UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Master 2 Modélisations économiques, statistiques et financières

Projet Data Mining

Etudiants :
Mehdi FERHAT Abla SEFRIOUI Zhanike YEVNIYEVA

23 janvier 2022

Table des matières

Introduction

League of Legends (LoL), est un jeu vidéo développé par Riot Games et est actuellement le jeu d'arène de combat en ligne multijoueur (MOBA) le plus joué au monde. En 2019, on comptait huit millions de joueurs simultanés par jour et la base de joueurs n'a cessé de croître depuis sa sortie en 2009. L'un des aspects essentiels de LoL est le jeu compétitif. Dans un mode de jeu classé classique, dix joueurs sont regroupés dans une partie pour former deux équipes de compétences de jeu à peu près égales. Ces deux équipes, composées de cinq joueurs chacune, s'affrontent pour détruire la base de l'équipe adverse (aussi appelé le Nexus).

Un matchmaking équitable est crucial pour l'expérience des joueurs et Riot Games a déclaré à maintes reprises que le matchmaking classé et la question de l'équité resterait plus que tout au centre de ses priorités . Cet objectif a persisté tout au long de l'histoire du jeu. Afin de créer des matchs équitables entre des joueurs de niveau approximativement équivalent, le matchmaking est déterminé à l'aide d'un système de classement d'ELO, similaire à celui utilisé à l'origine par les joueurs d'échecs. Bien que ce système se soit amélioré ces dernières années, il ne tient pas compte de la sélection des champions des joueurs lors de la formation des matchs.

En effet, LoL compte plus de 150 personnages jouables, appelés champions, qui ont un style de jeu et des capacités qui leur sont propres. Les joueurs choisissent un champion au début de chaque match, après que l'algorithme a formé les équipes. Cependant, les joueurs sont souvent plus performants avec certains champions qu'avec d'autres en raison de leurs différents niveaux d'expertise mécanique, qui se définit comme la connaissance qu'a un joueur des capacités et des interactions de son champion. Des niveaux plus élevés d'expertise mécanique sur des champions particuliers permettent aux joueurs de prendre de meilleures décisions, plus rapidement, ce qui est essentiel dans l'environnement à haute intensité du jeu. Étant donné que l'expertise mécanique a un tel impact sur les performances d'un joueur, elle peut donc causer une perte d'efficacité du joueur et donc avoir un impact similaire sur l'issue du match. Cependant, on peut tout à fait se poser la question de si un ou plusieurs champions feraient abstraction de cet aspect en permettant aux joueurs avec une moins bonne maîtrise de surpasser et dominer d'autres joueurs ?

Dans cet objectif, nous allons essayer à l'aide d'une base de données de plus d'1 million 500 000 parties solo classées de relever les différents déterminants de victoire d'une partie, notamment comme dit précédemment dans un souci d'équité, en faisant ressortir les champions les plus efficaces, s'ils existent. Premièrement en effectuant une analyse exploratoire des données, puis en utilisant des algorithmes d'apprentissage non supervisés et enfin des algorithmes d'apprentissage supervisés.

1 EDA

1.1 Pre-processing et traitement des valeurs manquantes

Le jeu de données est composé de 5 dataframes contenant le nom des champions et leur id (tel qu'il est décrit dans le code source du jeu), des statistiques relatives à l'identification des matches (durée, champion, id), les participants (localisation notamment) et enfin des statistiques in-Game (performances des joueurs/champions : éliminations, assistances etc.). Une jointure a donc été effectuée par l'id des champions, le dataset final contenant 1 834 520 observations pour 23 variables.

On crée une boucle qui permet de retourner les modalités de chacune des variables et excluant celles qui n'en ont qu'une seule (aucun pouvoir prédictif - explicatif).

```
"La variable id a 1834520 modalités"
"La variable matchid a 184069 modalités"
"La variable player a 10 modalités"
"La variable name a 136 modalités"
"La variable adjposition a 6 modalités"
"La variable team_role a 6 modalités"
"La variable win a 3 modalités"
"La variable kills a 47 modalités"
"La variable deaths a 33 modalités"
"La variable assists a 56 modalités"
"La variable turretkills a 12 modalités"
"La variable totdmgtochamp a 65654 modalités"
"La variable totheal a 37544 modalités"
"La variable totminionskilled a 562 modalités"
"La variable goldspent a 13209 modalités"
"La variable totdmgtaken a 71092 modalités"
"La variable inhibkills a 9 modalités"
"La variable pinksbought a 33 modalités"
"La variable wardsplaced a 183 modalités"
"La variable duration a 3057 modalités"
"La variable platformid a 4 modalités"
"La variable seasonid a 6 modalités"
"La variable version a 152 modalités"
```

L'étude des modalités peut nous donner une première indication sur la présence notamment de NaN et NA. En effet prenons par exemple les variables finies que l'on peut comptabiliser en jeu. On peut lire que turretkills qui correspond aux nombres de tourelles détruites par un champion, a 12 modalités, or c'est impossible puisqu'il n'existe que 22 tourelles (11 par

camp). De même, la variable "win" possède 3 modalités mais il n'existe sur League of Legends que 2 issues à une partie, la victoire ou la défaite, pas de match nul.

On peut ainsi commencer par effectuer un traitement des valeurs manquantes.

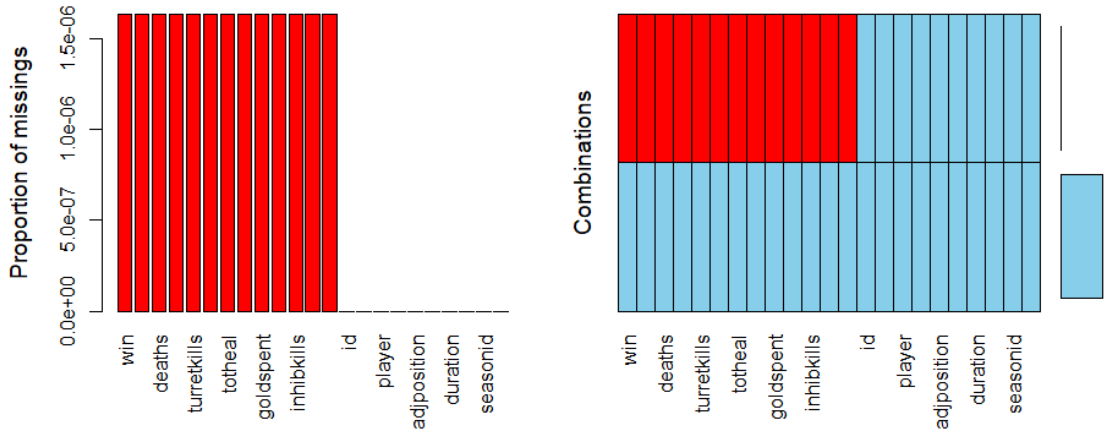
```
> print(paste("Il y a", sum(is.na(df_v)), "valeurs manquantes"))
[1] "Il y a 39 valeurs manquantes"
```

Sur une base de plus d'1 million d'observations on a donc seulement 39 valeurs manquantes. Voyons maintenant quelles variables sont concernées.

```
> res<-summary(aggr(df_v, sortVar=TRUE))$combinations

 Variables sorted by number of missings:
         Variable        Count
              win 1.635305e-06
            kills 1.635305e-06
           deaths 1.635305e-06
          assists 1.635305e-06
      turretkills 1.635305e-06
    totdmgtochamp 1.635305e-06
          totheal 1.635305e-06
 totminionskilled 1.635305e-06
        goldspent 1.635305e-06
      totdmgtaken 1.635305e-06
       inhibkills 1.635305e-06
      pinksbought 1.635305e-06
      wardsplaced 1.635305e-06
               id 0.000000e+00
          matchid 0.000000e+00
           player 0.000000e+00
             name 0.000000e+00
      adjposition 0.000000e+00
        team_role 0.000000e+00
         duration 0.000000e+00
       platformid 0.000000e+00
         seasonid 0.000000e+00
          version 0.000000e+00
```

On voit que 11 variables se distinguent (sans surprise les "turretkills" et "win" mentionnées avant) avec intéressemment le même nombre de valeurs manquantes pour chacune. 11 étant un multiple de 39 on peut déjà soupçonner que ces dernières suivent le même pattern, avec possiblement seulement 3 lignes/observations de NA.




```
> head(res[rev(order(res[,2])),])
                                                  Combinations   Count      Percent
1 0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0 1834517 9.999984e+01
2 0:0:0:0:0:0:1:1:1:1:1:1:1:1:1:1:1:1:1:1:1:0:0:0:0:0:0:0:0:0       3 1.635305e-04
```

Le graphique ainsi que la table des combinaisons vient confirmer nos hypothèses, les données manquantes surviennent simultanément pour chacune des variables et donc pour le même pattern 3 fois.

Sachant que les données ne sont pas collectées de manière déclaratives mais automatiques, on ne peut pas vraiment soupçonner d'interprétation à l'absence de valeurs si ce n'est un problème technique/logiciel qui affecterait l'ensemble des statistiques in-Game.

