

École polytechnique de Louvain

Sciences des données pour un jeu vidéo

Prédiction d'une partie de League of Legends

Auteur: **Amandine EVRARD**

Promoteur: **Jean-Charles DELVENNE**

Lecteurs: **Siegfried NIJSSEN, Eric PIETTE**

Année académique 2023–2024

Master [120] : ingénieur civil en sciences des données

Remerciements

Je tiens à remercier tout particulièrement Monsieur Delvenne pour son temps, sa patience, son aide et son écoute. Je tiens également à remercier toutes les personnes qui m'ont relue, écoutée et soutenue durant toute la rédaction de ce mémoire. Et pour finir, je voulais aussi vous remercier, vous, lecteurs, de prendre le temps de lire ce mémoire.

Résumé

League of Legends est un jeu compétitif comptant des milliers de joueur·euse·s. Sa compétition mondiale bat chaque années des records en termes d'audience dans le monde du jeu vidéo. Ainsi, tous·tes les joueur·euse·s rêvent de devenir meilleur·e, et notamment, de savoir ce qui influence réellement les résultats d'une partie à leur niveau et si l'on peut déterminer à l'avance les résultats d'une partie, en se basant sur l'historique de chacun·e ainsi que des champion·ne·s choisi·e·s pour la partie en cours. C'est essentiellement pour répondre à ces questions qu'a vu le jour ce travail. Afin de répondre à cette grande question qu'est la prédiction des résultats d'une partie, nous avons recensé un certains nombre de caractéristiques liée à la victoire. Nous avons choisis un échantillon de 200 joueur·euse·s représentatif·ve·s ainsi que leurs 100 dernières parties. Ensuite, nous avons appliqué 5 grands modèles de machine learning: La régression logistique, le K-Nearest Neighbours, le DecisionTree, le RandomForest et le support vector machine. Ces derniers nous ont permis de comprendre effectivement qu'il était possible de déterminer en partie les résultats d'une partie alors même que cette dernière n'avait pas été encore jouée. En faisant tourner les modèles une fois sur l'ensemble de nos joueur·euse·s, et une fois pour chacune des divisions, nous avons pu observer qu'il valait généralement mieux prédire les résultats des parties en se basant uniquement sur des parties du même niveau. Ceci a mis en exergue les différences existantes entre les niveaux de jeu et les comportements des joueur·euse·s évoluant à ces derniers.

De plus, nous avons pu observer que pour tous les niveaux de joueur·euse·s, la caractéristique ayant le plus d'importance est le taux de victoire du·de la joueur·euse sur le·a champion·ne sélectionné·e pour la partie. La deuxième caractéristique apparaissant le plus souvent est le taux de victoire du côté de la carte appelé le côté rouge, et correspondant au haut de la carte de jeu. Les autres caractéristiques ont des importances variées et moindre pour chaque niveau de jeu des joueur·euse·s. Nous pensons qu'il est possible d'affiner ces résultats en prenant plus en compte les rôles tenu par chaque joueur·euse. En conclusion, il est possible de prédire les résultats d'une partie de League of Legends, alors même que cette dernière est encore entrain de charger et en atteignant des niveaux de précisions tournant autour des 68%. Il serait cependant possible d'obtenir des résultats encore plus précis en créant des modèles tournant spécifiquement sur chaque rôle possible dans le jeu.

Abstract

League of Legends is a competitive game with thousands of players. Its global competition breaks audience records every year in the world of video games. Therefore, all players dream of becoming better, and in particular, of knowing what really influences the outcome of a game at their level, and whether it is possible to predict the outcome of a game in advance, based on the history of each player and the champions chosen for the current game. This is essentially why this work was created. In order to answer this important question of predicting game outcomes, we identified a number of characteristics related to victory. We chose a sample of 200 representative players and their last 100 games. Then, we tested 5 major machine learning models: Logistic regression, K- Nearest Neighbors, DecisionTree, RandomForest, and support vector machine. These allowed us to understand whether it was possible to predict the outcome of a game even before it had been played. By running the models once over our players, and once for each division, we found that it was generally better to predict game results based only on games of the same level. This highlighted the differences between levels of play and the behaviors of the players at each level.

Furthermore, we found that for all levels of players, the character trait having the greatest importance is the win rate of the player on the selected champion for the game. The second most frequently appearing characteristic is the win rate on the red side of the map, which corresponds to the top of the game map. Other traits have varying and lesser importance for each level of play. We believe that it is possible to refine these results by taking into account the roles played by each player.

In conclusion, it is possible to predict the outcome of a League of Legends game, even while it is still loading, and achieve levels of precision of around 68%. However, it would be possible to obtain even more precise results by creating models that specifically target each possible role in the game.

Table des matières

Table des matières	4
1 Introduction	2
2 Théorie et état de l'art	4
2.1 Modèles de machines learning	4
2.1.1 Régression logistique	4
2.1.2 DecisionTree	5
2.1.3 RandomForest	7
2.1.4 Bagging	8
2.1.5 K-Nearest Neighbour (KNN)	8
2.1.6 Distance de Minkowski	9
2.1.7 Support Vector Machin (SVM)	9
2.1.8 Calcul de précision	11
2.2 Matrice de corrélation	11
2.2.1 Espérance	11
2.2.2 Écart-type	12
2.3 Matrice de confusion	12
2.3.1 Courbe ROC	12
2.3.2 F1-Score	12
2.4 Définitions	13
2.4.1 E-sport	13
2.4.2 Twitch	13
2.4.3 Score de vision	13
2.5 Étude sur le sujet	13
3 Présentation du jeu League of Legends	14
3.1 Présentation générale du jeu	14
3.2 Mécanique du jeu	14
3.3 Communauté et e-sport	16
3.4 Aspects économiques	17
4 Exploration de la base de données	19
4.1 L'API de Riot	19
4.1.1 RiotWatcher	19
4.2 Échantillon de données	20

5	Analyse des caractéristiques	23
5.1	Présentations des caractéristiques	23
5.2	Matrice de corrélation	26
5.2.1	Sur tout le dataset	26
5.2.2	Matrices de corrélation par division	30
6	Les modèles et leurs résultats	31
6.1	Régression Logistique	32
6.1.1	Analyse sur l'ensemble des données	32
6.1.2	Analyse par division	33
6.2	DecisionTree	37
6.2.1	Analyse sur l'ensemble des données	37
6.2.2	Analyse par division	38
6.3	RandomForest	42
6.3.1	Analyse sur l'ensemble des données	42
6.3.2	Analyse par division	43
6.4	K-Nearest Neighbors (KNN)	46
6.4.1	Analyse sur l'ensemble des données	46
6.4.2	Analyse par division	47
6.5	Support Vector Classification (SVC)	49
6.5.1	Analyse sur l'ensemble des données	49
6.5.2	Analyse par division	51
7	Résultat final et comparaison des méthodes	55
7.1	Présentation des résultats	55
8	Conclusion et perspectives d'ouverture	60
	Bibliographie	62

Chapitre 1

Introduction

Pour commencer ce mémoire, nous vous proposons de parler du jeu qui est le sujet de ce dernier. Il s'agit de "League of Legends". Ce jeu est très populaire et est un de mes favoris. En effet, je ne peut plus compter le nombre d'heure passées à jouer, ou à explorer les histoires de tous les personnages incarnables dans le jeu. En effet, ce jeu comporte de multiples facette. Il y a bien entendu la faille de l'invocateur, qui est le nom de la carte ou se déroulent toutes les parties classées et ou se déroule la majorité du jeu, mais il existe aussi de nombreux modes de jeu différents. Cependant, quel que soit le mode de jeu, il faut incarner un·e champion·ne pour prendre part aux parties. Ces dernier·e·s sont nombreux·ses et ont chacun une histoire, un vécu et s'inscrivent dans un univers bien défini où règne la magie, les pirates et les combats de toutes sortes. Ainsi, au fur et à mesure du temps, je me suis passionnée pour le jeu. Cependant, bien que pleine de bonne volonté, je ne figure pas dans les meilleur·e· joueur·euse·s. Ainsi, après qu'une certaine frustration soit passée, je me suis demandé s'il était possible de prédire qui allait gagner une partie, alors que cette dernière n'était pas encore lancée, mais que les champion·ne·s étaient tout juste choisis par les joueur·euse·s. Les dés étaient-ils pipés? Le classement de l'autrice était-il dû à la malchance ou à l'absence de compétence? C'est au départ sur cette question égoïste que ce mémoire a commencé. Cependant, au fur et à mesure, un autre aspect c'est dessiné. Si nous pouvons prédire les résultats, nous pouvons aussi savoir ce qui influence la victoire pour chacun·e, et ainsi aider les joueur·euse·s à savoir sur quels points se concentrer afin de s'améliorer.

Ainsi, afin de présenter le résultat à cette grande question qu'est la prédiction du résultat d'une partie de League of Legends, nous avons découpé la synthèse de nos travaux. Vous pourrez donc commencer par trouver dans le Chapitre 2 l'ensemble de la théorie, de l'état de l'art et des modèles utilisés dans ce mémoire. Ensuite, nous présenterons plus profondément le jeu, ses mécaniques et ce qui en découle pour les joueurs. Pour continuer, vous pourrez trouver au Chapitre 4 une synthèse de la façon dont nous avons récupéré les informations sur les parties, et comment nous les avons triées et agglomérées. Viendra dans la continuité le Chapitre 5 qui prendra le temps d'analyser individuellement l'ensemble des caractéristiques qui vont servir à réaliser les prédictions, ainsi que leur corrélation avec la caractéristique de la victoire. Au Chapitre 6, vous pourrez trouver tous les modèles que nous avons exploré et leurs résultats. Vous y découvrirez l'importance d'une caractéristique particulière: *win_on_champ*. Nous y verrons aussi que les performances pour les modèles sont similaires mais que le modèle du K-Nearest Neighbour est celui qui performe le moins bien. De ce chapitre découlera le chapitre le plus important de ce mémoire, celui qui nous dira enfin si oui ou non, et comment, on peut prédire l'équipe victorieuse d'une partie de League of Legends, ainsi que sur quelle division de joueur·euse·s il est le plus facile de prédire le comportement.

Enfin, nous terminerons par une conclusion résumant l'ensemble de nos recherches et de ce travail, et nous proposerons certaines piste d'ouverture afin d'améliorer les résultats présentés.

[remerciement](#)

Chapitre 2

Théorie et état de l'art

Dans ce chapitre, nous verrons l'ensemble de la théorie nécessaire à la bonne compréhension du mémoire ci présent. Ce chapitre propose d'abord une revue de l'ensemble des modèles utilisés, suivi par des rappels théoriques statistiques et se termine sur un ensemble de définitions propres à ce mémoire.

2.1 Modèles de machines learning

Nous commencerons par définir ce qu'est un modèle de machine learning. Il s'agit d' *une forme d'intelligence artificielle (IA) qui est axée sur la création de systèmes qui apprennent, ou améliorent leurs performances, en fonction des données qu'ils traitent.*[1] En d'autres termes, il s'agit d'une structure qui permet de généraliser des motifs récurrents à partir de données d'apprentissage afin de faire des prédictions ou des classifications sur de nouvelles données.

Il existe deux type d'apprentissages: supervisé et non-supervisé. L'apprentissage supervisé repose sur des données étiquetées pour entraîner les modèles à prédire des sorties spécifiques, tandis que l'apprentissage non-supervisé cherche à identifier des structures intrinsèques dans les données sans directives explicites sur les sorties attendues. Ainsi, dans le cadre de ce mémoire, nous utiliserons des modèles supervisés. Nous allons développer ces points dans le paragraphe suivant.

2.1.1 Régression logistique

La régression logistique est un modèle linéaire. Commençons par définir ce qu'est un modèle linéaire.

En machine learning, un modèle linéaire est un modèle qui suppose une relation linéaire entre les variables d'entrées et celles de sorties. Il est basé sur l'hypothèse que les données peuvent être représentées par une fonction linéaire qui relie les variables caractéristiques à la variable cible. Mathématiquement, il est représenté par l'équation suivante:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots \beta_n x_n$$

Avec y la variable cible, β_i les coefficients du modèle et x_i les variables caractéristiques. Les modèles linéaire sont largement utilisés dans le domaine de l'apprentissage automatique en raison de leur simplicité, leur interprétabilité et leur efficacité. Ils servent autant à la régression qu'à la classification. Regardons maintenant plus spécifiquement la régression logistique. Il s'agit d'une technique de modélisation statistique utilisée pour prédire la valeur d'une variable binaire. Elle utilise la fonction de probabilité de l'évènement binaire suivante:

$$p(y = 1|x) = \frac{1}{e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n}}$$

Les coefficients β_i peuvent être estimés grâce à différentes techniques d'évaluation. Dans notre cas, c'est le solveur *liblinear* qui est le plus optimal. Il s'agit d'une méthode basée sur la descente de coordonnées [2]. La descente de coordonnées fonctionne en mettant à jour itérativement les coefficients du modèle en se déplaçant le long de chaque composante du vecteur de paramètres. A chaque pas, un seul coefficient est mis à jour, tandis que les autres sont maintenus fixes.

Avantages:

- Facilité d'interprétation des coefficients
- Efficacité même sur de grands ensembles de données

Limitations:

- Sensibilité aux valeurs extrêmes
- Nécessité d'une relation linéaire avec la caractéristique cible
- Biaisé en cas de classes déséquilibrées

2.1.1.1 Régularisation de Lasso

Le principe de la régularisation consiste à simplifier un modèle en pénalisant les valeurs extrêmes des paramètres ou en identifiant les paramètres les plus explicatifs. Pour cela, on définit une métrique qui mesure à la fois les valeurs des paramètres et leur nombre. On préfère ainsi un modèle avec moins de paramètres s'il obtient des résultats similaires à un modèle plus complexe. Le processus de régularisation implique la minimisation simultanée de deux expressions : La somme des résidus quadratiques liés au modèle lui-même et une expression combinant les coefficients du modèle. Pour la régularisation de Lasso, il faut minimiser la somme des valeurs absolues des coefficients, en pondérant par un facteur λ . Ce qui revient à minimiser l'expression suivante:

$$\min_{\beta_0, \beta_1, \dots, \beta_p} \frac{1}{2} \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{i,j} \right)^2 + \lambda \sum_{j=1}^p |\beta_j|$$

avec le vecteur de coefficient β_λ^L .

2.1.2 DecisionTree

Le DecisionTree est un algorithme basé sur un arbre de décision. Chaque nœud de l'arbre correspond à une caractéristique de notre ensemble de données, et chaque branche de l'arbre correspond à une valeur possible pour cette caractéristique. En évaluant le gain associé à chaque division, l'algorithme construit un arbre qui permettra de prédire la valeur recherchée.

Ce processus itératif divise l'espace des caractéristiques en sous-espaces plus petits et plus homogènes, en maximisant la séparation entre les différentes classes ou valeurs cibles. Le résultat final est un arbre de décision qui permet de faire des prédictions.

Regardons maintenant comment les sous-ensembles de l'arbre sont créés.

Notons les données au nœud m représentées par Q_m avec n_m échantillons. Pour chaque candidat

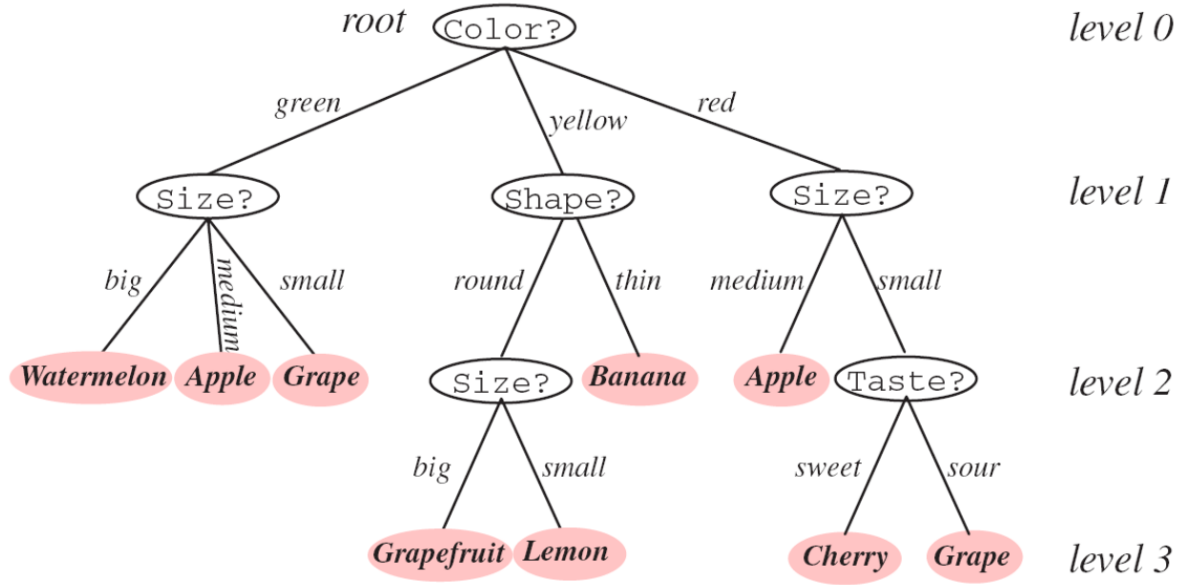


Figure 2.1: Exemple visuels d'arbre de décision [3]

de séparation en ensemble $\theta = (j, t_m)$, avec j la caractéristique et t_m le seuil de partitions en $Q_m^{left}(\theta)$ et $Q_m^{right}(\theta)$. Nous avons

$$Q_m^{left}(\theta) = \{(x, y) | x_j \leq t_m\}$$

$$Q_m^{right}(\theta) = \frac{Q_m}{q_m^{left}(\theta)}$$

La qualité d'une division du noeud m est calculée grâce à une fonction d'impureté $H()$. Cette fonction d'impureté peut prendre deux formes différentes selon le critère utilisé. Dans le cas du critère *gini*, pour l'impureté de Gini, nous avons $H(Q_m) = \sum_k p_{mk}(1 - p_{mk})$. Dans le cas des critères *log_loss* ou *entropy*, nous avons $H(Q_m) = -\sum_k p_{mk} \log(p_{mk})$. Ou $p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k)$ avec k les différentes valeurs possibles pour le noeud m . Le critère *entropy* calcule l'entropie de Shannon [4].

On va chercher à minimiser l'impureté :

$$\theta^* = \operatorname{argmin}_{\theta} G(Q_m, \theta)$$

Avec

$$G(Q_m, \theta) = \frac{n_m^{left}}{n_m} H(Q_m^{left}(\theta)) + \frac{n_m^{right}}{n_m} H(Q_m^{right}(\theta))$$

Il faut itérer sur les sous-ensembles $Q_m^{left}(\theta^*)$ et $Q_m^{right}(\theta^*)$ jusqu'à ce que nous ayons $n_m < \min_{samples}$ ou $n_m = 1$ et que la profondeur maximale autorisée soit donc atteinte.

Avantages:

- Facile à comprendre et à interpréter
- Ne reposent pas sur des hypothèses de linéarité

Limitations:

- Tendance au sur-apprentissage
- Sensibilité aux petits changements
- Inefficacité sur de trop grandes données

2.1.3 RandomForest

Pour le dire simplement, un RandomForest est un ensemble de DecisionTree. Il s'agit d'un méta-estimateur qui utilise différents sous-échantillons de l'ensemble de données dont il fait la moyenne pour améliorer la précision prédictive. Les RandomForest remédient à deux faiblesses des arbres de décision : la tendance à être sensible aux valeurs aberrantes et aux bruits, et le risque de surajustement. La forêt aléatoire combine les DecisionTree, bagging et l'aléatoire. Elle commence par utiliser l'échantillonnage, en générant plusieurs répliques de l'ensemble d'entraînement. Ensuite, elle applique un algorithme de DecisionTree à chaque ensemble de données nouvellement créé, en arrêtant les divisions de nœuds lorsque le nombre d'exemples restants est inférieur à un seuil spécifié. En outre, le caractère aléatoire, inhérent à la phase de bagging, persiste pendant la construction de l'arbre, où les nœuds sont divisés sur la base d'un sous-ensemble de caractéristiques sélectionnées au hasard. Le nombre de caractéristiques prises en compte est déterminé arbitrairement. Les prédictions sont générées en faisant passer l'échantillon non-vu par chaque arbre de décision. Pour les tâches de classification, la classe prédite est généralement déterminée par le vote majoritaire de tous les arbres de décision, tandis que pour les tâches de régression, la prédiction est la moyenne des sorties de tous les arbres de décision.

Avantages:

- Réduction du sur-apprentissage
- Robuste face aux fluctuations de données
- Mesure de l'importance des caractéristiques
- Efficace sur les grands ensembles de données

Limitations:

- Temps de calcul lent
- Mauvaise performance sur les données fortement corrélées
- Biais des arbres individuels

2.1.4 Bagging

Le bagging est la contraction de bootstrap aggregation. Il s'agit d'une méthode d'apprentissage basée sur le principe de la sagesse des foules. [5]

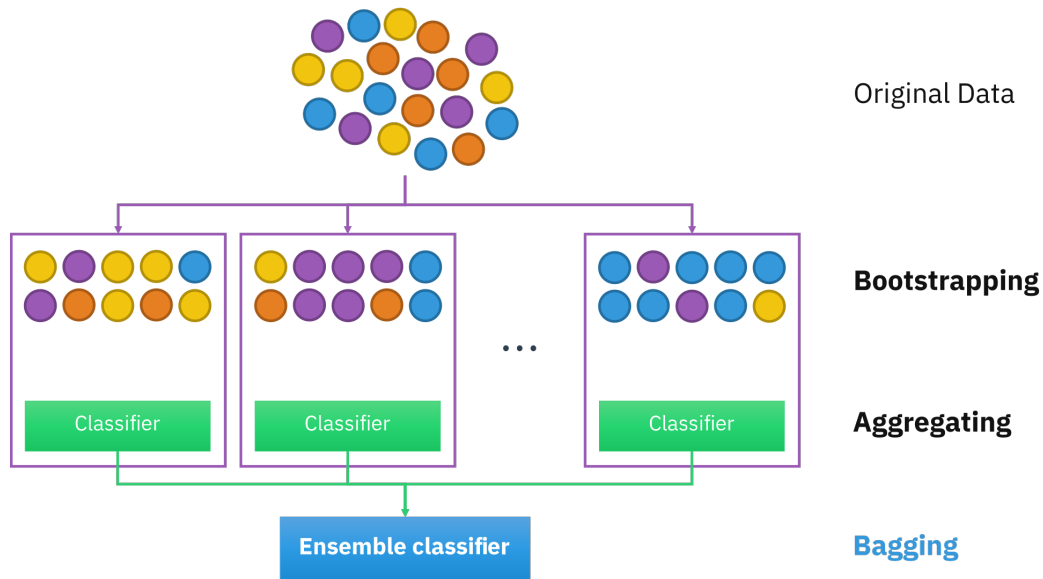


Figure 2.2: Illustration du concept de bagging[5]

Prenons un ensemble d'entraînement D de taille n . Premièrement, le bagging va générer m nouveaux ensembles d'entraînement D_i de taille n' , par échantillonnage uniforme avec remise à partir de D . Ainsi, certaines observations peuvent être répétées dans chaque D_i . Si $n' = n$, alors, l'ensemble D_i , tends vers $1 - \frac{1}{e} \approx 63.2\%$ d'éléments unique.

Ensuite, m modèles sont entraînés sur chaque ensembles d'échantillon. Pour finir, la prédiction est obtenue par vote majoritaire pour la classification des m modèles.

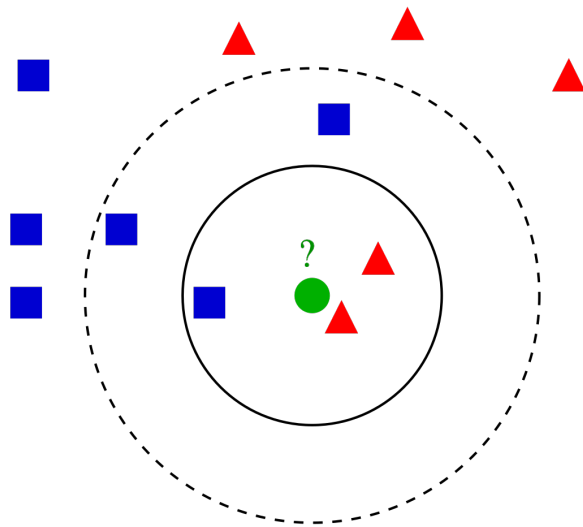
2.1.5 K-Nearest Neighbour (KNN)

Le K-Nearest Neighbour (KNN) est un modèle se servant de la position dans l'espace des points afin de faire une prédiction sur la classe d'un point encore non-vu.

Les exemples d'apprentissage sont représentés sous forme de vecteurs dans un espace multidimensionnel de caractéristiques, chacun étant associé à une étiquette de classe. Pendant la phase d'apprentissage de l'algorithme, seule la sauvegarde des vecteurs de caractéristiques et des étiquettes de classe des échantillons d'apprentissage est effectuée.

Dans la phase de classification, l'utilisateur définit une constante k . Un vecteur non étiqueté, souvent représenté par un point d'interrogation ou de test, est classé en attribuant l'étiquette la plus fréquente parmi les k échantillons d'apprentissage les plus proches de ce point d'interrogation.

La distance euclidienne est une mesure de distance largement utilisée pour les variables continues. Cependant, pour les variables discrètes, comme dans le cas de la classification de textes, une métrique différente peut être plus appropriée, comme la métrique de chevauchement (ou distance de Hamming).



Comment classer le point vert ? Si $k = 3$ (cercle en ligne pleine) il est classé comme un triangle car il y a deux triangles et seulement un carré dans le cercle considéré. Si $k = 5$ (cercle en ligne pointillée) il est classé comme un carré (3 carrés face à deux triangles dans le cercle externe).[6]

Figure 2.3: Illustration du concept de bagging[5]

Avantages:

- Facilité de compréhension
- Adaptabilité aux données non-linéaire
- Robuste aux bruits

Limitations:

- Sensibilité à la dimensionnalité
- Sensibilité à l'échelle

2.1.6 Distance de Minkowski

Généralisation des concepts de distance euclidienne et de Manhattan. Elle se calcule ainsi:

$$D(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}$$

Avec x et y des points dans un espace n -dimensionnel, x_i et y_i les coordonnées de respectivement x et y et p un paramètre strictement positif.[[7]]

2.1.7 Support Vector Machin (SVM)

La machine à vecteurs de support (SVM) est une technique d'apprentissage automatique utilisée pour construire un classificateur. Elle crée un hyperplan qui sépare les classes et qui est aussi éloigné que possible de tous les points d'échantillonnage dans le domaine des caractéristiques. Les points les plus proches de l'hyperplan optimal sont appelés vecteurs de support.[8] L'hyperplan optimal est dénoté $wx^T + b = 0$, avec w le vecteur de poids définissant une direction perpendiculaire à l'hyperplan, x les points situés sur l'hyperplan et b l'ordonnée à l'origine. Deux contraintes doivent être respectées par w et b :

$$wx_i^T + b \geq +1 \text{ if } y_i = +1$$

et

$$wx_i^T + b \leq -1 \text{ if } y_i = -1$$

Ainsi, l'objectif du SVM est de trouver les paramètres de l'hyperplan w et b afin de séparer les données, de respecter les deux contraintes et de maximiser la marge $\frac{2}{\|w\|}$. Pour maximiser la marge, il est possible de minimiser simplement le dénominateur et donc de trouver w et b .

Pour les problèmes plus complexes, il faut introduire un noyau dans l'algorithme. Les noyaux sont des fonctions qui ajoutent des dimensions aux données originales et permettent au SVM de résoudre linéairement le problème résultant. Il améliore également les performances de certains calculs. Les noyaux peuvent être représentés par

$$K(x, y) = \langle f(x) f(y) \rangle$$

Avec K la fonction noyau, et x, y désignant les vecteurs d'entrée à n dimensions, f est la fonction responsable de la mise en correspondance des vecteurs d'entrée de la basse dimension d'origine vers une dimension supérieure, et $\langle x, y \rangle$ désigne le produit scalaire entre x et y . Il vise à démontrer que les fonctions noyaux permettent à l'algorithme de calculer le produit scalaire de deux points de données dans le nouvel espace à haute dimension sans calculer explicitement la cartographie de ces points dans l'espace à haute dimension, ce qui permet d'économiser du temps de calcul.

Avantage:

- Efficacité dans les espaces de grande dimension
- Robustes face au surajustement
- Efficacité sur les problèmes de classification linéaire et non linéaires

Limitation:

- Sensibilité au choix du noyau et aux paramètres de régularisation
- Sensibilité à l'échelle des caractéristiques

2.1.7.1 Hinge loss

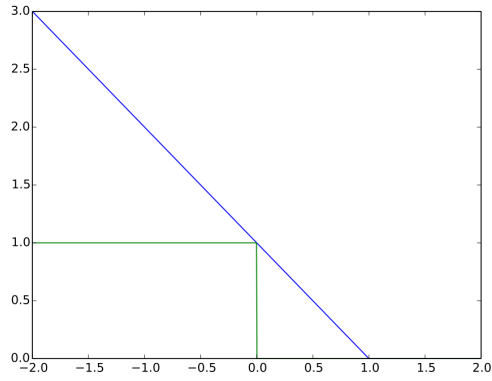


Figure 2.4: L'axe vertical représente la valeur de la perte de la charnière (en bleu) et de la perte zéro-un (en vert) pour $t = 1$, tandis que l'axe horizontal représente la valeur de la prédiction y .

Il s'agit d'une fonction de perte utilisée dans l'entraînement des classifieurs. Pour une sortie attendue $t = \pm 1$ et un classifieur y , la perte de Hinge est définie comme

$$l(y) = \max(0, 1 - t \cdot y)$$

[9]

2.1.8 Calcul de précision

Dans le cadre de ce mémoire, afin de calculer la précision des différents modèles, nous utiliserons la fonction **accuracy_score** de *sklearn*. Cette fonction prend en paramètre deux vecteurs : **y_pred** et **y_true**. Le vecteur **y_pred** correspond aux valeurs prédites par le modèle, tandis que **y_true** correspond aux véritables valeurs que le modèle a essayé de prédire. La fonction retourne la fraction des valeurs bien classées sur le nombre totale de valeur . [10]

2.2 Matrice de corrélation

Une matrice de corrélation est un tableau montrant à quel point les variables sont liées entre elles. *Calculer le coefficient de corrélation entre deux variables numériques revient à chercher et à résumer la liaison qui existe entre les variables à l'aide d'une droite.* [11] Dans le cadre de ce mémoire, nous avons choisi de faire usage des coefficients de corrélation linéaire de Bravais-Pearson. Le coefficient de corrélation r de deux caractéristiques X et Y est le ratio entre la covariance de deux produits et le produit de leurs écarts-type. Ainsi, nous avons:

$$r = \frac{Cov(X, Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}[(X - \mathbb{E}(X))(Y - \mathbb{E}(Y))]}{\sigma_X \sigma_Y} = \frac{\mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y)}{\sigma_X \sigma_Y}$$

avec $\mathbb{E}(X)$, l'espérance de X .

Afin de construire la matrice en elle-même, qui est une matrice carrée, nous calculons ce coefficient pour chaque paire de caractéristiques. Au final, la matrice est donc définie ainsi:

$$r_{i,j} = Cor(X_i, X_j)$$

2.2.1 Espérance

L'espérance $\mathbb{E}(X)$ est la moyenne pondérée sur X .

2.2.2 Écart-type

L'écart-type σ est une mesure de dispersion des données. Il s'agit de la racine carrée de la variance et est donc calculé ainsi:

$$\sigma = \sqrt{\frac{1}{n} \left(\sum_{i=1}^n x_i^2 \right) - \mu^2}$$

avec μ la moyenne sur l'ensemble des données.

2.3 Matrice de confusion

		Predicted classes	
		Negative 0	Positive 1
Actual classes	Negative 0	TN	FP
	Positive 1	FN	TP

La matrice de confusion est un tableau qui contient le nombre de vrais positifs (TP), de vrais négatifs (TN), de faux positifs (FP) et de faux négatifs (FN). [13]

Figure 2.5: Matrice de confusion [12]

2.3.1 Courbe ROC

Une courbe ROC, de l'anglais Receiver Operating Characteristic, est une représentation graphique des performances d'un modèle de classification binaire pour tous les seuils de classification. Elle représente graphiquement le compromis entre le taux de vrais positifs (TPR) et le taux de faux positifs (FPR) à différents seuils de décision, avec le TPR sur l'axe vertical et le FPR sur l'axe horizontal. Le modèle génère des scores de probabilité pour chaque instance, et ces scores sont utilisés pour assigner une classe en fonction d'un seuil ajustable. Pour chaque seuil possible, on calcule le taux de vrais positifs et le taux de faux positifs, où TPR est le ratio des vrais positifs sur le total des positifs réels, et FPR est le ratio des faux positifs sur le total des négatifs réels. Les valeurs de TPR sont tracées sur l'axe vertical et les valeurs de FPR sur l'axe horizontal, formant ainsi la courbe ROC

2.3.1.1 AUC

Il s'agit de l'aire sous la courbe ROC.

Elle est calculée pour fournir un résumé de la performance du modèle, avec une AUC de 100% indiquant une parfaite discrimination et une AUC de 50% indiquant une performance aléatoire.

2.3.2 F1-Score

Il s'agit d'une métrique définissant la précision d'un modèle. Cette métrique permet d'obtenir une précision plus nette lorsque les classes sont mal équilibrées. Dans le cadre de la classification

binaire, elle est calculée ainsi:

$$F_1 = 2 \cdot \frac{\text{précision} \cdot \text{rappel}}{\text{précision} + \text{rappel}}$$

$$\text{précision} = \frac{VP}{VP + FP}$$

$$\text{rappel} = \frac{VP}{VP + FN}$$

Avec VP les vrais positifs, FP les faux positifs et FN les faux négatifs

2.4 Définitions

2.4.1 E-sport

(Anglicisme) (*Jeux vidéo*) *Pratique des jeux vidéo en compétition, à la manière d'un sport.* [14]

2.4.2 Twitch

Plateforme de streaming vidéo en direct.

2.4.3 Score de vision

Le score de vision est calculé grâce aux balises de vision. Les balises de vision permettent de mettre de la vision sur la carte sur des zones où aucun allié·e ne se trouve. Elles ont une durée de vie limitée. Chaque minute de vie d'une balise de vision donne 1 point. Chaque minute de vie restantes à une balise de vision ennemie supprimée donne 1 point. Il existe cependant des conditions altérant ces différents points. Premièrement, si une balise de vision donne la vision sur la même zone qu'un·e allié·e, la balise de vision ne donnera que 75% des gains de visions. Si une balise de vision ne détecte aucun ennemi, la balise ne donnera que 0.5 points. [15]

2.5 Étude sur le sujet

Il existe une grande étude portant sur la prédiction des résultats d'une partie. Elle date de 2021 et est le résultat d'une collaboration de deux universités Américaines: l'université de Floride Central et l'université du Texas à Dallas. Elle se base sur 4 paramètres: les points de maîtrise d'un·e joueur·euse sur le·a champion·ne joué·e, le taux de victoire du·de la champion·ne incarnée, le nombre de parties jouées sur le·a champion·ne, ainsi que le nombre de parties récentes jouées sur le·a champion·ne. En utilisant différents modèles, elle tente de montrer que l'on peut prédire les résultats d'une partie classées de League of Legends.[16]

Chapitre 3

Présentation du jeu League of Legends

3.1 Présentation générale du jeu

League of Legends est l'un des jeux d'arène de bataille (Multiplayer Online Battle Arena, aka MOBA), les plus populaires au monde. Le jeu est créé et édité par la société Riot Games depuis 2009 [17]. Ce jeu vidéo est une référence en terme d'expérience vidéoludique compétitive, basée sur la tactique et la stratégie. League of Legends offre aux joueur·euse·s un terrain de jeu où la coopération d'équipe et la planification sont essentielles pour la victoire. Lors de chaque partie, chaque joueur·euse sélectionne le personnage qu'il incarnera parmi une liste de plus de 165 champion·ne·s disponibles chacun ayant des compétences spécifiques, créant un équilibre délicat entre habilité individuelle et collaboration.

Depuis le lancement du jeu, Riot Games a maintenu une dynamique d'innovation constante en introduisant de nouveaux et nouvelles champion·ne·s, des nouveaux modes de jeu et des mises à jour de contenu. Grâce à cela, League of Legends ne cesse d'agrandir sa communauté et de s'étendre au niveau mondial.

De cette manière, League of Legends a consolidé sa place de leader des MOBA, incarnant un phénomène culturel et une arène compétitive ludique captivant des millions de personnes. Lors de la dernière compétition internationale, League of Legends a établi un nouveau record en devenant l'événement e-sport le plus regardé de tous les temps, attirant une audience de 6 millions de personnes sur la diffusion en direct sur Twitch [18].

Ainsi, League of Legends est bien plus qu'un simple jeu, il incarne un phénomène culturel, une arène compétitive et un laboratoire constant d'innovations vidéoludiques, captivant des millions de joueur·euse·s à travers le monde et laissant une marque indélébile dans l'histoire des jeux en ligne.

3.2 Mécanique du jeu

League of Legends est un jeu de stratégie en ligne qui oppose deux équipes de cinq joueur·euse·s. Chaque participant·e incarne un·e champion·ne doté·e d'aptitudes uniques. Au cours d'une partie, les compétences du·de la champion·ne évoluent à mesure qu'ils progressent de niveau. Pour chaque partie, un·e champion·ne débute au niveau 1 et peut atteindre le niveau maximum de 18.

