Projet Machine Learning (ML)

Gagner à tout les coups un jeu de hasard

Étude de cas

Table des matières

Gagner à tout les coups un jeu de hasard **1**
Table des matières **2**
Réflexion sur les paris sportifs **4**
I. Méthode théorique pour faire du profit à tout les coups **4**
II. Prérequis avant de commencer le projet **6**
Roadmap initiale pour identifier les values bets **9**
Étape 1 : Apprendre les Fondations Statistiques **9**
Étape 2 : Le Langage de l'Incertitude **9**
Étape 3 : Structure des données à collecter sur 10 saisons **10**
Étape 4 : Régression Logistique Standard **10**
Étape 5 : Régression logistique par Lissage **11**
Étape 6 : Donner de la mémoire au modèle **11**
Étape 7 : Tests de diagnostic et Validation **11**
Étape 8 : Calibrage des bookmakers **12**
Rapport technique **13**
1. Régression linéaire simple **14**
2. La régression linéaire multiple **17**
2.1 Une matrice est un tableau 17
2.1.1 L'addition entre deux matrices 17
2.1.2 La soustraction entre deux matrices 18
2.1.3 La multiplication $A \times B$ 18
2.1.4 La multiplication scalaire 19
2.1.5 La transposition 19
2.1.6 L'inversion 19
3. Régression logistique standard (binaire) **27**
Trouver les coefficients avec la méthode de la descente de gradient 27
L'objectif de la méthode de Newton-Raphson 32
4. La régression logistique régularisée **38**
La fonction de coût avec régularisation L1 : 38
La fonction de coût avec régularisation L2 : 40
5. La régression logistique multinomiale régularisé L2 ..... **44**
6. La pondération temporelle dans la régression logistique régularisée L2 ..... **47**
7. Calcule du taux de confiance accorder au modèle **49**
8. Accroître la performance **50**

9. Le gradient boosting 51
10. Le XGBoost 56
11. Modèle de Long Short-Term Memory (LSTM) 60
12. L'apprentissage ensembliste 64
13. Calibration du modèle de régression 67
14. Embeddings 68
15. Apprentissage bout en bout avec entraînement conjoint 72

Réflexion sur les paris sportifs

Ma réflexion se porte sur le hasard, en particulier les jeux de hasard lier aux résultats sportifs. En effet, l'issue d'un jeu sportif est impossible à prédire à 100 %. Même s'il arrive souvent que le favori l'emporte, ce n'est pas toujours le cas. Il arrive parfois que l'outsider créer la surprise, et c'est ce qui fait la beauté du sport. Mon objectif ici, est de parvenir par un moyen quelconque à être gagnant peut importe la prédiction que je fais et d'en tirer un profit financier.

I. Méthode théorique pour faire du profit à tout les coups

Partons d'un constat de base des jeux de hasard :

- Une même action peut avoir plusieurs résultats différents : Par exemple , je peux lancer un dé trois fois et tombé sur trois différentes faces

Tout de même je pense que le hasard existe que par le manque de connaissance. Ce qui veux dire que si je lance un dé dans les mêmes conditions stricte plusieurs fois, je devrais obtenir à chaque fois le même résultat. Ainsi, j'estime que le hasard est un moyen de comblé le manque de connaissance dans un domaine. En effet, sans connaissance approfondit d'un domaine, il est difficile de percevoir les subtilités d'un événement. Alors ce qui peut nous paraître uniforme peut en réalité être composé d'une infinité de nuances.

Il y a donc deux façon de trouver avec certitude l'issue d'un événement :

- Comprendre toutes les nuances du domaine dans lequel se déroule l'événement et ainsi prendre en compte tout les paramètres pour trouver le bon issue.
- Choisir toutes les issues possibles de l'événement

Cependant, la première méthode est impossible à mettre en œuvre pour les événements sportifs ou tout autres événements liés au hasard, puisque comme évoqué le hasard est l'absence de connaissance assez approfondit et donc une impossibilité à connaître tout les paramètres d'un événement. La seconde méthode est quant à lui possible mais sans intérêt car choisir toutes les issues simultanément assure certainement de trouver l'issue exact, mais est sans valeur réelle.

En effet, la valeur d'une prédiction repose sur sa précision. Ainsi, on peut définir une bonne prédiction comme l'issue la plus certaine et précise possible.

Alors, si je ne peux ni obtenir la totalité des paramètres qui régissent une issue, ni choisir toutes les issues, quelles sont les possibilités qui me reste pour faire un choix, à la fois précis et juste ?

Il y a plusieurs solutions à explorer :

- Couvrir toutes les possibilités chez différents bookmakers à condition que je soi gagnant peu importe l'issue
- Avoir une évaluation plus fine que les bookmakers, en prenant en compte plus de paramètres ou/et en utilisant un modèle plus avancés.

