École Centrale de Nantes

INFO IA - PROJET
Rapport

Prédiction de parties pour le jeu League of Legends

Élèves :
Clément Auclin
Félix Doublet
Vincent Gallot
Paul Trincklin

Enseignants :
Morgan Magnin
Tony Ribeiro

28 octobre 2024

Table des matières

1 Présentation du projet

Ce rapport est une présentation du travail réalisé entre octobre 2022 et mars 2023 par notre groupe dans le cadre du projet de l'option informatique IA de Centrale Nantes lors de l'année 2022-2023.

La présentation qui suit ne se veut pas exhaustive, mais donne un aperçu global du travail réalisé au cours de ces derniers mois.

L'ensemble de nos travaux est disponibles sur GitHub :
https://github.com/VincentEnOptionRV/LoL_ML

1.1 Présentation du jeu League of Legends

League of Legends (abrégé LoL) est un jeu vidéo sorti en 2009 de type MOBA [1] free-to-play, développé et édité par Riot Games. Avec plus de 150 millions de joueurs mensuels, League of Legends fait partie des jeux les plus joués de l'histoire du jeu vidéo, et est souvent cité comme le jeu ayant la plus importante scène esport, avec des compétitions régionales et internationales regardées par des dizaines de millions de téléspectateurs.

Le mode principal du jeu voit s'affronter deux équipes de 5 joueurs en temps réel dans des parties d'une durée d'environ une demi-heure, chaque équipe occupant et défendant sa propre base sur la carte. Chacun des dix joueurs contrôle un personnage à part entière parmi les plus de 160 qui sont proposés. Ces personnages, connus sous le nom de "champions", disposent de compétences uniques et d'un style de jeu qui leur est propre. Ils gagnent en puissance au fil de la partie en amassant des points d'expérience ainsi qu'en achetant des objets, dans le but de battre l'équipe adverse. L'objectif d'une partie est de détruire le "Nexus" ennemi, une large structure située au centre de chaque base.

Chaque joueur choisit son personnage lors de la phase de sélection des champions, qui a lieu en amont de la partie. Lors de cette phase, chacun des 10 joueurs a la possibilité de bannir un champion, qui ne pourra alors pas être sélectionné pour cette partie. Ensuite, les joueurs de chaque équipe choisissent tour à tour un champion, leur choix étant visible de tous. Ainsi, un joueur effectuant son choix connaît les choix de ceux qui l'ont précédé, et peut adapter sa décision pour avoir une meilleure synergie avec son équipe, ou pour mieux contrer l'équipe adverse. La sélection des champions est une étape cruciale d'une partie de League of Legends, et une équipe peut commencer la partie avec un sérieux avantage si elle réussit à construire une combinaison de champions qui domine les champions adverses lors de leurs interactions.

LoL est un jeu en constante évolution par les mises à jour bihebdomadaires qui équilibrent la puissance des différents objets, champions, et autres éléments du jeu. À ce titre, la méta [2] évolue constamment et d'une semaine à l'autre, la puissance de certains champions peut varier grandement, ce qui influence leur taux de sélection par les joueurs lors de la sélection des champions.

1. Multiplayer Online Battle Arena, ou arène de bataille multijoueur en ligne
2. Francisation de META : Most Efficient Tactics Available, qui représente un consensus des joueurs sur les choix tactiques les plus forts du moment

1.2 Objectif du projet

L'objectif fonctionnel de ce projet est de constituer une preuve de concept d'un outil d'aide au joueur pour le mode en ligne compétitif de League of Legends sur des données disponibles publiquement. Nous nous sommes concentrés sur la prédiction de victoire théorique à partir des données disponibles au début d'une partie, puis sur la création d'un outil de conseil en sélection des champions.

D'un point de vue pratique, notre mission a été de comprendre et d'apprendre à résoudre des problématiques liées à l'exploitation de différents jeux de données par machine learning, de l'extraction de ces données via différentes API à leur structuration et analyse grâce aux bibliothèques dédiées aux sciences de données de python.

Au-delà de l'obtention de résultats convaincants ou de la production d'un outil performant, ce projet a pour but de nous permettre de mettre les mains dans un problème de manipulation de données tel que rencontré dans le quotidien d'une partie des ingénieurs en informatique.

1.3 Organisation

Dès le commencement du projet, il a été établi que la communication entre le groupe étudiant et les deux encadrants s'effectuerait par l'intermédiaire d'un serveur Discord dédié, logiciel de messagerie instantanée et de communication audio. La communication entre étudiants s'effectuait via un groupe Messenger dédié ainsi qu'oralement lors des temps de pause entre les cours.

Tout au long du projet, des réunions régulières ont été organisées pour faire un point sur l'avancement et discuter des objectifs au rythme d'une rencontre toutes les deux ou trois semaines.

1.4 Déroulement du projet

Le projet a été découpé en deux phases distinctes. La première a consisté jusqu'en janvier à récupérer les données, les comprendre et créer un modèle de Machine Learning capable de prédire au mieux la victoire ou la défaite des équipes. Nous avons eu l'occasion au cours de cette partie d'effectuer de la récupération de données à l'aide d'une API, d'effectuer du scrapping et du feature engineering pour optimiser au mieux des modèles de XGBoost. Ces modèles ont ensuite été comparés afin de mettre en évidence les différentes améliorations apportées au cours des premiers mois. Lors de la deuxième partie du projet, le groupe a été en grande partie séparé en deux tâches distinctes :

- Deux personnes étaient chargées de permettre de détecter les images des champions à partir de l'image de la draft (phase de sélection des personnages dans League of Legend) et d'appliquer des algorithmes de Deep Learning dessus afin de reconnaitre les différents champions. Pour cela plusieurs modèles ont été essayés (des FCNs et une modification de Resnet).
- Les deux autres membres du groupe devaient quant à eux trouver le moyen de recommander des champions à un joueur pour l'aider à en sélectionner un. Cette recommandation doit prendre en compte les autres champions choisis, les bannissements de champions ainsi que les champions régulièrement choisis par le joueur.

La phase finale du projet permet de réunir l'ensemble des étapes précédentes afin de créer une interface graphique utilisable par un joueur qui ne demanderait en entrée qu'une image du client LoL et qui serait capable de fournir une recommandation à un joueur en détectant l'ensemble des autres champions, etc. L'organisation du projet au cours des derniers mois est résumée dans le graphique ci-dessous.

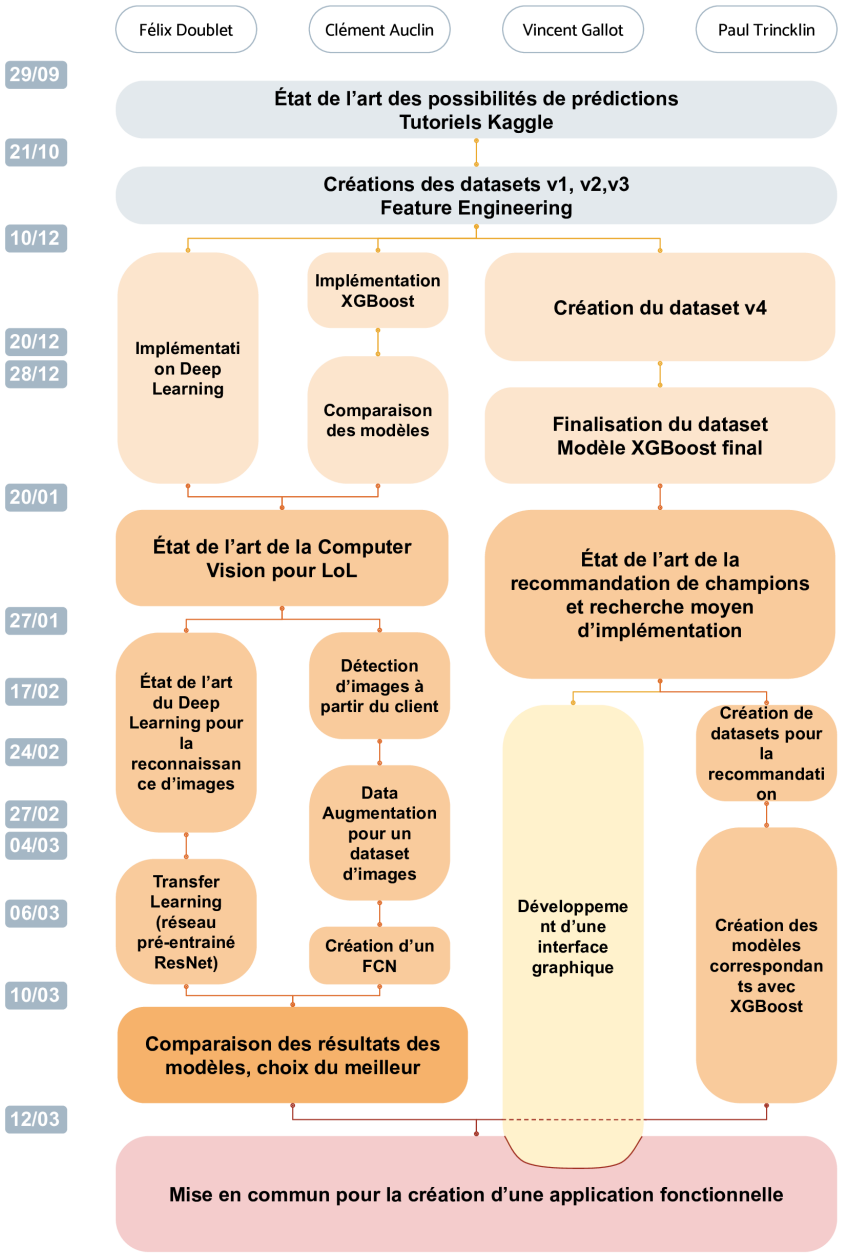


FIGURE 1 – Timeline du Projet

2 Outils utilisés

Afin de collaborer de concert et de pouvoir se partager du code, nous avons consigné toutes nos contributions dans un repository GitHub. Nous l'avons organisé en dossiers et

sous-dossiers afin de segmenter les différentes parties du projet et les différentes contributions.