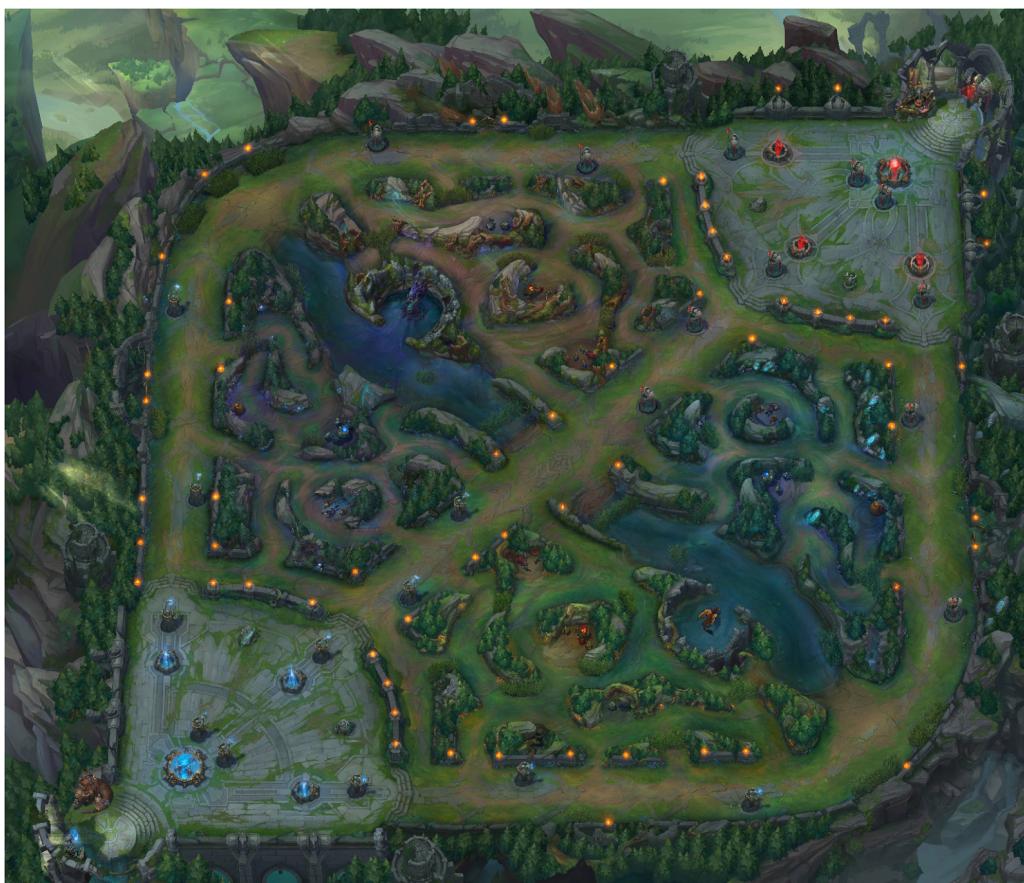


Figure 3.1: Faille de l'invocateur

L'acquisition de l'expérience nécessaire pour monter en niveau s'effectue principalement par l'élimination ou la participation à l'élimination de monstres, de sbires, ou d'adversaires. Parallèlement, les joueur·euse·s accumulent de la monnaie virtuelle, appelée «pièces d'or» (po), à travers ces mêmes actions. Ces po servent ensuite à acheter divers objets au cours de la partie, renforçant ainsi la puissance de leur champion·ne.

Les affrontements se déroulent sur une carte stratégique nommée la «Faille de l'invocateur» représentée à la figure 3.1.

L'objectif ultime de chaque partie est de détruire le Nexus ennemi 3.3, tout en protégeant le sien 3.2. Ces deux Nexus étant à l'opposé l'un de l'autre sur la carte.

Avec une variété impressionnante de plus de 165 champion·ne·s, chaque personnage assume un rôle spécifique et peut être joué de diverses manières sur les différentes voies de la Faille de l'invocateur, les chemins reliant les deux Nexus. Il existe cinq postes distincts, chaque joueur·euse·s occupant un rôle spécifique sur une voie spécifique : Toplaner, Midlaner, ADC, Support et Jungle.

Le·la Toplaner, positionné·e sur la voie du haut, est généralement un·e champion·ne conçu·e pour encaisser les dégâts ennemi·e·s pendant les combats d'équipe. Iel peut aussi être de type bruiser. Il s'agit là de champion·ne·s, mêlant adroitement la défense et l'attaque.

Le·la Midlaner occupe la voie du milieu. Souvent incarné·e par un·e mage infligeant des dégâts magiques ou un·e assassin·e dont le but est d'éliminer les adversaires plus fragiles.

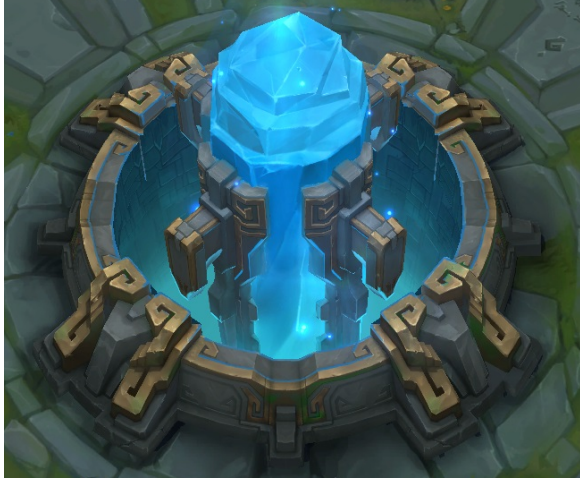


Figure 3.2: Nexus allié

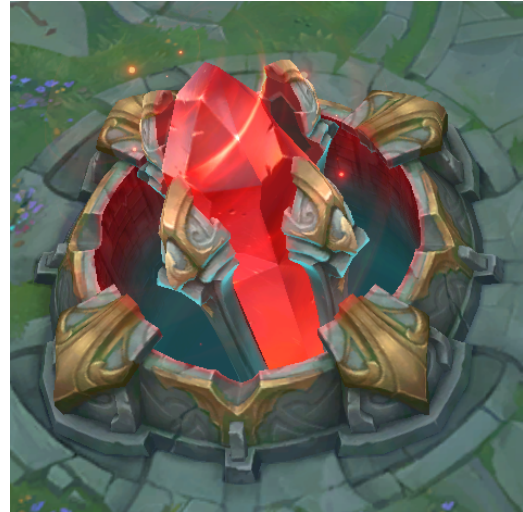


Figure 3.3: Nexus ennemi

L'ADC, pour «Attak Dommage Carry», joue sur la voie du bas avec le-la support. Ce rôle implique un-e champion-ne avec une faible résistance aux dégâts, ainsi que peu de points de vie, mais une capacité significative à infliger des dommages physiques importants. Le-la support, compagnon-ne de l'ADC, a pour mission de protéger son-sa partenaire, d'initier les combats, ou de résoudre des situations difficiles. Il s'agit généralement de champion-ne-s résistant-e-s, appelé-e-s tank, ou de mages doté-e-s de compétences de soin et de bouclier.

Pour finir, le rôle du Jungle est d'explorer la carte afin d'obtenir différents avantages pour le reste de son équipe. Iel peut également intervenir de temps à autre sur les voies afin de donner l'avantage à un membre de son équipe afin de tuer l'adversaire. Il s'agit donc d'un rôle important et polyvalent.

Afin d'aider les joueur-euse-s dans leur mission, une vague de 6 petits monstres appelé sbires, apparaît à chaque nexus, toutes les 30 secondes . Ces derniers ne font qu'avancer en ligne droite sur chaque voie et attaquent le-a premier-e ennemi-e rencontré-e, qu'iel s'agisse d'un-e champion-ne, d'un autre sbire, ou d'une tourelle. Les tourelles sont au nombre de 3 par voies. Elles protègent les champion-ne-s allié-e-s et attaquent tout-e ennemi-e à leurs portée. Il existe également deux tourelles qui protègent le nexus.

Cette mécanique complexe et stratégique, avec ses sbires incessants et ses tourelles défensives, ajoute une couche de profondeur à la dynamique de jeu, nécessitant une coordination minutieuse et une stratégie bien pensée de la part des joueur-euse-s pour naviguer à travers ces éléments clés de la Faille de l'invocateur.

3.3 Communauté et e-sport

La communauté de League of Legends a non seulement été un acteur majeur dans l'évolution du jeu depuis ses débuts en 2009, mais elle a également joué un rôle crucial dans l'ascension fulgurante de l'e-sport de League of Legends, en le propulsant parmi les scènes compétitives les plus prisées à l'échelle mondiale. L'originalité du genre MOBA et le modèle de distribution

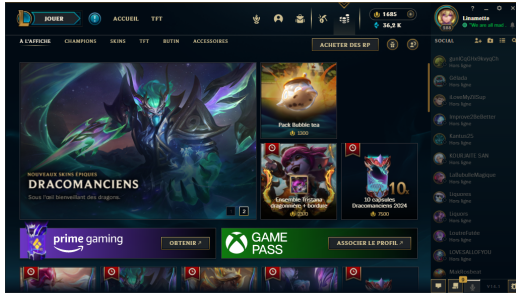


Figure 3.4: Boutique de League of Legends

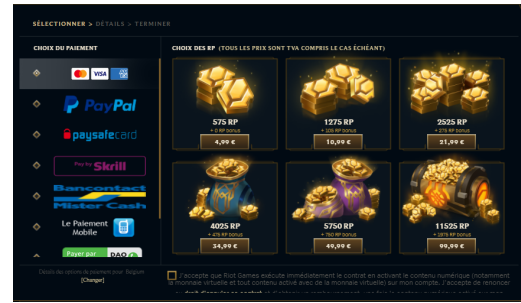


Figure 3.5: Boutique des RP

gratuit ont captivé l'attention, jetant les bases d'une communauté particulièrement dynamique.

En quelques années seulement, League of Legends est passé d'un jeu prometteur à un phénomène mondial, attirant des millions de joueur·euse·s [19]. Cette croissance exponentielle de la base de joueur·euse·s a été alimentée par des mises à jour fréquentes, des événements spéciaux et la création d'une expérience multijoueur immersive. La communauté, quant à elle, s'est rapidement organisée autour d'une multitude d'événements communautaires, allant de simples tournois amateurs à des rassemblements en ligne de grande envergure, tels que les worlds, la compétition mondiale officielle. Ces initiatives ont tissé des liens solides entre les joueur·euse·s, façonnant ainsi une culture communautaire unique.

Les canaux de communication officiels tels que les forums dédiés et des plateformes populaires comme Reddit ont fourni à la communauté un espace vital pour partager des expériences, discuter des mises à jour et s'engager dans des débats passionnés. Simultanément, la scène de création de contenu s'est épanouie avec l'avènement de streamers populaires, de vidéastes créatifs et d'artistes, tous contribuant de manière essentielle à la promotion du jeu.

L'univers e-sport de League of Legends a émergé de modestes tournois amateurs pour se transformer en ligues professionnelles hautement structurées. Les championnats du monde sont devenus des événements majeurs, captivant des millions de téléspectateurs à travers le globe. Les joueur·euse·s professionnel·le·s et les équipes ont émergé comme des icônes influentes, captivant l'attention et contribuant significativement à l'enthousiasme entourant le jeu. Les compétitions de League of Legends ont transcendé les frontières, attirant une audience mondiale et établissant l'e-sport de League of Legends comme une référence incontournable, surpassant de loin d'autres jeux dans ce domaine en termes d'impact et de popularité.

3.4 Aspects économiques

League of Legends offre une expérience de jeu initiale sans frais d'inscription, le téléchargement du jeu étant gratuit. Cependant, pour enrichir l'expérience des joueur·euse·s, des contenus payants sont proposés au sein du lanceur du jeu, accessibles via la boutique 3.4. Ces contenus incluent quatre catégories distinctes : les skins, les champion·ne·s, les butins, et les accessoires.

Les skins, représentant des apparences esthétiques pour les champion·ne·s, sont exclusivement disponibles contre des Riot Points (RP), la monnaie virtuelle payante du jeu 3.5. Bien que les skins n'affectent pas les performances des champion·ne·s, ils offrent une personnalisation visuelle unique et permettent ainsi d'habiller son·sa champion·ne à ses goûts. 3.8.

Les champion·ne·s peuvent être acquis en échange de RP ou d'essence bleue, une monnaie du jeu obtenue en participant à des parties. L'acquisition de champion·ne·s élargit le panel

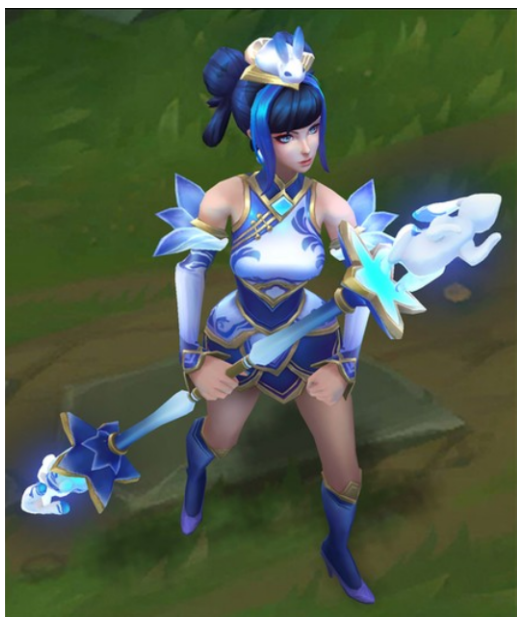


Figure 3.6: Skin : Lux de porcelaine



Figure 3.7: Skin : Lux astro-groove

Figure 3.8: Deux skins de Lux

de choix tactiques et stratégiques pour les joueur·euse·s, car la sélection d'un·e champion·ne pour jouer est limité·e à ceux que le·la joueur·euse possède. Il convient de noter que chaque joueur·euse débute avec un certain nombre de champion·ne·s gratuit·e·s, et le jeu propose un système de rotation hebdomadaire offrant l'accès à 16 champion·ne·s gratuit·e·s pour permettre aux joueur·euse·s de les essayer.

Les butins, composés de coffres et de capsules, sont disponibles exclusivement contre des RP. Ces éléments offrent généralement des skins de manière aléatoire, bien qu'ils puissent également contenir d'autres coffres, des champion·e·s, ou des essences.

Les accessoires comprennent divers éléments tels que des emotes pour communiquer pendant la partie, des skins pour les balises (utilisées pour maintenir la vision d'un endroit sur la carte), des tickets pour participer au Clash (une compétition accessible à tous et se déroulant depuis chez soi), des pages de rune, des icônes d'invocateur, et plus encore. La majorité de ces éléments est disponible à l'achat en RP, bien que certains puissent également être obtenus en échange d'essence bleue. Cette gamme variée d'accessoires offre aux joueur·euse·s la possibilité de personnaliser et d'enrichir leur expérience de jeu selon leurs préférences individuelles.

De surcroît, League of Legends organise périodiquement des événements spéciaux à l'intention de ses joueur·use·s. Au cours de ces événements, les joueur·euse·s ont l'opportunité d'acquérir de la monnaie événementielle, qu'ils peuvent ensuite échanger contre divers contenus en jeu. De manière additionnelle, ces événements offrent la possibilité d'acquérir des passes, disponibles en RP, permettant un accès privilégié à d'avantage de contenus exclusivement liés à l'événement.

Ces événements constituent une initiative dynamique de la part de League of Legends, visant à enrichir l'expérience des joueur·use·s en introduisant des éléments temporaires et des opportunités uniques d'acquisition de contenu, tels que des skins accessibles temporairement uniquement. Les monnaies événementielles et les passes renforcent l'engagement des joueur·euse·s tout en offrant des récompenses exclusives, contribuant ainsi à la diversité et à la vivacité du monde virtuel de League of Legends.

Chapitre 4

Exploration de la base de données

4.1 L'API de Riot

Dans le cadre de ce mémoire, Riot Games nous a autorisé l'accès à son API. Un API est une Interface de Programmation d'Application mise à disposition par Riot Games, permettant aux développeur·euse·s tiers d'accéder à diverses données relatives aux jeux de l'entreprise.

Parmi les sections de l'API, celles revêtant une importance particulière pour la rédaction de ce mémoire sont SUMMONER-V4, MATCH-V5 et LEAGUE-V4.

La section SUMMONER-V4 renferme l'ensemble des informations associées à un·e joueur·euse, constituant ainsi une source cruciale pour l'identification du joueur à travers son puuid, un identifiant unique propre à chaque utilisateur.

La section LEAGUE-V4, quant à elle, fournit des informations sur les différentes divisions des joueur·euse·s. Avec 10 divisions distinctes permettant de classer les joueur·euse·s selon leur niveau de compétence, cette section devient essentielle pour évaluer les performances des joueur·euse·s. Une explication détaillée sur ces divisions sera présentée dans la section dédiée à l'échantillon de données.

La section MATCH-V5 permet l'accès à toutes les informations relatives à un match, comprenant des détails tels que l'équipe victorieuse, les champion·ne·s joué·e·s, le nombre de sbires éliminés, les champion·ne·s banni·e·s, etc. Ces données sont cruciales pour l'analyse des performances individuelles et collectives au cours des parties.

L'accès à ces informations requiert la réalisation de requêtes à l'API, impliquant l'utilisation d'une clé d'API. L'obtention de cette clé résulte d'une demande détaillée soumise à Riot Games, qui évalue chaque requête de manière individuelle. Riot Games impose également des limitations, avec une restriction à 20 requêtes par seconde et à 100 requêtes toutes les deux minutes. Ces mesures sont mises en place pour garantir un usage éthique et équitable de l'API, évitant ainsi toute surcharge inappropriée de leurs serveurs.

4.1.1 RiotWatcher

RiotWatcher se profile comme une bibliothèque Python incontournable, spécialement conçue pour simplifier l'accès à l'API de Riot Games. Dans le contexte de la complexité inhérente aux interactions directes avec cette API, RiotWatcher émerge comme une solution élégante, offrant

une approche efficace et conviviale aux développeur·euse·s. C'est précisément pour ces raisons que j'ai pris la décision d'intégrer RiotWatcher dans le cadre de ce mémoire.

Cette bibliothèque présente plusieurs fonctionnalités clés qui simplifient significativement le processus de développement. Parmi ces fonctionnalités, on peut citer la facilitation des requêtes aux différentes sections de l'API de Riot Games et la gestion proactive des erreurs, améliorant ainsi de manière substantielle l'expérience des développeur·euse·s. RiotWatcher simplifie également l'accès à des sections spécifiques de l'API en offrant, par exemple, une méthode intuitive pour récupérer des informations sur un joueur, un match ou une ligue.

Ces fonctionnalités revêtent une importance cruciale pour mon mémoire, et permettent aux développeur·euse·s de se concentrer pleinement sur l'analyse des données plutôt que sur la gestion technique des requêtes.

4.2 Échantillon de données

League of Legends est estimé compter plus de 143 millions de comptes actifs à travers les cinq continents [19]. Riot Games possède des serveurs à travers le monde, stockant ainsi les données associées aux joueur·euse·s jouant dans leur région respective. Pour chaque appel à l'API, il est impératif de spécifier le serveur avec lequel on souhaite interagir.

Afin de rendre mon analyse plus spécifique et orientée, j'ai pris la décision de limiter mon étude au serveur *euw1* (Europe de l'Ouest).

À la date du 16 janvier 2024, j'ai recensé 1 060 649 joueur·euse·s actifs et classé·e·s sur ce serveur. Un·e joueur·euse est considéré·e comme classé·e lorsqu'il·e a participé·e à au moins cinq parties dans une file classée. Il est à noter que deux files classées existent : la file solo/duo et la file flexible. Dans la file solo, la création de groupe est limitée à deux joueur·euse·s, tandis que la file flexible permet la formation de groupes de 1, 2, 3, ou 5 joueur·euse·s. Il existe dix classements possibles : Fer, Bronze, Argent, Or, Platine, Émeraude, Diamant, Maître, Grand Maître, Challenger 4.1. Les rangs du niveau fer au niveau diamant compris sont sous divisé en 4 sections appelés tiers et numéroté de 1 à 4, en chiffres romain. Plus on est haut dans sa section, plus on est associé à un petit numéro. Ainsi, le·a joueur·euse classé·e Or II est mieux classé·e que celui classé·e Or IV.

Dans le cadre de notre analyse, nous avons choisi de n'inclure que les informations relatives à la file classée, en mettant particulièrement l'accent sur la file solo. Ceci veut donc dire que j'ai récupéré les parties en spécifiant **queue = 420** lors des appels à l'API qui m'ont permis de créer ma base de données. Ce qui correspond à la file solo. Les autres valeurs possibles sont 440 pour la file cassé en flex, 450 pour le mode ARAM (All Random, All Mid), qui est un mode particulier du jeu, ainsi que 700 pour le mode clash qui est correspond au tournois pour les joueur·euse·s amateurs organisé par Riot Games.

Le graphique 4.2 illustre la répartition des joueur·euse·s classé·e·s en date du 16 janvier 2024, pour le serveur de l'Europe de l'ouest.

Les colonnes de Grand Maître et Challenger ne sont pas vides. Elles semblent inexistantes car elles représentent respectivement 0.023% et 0.01% des joueur·euse·s. Étant donné que ces deux classements représentent à eux deux moins de 0.05% des joueur·euse·s, je n'ai pas pris en compte de données issues de ces deux catégories de joueurs.

Au vu de l'ampleur de la quantité de données, il est essentiel de choisir un échantillon. Nous avons choisi de travailler sur un échantillon de 17 381 matchs de 199 joueurs. Afin de créer cet échantillon, nous avons exécuté un script demandant 100 parties pour 200 joueur·euse·s.

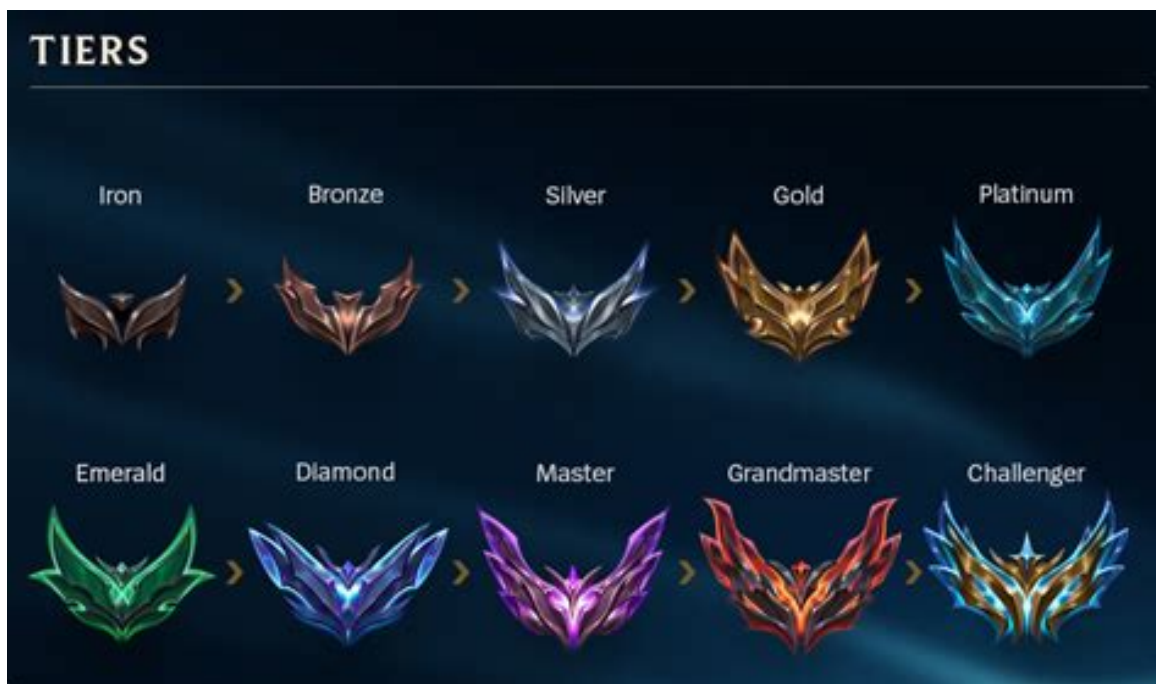


Figure 4.1: Icône par classement cherché en fr

Cette inadéquation entre les résultats demandés et ceux obtenus s'explique par le temps d'accès à l'API.

En effet, le temps de téléchargement de mes données se compte en jours. Étant donné que les serveurs de Riot Games sont régulièrement entretenus et mis à jour, il arrive fréquemment que des requêtes soient perdues, ce qui est donc à l'origine du décalage entre le nombre de parties totales attendues et celui obtenu. Concernant le·a joueur·euse manquant·e, il s'agit d'un coup du sort. Entre le moment où nous avons choisi nos joueur·euse·s et celui où toutes nos données ont été téléchargées, ce·tte dernier·e a été banni·e du jeu pour des raisons auxquelles nous n'avons pas accès, et de ce fait, nous n'avons pas pu obtenir d'information pour ce·tte joueur·euse. L'ensemble des données que nous avons téléchargées sont basées sur le patch 14.7 [20]

Sur le graphique 4.3, nous pouvons voir que l'échantillon choisi de manière aléatoire est assez représentatif à la réalité.

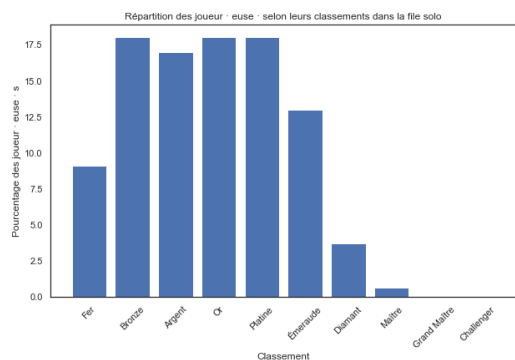


Figure 4.2: Répartition du niveau des joueur·euse·s

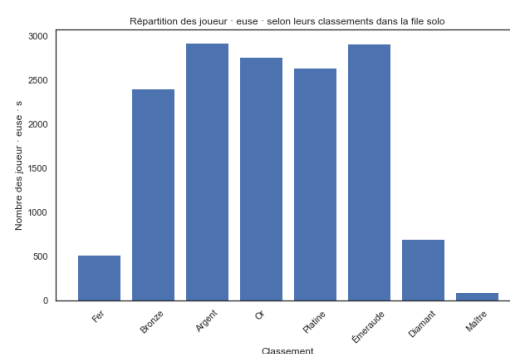


Figure 4.3: Répartition du niveau des joueur·euse·s sur l'échantillon étudié

Cependant, nous pouvons aussi constater que les catégories Fer, Diamant et Maître sont

significativement plus petite que les autres. Durant l'analyse des modèles, j'ai décidé de dédoubler mon travail. Ainsi, nous ferons les analyses une fois de façon globale et une autre fois en regardant plus spécifiquement pour chaque catégorie. Afin d'avoir assez de matière pour travailler, lors de cette deuxième partie d'analyse, nous avons décidé de gonfler les catégories sous-représentées pour qu'elles atteignent au moins chacune 1500 parties. Pour cela, nous avons donc fait de nouvelles requête à l'API et nous avons enregistré dans un deuxième fichier toutes les informations relatives aux parties rajoutées. Nous avons fait le choix de dédoubler l'analyse, car comme vous pourrez le constater plus tard, l'importance de certaines caractéristiques dépend fortement du niveau du-de la joueur·euse.

Chapitre 5

Analyse des caractéristiques

Dans ce chapitre, nous allons observer l'influence des différentes caractéristiques sur la caractéristique cible, à savoir la caractéristique *win*. Nous commencerons par énumérer et donner une brève explication de chaque caractéristique, ainsi que de leur type. Nous mettrons ensuite en avant l'impact des caractéristiques. Nous terminerons ensuite avec une analyse des matrices de corrélations.

Nous avons décidé pour l'analyse de l'influence de caractéristique, de faire des analyses à la fois sur la moyenne des données ainsi que sur la médiane de ces dernières. En effet, la moyenne étant sensible aux valeurs extrêmes, elle peut donner une image faussée de la centralité des données. Ceci n'est par contre pas le cas pour la médiane qui est beaucoup moins sensible aux valeurs extrême. Les différences de valeur entre la moyenne et la médiane nous permettrons de montrer l'asymétrie de la relation. Ainsi, si la moyenne est plus grande que la médiane, nous avons une distribution positivement asymétrique, mais dans le cas inverse, nous avons une distribution négativement asymétrique.

5.1 Présentations des caractéristiques

Dans cette section, nous allons analyser les différentes caractéristiques que nous avons choisies afin de faire notre prédiction.

Le fichier csv que nous avons créé contient 36 caractéristiques.

- *pseudo*[STRING]: Contient le pseudo du·de la joueur·euse
- *id_game*[STRING]: Contient le numéro d'identification de la partie.
- *nb_party_ally*[INT]: Contient le plus grand nombre de partie jouée avec un·e autre joueur·euse de la partie.
- *champion_i*, avec i de 0 à 10 [STRING]: Contient les noms des 10 champion·ne·s de la partie.
- *vision_score_mean*[INT]: Contient le score de vision moyen du·de la joueur·euse observé·e.
- *lane_played*[STRING]: Contient le rôle que le·la joueur ·euse joue pour la partie analysée.
- *main_lane*[STRING]: Contient le rôle que le·la joueur ·euse joue le plus souvent.
- *champion_played*[STRING]: Contient le nom du personnage incarné par le·la joueur·euse
- *main_champ*[STRING]: Contient le nom du personnage le plus souvent incarné par le·a joueur·euse
- *win*[BOOLEAN]: Contient le résultat du·de la joueur·euse pour la partie.
- *winrate*[STRING]: Contient le taux de victoire le·la joueur·euse.
- *minion_kills*[INT]: Contient le nombre de sbires tués en moyenne par partie par le·la joueur·euse.
- *ward_kill*[INT]: Contient le nombre de balises de vision tuées en moyenne par partie par le·la joueur·euse.
- *ward_placed*[INT]: Contient le nombre de balises de vision posées en moyenne par partie par le·la joueur·euse.
- *winrate_blue*[INT]: Correspond au taux de victoire du·de la joueur·euse lorsqu'il joue du coté bleu.
- *winrate_red*[INT]: Correspond au taux de victoire du·de la joueur·euse lorsqu'il joue du coté rouge.
- *side_played*[INT]: Contient le coté joué pour la partie par le·la joueur·euse, 100 pour le coté bleu et 200 pour le rouge
- *gold_earn*[INT]: Contient le nombre moyen de po gagnées par le·la joueur·euse par partie.
- *gold_spent*[INT]: Contient le nombre moyen de po dépensées par le·la joueur·euse par partie.
- *damage_taken*[INT]: Contient le nombre moyen de dégâts pris par partie par le·la joueur·euse.

- *damage_deal*[INT]: Contient le nombre moyen de dégâts pris par partie par le·la joueur·euse.
- *kda*[INT]: Contient le kda moyen du·de la joueur·euse. Le kda moyen correspond au nombre d'assistance et de mort infligée moyenne sur le nombre moyen de mort.
- *winning_losing_series* [INT]: Représente le nombre de victoire (positif) ou de défaite (négative) consécutive pour le·la joueur·euse avant la partie présentement jouée.
- *my_opponent*[STRING]: Contient le nom du·de la champion·ne joué·e par l'adversaire sur la même voie.
- *main_banned*[BOOLEAN]: Regarde si pour la partie en cours, le·la champion·e principalement joué·e par le·la joueur·euse a été banni·e.
- *game_duration*[INT]: Contient la durée moyenne des parties gagnées par le·la joueur·euse.
- *game_creation*[INT]: Contient l'heure à laquelle la partie à été lancée.
- *division*[STRING]: Contient le niveau du·e la joueur·euse (Fer, Bronze, Argent, Or, Platine, Emeraude, Diamand, Maître).
- *win_on_champ*[INT]: Contient le taux de victoire du·de la joueur·euse sur le·la champion·ne joué·e pour la partie.
- *nb_partie_w_champ*[INT]: Contient le nombre de parties jouées par le·la joueur·euse sur le·la champion·ne.

Une analyse détaillée de ces caractéristiques se trouve en annexe.

5.2 Matrice de corrélation

Regardons maintenant quels sont les caractéristiques corrélées les unes aux-autres et avec la caractéristique cible *win*.

5.2.1 Sur tout le dataset

Sur l'image 5.1, nous pouvons observer la matrice de corrélation pour l'ensemble des données. La première chose qui saute aux yeux, c'est que la colonne *main_banned* est vide. Ceci est du au fait que sur les parties téléchargées, il n'existe aucun joueur·euse qui a vu son champion principal se faire bannir.

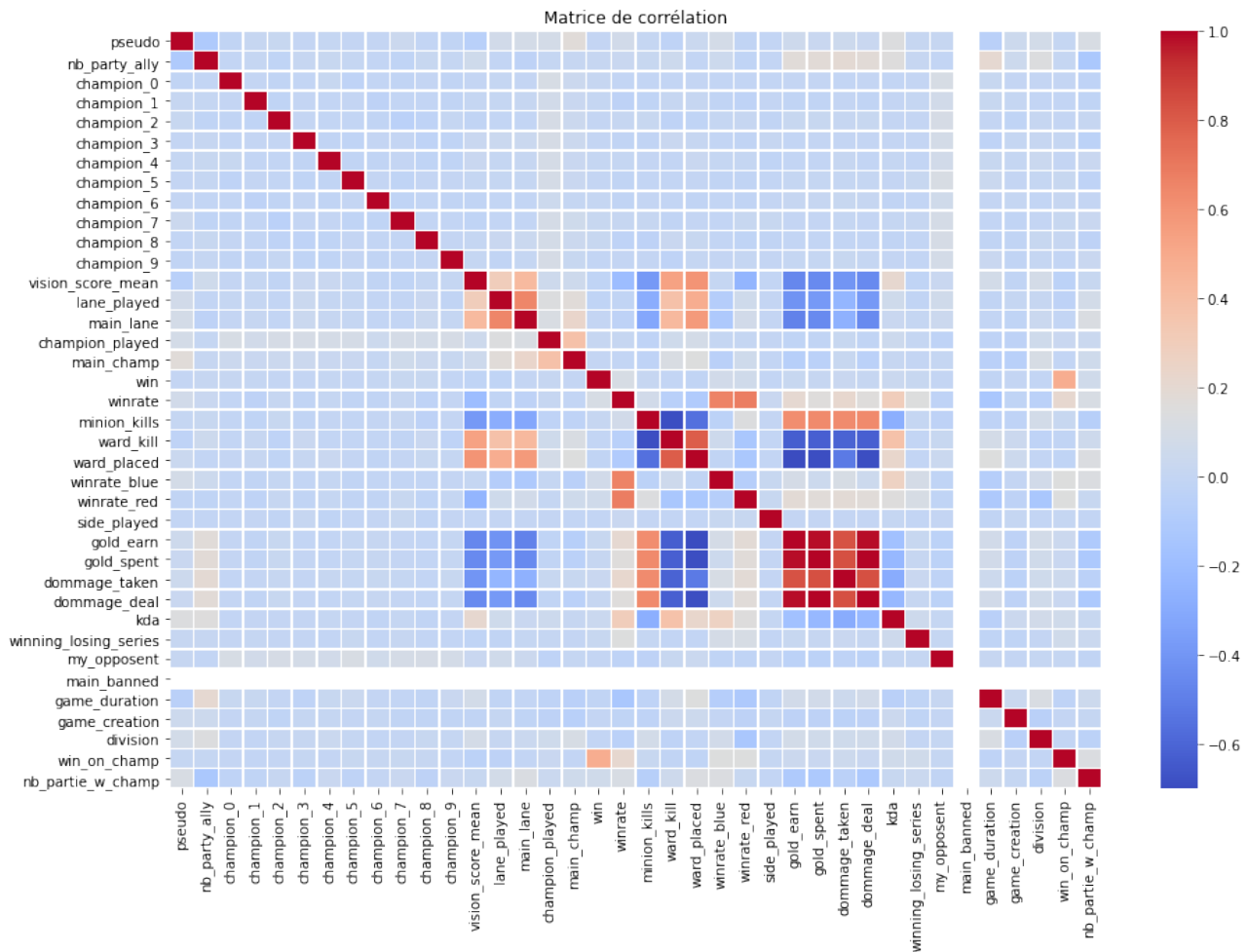


Figure 5.1: Matrice de corrélation de l'ensemble des données

Nous pouvons également voir un gros bloc rouge de données très corrélées. Il s'agit des caractéristiques *gold_earn*, *gold_spent*, *damage_taken* et *damage_deal*. Nous pouvons voir sur le tableau [5.1] les valeurs des corrélations en question.

Ce qui est par contre surprenant, c'est la corrélation entre *damage_deal* et *gold_spent*

Commençons par regarder les corrélations liées à *gold_earn*. Nous pouvons voir sur l'image [8.19], en annexe, une représentation visuelle des points dans l'espace. Comme nous nous y attendons avec de telle valeur de corrélation les points sont tous très regroupés dans les 3 cas de la ligne de régression. Essayons d'expliquer un peu ces valeurs. Il était évident que *gold_earn*

	<i>gold_earn</i>	<i>gold_spent</i>	<i>dommage_taken</i>	<i>dommage_deal</i>
<i>gold_earn</i>	1	0.9834	0.8399	0.9834
<i>gold_spent</i>	0.9834	1	0.8392	1
<i>dommage_taken</i>	0.8399	0.8392	1	0.8392
<i>dommage_deal</i>	0.9834	1	0.8392	1

Tableau 5.1: Tableau de corrélation entre *gold_earn*, *gold_spent*, *dommage_taken* et *dommage_deal*

serait corrélé avec *gold_spent*. En effet, lorsqu'on gagne de l'argent dans le jeu, on a tendance à le dépenser afin de s'améliorer, et comme on ne peut en général pas dépenser de l'argent qu'on a pas, il est normal que cette valeur soit très élevée.

La corrélation entre *gold_spent* et *dommage_taken* est un peu plus ardue à expliquer. Je pense qu'elle vient du fait que si l'on prend des dégâts, c'est que l'on participe à des combats, et qu'ainsi, en étant pro-actif sur le jeu, cela engendre de l'argent pour les joueur·euse·s. Pour finir, la corrélation entre *gold_spent* et *dommage_deal* est quant à elle également évidente. En effet, lorsque l'on fait des dégâts à un·e champion·, iel fini par mourir et ainsi par rapporter de l'argent.

Regardons maintenant les corrélations liées à *gold_spent*. Nous pouvons voir sur l'image [8.4], en annexe, les représentations visuelles correspondantes. Ce qui saute tout de suite aux yeux, c'est la perfection de la corrélation entre *gold_spent* et *dommage_deal*. En effet, nous pouvons voir que l'alignement des points est parfait. Ceci signifie qu'il suffit d'une des deux caractéristiques pour entièrement et parfaitement prédire l'autre. On peut expliquer ceci par le fait que l'argent est dépensé pour des objets dans le jeu. Dans le jeu, tous les objets ne permettent pas de faire directement de l'attaque en plus. En effet, certains sont entièrement consacré à la défense. Cependant, avec plus de défense, un·e champion·ne peut rester plus longtemps en jeu, et ainsi, faire à termes plus de dégâts. On peut aussi observer que bien que relativement épars, concernant la corrélation à *dommage_taken*, les points sont quand même bien alignés. Cette corrélation est légèrement moins évidente. En effet, quand on prend des dégâts, c'est que l'on se bat. Si on se bat, on gagne de l'argent, et si on gagne de l'argent, on peut le dépenser.

Il nous reste maintenant à observer la corrélation entre *dommage_deal* et *dommage_taken*. Nous pouvons observer sur le graphe suivant [8.3] la représentation visuelle de la corrélation. Il était attendu que ces deux caractéristiques soient corrélées. En effet, il est excessivement rare de faire des dégâts sans en subir et inversement.