En effet, League of Legends peut-être séparé en 2 processus : le client et le jeu. Les statistiques telles que l'id de la partie, du joueur, ou la lane sélectionnée sont ainsi directement affectées avant même le démarrage de la partie. Rien ne relie par exemple l'absence du nombre d'éliminations à celui du résultat final (win) ou de la quantité d'or dépensée. C'est pourquoi on peut légitimement penser à un facteur extérieur, hypothétiquement une déconnexion de la liaison client - jeu qui rendrait indisponible toutes les statistiques durant la partie.

On se place ainsi dans le cas de données MCAR, c'est-à-dire manquantes de façon complètement aléatoire. Puisque leur proportion est minime la précision des résultats n'est pas à questionner, on décide simplement de négliger ces observations en les supprimant.

1.2 Analyse des corrélations

On va maintenant procéder à une analyse des corrélations dans le but de vérifier la cohérence des corrélations. En effet s'il y a des variables dont on ne peut que soupçonner le lien, League of Legends étant un jeu structuré qui demande un certains nombres de tâches avant la victoire ou autre, certaines d'entre-elles devraient obligatoirement être corrélées négativements ou positivement au risque de relever un problème inhérent à la base.

L'analyse sera séparée en 3 selon la durée d'une partie pour mettre en exergue les déterminants pour une victoire expéditive (moins de 25min) mais également pour une partie qui traîne en longueur (plus de 40min).

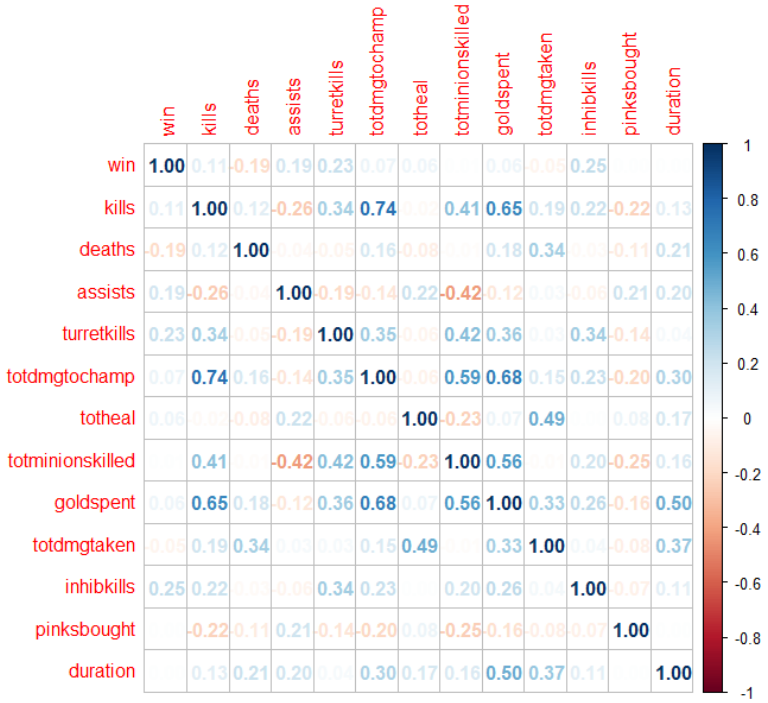


FIGURE 1 – *Corrélation des variables pour une game entière (toute la base)*

Pour le cas d'une partie indifférenciée au temps, sans surprise il n'y a pas de recette magique et aucune variable n'excède une corrélation supérieure à 0.25 (négative ou positive). De manière assez logique les inhibiteurs suivis des tourelles détruites ont les corrélations les plus élevées puisqu'on ne peut gagner une partie sans en détruire. Les inhibiteurs ayant une corrélation légèrement plus élevée étant donné qu'ils sont non seulement les plus éloignés mais aussi un élément de pression qui avantagent grandement pour l'accès au Nexus. Les kills et assists ont une corrélation positive tandis que les morts une corrélation négative ce qui est cohérent.

Pour ce qui est autres éléments de cohérence, marqués par une corrélation très élevée on a les dégats totaux infligés aux champions (totdmgtochamp) et les éliminations (kills) avec une corrélation de 0.74 de la même manière que l'or dépensé (0.68 - plus l'on dépense de l'argent

plus on est équipé avec de meilleures caractéristiques et donc une capacité d'élimination supérieure et inversement - plus l'on effectue d'éliminations plus on peut dépenser).

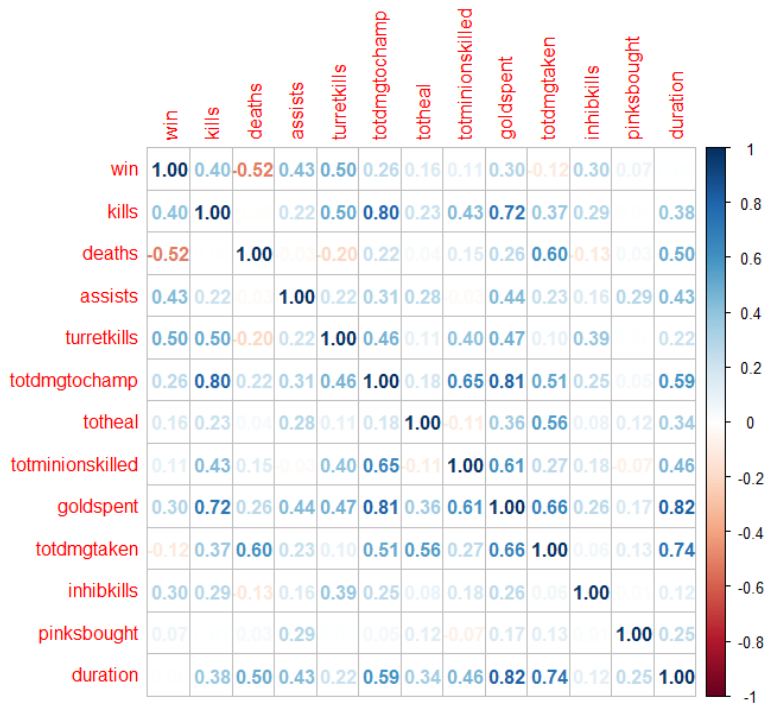


FIGURE 2 – *Corrélation des variables pour une partie de moins de 25min (très rapide)*

Pour ce qui est d'une partie inférieure ou égale à 25 minutes, beaucoup plus de certitudes semblent se dessiner. Les morts et le nombre de tourelles détruites sont maintenant déterminants au succès. Il y a effectivement un concept qui vient expliquer les différences aussi fortes selon la durée d'une partie : le scaling. Dans League of Legends, les joueurs et champions n'ont pas un niveau et un inventaire d'objets illimités, de ce fait à mesure qu'une partie s'étend sur la durée même si une équipe est dominée elle peut petit à petit rattraper son retard puisque les adversaires arrivent à un certains état de stationnarité. Ainsi le moment où l'écart et la domination est la plus élevée et suceptible de faire la différence peut être située dans les 25 premières minutes de jeu.

Les kills et les morts deviennent donc parmi les éléments les plus importants puisqu'ils permettent de rapidement surpasser ou être surpassé en termes d'or et donc d'équipement. On voit d'ailleurs que l'élément le plus corrélé au kills est maintenant l'or depensé passant de 0.65 précédemment à maintenant 0.81. Les tourelles détruites (turretkills) passent également de 0.25 à 0.5, grandement expliqué encore une fois par la durée d'une partie. Puisque ces dernières ont un nombre de points de vie limités et ne peuvent pas être soignées, à la longue elles finissent par tomber des deux côtés et ne sont plus un élément absolument déterminant (bien que nécessaire).

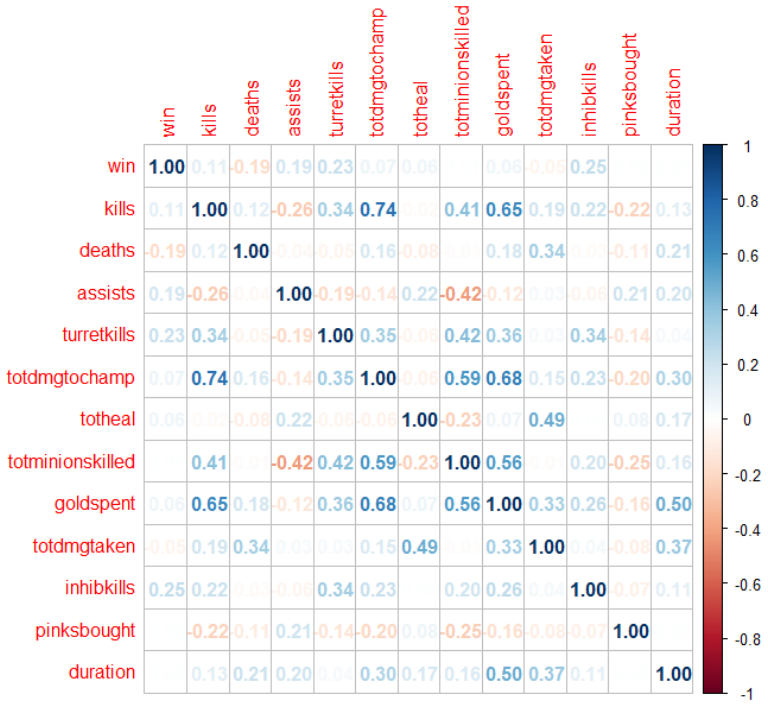


FIGURE 3 – *Corrélation des variables pour une partie de plus de 40min*

Enfin, les parties de plus de 40min il ne semble pas y avoir de grandes différences avec l'analyse indifférenciée, à titre indicatif la durée moyenne d'une partie si situe autour de 30min.

```
> mean(df_v$duration)
[1] 1833.816
```

1.3 Feature-engineering et détermination des champions les plus perfomants

Nous allons désormais essayer de mettre en évidence les personnages les plus effiaces. Pour cela nous créons deux nouvelles variables que sont le taux de victoire et le KDA ratio.