Nous avons travaillé essentiellement sous la forme de notebook iPython. Ce type de fichier permet de mêler des blocs en Markdown [3] avec des blocs en Python, rendant le code plus simple à exécuter ainsi que la relecture du code et le suivi du raisonnement plus limpide.

Aussi, nous avons appris à manier l'API REST de LoL afin de récolter les informations nécessaires à la création du dataset.

Enfin, pour mener à bien le projet, nous avons dû nous familiariser et utiliser les grandes bibliothèques Python pour le traitement des données et le Machine Learning :

— Pandas pour la manipulation des données et la création de features
— Scikit-Learn pour les algorithmes de Machine Learning
— TensorFlow et Keras pour les algorithmes de Deep Learning

Afin d'exploiter l'API du jeu ou de faire du webscraping [4] nous avons également été amenés à utiliser des bibliothèques telles que request ou BeautifoulSoup qui permettent d'envoyer des requêtes web http et extraire du contenu de la réponse.

Nous avons également utilisé des outils de visualisation tels que matplotlib ou seaborn afin d'obtenir une représentation du dataset ou de nos résultats.

3 Création du dataset

3.1 État de l'art

Durant la première partie du projet, nous nous sommes concentrés sur la création d'un jeu de données servant à entraîner un modèle de classification binaire pour le résultat d'une partie de *League of Legends*. On dispose, au moment du lancement de la partie, de nombreuses informations sur les 10 joueurs jouant la partie ainsi que sur les *champions* qu'ils ont sélectionnés. Cette tâche a été déjà traitée dans d'autres papiers ([1], [2], [3]) ainsi que sur d'autres *MOBAs* tels que DOTA 2 ([4], [5]). Une étude préliminaire de ces études nous ont permis d'identifier une liste de variables, ou *features*, qui semblent être déterminantes dans le résultat de la partie. Les meilleurs résultats obtenus dans l'état de l'art sont de l'ordre d'une précision de 70% à 75% de bons résultats, c'est donc cet objectif que nous nous sommes fixés dans cette étude. D'autres variantes de ce problème ont été étudiées, comme le fait d'intégrer des évènements se produisant au cours de la partie pour pouvoir en prédire le résultat, mais nous avons jugé plus intéressant de ne garder que les informations disponibles avant la partie.

3.2 Prise en main de l'API Riot

La première étape a été de se familiariser avec le fonctionnement de l'API et de comprendre comment l'utiliser pour récolter les données dont nous aurions besoin.

Pour cela, nous avons d'abord dû créer un compte sur le site développeur de Riot (https://developer.riotgames.com) afin d'obtenir une clef personnelle, valide pendant 24 h et renouvelable nous permettant d'effectuer des requêtes auprès de l'API.

3. Langage de balisage offrant une syntaxe facile à lire
4. Technique d'extraction de données depuis un site web

Une limite est également fixée : nous n'avons pas le droit à plus de 20 requêtes par seconde ou à plus de 100 requêtes toutes les 2 minutes.

Le portail développeur de Riot se présente avec une documentation complète ainsi qu'une présentation des différentes requêtes réalisables et des données envoyées correspondantes.

Les requêtes se présentent sous différentes catégories :

— Champion-Mastery : permet d'obtenir des données sur les champions les plus joués par un joueur donné
— League : permet d'obtenir les identifiants des joueurs classés à un rang donné
— Match : permet d'obtenir toutes les données sur une partie précise

3.3 Récolte des données

La récolte des données a été réalisée à partir de 3 sources : l'API riot *League of legends*, ainsi que deux sites tiers qui fournissent de nombreux renseignements sur les joueurs et sur les parties, `www.mobachampion.com` et `u.gg`.

Le nombre de requêtes acceptées par l'API Riot est limité dans le temps à 50 requêtes par minute. Pour chaque partie jouée, nous récoltons des informations sur les 10 joueurs ainsi que sur leurs 5 parties précédentes, car celles-ci donnent de bonnes indications sur la future partie d'un joueur, ce qui nous limite à la récolte des données d'une partie par minute. En se partageant les requêtes, nous avons récolté les données sur 2887 parties jouées sur le serveur européen, ainsi que les informations des prés de 30.000 joueurs ayant participé à ces parties. Ces joueurs sont classés en très grande majorité Platine (top 10% des joueurs).

Certaines données ne sont pas accessibles directement par l'API. Par exemple, c'est le cas du nombre de parties jouées par un joueur sur un champion donné (on s'intéresse en particulier à celui qu'il a sélectionné lors de sa partie). Pour obtenir ces données, il faudrait regarder toutes les parties jouées par un joueur au cours d'une même saison, nombre qui peut être très élevé (de l'ordre de plusieurs centaines), ce qui est beaucoup trop coûteux en nombre de requêtes. Ce genre de données a déjà été calculée par l'un des nombreux sites offrant des résumés sur les parties d'un joueur, car ces sites se mettent à jour en continu en récupérant en direct le résultat de toutes les parties jouées dans le monde.

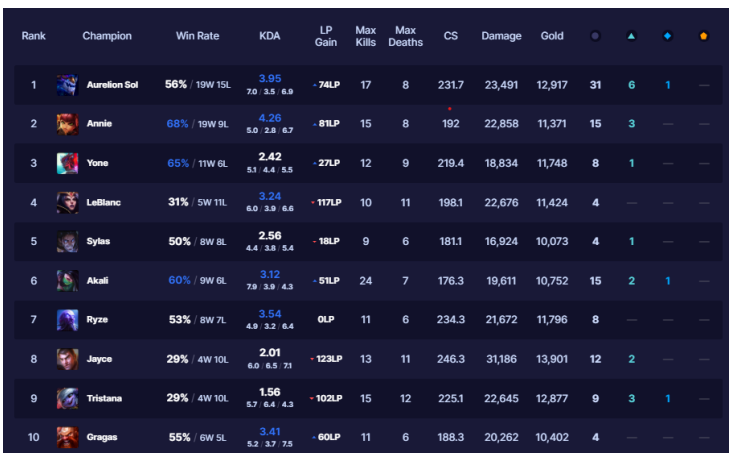


FIGURE 2 – Exemple d'interface du site `u.gg` pour un joueur donné

À l'aide de requêtes HTTP sur ce site et du traitement des données récoltées, nous avons pu compléter notre dataset avec plusieurs features très importante dans la prédiction du résultat final.

Finalement, chacun des 10 joueurs est décrit par 18 variables différentes, ce qui nous donne un total de 181 variables pour chaque partie en ajoutant le booléen Y qui donne l'équipe gagnante de la partie (bleue ou rouge).

Features	Signification de la donnée	Source	Type python
Y	Résultat de la partie	API (direct)	bool
CHAMP	Champion joué sur la partie	API (direct)	string
LVL	Niveau d'invocateur	API (direct)	int
TOTAL	Nombre de parties jouées sur la saison	API (direct)	int
GWR	Winrate en classé sur la saison	API (direct)	float
VET	Attribut vétéran du joueur	API (direct)	bool
RANK	Rang en classé solo/duo du joueur (elo)	API (direct)	list
HOT	Attribut "série de victoires du joueur"	API (direct)	bool
KDAG	KDA moyen sur les 5 dernières parties (tous champions)	API (5 games)	list
KDA	KDA moyen sur les 5 dernières parties (champion de la partie)	API (5 games)	list
WR	Winrate moyen sur les 5 dernières parties (champion de la partie)	API (5 games)	float
NB	Nombre de partie jouées sur le champion sélectionné parmis les 5 dernières	API (5 games)	int
FILL	Le joueur joue-t-il sur le même poste que ses 5 dernières parties (autofill)	API (5 games)	bool
VS	Winrate moyen sur le matchup entre les 2 champions d'un même poste	mobachampion.com	float
MAS	Niveau de maîtrise sur le champion joué	API (direct)	int
WRCH	Winrate de la saison sur le champion de la partie	scraping u.gg	float
WCH	Victoires de la saison sur le champion de la partie	scraping u.gg	int
LCH	Défaites de la saison sur le champion de la partie	scraping u.gg	int
TOTCH	Total de parties jouées dans la saison sur le champion de la partie	scraping u.gg	int

FIGURE 3 – Liste des features récupérées, par source et par type

3.4 Transformation et création de features

Utiliser les features récupérées directement est possible, mais il est préférable de les transformer pour qu'elles représentent mieux le problème que nous cherchons à résoudre. Ainsi, nous avons mis à profit notre expérience personnelle en tant que joueurs ou spectateurs de LoL en nous posant nous-même la question : " *Quelles informations étudions-nous, joueurs, quand on doit prédire le résultat d'une partie ?* "

Données non numériques Certaines données ne sont pas numériques. C'est le cas de la variable **RANK**, qui est de la forme [Division, Tier, Points], par exemple [Diamant, III,23]. Nous avons transformé cette liste en un *score elo* qui représente le niveau lors d'une partie compétitive du joueur. La liste précédente donne un elo de 2123 : 2000 points pour la division Diamant, 100 pour le tier III et 23 points supplémentaires.

Un autre exemple est le **KDA** (pour Kills/Deaths/Assists), qui est une liste de 3 entiers représentant le nombre de personnages tués, le nombre de morts et le nombre d'assistances. Une transformation souvent utilisée dans le jeu est de calculer la quantité $(K + A)/D$, qui représente la performance globale d'un joueur dans une partie.