Regardons maintenant la caractéristique *minninon_kills*. Nous pouvons voir qu'elle est corrélée avec les caractéristiques susmentionnées. Sur la table [5.2], nous pouvons voir les valeurs de ces dites corrélations.

	<i>gold_earn</i>	<i>gold_spent</i>	<i>dommage_taken</i>	<i>dommage_deal</i>
<i>minninon_kills</i>	0.6275	0.6409	0.6339	0.6409

Tableau 5.2: Corrélations avec la caractéristique *minninon_kills*

Regardons les représentations visuelles [5.6] associées et décortiquons les. Premièrement, comme dit juste avant, la caractéristique *minninon_kills* est corrélée avec *gold_earn*. Ceci est tout à fait normal, en effet, tuer des sbires donne de l'argent. *minninon_kills* est également corrélé avec *gold_spent* pour les raisons expliquer précédemment de quand un·e joueur·euse as de l'argent tu le dépenses. Maintenant, demandons-nous pourquoi la caractéristique *minninon_kills*

est corrélée à *dommage_deal* et *dommage_taken*. Pour le premier, je pense que c'est du au fait que les champion·e·s qui font des dégâts aux sbires sont aussi ceux qui en font aux champions. Pour le second, je pense que c'est du au fait qu'en général, les champion·e·s qui font des dégâts au sbires, sont ceux qui sont ciblé·e·s, du au fait que ce sont eux les menaces.

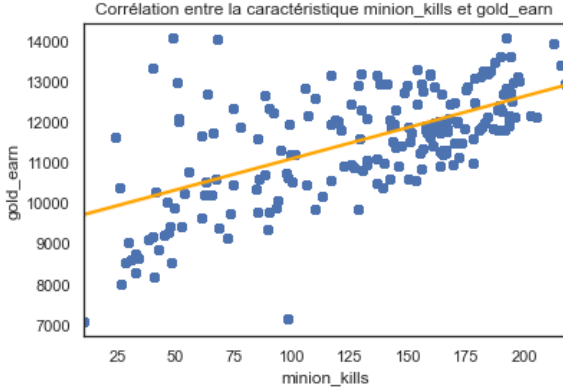


Figure 5.2: Corrélation = 0.6275

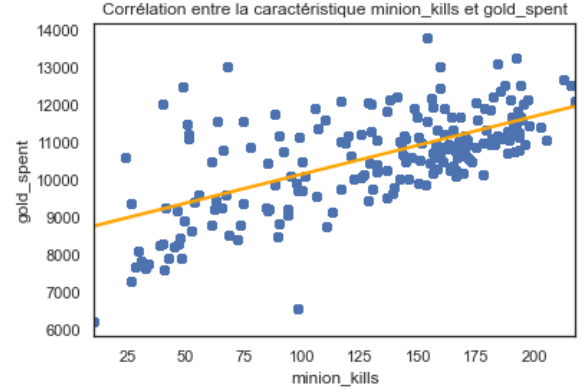


Figure 5.3: Corrélation = 0.6409

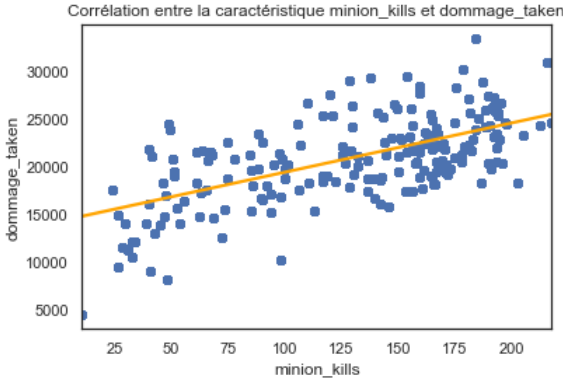


Figure 5.4: Corrélation = 0.6339

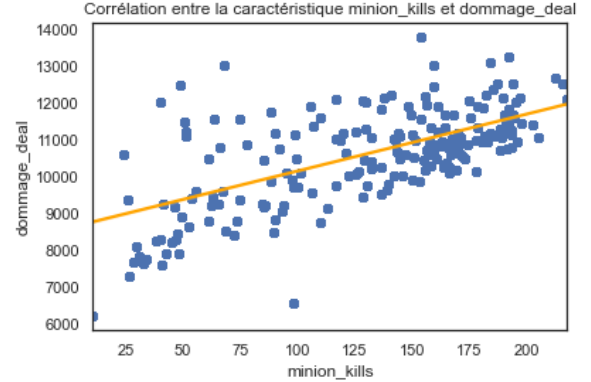


Figure 5.5: Corrélation = 0.6409

Figure 5.6: Corrélations avec la caractéristique *minnion_kills*

Nous pouvons également noté un petit bloc de trois caractéristiques qui sont corrélées entre-elles: *vision_score_mean*, *lane_played* et *main_lane*. Nous pouvons voir ci-dessous les valeurs associées à ces caractéristiques [5.3].

	<i>vision_score_mean</i>	<i>lane_played</i>	<i>main_lane</i>
<i>vision_score_mean</i>	1	0.3018	0.4209
<i>lane_played</i>	0.3018	1	0.6505
<i>main_lane</i>	0.4209	0.6505	1

Tableau 5.3: Caption

Étant donné que les caractéristiques *lane_played* et *main_lane* sont catégorielles, il est peu censé de faire les même représentations que pour les caractéristiques précédentes. Observons pour commencer plus attentivement la caractéristique *vision_score_mean*. On peut voir qu'elle est plus corrélée à *main_lane* qu'à *lane_played*. Concernant *lane_played*, on peut voir qu'elle est modérément corrélée à *vision_score_mean* et deux fois plus à *main_lane*. Ceci s'explique

aisément par le fait que la grande majorité du temps, les joueur·euse·s jouent sur la même voie en boucle.

Nous pouvons aussi observer deux caractéristiques très corrélées entre-elles: *ward_kill* et *ward_placed*. La valeur de leur corrélation est 0.7941 . Ceci s'explique très facilement aussi. Il existe sur la carte des endroits où prendre la vision est stratégique et important. Dans cette optique, les joueur·euse·s vérifient régulièrement ces points et détruisent les balises de visions ennemies placées, afin d'y mettre les leurs.

Nous pouvons aussi observer que ces deux caractéristiques sont aussi corrélées avec *vision_score_mean*, *lane_played* et *main_lane* que nous venons de voir. Comment vous l'aurez compris, ceci est dû au fait que le score de vision se calcule grâce entre autres aux nombres de balises de vision posée et retirée. Ci dessous, vous pouvez voir les valeurs de ces corrélations [5.4]. **est-ce assez? - de blabla, + de conclusion, ce qui est attendu vs surprenant**

	<i>vision_score_mean</i>	<i>lane_played</i>	<i>main_lane</i>
<i>ward_placed</i>	0.5953	0.4846	0.5739
<i>ward_kill</i>	0.5386	0.3780	0.4265

Tableau 5.4: Corrélation avec *ward_kill* et *ward_placed*

Nous pouvons aussi voir que les caractéristiques *winrate_blue* et *winrate_red* sont ensemble corrélées à *winrate*. Ceci s'explique par le fait que ces deux caractéristiques sont une décomposition de *winrate*, comme expliqué précédemment.

Regardons maintenant les cas de forte corrélations négatives. Pour rappel, une corrélation négative signifie que lorsque la valeur d'une caractéristique augmente, l'autre diminue.

Nous pouvons voir un premier bloc assez intense concernant les caractéristiques *ward_kill* et *ward_placed* avec *gold_earn*, *gold_spend*, *dommage_taken* et *dommage_deal*[5.5].

	<i>gold_earn</i>	<i>gold_spend</i>	<i>dommage_taken</i>	<i>dommage_deal</i>
<i>ward_kill</i>	-0.6274	-0.6267	-0.6087	-0.6267
<i>ward_placed</i>	-0.7002	-0.6921	-0.5193	-0.6926

Tableau 5.5: Caption

Attardons nous pour commencer sur *ward_kill*. En général, les joueur·euse·s tuent peu de balise de vision. La corrélation négative entre les caractéristiques *ward_kill* et *ward_placed* avec *gold_earn* et *gold_spend* s'explique facilement par le fait que les champion·e·s qui s'occupent de placer et retirer les balises de vision sont en générale les support·e·s. De par leur rôle, ils tuent en général peu d'ennemi·e et en conséquence gagnent et dépensent moins d'argent.

Concernant *ward_placed*, nous pouvons voir que la caractéristique avec la quelle elle est la moins corrélée négativement est *dommage_taken*. Ceci s'explique par le fait qu'au final, quel que soit le poste de chacun·e, il est du devoir de tou·te·s de mettre de la vision sur la carte et tout le monde prends des dégâts à un moment ou à un autre, même les meilleur·e·s joueur·use·s.

Pour finir, concentrons nous sur le principal: la caractéristique *win*. Ce qui saute aux yeux pour commencer, c'est qu'il n'existe qu'une seule caractéristique qui semble au premier abord réellement corrélée avec notre caractéristique cible. Il s'agit de la caractéristique *win_on_champ* et la valeur de sa corrélation est 0.4827. La deuxième caractéristique la plus corrélée est *winrate* avec 0.1048, ce qui est vraiment peu, mais qui est quand même mille fois plus que la valeur

la moins corrélée qui est *champion5* avec 0.000183. Nous pouvons donc d’ors et déjà nous demander si cette caractéristique ne sera pas la seule réellement prédictive.

5.2.2 Matrices de corrélation par division

Dans cette section, nous mettrons en évidence les différences significatives des matrices de corrélation par niveau en comparaison à la matrice global. On s’attend à ce qu’elles soient relativement similaire.

Observons maintenant ce qui se distingue lorsque nous calculons la matrice de corrélation par division[8.15]. Nous pouvons directement voir que même si toutes les matrices se ressemblent, elles ont chacune des spécificités, ce qui permet de mettre en lumière le fait qu’en fonction de son niveau, un·e joueur·euse se comporte différemment et qu’ainsi certaines caractéristiques sont plus importante selon la division. Nous pouvons ainsi constater que pour un·e joueur·euse en Fer, la caractéristique *pseudo* est globalement plus corrélées négativement que dans les autres divisions, surtout pour la caractéristique *ward_kill*.

Nous pouvons également observer une certaine disparité en fonction des niveaux autour des caractéristiques *vision_score_mean*, *lane_played*, *main_lane*, *champion_played* et *main_champ*. En effet, la ou ces caractéristiques sont de peu à pas corrélée en Fer, elles sont nettement plus marquée en Émeraude et Bronze. En Maître elles sont par contre assez disparate. Nous pouvons faire la même constatation pour l’ensemble de caractéristique

Les caractéristiques *vision_score_mean*, *gold_earn*, *gold_spent*, *damage_taken* et *damage_deal* sont quant à elles de plus en plus corrélées négativement au fur et à mesure des niveaux. Ce qui implique qu’elles prennent de plus en plus d’importance quand un·e joueur·euse s’améliore.

Ce qui par contre est commun à toutes les divisions, c’est que la caractéristique *win* ne présente de corrélation marquée qu’avec *win_on_champ*. Ceci semble signifier que quoi qu’il se passe, *win_on_champ* est une caractéristique déterminante extrêmement prédictive.

Chapitre 6

Les modèles et leurs résultats

Dans ce chapitre, nous analyserons 5 modèles que nous avons fait performer sur 4 cas distincts.

Le premier cas est le cas basique, celui où l'on ne touche pas aux données. Le deuxième cas est celui où l'on retire les caractéristiques corrélées à plus de 60% entre elles. Le troisième cas est celui où l'on garde uniquement la caractéristique *win_on_champ* et le dernier est celui où l'on retire uniquement cette caractéristique.

Cas 1	Ensembles des caractéristiques
Cas 2	Absence des caractéristiques corrélées à plus de 60% entre elles
Cas 3	Uniquement la caractéristique <i>win_on_champ</i>
Cas 4	Sans la caractéristique <i>win_on_champ</i>

Tableau 6.1: Les différents cas étudiés

Nous appliquerons cette méthodologie aux modèles de la régression logistique, du Decision-Tree, du RandomForest, du K-Nearest-Neighbor et du Support Vector Classifier. Pour chaque modèle et dans chaque cas, nous avons performé 5 validations croisées et pris la valeur moyenne comme référence. Nous nous baserons à chaque fois sur les matrices de confusion, le F1-Score ainsi que les courbes ROC et les scores AUC associés afin de pouvoir déterminer plus tard le meilleur modèle. Celui qui nous permet donc de prédire le résultat d'une partie durant le chargement de cette dernière.

Nous avons choisi ces modèles car ils sont considérés comme généralement adaptés à ce genre de problème. Le choix des valeurs des paramètres a été fait grâce à une recherche non-exhaustive de leur influence sur leur performance. Nous avons essayé de choisir les paramètres menant aux meilleures performances.

6.1 Régression Logistique

Pour la régression logistique, nous avons utilisé la bibliothèque *sklearn*, et plus précisément *sklearn.linear_model.LogisticRegression* avec les paramètres *penalty = 'l1'*, *solver = 'liblinear'*, *max_iter = 10000000*.

Le paramètre *penalty* contrôle la régularisation appliquée au modèle. Lui attribuer la valeur *l1* signifie appliquer la régularisation de Lasso.

Le paramètre *solver* correspond à la méthode utilisée pour résoudre le problème d'optimisation. Ici, il s'agit de la minimisation de la fonction de perte logistique. Pour finir, le paramètre *max_iter* contrôle le nombre maximum d'itérations autorisée afin d'attendre la convergence. Nous avons choisi un grand chiffre afin d'atteindre cette dite convergence.

6.1.1 Analyse sur l'ensemble des données

Commençons par regarder comment performe la régression logistique. Dans le tableau ci-dessous, nous pouvons observer les résultats sur la validation croisée et sur l'ensemble de test. Nous pouvons observer sur le tableau[6.2] que les différents cas performant mieux sur l'ensemble de test que sur la validation croisée. Il semblerait donc qu'ils aient bien réussi à trouver la bonne régression permettant un bon tri des parties. Le cas 4 fait marginalement mieux que l'aléatoire, mais nettement moins bien que les autres cas.

	Validation croisée	Ensemble de test
Cas 1	66.77%	66.22%
Cas 2	66.90%	66.35%
Cas 3	66.78%	66.72%
Cas 4	54.84%	53.00%

Tableau 6.2: Précision pour la régression logistique sur l'ensemble des données

Regardons maintenant les matrices de confusion afin de pouvoir comprendre comment sont triées les parties.[6.5]. Nous pouvons observer que les parties pour le cas 4 sont majoritairement(63.60%) prédites comme étant des défaites.

Analysons maintenant les taux de précision pour les victoires et les défaites. Nous pouvons voir sur le tableau synthétisant ces informations[6.6] que tous les cas prédisent mieux les défaites que les victoires. Cependant, les 3 premiers cas prédisent de manière similaire les victoires. Seul le cas 4 a une performance sur les victoires inférieur à 50%. Ce qui est dû, comme nous venons de le voir, à la tendance du modèle pour le cas 4 à tout classer comme des défaites. Cette tendance est tellement marquée que le F1-Score du cas 4 est inférieur également à 50%, démontrant ainsi que ce cas fait moins bien que l'aléatoire. Pour les 3 premiers cas, nous pouvons voir que les F1-Scores sont similaire aux scores de précision sur l'ensemble de test.

Regardons maintenant la courbe ROC[6.7] et les scores associées. Vu la similitude des scores AUC sur les 3 premiers cas, nous nous attendions à voir les courbes ROC confondues telles quelles le sont. De même, nous pouvons voir que la courbe ROC pour le cas 4 est très proche des chances de l'aléatoire, soulignant ainsi à quel point ce dernier modèle est mauvais et par là même, l'importance de la caractéristique *win_on_champ* pour la régression logistique. Nous verrons dans la section suivante si cela s'avère vrai pour l'ensemble des joueur·euse·s, à tous niveaux.

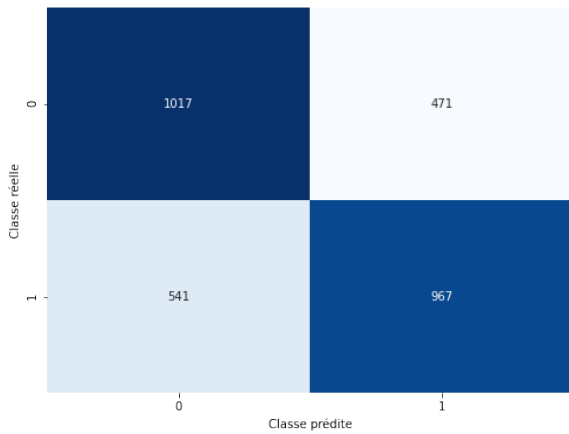


Figure 6.1: Cas 1

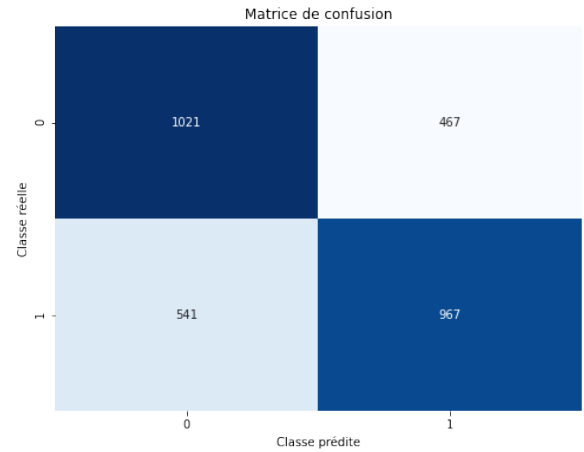


Figure 6.2: Cas 2

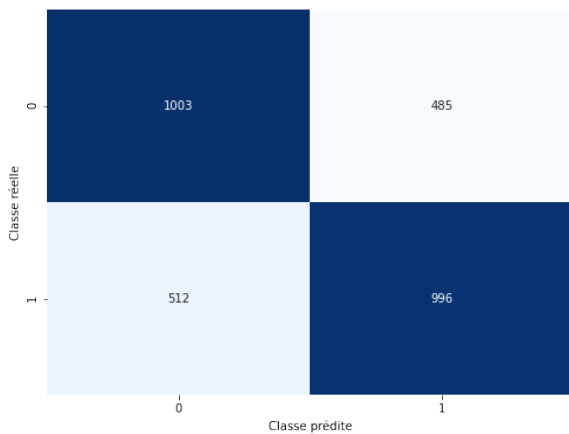


Figure 6.3: Cas 3

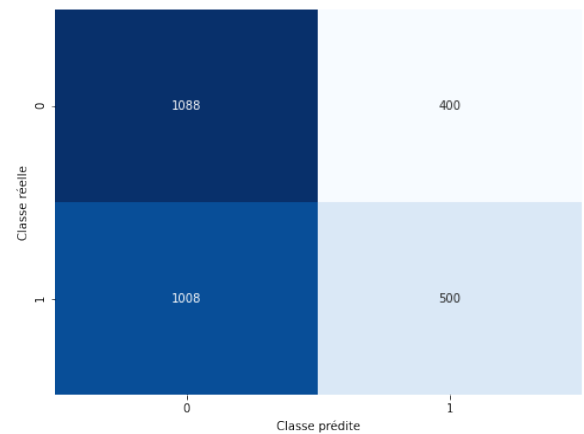


Figure 6.4: Cas 4

Figure 6.5: Matrices de confusion pour la régression logistique, sur l'ensemble des données

6.1.2 Analyse par division

Dans cette section, nous allons examiner comment se comporte la régression logistique lorsque nous l'appliquons sur les sous-ensembles de données classées par division dans les 4 cas défini précédemment[6.1]. Pour commencer, regardons les performances atteintes par nos modèles sur les 4 cas, pour la validation croisée (CV) et sur l'ensemble de test (ET)[6.3]. Nous pouvons observer que le cas 4 performe significativement moins bien que les 3 autres cas. De premier abord, il semblerait que ça soit le cas 3 qui a les meilleurs résultats pour la validation croisée. Cependant, ce dernier n'a pas toujours les meilleurs résultats sur l'ensemble de test. Ceci peut être vu comme étant dû au fait que même si la caractéristique *win_on_champ* est capitale en l'absence d'autres caractéristiques, le modèle a tendance à trop vouloir coller aux données (overfit).

Ainsi, nous pouvons par exemple voir que pour les divisions Fer, Argent, Or, Platine et Émeraude, le cas 1 performe mieux.

Regardons maintenant les matrices de confusion associées à ces différents cas. Pour des soucis de lisibilité de ces dernières, elles se trouvent en annexe(cas1: [8.28], cas4: [8.37], cas3: [8.46], cas4: [8.55]). Sur le tableau[6.4], nous avons synthétisé les informations importantes données par ces matrices, à savoir le taux de précision correcte pour une victoire ou une défaite.

	Défaite	Victoire	F1-Score	AUC
Cas 1	68.34%	64.12%	65.64%	76.95%
Cas 2	68.61%	64.12%	65.73%	76.99%
Cas 3	67.40%	66.04%	66.64%	76.85%
Cas 4	73.11%	33.15%	41.52%	55.87%

Figure 6.6: Scores de précision pour la régression logistique sur l'ensemble des données

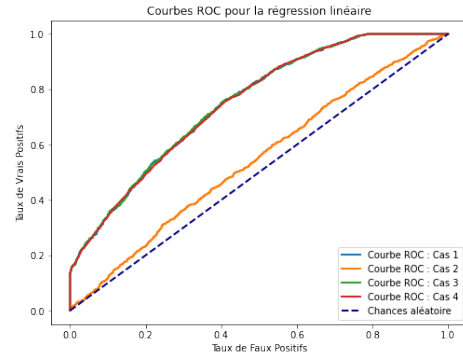


Figure 6.7: Courbes ROC pour la régression logistique sur l'ensemble des données

	Cas 1		Cas 2		Cas 3		Cas 4	
	VC	ET	VC	ET	VC	ET	VC	ET
Fer	62.94%	65.71%	63.92%	65%	65.53%	63.57%	62.94%	51.78%
Bronze	64.58%	63.82%	64.58%	65.90%	66.24%	64.24%	64.58%	54.67%
Argent	66.35%	69.11%	66.56%	69.11%	67.59%	68.25%	66.35%	53.75%
Or	66.00%	70.39%	66.32%	70.57%	67.67%	68.59%	66.00%	51.26%
Platine	66.03%	68.62%	66.33%	68.80%	66.44%	67.48%	66.05%	54.25%
Émeraude	64.69%	64.32%	64.47%	64.83%	66.88%	63.97%	64.69%	51.45%
Diamant	64.33%	63.71%	64.96%	63.12%	66.00%	65.19%	64.36%	51.32%
Maître	63.91%	66%	63.25%	66.33%	64.83%	68%	63.91%	52.66%

Tableau 6.3: Taux de précision pour les défaites et les victoires pour la régression logistique par division

Nous pouvons observer que pour les 3 premiers cas, la régression logistique a bien plus facile à trouver qu'une victoire en est une, qu'une défaite en est une. Ce fait est présent dans le cas 4, mais de manière moins marqué.

Nous allons maintenant regarder quels sont les F1-scores associés afin de savoir si ceci est du à une inégalité des classes[6.5]. Nous pouvons voir qu'entre les cas 1 et 2, les scores sont toujours sensiblement similaire. Cependant, contrairement à ce que l'on pouvait imaginer, le cas 3 ne performe pas mieux malgré une meilleure précision apparente. Ceci signifie que si l'on ne garde que la caractéristique *win_on_champ*, notre modèle a tendance à tout classer comme étant des victoires. Concernant le cas 4, nous pouvons observer qu'il performe de manière assez différente sur chaque division, mettant ainsi en exergue l'importance de la décomposition en sous-ensemble. Ainsi, nous pouvons voir que pour les Argent, le modèle de régression logistique performe moins bien que l'aléatoire, mais atteint le score honorable de 62.03% pour la catégorie Maître. Ainsi, il semblerait qu'il soit plus facile de prédire le comportement des joueur·euse· les mieux classé·e·s.

Pour finir, observons les courbes ROC[6.12] et les scores AUC associés[6.6].

Lorsque nous regardons les scores AUC, nous pouvons voir que dans les 3 premiers cas, ils tournent autour de 70%. Cependant, il faut noter que les résultats sont meilleurs pour la division Or de manière générale, et légèrement meilleure en moyenne dans le cas 3.

Si nous observons les courbes ROC de manière plus attentive, nous pouvons voir que dans le cas 1, c'est la division Argent qui performe le mieux. Dans le cas 2, nous pouvons voir que les divisions forment deux groupes distincts. Un premier groupe qui performe mieux et qui est

	Cas 1		Cas 2		Cas 3		Cas 4	
	Défaite	Victoire	Défaite	Victoire	Défaite	Victoire	Défaite	Victoire
Fer	59.86%	60.14%	58.74%	72.26%	53.84%	74.55%	48.95%	54.74%
Bronze	58.52%	69.96%	59.30%	69.5%	55.81%	71.30%	59.30%	49.32%
Argent	67.19%	71.43%	68.12%	71.42%	69.37%	67.29%	59.67%	46.62%
Or	70.66%	70.17%	69.49%	70.85%	72.20%	65.42%	54/05%	48.81%
Platine	64.50%	72.66%	64.88%	72.28%	66.79%	68.14%	53.81%	54.68%
Émeraude	59.21%	69.89%	60.19%	70.97%	56.90%	72.40%	41.44%	62.36%
Diamant	48.75%	79.39%	50.57%	78.18%	56.32%	74.54%	40.23%	63.03%
Maître	54.33%	74.57%	53.54%	74.57 %	55.91%	76.87%	33.07%	67.95%

Tableau 6.4: Taux de performance pour classer les victoires et les défaites pour la régression logistique

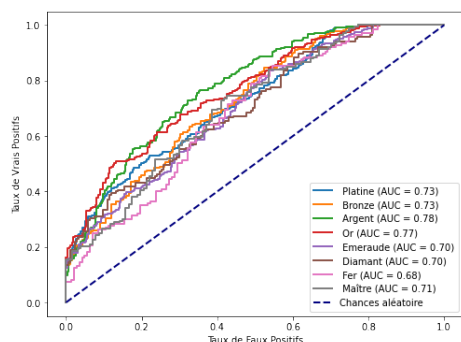


Figure 6.8: Cas 1

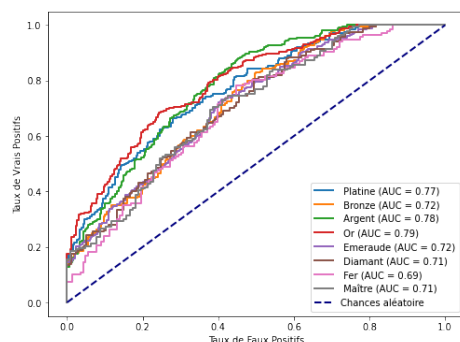


Figure 6.9: Cas 2

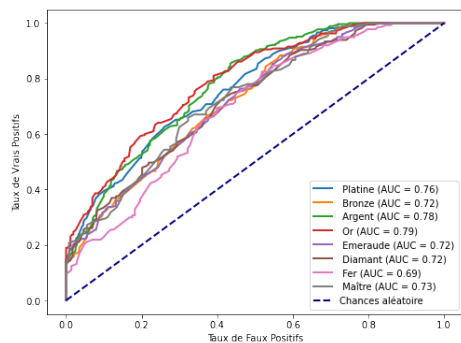


Figure 6.10: Cas 3

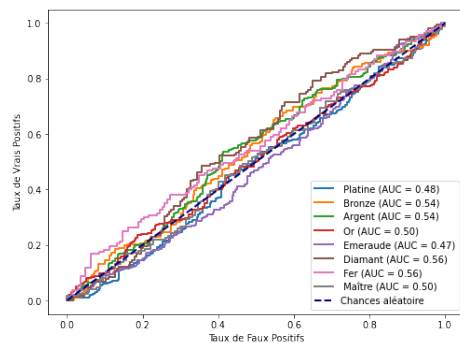


Figure 6.11: Cas 4

Figure 6.12: Courbes ROC par division pour la régression logistique

	Cas 1	Cas 2	Cas 3	Cas 4
Fer	67.13%	67.12%	66.87%	52.63%
Bronze	64.19%	64.18%	64.11%	50.23%
Argent	67.73%	68.10%	65.93%	47.48%
Or	71.63%	71.69%	68.82%	51.61%
Platine	70.03%	69.28%	67.91%	54.68%
Émeraude	65.21%	66.22%	66.01%	55.15%
Diamant	68.05%	67.89%	67.58%	55.76%
Maître	71.67%	71.46%	73.48%	62.03%

Tableau 6.5: F1-Score par division pour la régression logistique

	Cas 1	Cas 2	Cas 3	Cas 4
Fer	68.41%	69.36%	69.18%	55.18%
Bronze	72.53%	72.06%	72.56%	54.87%
Argent	77.78%	77.99%	77.75%	54.73%
Or	76.50%	79.31%	78.78%	53.75%
Platine	72.76%	76.68%	76.54%	56.23%
Émeraude	70.37%	71.50%	72.05%	51.90%
Diamant	70.20%	71.06%	72.45%	54.70%
Maître	70.56%	70.70%	72.96%	49.55%

Tableau 6.6: AUC sur les différents cas pour la régression logistique

composé des divisions Argent, Or et Platine. Ainsi qu'un second groupe composé des autres divisions, à savoir Maître, Diamant, Émeraude, Bronze et Fer. Ceci semble montrer que lorsque l'on retire les caractéristiques corrélées entre elles, il est plus facile pour le modèle de prédire si la partie finira en victoire ou en défaite pour des joueur·euse·s de niveau moyen. Dans le cas 3, nous pouvons observer que le modèle performe de manière relativement similaire pour toutes les divisions. Ce qui est aussi le vrai pour le cas 4 même si les performances laissent clairement à désirer.

Résumé: Sur l'ensemble des données, la régression logistique performe de manière similaire sur les différents cas, tant que la caractéristique *win_on_champ* est présente.

Cependant, ne garder que cette caractéristique n'est pas optimal. Le meilleur cas est celui où seules les caractéristiques corrélées à plus de 60% sont retirées, avec une précision de 68.11% et un F1-Score de 67.41% sur l'ensemble de test. Ce modèle est donc bien équilibré et performant sur les victoires comme sur les défaites. Il en va de même pour les données classées par division. Cependant, afin d'optimiser les résultats il faut utiliser l'ensemble des caractéristiques pour les divisions Fer, Platine et Diamant, afin d'obtenir une précision sur l'ensemble de test de respectivement 65.71%, 68.62%, et 63.71%. Sur le cas2, les caractéristiques corrélés à moins de 60%, les divisions ayant la meilleur précision sont les divisions Bronze, Argent, Or et Émeraude, avec respectivement des scores de 65.90%, 69.11%, 70.57%, 64.83%. Pour finir, seule la division Maître présente sa meilleure performance lorsque seule la caractéristique *win_on_champ* est utilisée, et a un score de 68%.

6.2 DecisionTree

Regardons maintenant comment performe le DecisionTree sur notre problème. Nous avons choisi d'utiliser comme paramètre `max_depth = 3`, `min_samples_split = 3`, `splitter = 'best'`, `random_state = 42` de la fonction `sklearn.tree.DecisionTreeClassifier`. Le paramètre `max_depth` représente la profondeur de l'arbre. Le paramètre `min_samples_split`, quant à lui représente le nombre minimum d'échantillon requis pour diviser un noeud. Le paramètre `splitter` désigne la stratégie choisie pour diviser un noeud. Avoir attribué la valeur "best" implique donc que le modèle va choisir la meilleure façon de se diviser. Afin de choisir cette meilleur façon, le modèle utilise le critère 'gini' qui est la valeur par défaut du paramètre `criterion`. Pour finir, le paramètre `random_state` contrôle l'aléatoire et permet la reproductibilité des résultats.

6.2.1 Analyse sur l'ensemble des données

Commençons donc par observer la précision du DecisionTree sur nos données d'entraînement avec la validation croisée et sur nos données de test ensuite[6.7]. Nous pouvons observer que dans les 3 premiers cas, pour la validation croisée, les résultats sont très similaires et tournent autour des 66%. Cependant, nous pouvons observer que lorsque nous testons le modèle sur l'ensemble de test, nous avons exactement le même résultat pour les 3 premiers cas.

Nous hypothétisons que ce résultat est donc du à l'importance déterminante de la caractéristique `win_on_champ`. Ce qui expliquerait la chute drastique de 10% de précision lorsqu'on retire cette caractéristique. Le quatrième cas quant à lui fournit des résultats 10% moins bons que les autres cas.

	Validation croisée	Ensemble de test
Cas 1	66.07%	65.64%
Cas 2	65.85%	65.64%
Cas 3	66.08%	65.64%
Cas 4	54.53%	55.09%

Tableau 6.7: Précision pour le DecisionTree sur l'ensemble des données

Ceci est du au fait que les 3 premiers modèles fournissent le même arbre. Ce dernier est représenté à la figure[6.13]. Nous pouvons observer que pour la branche de droite, seule la caractéristique `win_on_champ` apparaît. Nous pouvons voir dans la branche de gauche que les caractéristiques `nb_party_w_champ` et `winning_losing_series` sont également présente. La présence de `nb_party_w_champ` semble signifier qu'en plus du taux de victoire sur un·e champion·ne, il est également intéressant de savoir si ce·tte dernier·e a été régulièrement choisi·e par le·a joueur·euse.

Regardons maintenant dans le tableau[6.14] les différentes valeurs des scores permettant de mesurer la précision du DecisionTree. Ce qui saute aux yeux de prime abord, c'est le fait que ce modèle est bien meilleur pour prédire une défaite qu'une victoire dans les 3 premiers cas. A contrario, pour le cas 4, ce modèle prédit mieux les victoires. Ainsi, quand nous regardons les F1-Scores, nous pouvons voir que le modèle ne performe pas énormément mieux dans les 3 premiers cas que dans le quatrième. Cependant, les scores AUC eux sont d'une différence significative.

Ainsi, lorsque nous observons les courbes ROC[6.15] associées aux scores AUC, nous pouvons voir que les 3 premiers cas sont toujours bien confondus.

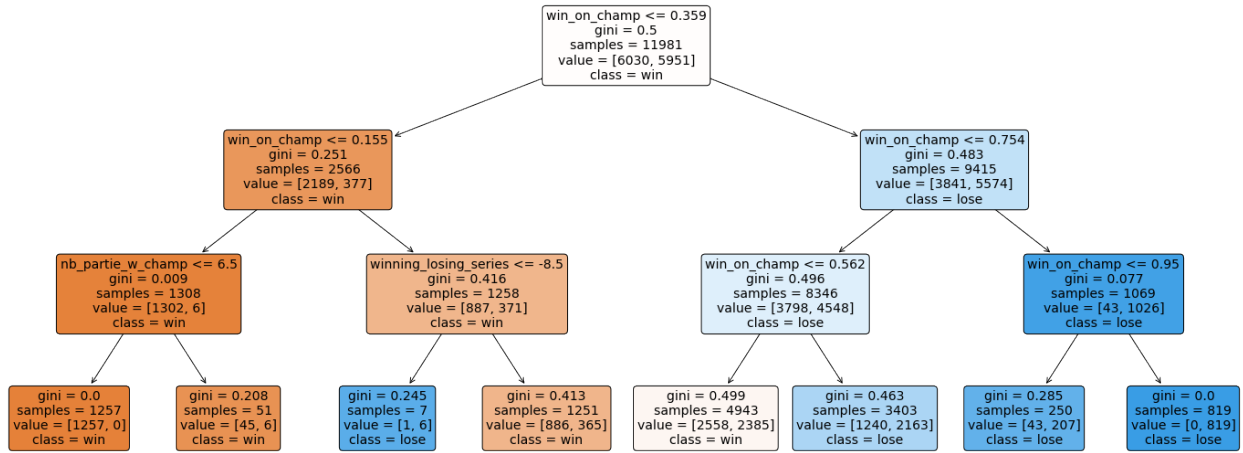


Figure 6.13: Arbre de décision dans les cas 1-2-3

	Cas 1-2-3	Cas 4
Défaite	78.14%	52.31%
Victoire	52.31%	58.04%
F1-Score	59.57%	55.57%
AUC	74.07%	56.31%

Figure 6.14: Scores de précision pour le DecisionTree sur l'ensemble des données

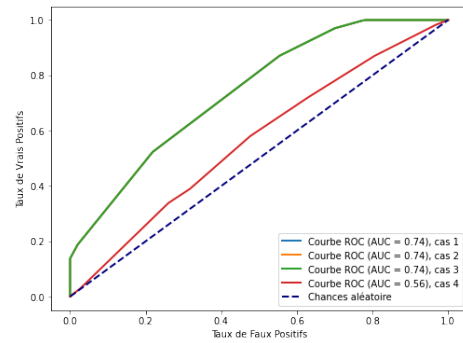


Figure 6.15: Courbes ROC pour le DecisionTree sur l'ensemble des données

Ainsi, comme pour la régression logistique, le DecisionTree est meilleur avec la caractéristique *win_on_champ*. Cependant, ces performances varient grandement si l'on cherche à prédire une victoire ou une défaite. Regardons donc maintenant si cette affirmation reste vraie lorsque nous appliquons le DecisionTree aux données regroupées par division.

6.2.2 Analyse par division

Pour commencer notre analyse par division pour le DecisionTree, commençons par regarder la précision sur la validation croisée (VC) pour les différents modèles ainsi que sur l'ensemble de test (ET)[6.8]. Nous pouvons observer pour commencer que pour les cas 1 et 2, il n'y a aucune différence. Ainsi, retirer ou non les caractéristiques corrélées n'a aucune influence. Ceci est sûrement dû au fait que le DecisionTree produit les mêmes arbres étant donné, l'importance de *win_on_champ* et du fait que la profondeur de l'arbre est limitée à 3, limitant ainsi l'influence d'un certains nombre de caractéristique. De plus, nous pouvons également observer que le cas 3 est le meilleur pour la validation croisée et performe de manière similaire que les deux premiers cas sur l'ensemble de test. Le cas 4 ne semble pas pouvoir faire mieux que l'aléatoire.