Le problème de ces solutions :

- La plupart des bookmakers standardise leurs côte avec les autres.
- Les bookmakers ont des moyens, une expertise et une expérience plus avancé que moi alors avoir un système de prédiction plus avancé est possible mais peu probable.

Cependant, les bookmakers n'ont pas pour objectif de prédire l'issue réelle d'un événement, mais de faire en sorte de gagner peut importe l'issue de l'événement. Ainsi, la méthode choisit par les bookmakers est de couvrir toutes les issues possible d'un l'événement. Voici les étapes de cette méthode :

- 1. Estimer les côtes réelles de toutes les issues de l'événement. L'addition de l'ensemble des probabilités doivent être égal à 1. Tel que :

$$S = \sum \frac{1}{cote_i} = 1$$

- 2. Une marge est appliquer au côtes pour garantir un profit quelque soit le résultat. L'addition de l'ensemble des probabilités doivent être strictement supérieur à 1. Tel que :

$$S = \sum \frac{1}{cote_i + marge} > 1$$

- 3. Ajustement des côtes en fonction du marché réelle. Le but est d'équilibrer les mises des parieurs, pour évité d'être trop exposer à un résultat particulier. Par exemple, dans le cas où un grand nombre de parieurs miseraient sur l'issue 1 d'un événement, la côte de cette issue baissera et la côte des autres issues augmentera, jusqu'à ce que le taux d'attractivité sur toutes les issues soit équilibré.

Ainsi, pour que le profit d'un parieur soit garantit quelque soit l'issue il faut que l'addition de l'ensemble des probabilités d'un événement soit inférieur à 1. Tel que :

$$S = \sum \frac{1}{cote_i} < 1$$

Ma stratégie consiste à identifier des paris à valeur positive (value bets) en testant l'hypothèse que les bookmakers se trompent dans leur calibration sur le long terme, car ce n'est pas leur bute en réalité. Par exemple, si je constate que des événements auxquels une cote de 2.00 (impliquant une probabilité de 50%) est attribuée se produisent en réalité avec une fréquence de 60%, alors la cote était sous-évaluée. Par conséquent les value bets devrait théoriquement apparaître à l'approche de l'événement, là où la plupart des mises se font et où le bookmaker sort de sont rôle de « parieurs ». Miser systématiquement sur ce type de situations devrait, en théorie, générer un profit.

Il peut y avoir plusieurs types de problème :

- Dilemme entre manque de données et précisions : Par exemple, il se peux qu'il y est peu de données pour des côtes à 2,01. Pour résoudre cela je peux utiliser des intervalles, mais ils sont arbitraires et pourrait fausser la précisions. Donc, une façon plus robuste serait d'utiliser la régression logistique par lissage.
- Gestion des saisons atypiques : On prenant un grand nombre de saisons pour obtenir une quantité de donnée exploitable statistiquement, il peut y avoir des données atypiques. C'est à dire qui diverge de la « norme » habituelle sur une saison ou des variations propre à chaque groupes de données. Donc, pour résoudre cela, la solution est de posé un diagnostique de stationnarité avec le test de Hausman, puis en fonction de celle-ci :

    - Données stable : utilisation du modèle de régression logistique avec pondération temporelle.
    - Une saison atypique pour cause exceptionnel (comme période Covid) : supprimer cette saison
    - Variations naturelles de chaque saisons
- Savoir si les observations sont dû au hasard ou à des phénomène systémiques : Il se peut que les variations entre les côtes du bookmaker et la probabilité réelles que cette issues se produisent, soit trop faibles pour être attribuer à un effet systémique. Donc, on attribue le résultat au hasard. Pour vérifier cela, il faut utilisé le test de Hosmer-Lemshow.
- Le taux de calibration diffère en fonction du temps : Cette à dire qu'a chaque période il y'aura une variations des paramètres , des paramètres prises en comptes et même des objectifs. Pour prendre en compte toutes ces variations temporelles et observer le moment le plus opportun, j'introduirais une variable temporel dans le modèle. La valeur de cette variable pour chaque côte, sera l'heure qui sépare la publication de la côte au début de l'événement.

II. Prérequis avant de commencer le projet

Les concepts utilisé sont :