Données non uniformément réparties En fonction du temps de jeu total des joueurs, il peut y avoir une grande variabilité des valeurs pour certaines features, avec beaucoup de valeurs très faibles pour la majorité des joueurs et quelques valeurs très élevées. Pour ces features, on prend alors le log pour avoir un résultat qui ressemble moins à une loi exponentielle, mais plutôt à une loi normale ou uniforme, qui sont traitées beaucoup plus efficacement par les algorithmes de machine learning. On a appliqué cette transformation pour les features **MAS**, **LVL** et **TOTAL**. Voici un exemple pour le nombre de parties jouées :

FIGURE 4 – Répartition du nombre de parties jouées par joueur avant et après log-transformation

Et voici un exemple pour les niveaux de maîtrise sur le champion joué :

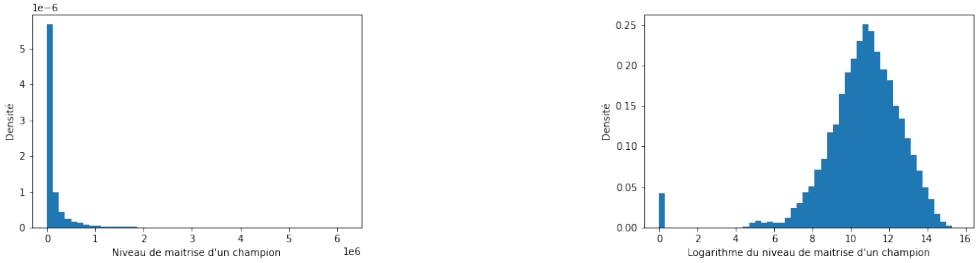


FIGURE 5 – Répartition des niveaux de maîtrise avant et après log-transformation

Concaténation de données par équipe Une des spécificités de notre dataset est que chaque feature est calculée pour chacun des 10 joueurs de la partie. Cela a la fois un coût important en termes de calcul, et une pertinence moindre, car il est plus difficile pour le modèle d'identifier une corrélation particulière avec le taux de victoire. De plus, les 5 rôles au sein d'une équipe sont relativement symmétriques, et on pourrait les utiliser conjointement plutôt que séparément. C'est pourquoi nous avons fait le choix de rassembler chaque variable au sein d'une même équipe en calculant le minimum, le maximum et la moyenne de la variable pour les 5 joueurs d'une même équipe. Cela permet de passer de 10 features à 6 pour chacune des variables, et de donner la même importance à tous les joueurs.

Normalisation On peut également normaliser les données, pour qu'elles soient de moyenne nulle et de variance 1. Cela est particulièrement utile lorsque l'on utilise des réseaux de neurones, mais inutile pour les modèles classiques.

Toutes les transformations présentées dans cette partie ont permis d'augmenter la performance des modèles que l'on va présenter plus loin dans le rapport.

3.5 Visualisation du dataset

Une fois le dataset constitué, et avant d'utiliser tout procédé de Machine Learning, les outils de statistiques et de visualisations classiques peuvent permettre une première analyse de nos données.

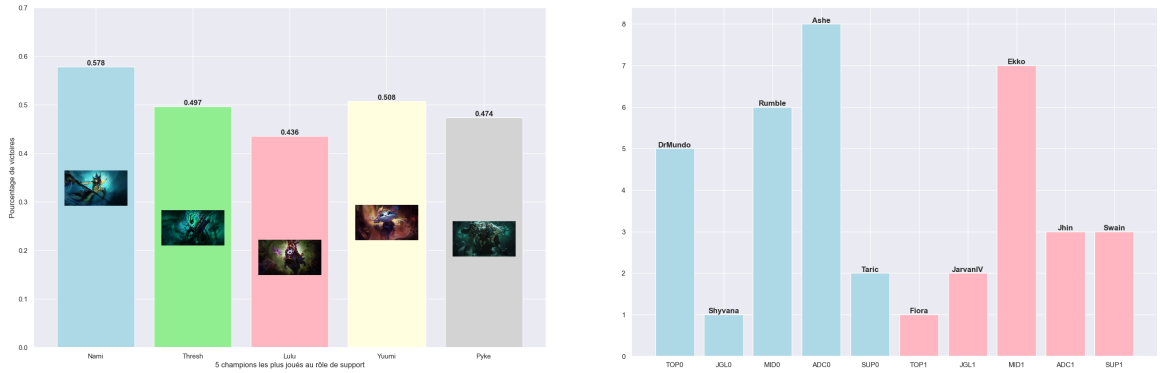
3.5.1 Statistiques et méta du dataset

On peut tirer de l'application des fonctions de bases de numpy quelques statistiques intéressantes sur le dataset :

- Il contient 2887 lignes i.e parties
- L'équipe bleue gagne en pourcentage 50,2% des parties
- Les parties récoltées ont eu lieu au rang platine (niveau intermédiaire)

On peut alors se demander quels sont les personnages les plus plébiscités par les joueurs et leur pourcentage de victoire associé.

Aussi, pour une partie donnée, au-delà de résultat, nous pouvons afficher qui a dominé la partie et à quel niveau en affichant les "kills" effectués par chaque personnage.



(a) Personnages les plus joués au rôle "Support" avec leur pourcentage de victoire

(b) Affichage des "kills" par personnage au cours d'un match

FIGURE 6 – Premiers affichages d'analyse du dataset

Ainsi, le joueur confirmé serait capable par ces simples visualisations de tirer des premières conclusions sur les personnages les plus forts et analyser a posteriori une partie.

3.5.2 Analyse de corrélation

Il est également possible de réaliser des analyses de corrélation sur ces données. Ce sont les multiples itérations de ces analyses qui ont notamment motivé la concaténation des features par équipes.

Initialement, nous avons comparé les variables deux à deux, afin de valider nos hypothèses sur l'influence de certains paramètres. Cela a permis de mettre en avant les éléments qui feront de "bonnes" features pour nos modèles.

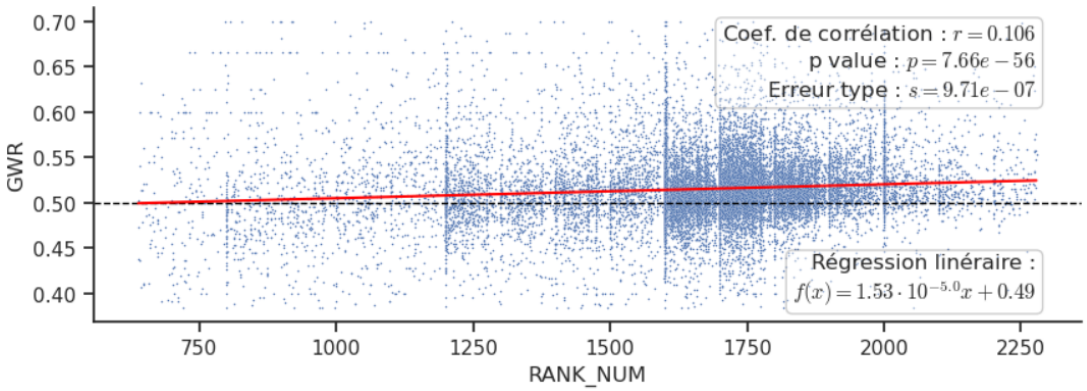
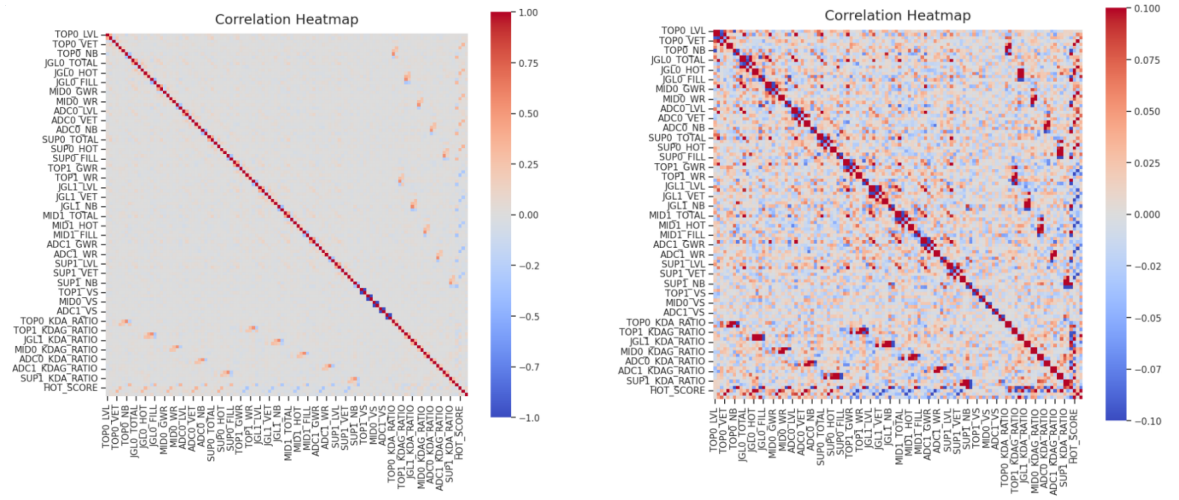


FIGURE 7 – Analyse de corrélation entre le rang d'un joueur et son taux de victoire

Si on observe l'exemple de la figure 7, on voit que l'on a cherché à étudier l'influence entre le classement d'un joueur (RANK_NUM) et son taux de victoire (GWR). On obtient un coefficient de corrélation $r = 0.106$, ce qui peut sembler être faible à première vue. Cependant, il faut se rappeler que la volatilité d'une partie de LoL est très importante, si bien que la plupart des informations disponibles avant le début de la partie ne permettent quasiment jamais de prédire avec confiance le résultat de la partie (ce qui est rassurant, on ne voudrait pas que la partie soit gagnée ou perdue avant même la première minute de jeu !). En conséquence, cette valeur du coefficient de corrélation est somme toute assez

forte (et cela est explicable : il n'est pas étonnant qu'il y ait une corrélation entre le niveau du joueur et le fait qu'il gagne plus souvent).

Afin d'avoir une vue d'ensemble de nos données, nous ne pouvons pas nous limiter à de la comparaison manuelle deux à deux. Ainsi, nous avons utilisé des outils comme la heatmap pour essayer de détecter la présence de motifs de corrélation. Et les résultats sont intéressants :



(a) Matrice de corrélation entre toutes les variables individuelles du jeu de données

(b) La même matrice avec une résolution plus importante

FIGURE 8 – Une autre forme d'analyse de corrélation : la heatmap

On peut se demander d'où proviennent ces motifs, notamment ceux que l'on aperçoit en dehors de la diagonale, même avec une large échelle de coloration. En réalité, ces données sont fortement corrélées, car elles proviennent du même joueur, et on les voit regroupées sous forme de blocs, car dans la version du jeu de données sans concaténation, les variables relatives à un même joueur sont adjacentes.