Regardons maintenant les matrices de confusions associées aux différentes cas et aux différentes divisions. Les matrices de confusion sont en annexes. (cas 1 et 2:[8.69], cas 3:[8.78], cas 4:[8.87])

	Cas 1		Cas 2		Cas 3		Cas 4	
	VC	ET	VC	ET	VC	ET	VC	ET
Fer	62.05%	59.64%	62.05%	59.64%	63.31%	61.07%	51.42%	48.21%
Bronze	63.33%	62.99%	63.33%	62.99%	64.61%	62.99%	49.97%	51.97%
Argent	65.83%	66.89%	65.83%	66.89%	66.31%	68.25%	50.38%	50.68%
Or	66.06%	68.05%	66.06%	68.05%	66.30%	68.59%	51.89%	53.06%
Platine	64.85%	66.72%	64.85%	66.72%	65.02%	67.48%	51.16%	45.74%
Émeraude	65.48%	62.26%	65.48%	62.26%	65.24%	62.26%	49.59%	51.80%
Diamant	64.18%	60.47%	64.18%	60.47%	64.92%	63.12%	50.44%	51.62%
Maître	62.89%	66%	62.89%	66%	64.31%	66.33%	50.25%	46%

Tableau 6.8: Précision sur la validation croisée et l'ensemble de test pour le DecisionTree par division

Observons ce que nous disent ces matrices en regardant leurs taux de précision pour les victoires et les défaites[6.9]. Commençons par analyser les cas 1 et 2. Nous pouvons observer que les performances sont généralement très disparates quand il s'agit de prédire une victoire ou une défaite. Ainsi, en Fer et en Maître, le modèle dépasse les 80% pour les victoires mais fait moins bien que l'aléatoire pour les défaites. En Bronze, nous observons la tendance inverse. Ainsi, seules les divisions Or et Platine parviennent à prédire de manière relativement similaire les défaites et les victoires.

Pour le cas 3, nous pouvons voir que pour la division Bronze, il n'y a aucune différence avec les cas 1 et 2. Le modèle dans ce cas ne parvient toujours pas à prédire correctement les défaites pour la catégories Maître. Peut être que ce phénomène est dû au fait que les victoires représentent 54.26% des éléments dans cette division. Notons également que pour les divisions Bronze, Émeraude et Maître, le modèle classe les victoires exactement de la même façon.

Regardons maintenant le cas 4. L'information la plus flagrante est le fait que ce modèle, comme nous pouvions nous y attendre au vu de ces performances, semble prendre ces décisions de manière aléatoire.

	Cas 1 et 2		Cas 3		Cas 4	
	Défaite	Victoire	Défaite	Victoire	Défaite	Victoire
Fer	32.16%	88.32%	64.33%	57.66%	47.55%	48.90%
Bronze	76.35%	47.53%	76.35%	47.53%	53.10%	50.67%
Argent	61.27%	71.56%	74.37%	60.90%	55.31%	45.11%
Or	69.88%	66.44%	72.20%	65.42%	52.51%	53.89%
Platine	64.12%	69.28%	66.79%	68.16%	45.03%	46.44%
Émeraude	72.03%	51.61%	72.03%	51.61%	51.64%	51.97%
Diamant	54.59%	66.66%	64.36%	61.81%	51.72%	51.51%
Maître	40.94%	84.39%	41.73%	84.39%	44.09%	47.39%

Tableau 6.9: Précision pour les victoires et les défaites pour le DecisionTree par division

Essayons de comprendre si ces performances hétéroclites sont dues à la répartition des données par classe. Pour cela, commençons par regarder les F1-Scores[6.10]. Nous pouvons voir que les F1-Score dans les cas 1, 2 et 3 sont de similaire à identique pour toutes les divisions sauf Fer. Pour le cas 4, aucune division ne semblent faire significativement mieux que le hasard.

Si nous regardons les scores AUC[6.11] et les graphes associés[6.20], nous pouvons bien

	Cas 1 et 2	Cas 3	Cas 4
Fer	68.16%	59.17%	48.02%
Bronze	54.35%	54.35%	49.45%
Argent	62.69%	63.52%	45.36%
Or	68.89%	68.92%	55.01%
Platine	67.76%	67.91%	46.35%
Émeraude	56.69%	56.69%	50.78%
Diamant	62.14%	62.00%	50.89%
Maître	74.11%	74.30%	50.31%

Tableau 6.10: F1-Scores

	Cas 1 et 2	Cas 3	Cas 4
Fer	65.82%	66.72%	47.16%
Bronze	70.77%	70.79%	50.72%
Argent	72.97%	74.90%	50.65%
Or	76.04%	76.37%	52.39%
Platine	72.98%	74.39%	45.34%
Émeraude	69.43%	69.46%	51.49%
Diamant	66.86%	69.75%	49.82%
Maître	68.17%	69.33%	47.07%

Tableau 6.11: AUC

évidemment observer que les graphes sont les mêmes dans le cas 1 et le cas 2. Dans ces deux cas, la division Or est celle qui est la mieux prédite. Les performances pour les divisions Bronze, Fer, Émeraude, Diamant et Maître sont similaires. Dans le cas 3, les divisions précédemment citées performent toujours de manière similaire. Cependant, cette fois, les divisions Or, Argent et Platine sont plus similaires. Pour finir, dans le cas 4, seul la division Or semble faire légèrement mieux que l'aléatoire.

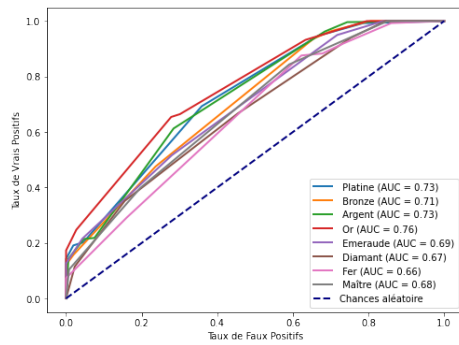


Figure 6.16: Cas 1

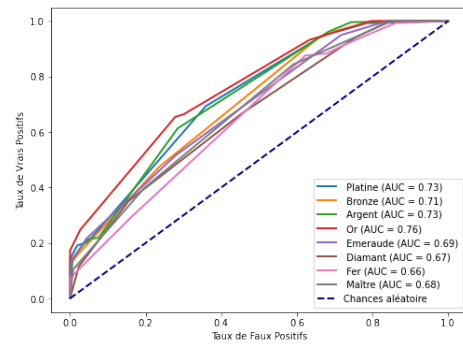


Figure 6.17: Cas 2

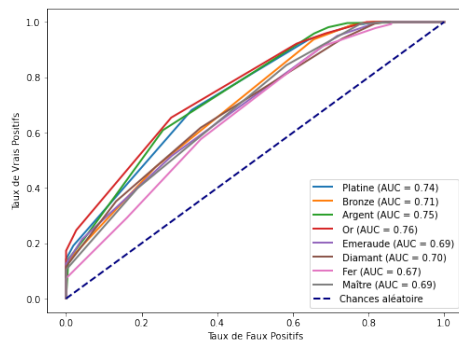


Figure 6.18: Cas 3

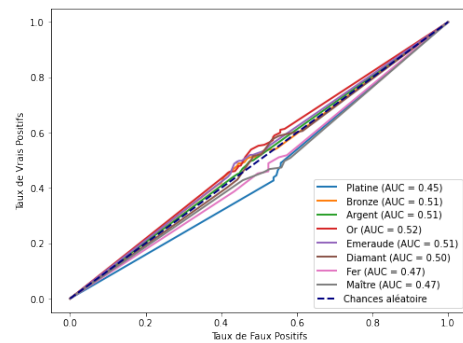


Figure 6.19: Cas 4

Figure 6.20: Courbes ROC par division pour le DecisionTree

De plus, lorsque nous comparons le DecisionTree par division à celui sur l'ensemble des données, nous pouvons voir que même si elles sont similaires, les performances ne mettent pas en avant les mêmes informations. Mettant ainsi encore une fois en évidence, l'importance de séparer les données par division afin de faire des prédictions intéressante.

Résumé: Le DecisionTree fourni des résultats similaires pour les 3 premiers cas sur l'ensemble des données. L'arbre trouvé étant le même à chaque fois. Ainsi, pour l'ensemble de test, lors de l'analyse sur l'ensemble des données, il fournit une précision de 65.64%.

Le DecisionTree ne fait aucune distinction entre les cas où toutes les caractéristiques sont conservées et ceux où les caractéristiques corrélées à plus de 60% sont retirées. Dans la division Fer, le modèle classe majoritairement les parties comme des victoires. Lorsque la caractéristique *win_on_champ* est retirée, les résultats sont toujours moins bon avec une précision inférieure à 60%. Pour optimiser les résultats, il est recommandé d'utiliser les cas 1 ou 2 pour les divisions Fer, Bronze, Émeraude et Diamant, avec des précisions respectives de 59.64%, 62.99%, 62.26% et 60.47%. Pour les divisions Argent, Or, Platine et Maître, il vaut mieux choisir le cas 3, utilisant uniquement *win_on_champ*, avec des précisions respectives de 68.25%, 68.59%, 67.46% et 66.33%.

En conclusion, garder uniquement la caractéristique *win_on_champ* n'améliore ni ne détériore significativement les performances. En revanche, la retirer mène à des résultats aléatoires.

6.3 RandomForest

Regardons maintenant le modèle du RandomForest. Étant donné qu'il s'agit là d'un modèle basé sur le DecisionTree, on s'attend à ce que les résultats soient de au minimum similaires à meilleurs. Afin de faire les analyses présentées dans cette partie, nous avons utilisés la fonction "RandomForestClassifier" de sklearn. Nous avons utilisé les paramètres $n_estimators = 400$, $criterion = 'gini'$, $max_depth = 4$ et $random_state=42$. Le paramètre $n_estimators$ correspond au nombre d'arbres dans la forêt. Ensuite, le paramètre $criterion$ est de la même façon que pour le DecisionTree la fonction de mesure de la qualité des divisions. Pareillement, le paramètre max_depth correspond à la profondeur maximum de l'arbre. Nous pouvons observer que les arbres de la RandomForest sont plus profonds de 1 étage que ceux du DecisionTree. Et à nouveau, pour finir, le paramètre $random_state$ contrôle l'aléatoire et permet la reproductibilité des résultats.

6.3.1 Analyse sur l'ensemble des données

Commençons par regarder les scores sur la validation croisée ainsi que sur l'ensemble de test[6.12]. Nous pouvons voir que pour la validation croisée, le modèle performe mieux lorsque l'on retire les variables corrélées à plus de 60% et mieux encore lorsque on l'entraîne uniquement sur la caractéristique *win_on_champ*. Cependant, lorsque l'on confronte le modèle à des données nouvelles et encore non-vues, nous pouvons observer que le cas 2 fait finalement moins bien que le cas 1. Cependant, le cas 3 reste le meilleur et le cas 4 permet à nouveau de mettre en évidence l'importance prédictive de la caractéristique *win_on_champ*.

	Validation croisée	Ensemble de test
Cas 1	65.84%	65.64%
Cas 2	65.97%	65.47%
Cas 3	67.17%	66.27%
Cas 4	54.70%	54.19%

Tableau 6.12: Précision pour le DecisionTree sur l'ensemble des données

Afin de mieux comprendre ce qu'il se passe, regardons maintenant les matrices de confusions associées aux différents cas[6.25]. Nous pouvons directement observer que dans le cas 4, le modèle réparti de manière aléatoire les victoires dans les classes. Cependant, il semble bien performer pour les défaites. Regardons le tableau des taux de précisions pour les victoires et les défaites afin de savoir si ce ressenti se confirme.

Nous pouvons voir [6.26] qu'au final, dans les deux premiers cas, le RandomForest ne parvient pas à prédire correctement les défaites. Le cas 3 y arrive mieux qu'eux, mais moins bien que le cas 4. Ce dernier quant à lui, comme dit plutôt, ne parvient pas à prédire correctement les victoires. Cependant, nous pouvons observer que le F1-Scores sont plus élevés dans les cas 1 et 2 que dans le cas 3, nous permettant ainsi de supposer que les faibles capacités de précision des défaites sont dues à la répartition des éléments dans l'ensemble de test. Lorsque nous regardons les scores AUC et les courbes associées[6.27], nous pouvons observer que le cas 3 se distingue des autres. En effet, même si les cas 1 et 2 sont bien similaires et proche du cas 3, ils n'en restent pas moins inférieurs. Le cas 4 ne performe pas bien.

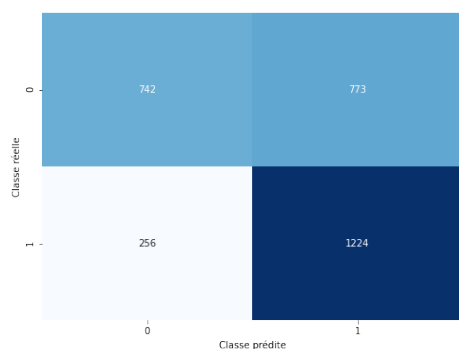


Figure 6.21: Cas 1

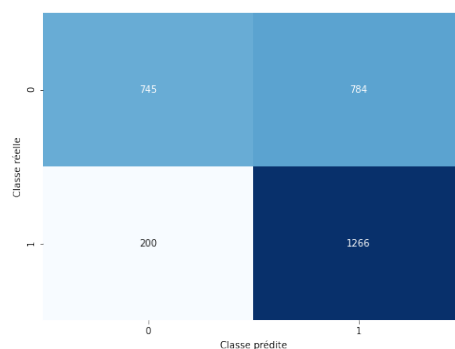


Figure 6.22: Cas 2

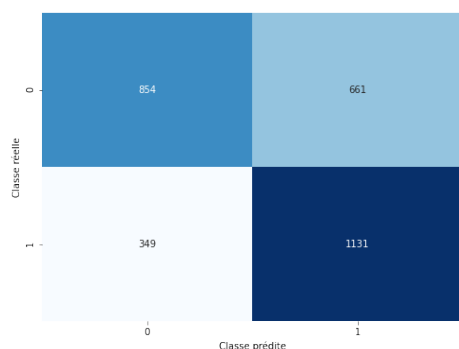


Figure 6.23: Cas 3

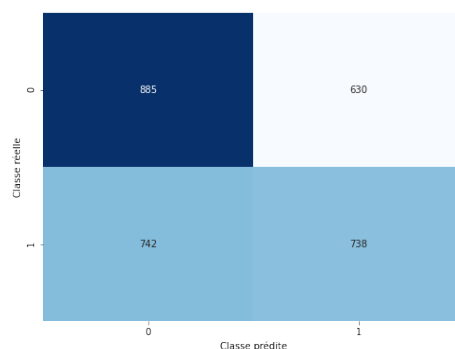


Figure 6.24: Cas 4

Figure 6.25: Matrices de confusion pour le RandomForest sur l'ensemble des données

6.3.2 Analyse par division

Regardons maintenant les performances du RandomForest si l'on travaille sur les divisions individuelles. Commençons par comparer les scores sur les validations croisées (VC) et sur l'ensemble de test (ET)[6.13]. Nous pouvons observer que le cas 3 est celui qui produit de meilleurs résultats. Cependant, la différence avec les cas 1 et 2 n'est pas significative. Nous pouvons aussi remarquer que pour la division Fer, dans le cas 2 et le cas 3, il n'y a aucune différence. Nous pouvons également observer que dans les 3 premiers cas, le RandomForest performe aussi bien sur la validation croisée que sur l'ensemble de test. Ceci tend à montrer que le modèle a trouvé les bons arbres et peut donc correctement prédire des nouvelles informations. Quand nous regardons le cas 4, nous pouvons voir que cette remarque est fausse. En effet, là où durant la validation croisée, le cas 4 a des performances correctes allant de 61 à 65%, sur l'ensemble de test, les résultats du modèle s'effondrent. Le pic d'effondrement se trouve en Diamant où le modèle perd 16% de précision entre la validation croisée et l'ensemble de test.

Regardons les matrices de confusions afin de mieux comprendre les résultats obtenus. Ces dernières se trouvent en annexe par soucis de lisibilité. Afin de mieux comprendre ces matrices, analysons les scores de précision pour les défaites et les victoires que nous pouvons en tirer[6.14].

Nous pouvons observer dans les 3 premiers cas une tendance significative du RandomForest à mieux prédire les victoires que les défaites. Dans le cas 4, nous pouvons observer qu'en maître en en diamant, les scores de précision pour les défaites sont sous les 20%. Ceci vient du fait que le modèle considère 80% des données comme des victoires. Regardons les F1-Scores pour voir si ces particularités sont du à la répartition initiale des données[6.15]. Nous pouvons observer que dans les 3 premiers cas, les différences en termes de précision ne sont pas marquantes. Pour le cas 4, les scores sont mitigés. Ainsi, il semblerait que la caractéristique *win_on_champ* soit

	Défaite	Victoire	F1-Score	AUC
Cas 1	48.97%	82.70%	70.41%	72.96%
Cas 2	46.79%	84.59%	70.77%	73.67%
Cas 3	56.36%	76.41%	69.13%	75.29%
Cas 4	58.41%	49.86%	51.82%	56.46%

Figure 6.26: Scores de précision pour le RandomForest sur l'ensemble des données

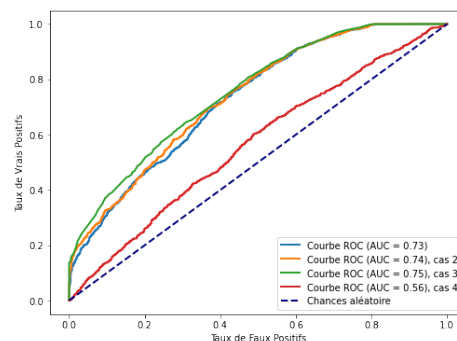


Figure 6.27: Courbes ROC pour le RandomForest sur l'ensemble des données

	Cas 1		Cas 2		Cas 3		Cas 4	
	VC	ET	VC	ET	VC	ET	VC	ET
Fer	61.69%	63.21%	62.14%	62.5%	61.69%	62.5%	61.69%	56.07%
Bronze	64.43%	62.99%	64.43%	64.86%	65.15%	62.78%	64.42%	58.00%
Argent	64.60%	67.91%	64.73%	68.77%	66.22%	68.25%	64.60%	53.92%
Or	64.87%	66.96%	64.96%	66.96%	66.68%	68.59%	64.87%	51.26%
Platine	64.27%	65.78%	64.64%	66.72%	66.30%	66.35%	64.26%	52.93%
Émeraude	64.43%	62.43%	64.43%	60.89%	65.63%	63.80%	64.43%	49.91%
Diamant	64.08%	61.06%	64.82%	61.06%	65.04%	62.83%	64.08%	48.08%
Maître	62.75%	65.33%	63.16%	66%	62.58%	65.66%	62.75%	53%

Tableau 6.13: Précision sur la validation croisée et l'ensemble de test pour le RandomForest par division

capitale pour la prédiction de la division Argent, mais beaucoup moins pour les divisions Fer et Émeraude et presque pas pour Maître.

Regardons pour finir les scores AUC[6.16] et les courbes ROC associées[6.39]. Nous pouvons voir que pour le cas 4, pour la division Maître, malgré un score honnête de 65.52% pour le F1-Score, le score AUC quant à lui est le seul, toutes divisions et tous cas confondu, à être sous les 50%. Globalement, c'est le cas 3 qui a le meilleure score AUC.

Nous pouvons observer sur les courbes ROC que comme attendu, dans le cas 4, les courbes sont très proches des chances aléatoires. Il semblerait au vu de ces dernières que le cas 3 soit le meilleur.

	Cas 1		Cas 2		Cas 3		Cas 4	
	Défaite	Victoire	Défaite	Victoire	Défaite	Victoire	Défaite	Victoire
Fer	58.74%	67.88%	58.04%	67.15%	61.53%	63.50%	49.65%	62.77%
Bronze	51.16%	76.68%	48.44%	83.85%	50.38%	77.13%	64.34%	50.67%
Argent	59.06%	78.57%	58.75%	80.82%	72.81%	62.78%	64.06%	41.72%
Or	54.82%	77.62%	52.89%	79.32%	72.20%	65.42%	67.18%	37.28%
Platine	50.76%	80.52%	49.61%	83.52%	55.34%	77.15%	55.34%	50.56%
Émeraude	42.10%	84.58%	39.47%	84.22%	47.36%	81.72%	30.92%	70.60%
Diamant	36.78%	86.6%	35.05%	88.48%	37.93%	89.09%	18.39%	79.39%
Maître	40.94%	83.23%	40.15%	84.97%	48.03%	78.61%	19.68%	77.45%

Tableau 6.14: Scores de précision pour la prédiction des victoires et des défaites par division pour le RandomForest

	Cas 1	Cas 2	Cas 3	Cas 4		Cas 1	Cas 2	Cas 3	Cas 4
Fer	64.35%	63.66%	62.36%	58.30%	Fer	67.17%	67.22%	68.24%	56.02%
Bronze	65.76%	68.87%	65.77%	52.80%	Bronze	71.53%	71.99%	72.49%	58.07%
Argent	68.97%	70.14%	64.23%	45.12%	Argent	75.07%	75.64%	77.41%	55.52%
Or	71.45%	71.88%	68.92%	44.89%	Or	74.08%	75.29%	77.99%	56.21%
Platine	70.37%	71.70%	69.83%	52.02%	Platine	73.28%	73.47%	76.18%	54.91%
Émeraude	68.30%	67.33%	68.36%	57.43%	Émeraude	68.47%	69.01%	71.71%	51.21%
Diamant	68.42%	68.86%	70%	59.81%	Diamant	65.75%	66.16%	71.16%	51.01%
Maître	73.46%	74.24%	72.53%	65.52%	Maître	66.39%	66.82%	71.41%	47.36%

Tableau 6.15: F1-Score par division pour le RandomForest

Tableau 6.16: Scores AUC par division pour le RandomForest

Résumé: Pour le RandomForest, sur l'ensemble des données, il est difficile de prédire correctement les défaites. La meilleure précision sur l'ensemble de test est de 66.27% dans le cas où l'on ne garde que la caractéristique *win_on_champ*. Il n'y a pas de différences significatives à garder ou retirer l'ensemble des autres caractéristiques. Ainsi, *win_on_champ* est une caractéristique extrêmement prédictive. Si on l'enlève, la précision sur l'ensemble de test passe à 54.19%.

Pour les divisions Fer, il vaut mieux garder l'ensemble des données, ce qui donne une précision sur l'ensemble de test de 67.88%. Pour les divisions Bronze, Argent, Or, Platine et Maître, il vaut mieux supprimer les caractéristiques corrélées à plus de 60%, afin d'obtenir sur l'ensemble de test une précision de respectivement 64.86%, 64.73%, 66.96%, 66.72%, 66%. Pour les divisions Émeraude et Diamant, il vaut mieux ne garder que la caractéristique *win_on_champ* afin cette fois d'obtenir des précisions de 63.80% et 62.83%. Sans la caractéristique *win_on_champ*, le RandomForest ne produit aucun bon résultat.

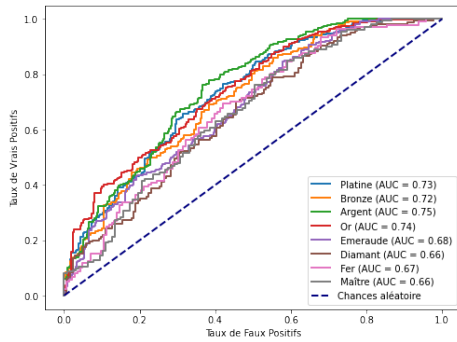


Figure 6.28: Cas 1

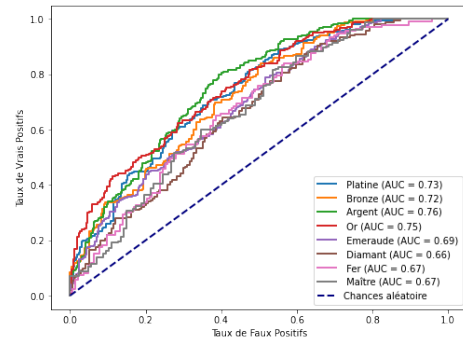


Figure 6.29: Cas 2

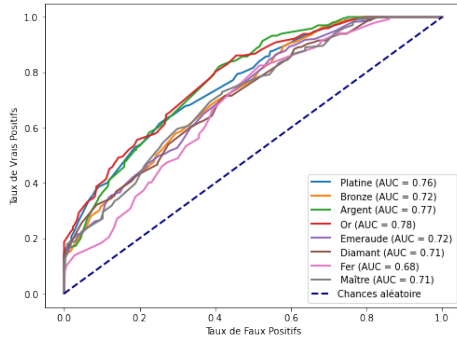


Figure 6.30: Cas 3

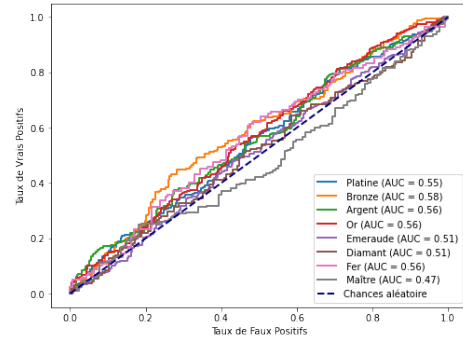


Figure 6.31: Cas 4

Figure 6.32: Courbes ROC par division pour le RandomForest

6.4 K-Nearest Neighbors (KNN)

Afin d'effectuer l'algorithme du KNN, nous avons utilisé la librairie sklearn, et plus précisément la fonction `KNeighborsClassifier`, avec les paramètres $n_neighbors = 10$ et $p=45$. Le paramètre $n_neighbors$ désigne le nombre de voisins pris en compte par l'algorithme. Le paramètre p quant à lui est le paramètre de puissance de Minkowski. Le reste des paramètres est laissé à leur valeur par défaut.

6.4.1 Analyse sur l'ensemble des données

Commençons comme à l'accoutumée par regarder la précision du KNN sur l'ensemble de test et la validation croisée[6.17]. Nous pouvons observer que les scores de ce modèle sont décevants sur les cas 1, 2 et 4, atteignant à peine la moitié des classifications. Seule le cas 3 se démarque et met en avant le fait que les caractéristiques autre que *win_on_champ* ne font qu'apporter du bruit. De plus, les cas 1, 2 et 4 semblent performer de la même manière sur l'ensemble de test.

	Validation croisée	Ensemble de test
Cas 1	50.62%	51.95%
Cas 2	50.48%	51.95%
Cas 3	64.02%	63.43%
Cas 4	50.62%	51.95%

Tableau 6.17: Précision pour le DecisionTree sur l'ensemble des données

Regardons les matrices de confusion afin de voir si cette affirmation est vraie. Ces dernières

se trouvent en annexe[8.92], mais nous pouvons voir sur le tableau[6.33] si dessous les données importantes. Lorsque nous observons les matrices de confusion, nous pouvons voir que le score obtenu sur l'ensemble de test est dû au fait que la majorité des données est classé comme des défaites par le KNN. Nous pouvons observer sur le tableau résumant l'ensemble des scores permettant de juger la précision du modèle[6.33], que dans tous les cas, le KNN peine à prédire les victoires mais ne s'en sort pas trop mal pour les défaites. Cependant, comme nous venons de le dire, cette performance apparente pour prédire les défaites est essentiellement due au fait que le modèle classe une majorité des parties comme des défaites. C'est pour ceci que dans les cas 1,2 et 4, les F1-Scores sont inférieurs à 50%.

	Défaite	Victoire	F1-Score	AUC
Cas 1	65.19%	38.36%	44.07%	51.30%
Cas 2	60.77%	42.89%	46.84%	51.96%
Cas 3	78.83%	47.63%	56.25%	72.73%
Cas 4	65.19%	38.36%	44.07%	51.30%

Figure 6.33: Scores de précision pour le KNN sur l'ensemble des données

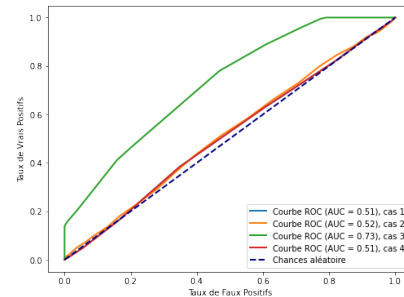


Figure 6.34: Courbes ROC pour le KNN sur l'ensemble des données

Lorsque nous observons la courbe ROC[6.34] et les scores AUC associés, nous pouvons observer que pour les cas 1,2 et 4, les courbes et les scores sont très similaires mais ne font pas vraiment mieux que l'aléatoire. Seul le cas 3 se distingue et semble apte à classer correctement les parties.

6.4.2 Analyse par division

Regardons maintenant si le KNN est capable de faire mieux lorsqu'il est appliqué non pas à toutes les divisions confondues, mais pour chaque division individuelle. Sur le tableau [6.18] ci-dessous, nous pouvons observer les résultats fournis par le KNN par division sur la validation croisée (VC) et l'ensemble de test (ET).

La première chose que nous pouvons observer est les résultats inférieur à 50%. Ces résultats implique que le modèle fait moins bien que l'aléatoire. Ceci est vrai pour la division Or sur le premier cas pour la validation croisée, mais pas sur l'ensemble de test. Ceci est également vrai pour le cas 2 par la division Argent, Or, Émeraude, Diamant, Maître sur la validation croisée. Pour toutes les divisions sauf Maître, sur le cas 2, le modèle a mieux performé sur l'ensemble de test. Nous pouvons supposer que cette diminution de précision du cas 1 au cas 2 implique le fait que la suppression des caractéristiques corrélées à plus de 60% produit une trop grande perte d'informations. Pour le cas 4, la division Or sur la validation croisée fait moins bien que l'aléatoire et que sur l'ensemble de test. Ce qui est l'inverse pour la division Platine.

Pour le cas 3, toutes les précisions sont supérieures à 50%, mettant ainsi encore une fois en exergue l'importance prédictive de la caractéristique *win_on_champ*.

Regardons maintenant si les matrices de confusions peuvent expliquer ces scores. Nous pouvons observer que comme pour les résultats obtenus sur l'ensemble des données, nous avons des scores si faibles parce que le KNN a tendance à tout classer comme des défaites.

Lorsque nous analysons le tableau contenant les taux de précision pour les victoires et les défaites[6.20], nous pouvons observer que les scores pour prédire les défaites sont bien meilleurs

	Cas 1		Cas 2		Cas 3		Cas 4	
	VC	ET	VC	ET	VC	ET	VC	ET
Fer	53.12%	55%	50.08%	50.71%	58.48%	58.92%	53.12%	55%
Bronze	50.18%	52.39%	52.62%	51.76%	61.51%	58.41%	50.18%	52.39%
Argent	51.74%	50%	49.52%	52.04%	62.38%	65.69%	51.74%	50%
Or	49.09%	51.98%	49.72%	52.70%	63.29%	60.64%	49.09%	51.98%
Platine	52.01%	49.71%	51.44%	51.41%	64.21%	65.78%	52.01%	49.71%
Émeraude	50.11%	52.83%	49.42%	54.71%	61.56%	58.83%	50.11%	52.83%
Diamant	52.06%	50.44%	48.66%	52.80%	60.61%	63.12%	52.06%	50.44%
Maître	51%	44%	46.91%	46.66%	59.08%	61.33%	51%	44%

Tableau 6.18: Précision sur la validation croisée et l'ensemble de test pour le KNN par division

que ceux pour prédire les victoires. Parmi tous les cas, seule la division Argent pour le cas 3 présente des résultats satisfaisants. Regardons si ces faibles scores sont dus à la répartition des classes en observant les F1-Scores, ainsi que les scores AUC et leurs courbes ROC associées.

	Cas 1		Cas 2		Cas 3		Cas 4	
	Défaite	Victoire	Défaite	Victoire	Défaite	Victoire	Défaite	Victoire
Fer	62.93%	46.71%	55.94%	45.25%	70.62%	46.71%	62.93%	46.71%
Bronze	62.40%	40.80%	62.40%	39.46%	50.77%	67.26%	62.40%	40.80%
Argent	62.81%	34.58%	67.18%	33.83%	60.93%	71.42%	62.81%	34.58%
Or	65.25%	40.33%	62.54%	44.06%	67.95%	54.23%	65.25%	40.33%
Platine	66.41%	33.33%	65.26%	37.82%	55.72%	75.65%	66.41%	33.33%
Émeraude	57.89%	47.31%	65.13%	43.36%	80.92%	34.76%	57.89%	47.31%
Diamant	54.59%	46.06%	58.04%	47.27%	62.64%	63.63%	54.59%	46.06%
Maître	44.09%	43.93%	51.18%	43.35%	51.96%	68.21%	44.09%	43.93%

Tableau 6.19: Taux de précision pour les défaites et les victoires pour le KNN par division

Résumé: Le modèle du KNN est en grande difficulté pour prédire les résultats des parties sur les cas 1, 2 et 4[6.1] Dans ces trois cas, les modèles considèrent une majorité des parties comme étant des défaites.

La meilleure performance revient donc au modèle tournant uniquement sur la caractéristique *win_on_champ* avec un score de 63.43%. Pour toutes les divisions, le meilleur score sur l'ensemble de test est fourni par le cas 3. Le tableau ci dessous résume les valeurs associées de précision[6.21]. Plus encore que pour les autres modèles, le KNN démontre l'importance de la caractéristique *win_on_champ*. De ce fait, à moins que cette dernière soit la seule du modèle, le KNN est incapable de fournir des résultats significativement meilleur que l'aléatoire. Pour toutes les divisions sauf Argent et Platine, ayant respectivement une précision de 65.49% et 65.78%, il vaut mieux avec le KNN, prédire les résultats d'une partie en se servant de l'ensemble des données.

	Cas 1		Cas 2		Cas 3		Cas 4	
	F1-Score	AUC	F1-Score	AUC	F1-Score	AUC	F1-Score	AUC
Fer	50.39%	55.29%	47.32%	49.59%	52.67%	65.29%	50.39%	55.29%
Bronze	44.28%	51.43%	43.13%	51.60%	60.00%	67.55%	44.28%	51.43%
Argent	38.57%	49.11%	39.04%	51.46%	65.40%	73.24%	38.57%	49.11%
Or	47.22%	52.92%	49.80%	54.27%	59.47%	71.63%	47.22%	52.92%
Platine	40.09%	48.95%	44.01%	52.00%	69.05%	70.32%	40.09%	48.95%
Émeraude	48.97%	51.30%	47.82%	55.02%	44.70%	66.03%	48.97%	51.30%
Diamant	47.50%	50.04%	49.36%	52.27%	62.68%	68.77%	47.50%	50.04%
Maître	47.50%	43.29%	48.38%	46.35%	67.04%	66.46%	47.50%	43.29%

Tableau 6.20: F1-Scores et AUC scores pour le KNN par division

	Fer	Bronze	Argent	Or	Platine	Émeraude	Diamant	Maître
Précision	58.92%	58.41%	65.69%	60.64%	65.78%	58.83%	63.12%	61.33%

Tableau 6.21: Précision sur l'ensemble de test sur le cas 3 pour le KNN

6.5 Support Vector Classification (SVC)

Lorsque nous avons tenté d'obtenir les meilleures performances possibles pour le SVC, nous nous sommes rendus compte que ces dernières arrivaient lorsque nous utilisons un noyau linéaire. C'est pour cette raison que nous avons décidé d'utiliser la fonction `LinearSVC` de `sklearn` avec les paramètres `random_state = 42`, `max_iter = 1000000`, `penalty = 'l1'`, `loss = 'squared_hinge'`, `dual = False` et `C = 15`. Comme toujours, le paramètre `random_state` permet la reproductibilité des tests. Le paramètre `max_iter` indique le nombre maximum d'itération. Il est à une grande valeur afin de permettre la convergence. `penalty` indique la norme utilisé pour la pénalisation. Lui imposer la valeur "l1" indique que le vecteur des coefficients est un vecteur creux, la majorité des éléments étant donc nul. Nous imposons le paramètre `loss` à 'squared_hinge', nous prenons la racine carrée de la fonction de perte Hinge. Il faut choisir `dual` comme faux quand le nombre de données est supérieur à celui de caractéristique. Pour finir, le paramètre `C` représente la régularisation. La force de la régularisation est inversement proportionnelle à `C`.

6.5.1 Analyse sur l'ensemble des données

Regardons donc pour commencer comment performe le `LinearSVC` sur la validation croisée et l'ensemble de test. Nous pouvons voir en observant le tableau[6.22] que même si les valeurs des précisions sur les 3 premiers cas pour la validation croisée ne diffère que de 0.05%, elles ont au final une différence de 1.07% sur l'ensemble de test. Ce qui multiplie ainsi la différence entre les deux par 21. Le cas 4 fait quant à lui plus de 10% moins bien que ces homologues.

	Validation croisée	Ensemble de test
Cas 1	66.79%	67.84%
Cas 2	66.74%	68.01%
Cas 3	66.79%	66.94%
Cas 4	54.58%	54.89%

Tableau 6.22: Précision pour le LinearSVC sur l'ensemble des données

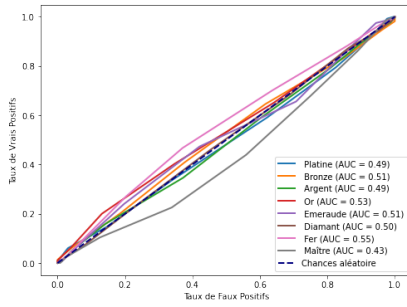


Figure 6.35: Cas 1

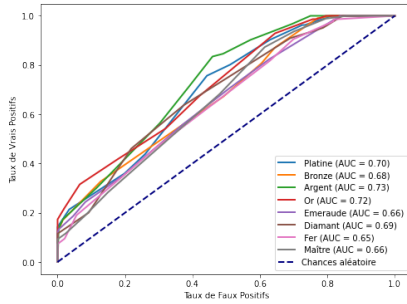


Figure 6.37: Cas 3

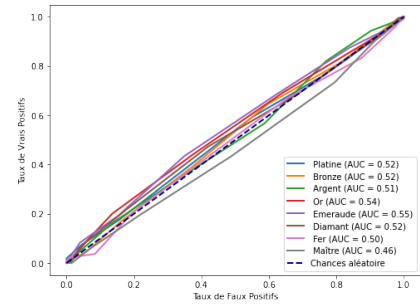


Figure 6.36: Cas 2

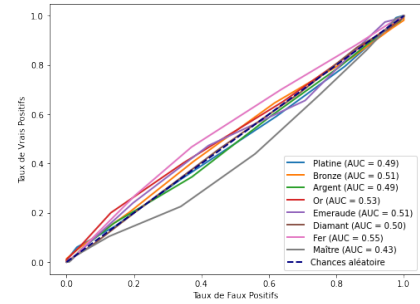


Figure 6.38: Cas 4

Figure 6.39: Courbes ROC par division pour le KNN

Regardons les matrices de confusion afin de voir et de comprendre comment les données ont été triées[6.44]. Nous pouvons observer que dans les cas 1 et 2, les données sont triées de manière similaire. Le cas 3 semble un peu plus avoir du mal à considérer les défaites comme telles mais semble faire légèrement mieux pour les victoires que les cas 1 et 2. Le cas 4 quant à lui considère 62.77% des données comme des défaites, soit une écrasante majorité.

