, cela nous donne une information. Nous en

concluons que le nombre de non-MVP est tellement grand par rapport au nombre de classés MVP que le modèle n'estime pas comme essentielle la bonne prédiction du score MVP.

II.2 Oversampling : régression linéaire avec du bruit

Fonction associée : add_nois_mvp dans fonctions_utiles.py

Comme nous venons de le voir, donner plus d'importance aux joueurs classés au MVP pourrait être une bonne idée. Pour cela nous faisons de l'oversampling. Nous créons donc des données bruitées (avec un bruit gaussien) sur chaque joueur classé. Nous en ajoutons autant qu'il en faut pour qu'il y ait autant de joueurs classés que de non classés.
Puis nous ré-appliquons le même procédé que précédemment, avec une régression linéaire. Cette méthode nous donne des meilleurs résultats : 20 bonnes prédictions du MVP et 3 bons podiums. Cela n'est pas encore concluant mais cela a le mérite de nous montrer une transformation de la base qui peut nous être utile, nous réutiliserons cette méthode d'oversampling dans la suite.

II.3 Undersampling : classification puis régression linéaire

La deuxième solution pour augmenter l'importance des joueurs classés au MVP est de réduire celle des joueurs non classés. En effet, on voit que l'un des problèmes lors de la régression vient du fait que des joueurs très mauvais obtiennent un score de 0 et que de bons joueurs obtiennent aussi un score de 0. En fait, les joueurs non classés faussent la régression. C'est pourquoi on a eu l'idée d'essayer de supprimer ces joueurs à l'aide d'une première classification entre joueurs classés au MVP et joueurs non classés au MVP.

II.3.1 Classification

Fichiers associés : classification.py

Pour la classification, on a utilisé des classificateurs de Sklearn : Support vector classifier (avec différents kernel) et Quadratic discrimant analysis. Nous voulons que notre classificateur ne passe pas à côté de joueurs classés au MVP, c'est-à-dire peu de faux négatifs, donc nous voulons un haut recall. Mais nous souhaitons quand même avoir une bonne précision car sinon la classification n'aurait pas vraiment d'intérêts, car elle conserverait trop de joueurs. Nous allons donc comparer les classificateurs énumérés précédemment.

Classificateur	Recall	Precision	F1
SVC_linear	0.92	0.369	0.524
SVC_poly	0.914	0.433	0.586
SVC_sigmoid	0.801	0.209	0.325
SVC_RBF	0.926	0.373	0.53
QDA	0.960	0.241	0.384

Même si le QDA a le meilleur recall, on va favoriser le meilleur F1 qui est obtenu avec le SVC polynomial de degré 2.

II.3.2 Régression

Fichiers associés : class_reg.py

Désormais, une fois que nous avons en théorie gardés uniquement les joueurs classés au MVP, nous voulons affecter un score à chacun des joueurs choisis par le classificateur. Nous utilisons de nouveau une régression linéaire, que nous entraînons uniquement sur les joueurs classés au MVP. Puis nous nous servons de ce modèle pour prédire les scores MVP des saisons à tester.

II.3.3 Résultats

Fichiers associés : compare.py

En essayant cet méthode sur les 36 années, nous obtenons 22 bons résultats sur le MVP et 5 podium dans l'ordre.

II.3.4 Ceiling analysis

Fonctions associés : score_class_perf_reg dans class_reg.py

Pour savoir quelle partie était à améliorer dans cette méthode, nous avons réaliser une sorte de ceiling analysis. Nous avons donc tester le résultat de la régression si nous avions parfaitement classifié les joueurs. On obtient de meilleurs résultats mais ce n'est pas significatif (23 bons résultats sur le MVP et 5 podium dans l'ordre). Cela nous indique qu'il est plus important d'améliorer la partie régression que la partie classification.