Afin d'obtenir un classement préalable de nos features, on peut réaliser une régression linéaire multiple, qui va calculer un coefficient de corrélation entre notre variable Y ("gagner la partie") et toutes nos autres features. Pour obtenir un résultat plus utile dans le cadre du Machine Learning, on utilise le modèle de régression LASSO, qui a la particularité d'être un modèle *sparse* (parcimonieux), avec un grand nombre de coefficients nuls, ce qui peut permettre de faire de la sélection de features.

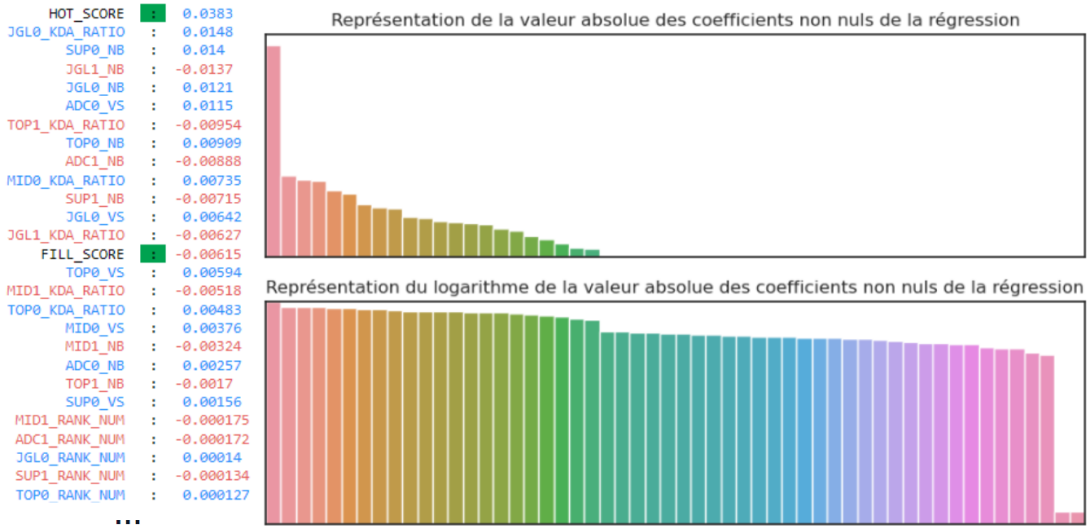


FIGURE 9 – Régression linéaire multiple (LASSO) avec notre jeu de données

Ces résultats permettent réellement de justifier de passer de données relatives au rôle à des données relatives à l'équipe entière. En effet, on observe une grande asymétrie dans la représentation des features relative aux différents rôles, mais aussi et surtout dans la représentation des features relatives aux différentes équipes. Par exemple : pour quelle raison le KDA (ratio valant $\frac{\text{kills+assists}}{\text{deaths}}$) du joueur "Jungler" de l'équipe bleue serait plus de deux fois plus important que celui du joueur "Jungler" de l'équipe rouge ? Les deux couleurs d'équipes, bien qu'apportant de très légères différences en termes d'expérience de jeu (en moyenne, le taux de victoire du côté bleu tourne autour des 50.5%) n'ont pas d'influence suffisante sur le déroulement de la partie pour justifier un tel décalage. Ainsi, en passant à des données d'équipes plutôt que de rôle, on évite ces biais apportés par notre échantillon relativement petit de parties.

4 Prédiction de victoire

Dans cette section, nous avons essayé de réaliser une classification binaire sur les données : prédire l'équipe gagnante pour chacune des parties. Nous avons visé un score de 70% de bonnes prédictions d'après les résultats d'autres études. Le déroulement d'une partie étant soumis à beaucoup d'évènements aléatoires, il est difficile d'obtenir de meilleurs résultats, même en ayant des informations complètes au début de la partie.

4.1 Modèle retenu

4.1.1 Démarche

Nous avons ajouté au fur et à mesure des nouvelles features, ce qui nous donne 4 versions de notre dataset. Pour chacun des datasets, nous avons testé plusieurs modèles de classification classiques (forêts aléatoires, gradient boosting) ainsi que des réseaux de neurones. Nos données sont essentiellement tabulaires et peu structurées, nous nous sommes concentrés sur les modèles classiques qui donnent de meilleurs résultats. Au cours de cette démarche, voici les différents leviers d'action que nous avons utilisé pour améliorer les performances :

- Ajouter des parties dans le dataset (ce qui conduit rapidement à un plafond sur les performances)
- Ajouter de nouvelles features, en utilisant nos connaissances et le travail des autres études citées dans l'état de l'art.
- Modifier la structure de notre dataset, en réalisant du feature engineering, par exemple en calculant de nouvelles variables à partir des précédentes (moyenne, passage en échelle log lorsque c'est pertinent)
- Tester un nouveau modèle.
- Faire de la sélection sur les hyperparamètres du modèle ou *tuning*.
- Réaliser du vote majoritaire sur plusieurs modèles construit différemment (par exemple prendre 3 modèles sur 3 datasets et choisir le vote le plus présent sur les 3).

Chacune des modifications a permis de gagner de petites performances sur les résultats. Dans tous les cas, le modèle le plus performant a été le gradient boosting de la librairie *XGBoost*.

4.1.2 Gradient Boosting et XGBoost

En machine learning, le *Boosting* est une méthode ensembliste : le principe est de combiner plusieurs modèles simples dont la combinaison à une performance globale meilleure que chacun des modèles individuellement. Les modèles sont construit successivement de la manière suivante : chaque nouveau modèle cherche à prédire correctement les données sur lesquelles les modèles précédents se sont trompés. Dans le *Gradient Boosting*, ce nouveau modèle est construit en minimisant l'erreur globale de tous les modèles par une descente de gradient. XGBoost est une version régularisée de ce principe (voir [6]), qui permet d'obtenir de meilleurs résultats en réduisant le sur-apprentissage. On utilise comme modèle de base des arbres de décisions CART. Ces modèles disposent de nombreux paramètres (profondeurs des arbres, nombre de valeurs par feuille, nombre de features utilisées) mais les paramètres par défauts donnent généralement les meilleurs résultats. Nous avons cher-

ché à optimiser directement les 2 principaux paramètres du gradient boosting : le nombre d'estimateurs et le *Learning rate*.

4.1.3 Tuning du modèle

Pour chercher si l'on peut améliorer le modèle, on réalise de la validation croisée avec 15 blocs pour différentes valeurs des paramètres. Voici ci-dessous les résultats en faisant varier **N** le nombre d'arbres de décision d'un modèle et **lr** le *learning rate*, ou taux d'apprentissage de la descente de gradient.

lr \ N	**100**	**200**	**500**
0.02	66.70 %	69.72 %	69.82 %
0.05	70.10 %	70.17 %	69.48 %
0.1	70.51 %	70.13 %	69.37 %
0.2	69.51 %	69.30 %	68.54 %

TABLE 1 – Efficacité d'un modèle XGBoost pour différents hyperparamètres

Les valeurs $lr = 0.1$ et $N = 100$ sont celles qui ont donné le meilleur score de 70.51% de bonnes prédictions sur le résultat de la partie. Nous avons conservé ces paramètres dans tout le reste du projet à chaque fois que nous avons fait appel à un modèle de Gradient Boosting.

4.2 Résultats obtenus

Notre modèle obtient une performance globale de 70.5% de bonnes prédictions, ce qui est cohérent avec le travail réalisé par d'autres auteurs et nos attentes au début du projet. On peut regarder en détail les probabilités d'appartenance à chaque classe au lieu de simplement regarder la classe la plus probable, ce qui donne une courbe de densité sur les probabilités. On a séparé cette courbe en 2 en fonction de la vraie classe.

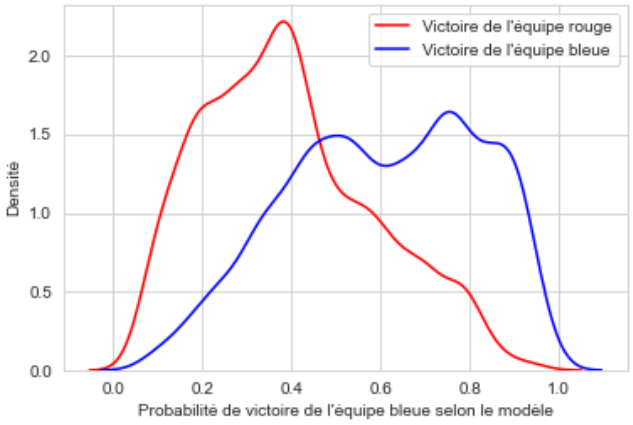


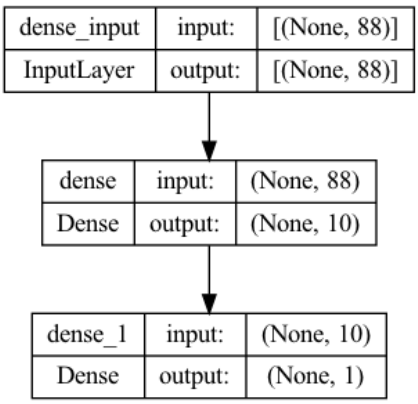
FIGURE 10 – Comparaison du vrai résultat en fonction de la probabilité donnée par le modèle

On voit bien que les probabilités calculées de victoire de l'équipe bleue ont tendance à être plus élevées si elle a réellement gagné, et inversement pour l'équipe rouge. Si le modèle nous donne une probabilité de victoire de 80% pour l'équipe bleue, il y a 3 fois plus de chances que l'équipe bleue gagne par rapport à la rouge. Au contraire, si le modèle prédit 20%, il y a 4 fois plus de chances que ce soit l'équipe rouge qui gagne. Nous avons donc optimisé les modèles au mieux de ce qui est possible, les pistes pour améliorer l'efficacité sont de rajouter plus de données ou d'essayer de trouver de nouvelles features utiles à la prédiction.