Nous allons maintenant regarder plus précisément les pourcentages de victoire et de défaites bien classées.[6.45]. Nous pouvons observer, comme nous le pensions, que pour les cas 1 et 2, le LinearSVC arrive aussi bien à trier les défaites que les victoires. Le cas 3 quant à lui, à une nette facilité à trier les victoires mais peine plus sur les défaites. Cependant, il arrive quand même à un score honorable de 56.02% de défaite bien classée. A l'inverse, le cas 4 a plus facile à trouver les défaites que les victoires, ce qui s'explique par le fait qu'il a une certaine tendance à considérer toutes les parties comme des défaites.

Lorsque nous regardons les F1-scores, nous pouvons étonnement voir que c'est le cas 3 qui à le meilleur résultat. Il semblerait donc que ces performances mitigées soient dues à l'inégalité de répartition initiales des données. Les cas 1 et 2 performent de manière similaire. Le cas 4 quant à lui se distingue par un score inférieur à 50%, mettant ainsi encore une fois en évidence le fait qu'il n'arrive pas à bien classer les données et que ces performances sont dues à la répartition initiales des parties entre les victoires et les défaites.

Pour finir, attardons nous sur les scores AUC et leurs courbes ROC associées[6.46]. A nouveau, nous pouvons observer que les cas 1, 2 et 3 ont des scores AUC similaires, même si le cas 2 semble se démarquer en étant légèrement meilleur. Ce qui n'est pas significatif, quand nous regardons à quel point les 3 courbes sont confondues. Le cas 4 quant à lui persiste et signe à être moins bon que les autres.

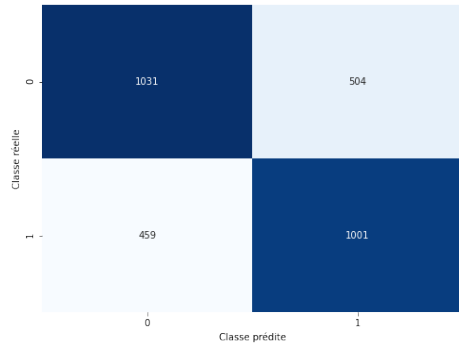


Figure 6.40: Cas 1

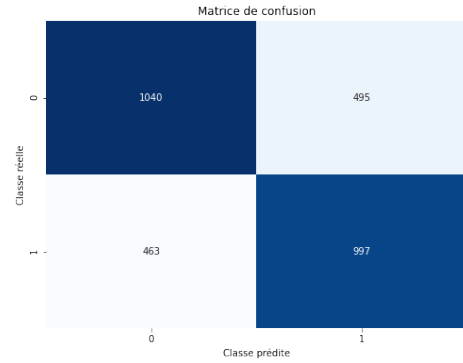


Figure 6.41: Cas 2

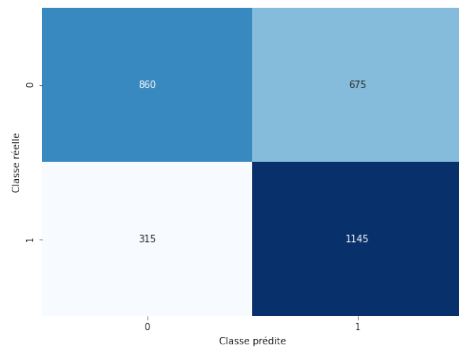


Figure 6.42: Cas 3

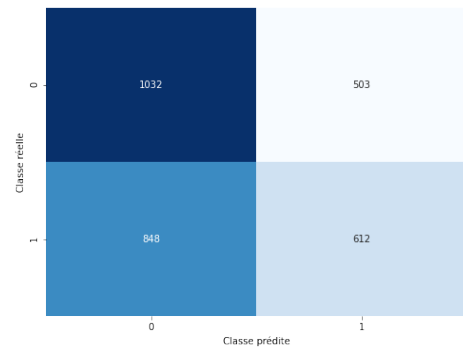


Figure 6.43: Cas 4

Figure 6.44: Matrices de confusion pour le LinearSVC sur l'ensemble des données

6.5.2 Analyse par division

Regardons maintenant si le LinearSVC performe autant lorsqu'il est appliqué à chaque division individuellement plutôt que sur l'ensemble des données.

Nous pouvons voir sur le tableau[6.23] les performances pour la validation croisée (VC) et sur l'ensemble de test (ET). Il est à noter que pour la division Maître, sur les cas 1 et 4, il n'y a pas eu convergence. Premièrement, nous pouvons observer que pour le cas 4, même si les performances sur la validation croisée sont similaires aux autres cas, ce n'est pas le cas pour la prédiction sur l'ensemble de test. Il semblerait donc que le LinearSVC n'arrive pas sans la caractéristique *win_on_champ* à correctement prédire des données non-vues. Pour les cas 1 et 2, en général, le modèle performe mieux sur l'ensemble de test que sur la validation croisée. Le cas 3 semble, toutes divisions confondues, donner les meilleurs résultats.

Regardons les matrices de confusions qui sont en annexes et les valeurs des taux de précisions sur les victoires et les défaites[6.25]. Nous pouvons observer que dans le cas 1, le LinearSVC prédit toujours mieux les victoires que les défaites. Ainsi, pour la division Maître, le taux de précision pour les défaites n'est que de 51.96% et de 45.40% pour la division Diamant. Cette tendance se confirme pour le cas 2, même si les scores sont globalement meilleurs. Ce qui est dû au fait que le cas 2 performe légèrement mieux que le cas 1. Dans le cas 3, on distingue une grosse perte de précision pour les défaites pour les divisions Bronze, Émeraude et Diamant, par rapports aux cas 1 et 2. Le cas 4 peine en toute circonstance à produire des résultats satisfaisants. Nous pouvons cependant remarquer que les victoires sont mieux prédites sur les divisions Fer, Émeraude et Diamant. Pour les autres divisions, ce sont les défaites qui sont les mieux prédites.

Comparons maintenant ces résultats aux F1-Scores et ainsi qu'aux scores AUC et à leur

	Défaite	Victoire	F1-Score	AUC
Cas 1	67.16%	68.56%	67.52%	75.65%
Cas 2	67.75%	68.28%	67.54%	75.69%
Cas 3	56.02%	78.42%	69.81%	75.64%
Cas 4	67.23%	41.91%	47.53%	56.20%

Figure 6.45: Scores de précision pour le LinearSVC sur l'ensemble des données

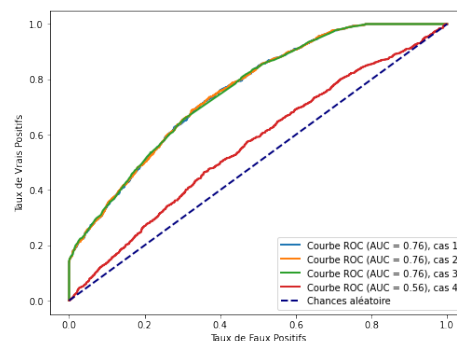


Figure 6.46: Courbes ROC pour le LinearSVC sur l'ensemble des données

	Cas 1		Cas 2		Cas 3		Cas 4	
	VC	ET	VC	ET	VC	ET	VC	ET
Fer	63.03%	64.28%	63.66%	64.28%	65.44%	63.57%	5419%	55.71%
Bronze	65.10%	63.82%	65.26%	65.48%	66.35%	62.78%	5247%	55.09%
Argent	65.62%	69.11%	66.09%	69.79%	67.63%	68.94%	5324%	55.63%
Or	65.91%	68.41%	66.00%	69.49%	67.04%	68.59%	5086%	53.79%
Platine	66.44%	68.62%	66.25%	68.24%	66.96%	66.35%	5380%	53.49%
Émeraude	64.43%	64.83%	64.56%	65.01%	66.96%	63.80%	5276%	52.14%
Diamant	64.60%	62.24%	64.52%	61.65%	66.74%	64.01%	5206%	52.21%
Maître	62.91%	65%	63.16%	65.66%	64.33%	68%	62.91%	53.33%

Tableau 6.23: Précision sur la validation croisée et l'ensemble de test pour le LinearSVC par division

courbe ROC associées. Nous pouvons observer que le cas 1 a le meilleur F1-Score pour les divisions Fer et Platine. Le deuxième cas est a le meilleur F1-Score pour les divisions Bronze et Or. Pour finir, le cas 3 fournit les meilleurs F1-scores pour toutes les autres divisions.

Si nous prenons maintenant le temps d'observer les courbes ROC[6.51], nous pouvons observer, que comme attendu, le cas 4 ne produit pas de résultats très différents de l'aléatoire, ayant même une courbes sous les chances aléatoire pour la division Maître. Il était de même attendu que les courbes des cas 1 et 2 se ressemblent grandement. Nous pouvons observer sur ces deux cas qu'il y a en réalité deux groupes de divisions. Les divisions Or, Platine et Argent qui performant significativement mieux que les autres divisions. Ce constat eut aussi s'appliquer au cas 3 qui fait cependant mieux pour toutes les divisions que tous les autres cas.

	Cas 1		Cas 2		Cas 3		Cas 4	
	Défaite	Victoire	Défaite	Victoire	Défaite	Victoire	Défaite	Victoire
Fer	57.34%	71.53%	58.04%	70.80%	60.83%	66.42%	50.34%	61.31%
Bronze	57.75%	70.85%	58.91%	73.09%	50.38%	77.13%	60.85%	48.43%
Argent	67.18%	71.42%	67.5%	72.55%	60.31%	79.32%	62.5%	47.36%
Or	66.79%	69.83%	67.56%	71.18%	72.20%	65.42%	56.75%	51.18%
Platine	63.35%	73.78%	64.12%	72.28%	55.34%	77.15%	54.19%	52.81%
Émeraude	56.25%	74.19%	55.92%	74.91%	47.36%	81.72%	41.77%	63.44%
Diamant	45.40%	80%	45.40%	78.78%	40.80%	88.48%	39.08%	66.06%
Maître	51.96%	74.56%	53.54%	74.56%	55.90%	76.87%	65.31%	37.01%

Tableau 6.24: Taux de précision pour les défaites et les victoires pour le LinearSVC par division

	Cas 1		Cas 2		Cas 3		Cas 4	
	F1-Score	AUC	F1-Score	AUC	F1-Score	AUC	F1-Score	AUC
Fer	66.21%	68.75%	65.98%	69.00%	64.08%	69.14%	57.53%	55.87%
Bronze	64.48%	70.94%	66.26%	71.52%	65.77%	72.66%	50%	56.45%
Argent	67.73%	77.33%	68.56%	77.59%	69.86%	77.74%	49.21%	55.41%
Or	70.18%	77.82%	71.30%	78.51%	68.92%	78.70%	54.12%	55.23%
Platine	70.35%	75.72%	69.67%	76.23%	69.83%	76.72%	53.40%	56.16%
Émeraude	66.88%	71.15%	67.21%	71.21%	68.36%	72.05%	55.92%	52.28%
Diamant	67.34%	70.12%	66.66%	70.51%	70.53%	72.44%	57.36%	54.78%
Maître	71.07%	69.73%	71.46%	70.45%	73.48%	72.72%	61.74%	49.23%

Tableau 6.25: F1-Scores et scores AUC pour le LinearSVC par division

Résumé: Afin de garantir un équilibre entre précision et victoire et défaites bien classée, il vaut mieux utiliser le modèle sans les données corrélées à plus de 60%. Il a une performance sur l'ensemble de test de 68.01%. Cependant, il a un moins bon F1-Score (67.54%) que le cas où l'on effectue la prédiction uniquement sur la caractéristique *win_on_champ* (69.81%).

Le LinearSVC semble donc également essentiellement dépendre de la caractéristique *win_on_champ*, mais ne considère pas les autres caractéristiques comme du bruit. Sans cette caractéristique, les performances du modèle s'effondrent et il a tendance à considérer une majorité de parties comme des défaites. Afin de maximiser les résultats obtenus, il faut considérer de prendre les divisions Fer et Platine sur l'ensemble des caractéristiques pour obtenir une précision sur l'ensemble de test de respectivement 64.28% et 68.62%. Afin d'avoir les meilleurs résultats pour les divisions Bronze et Or, il vaut mieux se servir du modèle n'ayant pas les caractéristiques corrélées à plus de 60%. Ce qui leur donne respectivement une précision de 65.48% et 69.49% sur l'ensemble de test. Pour finir, pour les divisions restantes, à savoir, Argent, Émeraude, Diamant et Maître, il vaut mieux effectuer une prédiction sur un modèle uniquement entraîné sur la caractéristique *win_on_champ*. Cela leur donne une précision respectivement sur l'ensemble de test de 63.57%, 68.94%, 63.80%, 64.01% et 68%.

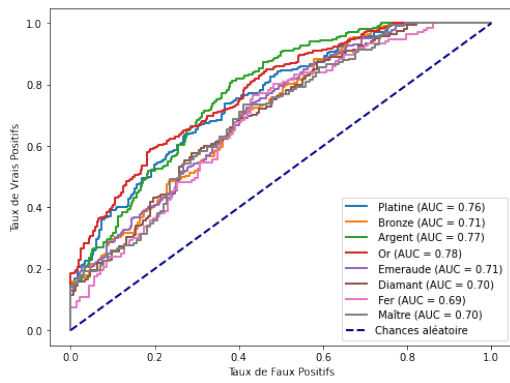


Figure 6.47: Cas 1

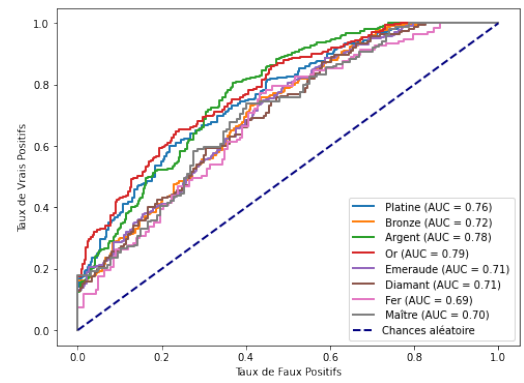


Figure 6.48: Cas 2

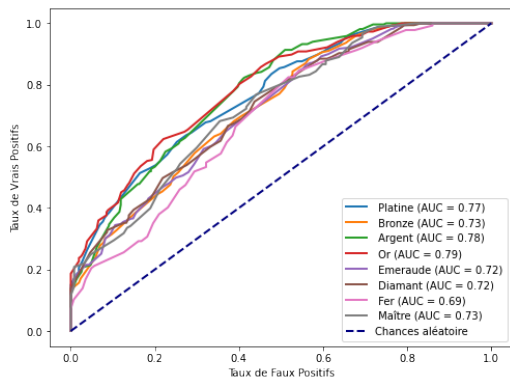


Figure 6.49: Cas 3

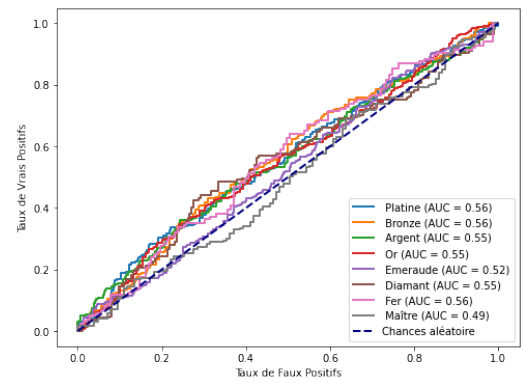


Figure 6.50: Cas 4

Figure 6.51: Courbes ROC par division pour le LinearSVC

Chapitre 7

Résultat final et comparaison des méthodes

Ce chapitre est l'apothéose de ce mémoire. Nous allons maintenant pouvoir utiliser tous les résultats précédemment obtenu. Ainsi, nous pourrons décider quel modèle utiliser dans quel cas et pourquoi, afin de prédire le mieux possible le résultat d'un match en cours.

7.1 Présentation des résultats

Premièrement, commençons par observer dans le tableau[7.1] ci-dessous quel modèle a le mieux performé pour chaque division. Ce tableau met en évidence le fait que les modèles linéaires fonctionnent mieux sur notre problème.

On peut également observer que c'est plus précisément la régression logistique qui fournit les meilleurs résultats. Cependant, souvent, la décision du choix de l'un ou de l'autre ne fut pas facile tant les scores étaient similaire.

Nous pouvons voir aussi qu'exactement pour la moitié des divisions, il serait mieux de ne pas se servir des modèles entraîné par division individuelle, mais sur l'ensemble des données. Nous avons fait cette hypothèse en nous basant sur l'idée suivante: "Si la précision par division individuelle est moins bonne que la précision global, autant garder la précision globale". Nous verrons un peu plus tard si ceci est vrai. Tout d'abord, analysons les divisions ayant de meilleurs performance de manière individuelle. Il s'agit des divisions médianes. Ceci semble indiquer qu'il est plus facile de prédire le comportement des joueur·euse·s moyen que des joueur·euse·s les meilleur·e·s et les moins bon·ne·s.

	Cas	Modèle	Précision
Fer	2	régression logistique, sur l'ensemble des données	66.72%
Bronze	2	régression logistique, sur l'ensemble des données	66.72%
Argent	3	LinearSVC, par division	69.49%
Or	1	régression logistique, par division	70.57%
Platine	1	régression logistique, par division	68.62%
Émeraude	3	LinearSVC, par division	68.84%
Diamant	2	régression logistique, sur l'ensemble des données	66.72%
Maître	3	régression logistique, par division	68%

Tableau 7.1: Meilleure précision pour chaque division

Commençons par les divisions Argent et Émeraude, qui performant mieux sur le LinearSVC. Commençons par regarder l'influence des coefficients dans chaque cas. Sur l'image [7.1], nous avons choisis de ne montrer que les 10 coefficients avec la plus grande valeur absolue. En effet, dès le quatrième coefficient, nous sommes si proches de zéro, que ça ne valait pas la peine de les afficher. Nous pouvons voir que la caractéristique avec la plus grande influence est bel et bien *win_on_champ*, avec une magnitude de 2.12. Elle est suivie par les caractéristiques *winrate_blue* et *winrate_red*, avec des valeurs de 0.23 et -0.18. Nous pouvons voir que ces deux là sont de signe opposé. Ceci est normal. En effet, comme nous l'avons expliqué, les joueur·euse·s se voient attribuer en début de partie un côté, bleu ou rouge. Nous savons que les joueur·euse·s performant mieux du côté bleu. Ainsi, il n'y a donc rien de surprenant à ce que ces deux valeurs soient de signe opposé. La quatrième caractéristique est *main_lane*. Cependant, vu qu'elle a une valeur de 0.009, nous n'allons pas nous attarder dessus, ni sur les suivants ayant une magnitude encore plus faible. Au vu des faibles valeurs des autres caractéristiques, nous nous sommes demandés quels seraient les résultats si on ne faisait tourner le modèle que sur les caractéristiques *win_on_champ*, *winrate_blue* et *winrate_red*.

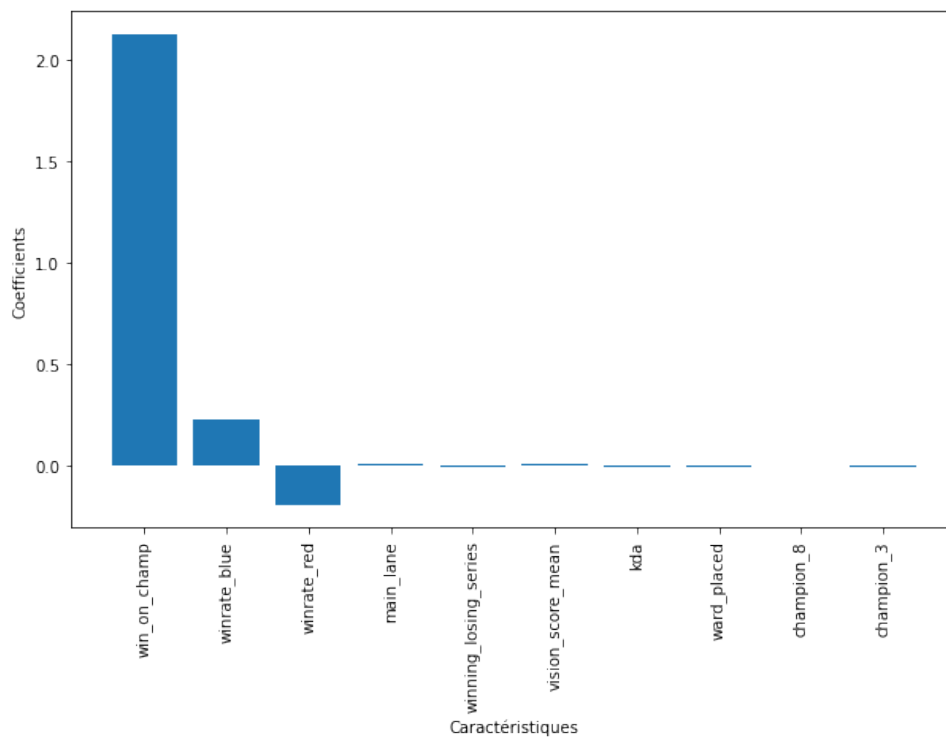


Figure 7.1: Poids attribués à chaque caractéristique, pour la division Argent, sur le LinearSVC

Nous avons synthétisé les précisions obtenues dans le tableau [7.2]. Nous pouvons y observer que même si la précision sur la validation croisée et le score AUC c'est amélioré, ce n'est pas le cas pour les résultats sur l'ensemble de test et le F1-Score. En observant les valeurs des coefficients de ce nouveau modèle, nous obtenons pour *win_on_champ*, *winrate_blue* et *winrate_red*, respectivement 2.00, 0.10, -0.04. Ainsi, ne garder quels fait décroître l'importance qu'on ces caractéristiques sur le modèle.

Nous pouvons en conclure que même si la valeurs des coefficients des autres caractéristiques sont faibles, elles n'en demeurent pas moins utiles.

Attardons nous maintenant sur la division Émeraude. L'équation de l'hyperplan est donnée par $2.01x - 0.99 = 0$. Ainsi, nous pouvons voir que la valeur du coefficient de *win_on_champ* est inférieur sur la division Émeraude par rapport à la division Argent. Pour la division

	VC	ET	F1-Score	AUC
Précision	67.20%	68.43%	68.04%	77.64%

Figure 7.2: Scores de précision pour le LinearSVC sur la division Argent

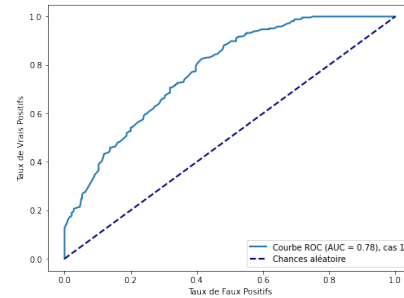


Figure 7.3: Courbes ROC pour le LinearSVC sur la division Argent

Maître, l'équation de l'hyperplan est donnée par $5.06x - 2.62 = 0$. Nous pouvons donc voir que même si les résultats pour les divisions Émeraude et Maître sont meilleur uniquement avec la caractéristique *win_on_champ*, ce n'est pas pour autant que leur hyperplans sont les mêmes. En effet, au vu de son coefficient, la caractéristique *win_on_champ* est plus de deux fois plus importante pour les Maître que pour les Émeraude.

Regardons maintenant les divisions Or et Platine, qui performant mieux sur la régression logistique et comparons-les. Commençons par regarder l'importance des coefficients pour chaque division. Nous pouvons observer[7.6] pour commencer que la caractéristique *win_on_champ* à des valeurs de coefficients similaires pour ces deux divisions. En effet, elle vaut 5.88 pour la division Or et 5.81 pour la division Platine. Pour ces deux divisions, la seconde caractéristique avec le plus de poids est *winrate_red*. Elle est cependant plus de deux fois et demie plus importante pour la division Or que pour la division Argent. Ainsi, elle vaut -0.33 en Or et -0.12 en Platine. Dès la troisième caractéristique, l'ordre d'importance change. Ainsi, pour la division Or, dans l'ordre les divisions importantes sont *ward_kill*, *vision_score_mean* et *main_lane*, avec des coefficients de valeurs respective de 0.13, 0.04 et -0.04. Pour la division Platine, nous avons par contre par ordre d'importance décroissante, *lane_played*, *winning_losing_series* et *main_lane*, avec des coefficients de respectivement -0.06, -0.03 et 0.03. Ainsi, nous pouvons observer que même si ces coefficients sont de faibles valeurs, ils sont du même ordre de grandeur. De plus, ce ne sont pas les mêmes caractéristiques qui sont mise en deuxième et troisième position. Ceci tend à montrer que pour les joueur·euse·s Or, ce qui compte le plus c'est d'abord la vision, par le fait de supprimer les balises de visions ainsi que le score de vision de manière générale. Par contre, pour la division Platine, il semble que la voie sur la quel le·la joueur·euse se trouve et sa suite de série de victoire et défaite soient plus importantes. Il est probable que la caractéristique *lane_played* soient si haute à cause du déséquilibre entre les rôles. En effet, 28% des parties sont jouées sur la voie du milieu, 23% sur la voie du haut, 19% sur la voie du support, 15% sur la voie du bas et 14% dans la jungle. La division Or quant à elle contient 34% de parties jouées sur la voie du bas, mais les autres divisions sont équitablement représentée à hauteur de plus ou moins 16% chacune.

Regardons maintenant si les performances de la régression linéaire sont meilleures lorsque nous prenons uniquement les 5 caractéristiques ayant le plus d'impact sur ces deux divisions. Lorsque nous regardons sur le tableau[7.7] recensant les scores important pour ces deux divisions, nous pouvons observer, malgré une hausse de précision sur la validation croisée et le score AUC, une perte de précision sur l'ensemble de test et le F1-Score pour les deux divisions. Ceci semble indiquer que le modèle a plus tendance à ce sur-ajuster s'il est dépossédé de ces autres caractéristiques.

Observons maintenant une par une les divisions qui ont une précision inférieure à la précision

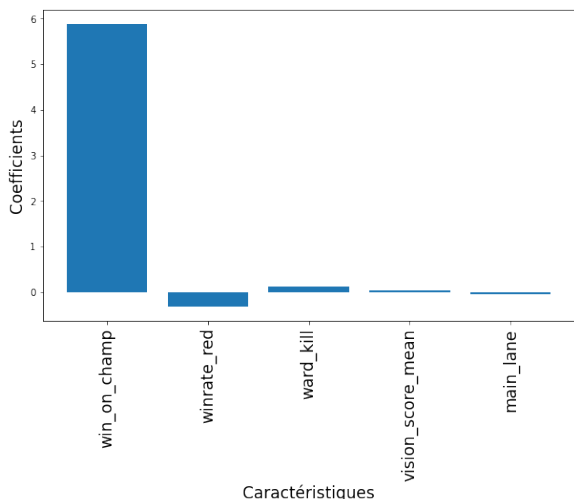


Figure 7.4: division Or

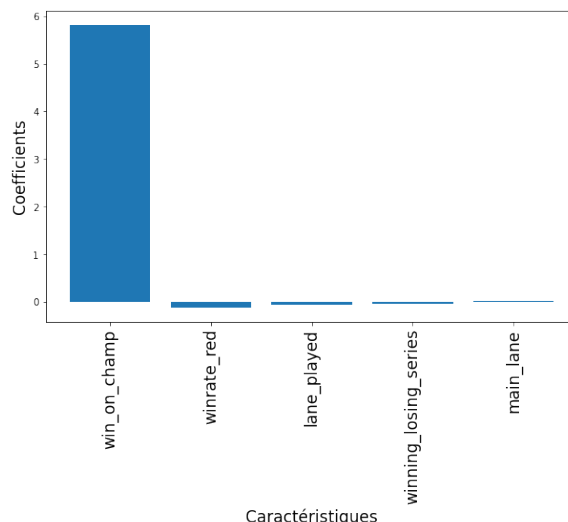


Figure 7.5: division Platine

Figure 7.6: Amplitude des coefficients la régression logistique

	VC	ET	F1-Score	AUC
Or	67.31%	68.77%	69.16%	78.63%
Platine	67.05%	67.86%	69.53%	75.85%

Figure 7.7: Scores de précision pour la régression logistique sur la division Or et Platine

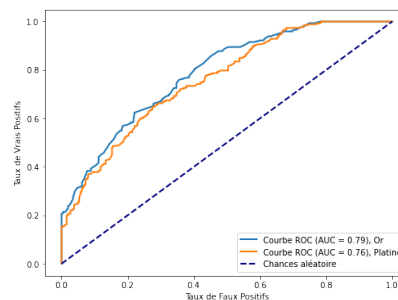


Figure 7.8: Courbes ROC pour la régression logistique sur la division Or et Platine

sur l'ensemble des données. Rappelons que dans ce cas, nous avons supposé qu'il valait mieux se servir des modèles sur l'ensemble des données. Afin de confirmer cette hypothèse, nous avons vérifié pour chaque division concernée quelle était la précision associée. Ces valeurs de précisions sont présentées dans le tableau [7.2] ci-dessous.

	Fer	Bronze	Diamant
Précision	63.63%	66.25%	73.72%

Tableau 7.2: Précision dans l'ensemble des données

Nous pouvons observer que cette supposition n'est ni fausse ni vraie. Elle varie selon les divisions. Ainsi, pour la division Diamant, il y a une nette amélioration, en se servant des prédictions faites par la régression logistique sur l'ensemble des données, on se rend compte que la précision réelle pour la division Diamant est en fait de 73.72%, soit plus de 7% meilleure. La précision sur la division Bronze est également légèrement inférieure, mais pas de manière significative. Ce n'est pas le cas pour la division Fer qui passe à 63.63%. Ainsi, en réalité, les meilleurs performances pour la division Fer sont alors lors sur le cas 1, sur la régression logistique par division ou elle atteint 65.71%. Quoiqu'il en soit, le coefficient associé à la caractéristique *win_on_champ* est de 5.89, ce qui est du même ordre de grandeur que pour les divisions ayant

de meilleure performance individuelle.

Nous venons de voir que parfois, la précision d'une division dans l'ensemble des données est bien meilleure que la valeur moyenne sur l'ensemble des données. Nous allons donc nous intéresser à toutes les divisions dans la régression logistique et dans le LinearSVC, qui sont nos deux meilleurs modèles. Nous regardons ceci pour le cas 3 étant donné qu'il s'agit là du meilleur cas pour ces deux modèles. Observons les résultats sur le tableau [7.3.]

	LinearSVC	Regression logistique
Fer	60.78%	63.63%
Bronze	58.50%	66.25%
Argent	67.16%	65.10%
Or	63.50%	70.11%
Platine	58.65%	64.23%
Émeraude	60.88%	66.84%
Diamant	68.42%	73.71%
Maître	76.47%	63.63%

Tableau 7.3: Meilleure précision pour chaque division

Nous pouvons ainsi mieux nous représenter les disparités présentes dans la prédiction de chaque division dans l'ensemble des données. Ceci nous permet de finalement obtenir le tableau[7.4] suivant. Il synthétise la précision maximum atteinte pour chaque division.

	Fer	Bronze	Argent	Or	Platine	Émeraude	Diamant	Maître
Précision	63.63%	65.71%	65.90%	69.49%	70.57%	68.62%	68.84%	76.47%

Tableau 7.4: Précision maximum atteinte pour chaque division

Ainsi, il finalement, il vaut mieux pour la division Fer se servir du cas 1 sur la régression logistique. Pour la division Bronze c'est aussi la régression qui performe le mieux, mais sur le cas 2. Pour finir, la division Maître a de meilleurs résultats sur le cas 3 pour le LinearSVC.

Chapitre 8

Conclusion et perspectives d'ouverture

Nous allons maintenant synthétiser tout ce que nous avons pu découvrir dans ce mémoire. Après une explication détaillée du jeu sujet de ce mémoire, League of Legends, nous avons étudié l'effet des modèles de régression logistique, DecisionTree, RandomForest, K-Nearest Neighbour et LinearSVC sur notre problème. Nous avons pu observer que les modèles linéaires performaient mieux que les modèles non linéaires. Cependant, nous avons constaté que, pour chaque modèle et en toutes circonstances, la caractéristique *win_on_champ* est essentielle et extrêmement prédictive. Ainsi, dans certains cas, les résultats des modèles étaient meilleurs lorsque seule cette dernière était présente.

Nous pouvons donc répondre à la question qui nous a brûlé les lèvres tout du long: Oui, il est possible de prédire le résultat d'une partie pour des joueur·euse·s de tous niveaux, avec une précision tournant entre 65 et 76%, comme recensé dans le tableau [7.4]/ . Chaque division de joueur·euse·s a ces propres spécificité et doit se concentrer sur certains aspects particulier afin de gagner. Cependant, ce qui est commun à tous·tes, c'est l'importance du taux de victoire du·de la champion·ne joué·e. Ainsi, le premier conseil à donner à tout le monde, c'est de privilégier les champion·ne·s sur lesquels iels sont les meilleurs. En général, il s'agit des champion·nes les mieux connu·e·s. Nous avons également pu observer que la caractéristique *winrate_red* est la deuxième plus importante dans toutes les analyses impliquant plus que juste la caractéristique *win_on_champ*.

Ainsi, nous avons tout du long performés nos analyses sur les divisions. Il existe cependant une autre approche qu'il faudrait explorer afin d'obtenir probablement des résultats plus poussés. Il s'agit du poste joué. En effet, comme nous l'avons expliqué, chaque poste, chaque voie à ces spécificités, impliquant automatiquement des caractéristiques importantes différentes dues aux différentes manières de jouer. Afin de faire un premier pas dans cette analyse, nous avons rassemblé pour la régression logistique et le LinearSVC les précisions pour chaque poste sur l'ensemble des données et l'ensemble des caractéristiques[8.1].

	Toplaner	Support	Midlaner	ADC	Jungler
Régression logistique	65.34%	64.56%	65.06%	69.89%	69.85%
LinearSVC	61.07%	59.73%	62.95%	62.88%	66.18%

Tableau 8.1: Précision par rôle pour le cas 1 sur l'ensemble des données

Nous pouvons observer pour la régression logistique que les rôles d'ADC et de Jungler ont une meilleure performance que la moyenne des autres postes. Pour le LinearSVC, nous pouvons observer une grosse différence entre la précision du rôle de support et la précision moyenne. Ces deux éléments tendent donc à montrer qu'il serait intéressant de se pencher sur la décomposition

par rôle et la combiner avec la décomposition par division.

Il existe cependant d'autres pistes d'exploration possible. Premièrement, nous avons travaillé uniquement sur la file classée en solo ou duo. Cependant, il existe d'autres files, notamment la file classée flexible. Nous pourrions envisager d'intégrer des parties de ces autres files afin de voir si cela influence sur les résultats.

Afin d'étendre l'utilité de ce mémoire, nous pourrions envisager de prendre en comptes d'autres caractéristiques tels que les objets achetés, les dragons tués ou la durée actuelle de la partie. Ainsi, nous pourrions avoir une prédiction non pas en début de partie, mais en temps réelle, de telle manière que les joueur·euse·s puissent voir en temps réelle comment leurs actions influence sur leurs chance de victoire.

Il serait sûrement intéressant de comparer nos résultats à ceux que nous obtiendrons en faisant ce même travail sur les autres régions. Peut être ceci pourrait-il aider à comprendre pourquoi la Corée du sud domine la scène compétitive et ainsi donner une meilleure chance à nos équipe nationales.

Nous avons également pu voir que nous n'avions pas trouvé de partie où les joueur·euse·s sont dans des situations où leur champion·ne·s principale est bannies. Nous pourrions chercher spécifiquement ces parties afin de savoir ce qu'il en est réellement de l'influence de cette caractéristique sur les performances des joueur·euse·s.

Depuis le patch 14.9, Riot a mis en place Vanguard. Il s'agit d'un logiciel anti-triche. Depuis sa mise en place, les taux de victoires de certain·ne·s champion·ne·s sont en chute libre. Ainsi, il pourrait être intéressant de refaire les analyses par champion avec Vanguard.

Ainsi, nous avons pu voir que oui, on peut prédire les résultats d'une partie de League of Legends, mais il existe sans aucun doutes différentes façons d'améliorer ces prédictions.

Bibliographie

- [1] Oracle. Qu'est-ce que le machine learning ? <https://www.oracle.com/fr/artificial-intelligence/machine-learning/what-is-machine-learning/>. [En ligne; consulté le 05/04/2024].
- [2] The scikit-learn developers. Scikit-learn - guide - Modèles linéaire. https://runebook.dev/fr/docs/scikit_learn/modules/linear_model. [En ligne; consulté le 15/04/2024].
- [3] Fabien Moutarde. Arbres de Décision et Forêt Aléatoire. https://people.minesparis.psl.eu/fabien.moutarde/ES_MachineLearning/Slides/coursFM_AD-RF.pdf. [En ligne; consulté le 23/05/2024].
- [4] scikit-learn developers (BSD License). 1.10. Decision Trees. <https://scikit-learn.org/stable/modules/tree.html#tree>. [En ligne; consulté le 16/04/2024].
- [5] Wikipédia. Bootstrap aggregating. https://fr.wikipedia.org/wiki/Bootstrap_aggregating. [En ligne; consulté le 25/05/2024].
- [6] Wikipédia. Arbres de Décision et Forêt Aléatoire. https://fr.wikipedia.org/wiki/M%C3%A9thode_des_k_plus_proches_voisins. [En ligne; consulté le 26/05/2024].
- [7] Wikipédia. Minkowski distance. https://en.wikipedia.org/wiki/Minkowski_distance. [En ligne; consulté le 30/05/2024].
- [8] Wikipédia. Arbres de Décision et Forêt Aléatoire. https://fr.wikipedia.org/wiki/Machine_%C3%A0_vecteurs_de_support. [En ligne; consulté le 26/05/2024].
- [9] Wikipédia. Hinge loss. https://en.wikipedia.org/wiki/Hinge_loss#/media/File:Hinge_loss_vs_zero_one_loss.svg. [En ligne; consulté le 26/05/2024].
- [10] sklearn. sklearn.metrics.accuracy_score. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html. [En ligne; consulté le 1/05/2024].
- [11] Wikipédia. Corrélation (statistiques). [https://fr.wikipedia.org/wiki/Corr%C3%A9lation_\(statistiques\)](https://fr.wikipedia.org/wiki/Corr%C3%A9lation_(statistiques)). [En ligne; consulté le 29/04/2024].
- [12] towardsai.net. Hinge loss. <https://towardsai.net/p/data-science/how-to-evaluate-you-model-using-the-confusion-matrix>. [En ligne; consulté le 26/05/2024].
- [13] ICHI.PRO. Hinge loss. <https://ichi.pro/fr/sur-les-courbes-roc-et-precision-recall-213574150750732>. [En ligne; consulté le 26/05/2024].