- 1. Fondations statistiques
    - Concept à apprendre : statistiques descriptives
    - Pourquoi en premier : c'est la base
    - Liens : prérequis absolus pour tous les autres modules.
    - Compétences à acquérir :
        - Définir ce qu'est une variable (dépendante/indépendante, qualitative/quantitative).
        - Calculer et interpréter des mesures de tendances central (moyenne, médiane) et de dispersion (écart-type, variance).
        - Comprendre visuellement une distribution de données (via un histogramme par exemple).
        - Objectif concret : Pouvoir décrire numériquement un ensemble de cotes de bookmakers
- 2. Le langage de l'incertitude
    - Concept à apprendre : Les probabilités
    - Pourquoi en second : les statistiques décrivent le passé, les probabilités prédisent le futur. C'est le cœur de l'analyse des paris.
    - Liens : Appliquer un cadre probabiliste au concepts de variables de l'étape 1.
    - Compétences à acquérir :
        - Convertir une cote brute (ex : 2,5) en probabilité implicite (1/2,5 = 40%).
        - Comprendre ce qu'est un « value bet » : si ton modèle estime une probabilité à 50 % pour un événement coté à 2.5 (40%), il y a une value.
        - Calculer l'espérance de gain (Excepted Value) d'un pari.
        - Objectif concret : Analyser une liste de cotes et identifier, sur la base des probabilités implicites, les bookmakers qui semblent offrir des valeurs anormales.
- 3. L'outil de base de la prédiction
    - Concept à apprendre : La régression logistique standard
    - Pourquoi maintenant ? : Après avoir appris le concept de probabilité, il faut apprendre à les prédire avec des données.

    - Liens : Permet d'utiliser les variables apprise à l'étape 1 pour prédire une probabilité étape 2. Et prérequis obligatoire pour comprendre sa version « lissé ».
    - Compétences à acquérir :
        - Comprendre la fonction sigmoïde et comment elle transforme une combinaison linéaire de variables en une probabilité entre 0 et 1.
        - Comprendre le principe du Maximum de Vraisemblance pour estimer les coefficients du modèle.
        - Interpréter les coefficients : « Toutes choses égales par ailleurs, une augmentation de X augmente/diminue la probabilité que Y=1 ».
        - Objectif concret : Construire un modèle simple pour prédire la probabilité qu'une équipe gagne en fonction de variables simples (ex : classement, forme récente).
- 4. L'amélioration cruciale (Anti-sur-apprentissage)
    - Concept à apprendre : La régression logistique par lissage (régularisation)
    - Pourquoi maintenant ? : La version standard sur-apprend. Pour être profitable sur de nouvelles données (de nouveaux matchs), le modèle doit être robuste. Le lissage est la solution.
    - Liens : Extension directe de la régression logistique standard. Au lieu de juste maximiser la vraisemblance, on y ajoute une contrainte (la pénalité) pour empêcher les coefficients de devenir trop grands.
    - Compétences à acquérir :
        - Expliquer la différence entre pénalité L1 (Lasso) et L2 (Ridge).
        - Comprendre le rôle de l'yperparamètre lambda (λ)qui contôle la force du lissage.
        - Entraîner un modèle qui généralise mieux à des nouvelles données non vues.
        - Objectif concret : Exploiter les potentielles inefficacités des bookmakers avec un modèle robuste, moins sujet aux illusions causés par le bruit dans les données historiques.
- 5. Donner de la mémoire au modèle
    - Concept à apprendre :Les variables & la pondération Temporelle
    - Pourquoi maintenant ? : Un modèle statistique traite toutes les données de la même façon. Or, un match d'il y a 5 ans est moins pertinent qu'un match d'il y a 1 mois. Il faut incorporer la dimension temporelle.
    - Liens : Adapter le modèle de régression lissé de l'étape 4 pour qu'il accorde plus de poids (influence) aux observations les plus récentes.
    - Compétences à acquérir :
        - Créer des variables explicites qui capturent le temps (ex : jours écoulé depuis le match)
        - Objectif concret : Prendre en compte la valeur du temps et les variations de forme, de composition d'équipe, etc., pour avoir des prédictions plus réactives.
- 6. La phase de diagnostic et validation
    - Concepts à apprendre : Test de Hosmer-Lemeshow et test de Hausman
    - Pourquoi maintenant ? : Une fois le modèle construit, il faut vérifier s'il est bon et fiable. Ces tests sont le kit de diagnostic.
    - Liens :
        - Hosmer-Lemeshow test la qualité de l'ajustement du modèle de l'étape 4, en répondant à la question : Mes probabilités prédites sont-elles conformes aux probabilités observées ? (Ex : Sur 100 événements prédits à 70%, y'en à t-il environ 70 qui se sont produits ?).

        - Hausman : Test la stabilité des coefficients dans le temps (stationnarité), de l'étape 5. Et répond à la question : Les relations qu j'ai trouvées (l'impact de la forme de l'équipe sur la victoire) sont-elles stables dans le temps ou ont-elles changé ?
    - Compétences à acquérir :
        - Exécuter et interpréter les test Hosmer-Lemeshow
        - Exécuter et interpréter les test de Hausman
        - Objectif concret : Vérifier scientifiquement si les variations et les performances du modèle sont systémiques ou juste dues au hasard. C'est à dire être capable de dire : « mon modèle est solide » avec des arguments statistiques.
- 7. Synthèse et application stratégique
    - Concept à apprendre : Le calibrage (dans le contexte de l'analyse des bookmakers)
    - Pourquoi en dernier ? : Parce que c'est la synthèse de tout le reste. C'est l'application stratégique de tous les outils.
    - Liens : C'est l'utilisation du modèle de l'étape 4, l'ajustement temporelle de l'étape 5 et la validation de ces dernières à l'étape 6, qui permet d'évaluer la performances des bookmakers.
    - Objectif concret final : Identifier des patterns d'inefficacité persistants chez les bookmakers (ex : Ce bookmaker a tendance à surestimer les grandes favorites en Ligue 1) et concentrer les paris sur ces failles spécifiques.