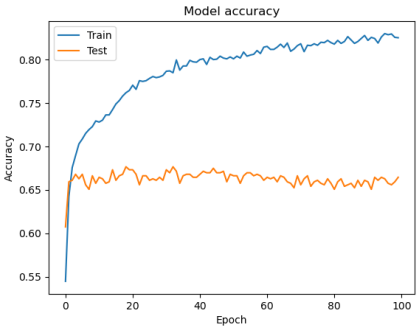
4.3 Tests de réseaux de neurones

Bien que les réseaux neuronaux ne soient pas réputés pour leurs résultats sur des données tabulaires tels que les nôtres, nous avons à des fins de comparaison entraînés des réseaux de neurones simples.

Après avoir standardisé nos features afin d'assurer la convergence du modèle, nous avons commencé par un modèle simple, à base d'une seule couche cachée et obtenus les résultats ci-après.



(a) Architecture du modèle



(b) Résultats obtenues

FIGURE 11 – Modèle à une couche cachée de 10 neurones et les résultats obtenus

Les résultats obtenus ne sont pas tant éloignés de ceux obtenus par XGBoost mais l'on peut être contrarié par l'overfitting [5] que semble présenter le modèle au vu de la diminution de la précision sur les données de train.

On va alors par la suite ajouter de la régularisation [6] afin de limiter l'overfitting.

Deux autres tests ont alors été réalisés :

— Le même modèle à une couche cachée en rajoutant de la régularisation l1 [7]
— Un modèle à trois couches cachées en "entonnoir", toujours avec de la régularisation l1

5. Tendance du modèle à perdre sa capacité de généralisation en "sur-apprenant" les données de train
6. Technique de généralisation de l'apprentissage
7. On diminue l'importance des poids trop importants qui peuvent bloquer l'évolution d'autres poids

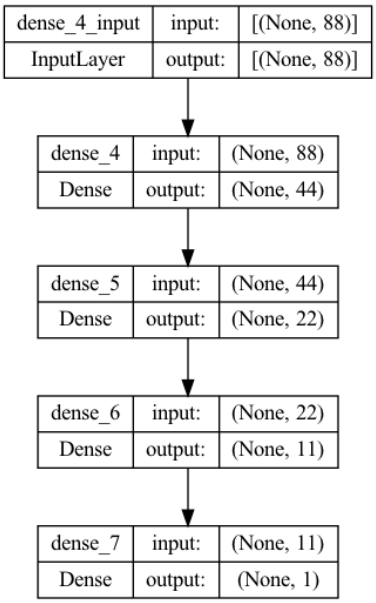
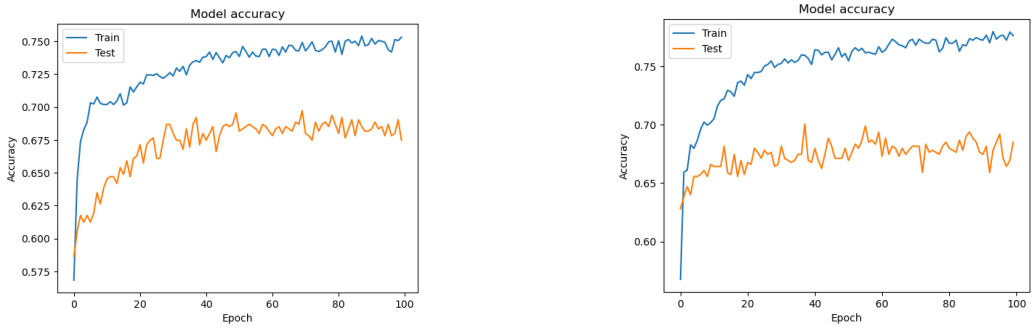


FIGURE 12 – Modèle en "entonnoir"



(a) Résultats du premier modèle avec ajout de la régularisation l1

(b) Résultats du modèle "entonnoir", avec régularisation l1

FIGURE 13 – Comparaison des résultats obtenus avec les deux modèles

De ces tests, nous pouvons tirer quelques conclusions quant à l'utilisation d'un réseau de neurone comme modèle :

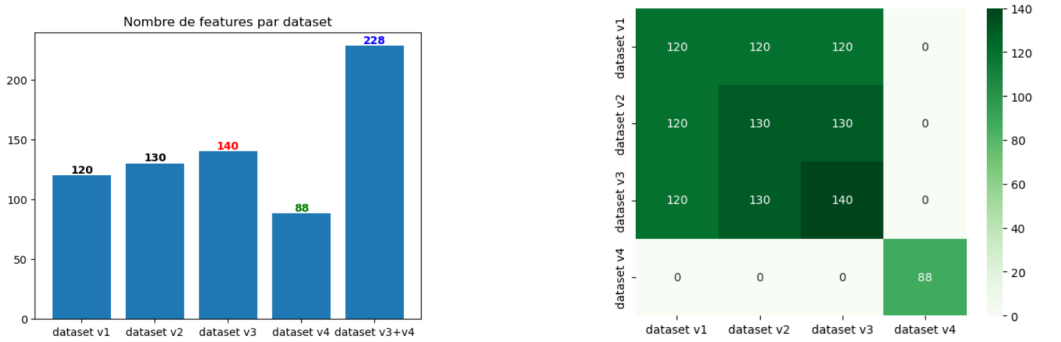
- La précision varie significativement entre deux epochs : le résultat que l'on essaie de prédire n'est pas entièrement déterministe
- La régularisation l1 améliore la précision et contre l'overfitting
- Le modèle en "entonnoir", plus complexe, n'améliore pas significativement les résultats

4.4 Comparaison des modèles

Nous pouvons désormais venir comparer les différents modèles qui ont été créés au cours du projet pour la prédiction de victoires, afin de pouvoir visualiser les améliorations qui ont été réalisées.

4.4.1 Comparaison des features

Pour commencer, il est intéressant de venir observer le nombre de features de chacun des datasets créés. On remarque alors une forte diminution, de presque un facteur 2, entre la troisième et la quatrième versions des datasets. Cela est dû à une refonte complète des features : au lieu de venir les calculer pour chaque joueur (donc pour chaque poste), on vient les calculer de manière globale pour chaque équipe en faisant des moyennes, des maximums et des minimums. De plus, les trois premiers datasets ont été créés sur le même principe : les datasets n'ont subi que des ajouts de colonnes jusqu'au v3 (c'est-à-dire que toutes les autres features sont les mêmes). Le dataset v4 ne possède donc aucune correspondance avec les autres features des autres modèles, car il s'agit d'un remaniement complet de celles-ci.



(a) Nombre de features par modèle (b) Corrélation des features des modèles

FIGURE 14 – Comparaison des features des modèles

Ensuite, l'analyse de la Feature Importance de chaque modèle nous renseigne un peu plus sur leurs fonctionnements. Cette Feature Importance est calculée en prenant le nombre de fois ou apparait une features dans les choix des arbres de XGBoost, que l'on réécrit ensuite en pourcentage ou 100% correspond à la feature à la plus utilisée et 0% celle la moins utilisée :

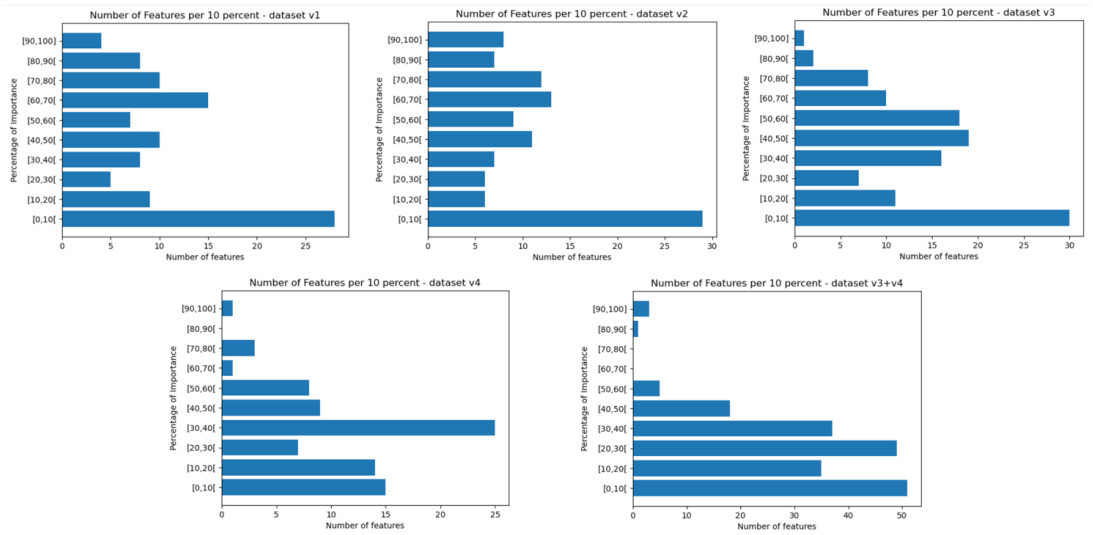


FIGURE 15 – Feature Importance des modèles

Bien que tous les graphiques soient similaires du point de vue de leurs formes, il existe des différences notoires dans l'utilisation des features. En effet, les datasets v1, v2, v3 utilise tous les trois très peu entre 25 et 30 features, mais on remarque qu'une seule et unique feature est très utilisé pour le dataset v3, malgré le fait que seules dix d'entre elles soient nouvelles par rapport à la v2.

La même observation peut-être faite pour le dataset v4, qui après le remaniement de la v3 voit une seule variable être très utilisée. On observe même ici un delta assez conséquent d'utilisation entre la feature la plus utilisée et celles qui suivent. Ce delta dans lequel aucune features ne se trouve est encore plus visible dans le dataset composé de la somme du v3 et du v4.

Comme dit précédemment, les trois premières versions du dataset sont très similaires, mais ne semblent pas utiliser les features de la même manière. Afin de vérifier cette hypothèse, on construit un graphique de l'évolution de l'importance des features entre les trois premières versions du dataset. On observe alors d'importantes variations dans l'utilisation des features.

