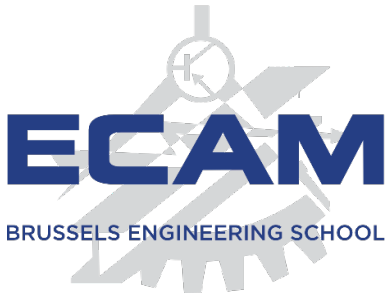

2025

AI LAB : League of Legends player classification

DUFOUR Arthur 18194

JARDINET Louis 195030

Table of Contents

1. Introduction

Ce projet vise à développer un modèle prédictif capable de déterminer, à partir des statistiques individuelles d'un joueur, si son équipe a remporté ou perdu la partie. Au-delà de la simple prédiction, notre approche permet également d'évaluer l'impact de chaque joueur sur le résultat final et de créer un système de classement des joueurs basé sur leurs performances, ainsi que de déterminer sur quelle statistique se concentrer dans le but d'obtenir une victoire.

2. Exploration des données utilisées

2.1 Différents Datasets

L'analyse s'appuie sur trois ensembles de données principaux :

- game_metadata.csv : Contient les informations générales sur les matchs (identifiants, dates, tournois, etc.)
- game_events.csv : Enregistre les événements importants durant les parties (éliminations, objectifs, etc.)
- game_players_stats.csv : Compile les statistiques individuelles des joueurs (kills, deaths, assists, farm, etc.)

Afin de déterminer le score des joueurs sur les résultats, notre implémentation s'est essentiellement concentrée sur le dataset lié aux statistiques des joueurs, les 2 autres étant non pertinents.

2.2 Analyse du dataset

L'analyse du dataset va nous permettre de correctement établir les critères amenant à une victoire.

Pour commencer, il est important de vérifier dans la distribution des rôles si ceux-ci possèdent le même taux de victoire. Nous calculons la moyenne de la colonne « win » pour chaque rôle.

```
Taux de victoire par rôle:
role
Bot        0.499987
Jungle     0.500000
Mid        0.500000
Support    0.499987
Top        0.500000
Name: win, dtype: float64
```

Figure 1 - Taux de victoire par rôle

Le taux de victoire étant presque identique, le choix du rôle ne sera pas un caractère déterminant pour une victoire. Ce qui nous semble tout à fait logique, les parties se jouant toujours avec les 5 rôles présents, il y aura toujours 1 Top gagnant et 1 Top perdant, par exemple. Cela nous permet néanmoins de vérifier que le dataset est complet en fonction des rôles. En revanche, le rôle choisi va impacter certaines statistiques comme nous pouvons le voir à la figure 3.