Roadmap initiale pour identifier les values bets

Objectif :

Développer en 15 jours (à raison de 8h/jour, soit du 22 août au 5 septembre 2025) un système d'analyse de paris sportifs basé sur la data, avec un budget maximal de 100 €, puis générer 300 € de revenus dans les 30 premiers jours suivants et atteindre 1000 € de revenus mensuels récurrents (MRR) d'ici 3 mois. Pour y parvenir, je suivrais un plan intense en 8 étapes : 1) Maîtriser les bases statistiques, 2) Comprendre les probabilités et value bets, 3) Collecter des données historiques granuleuses, 4) Construire un modèle de régression logistique standard, 5) L'appliquer avec un lissage pour plus de robustesse, 6) Intégrer la dimension temporelle, 7) Valider le modèle avec des tests statistiques avancés, et 8) Identifier et exploiter les biais systématiques des bookmakers. Le tout en utilisant principalement Python et des sources de données gratuites ou très économiques, tout en gérant le risque de sur-apprentissage et la volatilité inhérente aux paris.

Étape 1 : Apprendre les Fondations Statistiques

Livrables :

- Document de compréhension :
    - Compréhension des types de variables (qualitatives/quantitatives, continues/discrètes)
    - Mesures de tendance centrale et de dispersion avec exemples concrets des paris sportifs
    - Notion de distribution et de variance appliquée aux cotes
- Tâches pratique : Rédiger une analyse descriptive complète d'un jeu de données de test (50 matchs) incluant moyenne, médiane, écarts-types des côtes.

Outils à utilisés :

- python + libraire math

Budget : 0 €

Durée : 8 h

Étape 2 : Le Langage de l'Incertitude

Livrables :

- Documents de compréhension :
    - Probabilités simples, jointes, conditionnelles
    - Conversion cote → probabilités implicite : P = 1/cote
    - Value bet
- Tâche pratique : Analyser une liste de 100 cotes de 3 bookmakers différents et identifier :
    - Le bookmaker avec la marge moyenne la plus faible
    - Les cotes présentant les écarts les plus importants entre bookmakers
    - Les value bets potentiels sur la base de probabilités historiques simples

Outils à utilisés :

- Odds-API + Python : récupérer les données
- Fichier CSV : Stocker les données

- Python + pandas + numpy : Manipuler les données
- Matplotlib + seaborn : visualiser les données

Budget : 0 €

Durée : 12 h

Étape 3 : Structure des données à collecter sur 10 saisons

Récupérer les informations suivants d'un événement :

- Les cotes [1N2] d'ouverture à fermeture avec (granularité pas ou début)
- Dates et heure exacte de la publications de la côte
- Sports, pays, compétitions
- Résultat final

Outils à utilisé :

- Football.data.org : pour récupérer les données nécessaires
- Fichier CSV: pour stocker les données
- Python + pandas + numpy : Manipuler les données
- Matplotlib + seaborn : visualiser les données

Outils alternatif pour récupérer les données nécessaires :

- Odds API
- Ods Portal
- Betfair Historical Data

Budget : à déterminer

Durée : 20 h

Étape 4 : Régression Logistique Standard

Livrables :

- Documents de compréhension :
    - Principe de la fonction sigmoïde et transformation logit
    - Maximum de vraisemblance pour l'estimation des paramètres
    - Interprétation des coefficients : odds ratios
- Tâches pratique : Construire un modèle prédictif simple :
    - Variable dépendante : le bookmaker a t-il raison ? (1/0)
    - Variables indépendantes pour l'estimation des paramètres
    - Évaluation de la performance avec matrice de confusion et courbe ROC

Outils utilisé :

- Python + scikit-learn
- Python + pandas + numpy : Manipuler les données
- Matplotlib + seaborn : visualiser les données

Étape 5 :

Budget : 0 €

Durée : 16 h

Étape 5 : Régression logistique par Lissage

Livrables :

- document de compréhension :
    - Sur-apprentissage et besoin de régularisation
    - Différence entre pénalités L1 (lasso) et L2 (Ridge)
    - Sélection du paramètre lambda par validation croisée
- Taches pratique : Améliorer le modèle précédent :
    - Implémentation de la régularisation L1 et L2
    - Comparaison des performances avec et sans lissage
    - Identification des variables vraiment importantes

Outils utilisé :

- Python + scikit-learn Python
- pandas + numpy : Manipuler les données
- seaborn : visualiser les données