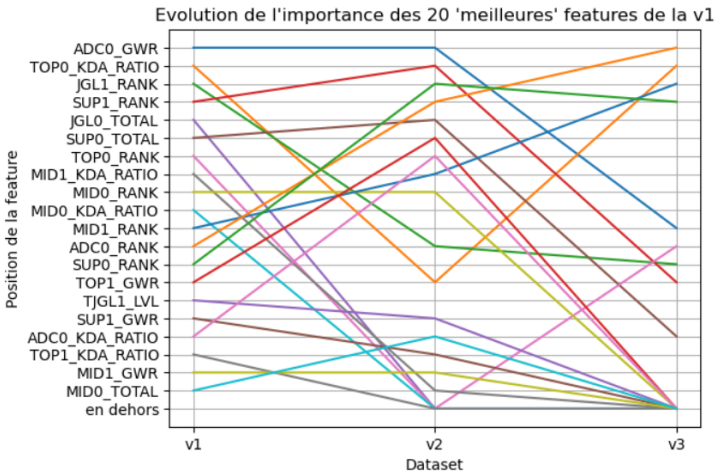
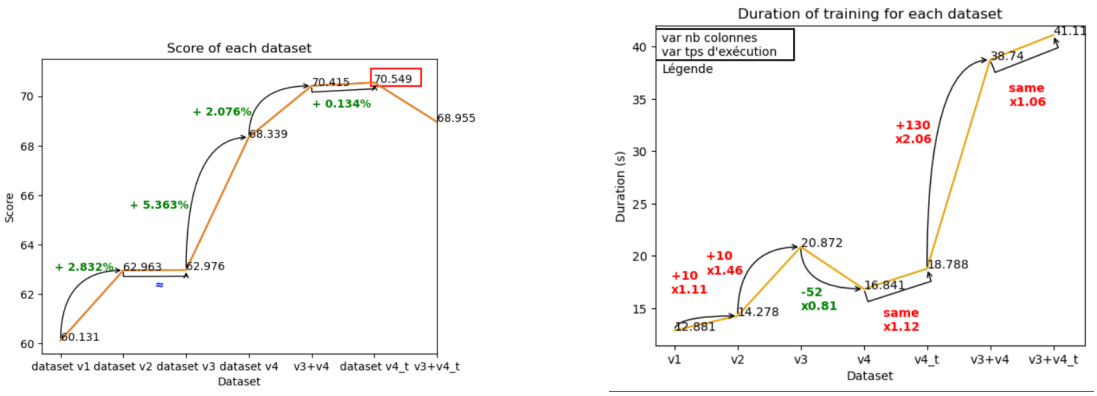


FIGURE 16 – Évolution de l'importance des 20 "meilleures" features du premier dataset

Pour terminer cette comparaison des datasets, on peut venir directement comparer les scores des différents modèles (qui correspondent au pourcentage de bonnes prédictions) ainsi que les temps de calcul nécessaires à leur entrainement. On vient rajouter aux précédents datasets, une version *transformée* du dataset v4.



(a) Score des différents datasets

(b) Temps d'entrainement des datasets

FIGURE 17 – Comparaison de l'efficacité des datasets

La comparaison de ceux deux graphiques permet de faire apparaître certaines informations jusque-là inconnues. En effet, on remarque que le dataset v3, qui n'est qu'une version du dataset v2 avec 10 colonnes supplémentaires, obtient un score sensiblement égal à celui de la deuxième version. Cependant, bien que les scores soient similaires, le temps de train a lui été augmenté de presque 50%, ce qui vient mettre en évidence une forte différence d'utilisation des features entre ces deux modèles. À l'inverse, le passage de la troisième à la quatrième version à rendu bien plus efficace le modèle. Le temps de calcul a diminué de près de 20% et le score a lui augmenté de plus de 5%. Pour terminer, la version modifiée du dataset v4 donne presque le même score que la somme du v3 et du v4 (il est même un peu meilleur), mais est surtout bien plus rapide (plus de 2 fois plus rapide).

5 Création d'un outil client utilisable

Une fois que les modèles de prédiction de résultats ont dépassé les 70% de bonnes prédictions, nous avions atteint notre objectif en termes de performances, fixé dès le début du projet. Cependant, la perspective de créer un outil client utilisable en conditions réelle nous a inciter à emmener le projet dans cette direction. Ainsi, nous avons envisagé l'intégration de notre modèle de prédiction dans une pipeline complète, qui permettrait à un utilisateur tiers de recevoir des conseils ou des recommandation directement à partir du client League of Legends, sans avoir à interagir avec le moindre code Python.

Ce nouvel objectif définit alors trois grands axes :

— La récupération des données depuis le client LoL.
— L'utilisation de ces données pour proposer une plus-value à l'utilisateur
— La création d'une interface utilisateur lisible et facile d'utilisation.

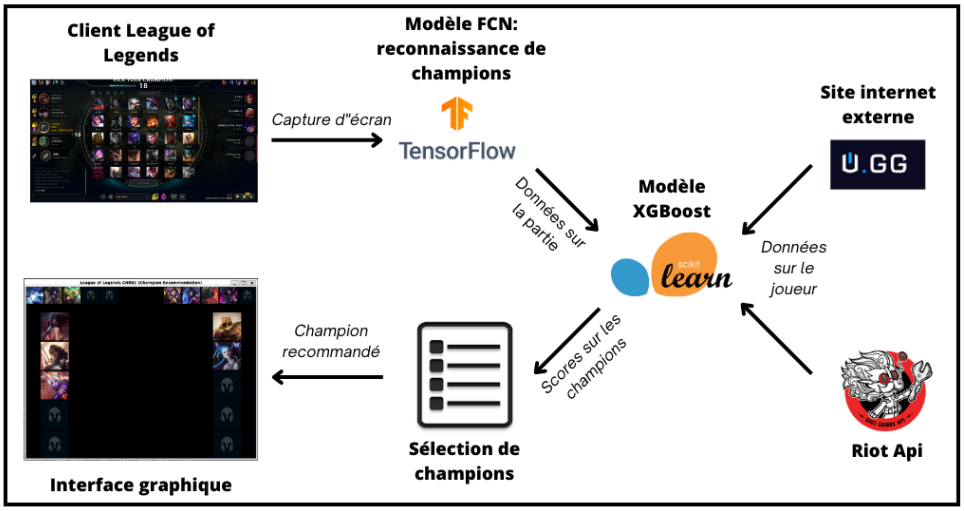


FIGURE 18 – Pipeline de notre outil de recommandation de champion

5.1 Vision par ordinateur : repérage des champions lors de draft

L'objectif ici est de réussir à détecter automatiquement les images des champions lors de leur sélection, afin de pouvoir en suggérer un pour le joueur lorsque c'est son tour de choisir. Cette détection peut se décomposer en deux étapes distinctes :

— La détection des emplacements des images
— La reconnaissance des champions eux-mêmes à l'aide d'algorithme de Machine Learning.

5.1.1 Détection des icônes des champions

Pour cette première étape, nous cherchons à détecter les emplacements des images de champions. Initialement, il était prévu de créer un algorithme de Deep Learning pour les détecter automatiquement. Cependant, nous sommes parvenus à une solution plus simple et moins coûteuse en temps d'implémentation et de calcul. La détection des images se fait alors de la manière suivante : les différentes images (que ce soit les bans ou les picks des joueurs) se situent tout le temps dans les mêmes zones du client de League of Legends. Nous utilisons alors les proportions de cette application pour directement déterminer des bandes contenant les images. Les bans sont tout le temps au même emplacement des deux côtés de l'image. Cependant, ce n'est pas exactement les cas des champions choisis par les joueurs. En effet, ceux-ci peuvent être décalés vers la gauche ou la droite en fonction de divers paramètres. Il a alors été choisi de commencer par récupérer les champions sur les deux bandes verticales où les images pourraient se trouver.

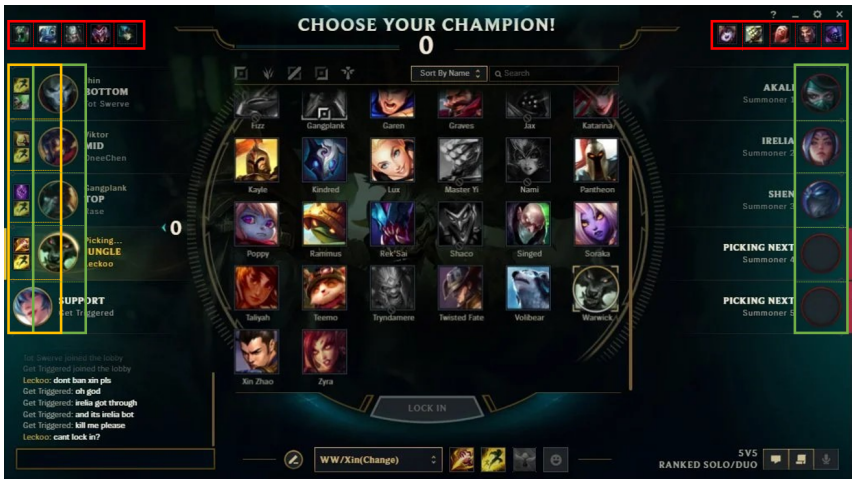


FIGURE 19 – Zones récupérées (en vert, jaune et rouge)

Les zones jaunes et vertes correspondent aux picks et les rouges aux bans. Une fois cette étape passée, on utilise pour les picks un algorithme utilisant OpenCV pour détecter les cercles dans l'image et ainsi conclure sur où est chaque image.

5.1.2 Reconnaissance des champions

Une fois les images récupérées, il est nécessaire de détecter les champions qui ont été sélectionnés. Pour cela, on utilise un réseau de neurones de 23 couches de type FCN (Fully Convolutional Network). Ce type de réseau permet d'effectuer de la reconnaissance d'images de la même manière que les CNN (Convolutional Neural Network), mais ils permettent également de prendre en entrée des images de tailles différentes, ce que ne permettent pas les CNN.

Afin d'entrainer ce modèle le mieux possible, les données fournies par le site de League of Legends ne suffisaient pas, en effet celles-ci étaient de bonnes qualités, très détaillées et centrées sur l'ensemble du personnage (comme un portrait de celui-ci), alors que les images détectées sur le client sont de basses qualités (les images des bans font environ 30x30 pixels) et centrées exclusivement sur les visages des personnages. De plus, les images détectées ne sont pas exactement centrées, car l'algorithme de détection ne fonctionnant qu'avec des ratios de valeurs, il est fortement probable que l'image réelle ne soit pas parfaitement au milieu de la zone de détection.

Nous avons alors effectué de la Data Augmentation sur les images fournies par le site. Ce processus revient à augmenter le nombre de données en appliquant des transformations sur les images. Ainsi, nous avons pu passer d'une image par champions à presque 200. Les traitements appliqués sont les suivants :

- Variation de qualité (diminution du nombre de pixels pour correspondre aux picks et aux bans)
- Variation des zones de l'image : on rogne l'image est des endroits un peu différents pour que la tête du personnage ne soit pas tout le temps centrée
- Variation de luminosité : tous les écrans n'ayant pas la même luminosité, il est important de faire varier la luminosité des images du dataset afin de détecter les champions de la meilleure manière possible
- Variation du nombre de pixels par effet de *blur*

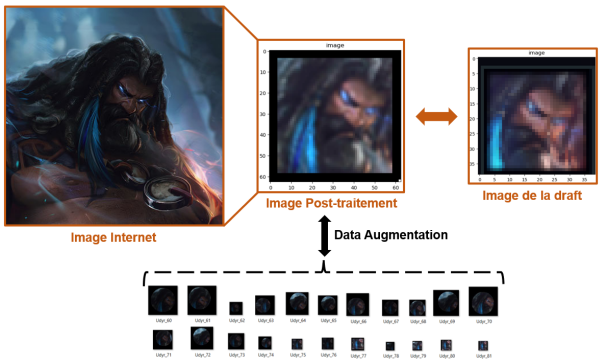


FIGURE 20 – Data Augmentation sur les images

On obtient finalement notre dataset en passant de 162 images à plus de 28 000. On repartit les images de manière à ce qu'environ 20% soit pour le test du modèle et 80% pour le train. De plus, on répartit les données de manière à ce qu'il y ait approximativement autant d'images ressemblant aux picks qu'aux bans. On entraîne le FCN sur 4 epochs. Afin d'estimer les capacitées du modèle, on choisit de regarder la proportion de bonnes prédictions ainsi que la proportion de bonnes prédictions se situant dans le top 5 : c'est-à-dire que le champion que le modèle aurait dû prédire pour une image se situe dans les 5 champions ayant les probabilités les plus fortes d'être celui de l'image d'après les calculs du modèle (on ne récupère donc pas seulement le champion prédit, mais les 5 plus probables champions). On obtient alors comme statistiques sur le modèle :

```
Prédictions des "picks" :
top 1 12/26 moyenne : 46.15%
top 5 19/26 moyenne : 73.08%

Prédictions des bannissements :
top 1 15/29 moyenne : 51,72%
top 5 21/29   moyenne : 72,41%
```

FIGURE 21 – Statistiques de prédictions du modèle final

On observe que presque une fois sur deux, le modèle sort directement le bon champion. De plus, d'après les résultats obtenus, le modèle semble légèremebt plus à l'aise avec les images des bans qui pourtant sont celles avec la plus petite définition. Globalement, on a quasiment trois fois sur quatre le bon champion dans le top 5 des prédictions.

Nous avions essayé de lancer un autre modèle, basé sur l'architecture de Resnet. Cependant, les résultats étaient bien moins probants, puisque qu'aucun pourcentage ne dépassait 13,5%, que ce soit pour les top1 où même les top5. Plus visuellement, voici ce qu'il est possible d'obtenir à la fin pour une prédiction :

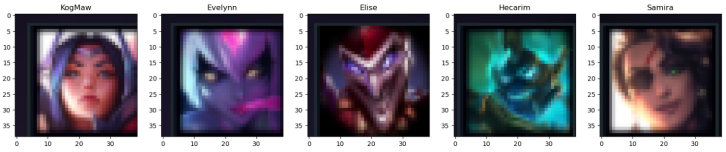


FIGURE 22 – Exemple de prédiction pour les picks de l'équipe 1

Les noms des prédictions sont marqués au-dessus des images des champions choisis. Sur l'exemple, trois champions sont bien prédits :Evelynn, Hecarim et Samira. Les prédictions sont également renvoyées sous forme d'un dictionnaire de listes afin de pouvoir les utiliser dans d'autres parties du code.

5.1.3 Transfer Learning : Utilisation de modèle ResNet

Commençons par définir le Transfer Learning dans le cas du Deep Learning. Il s'agit d'une méthode ayant pour but de réutiliser un réseau de neurones déjà entraîné pour une certaine tâche afin d'en réaliser un autre. Concrètement, il s'agit d'entraîner à nouveau une petite partie du réseau (les dernières couches) en fixant les autres. Si nécessaire, on modifie la couche finale pour avoir la sortie voulue.

Dans notre cas, les CNN étant relativement longs à entraîner et nécessitant un grand nombre de données afin d'être correctement entraînés, l'idée est d'utiliser une architecture déjà existante et fonctionnement bien pour des images de tous types et de l'adapter à la reconnaissance de champions.

Notre choix s'est alors porté sur le modèle Resnet50, notamment disponible via keras qui est un des plus populaires réseaux convolutionnels disponibles.

Resnet50 a été entraîné sur la dataset ImageNet en résolution 224×224 qui contient plus de 14 millions d'images labelisées.

Resnet50 prend son nom du fait qu'il est composé de 50 couches cachées et l'architecture est donnée ci-après.

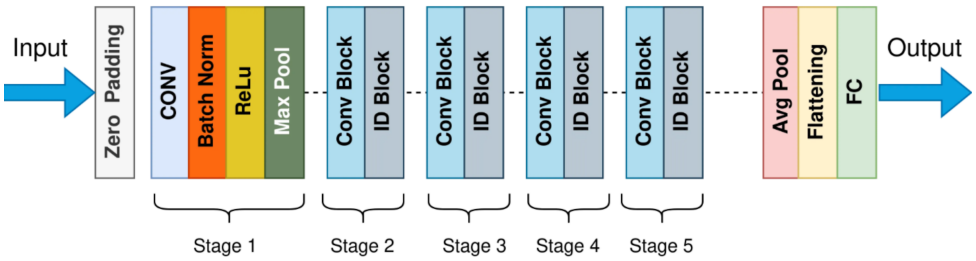


FIGURE 23 – Architecture de resnet50

Nous avons alors rajouté une couche finale au modèle consistant en 162 neurones (nombre de champions) et en un softmax afin d'avoir une probabilité d'appartenance à chaque classe. Aussi, nous avons "dégelé" les deux couches précédentes afin de permettre au modèle d'être plus précis et d'avoir plus de possibilités d'apprendre.

Nous avons alors entraîné le réseau sur le dataset présenté précédemment, utilisé pour entraîner le FCN. Malheureusement, même si le modèle converge et obtient des très bons résultats sur le dataset de test, ceux obtenus avec les images récoltées depuis l'écran client sont assez mauvais.

En effet, sur les tests réalisés, seulement 11% des prédictions étaient correctes et le modèle avait tendances à prédire trop souvent un même champion.

Malgré des tentatives pour tenter d'améliorer les performances du modèle, nous n'avons pas réussi à atteindre des résultats satisfaisants.

Une hypothèse est que ResNet50 a été entraîné sur des millions d'images très génériques, trop éloignées des icônes de champions de l'on voulait lui faire reconnaître. Aussi, les images que l'on récupère de l'écran de la draft ont des dimensions bien inférieures à

l'entrée 224×224 de resnet (généralement entre 30×30 et 50×50) et le redimensionnement de ces images a pu nuire aux performances.

5.2 Modèles de prédictions : suggestion de champions

Dans cette partie, l'objectif est de réintégrer les données, les modèles et les méthodes étudiées dans les parties 3 et 4 pour pouvoir construire un modèle de recommandation de champion pour les joueurs. Plusieurs outils ont déjà été proposés, et certaines études ont été réalisées pour tester différentes structures de modèles ([7], [8]). On peut choisir plusieurs critères de sélection : recommander un champion qui peut plaire au joueur, un nouveau champion ressemblant à ceux qu'il a déjà joués, ou encore celui que choisissent souvent les joueurs dans sa situation. Nous avons de prendre comme critère d'essayer de proposer le champion qui donne le plus de chances de gagner la partie pour garder un esprit compétitif propre au jeu, car c'est souvent ce que préfèrent également les joueurs. La phase de sélection des champions, qui précède la partie, est appelée *draft* : chacun des 10 joueurs sélectionne un champion l'un après l'autre, et l'on peut voir les champions déjà sélectionnés par ses alliés et adversaires avant de faire son choix. Pour réutiliser le travail déjà effectué, nous avons décidé d'adopter l'approche suivante :

— Récolte des champions joués par un joueur dans la saison par *webscrapping*.
— Pour chaque champion, on rassemble toutes les données pour créer une partie fictive.
— On calcule la probabilité de victoire à l'aide d'un modèle XGBoost pur chaque partie fictive.
— On recommande finalement les champions donnant les plus grandes chances de victoire au joueur.

Tous les joueurs ne disposent pas du même niveau d'information avant de faire leur sélection. Le joueur qui est *first pick*, c'est-à-dire qui choisit en premier, ne connait aucun des autres champions joués, alors que le *last pick*, qui sélectionne en dernier, dispose des informations sur tous ses alliés et ses adversaires. On peut donc utiliser un modèle plus performant sur le dernier que sur le premier. C'est pour cela que nous avons reconstruit 10 modèles XGBoost, un par position, pour pouvoir affiner notre prédiction. Pour chaque position, on tronque les données pour ne garder que celles dont disposerait réellement un joueur à cette place. Par aillers, certaines données ne sont plus disponibles, en particulier les pseudos des joueurs, qui ont été rendus anonymes par l'éditeur du jeu au cours de notre projet. On garde donc uniquement les données du joueur auquel on veut recommander un champion (que l'on connait), ainsi que les rôles et les champions des joueurs ayant déjà fait leur choix. Voici l'efficacité des 10 modèles en fonction de la position sur les données tronquées (les hyperparamètres et le découpage train/test utilisés en partie 4 restent identiques) :

Chacun des modèles dispose de 12 features propres au joueur : 6 constantes, comme le niveau du joueur ou son nombre de parties jouées, et 6 variables qui dépendent du champion joué (taux de victoire sur le champion, pourcentage de victoire contre le champion ennemi). On rajoute en plus deux variables à chaque position, le champion et le rôle des joueurs ayant déjà sélectionné leur champion. On conserve de bons résultats malgré le fait que les features importantes ont presque été divisés d'un facteur 10.

Comme on a pu le voir en partie 4, le modèle renvoie une liste d'estimateurs pondérés par un poids. Pour choisir la classe à renvoyer à partir d'un échantillon, on fait voter

Position	Score	Nombre de features
1	55.40 %	12
2	55.23 %	14
3	55.82 %	16
4	57.13 %	18
5	54.71 %	20
6	57.03 %	22
7	55.50 %	24
8	55.82 %	26
9	55.54 %	28
10	57.34 %	30

TABLE 2 – Taux de bonnes prédictions des modèles pour chaque ordre de sélection

chacun des estimateurs, chaque vote étant pondéré par son poids. Finalement, la classe ayant reçu le plus grand score est sélectionné, ce qui donne l'une des deux classes. Plutôt que de regarder la classe, on peut regarder les votes des différents estimateurs, ce qui donne une probabilité d'appartenance à l'une des deux classes. On peut utiliser cette valeur comme un score. En itérant sur plusieurs champions, qui vont donner plusieurs données d'entrée différente au modèle, on peut avoir un score pour chaque champion, et renvoyer celui qui donne la plus grande probabilité de victoire au joueur.

Voici ci-dessous un exemple de l'utilisation de l'algorithme sur une partie fictive et un joueur choisi au hasard : on a représenté les 15 champions joués par ce joueur lors de la saison courante, le score retourné par notre modèle et les 6 variables qui changent en fonction du champion (respectivement le taux de victoire du champion contre celui de l'adversaire, le nombre de points de maîtrise qui représentent l'expérience du joueur sur ce champion, ainsi que son nombre de victoires, de défaites, son pourcentage de victoire et son nombre total de parties pour chaque champion).

On remarque que les 5 champions ayant le plus gros score sont les 5 pour lesquels le joueur a le plus de parties jouées et le plus de victoires. Une analyse qualitative de ce modèle a été réalisée sur plusieurs parties et plusieurs joueurs de notre base de données, ainsi que sur de nouveaux joueurs de niveaux différents, et on a pu vérifier que les résultats étaient cohérents d'après notre expérience du jeu et de notre connaissance des données.

Champion	Score	VS	MAS	WCH	LCH	WRCH	TOTCH
Kha'zix	61.3 %	50.9	1.558.523	168	141	54.4 %	309
Jarvan IV	67.7 %	49.5	1.085.043	145	110	56.9 %	255
Diana	64.8 %	50.9	118.140	83	75	52.5 %	158
Elise	49.5 %	49.0	1.493.950	85	60	58.6 %	145
Shyvana	57.8 %	53.8	521.774	61	62	49.6 %	123
Nocturne	38.9 %	51.1	429.298	45	28	61.6 %	73
Karma	26.9 %	50.7	104.578	26	27	49.1 %	53
Karthus	29.6 %	50.5	86.640	18	19	48.6 %	37
Wukong	36.5 %	50.6	53.799	7	12	36.8 %	19
Viego	30.1 %	51.0	9.762	6	7	46.2 %	13
Ezreal	37.7 %	49.4	284.317	6	4	60.0 %	10
Kayn	24.9 %	52.7	75.494	3	5	37.5 %	8
Bel'veth	22.5 %	54.5	6.151	4	3	57.1 %	7
Varus	35.5 %	51.9	35.544	4	2	66.7 %	6
Dr Mundo	12.4 %	54.7	49.818	1	1	50.0 %	2

TABLE 3 – Résultats de l'algorithme pour une partie fictive et différents champions

5.3 Création d'une interface graphique

Dans le but de fournir un outil agréable à utiliser, la qualité de l'interface graphique est un critère majeur. Ainsi il a fallu apprendre à utiliser une bibliothèque Python permettant la conception d'une interface utilisateur élégante mais qui reste suffisamment simple d'utilisation afin de pouvoir terminer cette interface graphique dans les temps.

La bibliothèque *Kivy* fait partie des bibliothèques qui proposent un tel compromis, ce qui en fait une des bibliothèques GUI les plus populaires, notamment auprès de la communauté smartphone, de par la possibilité de créer un code compatible avec quasiment toutes les plateformes. Elle repose sur un système de widgets à positionner les un dans les autres, ce qui donne à notre application une structure d'arbre, ce qui est utile pour la gestion d'événements complexes. Il existe de nombreux widgets prédéfinis, implémentés sous forme de classes (label, image, bouton, boîte de texte, etc.), mais il est toujours possible de créer soi-même des widgets héritant des propriétés des widgets prédéfinis (exemple : on peut créer une classe *CustomButton* fille de la classe *Button* pour donner une personnalisation commune à tous les boutons de cette classe). De plus, il est possible d'associer une grande quantité de callbacks aux widgets, qui peuvent être déclenché par une horloge, des clics, des mouvements de souris, des entrées clavier, et d'autres encore. L'utilisation de notre interface graphique se fait de la manière suivante :

(a) Prendre une capture d'écran de la draft

(b) Ouvrir l'application puis entrer son pseudo

(c) Indiquer qui a commencé à choisir en premier

(d) Indiquer son rôle

(e) Indiquer sa position dans la draft

(f) Entrer le chemin relatif de la capture d'écran

(g) Les champions choisis et bannis sont reconnus depuis la capture

(h) Si besoin, on peut modifier les résultats de la prédiction pour corriger les erreurs

(i) Recommandation de champions avec score associé

FIGURE 24 – Les différentes étapes de l'utilisation de l'interface

6 Conclusion et perspectives

Au cours de ce projet, nous avons pu utiliser de nombreuses méthodes différentes de Data Science. Nous avons utilisé du Machine Learning sur des données tabulaires pour la prédiction de victoires (avec une forte utilisation de feature engineering), avons effectué de la Computer Vision ainsi que de la Data Augmentation pour la détection et la reconnaissance des champions, et avons utilisé des réseaux de neurones tout au long du projet. Finalement, nous avons créé un programme capable, en ne prenant en entrée qu'une image du client de League of Legends, de récupérer les données nécessaires, prédire des victoires ou des défaites en fonction des choix de personnages s'offrant au joueur, et enfin lui fournir une recommandation de manière à optimiser ses chances de victoire. L'interface graphique, en tant que pipeline globale du projet, permet de lier ces étapes et de les enchaîner de manière fluide.

Nous avons rencontré certaines difficultés au cours du projet, notamment au cours du travail sur la reconnaissance d'images. Il a été difficile de réussir à créer un dataset à la fois assez semblable aux images disponibles lors de la sélection des champions, sans pour autant tomber dans de l'overfitting. Il est sûrement possible d'améliorer les performances en retravaillant de manière encore plus approfondie la création du dataset. De plus, ne disposant pas d'ordinateurs très puissants, l'entraînement des réseaux de neurones a pu prendre un temps non négligeable.

Afin de poursuivre ce projet, il aurait pu être intéressant d'effectuer du Transfer Learning, c'est-à-dire adapter nos programmes pour l'utilisation de données de jeux similaires dans lesquels on retrouve des concepts proches de ceux LoL comme Dota 2 (un autre MOBA) ou encore Valorant (FPS) qui utilise la même API que League of Legends.

Pour conclure, nous considérons ce projet comme une réussite et sommes heureux d'avoir pu le mener à bien, depuis la prise en main de l'API à la conception d'un outil graphique utilisable. Nous sommes confiants dans l'idée qu'il est possible de se baser sur ce projet pour créer une application utilisable par n'importe quel joueur au même titre que les applications tierces déjà existantes. Il faudrait néanmoins, entre autre, réentraîner régulièrement les modèles en actualisant les données à la dernière version de League of Legends afin de suivre la méta ainsi que l'ajout de nouveaux champions.

FIGURE 25 – Le meilleur joueur de l'histoire de League of Legends : Faker

Références

[1] Tiffany D. Do, Seong Ioi Wang, Dylan S. Yu, Matthew G. McMillian, and Ryan P. McMahan. Using machine learning to predict game outcomes based on player-champion experience in league of legends. *CoRR*, abs/2108.02799, 2021.

[2] Lincoln Magalhães Costa, Rafael Gomes Mantovani, Francisco Carlos Monteiro Souza, and Geraldo Xexéo. Feature analysis to league of legends victory prediction on the picks and bans phase. pages 01–05, 2021.

[3] Victoria J. Hodge, Sam Devlin, Nick Sephton, Florian Block, Peter I. Cowling, and Anders Drachen. Win prediction in multiplayer esports : Live professional match prediction. *IEEE Transactions on Games*, 13(4) :368–379, 2021.

[4] Kodirjon Akhmedov and Anh Huy Phan. Machine learning models for DOTA 2 outcomes prediction. *CoRR*, abs/2106.01782, 2021.

[5] Dong-Hee Kim, Changwoo Lee, and Ki-Seok Chung. A confidence-calibrated MOBA game winner predictor. *CoRR*, abs/2006.15521, 2020.

[6] Tianqi Chen and Carlos Guestrin. Xgboost : A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[7] Tiffany D. Do, Dylan S. Yu, Salman Anwer, and Seong Ioi Wang. Using collaborative filtering to recommend champions in league of legends. *CoRR*, abs/2006.10191, 2020.

[8] Hojoon Lee, Dongyoon Hwang, Hyunseung Kim, Byungkun Lee, and Jaegul Choo. DraftRec : Personalized draft recommendation for winning in multi-player online battle arena games. In *Proceedings of the ACM Web Conference 2022*. ACM, apr 2022.