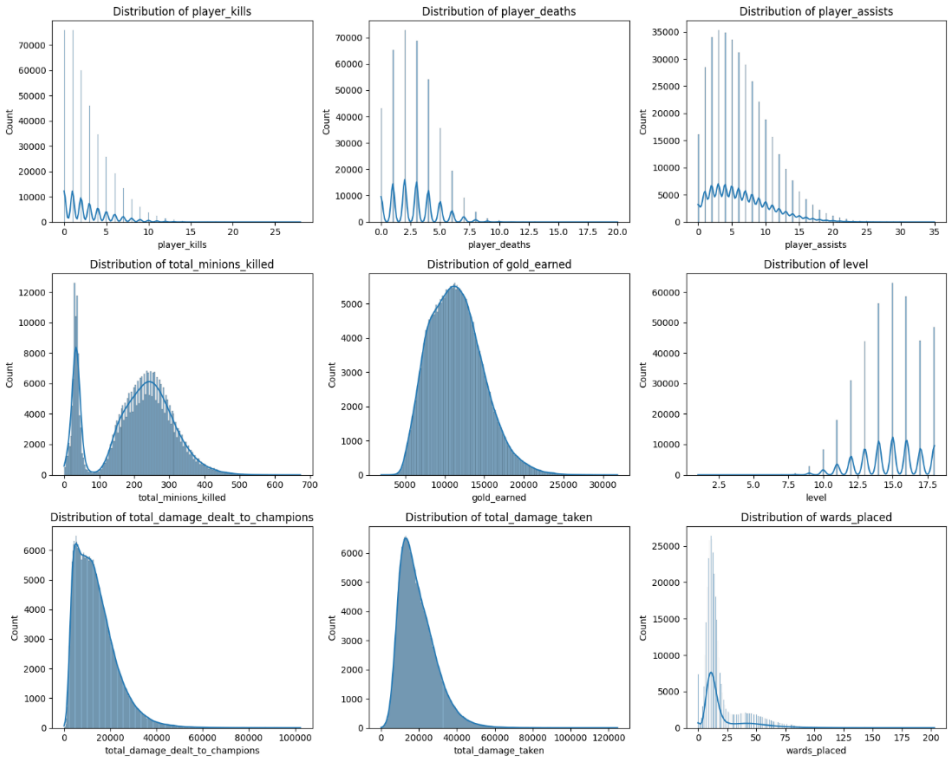

Figure 2 - Distribution des statistiques

En effet, les statistiques de jeu comme le nombre de minions tués, les dégâts infligés au champion et les dégâts encaissés dépendent fortement des rôles choisis par les joueurs. Typiquement, le bot ou le mid seront les choix les plus représentés dans ces dernières statistiques. Nous constatons que le rôle Support réalise bien plus d'assists et bien moins de kills sur joueur ou minion que tous les autres et a donc des statistiques d'argent gagné et de level plus faibles. Nous constatons que les écarts entre les autres rôles ne sont pas extrêmement importants.

Sur la figure 2, nous pouvons également noter que les nombres de « player_kills », « player_deaths » et « player_assists » ont une variation bien plus faible par joueurs par partie comparés aux autres données. On peut donc émettre l'hypothèse que la majorité des joueurs auront des statistiques proches les uns des autres pour ces features.

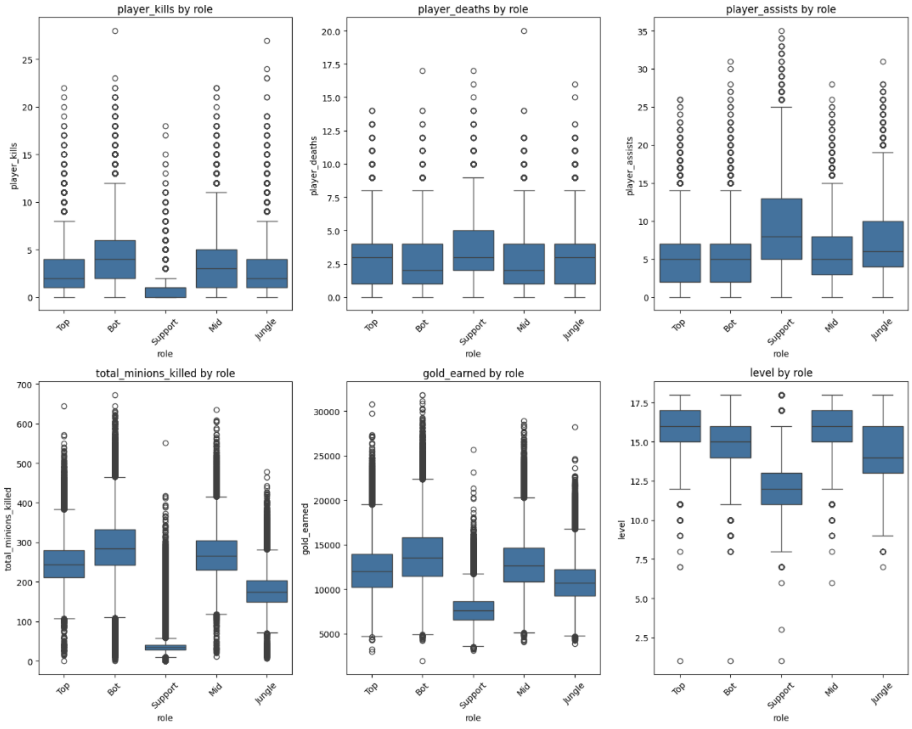

Figure 3 : Player stats by role

Enfin, l'analyse du dataset va nous permettre de corréler les différentes données avec la variable « win » correspondant à une victoire. Une matrice de corrélation (figure 4) peut être générée afin de présélectionner les features à garder dans notre modèle, commentées plus bas.