Budget : 0 €

Durée : 16 h

Étape 6 : Donner de la mémoire au modèle

Livrables :

- Documents de compréhension :
    - Pondération temporelle : concepts de décroissance exponentielle
    - Variables temporelles : jours avant match, saison, moment dans la saison
    - Adaptation des cotes dans le temps : modélisation des mouvements
- Tâche pratique :
    - Création de variables temporelles pertinentes
    - Implémentation de pondérations basées sur la récence
    - Test de différentes fonctions de décroissance (exponentielle, linéaire, etc)

Outils utilisé :

- Python + scikit-learn
- pandas + numpy : Manipuler les données
- Matplotlib + seaborn : visualiser les données

Budget : 0 €

Durée : 14 h

Étape 7 : Tests de diagnostic et Validation

Livrables :

- Documents de compréhension
    - Test de Hosmer-Lemeshow : qualité de l'ajustement du modèle
    - Test de Hausman : stabilité des paramètres dans le temps
    - Intérprétation des p-values et décision statistique
- Tâche pratique :
    - Application des tests de diagnostic
    - Analyse de la stationnarité des relations
    - Validation sur les différentes périodes hors échantillon

Outils à utilisé :

- Python + statsmodels
- pandas + numpy : Manipuler les données
- Matplotlib + seaborn : visualiser les données

Budget : 0 €

Durée : 18 h

Étape 8 : Calibrage des bookmakers

Livrables :

- Documents de compréhension :
    - Biais systématiques des bookmakers
    - Identification des patterns d'inefficacité persistants
    - Différence entre erreur aléatoire et biais structural
- Tâche pratique : analyse de calibration :
    - Comparaison probabilités prédites vs. Résultats observés
    - Identification des situations où les bookmakers se trompent systématiquement
    - Création d'une stratégie de paris basée sur ces inefficacités

Outils à utilisé :

- Python + statsmodels
- pandas + numpy : Manipuler les données
- Matplotlib + seaborn : visualiser les données
- Scrapping de site comme Flashscore ou OddsPortal via Scipt Github : voir ou sont les cotes les plus avantageuses

Budget : 0 €

Durée : 16 h

Rapport technique

1. Régression linéaire simple

La régression linéaire simple est une fonction qui permet de trouver une variable dépendante à partir d'une variable indépendante. La formule de la fonction est :

$$Y = \beta_0 + \beta_1 \times X + \varepsilon$$

Y = La variable dépendante à trouver.

X = La variable indépendante arbitraire.

β_0 = La valeur moyenne de Y si $X = 0$. Cela est interprétable seulement si $X = 0$ est dans le domaine de validité du modèle

β_1 = La valeur de variation moyenne de Y en fonction de X.

ε = L'écart entre la valeur observé Y_i et la valeur prédite $\hat{Y}_i$.

Pour trouver la valeur des coefficients β_0 et β_1, il faut utilisé la méthode du moindre carré.

La méthode du **moindre au carré** trace une courbe (dans notre cas, une ligne droite) au plus près d'un ensemble de points. Les points ici, sont les valeurs X et Y. Cela permet de trouver une estimation des valeurs β, moyenne de Y en fonction de X. β_0 étant donc la valeur moyenne des Y par rapport à $X = 0$. Y est la valeur observables en dehors possiblement de la courbe et $\hat{Y}$ est le point sur la courbe le plus proche de celui-ci. Autrement dit ce tracé est la somme des écarts de tout les Y par rapport au $\hat{Y}$ au carré pour éviter que les écarts s'annulent et donner plus de poids au grandes erreurs de prédiction. Mathématiquement cela se traduit tel que :

$$\min_{\beta_0,\beta_1,\ldots,\beta_k} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2$$

Ainsi les formules pour trouver les valeurs β à partir de la formule de régression simple, sont :

$$\beta_1 = \left[\Sigma\left(X_i - \overline{X}\right)\left(Y_i - \overline{Y}\right)\right] \Big/ \Sigma\left(X_i - \overline{X}\right)^2$$

Et

$$\beta_0 = \overline{Y} - \beta_1 \times \overline{X}$$

$\overline{X}$ = La moyenne des variables de X.

$\overline{Y}$ = La moyenne des variables de Y.

Σ = La somme des observations des données pré-existant.

Avec le résultat de β_1 et β_0, on peut trouver ε. Tel que :

$$\varepsilon_i = Y_i - \hat{Y}_i = Y_i - (\beta_0 + \beta_1 \times X_1)$$

$\hat{Y}_i$ = La valeur prédite avec X_i

Il est à noter que pour utiliser la régression linéaire Les valeurs de Y doivent être présent en totalité pour être prédictible en fonction de X, contrairement à se dernier qui n'a pas besoin de couvrir toutes les valeurs.