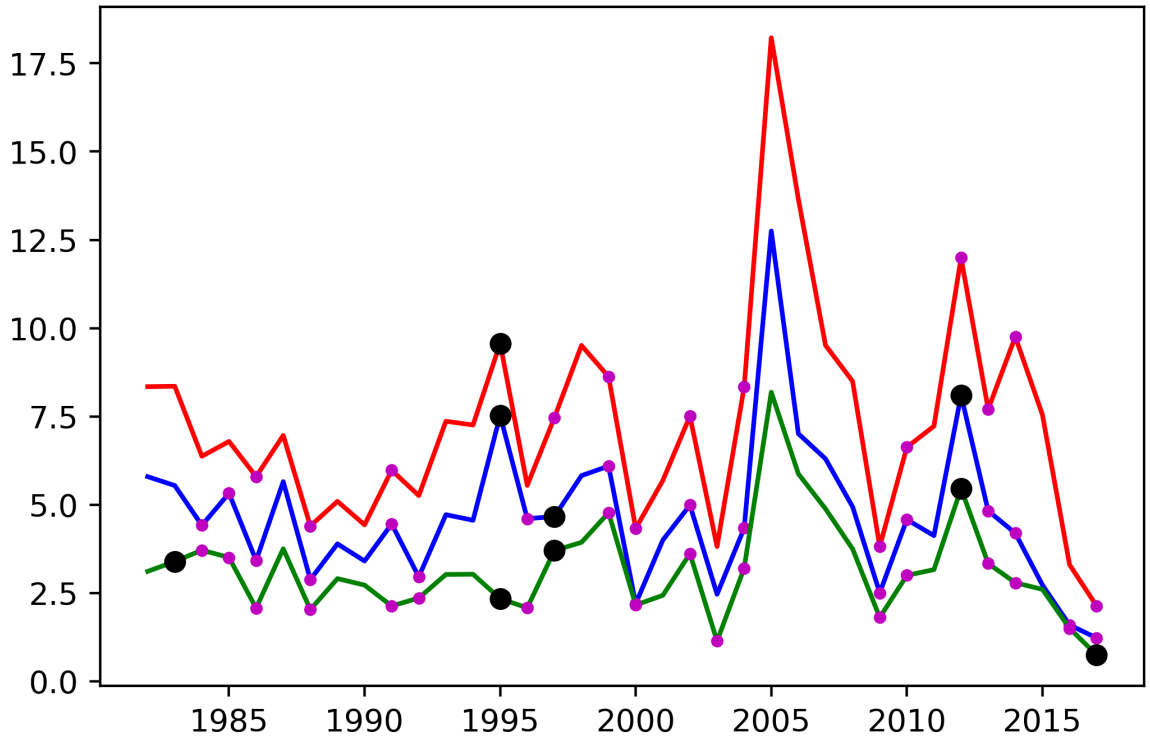
II.4 Synthèse

Fichiers associés : compare.py

Nous avons comparé les 3 méthodes pour voir si il y avait une évolution. Tout d'abord avec les chiffres que nous avons donné nous voyons très bien l'évolution : 15 premières places justes pour la régression linéaire simple, 20 pour la régression linéaire avec bruit et enfin 22 pour la classification suivi de la régression. Pour mieux visualiser ces résultats, nous avons tracé les résultats chacune des méthodes sur chaque année. Les résultats se composent d'un score que nous attribuons au résultat de cette manière :

$$\frac{\sum_{i=0}^{n_{annee}} 1/i * |i - i_{predit}|}{\sum_{i=0}^{n_{annee}} 1/i} \tag{2}$$

où i est le classement réel d'un joueur, i_{predit} le classement prédit du joueur et n_{annee} le nombre de joueurs classés pour l'année considéré.
C'est donc la moyenne pondérée des écarts absolues entre le rang prédit et le rang réel. Les coefficients de pondération, sont l'inverse du rang réel, car bien prédire les premières places est plus important. Les résultats se composent aussi d'un point noir pour dire si le podium prédit est le bon et d'un point magenta signifiant une bonne prédiction du vainqueur.