Le premier tableau correspond aux 10 meilleurs champions selon leur taux de victoire. On peut remarquer qu'il n'y a pas forcément de lien avec le ratio des éliminations et assistances. Par ailleurs sans regarder le nom du champion on peut s'interroger à la vue du KDA du rôle offensif des personnages en question. Ainsi sans connaissances préalables par rapport au jeu, Ivern, Sona, Janna caractérisés par un KDA inférieurs à 6 peuvent laisser un soupçon de personnage de soutien, ce qui est effectivement le cas lorsque l'on regarde la documentation du site officiel, ce sont tous des supports. Un support se caractérise par le fait d'aider son

coéquipier a tuer les sbires (totminionskilled) tranquillement en gênant l'adversaire, le soigner si besoin ou le protéger de dégâts.

	Champion	Victoires	Parties	Kills	Deaths	Assists	Win rate (%)	KDA Ratio
1	Ivern	4578	8194	2.61	4.09	12.98	55.87	5.78
2	Anivia	4194	7785	6.23	4.66	7.39	53.87	7.82
3	Xerath	3357	6273	7.09	5.34	8.36	53.52	8.66
4	Sona	7529	14090	2.93	5.57	13.52	53.44	5.36
5	Ahri	19949	37423	7.08	5.43	7.51	53.31	8.46
6	Janna	12856	24296	0.86	3.94	14.15	52.91	4.46
7	Skarner	1116	2111	4.71	4.95	8.95	52.87	6.51
8	Pantheon	5934	11305	7.87	6.31	6.63	52.49	8.92
9	Amumu	7118	13584	4.59	5.30	10.65	52.40	6.60
10	Draven	10633	20327	7.62	6.41	6.30	52.31	8.61

TABLEAU 1: Les 10 meilleurs champions en terme de taux de victoire (Win rate)

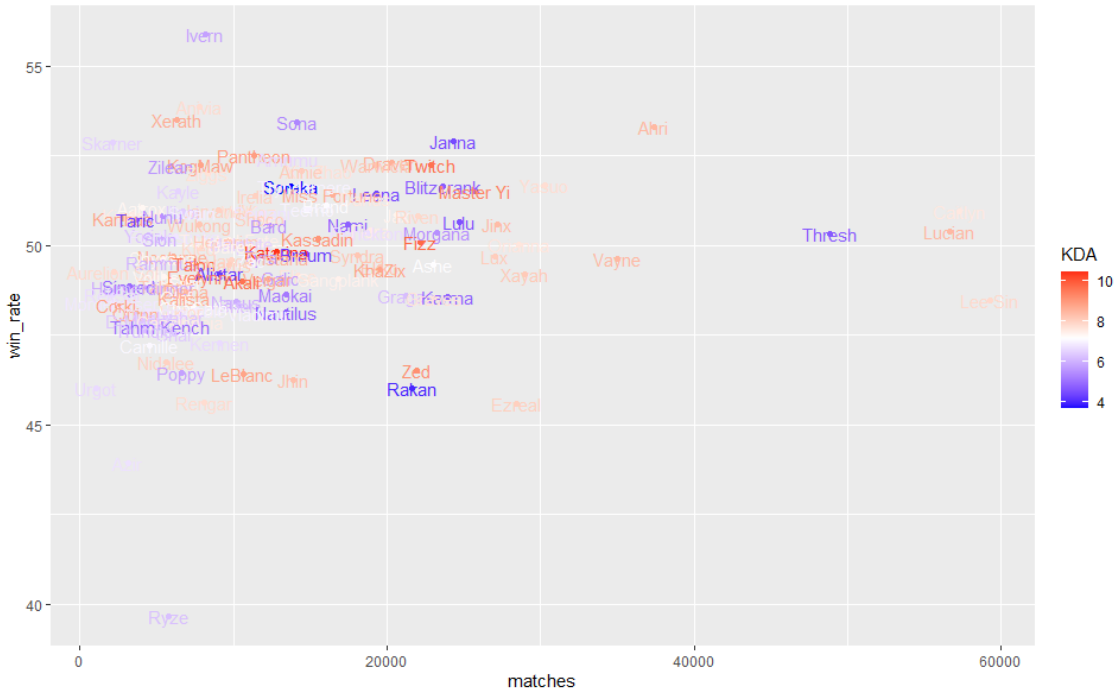


FIGURE 4 – *Scatterplot des champions en fonction du taux de victoire et nombre de totals de parties*

Globalement aucun champion ne semble superformer au point d'excéder un taux de victoire de 60% et pour les meilleurs on semble se stabiliser un peu au dessus des 50%. Même constat pour le bas de la liste symétriquement ou le taux de victoires se stabilise dans les 45% sauf étonnemment pour un champion du nom de Ryze qui descend en dessous de 40% avec

39.65% de Win rate néanmoins à relativiser tout comme pour Azir étant donné le peu de parties effectuées.

	Champion	Victoires	Parties	Kills	Deaths	Assists	Win rate (%)	KDA Ratio
1	Zed	10209	21948	8.28	6.39	5.57	46.51	9.15
2	Poppy	3057	6582	4.24	5.19	8.01	46.44	5.78
3	LeBlanc	4939	10642	7.77	5.77	5.98	46.41	8.81
4	Jhin	6436	13911	6.91	5.87	8.36	46.27	8.34
5	Urgot	470	1021	5.59	6.02	6.31	46.03	6.64
6	Rakan	9946	21616	1.46	5.46	13.85	46.01	4.00
7	Rengar	3706	8121	6.87	6.51	6.20	45.63	7.82
8	Ezreal	12951	28397	6.59	5.39	7.85	45.61	8.05
9	Azir	1391	3166	5.77	6.57	6.47	43.94	6.75
10	Ryze	2275	5737	5.26	6.10	5.98	39.65	6.24

Tableau 2: Les 10 pires champions en terme de taux de victoire

Avoir une idée des personnages les plus efficaces est une bonne chose, mais il y a encore un paramètre précis à prendre en compte qui est la lane. En effet, elle est un élément déterminant puisqu'elle définit grandement de quelle manière le champion va être joué. Dès lors, puisque l'on fait appel à un jeu de combinaison (champion-lane) nous allons considérer un seuil minimal de 100 matches pour éviter des biais.

	Champion	Lane	Victoires	Parties	Kills	Deaths	Assists	Win rate (%)	KDA Ratio
1	Twitch	TOP	115	192	8.60	6.08	6.27	59.90	9.63
2	Brand	TOP	93	158	6.25	6.07	7.22	58.86	7.44
3	Aurelion Sol	JUNGLE	128	224	7.65	5.98	9.17	57.14	9.18
4	Ivern	JUNGLE	4380	7772	2.65	4.07	13.12	56.36	5.87
5	Pantheon	MID	336	599	8.33	5.99	6.34	56.09	9.39
6	Annie	DUO_CARRY	102	182	8.21	6.62	7.92	56.04	9.41
7	Draven	TOP	137	246	6.86	6.67	4.47	55.69	7.53
8	VelKoz	TOP	120	216	6.97	6.24	6.51	55.56	8.01
9	Zilean	MID	910	1656	4.37	4.01	9.50	54.95	6.74
10	Anivia	MID	3757	6936	6.36	4.59	7.29	54.17	7.95

Tableau 3: Les meilleures combinaisons poste-champion en terme de taux de victoire ($>$ 100 parties)

Nous avons donc un détail un peu plus précis des dispostions les plus efficaces. Si l'on compare aux champions qui apparaissaient sur le tableau et graphique précédent seuls Anivia, Pantheon, Ivern, Draven sont encore présents avec le TOP étant la lane la plus représentée. Il est à noter que contrairement à nos assomptions précédentes il n'y a aucun support, du moins en lane, qui n'est joué avec notamment Ivern qui est joué en Jungle. Si le Win rate semble également encore fluctuer autour des 55% et ne pas excéder les 60% il est en moyenne légèrement plus élevé (le TOP 10 était à 52.31 contre 54.17% désormais). De plus le KDA Ratio apparait en moyenne comme beaucoup plus élevé, largement expliqué par l'apparition de champion et rôle/lane plus offensives.