Exemple :

L'objectif est de définir le nombre de but des joueurs sur une saison par rapport à leur nombre de matchs, avec des données tel que :

– 5 but pour 10 match
– 5 but pour pour 12 match
– 5 but pour 15 match
– 8 but pour 16 match
– 12 but pour 16 match
– 17 but pour 17 match

Ousman Démbélé blesser cette saison jouera 13 match, je me demande combien de but il marquera ?

X étant 13 match et l'inconnu à prédire étant les buts marqué à l'issue de la saison. Il faut utiliser cette équation :

$$\hat{Y} = \beta_0 + \beta_1 \times X$$

Mais pour cela il faut d'abord définir les variables, on commençons par β_1, dont voici la formule :

$$\beta_1 = \left[\Sigma\left(X_i - \overline{X}\right)\left(Y_i - \overline{Y}\right)\right] \Big/ \Sigma\left(X_i - \overline{X}\right)^2$$

La moyenne de match est de $\frac{10+12+15+16+16+17}{6} = 14{,}33 = \overline{X}$

La moyenne de but est de $\frac{5+5+5+8+12+17}{6} = 8{,}67 = \overline{Y}$

$\left(X_1 - \overline{X}\right)\left(Y_1 - \overline{Y}\right) = (10 - 14{,}33)(5 - 8{,}67) = (-4{,}33)(-3{,}67) = 15{,}89$

$\left(X_2 - \overline{X}\right)\left(Y_2 - \overline{Y}\right) = (12 - 14{,}33)(5 - 8{,}67) = (-2{,}33)(-3{,}67) = 8{,}55$

$\left(X_3 - \overline{X}\right)\left(Y_3 - \overline{Y}\right) = (15 - 14{,}33)(5 - 8{,}67) = (0{,}67)(-3{,}67) = -2{,}56$

$\left(X_4 - \overline{X}\right)\left(Y_4 - \overline{Y}\right) = (16 - 14{,}33)(8 - 8{,}67) = (1{,}67)(-0{,}67) = -1{,}12$

$\left(X_5 - \overline{X}\right)\left(Y_5 - \overline{Y}\right) = (16 - 14{,}33)(12 - 8{,}67) = (0{,}67)(3{,}33) = 2{,}23$

$\left(X_6 - \overline{X}\right)\left(Y_6 - \overline{Y}\right) = (17 - 14{,}33)(17 - 8{,}67) = (2{,}67)(8{,}33) = 22{,}24$

$\Sigma\left(X_i - \overline{X}\right)\left(Y_i - \overline{Y}\right) = 15{,}89 + 8{,}55 + (-2{,}56) + (-1{,}12) + 2{,}23 + 22{,}24 = 45{,}23$

$\left(X_1 - \overline{X}\right)^2 = (10 - 14{,}33)^2 = (-4{,}33)^2 = 18{,}75$

$\left(X_2 - \overline{X}\right)^2 = (12 - 14{,}33)^2 = (-2{,}33)^2 = 5{,}43$

$\left(X_3 - \overline{X}\right)^2 = (15 - 14{,}33)^2 = 0{,}67^2 = 0{,}45$

$\left(X_4 - \overline{X}\right)^2 = (16 - 14{,}33)^2 = (1{,}67)^2 = 2{,}79$

$\left(X_5 - \overline{X}\right)^2 = (16 - 14{,}33)^2 = (1{,}67)^2 = 2{,}79$

$\left(X_6 - \overline{X}\right)^2 = (17 - 14{,}33)^2 = 2{,}67^2 = 7{,}13$

$\Sigma\left(X_i - \overline{X}\right)^2 = 18{,}75 + 5{,}43 + 0{,}45 + 2{,}79 + 2{,}79 + 7{,}13 = 37{,}34$

Alors

$$\beta_1 = 45{,}23/37{,}34 = 1{,}21$$

Ainsi, on peut trouver β_0, tel que :

$$\beta_0 = \overline{Y} - \beta_1 \times \overline{X}\ 8{,}67 = 8{,}67 - (1{,}21 \times 14{,}33) = -8{,}67$$

On peut enfin utiliser la formule de régression linéaire simple tel que :

$$\hat{Y} = \beta_0 + \beta_1 \times X = -8{,}67 + 1{,}21 \times 13 = 7{,}06$$

Donc, avec 13 match d'Oussman Démbélé cette saison, il est estimer selon le modèle qu'il marque 7 but à la fin de saison.

La régression linéaire simple permet d'estimer une variable dépendante à partir d'une variable indépendante, mais dans la réalité plusieurs variables indépendantes définissent la valeur d'une variable dépendante. On reprenons, mon exemple : Un joueur ayant une meilleur qualité de frappes, de placement, une meilleur endurance, un meilleur mental etc, marquera plus de but qu'un joueur ayant un peu moins ces qualités là.