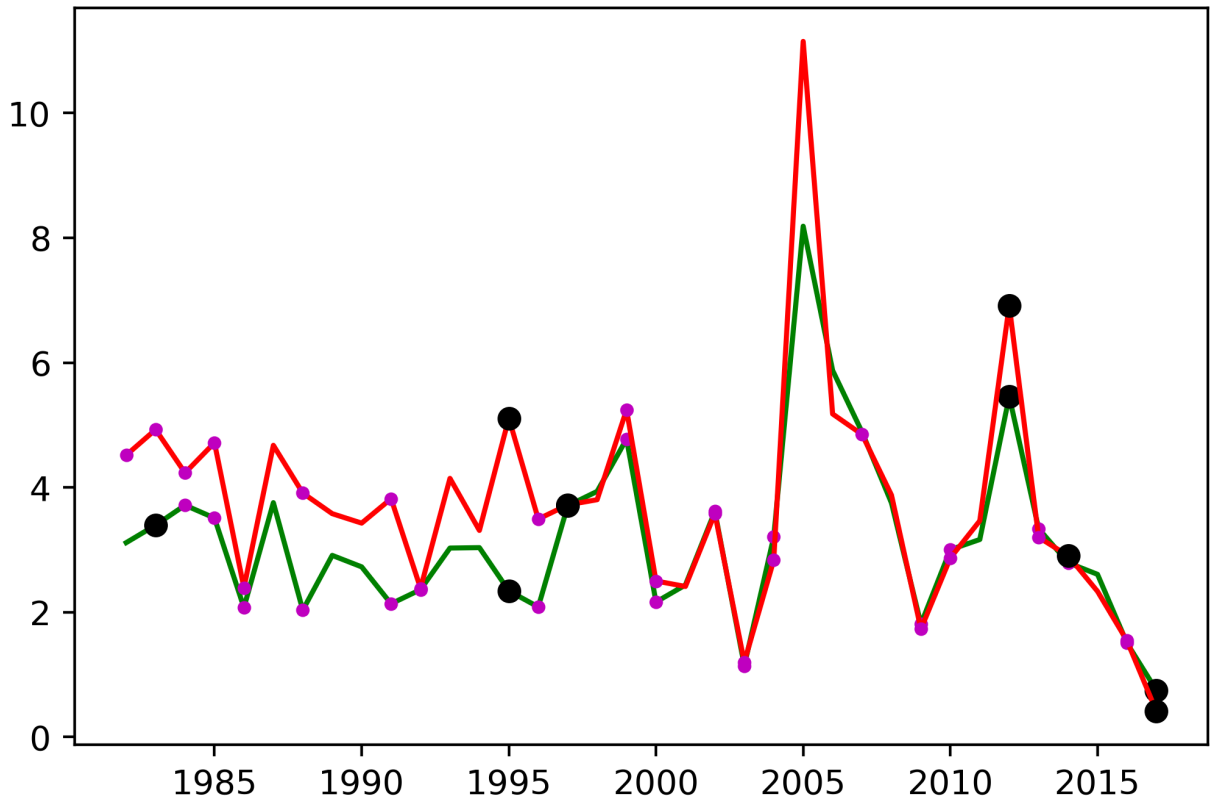
Rouge : régression linéaire ; Bleu : régression linéaire avec du bruit ; Vert : classification puis régression
Ainsi on voit que l'erreur a diminué pour chaque années en fonction de la méthode utilisée.

III Régression polynomiale

Fichiers associés : poly_reg.py

Étant donné que le meilleur classificateur était le polynomial, il semble cohérent de vouloir faire la régression avec un modèle polynomial. Nous reprenons les mêmes features que précédemment mais en rajoutant les termes d'ordres 2. Cela nous fait alors 36 features. De plus dans la méthode précédente, l'oversampling avait donné des bons résultats. Nous réutilisons donc ce processus dans cette méthode. Enfin pour que l'entraînement reste rapide, nous utilisons désormais une descente de gradient en mini-batch.

Pour évaluer ce nouveau modèle, on utilise la même méthode que précédemment et nous gardons comme référence le modèle de classification puis régression.