	Champion	Lane	Victoires	Parties	Kills	Deaths	Assists	Win rate (%)	KDA Ratio
1	Rakan	MID	45	148	3.39	3.91	6.78	30.41	5.12
2	Vayne	MID	71	234	4.00	4.10	2.73	30.34	4.66
3	Amumu	MID	35	118	2.21	3.20	3.78	29.66	3.39
4	Jinx	JUNGLE	36	123	3.26	4.55	4.19	29.27	4.18
5	Ashe	JUNGLE	31	110	3.11	4.86	5.11	28.18	4.16
6	Twitch	MID	38	140	5.06	3.51	2.91	27.14	5.89
7	Ashe	MID	26	102	3.47	3.83	4.70	25.49	4.70
8	Caitlyn	MID	63	266	3.36	3.66	3.39	23.68	4.29
9	Jinx	MID	19	124	2.30	2.07	1.94	15.32	3.23
10	Thresh	MID	9	161	0.61	0.83	0.96	5.59	1.78

Tableau 4: Les pires combinaisons poste-champion en terme de taux de victoire (> 100 parties)

Enfin, concernant le bas du classement une tendance semble se dessiner sur la lane du milieu surreprésentée. Les taux de victoires atteignent difficilement les 30% et descendent jusqu'à 5.59% pour Thresh lorsqu'il est joué au milieu. Avec seulement 9 victoires enregistrées sur 161 parties, elle se place comme étant la pire combinaison possible. Lorsque l'on s'intéresse au KDA Ratio c'est le même constat, alors qu'il fluctuait en moyenne entre environ 6 et 9 au global sur les précédents tableaux (et ce même dans les 10 pires champions) il est ici compris entre 1.78 et 5.89 pour des lanes qui sont pourtant essentiellement offensives. Même pour ce qui est du rôle intrinsèque des champions, seul Thresh et Rakan sont des support à l'origine et peuvent être considérés comme hors de position. Les champions comme Ivern, Sona, Janna en tenant compte de leur KDA Ratio (voir tableau 1) se placeraient ainsi dans ce classement comme parmi les meilleurs au milieu de personnages offensifs.

2 L'apprentissage non supervisé

Comme nous le savons généralement, les algorithmes d'apprentissage sont essentiellement décrits par deux groupes principaux : l'apprentissage supervisé et l'apprentissage non supervisé. Cette section du rapport est consacrée à l'apprentissage non supervisé. L'objectif de cette approche est de trouver une structure dans des données non étiquetées sans cibles spécifiques. Avec ce type d'algorithme, nous aimerions trouver des sous-groupes homogènes au sein d'une population appelée clusters, ainsi que tenter de trouver des modèles dans les caractéristiques des données, ce qui est plus applicable en cas de grand nombre de variables (features). Une grande partie de cette analyse implique la construction de clusters. Voici les types de clustering les plus courants de Machine Learning :

- Classification hiérarchique
- K-means clustering
- K-NN (k voisins les plus proches)
- Analyse des composants principaux
- Décomposition en valeurs singulières
- Analyse en composantes indépendante

Les arbres de décision sont une méthode graphique pour représenter les choix et leurs conséquences. Il s'agit d'une technique d'exploration de données et d'apprentissage automatique populaire. C'est un type d'algorithme d'apprentissage supervisé et peut être utilisé pour la régression ainsi que pour les problèmes de classification. Nous utiliserons cette méthode afin de pouvoir combiner des algorithmes d'apprentissage non supervisé dans notre étude.

L'arbre de décision présenté (Figure 5) explore le taux de victoire en fonction de la position ajustée, des éliminations, des décès et des nombres d'assistances pour chaque personnage. Dans l'ensemble, nous pouvons voir que le taux de victoire pour toutes les branches d'arbre calculées ne dépasse jamais 50%, le taux de victoire le plus élevé pour les joueurs correspondant à un plus grand nombre de matchs joués, ce qui est logique. Quant aux positions de jeu, elles n'ont pas un impact important sur le taux de victoire, en particulier pour les personnages avec moins de 241 matchs joués.

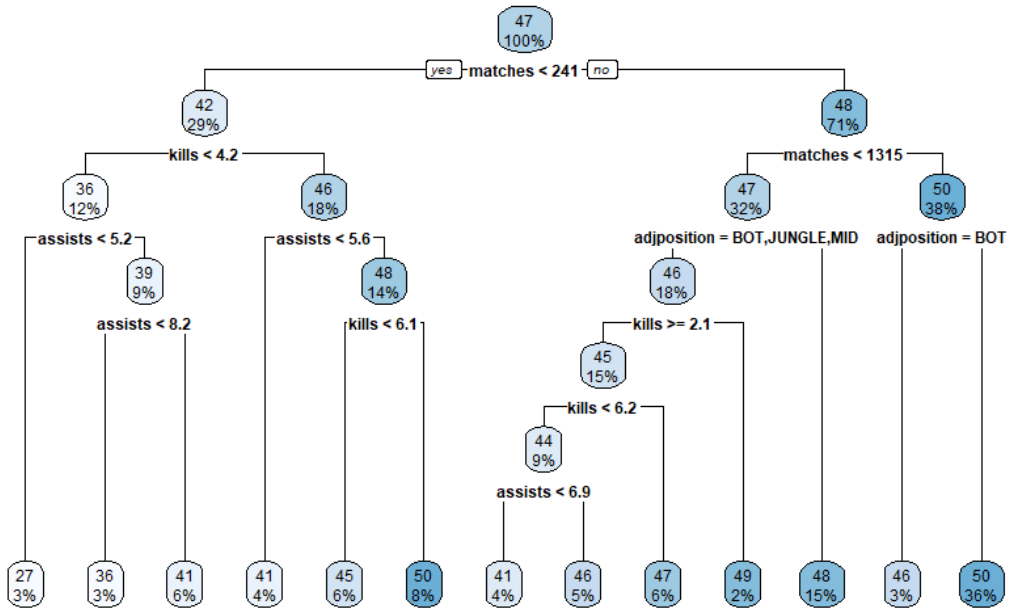


FIGURE 5 – *Arbre de décision pour les statistiques de taux de victoire*

Le clustering hiérarchique est un algorithme qui construit une hiérarchie de clusters. Cela commence par toutes les données qui sont affectées à un cluster qui leur est propre. Il est généralement utilisé lorsque le nombre de clusters n'est pas connu à l'avance. Ici, nous avons utilisé la méthode Hierarchical Clustering on Principal Components car elle nous permet de combiner les trois méthodes standard utilisées dans les analyses de données multivariées :

— Méthodes des composants principaux (comme PCA, MCA)
— Classification hiérarchique
— Clustering de partitionnement, en particulier la méthode k-means

Comme nous avons à la fois des variables catégorielles et numériques, nous effectuons d'abord un prétraitement des données afin d'effectuer une analyse HCPC. Afin d'effectuer une analyse de clustering sur des données catégorielles, l'analyse des correspondances et

l'analyse des correspondances multiples (MCA, pour l'analyse de variables catégorielles multidimensionnelles) peuvent être utilisées pour transformer des variables catégorielles en un ensemble de quelques variables continues (les composantes principales). L'analyse typologique peut ensuite être appliquée aux résultats. Dans ce cas, cette méthode peut être considérée comme des étapes de pré-traitement qui permettent de calculer le clustering sur des données catégorielles. Dans la situation où nous avons un ensemble de données multidimensionnel contenant plusieurs variables continues, l'analyse en composantes principales (ACP) peut être utilisée pour réduire la dimension des données en quelques variables continues contenant les informations les plus importantes dans les données. Ensuite, vous pouvez effectuer une analyse de cluster sur les résultats de l'ACP. L'étape PCA peut être considérée comme une étape de débruitage qui peut conduire à un clustering plus stable. Cela peut être très utile si vous avez un grand ensemble de données avec plusieurs variables.

L'algorithme de la méthode HCPC, tel qu'implémenté dans le peut être résumé comme suit :

- Calculer les méthodes des composantes principales : PCA, MCA selon les types de variables dans le jeu de données et la structure du jeu de données.
- Calculer le clustering hiérarchique : le clustering hiérarchique est effectué en utilisant le critère de Ward sur les composants principaux sélectionnés. Le critère de Ward est utilisé dans le clustering hiérarchique car il est basé sur la variance multidimensionnelle comme l'analyse en composantes principales.
- Choisir le nombre de clusters en fonction de l'arbre hiérarchique : Un premier partitionnement est effectué en coupant l'arbre hiérarchique.
- Effectuer un clustering K-means pour améliorer la partition initiale obtenue à partir du clustering hiérarchique. La solution finale de partitionnement, obtenue après consolidation avec k-means, peut être (légèrement) différente de celle obtenue avec le clustering hiérarchique.

Nous commençons par recalculer l'analyse en composantes principales (ACP). Il est possible de visualiser les individus sur la carte des composantes principales et de colorier les individus selon le cluster auquel ils appartiennent (Figure 6). La fonction fviz_cluster() peut être utilisée pour visualiser des clusters individuels.