- [14] La langue française. E-sport. <https://www.lalanguefrancaise.com/dictionnaire/definition/e-sport>. [En ligne; consulté le 30/04/2024].
- [15] StuffGaming. Guide complet de la vision et des balises sur League of Legends. <https://stuffgaming.fr/guide-complet-de-la-vision-et-du-warding-sur-league-of-legends/>. [En ligne; consulté le 29/05/2024].
- [16] Tiffany D. Do, Seong Ioi Wang, Dylan S. Yu, Matthew G. McMillian, Ryan P. McMahan. Using Machine Learning to Predict Game Outcomes Based on Player-Champion Experience in League of Legends. <https://arxiv.org/pdf/2108.02799>. [En ligne; consulté le 26/05/2024].
- [17] Riot Games. Qui nous sommes. <https://www.riotgames.com/fr/Qui-nous-sommes>. [En ligne; consulté le 09/01/2024].
- [18] Carver Fisher. LoL Worlds 2023 shatters records as the most viewed esports event ever. <https://www.dexerto.com/league-of-legends/lol-worlds-2023-shatters-records-as-the-most-viewed-esports-event-ever-2390533/>. [En ligne; consulté le 09/01/2024].
- [19] Active player io. League of Legends Live Player Count and Statistics. <https://activeplayer.io/league-of-legends/>. [En ligne; consulté le 10/01/2024, 30/01/2024].
- [20] Riot Sakaar. Note de patch 14.7. <https://www.leagueoflegends.com/fr-fr/news/game-updates/patch-14-7-notes/>. [En ligne; consulté le 1/05/2024].
- [21] Dot Esport. One side has a shockingly high win rate at LoL Worlds 2023. <https://dotesports.com/league-of-legends/news/one-side-has-a-shockingly-high-win-rate-at-lol-worlds-2023>. [En ligne; consulté le 10/05/2024].

Annexe

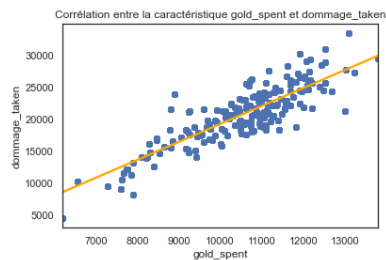


Figure 8.1: Corrélation = 0.8392

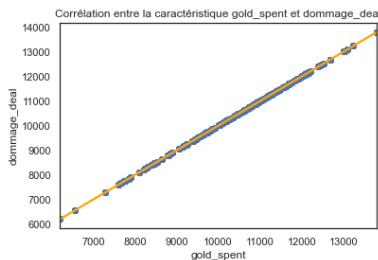


Figure 8.2: Corrélation = 1

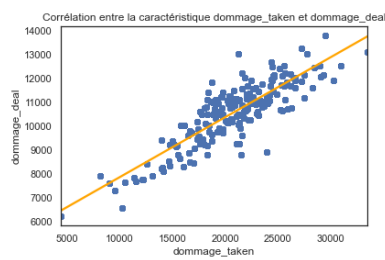


Figure 8.3: Corrélation = 0.8392

Figure 8.4: Corrélation avec la caractéristique *gold_spent*

	Caractéristique 1	Caractéristique 2	Caractéristique 3
Fer	<i>win_on_champ</i> : 5.0436	<i>ward_placed</i> : 0.1217	<i>lane_played</i> : 0.0288
Bronze	<i>win_on_champ</i> : 6.1185	<i>winrate_red</i> : -0.5514	<i>main_lane</i> : -0.1245
Argent	<i>win_on_champ</i> : 6.0926	<i>vision_score_mean</i> : 0.1010	<i>ward_kill</i> : 0.0627
Or	<i>win_on_champ</i> : 5.888	<i>winrate_red</i> : -0.3238	<i>ward_kill</i> : 0.1275
Platine	<i>win_on_champ</i> : 5.8147	<i>winrate_red</i> : -0.1213	<i>lane_played</i> : -0.0591
Émeraude	<i>win_on_champ</i> : 6.089	<i>winrate_red</i> : -0.6486	<i>kda</i> : -0.1317
Diamant	<i>win_on_champ</i> : 5.6317	<i>winrate_red</i> : -0.1048	<i>kda</i> : -0.0466
Maître	<i>win_on_champ</i> : 5.6231	<i>vision_score_mean</i> : 0.3849	<i>main_lane</i> : -0.0639

Tableau 8.2: Importances des caractéristiques par division dans la régression logistique pour le cas 1

Validation	Score 1	Score 2	Score 3	Score 4	Score 5	Score moyen
1	67.91%2	67.40%	67.27%	66.40%	66.11%	67.02%
2	67.33%	66.24%	67.03%	66.11%	68.19%	66.98%
3	67.71%	65.85%	67.95	65.91%	66.87%	66.86%
4	66.75%	66.94%	68.36%	67.61%	67.82%	67.5%
5	65.46%	65.82%	68.24%	67.90%	67.7%	67.02%

Tableau 8.3: Validation croisée pour la régression logistique pour le cas 1

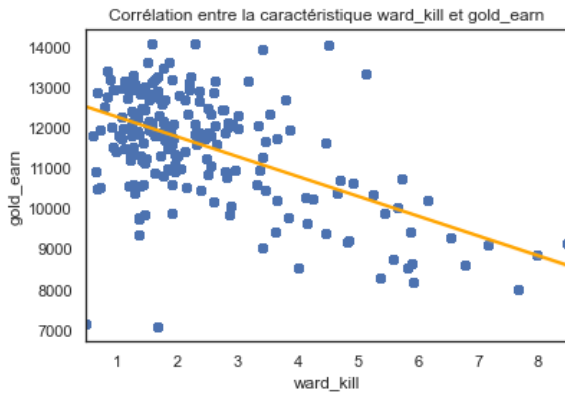


Figure 8.5:
Corrélation = -0.6274

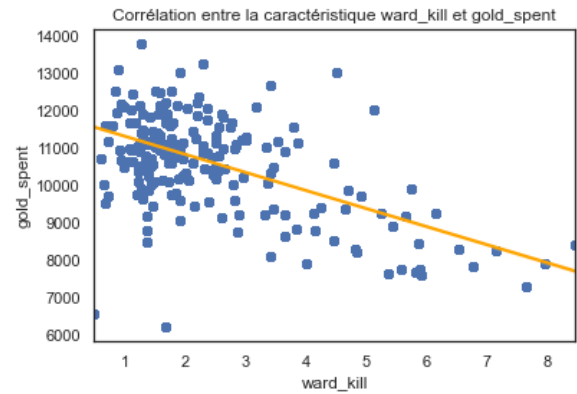


Figure 8.6:
Corrélation = -0.6267

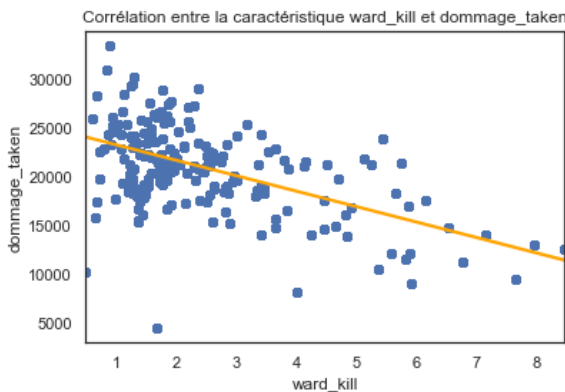


Figure 8.7:
Corrélation = -0.6087

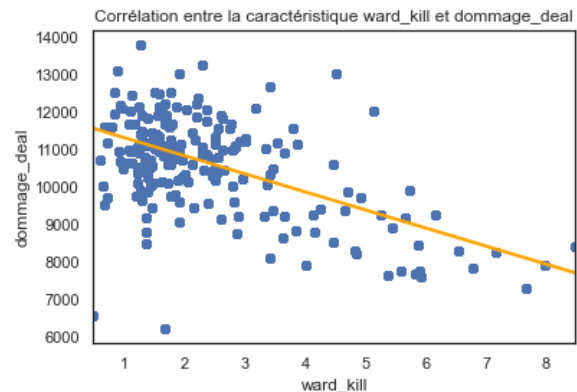


Figure 8.8:
Corrélation = -0.6267

Figure 8.9: Corrélation *ward_kill*

Analyse des caractéristiques

vision_score_mean

Il nous semblait important d'inclure dans nos caractéristiques le score de vision. En effet, il s'agit d'un paramètre important dans une partie. Avoir ou non la vision sur un endroit stratégique peut se révéler l'origine d'une défaite ou d'une victoire.

Afin d'analyser cette importance, nous avons fait deux graphiques. Le premier [8.93] montre le pourcentage de victoire par division lorsque le score moyen du joueur est supérieur à la moyenne de vision de l'entièreté des données, qui est de 0.2916 et le deuxième par rapport à la médiane [8.94], qui vaut 0.2. Comme nous le pensions de prime abord, le score de vision a bien de l'importance sur le fait de gagner une partie. Cependant, lorsque nous regardons les graphes, nous pouvons voir que ce n'est pas exactement de la façon dont nous le pensions. En effet, si nous regardons le graphe où nous avons mis le seuil à la moyenne [8.93], nous pouvons voir qu'un moins bon score de vision permet de gagner plus facilement.

Par contre, lorsque nous regardons par rapport à la médiane [8.94], nous pouvons voir que ce schéma est moins marqué. En effet, on peut observer que le score de vision influence significativement différemment chaque division. Ainsi, pour les divisions Argent, Platine, Diamant et Maître, nous obtenons le résultat attendu, à savoir un meilleur taux de victoire avec

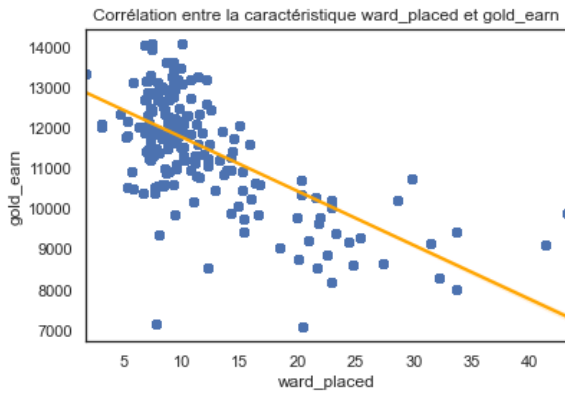


Figure 8.10:
Corrélation = -0.7002

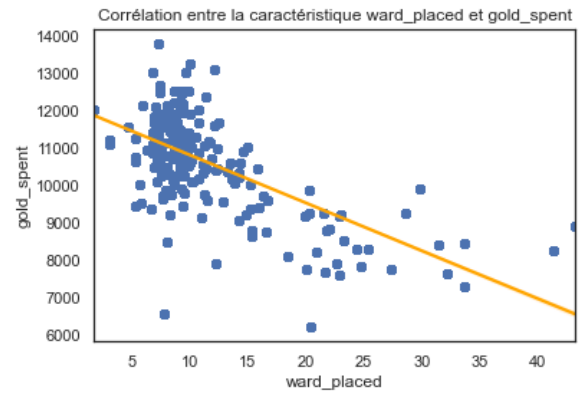


Figure 8.11:
Corrélation = -0.6921

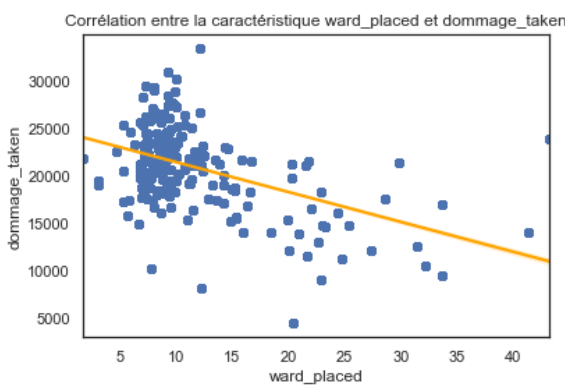


Figure 8.12:
Corrélation = -0.5193

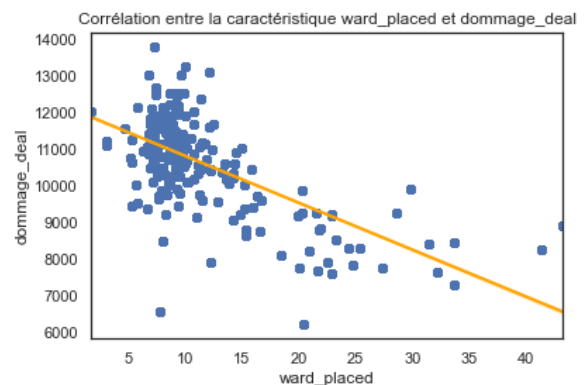


Figure 8.13:
Corrélation = -0.6926

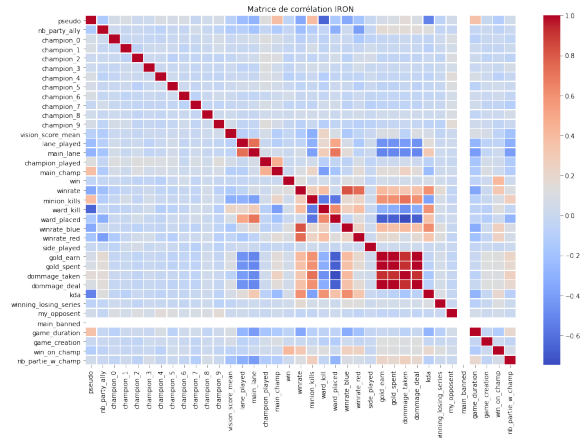
Figure 8.14: Corrélation *ward_placed*

un bon score de vision, mais toujours pas pour les autres catégories.

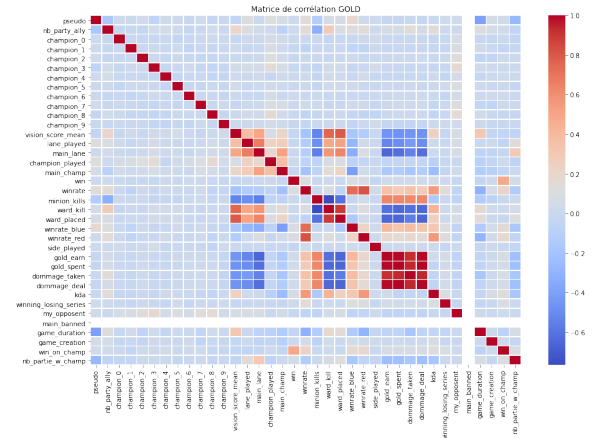
Si nous faisons maintenant cette analyse en comparant non plus par rapport à la moyenne et la médiane totale, mais par division, nous obtenons les graphes ci dessous. Lorsque nous regardons par rapport à la moyenne[8.95], les constats précédents sont toujours valides mais moins marqués. Ainsi, on peut même observer qu'en Argent, il vaut légèrement mieux avoir un bon score de vision pour gagner. A l'inverse, les observations faites pour la médiane ne s'appliquent plus. En effet, lorsque nous regardons le graphique [8.96], nous pouvons voir que même si c'est moins marqué que pour la moyenne, il vaut quand même mieux ne pas avoir un trop bon score de vision pour gagner une partie.

ward_kill

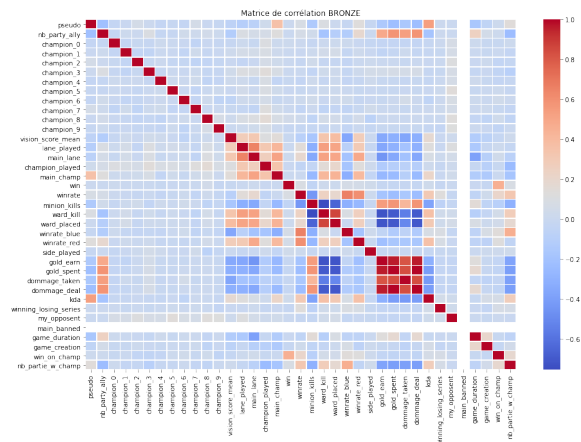
Regardons maintenant plus spécifiquement la caractéristique *ward_kill* qui est une composante de *vision_score_mean*. Comme pour cette dernière, nous avons regardé le taux de victoire par rapport à la moyenne [8.97], qui est de 2.537 et par rapport à la médiane [8.98], qui vaut 1.95. Concernant la moyenne, nous pouvons voir que de la même façon que pour *vision_score_mean*, en général, il vaut mieux ne pas trop retirer de balise de vision de la carte afin de gagner la partie. Ceci peut sembler aberrant mais peut en partie être expliqué par le fait que les joueur·euse·s sont souvent dissipé·e·s par ces balises. En effet, il arrive très souvent qu'en se concentrant sur



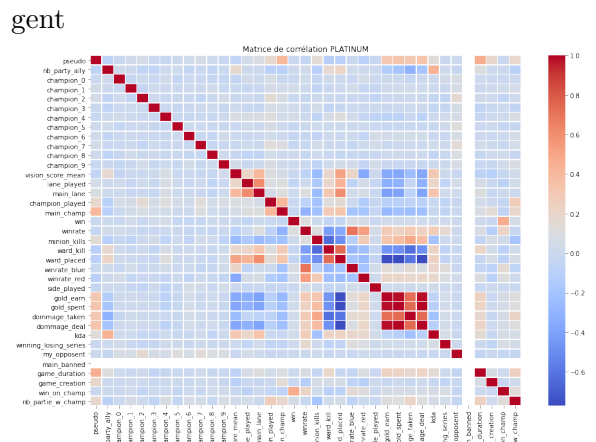
(a) Matrice de corrélation pour la division Fer



(c) Matrice de corrélation pour la division Argent



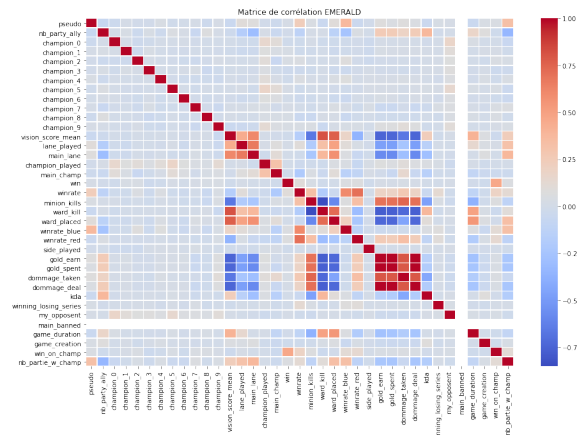
(b) Matrice de corrélation pour la division Bronze



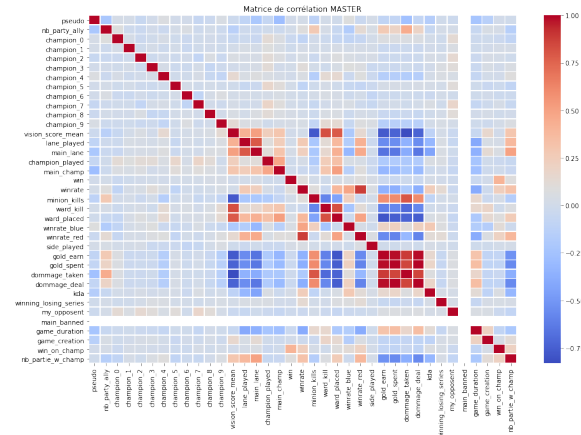
(d) Matrice de corrélation pour la division Or

la balise, des joueur·euse·s loupent des actions à coté d'eux ou même meurent à cause du temps perdu durant la destruction de la balise. On peut d'ailleurs observer que la différence entre les pourcentages de victoires en ayant tué plus de balises de vision que la moyenne a tendance à diminuer quand le niveau des joueur·euse·s augmente. Ce qui est sûrement dû au fait que ces dernier·e·s sont plus concentré·e·s et se font moins avoir par ce genre d'erreur. Quand on regarde par rapport à la médiane [8.98], on peut voir que les affirmations que nous venons d'avancer se vérifient. En effet, plus les joueur·euse·s sont bon·ne·s, plus il vaut mieux pour ces dernier·e·s éliminer les balises qui dévoilent leur position.

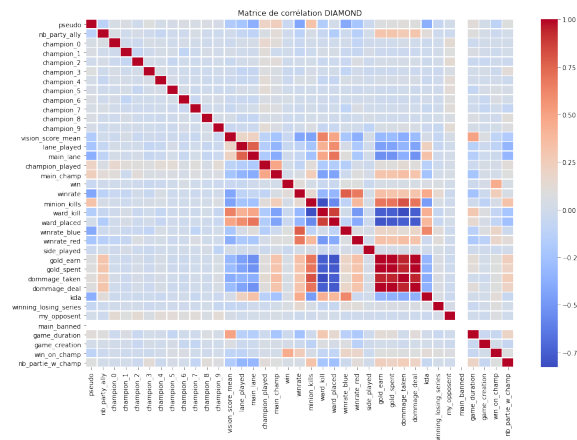
Si on regarde maintenant en comparant non plus à la moyenne total, mais à la moyenne par division [8.99] et la médiane par division[8.100], nous observons des résultats différents. Premièrement, lorsque nous regardons par division, en moyenne il vaut toujours mieux ne pas retirer la vision de la carte, sauf pour les Fer, qui pour une raison à ce jour inexpliquée ont drastiquement intérêt à retirer toutes les balises de visions qu'ils voient. A contrario, lorsque nous comparons à la médiane les résultats, nous obtenons enfin le résultat attendu. Ainsi, il vaut mieux pour chacun de retirer les balises de visions qu'il voit afin de rester discret et de garder un certains avantage.



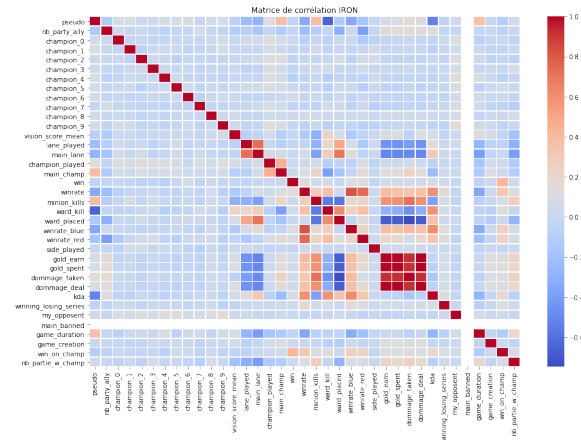
(e) Matrice de corrélation pour la division Platine



(g) Matrice de corrélation pour la division Diamant



(f) Matrice de corrélation pour la division Emeraude



(h) Matrice de corrélation pour la division Maître

Figure 8.15: Ensemble des matrices des corrélations

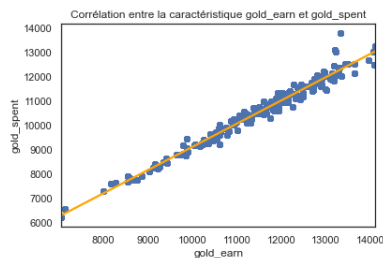


Figure 8.16:
Corrélation = 0.9834

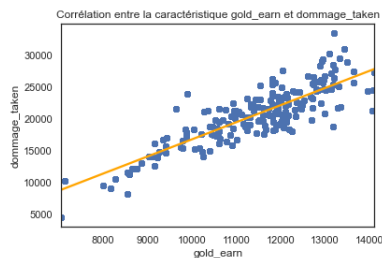


Figure 8.17:
Corrélation = 0.8399

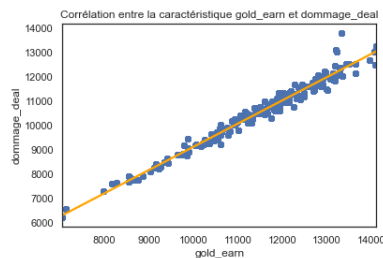


Figure 8.18:
Corrélation = 0.9834

Figure 8.19: Corrélation avec la caractéristique *gold_earn*

Validation	Score 1	Score 2	Score 3	Score 4	Score 5	Score moyen
1	67.37%	67.32%	66.44%	68.69%	66.23%	67.21%
2	66.92%	67.78%	66.45%	67.40%	67.32%	67.17%
3	67.42 %	65.32%	66.53%	68.40%	67.15%	66.96%
4	66.67%	66.82%	66.53%	67.40%	67.74%	67.03%
5	66.75%	66.65%	66.74%	67.61%	67.82%	67.11%

Tableau 8.4: Validation croisée sans les caractéristiques corrélées entre-elles à plus de 60% pour la régression logistique pour le cas 2

ward_placed

La caractéristique *ward_placed* est également une composante de *vision_score_mean*. Suivons à nouveau la même procédure afin de découvrir si oui ou non, les joueur·euse·s ont intérêt à placer des balises de vision sur la carte afin d’espionner leurs adversaires.

Comme précédemment, lorsque nous comparons le taux de victoire en prenant un seuil de 11.50, correspondant à la moyenne[8.101], nous pouvons à nouveau observer qu’il semblerait qu’il vaille mieux ne pas placer beaucoup de balise de vision afin de remporter sa partie. Mais lorsque nous regardons par rapport à la médiane[8.102] qui est de 9.21, nous obtenons quelque chose de tout à fait différent. En effet, nous pouvons voir qu’à bas niveau (Fer, Bronze, Argent), il vaut mieux placer beaucoup de balise de vision pour gagner, alors que cela semble être moins le cas à haut niveau, notamment en Maître, Diamant et Émeraude. Nous émettons l’hypothèse suivante: Cette différence s’expliquerait par le fait qu’à haut niveau, même sans les voir, les joueur·euse·s ont une bonne idée d’où se trouve leurs adversaires et ont donc moins besoin de savoir exactement où afin d’agir en fonction.

Faisons maintenant l’analyse sur les moyennes et médianes par division. En termes de moyenne[8.103], rien ne change. Il semblerait qu’il vaille mieux ne pas trop mettre de balises de vision pour gagner. La où par contre, on peut voir une différence significative, c’est pour la médiane[8.104]. En effet, il n’y a lorsqu’on regarde par division, aucun doute: il vaut mieux, même si ce n’est pas forcément très marqué, placer de la vision sur la carte pour gagner.

lane_played et *main_lane*

Les caractéristiques *lane_played* et *main_lane* semblent être capitales et sont liées entre elles. En effet, les joueur·euse·s ont joué·e·s sur leur voie dans 72.31% du temps en moyenne. On peut voir cette donnée pour chaque division sur le tableau [8.7]. En regardant le tableau, on peut voir que cette affirmation est de plus en plus vraie en montant de niveau. Nous pensons que

Validation	Score 1	Score 2	Score 3	Score 4	Score 5	Score moyen
1	66.96%	67.51%	66.32%	67.07%	68.24%	67.22%
2	66.58%	66.15%	68.19%	68.45%	66.78%	67.23%
3	67.46%	67.78%	66.65%	67.65%	66.57%	67.22%
4	67.92%	66.28%	67.32%	67.654424%	66.9449	67.22%
5	67.92%	67.69%	65.61%	68.28%	66.61%	67.22%

Tableau 8.5: Validation croisée avec uniquement la caractéristique *win_on_champ* pour la régression logistique pour le cas 3

Validation	Score 1	Score 2	Score 3	Score 4	Score 5	Score moyen
1	56.82%	54.00%	54.22%	53.13%	53.42%	54.32%
2	54.23%	54.51%	56.14%	53.92%	53.42%	54.44%
3	54.65%	53.42%	53.17%	54.79%	54.67%	54.14%
4	52.69%	54.59%	54.13%	55.34%	55.71%	54.49%
5	53.07%	55.43%	55.39%	54.79%	53.92%	54.52%

Tableau 8.6: Validation croisée sans la caractéristique *win_on_champ* pour la régression logistique pour le cas 4

ceci est dû au fait que les joueur·euse·s ont une prédilection de base pour une voie et qu’afin de monter, il faut ce concentrer sur cette dernière.

	Fer	Bronze	Argent	Or	Platine	Émeraude	Diamant	Maître
Pourcentage	61.64%	67.8%	66.22%	67.61%	69.38%	70.28%	71.08%	72.31%

Tableau 8.7: Pourcentage de parties jouées sur sa voie principale, en fonction du niveau

Lorsque l’on regarde plus attentivement les pourcentages de victoire en fonction du fait d’être oui ou non sur sa voie principale [8.105], on peut voir une tendance significative se dégager. Les joueur·euse·s sont meilleur·e·s lorsqu’iels se concentrent sur leur voir principal. La seule division échappant à cette constatation est le Fer. Nous pensons que cela est dû au fait que cette catégorie de joueur·euse contient ce que le jeu fait de pire en termes de capacité de jeu. Ainsi, ces derniers ne sachant pas vraiment jouer testent un peu tout et n’importe quoi, avec des résultats mitigés.

main_champ

Il nous venait instinctivement que le-la champion·ne le-la plus souvent incarné·e par un·e joueur·euse devait avoir de l’influence sur la victoire de ce-tte dernier·e. En effet, il semble légitime d’être meilleur·e sur un·e champion·ne maintes fois joué·e. Nous pouvons voir sur le graphique [8.106], que cette supposition est vraie. En effet, quel que soit le niveau de-de la joueur·euse, le pourcentage de parties gagnées est supérieur quand on joue sur un·e champion·ne que l’on connaît bien. Cette tendance est donc observable pour tous mais à un degré différent. En effet, quand on regarde la colonne Fer, on peut voir qu’il n’y a que 3.73% de différence sur le pourcentage de parties gagnées selon que oui ou non le-la jouer·euse ait son·sa champion·ne principal. La division pour laquelle cette différence est la plus marquée est Argent avec 11.76% suivit de près par Émeraude avec 10.49% et Maître avec 10.37%.

winrate

Le taux de victoire est bien entendu une variable importante, et comme nous pouvons le voir sur l'image [8.107], le taux de victoire par coté varie selon le niveau.

minion__kills

Le nombre de minions tués est un facteur important pour la victoire du jeu. En effet, chaque élimination de ces petits monstres apporte de l'argent et ainsi des ressources vitales afin d'équiper son·sa champion·ne et devenir une menace pour l'équipe ennemie. Nous pouvons voir sur le graphique[8.108], qu'en effet, même si la différence n'est pas très élevée, en général, les parties gagnées sont celles où les joueur·euse·s ont tués le plus de sbires.

side__played

Pour les joueur·euse·s de LoL, il est un fait bien connu que le coté de la carte sur lequel on joue a une influence sur la victoire. En effet, cette fois encore, lors du championnat des worlds 2023, nous avons pu observer ce phénomène. Lors de cette compétition internationale majeure, nous avons pu voir que le coté bleu avait un taux de victoire de 76.7% [21]. Les raisons de ce déséquilibre ne sont pas le sujet de ce mémoire. Cependant, si l'on regarde le graphique[8.109], on peut observer que ce déséquilibre s'applique à tout le monde, en proportion variable. Ainsi, le coté bleu semble être favorable à tous, sauf aux joueur·euse·s classé·e·s Bronze et Maître. Si nous regardons au total, nous obtenons un taux de victoire de 50.54% pour le coté bleu contre 48.80% pour le coté rouge.

gold__earn* et *gold__spent

Comme dit précédemment, l'argent est une variable capital dans le jeu, étant donné qu'il permet de mieux équiper ces champion·ne·s, d'acheter des balises de visions, en un mot, de prendre l'avantage sur son adversaire. Lorsqu'on analyse les graphiques[??], on peut voir qu'ils sont excessivement ressemblant. Nous verrons plus tard pourquoi précisément, mais il s'agit notamment du fait qu'il existe un coefficient de corrélation de 0.98 entre les deux. La deuxième chose qu'on peut observer, c'est que même s'il existe une différence en termes de po pour une victoire et une défaite, elle est fort faible. Nous nous attendons donc à ce que plus tard, cette caractéristique soit peu utilisées par nos modèles.

dommage__taken* et *dommage__deal

A nouveau, tout comme pour *gold__earn* et *gold__spent*, nous pouvons voir que les graphiques des dégâts par rapport à la victoire et la défaite sont très ressemblant pour les deux caractéristiques [??].

kda

La caractéristique *kda* qui est définie comme la somme d'assistance et d'élimination sur le nombre de mort est également un facteur important. On peut voir sur le graphique [8.114] que pour toutes les divisions, avoir un bon kda est synonyme de victoire.