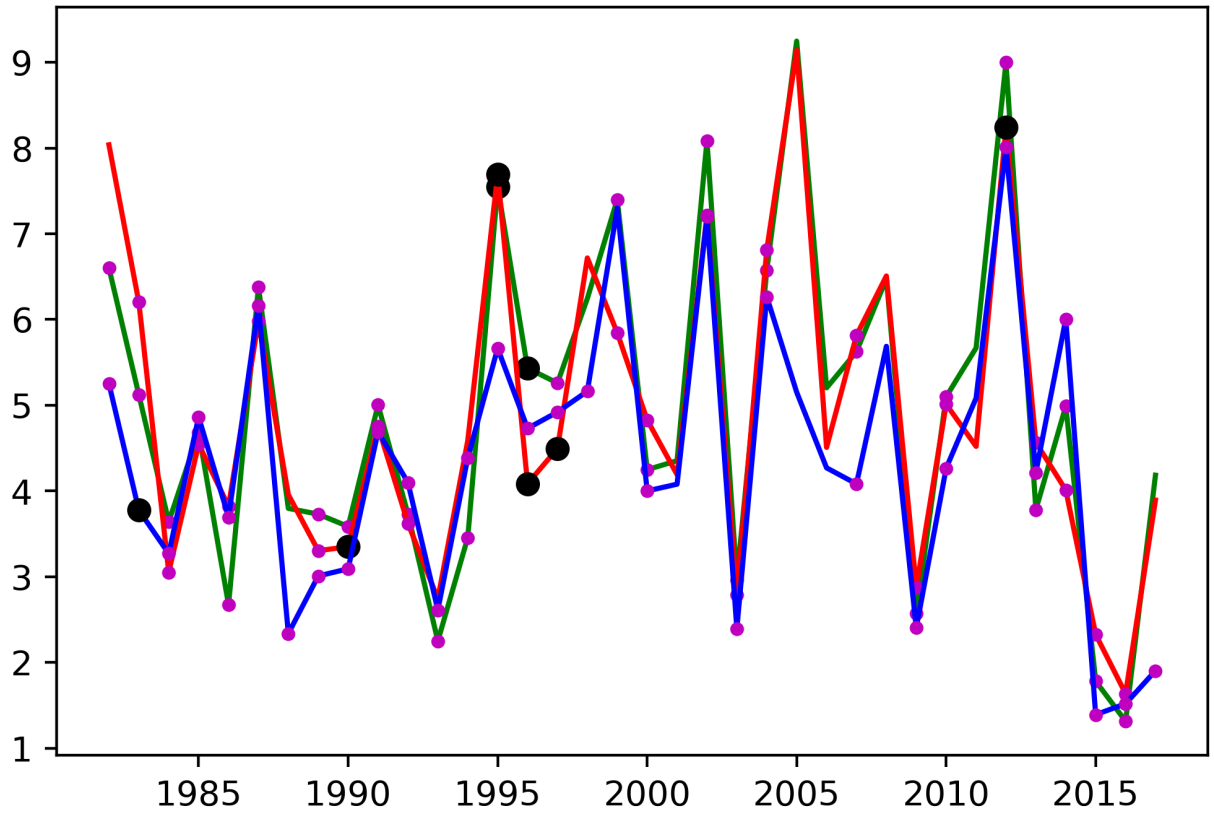


Vert : classification puis régression ; Rouge : Regression polynomiale.

Cette fois, on obtient 23 bonnes premières places et 5 bons podium. Cependant on remarque que l'erreur sur l'ensemble du classement est plus grande.

IV Un exemple de modèle plus performant : la Random Forest

Enfin nous avons décidé d'utiliser une random forest. Les résultas sont satisfaisants mais dépendent beaucoup des hyperparamètres du modèle. Pour preuve voici les résultats avec 3 paramètrages différents.