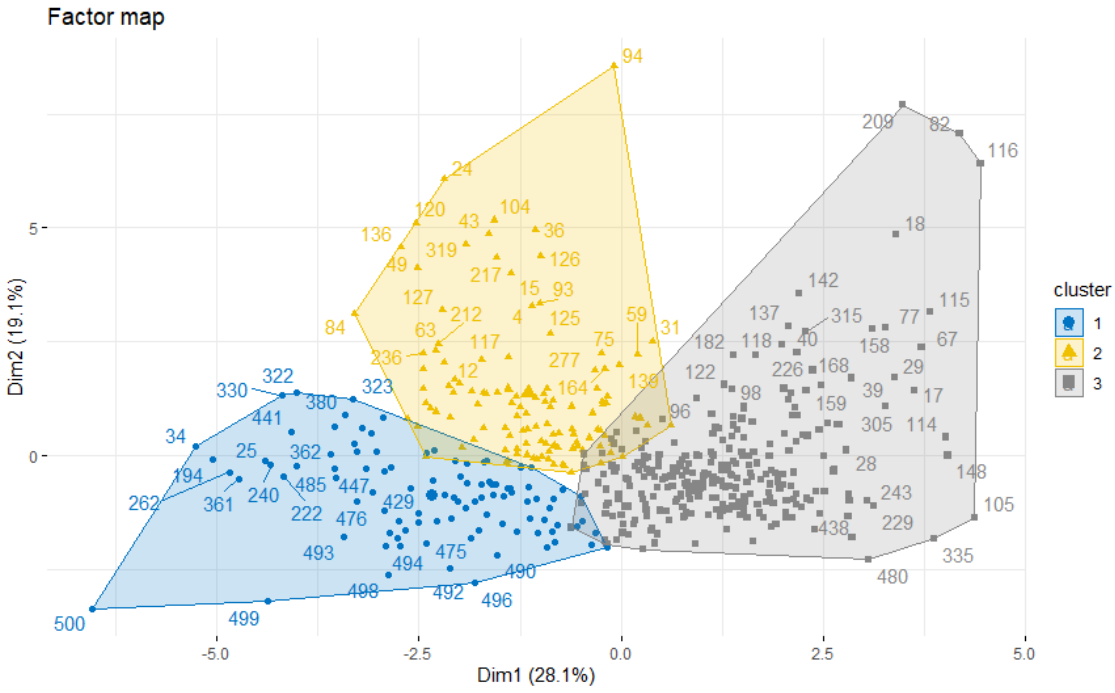


FIGURE 6 – *Carte factorielle par clusters*

Ensuite, nous pouvons également dessiner un graphique en trois dimensions combinant le clustering hiérarchique et la carte factorielle à l'aide de la fonction de base R (Figure 7).

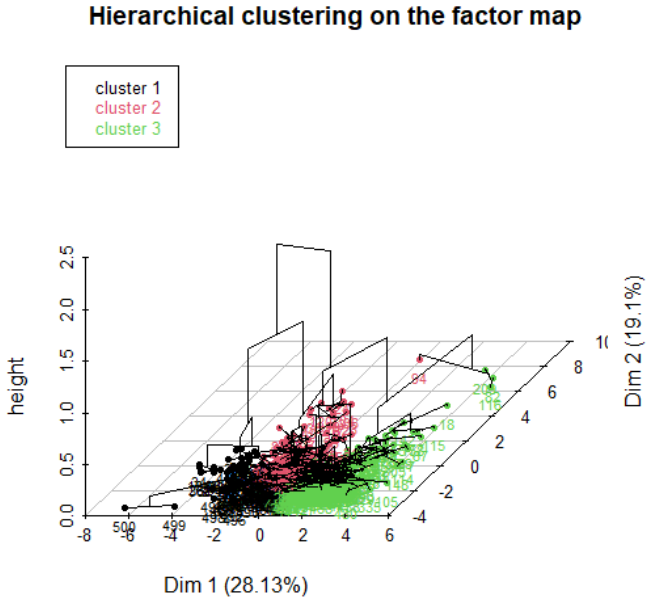


FIGURE 7 – *Regroupement hiérarchique par clusters sur la carte factorielle*

Nous procédons à l'analyse des variables quantitatives qui décrivent le plus chaque cluster. Cette information peut être utile pour interpréter l'association de chaque variable avec un cluster spécifique (Figure 8).

```
> res.hcpc$desc.var$quanti
$`1`
           v.test Mean in category  Overall mean sd in category   Overall sd      p.value
Dim.4    9.796287        0.6987522 -3.782391e-17      0.4764132    0.7867051 1.168001e-22
Dim.5    3.331341        0.2361272 -9.424753e-18      0.4963549    0.7817669 8.642876e-04
Dim.2   -4.388226       -0.3360388 -8.374551e-17      0.6731657    0.8445994 1.142792e-05
matches -4.770592      628.2448980  3.643948e+03    787.2851837 6972.1481331 1.836854e-06
win     -4.833980      273.2346939  1.824450e+03    362.8206900 3539.3014101 1.338303e-06
assists -7.030191        5.9325652  7.487158e+00      2.2323494    2.4389326 2.062504e-12
Dim.1   -8.799663       -0.6851858 -1.556741e-16      0.8299758    0.8588011 1.372278e-18
kills   -11.032961       3.4423785  5.383688e+00      1.3808158    1.9406755 2.649954e-28
KDA     -12.763535       4.8170339  6.727848e+00      1.0397107    1.6511928 2.620332e-37
deaths  -13.248346       4.4886063  5.731251e+00      1.2238687    1.0345123 4.612589e-40
win_rate -13.300565     39.4635277  4.661068e+01      7.3667477    5.9267010 2.297257e-40

$`2`
           v.test Mean in category  Overall mean sd in category Overall sd      p.value
assists  15.877996       10.6063055  7.487158e+00      2.0607249  2.4389326 9.000813e-57
win_rate  2.645576       47.8735901  4.661068e+01      3.7291991  5.9267010 8.155203e-03
Dim.5    -2.826800       -0.1779970 -9.424753e-18      0.6077047  0.7817669 4.701569e-03
Dim.4    -6.917946       -0.4383586 -3.782391e-17      0.7941160  0.7867051 4.582394e-12
KDA      -8.367920        5.6149494  6.727848e+00      0.8529020  1.6511928 5.864423e-17
kills   -10.579697        3.7299501  5.383688e+00      1.1579077  1.9406755 3.701515e-26
Dim.2   -13.739600       -0.9346850 -8.374551e-17      0.5578864  0.8445994 5.879956e-43

$`3`
           v.test Mean in category  Overall mean sd in category   Overall sd      p.value
kills    17.910635        6.7406922  5.383688e+00      1.0665073    1.9406755 9.741787e-72
KDA      17.401533        7.8496154  6.727848e+00      1.0449431    1.6511928 8.032473e-68
Dim.2    15.294355        0.5043121 -8.374551e-17      0.5602874    0.8445994 8.338250e-53
deaths   10.190556        6.1428280  5.731251e+00      0.6077695    1.0345123 2.185108e-24
win_rate  8.391033       48.5522145  4.661068e+01      3.9086145    5.9267010 4.818882e-17
Dim.1     5.722443        0.1918631 -1.556741e-16      0.8784266    0.8588011 1.050033e-08
win       3.314167     2282.3908451  1.824450e+03   4013.4757546 3539.3014101 9.191656e-04
matches   3.303135     4543.0528169  3.643948e+03   7926.2738724 6972.1481331 9.561045e-04
assists  -7.976884        6.7276183  7.487158e+00      1.2390557    2.4389326 1.500735e-15
```

FIGURE 8 – *l'analyse des variables quantitatives*

Enfin, des individus représentatifs de chaque cluster peuvent être extraits. Pour chaque cluster, les 5 individus les plus proches du centre du cluster sont affichés. La distance entre chaque individu et le centre du cluster est fournie (Figure 9).

```
> res.hcpc$desc.ind$para
Cluster: 1
     424      429      447      423      436
1.155476 1.165041 1.229602 1.286077 1.546234
------------------------------------------------
Cluster: 2
     290      149      117       72      246
1.051774 1.105307 1.170118 1.170945 1.411252
------------------------------------------------
Cluster: 3
     179      253      134      206      250
1.419584 1.451844 1.454484 1.479388 1.520113
```

FIGURE 9 – *Les individus représentatifs de chaque cluster*

Nous avons décrit comment calculer le clustering hiérarchique sur les composants principaux (HCPC). Cette approche est utile lorsque vous disposez d'un grand ensemble de données contenant des variables continues, une analyse en composantes principales peut être utilisée pour réduire la dimension des données avant l'analyse de classification hiérarchique. Il est également utile lorsque vous disposez d'un ensemble de données contenant des variables catégorielles, une analyse de correspondance multiple peut être utilisée pour transformer les variables catégorielles en quelques composants principaux continus, qui peuvent être utilisés comme entrée de l'analyse de cluster. Nous avons utilisé le package FactoMineR pour calculer le HCPC et le package factoextra R pour une visualisation élégante des données basée sur ggplot2.

Enfin, nous sommes en mesure d'appliquer les informations sur les clusters associés pour chaque observation afin de reconstruire l'arbre de décision (Figure 10). On voit ici que le fait d'être associé au cluster 1 montre un taux de réussite inférieur à 39% contre 48%.

En ce qui concerne l'approche d'apprentissage non supervisé, il est nécessaire de mettre en évidence l'inconvénient de sorte que vous ne pouvez pas obtenir d'informations précises concernant le tri des données, et la sortie car les données utilisées dans l'apprentissage non supervisé sont étiquetées et inconnues. Ainsi, nous résultons en une moindre précision des résultats parce que les données d'entrée ne sont pas connues et non étiquetées par les personnes à l'avance. Cela signifie que la machine doit le faire elle-même. Nous devons donc effectuer une analyse approfondie en interprétant et en étiquetant les classes qui suivent cette classification. Ainsi, plus loin nous procédons avec des méthodes d'apprentissage supervisé.

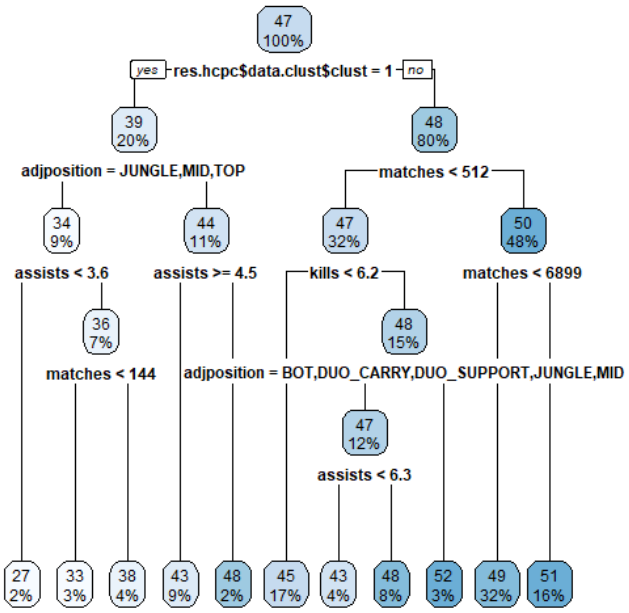


FIGURE 10 – *Arbre de décision pour les statistiques de taux de victoire avec classification par clusters*

3 L'apprentissage supervisé

En utlisant les données dont on dispose, on pourrait par exemple faire la prédiction de la position d'un champion en ayant des informations sur le nombre total de matchs où il a été joué, le taux de parties gagnées, les nombres de kills/deaths/assists (KDA <3), etc. On peut dire qu'un champion avec un haut taux d'assists a de fortes chance d'être un Support, et un autre qui tue beaucoup est probablement un ADC/Bot.

Pour faire cela, nous nous appuyons sur un modèle de forêts aléatoires (Random Forest) qui, entre autres, peut nous aider à classer les variables explicatives en fonction de leurs liens avec la variable à expliquer.

Comme la variable à expliquer est catégorielle, on s'intéresse à l'arbre de classification. Le critère le plus connu de segmentation étant l'indice d'impureté de Gini, notamment utilisée par l'algorithme CART.

Le concept de pureté fait référence au caractère discriminant de la séparation effectuée par un noeud. En clair, une séparation est dite pure quand chaque partie post-split contient des éléments d'une même classe. À l'inverse, le maximum d'impureté est atteint lorsque chaque séparation contient la même probabilité d'éléments de chaque classe. Imaginons un dataset contenant AABB — la segmentation la plus pure classe en deux groupes AA et BB, et la plus impure en AB et AB. Dans le cas du random forest, il s'agit donc de trouver la segmentation qui donne des résultats le plus purs possibles.

Avec une Random Forest nous construisons une « forêt d'arbres » (donc, plusieurs arbres de décision), de manière aléatoire. C'est pour cela que l'on parle de « ensemble method » pour qualifier cette approche : une multitude de modèles « faibles » sont combinés pour créer un modèle robuste.

Une fois ce modèle construit, les données nouvelles sont « lancées » dans tous les arbres, qui votent pour la classe de sortie : pour une régression, en faisant une moyenne, pour une classification en votant à la majorité.

Après un nettoyage de données, on sépare le dataframe en deux set : train, test. Ensuite on applique notre modèle en précisant comme Target la variable "adjposition" représentant la position du champion.

```
randomForest(formula = adjposition ~ ., data = train, ntree = 20,      mtry = 6, importance = TRUE, na.action = na.omit)
               Type of random forest: classification
                     Number of trees: 20
No. of variables tried at each split: 5

        OOB estimate of  error rate: 48%
Confusion matrix:
            BOT DUO_CARRY DUO_SUPPORT JUNGLE MID TOP class.error
BOT          26         0           3      3   4   4   0.3500000
DUO_CARRY     0        15           0      9   6   2   0.5312500
DUO_SUPPORT   3         0          36      7   1   3   0.2800000
JUNGLE        9         8           9     26  12  19   0.6867470
MID           1         6           0     11  43  12   0.4109589
TOP           6         0           4     11  15  36   0.5000000
```

Figure 11 – Description du modèle Random Forest

On voit affiché le type d'arbre (ici classification), des paramètres, l'OOB (Out Of Bag) error rate, et la matrice de confusion.

Chaque arbre est entraîné sur une partie des données, qui est considérée comme « in-bag ». Ce qui permet à chaque arbre, une fois construit, d'estimer son taux d'erreur sur les

données qu'il a laissé « out-of-bag » : plus ce taux est faible, plus le modèle est juste. Le graphique suivant montre un histogramme à travers lequel on peut voir le nombre de fois qu'un individu a été laissé « out of bag » :

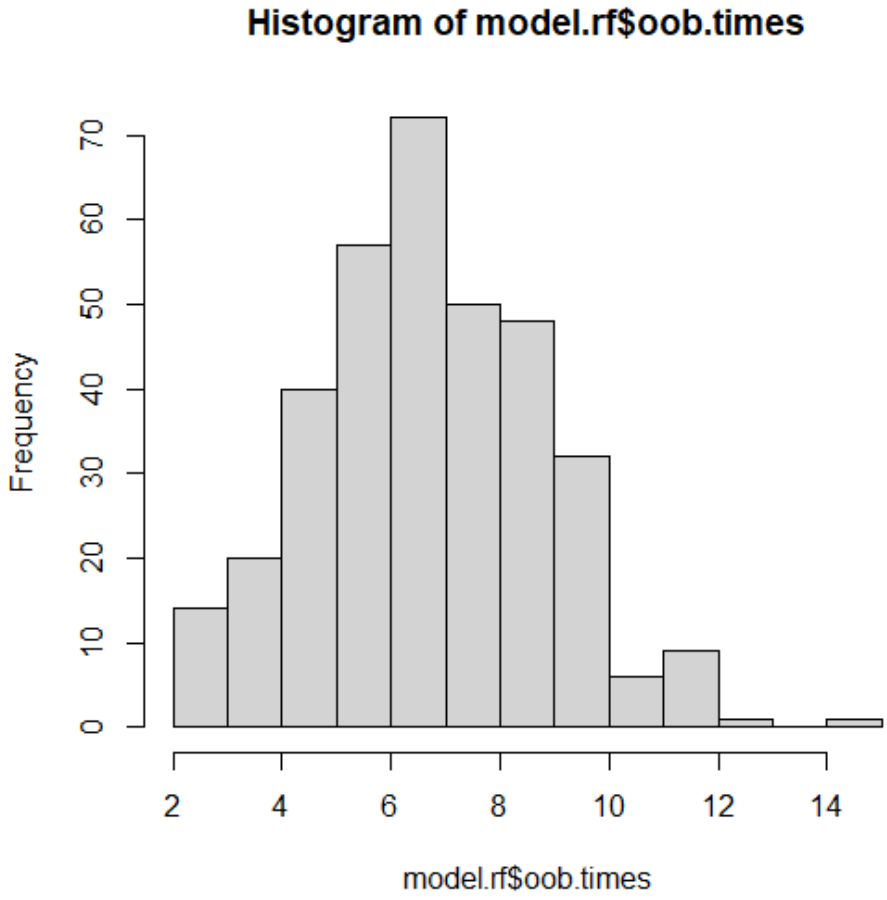


FIGURE 12 – Histogramme OOB

Précédemment obtenue, la matrice de confusion (en colonne, la prédiction de l'algorithme, et en ligne la modalité réelle) nous montre que 36 individus de la calsse réelle "TOP" ont été affectés à la bonne classe. La diagonale représente le nombre de bien classés pour toutes les modalités. Juste à côté, le class.error indique la proportion de mauvais résultats. On comprend aussi que la classe la moins bien prédite est celle de "JUNGLE", et la mieux prédite est "SUPPORT"

On peut également s'intéresser à l'élément "votes" du modèle :

	BOT	DUO_CARRY	DUO_SUPPORT	JUNGLE	MID	TOP
1	0.0000000	0.0000000	0.0000000	0.2857143	0.2857143	0.4285714
2	0.0000000	0.3333333	0.0000000	0.1111111	0.4444444	0.1111111
3	0.2857143	0.0000000	0.5714286	0.1428571	0.0000000	0.0000000
4	0.0000000	0.0000000	0.0000000	0.7500000	0.0000000	0.2500000
5	0.0000000	0.1000000	0.0000000	0.1000000	0.3000000	0.5000000

FIGURE 13 – 5 premières lignes des votes du modèle de RF

Ici, on remarque que le premier individu a été classé en MID/JUNGLE dans 28,6% des cas où il était « OOB », et 42,9% des fois en TOP.

Puis, on s'intéresse à "l'importance" du modèle : il s'agit là de la diminution moyenne de l'impureté apportée par chaque variable. Elle est calculée par l'index de Gini : la diminution pour chaque noeud est cumulée, puis une moyenne sur l'ensemble des arbres est effectuée. Cet indicateur se trouve ensuite dans le modèle :

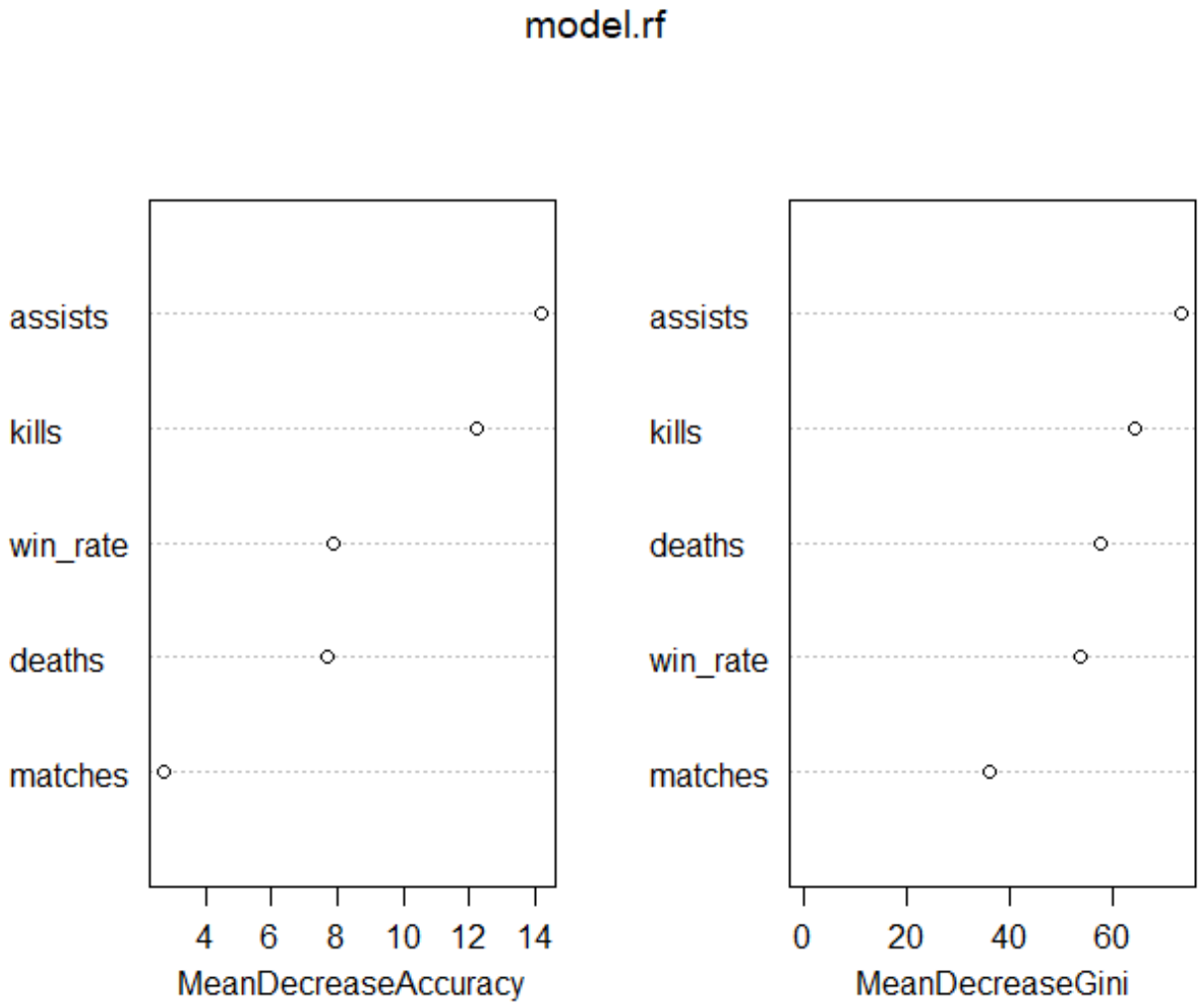


FIGURE 14 – Importance des variables dans le modèle RF

Dans le modèle que l'on a calculé, les 2 critères qui comptent le plus pour distinguer les positions des champions sont le taux d'assists et le taux de kills.

Pour une meilleure prédiction, on peut changer certains paramètres du modèle, notamment le "mtry", qui désigne le nombre de variables testées à chaque split. La valeur par défaut étant la racine carrée du nombre de variables. On réalise ainsi le benchmark (librairie : microbenchmark) qui suit (avec mtry = 1, puis 2, puis 3, puis 4) :

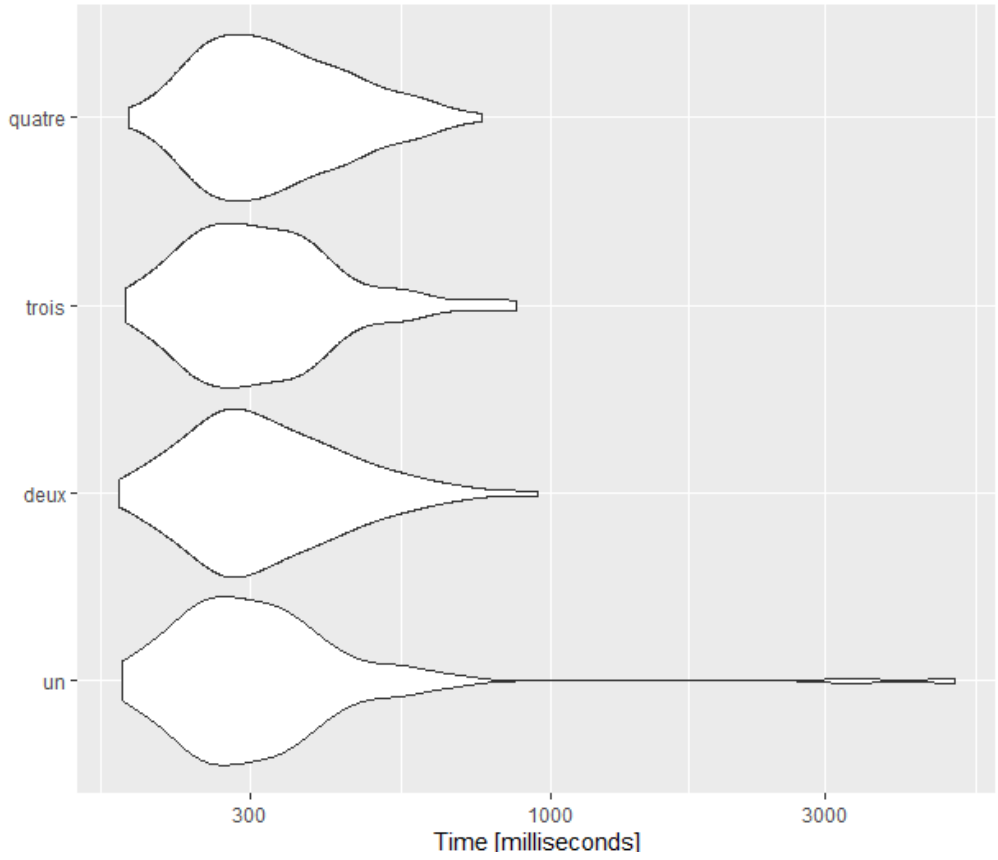


FIGURE 15 – Benchmark de performance du modèle en modifiant "mtry"

On réalise aussi un autre benchmark de perfomance en demander au modèle d'évaluer, ou pas, l'importance des variables :

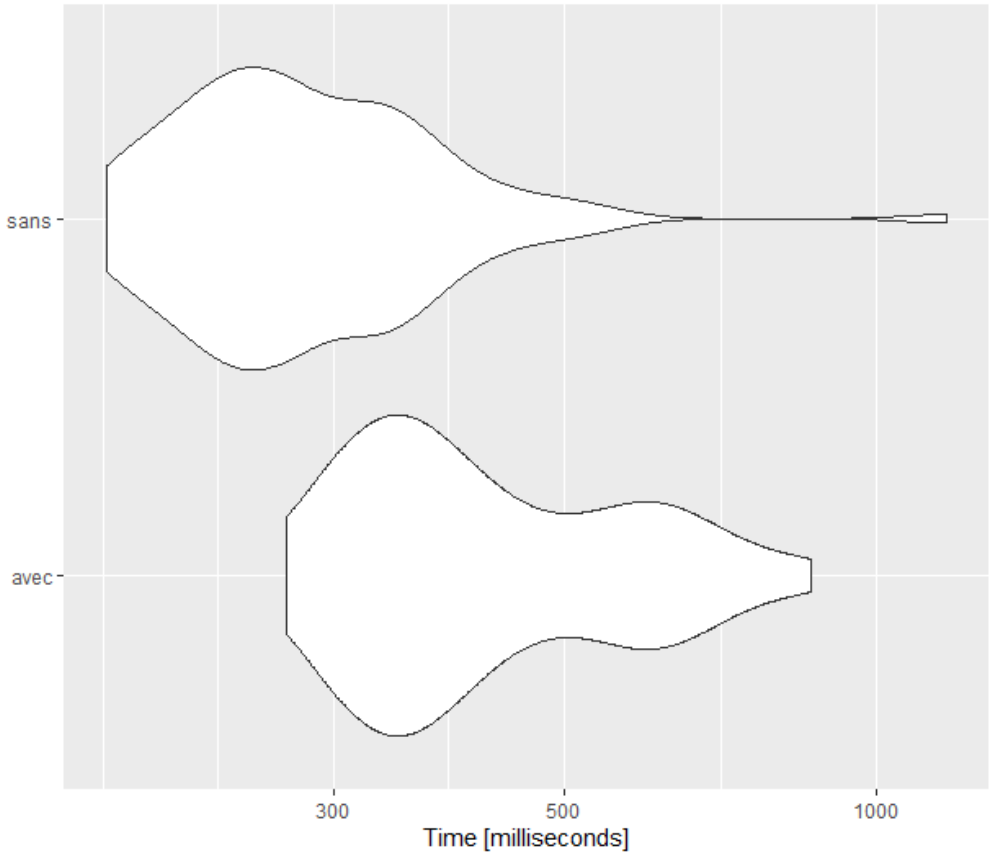


FIGURE 16 – Benchmark de performance du modèle en modifiant "importance"