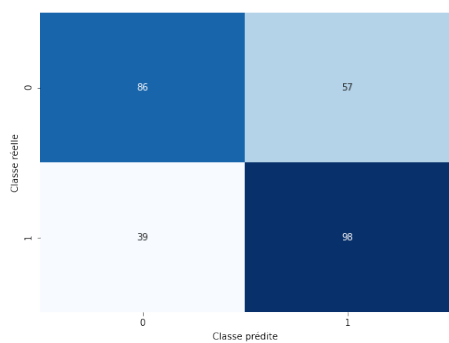


Figure 8.20: Fer

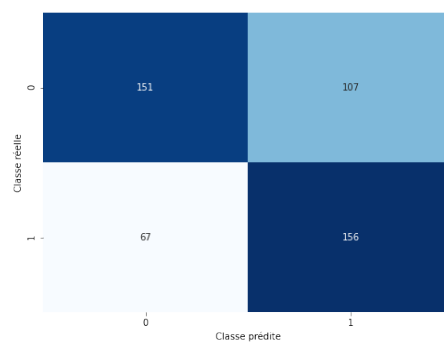


Figure 8.21: Bronze

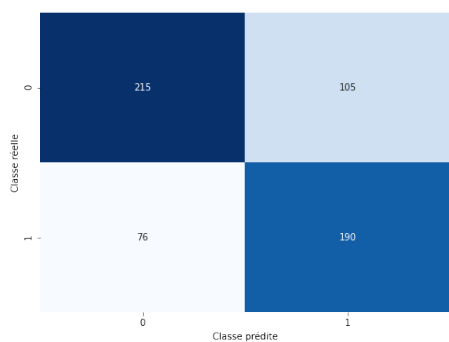


Figure 8.22: Argent

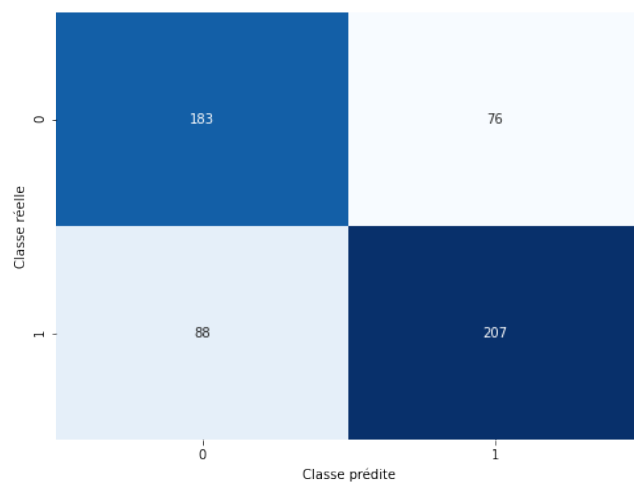


Figure 8.23: Or

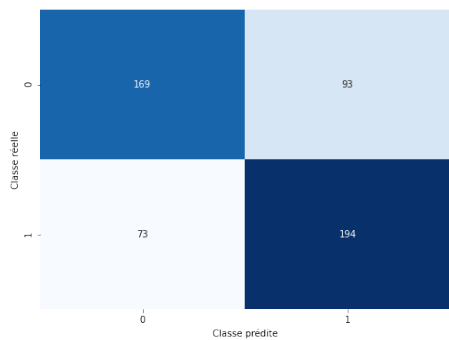


Figure 8.24: Platine

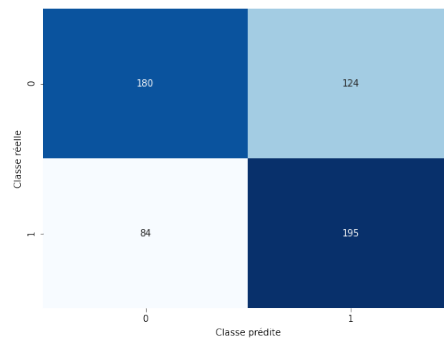


Figure 8.25: Émeraude

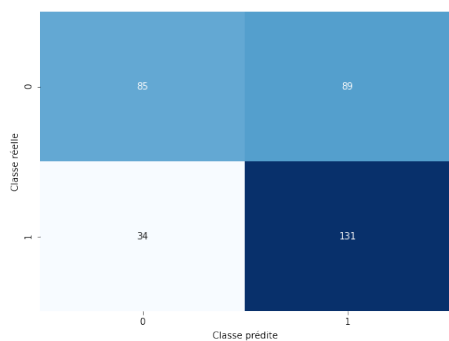


Figure 8.26: Diamant

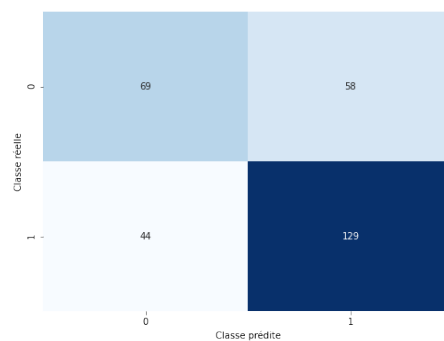


Figure 8.27: Maître

Figure 8.28: Matrices de confusion pour la régression logistique dans le cas 1

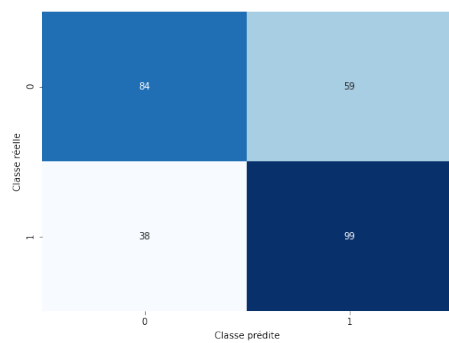


Figure 8.29: Fer

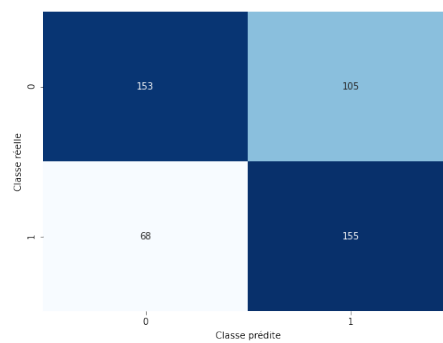


Figure 8.30: Bronze

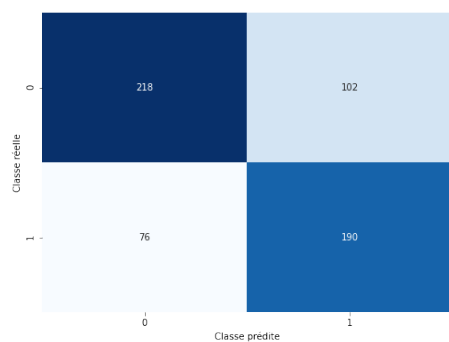


Figure 8.31: Argent

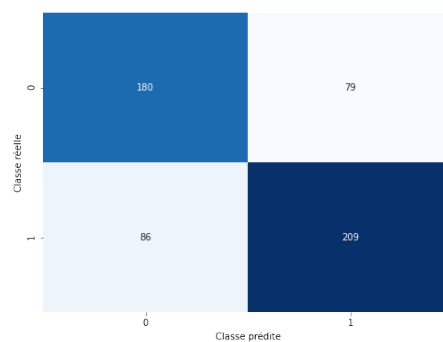


Figure 8.32: Or

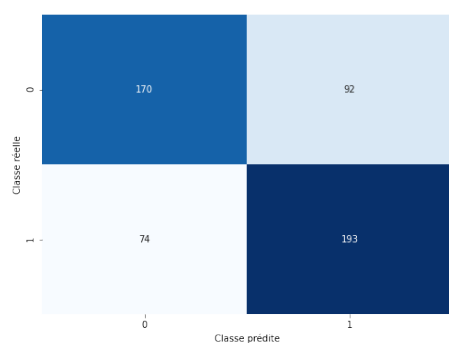


Figure 8.33: Platine

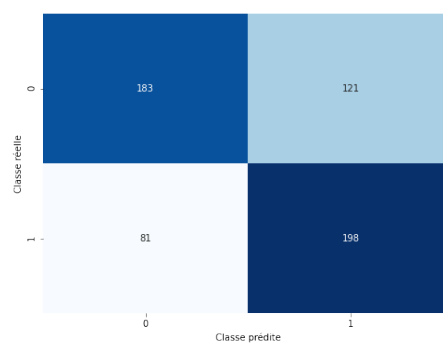


Figure 8.34: Émeraude

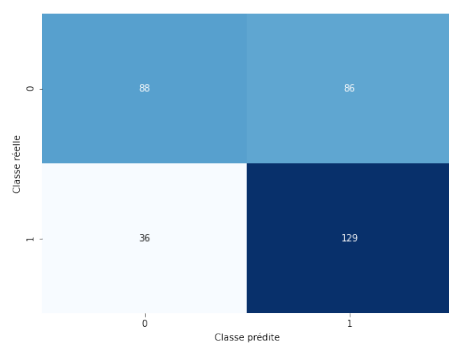


Figure 8.35: Diamant

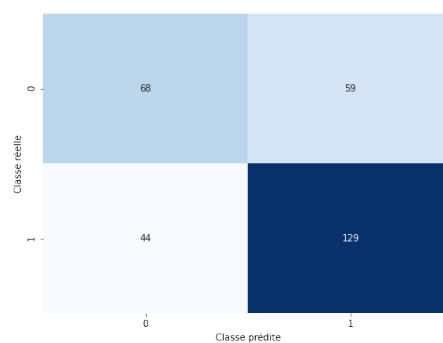


Figure 8.36: Maître

Figure 8.37: Matrices de confusion pour la régression logistique dans le cas 2

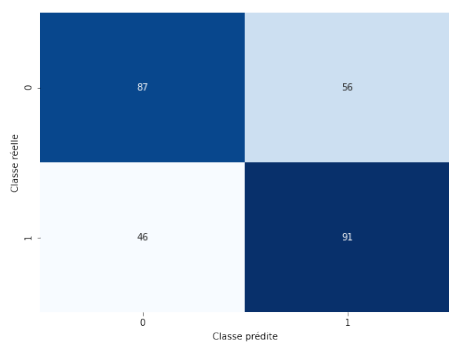


Figure 8.38: Fer

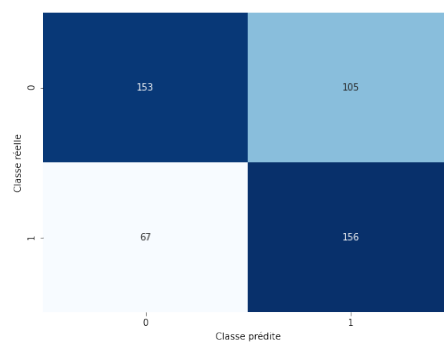


Figure 8.39: Bronze

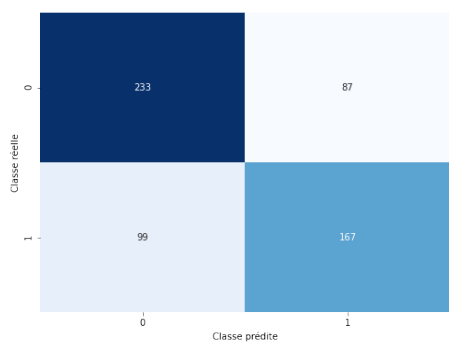


Figure 8.40: Argent

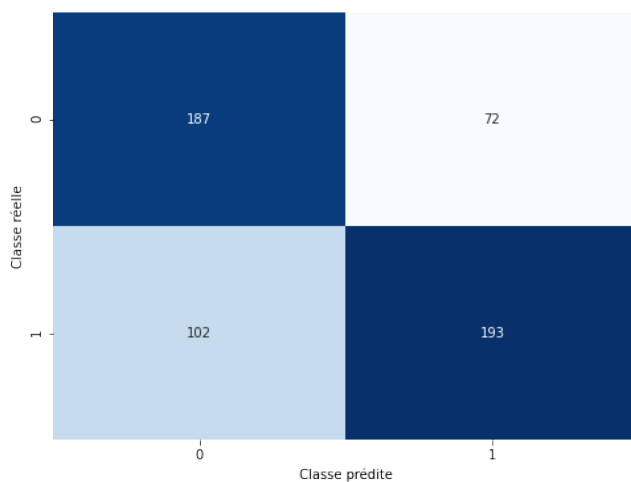


Figure 8.41: Or

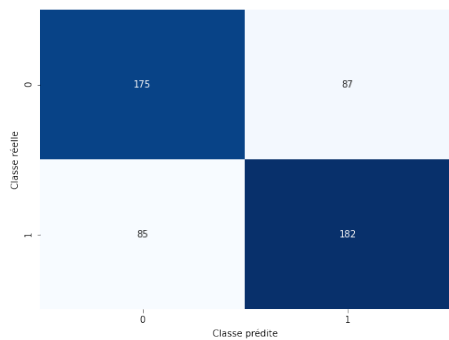


Figure 8.42: Platine

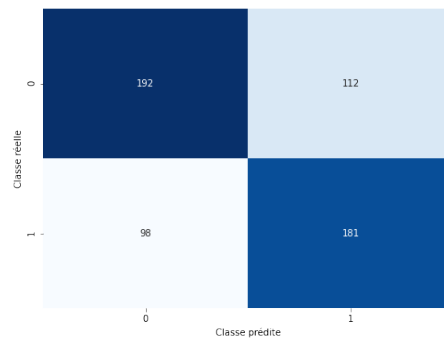


Figure 8.43: Émeraude

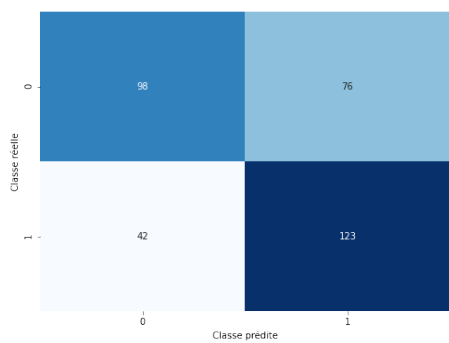


Figure 8.44: Diamant

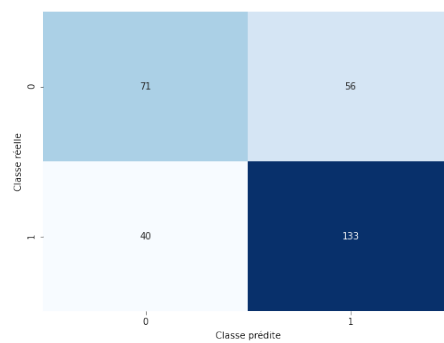


Figure 8.45: Maître

Figure 8.46: Matrices de confusion pour la régression logistique dans le cas 3

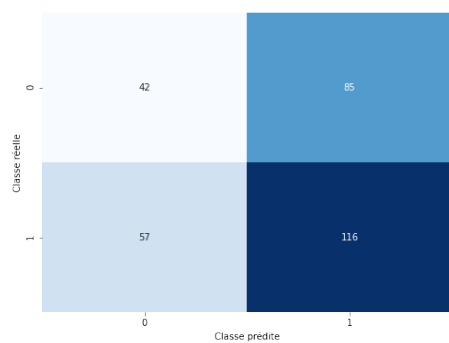


Figure 8.47: Fer

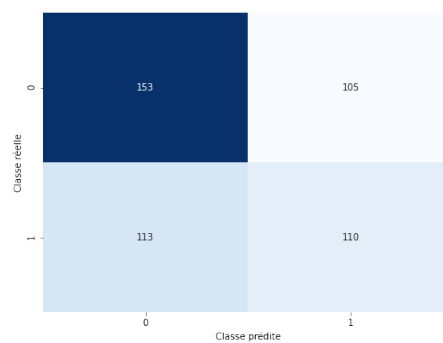


Figure 8.48: Bronze

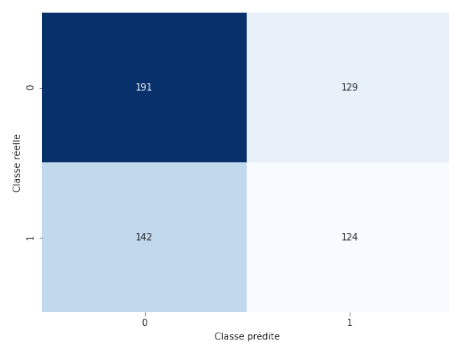


Figure 8.49: Argent

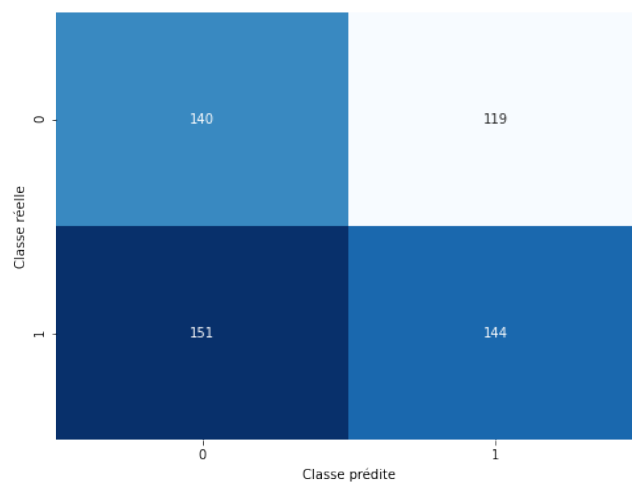


Figure 8.50: Or

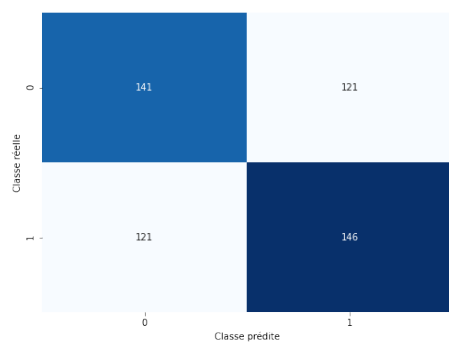


Figure 8.51: Platine

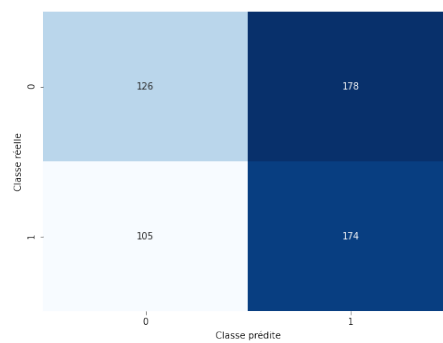


Figure 8.52: Émeraude

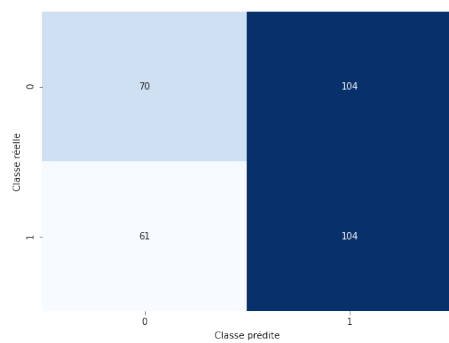


Figure 8.53: Diamant

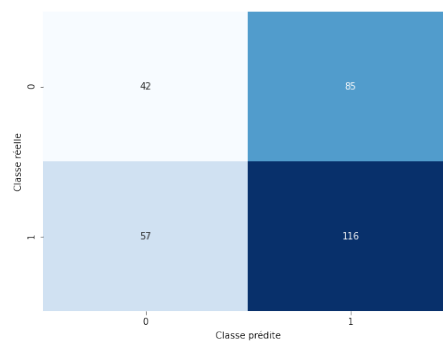


Figure 8.54: Maître

Figure 8.55: Matrices de confusion pour la régression logistique dans le cas 3

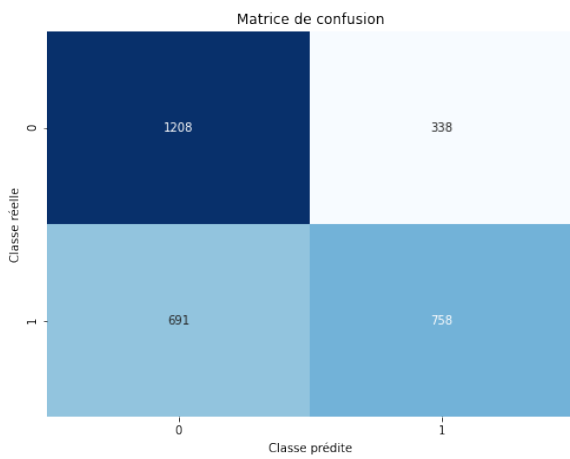


Figure 8.56: Cas 1

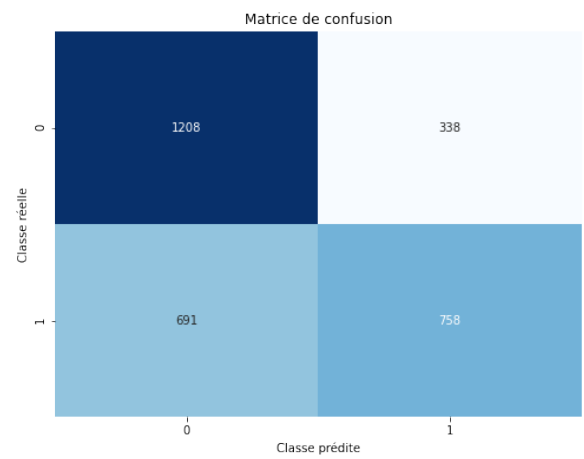


Figure 8.57: Cas 2

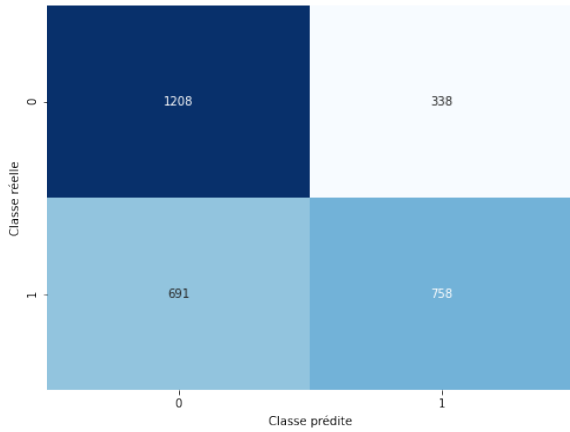


Figure 8.58: Cas 3

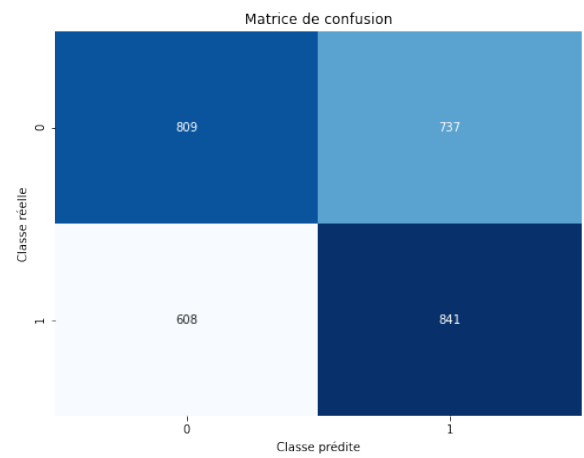


Figure 8.59: Cas 4

Figure 8.60: Matrices de confusion pour le DecisionTree sur l'ensemble des données

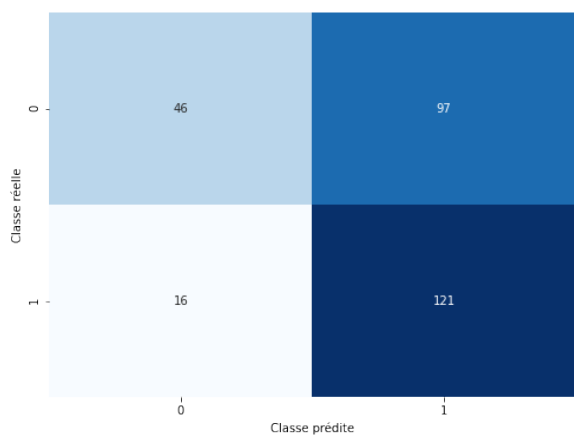


Figure 8.61: Fer

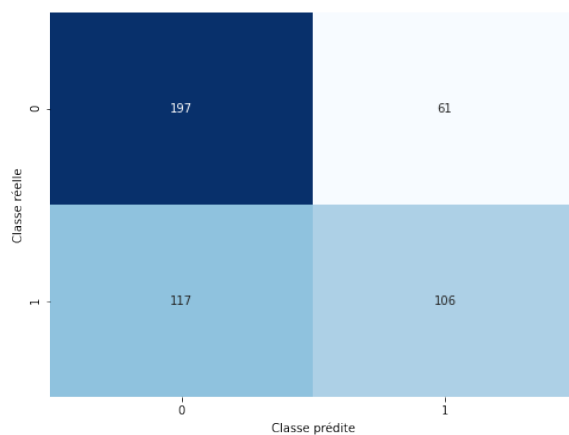


Figure 8.62: Bronze

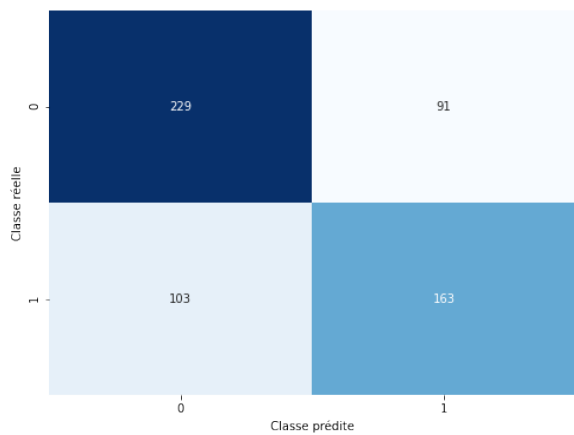


Figure 8.63: Argent

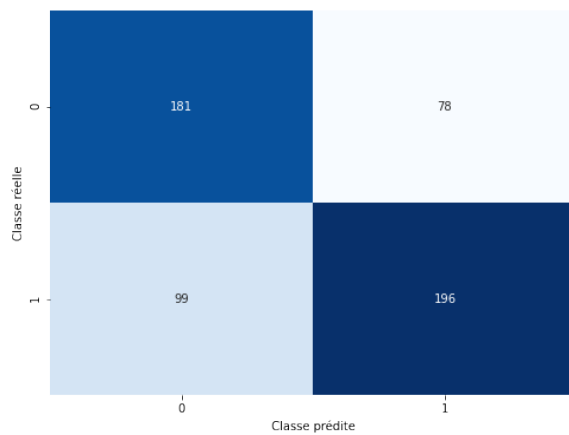


Figure 8.64: Or

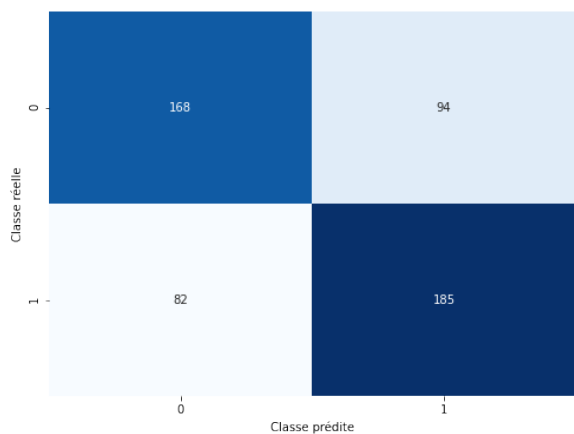


Figure 8.65: Platine

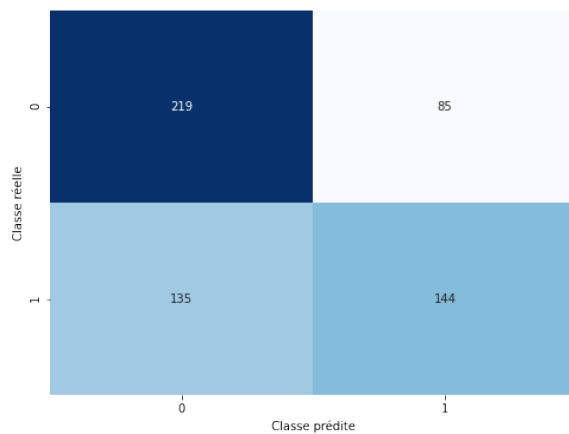


Figure 8.66: Émeraude

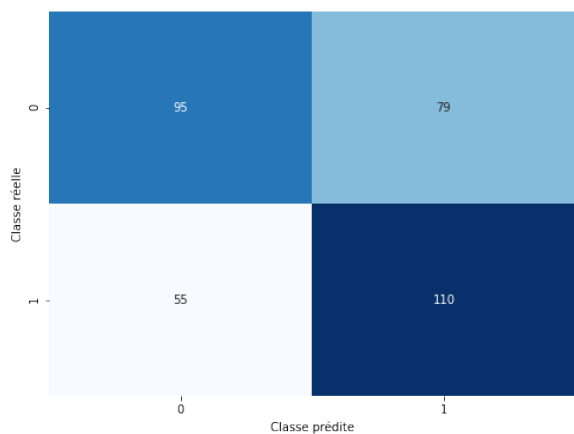


Figure 8.67: Diamant

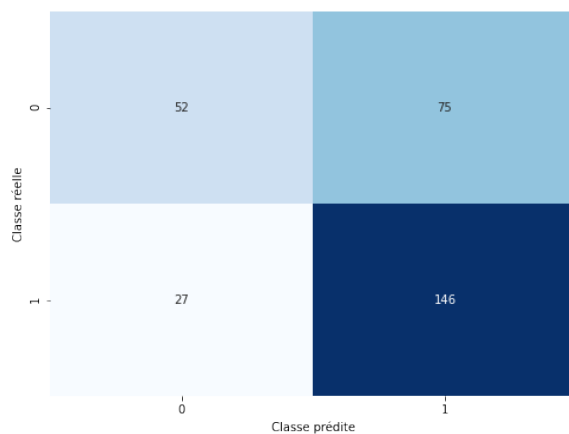


Figure 8.68: Maître

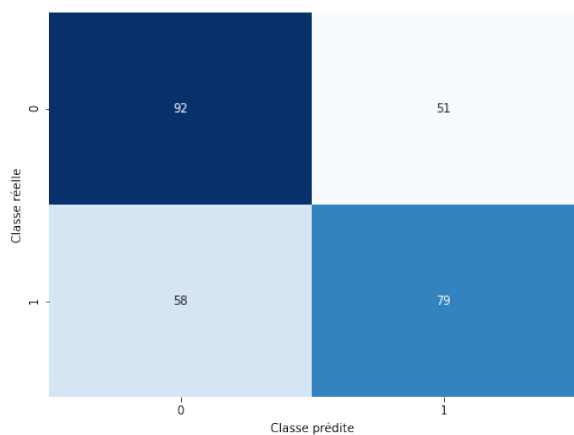


Figure 8.70: Fer

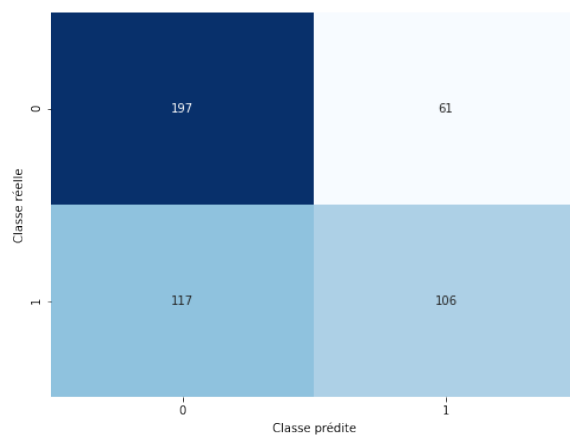


Figure 8.71: Bronze

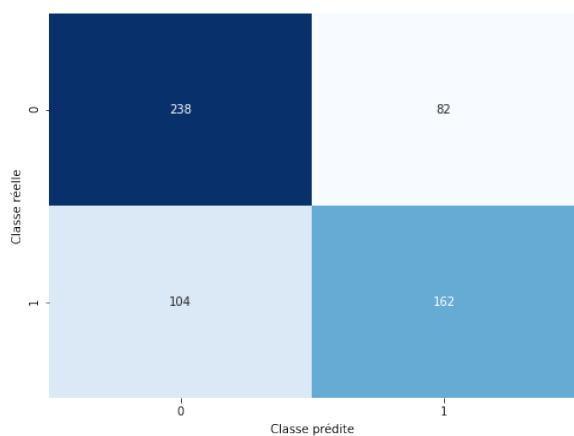


Figure 8.72: Argent

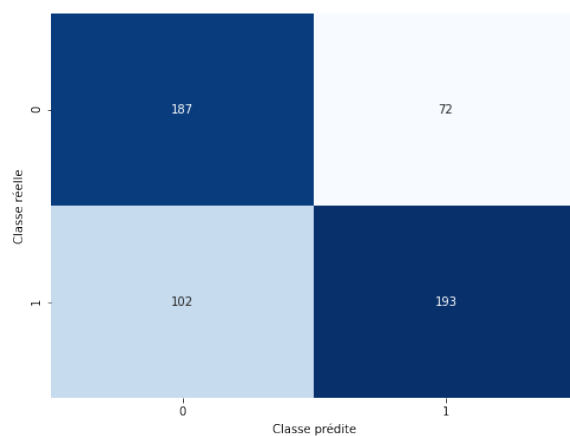


Figure 8.73: Or

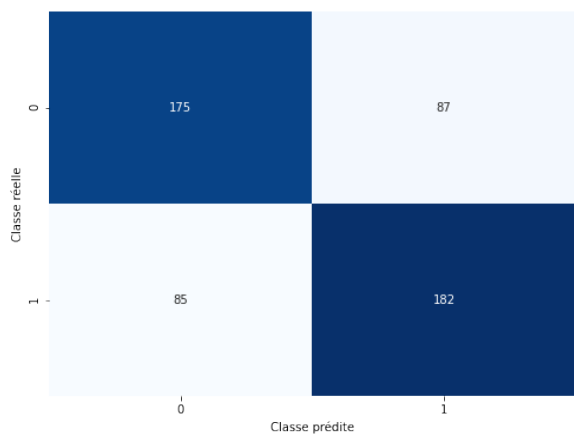


Figure 8.74: Platine

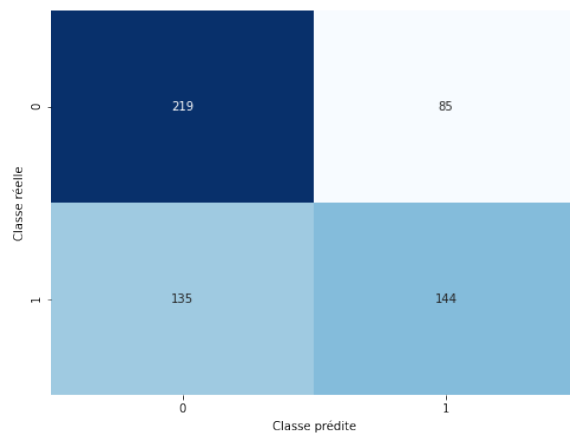


Figure 8.75: Émeraude

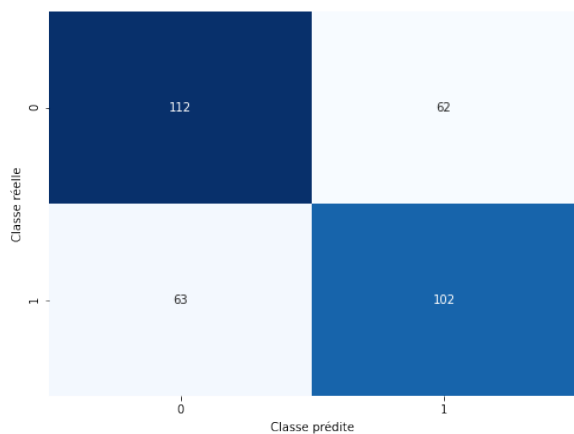


Figure 8.76: Diamant

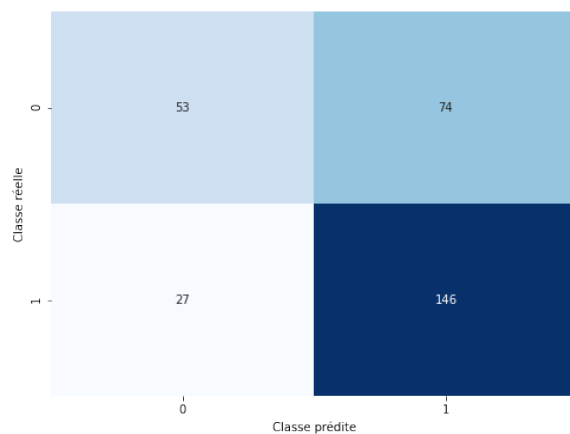


Figure 8.77: Maître

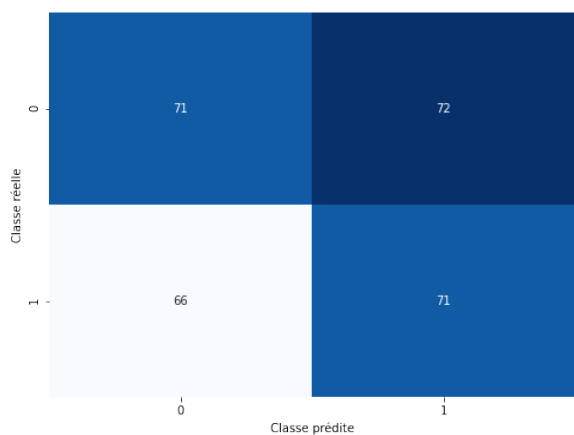


Figure 8.79: Fer

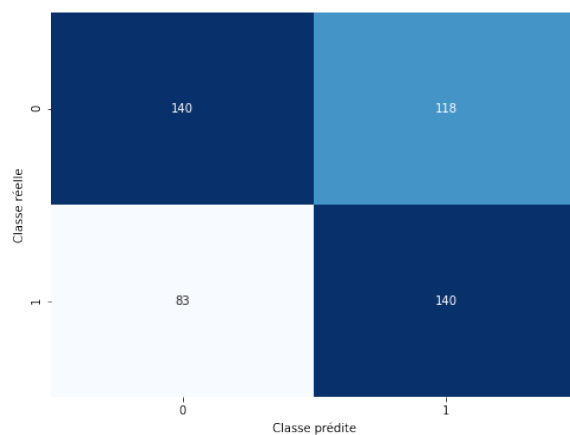


Figure 8.80: Bronze

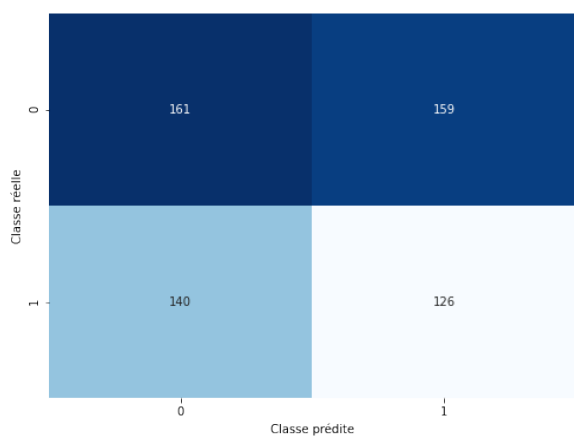


Figure 8.81: Argent

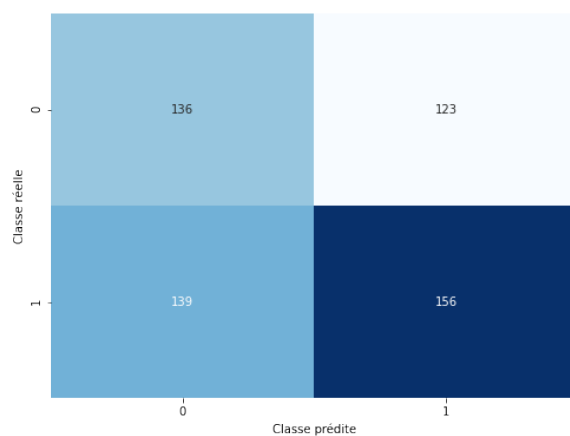


Figure 8.82: Or

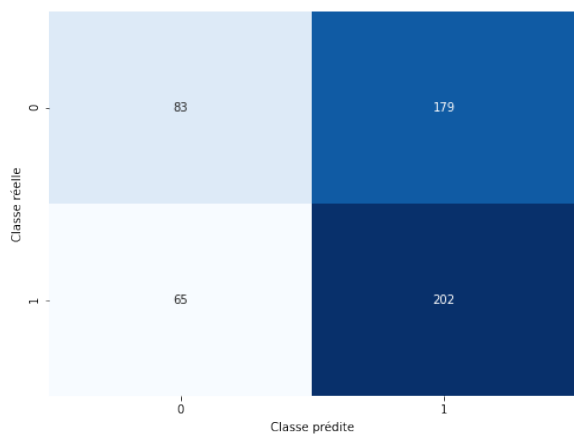


Figure 8.83: Platine

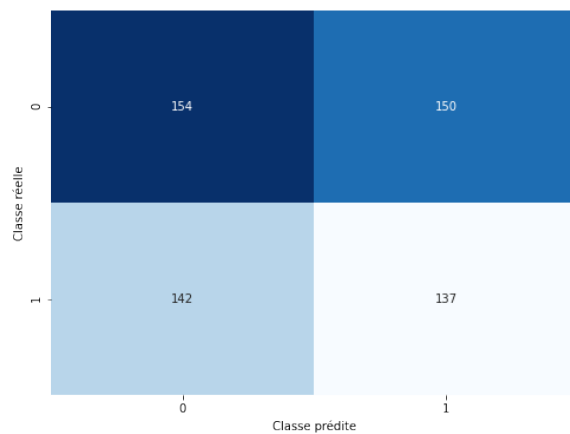


Figure 8.84: Émeraude

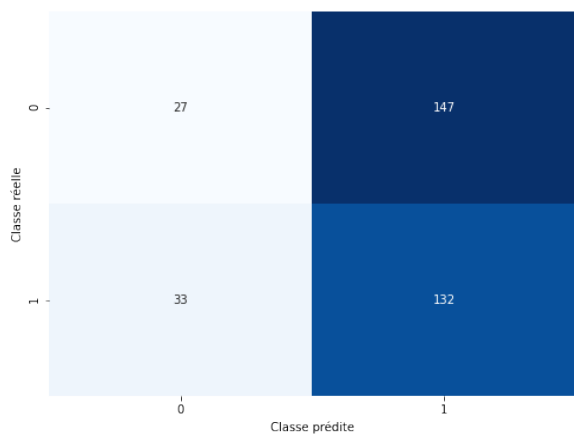


Figure 8.85: Diamant

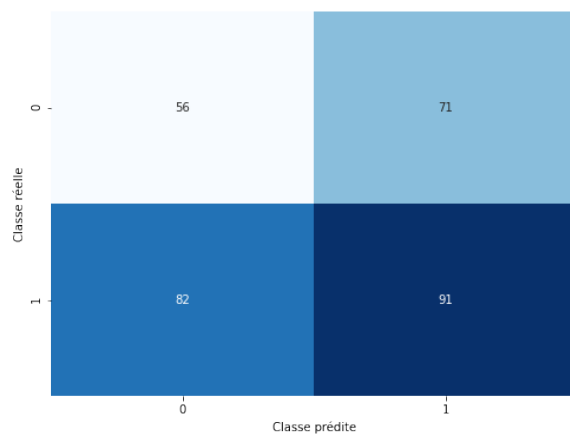


Figure 8.86: Maître

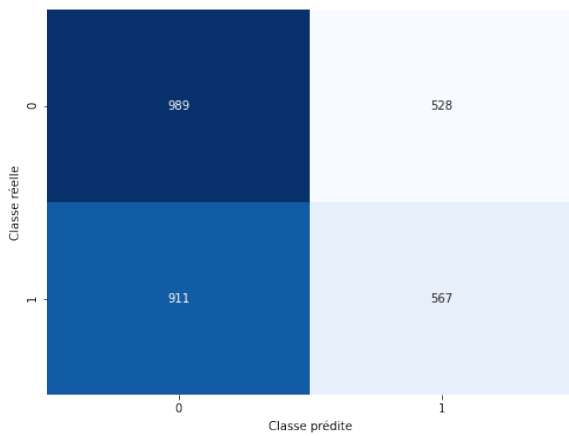


Figure 8.88: Cas 1

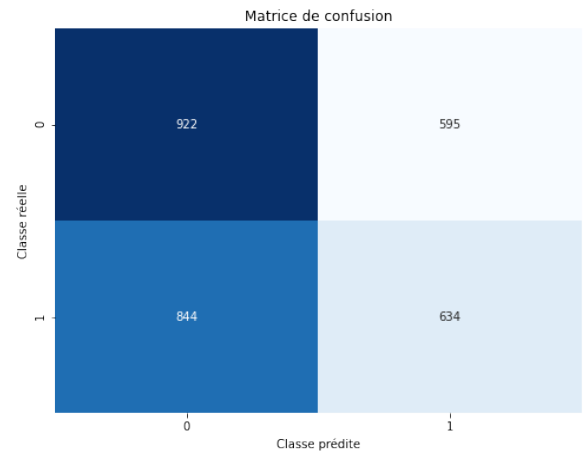


Figure 8.89: Cas 2

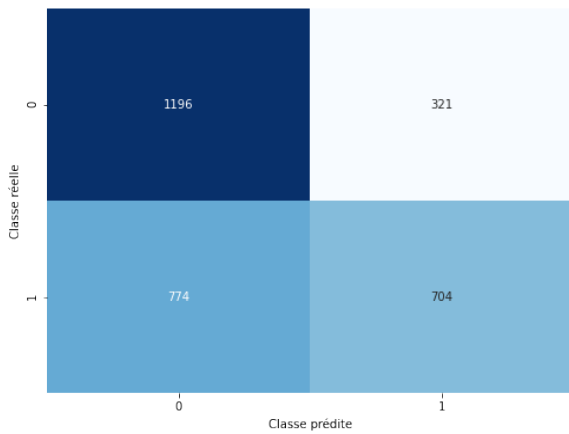


Figure 8.90: Cas 3

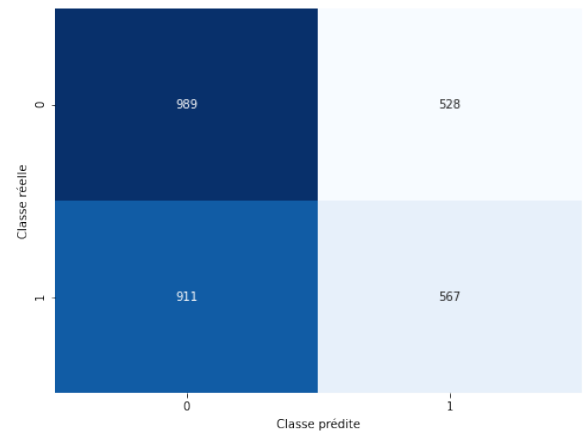


Figure 8.91: Cas 4

Figure 8.92: Matrices de confusion pour le KNN sur l'ensemble des données

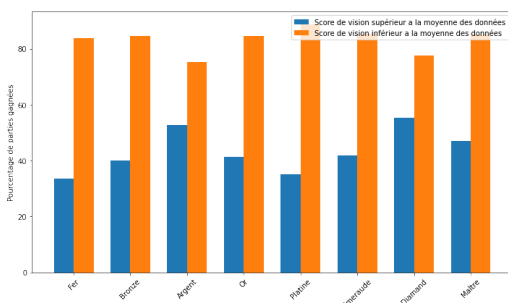


Figure 8.93: Pourcentage de victoire comparé au score moyen calculé sur l'ensemble du fichier