Ce qui diffère principalement lors des différents paramètrages, c'est le maximum de profondeur pour chaque arbre. On remarque que plus la profondeur est élévée et plus la prédiction sur l'ensemble des classés est bonne mais qu'on perd de la précision sur les premiers. Cela se traduit qualitativement par des chiffres allant de 26 premiers justes et 5 bons podium jusque 30 premiers justes et 1 seul bon podium.

Remarques conclusives

Nous avons implémenté une méthode pour obtenir le bon MVP avec plus de 80% de réussite pour la prédiction du MVP. Cela peut sembler très bon en sachant qu'il y a environ 400 joueurs par saison. Cependant, il faut relativiser la portée de ce résultat, en effet comme l'a dit Tony Parker, le titre de MVP se joue très généralement entre seulement deux joueurs. Toutefois, nous pensons que cela reste des résultats très honnêtes pour plusieurs raisons :

- Le nombre de saisons n'est pas assez grand pour que l'algorithme ait vue une assez grande variété de MVP. Il faut savoir que la façon dont on joue en NBA a évolué dans le temps, nous sommes passés d'un jeu très physique où les intérieurs étaient dominants et remportaient souvent le trophée à un jeu plus orienté vers le shoot à 3 points qui a permis à des joueurs comme Stephen Curry de remporter le trophée.
- L'élection du MVP de l'année ne repose pas sur des phénomènes physiques comme la météo par exemple : il y a des critères subjectifs qui sont difficiles à quantifier. On peut citer le style de jeu, l'impact extra-sportif ou encore des matchs légendaires qui marquent les esprits.
- Les journalistes peuvent se tromper, on remarque que dans tous les graphiques l'erreur la plus haute est en 2005 que nous avions mentionné en introduction : les modèles ne classaient jamais Steve Nash dans le top 3. Ainsi, cette erreur d'un point de vue algotrithmique nous donne un autre regard sur la réalité et n'est pas vraiment une erreur "réelle".

De manière plus globale, ce modèle pourrait être utile pour aider les journalistes à voter parmis une multitude de joueurs et de lignes statistiques en leur proposant un premier classement qu'ils peuvent ensuite modifier selon l'importance qu'ils accordent à certaines caractéristiques.

Annexe

Résultats régression linéaire

Year	Score	Premier	Podium
1982.0	8.34586485218888	0.0	0.0
1983.0	8.356134803303728	0.0	0.0
1984.0	6.37538132383778	0.0	0.0
1985.0	6.79467545969984	0.0	0.0
1986.0	5.798890720711755	1.0	0.0
1987.0	6.965135410047613	0.0	0.0
1988.0	4.40161959657409	1.0	0.0
1989.0	5.0991863040478735	0.0	0.0
1990.0	4.4336489126066585	0.0	0.0
1991.0	5.980914855990913	1.0	0.0
1992.0	5.261623650966284	0.0	0.0
1993.0	7.366809116809117	0.0	0.0
1994.0	7.259529226989913	0.0	0.0
1995.0	9.570938744243188	1.0	1.0
1996.0	5.550754701246716	0.0	0.0
1997.0	7.4679217313505495	1.0	0.0
1998.0	9.511975940543584	0.0	0.0
1999.0	8.626451377176501	1.0	0.0
2000.0	4.32961641065026	1.0	0.0
2001.0	5.688887523372278	0.0	0.0
2002.0	7.512657090178878	1.0	0.0
2003.0	3.8196640037858733	0.0	0.0
2004.0	8.335490204828414	1.0	0.0
2005.0	18.22358555984801	0.0	0.0
2006.0	13.652299227040482	0.0	0.0
2007.0	9.515985615536877	0.0	0.0
2008.0	8.497142006144346	0.0	0.0
2009.0	3.8217518806579114	1.0	0.0
2010.0	6.631192778991081	1.0	0.0
2011.0	7.229231342356887	0.0	0.0
2012.0	11.994270771167221	1.0	0.0
2013.0	7.708012722940131	1.0	0.0
2014.0	9.757943573876197	1.0	0.0
2015.0	7.550757575757575	0.0	0.0
2016.0	3.3063270559544784	0.0	0.0
2017.0	2.135243864921788	1.0	0.0

Résultats régression linéaire avec bruit

Year	Score	Premier	Podium
1982.0	5.797870217235967	0.0	0.0
1983.0	5.54601899494358	0.0	0.0
1984.0	4.4192603925098926	1.0	0.0
1985.0	5.329092435649655	1.0	0.0
1986.0	3.423342950403395	1.0	0.0
1987.0	5.656527261055367	0.0	0.0
1988.0	2.884133350094039	1.0	0.0
1989.0	3.8950829381761163	0.0	0.0
1990.0	3.409200085982112	0.0	0.0
1991.0	4.458326135637441	1.0	0.0
1992.0	2.961338157784656	1.0	0.0
1993.0	4.721509971509971	0.0	0.0
1994.0	4.563054990636893	0.0	0.0
1995.0	7.529437837286338	1.0	1.0
1996.0	4.607977525747404	1.0	0.0
1997.0	4.667158524393659	1.0	1.0
1998.0	5.827024326056147	0.0	0.0
1999.0	6.095692609642935	1.0	0.0
2000.0	2.2025476079869635	1.0	0.0
2001.0	4.001597654432613	0.0	0.0
2002.0	4.9932978280854385	1.0	0.0
2003.0	2.467239805748668	0.0	0.0
2004.0	4.341005845865499	1.0	0.0
2005.0	12.751422934982159	0.0	0.0
2006.0	7.013625049128782	0.0	0.0
2007.0	6.303110070160527	0.0	0.0
2008.0	4.944450271547447	0.0	0.0
2009.0	2.507841387224276	1.0	0.0
2010.0	4.583554776172391	1.0	0.0
2011.0	4.127778467651806	0.0	0.0
2012.0	8.096789244674962	1.0	1.0
2013.0	4.82967846232981	1.0	0.0
2014.0	4.197985929340026	1.0	0.0
2015.0	2.730681818181818	0.0	0.0
2016.0	1.600731608183173	1.0	0.0
2017.0	1.2301157204723479	1.0	0.0

Résultats classification puis régression

Year	Score	Premier	Podium
1982.0	3.1153247775312733	0.0	0.0
1983.0	3.3959163095538196	1.0	1.0
1984.0	3.716135562137935	1.0	0.0
1985.0	3.5102763582576983	1.0	0.0
1986.0	2.0751777736719674	1.0	0.0
1987.0	3.756364692322564	0.0	0.0
1988.0	2.0339011956414694	1.0	0.0
1989.0	2.9116109541571604	0.0	0.0
1990.0	2.7275395097150437	0.0	0.0
1991.0	2.135726130754282	1.0	0.0
1992.0	2.3630364761984435	1.0	0.0
1993.0	3.0292022792022792	0.0	0.0
1994.0	3.035599943619999	0.0	0.0
1995.0	2.3429961104137376	1.0	1.0
1996.0	2.085023766834841	1.0	0.0
1997.0	3.7066692031768156	1.0	1.0
1998.0	3.937318661513445	0.0	0.0
1999.0	4.777115882095205	1.0	0.0
2000.0	2.164289555490651	1.0	0.0
2001.0	2.4374149600147472	0.0	0.0
2002.0	3.6179762008699976	1.0	0.0
2003.0	1.1420265687147186	1.0	0.0
2004.0	3.207648712570842	1.0	0.0
2005.0	8.187382508553906	0.0	0.0
2006.0	5.876457487226516	0.0	0.0
2007.0	4.887587700662223	0.0	0.0
2008.0	3.7512468443568334	0.0	0.0
2009.0	1.8106591865357642	1.0	0.0
2010.0	3.002732622564577	1.0	0.0
2011.0	3.1660250838196946	0.0	0.0
2012.0	5.459553597586779	1.0	1.0
2013.0	3.339582830329306	1.0	0.0
2014.0	2.7939854379230424	1.0	0.0
2015.0	2.6075757575757574	0.0	0.0
2016.0	1.5062999593551012	1.0	0.0
2017.0	0.7490829921669783	1.0	1.0