2. La régression linéaire multiple

La régression linéaire multiple permet de prendre en compte tout ces paramètres. La formule est :

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k + \varepsilon$$

Pour que cette formule fonctionne de façon optimal :

- Il faut que les paramètres soit aussi décorrélé les uns des autres que possible pour qu'elle puissent mesurer l'effet des variables toutes choses par ailleurs. Ce qui signifie que les coefficient mesure l'effet d'une variable lorsque les autres sont maintenus constantes. La multicolinéarité entrave cela.
- Le nombre d'observations doit être supérieur au nombre de variables par observations.

Les valeurs des coefficients sont déduites avec cette formule matricielle :

$$\beta = \left(X^T X\right)^{-1} X^T Y$$

X = Une matrice composé des variables explicatives en plus d'une colonne constante de « 1 ». La constante permet de calculer β_0. La taille est de $n \times (k+1)$

Y = Une matrice composé d'une seule colonne (ou vecteur), qui contient les valeurs expliquées

β = Le vecteur (ou matrice à une colonne) des coefficient à estimer

β , est un vecteur contenant des coefficients individuels. Chaque éléments de β correspond à la colonne correspondante de la matrice X : β_0 correspond à la colonne de la constante, β_1 à la première variable X_1, etc.

2.1 Une matrice est un tableau

composé de plusieurs colonnes et lignes de valeurs. Utilisé pour les opérations nécessitant plusieurs résultat. Voici les *opération sur des matrices* :

2.1.1 L'addition entre deux matrices

consiste à additionner chaque éléments d'une matrice par son miroir de l'autre matrice. Ainsi, pour additionner deux matrices il faut qu'ils aient les mêmes dimensions (colonnes et lignes de tailles identiques).

Exemple :

Nous avons deux footballeurs, blesser la quasi-totalité des deux dernières saisons. Pendant ces saisons, ils ont joués chacun quatre matchs, deux par saisons. Il est temps maintenant de se séparer du moins performant. Pour cela, on se basera sur leurs buts. Objectif : à partir des deux tableau représentant chacun le nombre de but par match des deux joueurs, calculer leur nombre de but par saisons. Dans les deux tableaux que nous avons les colonnes représente respectivement les joueurs 1 et 2 et les lignes les matchs 1 et 2. Ainsi :

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$

Où les lignes de la matrice résultat représente respectivement la saison 1 et 2. Donc, pendant la saison 1 le joueur 1 a marqué 6 buts et le 2 en a marqué 8 et la seconde saison, le joueur 1 on a marqué 10 et le 2 on marqué 12. Le joueur 1 est par conséquent remercier de ses services.

2.1.2 La soustraction entre deux matrices

Est le même principe que l'addition, sauf que l'on soustrait

Exemple :

On aimerais connaître si le club n'a pas dépassé son budget alloué à chacun des joueurs pour chaque saison. Pour chaque but marqué la prime est de 1000 euros. Le budget de la première saison et de 50 000 euros pour le joueurs 1 et de 30 000 pour le joueur 2. Et la saison suivante cela s'est inversé. Ainsi

$$\begin{bmatrix} 50\,000 & 30\,000 \\ 30\,000 & 50\,000 \end{bmatrix} - \begin{bmatrix} 6000 & 8000 \\ 10\,000 & 12\,000 \end{bmatrix} = \begin{bmatrix} 44\,000 & 22\,000 \\ 20\,000 & 38\,000 \end{bmatrix}$$

2.1.3 La multiplication $A \times B$

Est possible que si le nombre de colonnes de *A* est égal au nombre de lignes de *B*. Voici la formule de multiplication :

$(A \times B)\,[i,j] = \Sigma\,([i,k] \times B\,[k,j])$ pour k de 1 à n.

i = Numéro d'une ligne en partant de 1

j = Numéro d'une colonne en partant de 1

k = Une variable d'itération, qui parcoure de 1 à n, n étant le nombre de colonne et de ligne qu'on en commun les deux matrices.

Concrètement, **la multiplication matricielle consiste à multiplier chaque ligne de la première matrice par chaque colonne de la seconde**. La matrice résultante possède le même nombre de lignes que la première et le même nombre de colonnes que la seconde. Chaque élément de cette matrice est obtenu en additionnant les produits des éléments correspondants de la ligne de la matrice A et de la colonne de la matrice B.

Il faut aussi savoir que les éléments de la multiplication matricielle ne sont pas commutative.

Exemple :

Nous avons le nombre de buts des trois premiers buteurs global, dans leur championnats et coupe national respectif. Respectivement en championnat et en coupe le joueur 1 a marqué 12 et 2 buts cette saison, le joueur 2 à marqué 15 et 5 buts et le joueur 3 à marqué 36 et 13 buts. Ces joueurs évoluant tous dans des compétions différentes, ont des coefficients par buts différentes. Alors, le joueur 1 qui évolue en Angleterre à un coefficient 1 en championnat et un coefficient 2 en coupe, le joueur 2 évoluant en France à un coefficient 4 en championnat et un coefficient 3 en coupe, enfin le joueur 3 évoluant au Portugal à un coefficient 2 en championnat et 1 en coupe.