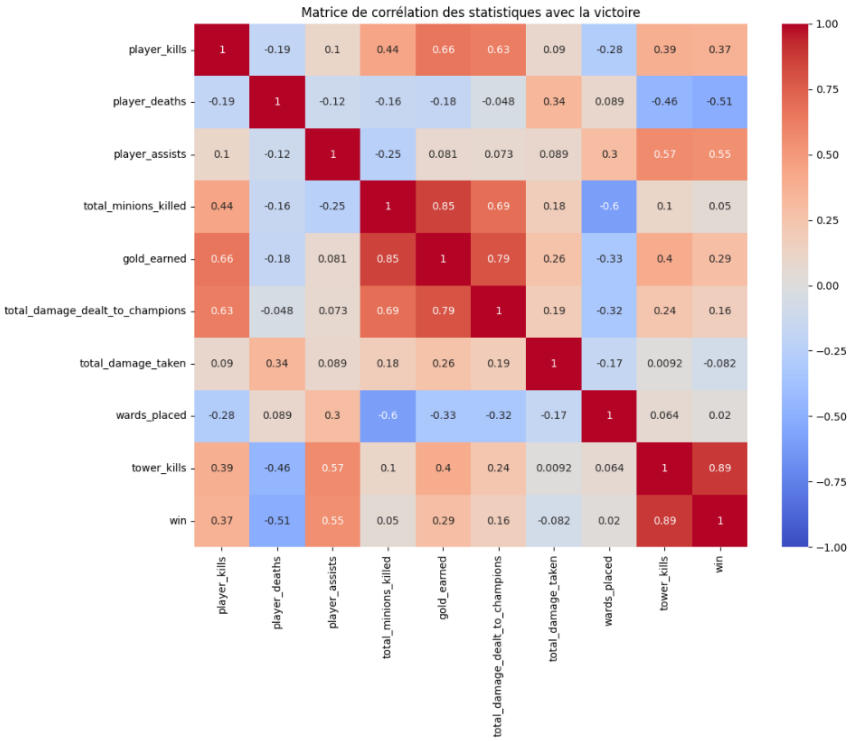

Figure 4 - Matrice de corrélation des statistiques avec la victoire

3 Définition des features

3.1 Sélection de features préexistantes

Nous avons déterminé que les features pertinentes à garder sont celles qui concernent les statistiques in game et varient entre les parties. Voici les features que nous avons donc retiré pour entrainer le modèle :

- game_id, player_id, player_name, team_id, team_name, team_acronym, rôle,champion_name car elles sont descriptives vis-à-vis du joueur et ne représentaient pas un indicateur de performance
- game_length, team_kills car elles concernent toute l'équipe et ne sont donc pas pertinentes pour classer les joueurs individuellement.
- « win » étant la target feature, elle a, elle aussi, été retirée du dataset pour être placée dans la variable « y ».

Enfin, avant d'entraîner notre modèle, nous avons choisi d'éviter les features trop corrélée avec la target « win ». Le résultat ci-dessous nous permet de penser que la feature « tower_kills » serait trop corrélée, celle-ci étant une statistique clé pour obtenir la victoire car la destruction des tours augmente énormément les chances de détruire le nexus adverse.

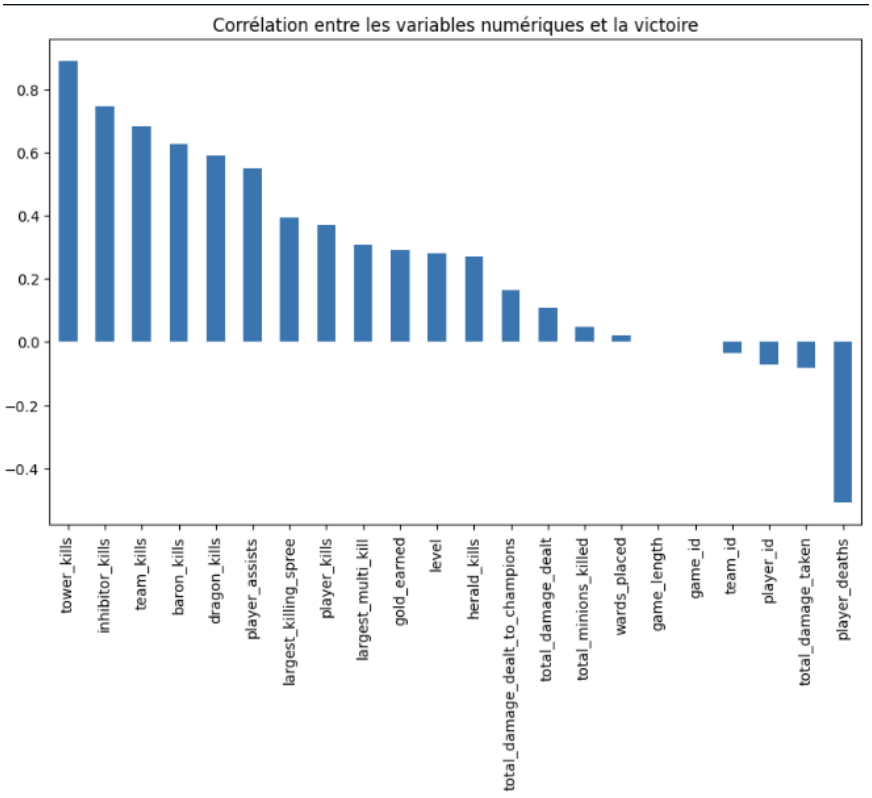

Figure 5 - Corrélation entre les variables numériques et la Victoire

3.2 Création de nouvelles features

En tout, 7 features ont été crées :

- Kda signifie la somme des kills et assists divisés par le nombre de morts, un indicateur très répandu
- Kill_participation représente les kills du joueur divisés par ceux de l'équipe
- Cs_per_minute, gold_per_minute, Dmg_taken_per_minute et vision_per_minute concernent respectivement le nombre de minions tués, l'argent gagné, les dégâts subis et les wards placés divisés par la durée de la partie en minute
- Damage_per_gold constitue les dégats totaux infligés aux champions divisés par l'argent gagné

4. Pre-processing des données

En utilisant la fonction « game_players_stats.isnull().sum() », nous avons observé que 290 données étaient manquantes dans la feature « team_acronym ». Celle-ci n'étant pas pertinente pour la tâche à réaliser comme expliqué ci-dessous, nous avons choisi d'ignorer ces valeurs manquantes.

De même, aucune variable catégorique n'a été jugée pertinente de garder pour les raisons listées ci-dessus.

Toutes les données ont été mise à l'échelle dans notre fonction de pre-processing grâce à la fonction scaler.fit_transform de StandardScaler. Ensuite, le training set est séparé du validation set, avec un validation_set représentant 20% du dataset. Cette fonction est appelée avant d'entraîner chaque modèle.

5. Comparaison des modèles et sélection des hyperparamètres

Nous avons choisi d'entrainer 3 modèles pour la tâche de classification binaire qui nous était donnée : les modèles testés et leurs hyperparamètres sont détaillés ci-dessous. Chaque modèle a été testé pour chaque rôle différent.

- Random Forest : Algorithme basé sur le concept d'arbres de décision. Il construit plusieurs arbres pendant l'entraînement, puis combine leurs prédictions pour produire un résultat plus robuste et plus précis.
    - N_estimators : nombre d'arbres dans la forêt. Plus il y en a, plus le modèle est stable et robuste, mais plus il est lent.
    - Max_depth : profondeur maximale des arbres. Une profondeur illimitée (None) peut entraîner de l'overfitting donc on teste aussi des limites de 10 et 20.
    - Min_samples_split : nombre minimal d'échantillons pour diviser un nœud.

- SVM : algorithme de classification visant à séparer les classes avec une frontière optimale, appelée hyperplan.
    - C : paramètre de régularisation. Un petit C laisse passer plus d'erreurs pour avoir une marge plus large (modèle plus général), un grand C pénalise fortement les erreurs (modèle plus rigide). Nous avons choisi de tester les paramètres 0.1, 1 et 10
    - kernel : fonction qui transforme les données pour les rendre séparables. L'argument « linear » génère un modèle rapide et fonctionne bien si les données sont linéairement séparables, tandis que rbf (radial basis function) permet de modéliser des frontières plus complexes.

- XGBoost, Extreme Gradient Boosting : Contrairement à Random Forest qui construit les arbres en parallèle, XGBoost construit ses arbres de manière séquentielle. Chaque nouvel arbre corrige donc les erreurs des arbres précédents. On utilise la méthode de boosting par gradient.
    - learning_rate : taux d'apprentissage. Plus il est petit, plus l'apprentissage est lent mais précis. 0.01 convergera plus lentement, 0.3 est plus agressif et 0.1 entre les 2, en théorie.
    - N_estimators et max_depth sont expliqués plus haut
- MLP, le perceptron multicouche : réseau de neurones artificiel de type feedforward. La fonction d'activation par défaut est ReLu, que nous avons jugée adéquate.
    - hidden_layer_sizes : définit la structure du réseau. (50,) = 1 couche de 50 neurones. C'est un modèle plus simple à priori. (50, 50) teste une architecture plus profonde pour tenter d'apprendre des relations plus complexes. Une multitude de valeurs ont été testées pour ce paramètre, le MLP étant le modèle qui donnait les meilleures performances.
    - alpha : paramètre de régularisation L2. Celui-ci permet d'éviter que les poids deviennent trop grands, ce qui peut mener à de l'overfitting. On teste différentes forces de régularisation : 0.0001, 0.001, 0.01.

Les hyperparamètres optimaux sont trouvés automatiquement grâce à GridSearchCV, un outil de scikit-learn qui teste toutes les combinaisons possibles d'un ensemble de valeurs prédéfinies et sélectionne les meilleurs hyperparamètres en utilisant une validation croisée (ici cv=3 ce qui correspond à une validation croisée à 3 plis). Cet outil choisit ensuite la meilleure combinaison selon un critère d'évaluation. Dans notre cas, nous avons choisi l'accuracy.

Le meilleur modèle est déterminé en maximisant l'AUC. L'AUC, pour Area Under the Curve est une mesure qui évalue la capacité d'un modèle de classification binaire à distinguer entre les classes. Elle correspond à l'aire sous la courbe ROC, qui trace le taux de vrais positifs (sensibilité) contre le taux de faux positifs pour différents seuils de décision. Une AUC de 1 indique une séparation parfaite des classes, 0,5 correspond à un modèle aléatoire, et moins de 0,5 signifie que le modèle se trompe plus qu'il ne réussit. Contrairement à la simple précision, nous avons choisi d'optimiser notre modèle en fonction de l'AUC car cette mesure est robuste aux déséquilibres de classes et donne une vue globale de la performance du modèle.

Le modèle qui a à chaque fois le mieux performé est le perceptron multicouche, avec des AUC aux alentours de 98% et des scores F1 ($F1\_score = 2 . \frac{Précision . Rappel}{Précision+Rappel}$) proches de 92%. Les meilleurs hyperparamètres pour ce modèle étaient :

- Soit un modèle plus profond comprenant 20 couches de 20 neurones avec un paramètre de régularisation plus faible de 0.001 pour les rôles mid et bot.

- Soit un modèle plus simple comprenant une couche de 50 neurones pour un paramètre de régularisation alpha 10 fois plus grand de 0.01 pour les rôles top, support et jungle.

6. Conclusion : résultats des missions

6.1 Prédiction de victoire ou défaite

Pour conclure, nous avons réussis à entraîner un modèle de prédiction de victoire ou de défaite qui analyse chaque rôle en fonction des features importantes liées à la victoire.

Pour chaque analyse, nous effectuons un entrainement suivant quatre modèles, comme expliqué au point précédent. Le modèle le plus efficace pour chaque rôle est le Neural Network avec lequel nous obtenons un score F1 de 92% environ (Figure 6).

Le réseau de neurones possède également un « AUC » très élevé (~0.98) qui témoigne d'une excellente capacité prédictive.

```
==================================================
Analyse pour le rôle: Top
==================================================

Entraînement du modèle Random Forest pour le rôle Top...
Meilleurs paramètres pour Random Forest: {'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 200}
Accuracy: 0.9048, Precision: 0.8924, Recall: 0.9175, F1: 0.9048, AUC: 0.9685

Entraînement du modèle SVM pour le rôle Top...
Meilleurs paramètres pour SVM: {'C': 1, 'kernel': 'rbf'}
Accuracy: 0.9203, Precision: 0.9083, Recall: 0.9324, F1: 0.9202, AUC: 0.9762

Entraînement du modèle XGBoost pour le rôle Top...
Meilleurs paramètres pour XGBoost: {'learning_rate': 0.3, 'max_depth': 3, 'n_estimators': 200}
Accuracy: 0.9150, Precision: 0.9040, Recall: 0.9258, F1: 0.9147, AUC: 0.9758

Entraînement du modèle Neural Network pour le rôle Top...
Meilleurs paramètres pour Neural Network: {'alpha': 0.01, 'hidden_layer_sizes': (50,)}
Accuracy: 0.9202, Precision: 0.9067, Recall: 0.9343, F1: 0.9203, AUC: 0.9789

Meilleur modèle pour Top: Neural Network (AUC: 0.9789)
```

Figure 6 - Analyse des modèles pour le rôle Top

Il est important de noter que les autres modèles d'entrainement comme SVM et XGBoost sont de très bonnes alternatives. En effet, ils obtiennent également un score F1 à 92% et un AUC de 97%.

Ensuite, grâce à une matrice de confusion (Figure 7) , nous pouvons évaluer la performance de classification du modèle de réseau neuronal. « 6891 » correspond à un vrai négatif, le modèle a correctement prédit une défaite. « 6897 » correspond à un vrai positif, le modèle a

correctement prédit une victoire. Les deux autres labels correspondent aux erreurs de prédictions. Nous pouvons conclure que le modèle possède une bonne performance globale, avec une erreur de 0.1% environ, et une tendance équilibrée.

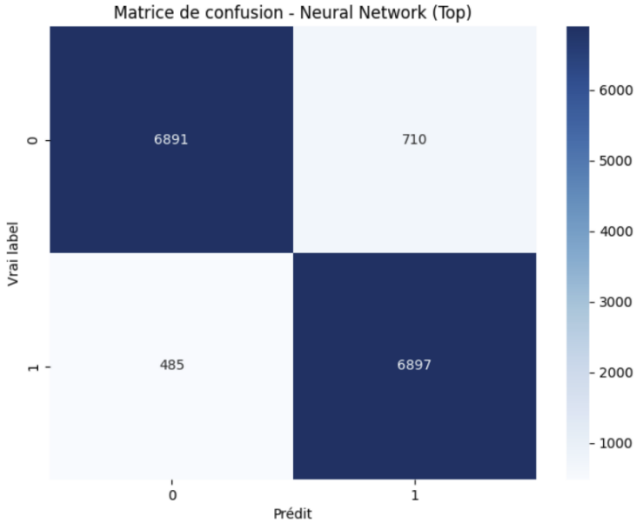

Figure 7 - Matrice de confusion - Neural Network

Enfin, en analysant l'évolution de la fonction de perte (dans l'exemple du Top, Figure 8) ainsi que le score F1 et l'AUC, nous pouvons conclure que la courbe de perte diminue rapidement pendant les premières itérations et se stabilise progressivement ensuite vers une valeur plancher et qu'ainsi les courbes F1 et AUC se dirigent très rapidement vers les valeur plancher énoncées plus haut. Ceci indique que le modèle apprend efficacement sans overfitting.

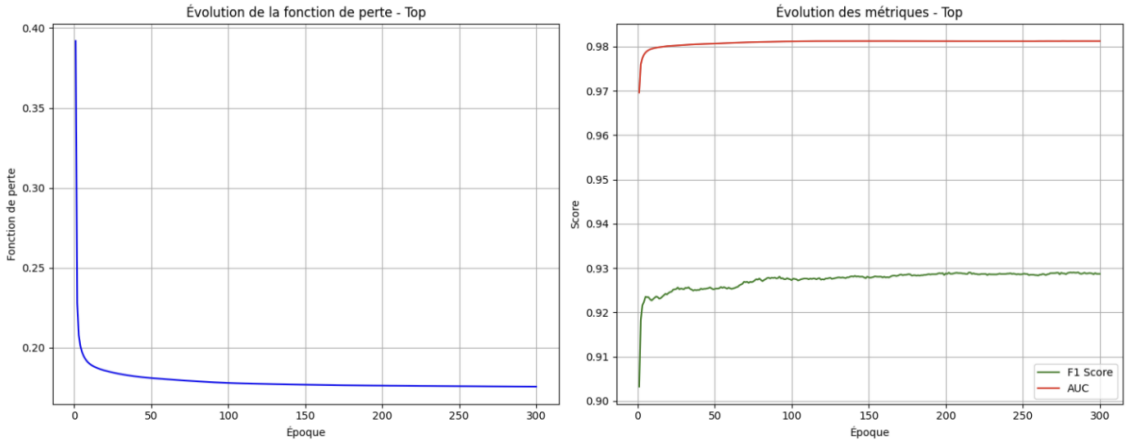

Figure 8 - évaluation des pertes en fonction des epoch et évolution des métriques

6.2 Sélection des meilleurs Joueurs

Notre modèle est capable de faire un classement des meilleurs joueurs en fonction de le leur rôle.

Ce classement s'opère grâce à la fonction « Calculate_player_skill_ranks ». Elle collecte les probabilités de victoire pour chaque rôle pour ensuite les convertir en percentiles relatifs au rôle et calculer sa moyenne. On va normaliser les performances des joueurs avec au moins cinq parties pour permettre une comparaison équitable entre joueurs d'un même rôle.

Le résultat affiche les cinq meilleurs joueurs (exemple figure 9) pour chaque rôle et sauvegarde les rangs d'habileté dans un fichier csv.

```
Top 5 joueurs pour le rôle Top:
1. Putin - Percentile: 76.58 (sur 38 parties)
2. prey - Percentile: 74.50 (sur 8 parties)
3. Millicent - Percentile: 70.25 (sur 8 parties)
4. GreatGary - Percentile: 69.75 (sur 12 parties)
5. Shemek - Percentile: 65.77 (sur 61 parties)
```

Figure 9 - Top 5 des joueurs en Top

Selon nos analyses, une équipe de rêve comprenant les meilleurs joueurs de chaque catégorie serait composée de Putin en Top, Yåtø en Bot, Nerzhul en support, Druust en Mid et Maxou en Jungle. Cependant, il est à noter que l'esprit d'équipe et la communication entre joueurs sont des paramètres qui n'ont pas été évalués et le temps de communication par joueur par partie ou le nombre total de parties jouées avec la même équipes seraient des statistiques à prendre en compte pour déterminer les joueurs les plus aptes à jouer ensemble.

6.3 Facteurs clés de victoire : Analyse SHAP

Enfin, l'analyse SHAP, réalisée sur chaque rôles, nous révèle l'impact de chaque feature sur la prédiction de victoire pour un rôle donné. Les caractéristiques sont classées par ordre d'importance de haut en bas et la coloration indique la valeur de chaque feature (bleu = valeur basse, rouge = valeur élevée). L'axe horizontal montre l'impact sur la prédiction de victoire : Les valeurs négatives (à gauche) diminuent la probabilité de victoire et les valeurs positives (à droite) augmentent la probabilité de victoire

Dans cet exemple concernant le rôle Bot (Figure 10), on observe qu'un kda élevé, gold_earned ou encore kill_participation sont des features plus fortement liées à une prédiction de victoire tandis que le « total damage dealt to champions », par exemple, n'impacte que peu cette prédiction. On constate aussi que peu importe le rôle, le ratio kda reste la feature la plus importante. La participation aux killls est la seconde feature la plus importante pour les rôles Top et Jungle et la 3ème la plus importante pour les rôles Mid et Bot.

L'argent récupéré est la seconde feature en terme d'importance pour les rôles Support, Bot et Mid la 3ème la plus importante pour les rôles Top et Jungle.

Il est intéressant de noter que le rôle support a comme 3ème plus importante feature non pas kill_participation mais bien player_assists, ce qui parait logique vu leur rôle de soutien.

En résumé, et comme mentionné ci-dessus, peu importe le rôle, maximiser le ration Kill+Assists/Morts « Kda » semble de loin être un facteur clé pour s'assurer la victoire.

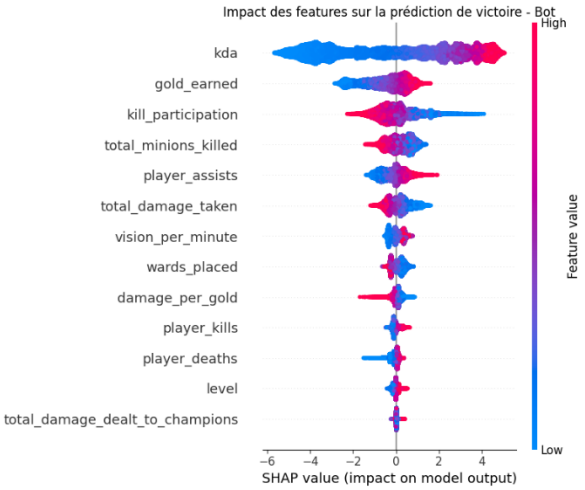

Figure 10 - Analyse SHAP pour le rôle BOT

Références

Lien vers notre repository github : https://github.com/Jardilou/AI_Project