Résultats régression polynomiale

Year	Score	Premier	Podium
1982.0	4.517814015322142	1.0	0.0
1983.0	4.928406041014845	1.0	0.0
1984.0	4.2342023577984635	1.0	0.0
1985.0	4.717604821623718	1.0	0.0
1986.0	2.3927815124856093	1.0	0.0
1987.0	4.675763065656397	0.0	0.0
1988.0	3.910533062493144	1.0	0.0
1989.0	3.580083876509432	0.0	0.0
1990.0	3.4285927905868276	0.0	0.0
1991.0	3.820057284736454	1.0	0.0
1992.0	2.3840876767338743	1.0	0.0
1993.0	4.145655270655271	0.0	0.0
1994.0	3.311580049533859	0.0	0.0
1995.0	5.1098191354932485	1.0	1.0
1996.0	3.493677813451194	1.0	0.0
1997.0	3.7209121363384816	1.0	1.0
1998.0	3.8039100514334803	0.0	0.0
1999.0	5.245886701798821	1.0	0.0
2000.0	2.50083838220303	1.0	0.0
2001.0	2.413221907178033	0.0	0.0
2002.0	3.5837976919876944	1.0	0.0
2003.0	1.1995245123998604	1.0	0.0
2004.0	2.835030568075326	1.0	0.0
2005.0	11.149181440609118	0.0	0.0
2006.0	5.178566749639723	0.0	0.0
2007.0	4.85008238939612	1.0	0.0
2008.0	3.87714829941559	0.0	0.0
2009.0	1.7307152875175313	1.0	0.0
2010.0	2.86988843584254	1.0	0.0
2011.0	3.4662858562026573	0.0	0.0
2012.0	6.917284258726765	1.0	1.0
2013.0	3.201583019283862	1.0	0.0
2014.0	2.905788651900961	1.0	1.0
2015.0	2.3344696969696965	0.0	0.0
2016.0	1.5440997154857068	1.0	0.0
2017.0	0.4136592753863265	1.0	1.0

Résultats random forest 26 5

Year	Score	Premier	Podium
1982.0	8.03703810867063	0.0	0.0
1983.0	6.202610877978618	1.0	0.0
1984.0	3.053344835332404	1.0	0.0
1985.0	4.546006439344377	1.0	0.0
1986.0	3.822116971365134	1.0	0.0
1987.0	5.983280625608206	1.0	0.0
1988.0	3.9574771828883883	0.0	0.0
1989.0	3.3043106437693046	1.0	0.0
1990.0	3.3510780474583868	1.0	1.0
1991.0	4.751863672395202	1.0	0.0
1992.0	3.622657491843052	1.0	0.0
1993.0	2.742877492877492	0.0	0.0
1994.0	4.586835269717898	0.0	0.0
1995.0	7.694914602214329	1.0	1.0
1996.0	4.085576241504398	1.0	1.0
1997.0	4.495688631026788	1.0	1.0
1998.0	6.717956583220431	0.0	0.0
1999.0	5.8443984118504995	1.0	0.0
2000.0	4.829794988906995	1.0	0.0
2001.0	4.1913937340344285	0.0	0.0
2002.0	7.216857631488392	1.0	0.0
2003.0	2.790041802345889	1.0	0.0
2004.0	6.814744076040875	1.0	0.0
2005.0	9.141020699680658	0.0	0.0
2006.0	4.509236211188261	0.0	0.0
2007.0	5.815955561543324	1.0	0.0
2008.0	6.507807739762823	0.0	0.0
2009.0	2.87415529771771	1.0	0.0
2010.0	5.012264526470935	1.0	0.0
2011.0	4.520427169998758	0.0	0.0
2012.0	8.239466574073068	1.0	1.0
2013.0	4.566485760174678	1.0	0.0
2014.0	4.011960584416723	1.0	0.0
2015.0	2.327651515151515	1.0	0.0
2016.0	1.637041051348056	1.0	0.0
2017.0	3.8943133681993514	0.0	0.0

Résultats random forest 28 3

Year	Score	Premier	Podium
1982.0	6.601209065239607	1.0	0.0
1983.0	5.123326635922834	1.0	0.0
1984.0	3.641563525858203	1.0	0.0
1985.0	4.684595904717147	1.0	0.0
1986.0	2.673594036469066	1.0	0.0
1987.0	6.381383963202204	1.0	0.0
1988.0	3.7977537985816925	0.0	0.0
1989.0	3.7307294421114627	1.0	0.0
1990.0	3.586912027215207	1.0	0.0
1991.0	5.004471911195451	1.0	0.0
1992.0	3.73086777378064	1.0	0.0
1993.0	2.251068376068376	1.0	0.0
1994.0	3.455892716911987	1.0	0.0
1995.0	7.550641141971152	1.0	1.0
1996.0	5.432014343493307	1.0	1.0
1997.0	5.259967070028091	1.0	0.0
1998.0	6.245942149417455	0.0	0.0
1999.0	7.3994629641856084	1.0	0.0
2000.0	4.249274093623821	1.0	0.0
2001.0	4.353590772228903	0.0	0.0
2002.0	8.086768731414955	1.0	0.0
2003.0	2.9659609469189077	1.0	0.0
2004.0	6.572644919452005	1.0	0.0
2005.0	9.246537490924586	0.0	0.0
2006.0	5.20332765622953	0.0	0.0
2007.0	5.624801799094232	1.0	0.0
2008.0	6.477439102992667	0.0	0.0
2009.0	2.569679969399464	1.0	0.0
2010.0	5.09936418112769	1.0	0.0
2011.0	5.6686948963119335	0.0	0.0
2012.0	8.997167938020159	1.0	0.0
2013.0	3.782429325747158	1.0	0.0
2014.0	4.989436998094989	1.0	0.0
2015.0	1.7829545454545452	1.0	0.0
2016.0	1.316081831730118	1.0	0.0
2017.0	4.184726790827556	0.0	0.0

Résultats random forest 30 1

Year	Score	Premier	Podium
1982.0	5.251752452630547	1.0	0.0
1983.0	3.776469846824538	1.0	1.0
1984.0	3.2703067209031014	1.0	0.0
1985.0	4.862319379159709	1.0	0.0
1986.0	3.6954981373835567	1.0	0.0
1987.0	6.165237698653199	1.0	0.0
1988.0	2.3383273830160145	1.0	0.0
1989.0	3.009480140639658	1.0	0.0
1990.0	3.093113019747848	1.0	0.0
1991.0	4.692016456091952	1.0	0.0
1992.0	4.0990650882623605	1.0	0.0
1993.0	2.6125356125356127	1.0	0.0
1994.0	4.386867486861447	1.0	0.0
1995.0	5.664854983077666	1.0	0.0
1996.0	4.730288120752199	1.0	0.0
1997.0	4.92119885817611	1.0	0.0
1998.0	5.163656766098492	1.0	0.0
1999.0	7.299288303753097	0.0	0.0
2000.0	4.001514050732497	1.0	0.0
2001.0	4.079961726520184	0.0	0.0
2002.0	7.198985173049565	1.0	0.0
2003.0	2.3954546991019816	1.0	0.0
2004.0	6.266491141951893	1.0	0.0
2005.0	5.152896350034264	0.0	0.0
2006.0	4.270404821171231	0.0	0.0
2007.0	4.0833428331588815	1.0	0.0
2008.0	5.6874173807981565	0.0	0.0
2009.0	2.4044370776488586	1.0	0.0
2010.0	4.266602833751115	1.0	0.0
2011.0	5.0828262759220175	0.0	0.0
2012.0	8.010514437005611	1.0	0.0
2013.0	4.2159120731464075	1.0	0.0
2014.0	6.005148793734495	1.0	0.0
2015.0	1.3924242424242423	1.0	0.0
2016.0	1.5195772930497224	1.0	0.0
2017.0	1.905033485576354	1.0	0.0