Enfin, on teste l'utlisation de CARET (temps d'exécution très long) (Classification And REgression Training), qui offre une utilisation personnalisable de plusieurs algorithmes de machine learning : (sans préciser le nombre d'arbres, c'est 1 par défaut)

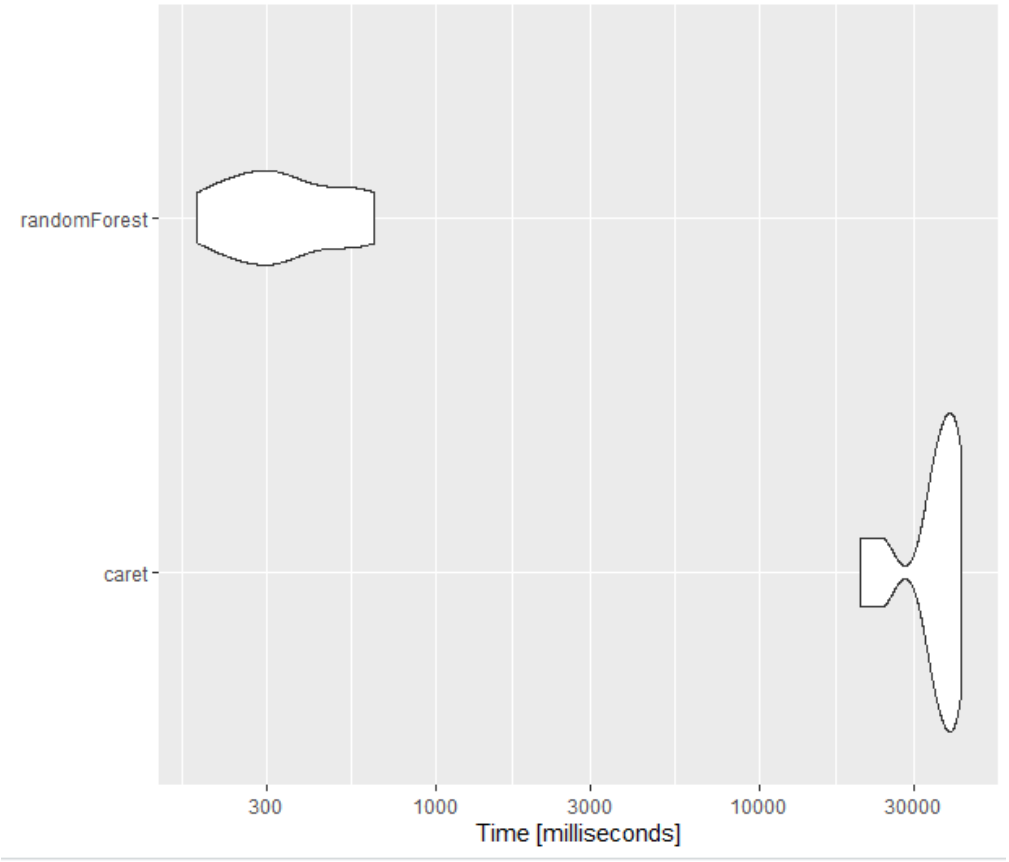


FIGURE 17 – Benchmark de performance du modèle, entre CARET, et RANDOMFOREST

Toujours en utilisant la fonction train de la librairie CARET, on a tenté de faire du boosting avec XGBoost, mais l'installation de la librairie sur RStudio a rencontré des bugs incompréhensibles. On a quand même pu l'installer, mais par la suite on arrivait pas à faire tourner le modèle ; on avait des warning qu'on a pas pu gérer. Sur certains forums, d'autres personnes ont rapporté le même problème, et aucune solution n'a été trouvée pour l'instant :

```
[14:32:08] WARNING: amalgamation/../src/c_api/c_api.cc:718: `ntree_limit` is deprecated, use `iteration_range` instead.
[14:32:08] WARNING: amalgamation/../src/c_api/c_api.cc:718: `ntree_limit` is deprecated, use `iteration_range` instead.
[14:32:10] WARNING: amalgamation/../src/c_api/c_api.cc:718: `ntree_limit` is deprecated, use `iteration_range` instead.
[14:32:10] WARNING: amalgamation/../src/c_api/c_api.cc:718: `ntree_limit` is deprecated, use `iteration_range` instead.
[14:32:11] WARNING: amalgamation/../src/c_api/c_api.cc:718: `ntree_limit` is deprecated, use `iteration_range` instead.
[14:32:11] WARNING: amalgamation/../src/c_api/c_api.cc:718: `ntree_limit` is deprecated, use `iteration_range` instead.
[14:32:13] WARNING: amalgamation/../src/c_api/c_api.cc:718: `ntree_limit` is deprecated, use `iteration_range` instead.
[14:32:13] WARNING: amalgamation/../src/c_api/c_api.cc:718: `ntree_limit` is deprecated, use `iteration_range` instead.
[14:32:15] WARNING: amalgamation/../src/c_api/c_api.cc:718: `ntree_limit` is deprecated, use `iteration_range` instead.
[14:32:15] WARNING: amalgamation/../src/c_api/c_api.cc:718: `ntree_limit` is deprecated, use `iteration_range` instead.
```

FIGURE 18 – Warning en faisant tourner un modèle XGBoost

Néanmoins, pour reprendre tout ce qui a été vu dans le cadre de ce cours, nous avons fait du bagging (en utilisant la libraire "ipred"), toujours pour prédire la même chose :

```
Confusion Matrix and Statistics


bag1          BOT DUO_CARRY DUO_SUPPORT JUNGLE MID TOP
  BOT          15         0           3      4   0   1
  DUO_CARRY     1         8           0      7   3   2
  DUO_SUPPORT   1         0          19      0   0   0
  JUNGLE        1         2           2     17  11   5
  MID           0         0           0      4  17   2
  TOP           1         0           0      3   3  18

Overall Statistics

               Accuracy : 0.6267
                 95% CI : (0.544, 0.7042)
    No Information Rate : 0.2333
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5474

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: BOT Class: DUO_CARRY Class: DUO_SUPPORT Class: JUNGLE Class: MID Class: TOP
Sensitivity              0.7895          0.80000             0.7917        0.4857     0.5000     0.6429
Specificity              0.9389          0.90714             0.9921        0.8174     0.9483     0.9426
Pos Pred Value           0.6522          0.38095             0.9500        0.4474     0.7391     0.7200
Neg Pred Value           0.9685          0.98450             0.9615        0.8393     0.8661     0.9200
Prevalence               0.1267          0.06667             0.1600        0.2333     0.2267     0.1867
Detection Rate           0.1000          0.05333             0.1267        0.1133     0.1133     0.1200
Detection Prevalence     0.1533          0.14000             0.1333        0.2533     0.1533     0.1667
Balanced Accuracy        0.8642          0.85357             0.8919        0.6516     0.7241     0.7927
```

FIGURE 19 – Performance du modèle de bagging sur la target "adjposition"

Pour rappel, La sensibilité est définie comme la proportion de résultats positifs par rapport au nombre d'échantillons réellement positifs. Ici, les positions qu'on prédit le plus souvent correctement sont BOT/CARRY/SUPORT.

Conclusion

Après une analyse exploratoire des données, nous avons réalisé un apprentissage non supervisé en nous intéressant à des clusters de champions, puis une un apprentissage supervisé en appliquant des algorithmes tels que que le Random Forest ou le Bagging pour prédire la position d'un champion.

En guise de conclusion, chaque personnage est doté de ses atouts et ses faiblesses. Celui qui fait le plus de kills n'est pas forcément le meilleur, car par exemple un champion en support sera là pour aider les autres en les soignant ou bloquant l'équipe adverse, et donc il ne fera pas nécessairement de kills.

Les données qu'on a pu trouver sont intéressantes, et nous pensons qu'il est possible et qu'il serait même rentable de faire une analyse encore plus poussée afin de maximiser ses chances de gagner, notamment en rajoutant des données sur les objets/potions que peuvent détenir les champions pour amélioer leurs statistiques.