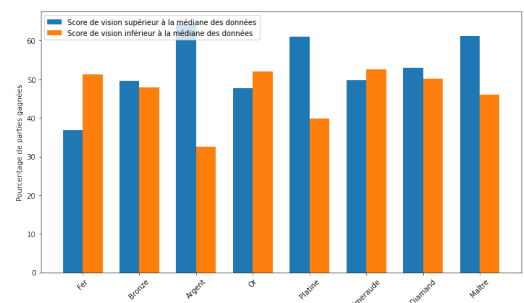


Figure 8.94: Pourcentage de victoire comparé au score médian calculé sur l'ensemble du fichier

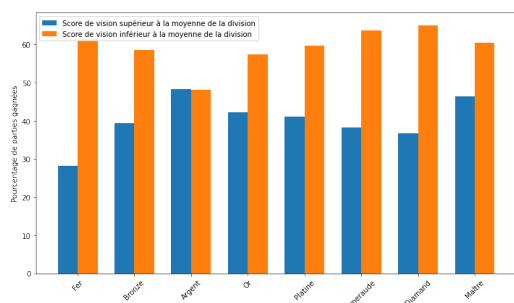


Figure 8.95: Pourcentage de victoire comparé au score moyen calculé sur l'ensemble du fichier

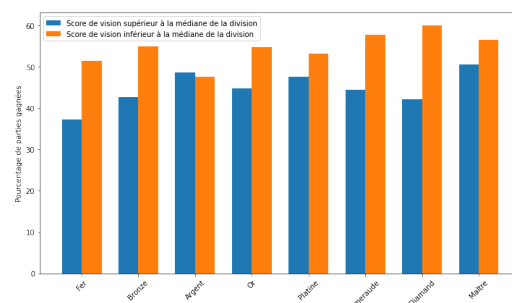


Figure 8.96: Pourcentage de victoire comparé au score médian calculé sur l'ensemble du fichier

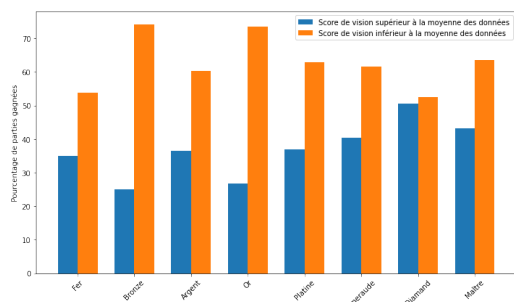


Figure 8.97: Pourcentage de victoire comparé au score moyen calculé sur l'ensemble du fichier

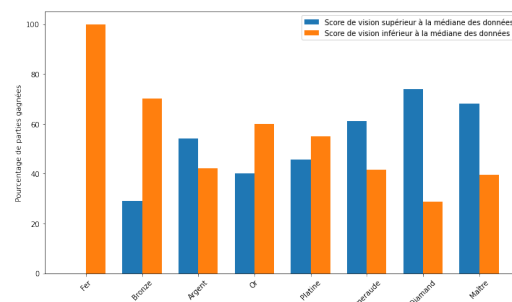


Figure 8.98: Pourcentage de victoire comparé au score médian calculé sur l'ensemble du fichier

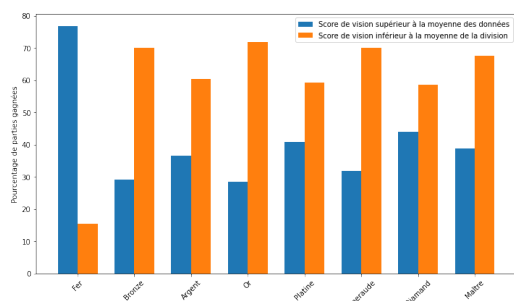


Figure 8.99: Pourcentage de victoire comparé au score moyen calculé par division

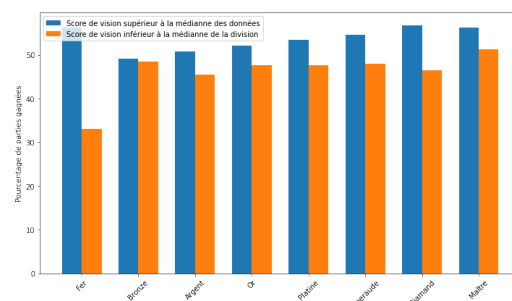


Figure 8.100: Pourcentage de victoire comparé au score médian calculé par division

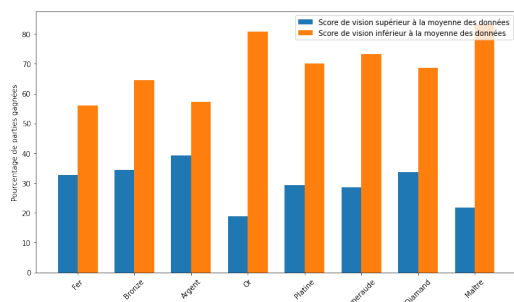


Figure 8.101: Pourcentage de victoire comparé au score moyen calculé sur l'ensemble du fichier

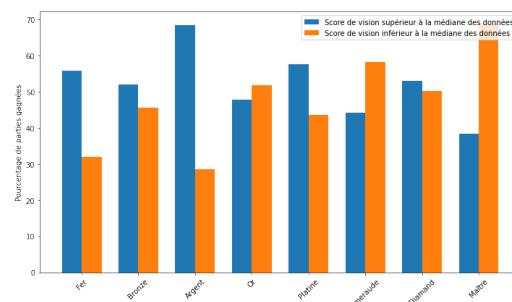


Figure 8.102: Pourcentage de victoire comparé au score médian calculé sur l'ensemble du fichier **probleme de label**

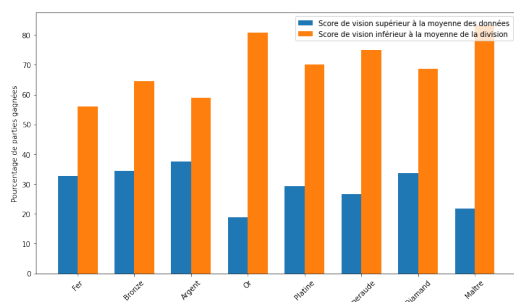


Figure 8.103: Pourcentage de victoire comparé au score moyen calculé par division

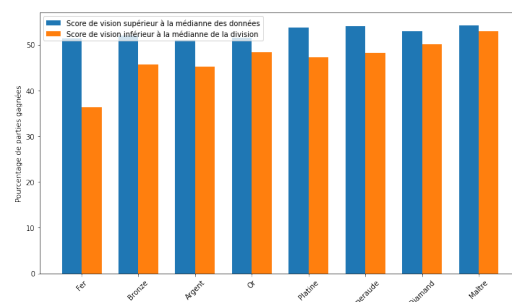


Figure 8.104: Pourcentage de victoire comparé au score médian calculé par division

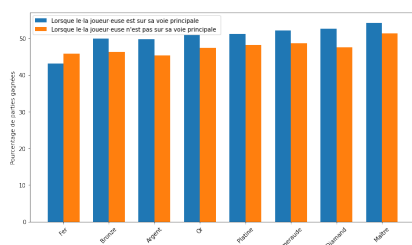


Figure 8.105: Pourcentage de partie gagnée selon que l'on joue sur sa voie principale ou non

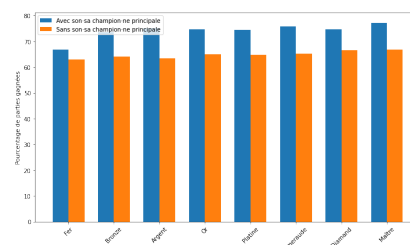


Figure 8.106: Pourcentage de victoire par division selon le fait qu'on ai joué avec le champion avec lequel on joue le plus souvent

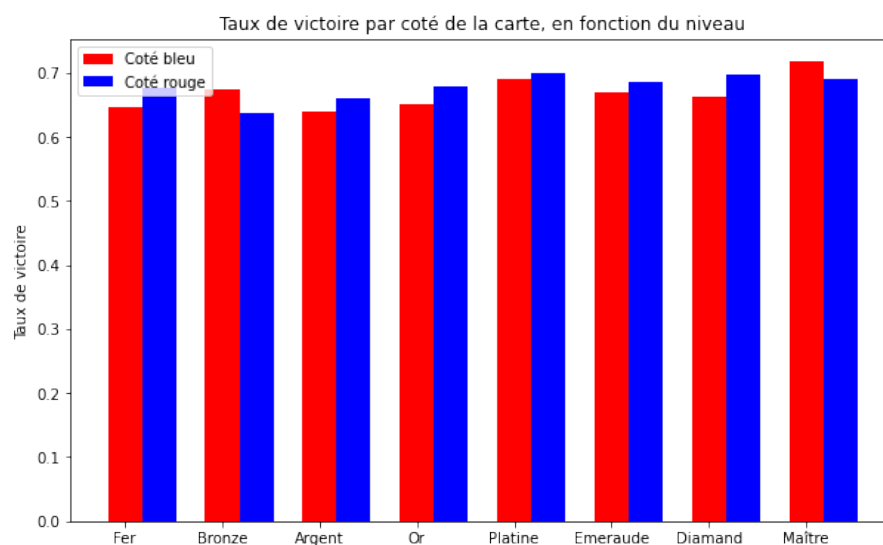


Figure 8.107: Taux de victoire par division

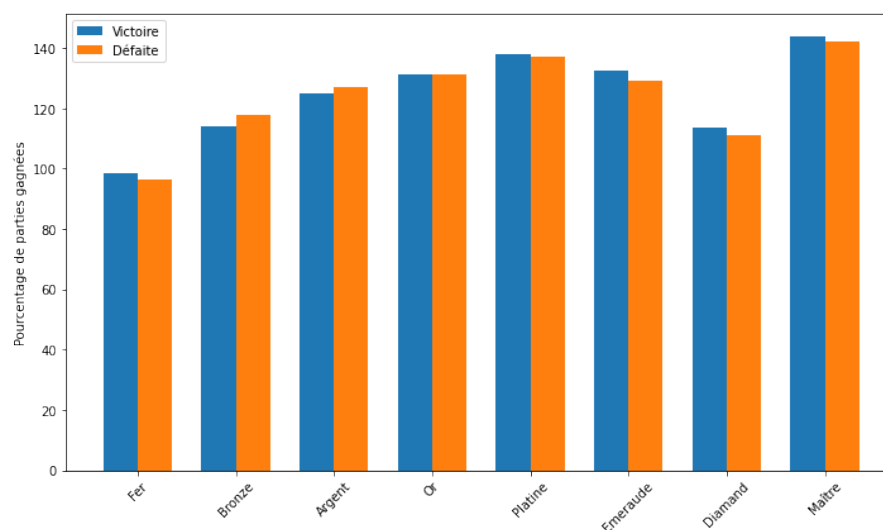


Figure 8.108: Nombre de sbires tués en moyenne par rapport à la défaite ou la victoire

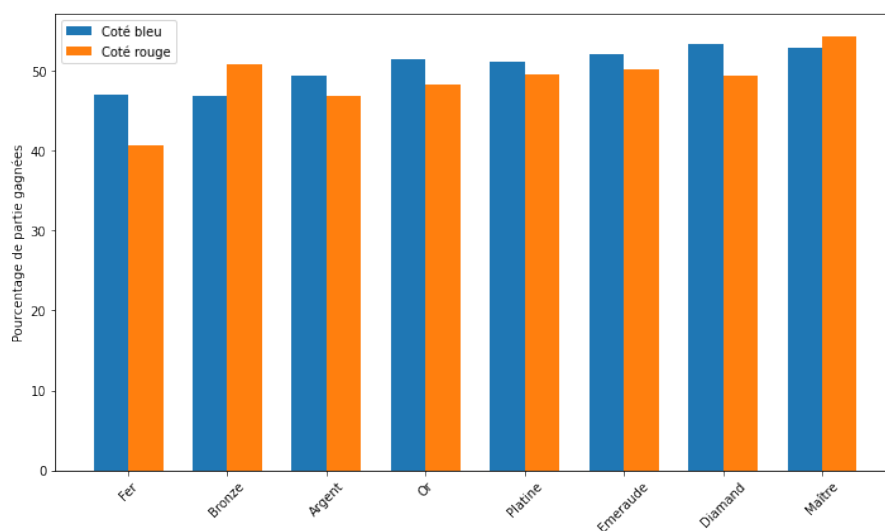


Figure 8.109: Caption

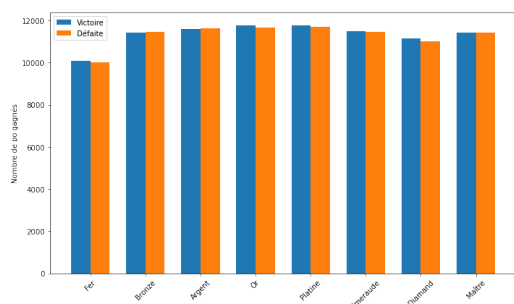


Figure 8.110: Nombre de po gagné en fonction de la victoire et de la défaite

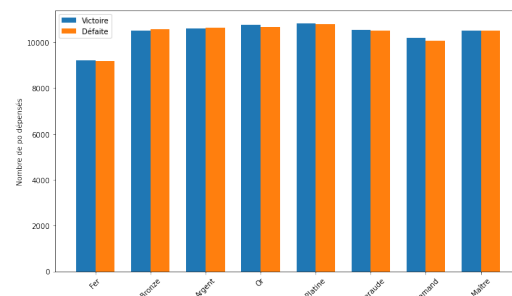


Figure 8.111: Nombre de po dépensé en fonction de la victoire et de la défaite

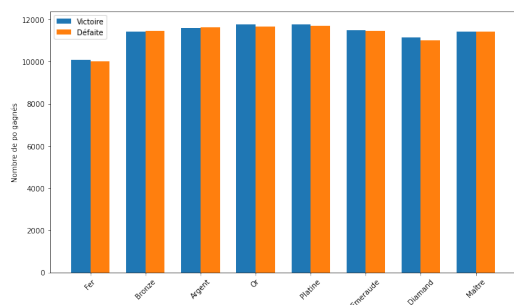


Figure 8.112: Nombre de dégâts fait en fonction de la victoire et de la défaite

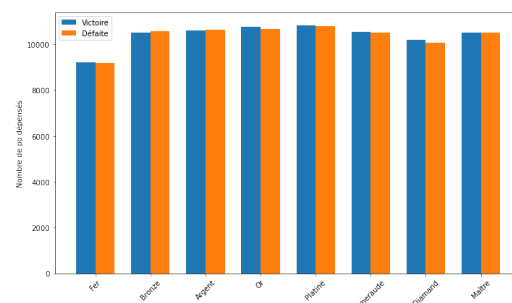


Figure 8.113: Nombre de dégâts subit en fonction de la victoire et de la défaite

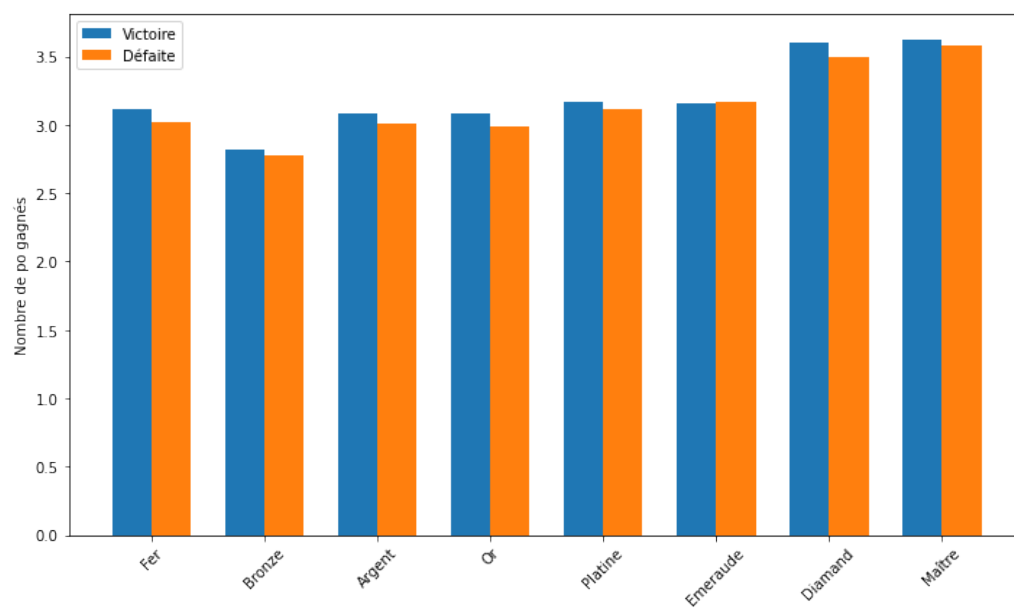


Figure 8.114: kda_mean_winrate

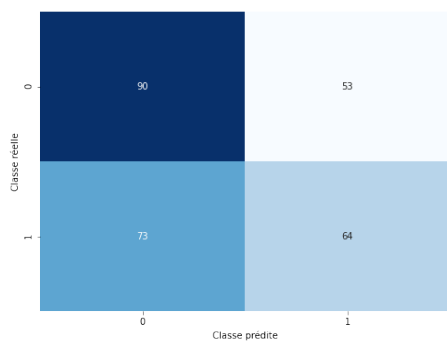


Figure 8.115: Fer

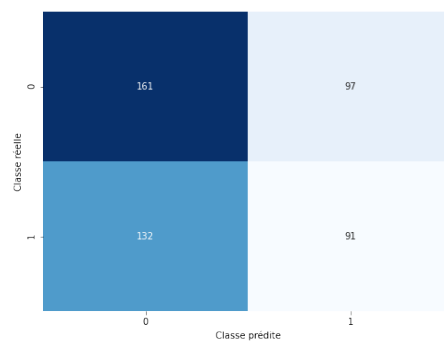


Figure 8.116: Bronze

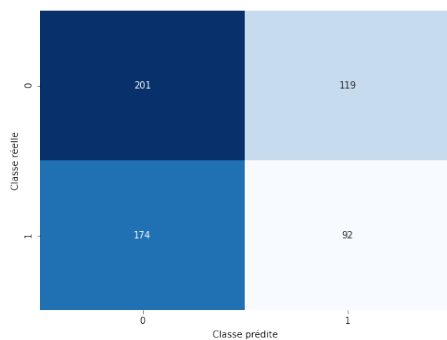


Figure 8.117: Argent

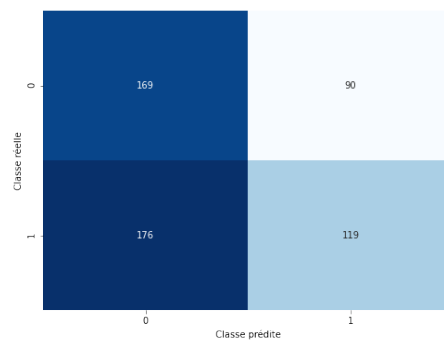


Figure 8.118: Or

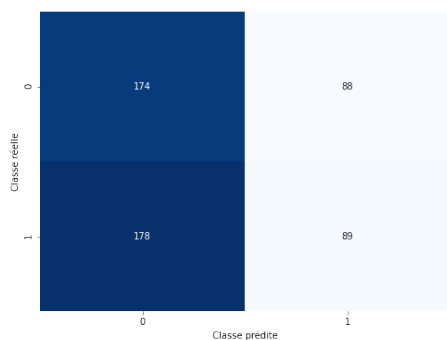


Figure 8.119: Platine

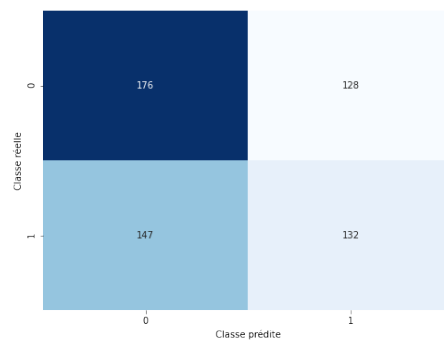


Figure 8.120: Émeraude

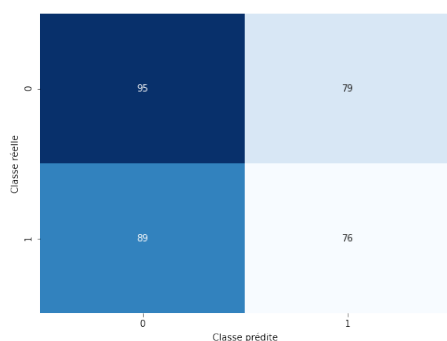


Figure 8.121: Diamant

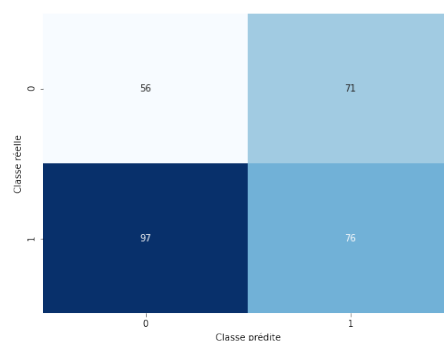


Figure 8.122: Maître

Figure 8.123: Matrices de confusion pour le KNN dans le cas 1, par division

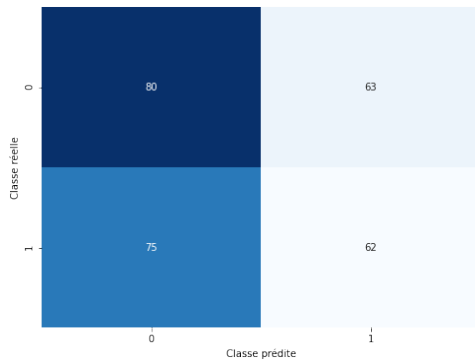


Figure 8.124: Fer

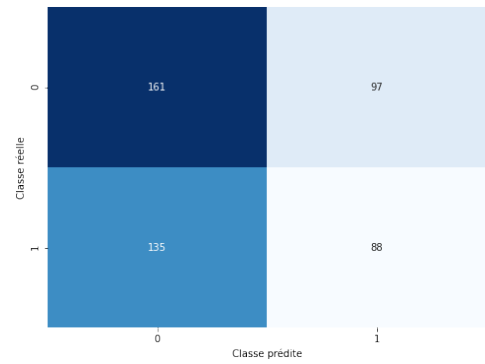


Figure 8.125: Bronze

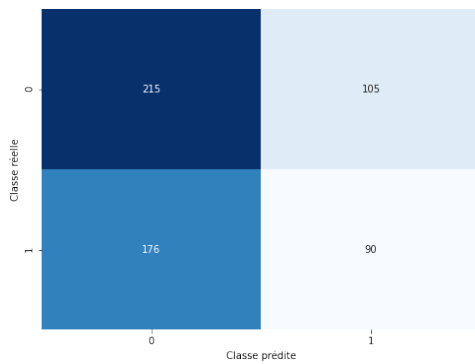


Figure 8.126: Argent

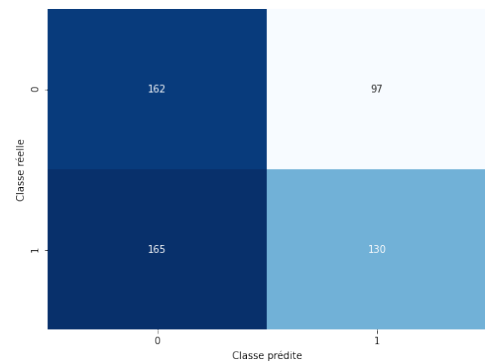


Figure 8.127: Or

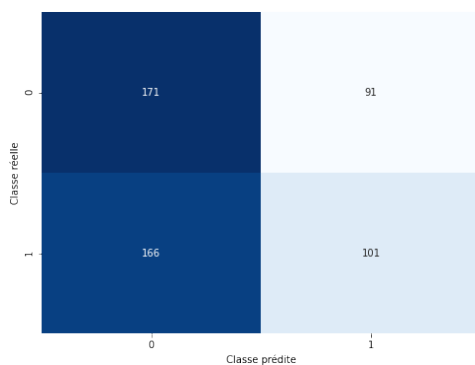


Figure 8.128: Platine

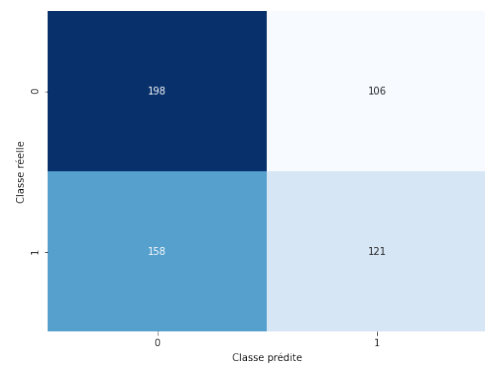


Figure 8.129: Émeraude

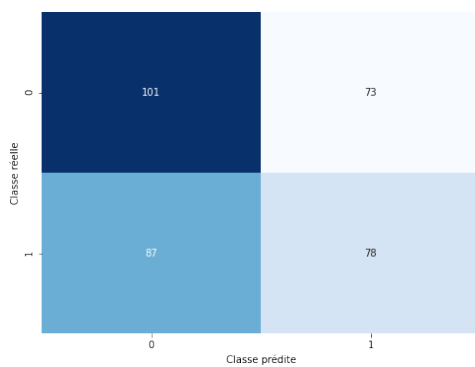


Figure 8.130: Diamant

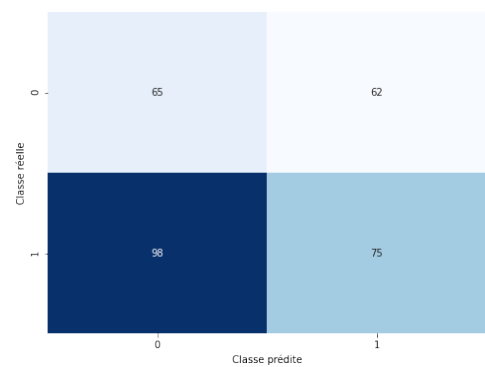


Figure 8.131: Maître

Figure 8.132: Matrices de confusion pour le KNN dans le cas 2, par division

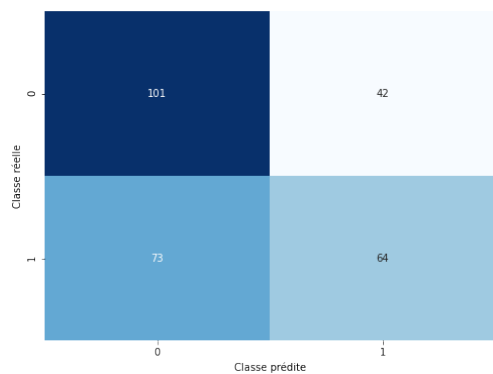


Figure 8.133: Fer

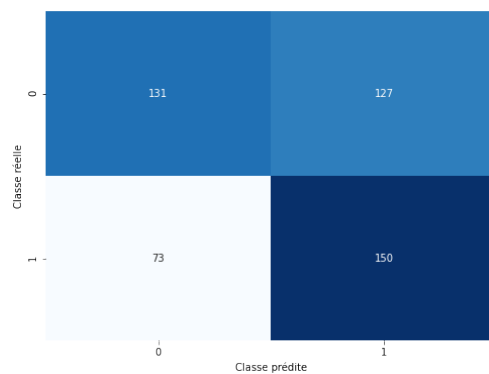


Figure 8.134: Bronze

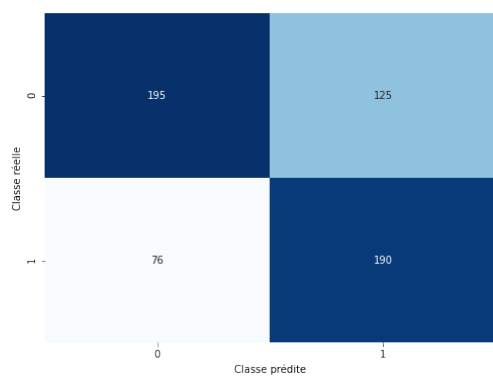


Figure 8.135: Argent

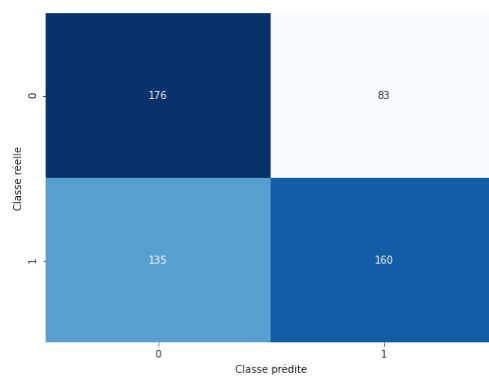


Figure 8.136: Or

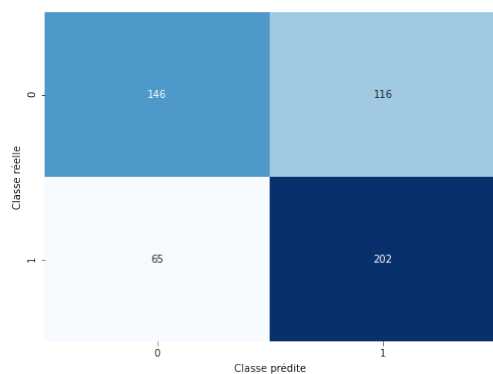


Figure 8.137: Platine

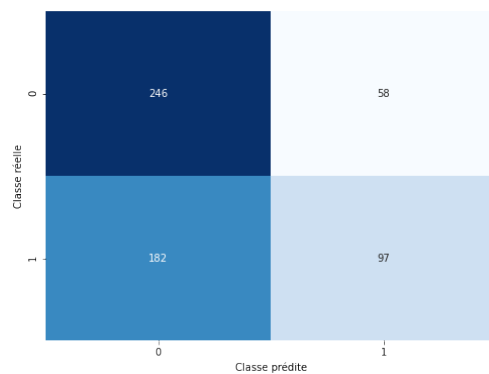


Figure 8.138: Émeraude

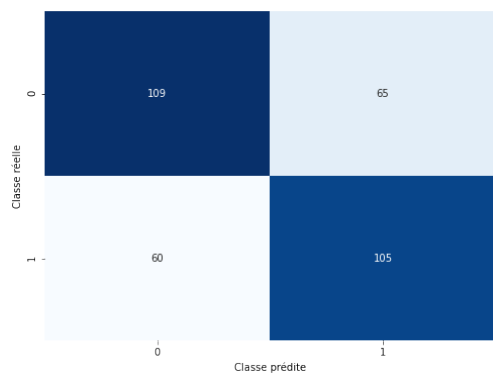


Figure 8.139: Diamant

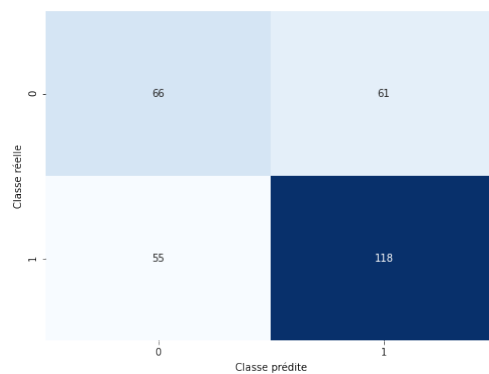


Figure 8.140: Maître

Figure 8.141: Matrices de confusion pour le KNN dans le cas 3, par division

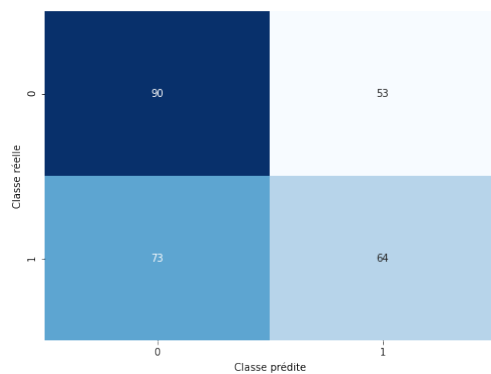


Figure 8.142: Fer

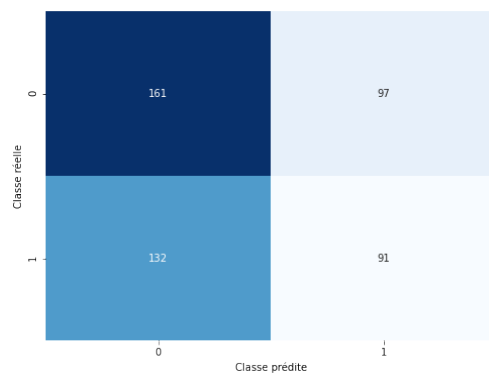


Figure 8.143: Bronze

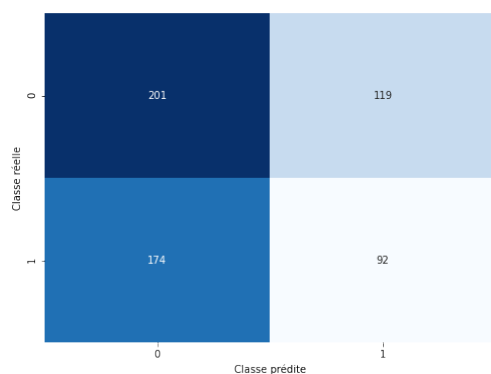


Figure 8.144: Argent

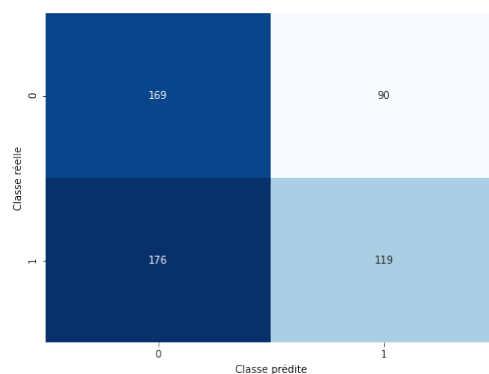


Figure 8.145: Or

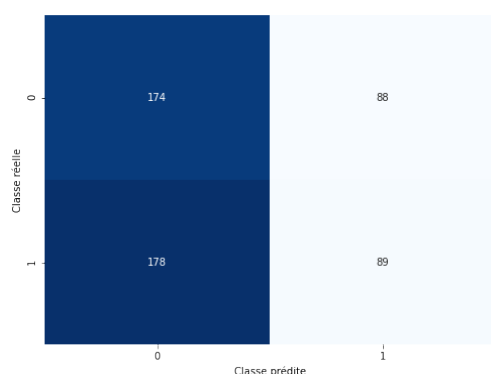


Figure 8.146: Platine

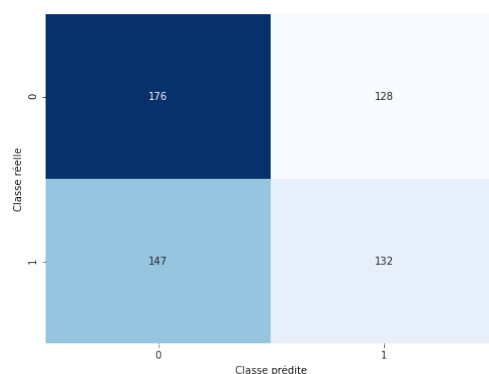


Figure 8.147: Émeraude

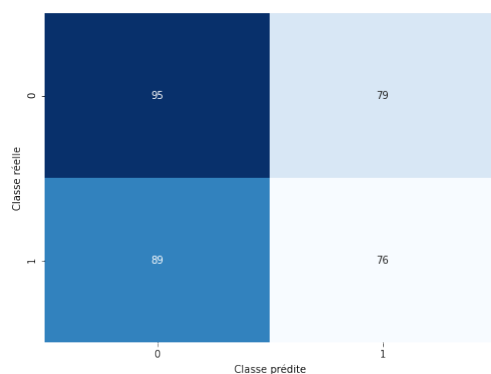


Figure 8.148: Diamant

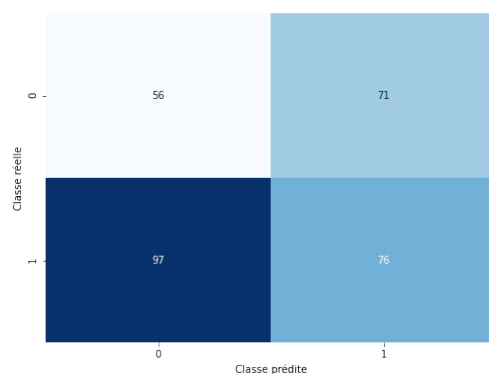


Figure 8.149: Maître

Figure 8.150: Matrices de confusion pour le KNN dans le cas 4, par division

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl