PRÉDIRE L'ISSUE D'UN MATCH DE FOOTBALL PAR APPRENTISSAGE AUTOMATIQUE

MATHIS ANTONETTI, GASPARD CHONÉ-DUCASSE

1. Introduction

Dans le cadre du cours *Apprentissage et génération par échantillonnage aléatoire* du Professeur Sétphane Mallat au Collège de France, nous avons eu l'opportunité de participer au Data Challenge, sur des données fournies par Qube Research & Technologies, avec pour objectif de prédire les résultats des matchs de football. Ce défi nous a plongés au cœur de l'analyse et de la modélisation de données complexes, mettant en lumière les défis inhérents à la prédiction dans le domaine du sport.

1.1. **Données.** Les données mises à notre disposition sont divisées en deux catégories principales : celles concernant les équipes et celles relatives aux joueurs. Pour un match, les mêmes statistiques nous sont fournies pour l'équipe jouant à domicile et celle à l'extérieur. Nous concaténons les *features* construites pour chacune des équipes pour construire l'*input* final de notre modèle.

Se pencher sur les données relatives aux équipes a révélé l'importance cruciale des différents championnats, avec des profils de données distincts pour chaque compétition. Cette diversité présente un défi majeur en termes de généralisation, notamment lorsqu'il s'agit d'appliquer nos modèles aux données de test qui concernent des championnats inconnus.

De même, les données des joueurs présentent des complexités. En effet, la fluctuation du nombre de joueurs par équipe ainsi que l'absence fréquente d'informations sur les positions des joueurs ont rendu leur exploitation difficile et nécessitant des approches spécifiques.

1.2. **Modélisation.** Pour relever ce défi de prédiction, nous avons, dans un premier temps, exploré les méthodes classiques telles que la régression logistique et les méthodes de *boosting*. Puis dans un second temps, nous avons expérimenté avec une approche novatrice reposant sur une architecture *Transformer* personnalisée. Pour cette dernière, nous avons utilisé une procédure d'entraînement alternée novatrice, permettant une meilleure généralisation.

1.3. **Augmentation de la donnée.** Finalement, nous avons explorée l'idée pratique d'augmenter la quantité de données en utilisant les profils des données manquantes du dataset de test pour simuler les particularités des championnats inconnus. Nous avons alors entraîné des modèles différents en fonction du nombre de données manquantes, améliorant ainsi la robustesse et la généralisation de nos prédictions.

Dans ce rapport, nous présentons en détail notre méthodologie, les résultats obtenus ainsi que les perspectives d'amélioration.

2. Analyse de donnée & diagnostic

Dans cette section dédiée à l'analyse approfondie des données, nous explorons d'une part les données relatives aux équipes et d'autre part les défis rencontrés dans l'exploitation des données relatives aux joueurs.

Date: Mars 2024.

Valeurs Manquantes		
League	**Moyenne**	**Somme des écarts-types par features**
Bundesliga	0.220	3.296
Eredivisie	8.034	13.13
J-League	0.672	5.502
La Liga	0.236	3.646
League One	63.24	56.93
League Two	49.35	58.32
Liga Portugal	0.136	3.057
Ligue 1	0.140	3.233
Ligue 2	5.817	9.499
Premier League	0.081	1.527
Pro League	41.86	42.36
Serie A	0.399	4.028
Superliga	3.480	17.72
Test	**5.858**	**29.267**

TABLE 1 – Statistiques sur les valeurs manquantes dans les données regroupées par championnats.

2.1. **Équipes.** L'exploration des données relatives aux équipes met en lumière l'impact significatif des championnats sur les données. Cette découverte est cruciale car elle souligne la nécessité d'adapter nos modèles à la diversité des compétitions. Ces disparités entre championnats sont illustrées dans la table 1 qui présente des statistiques sur les valeurs manquantes dans la donnée.

Nous montrons par exemple, que pour des championnats moins suivis par le grand public, tels que la Ligue One et la Ligue Two, troisième et quatrième divisions anglaises, nous avons significativement moins de données disponibles (ce qui est attendu.)

Mais, nous prouvons, de plus, que cette diversité des championnats pose un défi majeur pour la généralisation de nos modèles aux données de test. En effet, les statistiques de la table 1 montrent que la donnée de test, contenant beaucoup de valeurs manquantes, est de mauvaise qualité et regroupe des championnats au profils très différents. En pratique, lorsque nous appliquons nos algorithmes entraînés sur des championnats bien connus sur ces données provenant de compétitions inconnues et moins fournies en données, nous constatons une baisse de performance notable que nous allons chercher à minimiser.

2.2. **Joueurs.** Les données individuelles des joueurs sont moins évidentes à exploiter car le nombre de joueurs par équipe peut varier (entre 18 et 25 pour l'ensemble d'entrainement par exemple) et beaucoup de données sont manquantes. Typiquement, une équipe de foot complète doit forcément être composée d'au moins un garde, un attaquant, un défenseur et un milieu de terrain. Cependant, si l'on regarde les matchs pour lesquels c'est le cas pour les deux équipes (HOME et AWAY) dans l'ensemble d'entrainement, on s'aperçoit qu'il n'y en a que 5785 sur 12302.

En entrainant une régression logistique pour prédire le rôle de chaque joueur, on obtient une précision de $\sim 80\%$ et si on fait la même vérification que précédemment, on obtient en rajoutant ces valeurs prédites aux endroits où le rôle du joueur est manquant un nombre de matchs "normaux" de 10834 ce qui est beaucoup mieux. Avec ce modèle, on peut donc exploiter $\sim 88\%$ des données d'entraînement individuelles.

Pour exploiter ces données, il faut les mettre sous un format (n, p) où n est le nombre de matchs et p un nombre de *features*. Nous avons développé deux méthodes dans ce but :

— Méthode 1 : Faire la moyenne de chaque *feature* sur tous les joueurs d'une même équipe avec le même rôle. Ainsi, on obtient $p = 4f$ où f est le nombre de *features* des données individuelles des joueurs.

— Méthode 2 : Faire la moyenne pondérée par le temps de jeu de chaque *feature* sur tous les joueurs d'une même équipe avec le même rôle. Nous sélectionnons ensuite par position les *features* les plus pertinentes, et nous obtenons $p = 1,5f$.

3. Modèles

3.1. **Boosting.** Un modèle populaire pour ce type de challenge est CatBoost auquel on associe généralement une *feature selection* préalable pour éliminer du bruit inutile (voir par exemple [1]). Ainsi, on commence par utiliser CatBoost (avec une sélection de modèle pour éviter le surapprentissage) sur l'ensemble des données d'équipes ce qui donne un benchmark de 50% sur l'ensemble d'entrainement et 48.74% sur l'ensemble de test public.

Tout l'enjeu est de trouver une bonne représentation des données en entrée. Nous avons essayé de faire une *feature selection* par le gain d'information mutuelle mais le résultat sur l'ensemble de test est quasi-identique au benchmark donc nous nous sommes concentré sur d'autres pistes.

3.2. **Transformers.** Afin d'établir un benchmark pour une architecture de réseaux de neurones, nous avons d'abord essayé une régression logistique sur les données des équipes qui donnait 48.56% sur l'ensemble d'entrainement complet et 47.63% sur le test public (on constate encore une fois l'écart entre les deux datasets). En s'inspirant de [1], nous avons utilisé des trasnformers pour améliorer ce résultat numérique. L'architecture finale de type transformers retenue pour le modèle MCTeam est décrite par la Figure 1.

Ce modèle prend toutes les données des équipes en entrée et généralise la régression logistique en rajoutant la partie encadrée en gris. Le premier batchnorm permet de réduire l'influence du biais entre la distribution d'entrainement et la distribution de test en normalisant les données en entrée. Ensuite, on simplifie les *features* en diminuant leur nombre avec une couche fully-connected puis on renormalise pour effectuer ensuite une régression logistique.

On doit s'assurer que le modèle généralise bien aux autres leagues qui sont dans les données de test mais pas dans les données d'entrainement. Pour ce faire, nous nous sommes inspiré de [2] pour entrainer le modèle de manière alternée. On note f la fonction de classification (partie rouge) et φ la fonction de

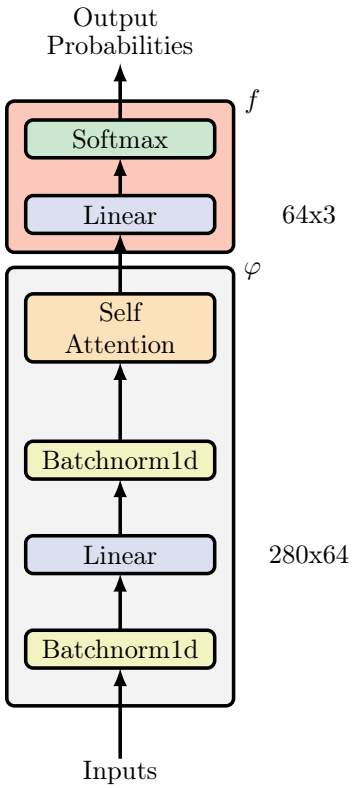


Figure 1 – Architecture du modèle MCTeam

représentation (partie grise). D'abord, on optimise φ et f avec

$$f^{(t+1)}, \varphi^{(t+1)} = \underset{f,\phi}{\operatorname{argmin}} \sum_{i=1}^{n} w_i^{(t)} L(f(\varphi(x_i)), y_i),$$

où L est une cross-entropy loss, puis on optimise w avec

$$w^{(t+1)} = \underset{w}{\operatorname{argmin}} \sum_{i \neq j} \|\hat{\Sigma}_{Z_{:,i}^{(t+1)}, Z_{:,j}^{(t+1)}, w}\|_F^2 + \frac{\lambda n}{\sum_{i=1}^{n} w_i},$$

où $Z^{(t+1)} = \varphi^{(t+1)}(x)$, $\|\cdot\|_F$ est la norme de Frobenius et $\lambda > 0$ est un hyperparamètre de pénalisation. $\hat{\Sigma}_{A,B,w}$ est défini pour $u, v \in (H_{RFF})^{n_{AB}}$ par

$$\hat{\Sigma}_{A,B,w} = \frac{1}{n-1} \sum_{i=1}^{n} \left(w_i u(A_i) - \frac{1}{n} \sum_{j=1}^{n} w_j u(A_j) \right)^T \left(w_i v(B_i) - \frac{1}{n} \sum_{j=1}^{n} w_j v(B_j) \right),$$

avec l'espace de fonctions $\mathcal{H}_{RFF} = \{x \longmapsto \sqrt{2}\cos(\omega x + \phi) \mid \omega \sim \mathcal{N}(0,1), \phi \sim \mathcal{U}([0, 2\pi])\}$. Dans [2], les auteurs utilisent une optimisation sous la contrainte $\frac{1}{n}\sum_{i=1}^{n} w_i = 1$ mais nous avons fait le choix de tester d'abord une pénalisation ce qui est plus adapté au framework naturel de PyTorch.

Pour approfondir cette piste de recherche, il serait possible de réécrire cette optimisation alternée comme un problème d'optimisation à deux niveaux [3] afin de pouvoir capturer dans l'optimisation, l'interdépendance entre les deux étapes. Les nouvelles méthodes adaptives [4] construites pour la résolution de ces problèmes permettraient d'accélérer l'entraînement ainsi que de résoudre les questions du choix des hyperparamètres.

4. Expériences

Cette section se concentre sur les difficultés expérimentales que nous avons pu rencontrer lors du projet. Nous passons donc sur les entraînements et des recherches d'hyperparamètres pour les algorithmes classiques. Par ailleurs, nous décrivons une astuce pratique que nous avons développée pour tenter de résoudre les problèmes de généralisation.

4.1. **Transformers.** On entraine MCTeam de manière classique puis on relance un entrainement "from scratch" en utilisant la décorrélation décrite précédemment (le modèle obtenu est appelé MCStableTeam) pour $\lambda = 500$. Voici ce qu'on obtient :

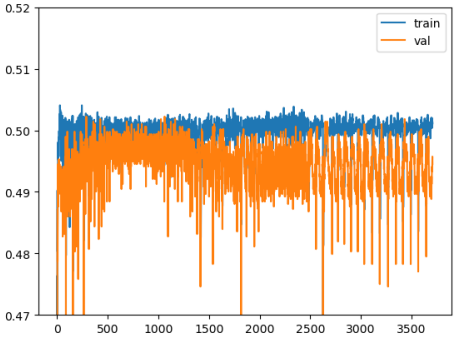


Figure 2 – Entrainement de MCTeam sur 3500 epochs

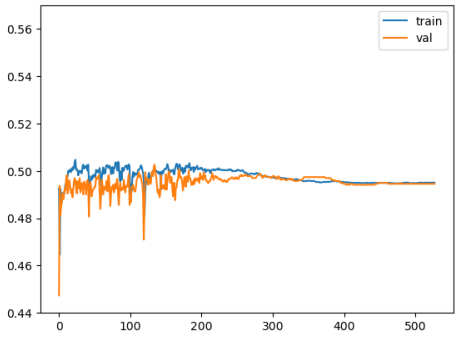


Figure 3 – Entrainement de MCStableTeam sur 500 epochs

On constate que l'entrainement est stabilisé par la décorrélation mais les résultats numériques obtenus sur l'ensemble d'entrainement restent quasi-identiques. En revanche, on constate une amélioration des résultats (voir ci-après) avec MCStableTeam sur l'ensemble de test, ce qui montre une meilleure généralisation "out-of-distribution".

En rajoutant les données individuelles des joueurs avec la méthode 1, on obtient :

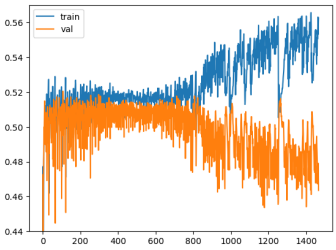


FIGURE 4 – Entrainement de MCPlayer avec toutes les données sur 1500 epochs

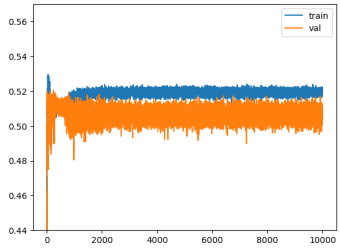


FIGURE 5 – Entrainement de MCPlayer avec une *feature selection* en amont sur 10000 epochs

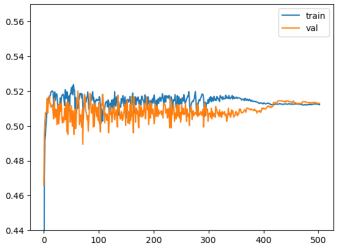


FIGURE 6 – Entrainement de MCStablePlayer avec une *feature selection* en amont sur 500 epochs

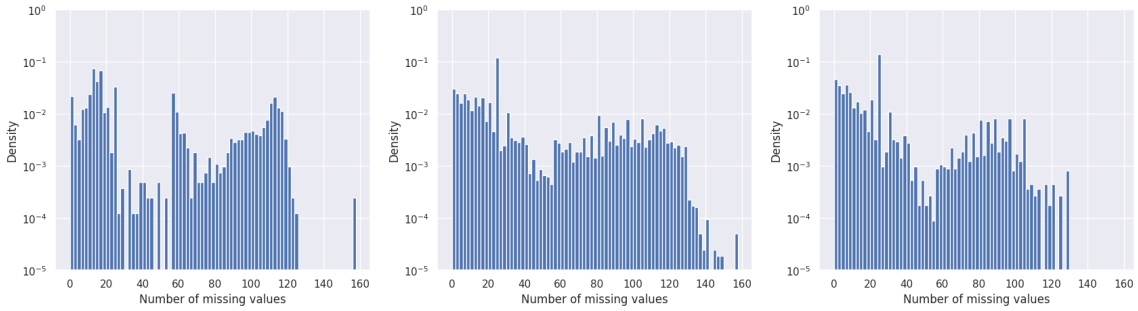
On voit que si l'on met toutes les données en entrée du modèle, celui-ci surapprend complètement et ne généralise pas bien sur l'ensemble de validation. Pour remédier à ce problème, nous avons sélectionné les 512 meilleures *features* en terme de gain d'information mutuelle sur le résultat du match. On observe alors un résultat plus stable et une meilleure généralisation. Le modèle ainsi obtenu est appelé MCPlayer. En utilisant l'optimisation à deux niveaux, on obtient sans surprise un entrainement plus stable et le modèle obtenu généralise mieux aux données de test même si l'écart reste important (2.06% d'écart d'après les résultats numérique ci-après).

4.2. **Augmentation de la donnée.** L'analyse des données a révélé le défi majeur lié à la généralisation des modèles aux championnats aux données lacunaires présents dans le dataset de test. Pour adresser cette difficulté, nous avons eu l'idée de modifier les données d'entraînement afin de refléter ces lacunes observées dans les championnats moins documentés.

Nous considérons le manque de données comme une signature d'un championnat et nous voulons transférer cette signature des championnats de test à ceux de l'entraînement. Pour ce faire, nous créons des masques de valeurs manquantes à partir de la donnée de test et nous les appliquons aléatoirement lors d'un sur-échantillonage de la donnée d'entraînement.

La figure 7 présente la distribution des valeurs manquantes avant, et après sur-échantillonage et nous permet de comparer avec la donnée de test. Nous obtenons une donnée plus représentative de ce qui est observé dans le test et nous avons de plus augmenté la taille de notre dataset d'entraînement qui passe de 12303 lignes à 88359 lignes.

En adoptant cette approche, nous diversifions les données d'entraînement en intégrant des profils de championnats divers et surtout nouveaux, ce qui permet de mieux préparer nos modèles à généraliser pour la prédiction sur des données de test. De plus, afin de simuler l'entraînement d'un modèle spécifique par championnat, nous proposons d'entraîner des modèles distincts en fonction de la quantité de données disponibles pour chaque ligue. Cette stratégie permet de mieux capturer les spécificités des compétitions et d'améliorer ainsi la robustesse et la précision de nos prédictions.

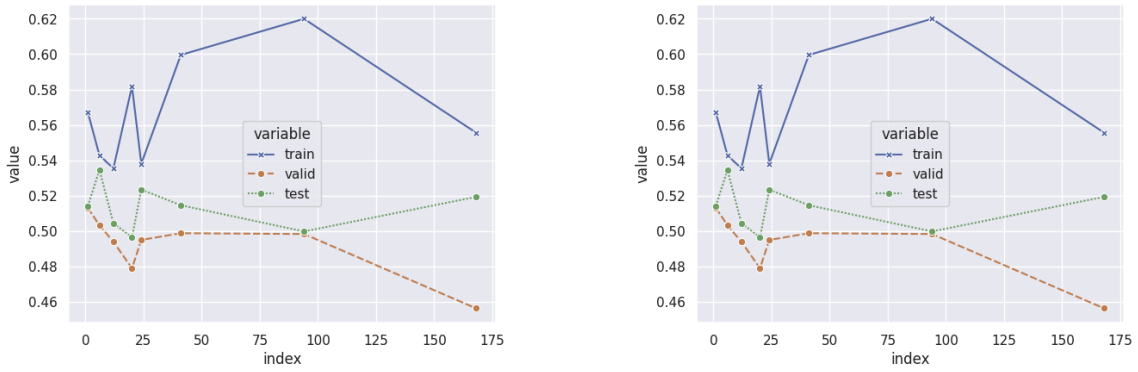


Originale Modifiée Test

FIGURE 7 – Histogramme du nombre de valeurs manquantes (en ne considérant que la donnée incomplète) pour la donnée d'entraînement non-modifié (à gauche), le résultat de notre ré-échantillonage (au centre) et de la donnée de test (à droite).

Les sous-divisions du dataset d'entraînement, 8 bins calculés avec le nombre de valeurs manquantes, sont très différentes et il est difficile d'entraîner un modèle avec les mêmes paramètres sur toutes. Nous avons donc essayer d'effectuer une *grid-search* par division afin d'obtenir des résultats adaptés pour toutes. Cependant nous avons subi assez largement l'*overfitting*.

Nous avons testé cette méthode avec des modèles Catboost qui sont simples et rapides à entraîner et qui avait très bien fonctionné sur ce challenge sans sur-échantillonage. Et nous avons aussi appliqué cette nouvelle méthode à notre modèle MCStableTeam.



CatBoost MCStableTeam

FIGURE 8 – Performance des modèles sur le training, validation et testing dataset en fonction du nombre de valeurs manquantes dans la donnée.

Les résultats sont positifs, nous avons réalisé un meilleur score avec cette méthode et Catboost que celui obtenu sans cette méthode. Mais les performances calculées par bins de nos modèles (figure 8) montrent que nos choix d'hyperparamètres ne sont pas optimaux.

Résultats numériques			
Modèle	**Train**	**Score public**	**Score privé**
Régression logistique	0.4865	0.4763	?
MCPlayer	0.511	0.4843	?
MCTeam	0.5	0.4866	?
MCStableTeam	0.5	0.4878	?
MCStablePlayer	**0.5124**	**0.4918**	?
Catboost	0.5	0.4874	?
Split na + Catboost	0.5	0.4889	0.4840

5. Conclusion

Le problème de prédire l'issue d'un match de football est devenu un sujet scientifique important ces dernières décennies avec le développement de l'apprentissage automatique. Cependant, cela se fait toujours sur une league particulière ou dans un contexte où le modèle a accès aux leagues (et donc au niveau de compétition), à la situation géographique, etc... . Ce challenge est original car il demande de résoudre le problème dans un contexte plus difficile où l'on ne connait ni les leagues des équipes, ni la situation géographique des matchs. Malgré cela, nous avons obtenu un score raisonnable (6^e en public et 3^e en académique public le 18 mars) avec une solution finalement assez simple à mettre en oeuvre (MCStablePlayer) et nous apportons d'autres idées prometteuses (augmentation de données). On constate cependant que le choix des hyperparamètres pour la méthode split na nous a probablement mené à un sur-apprentissage sur les données de test ce qui explique peut-être l'écart entre le score public et le score privé.

Références

[1] C. Yeung, R. Bunker, R. Umemoto, and K. Fujii, "Evaluating soccer match prediction models : A deep learning approach and feature optimization for gradient-boosted trees," *Arxiv*, 2023.

[2] X. Zhang, P. Cui, R. Xu, L. Zhou, Y. He, and Z. Shen, "Deep stable learning for out-of-distribution generalization," pp. 5368–5378, jun 2021. [Online]. Available : https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.00533

[3] J. Lorraine, P. Vicol, and D. Duvenaud, "Optimizing millions of hyperparameters by implicit differentiation," *CoRR*, vol. abs/1911.02590, 2019. [Online]. Available : http://arxiv.org/abs/1911.02590

[4] C. Fan, G. Choné-Ducasse, M. Schmidt, and C. Thrampoulidis, "Bisls/sps : Auto-tune step sizes for stable bi-level optimization," 2023.