L'objectif est de connaître le nombre de points global que les joueurs ont atteint pour chaque saison et compétitions. Ainsi :

$$\begin{bmatrix}12 & 15 & 36\\ 2 & 5 & 13\end{bmatrix} \times \begin{bmatrix}2 & 1\\ 4 & 3\\ 6 & 5\end{bmatrix} = \begin{bmatrix}300 & 237\\ 102 & 82\end{bmatrix}$$

Tel que :

$([i,k] \times B\,[k,j]) = C[1,1] = 12 \times 2 + 15 \times 4 + 36 \times\ 6 = 300$

$([i,k] \times B\,[k,j]) = C\,[1,2] = 12 \times 1 + 15 \times 3 + 36 \times 5 = 237$

$([i,k] \times B\,[k,j]) = C\,[2,1] = 2 \times 2 + 5 \times 4 + 13 \times 6 = 102$

$([i,k] \times B\,[k,j]) = C\,[2,2] = 2 \times 1 + 5 \times 3 + 13 \times 5 = 82$

Ainsi, pendant la première saison les joueurs ont atteint dans leur ensemble 300 points en championnat et 102 en coupe. La deuxième saison, ils ont atteint 237 ont championnat et 82 en coupe. On peut par conséquent constaté une baisse de performance générale.

2.1.4 La multiplication scalaire

Consiste à multiplier tout les éléments d'une matrice par un nombre.

Exemple :

On souhaite obtenir les gains des deux prochains paris de trois parieurs, en sachant que le premier mise 10 et 40 euros, le second 20 et 50 euros et le dernier 30 et 60 euros. La cote pour tout ces paris et de trois. Ainsi :

$$3 \times \begin{bmatrix}10 & 20 & 30\\ 40 & 50 & 60\end{bmatrix} = \begin{bmatrix}30 & 60 & 90\\ 120 & 150 & 180\end{bmatrix}$$

Donc, le parieur 1 gagnera potentiellement 30 et 120 euros, le second 60 et 150 euros, et le troisième 90 et 180 euros.

2.1.5 La transposition

Consiste à échanger (ou plutôt inverser) les lignes et les colonnes d'une même matrice. Pour une matrice X, sa transposition s'écrit : X^T

Exemple :

J'ai une représentation de trois parieurs par leurs deux dernières mises. L'objectif est de changer de perspective pour avoir une représentation de l'avant dernier et du dernier paris sur les colonnes, au lieu d'une colonne représentant les parieurs, tel que :

$$X = \begin{bmatrix}30 & 60 & 90\\ 120 & 150 & 180\end{bmatrix} \rightarrow \begin{bmatrix}30 & 120\\ 60 & 150\\ 90 & 180\end{bmatrix} = X^T$$

2.1.6 L'inversion

Ne peut se produire que si la matrice est au carré. L'inverse d'une matrice Aest la matrice A^{-1}, tel que $A^{-1} \times A = I$. La formule générale pour trouver l'inverse d'une matrice est :

$$A^{-1} = (1/|A|) \times C^T$$

Où :

C^T = Transposition de la matrice des cofacteurs :

Pour une matrice carré A de taille $n \times n$, la matrice des cofacteurs C est une matrice de même dimension où chaque élément $C[i,j]$ est le cofacteur de l'élément $A[i,j]$.

La formule du cofacteur est :

$$C\,[i,j] = (-1)^{i+j} \times |M[i,j]|$$

Où :

$M\,[i,j] =$ Matrice de la sous matrice obtenue en supprimant la ligne i et la colonne j. À noter que par convention le déterminant d'une matrice vide est toujours égal à 1.

$(-1)^{i+j}$ = Signe alterné dépendant de l'addition $i + j$, si c'est pair +1 et si impair c'est -1

$|A|$ ou $det(A)$ est le déterminant de la matrice. Plusieurs formules en fonction du déterminant de la matrice :

Pour une matrice 1×1, tel que $A = [a]$, la formule est : $|A| = a$.

Pour une matrice 2×2, tel que $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, la formule est : $|A| = a \times d - b \times c$.

Pour une matrice 3×3, tel que $A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$, la formule est :

$$|A| = a\,(ei - fh) - b\,(di - fg) + c(dh - eg)$$

Pour les matrices $A = n \times n$ et $4 \leq n \geq 6$, on utilise le développement par cofacteurs (ou le développement de Laplace) et pour $6 < n$, on utilise la méthode d'élimination de Gauss.

La formule du développement par cofacteurs est :

$$|A| = \Sigma(-1)^{i+j} \times a_{ij} \times |M_{ij}|$$

$a_{ij} =$ élément à la ligne i et à la colonne j.

$M_{ij} =$ Matrice mineure obtenue en supprimant la ligne i et la colonne j.

$(-1)^{i+j} =$ Signe de la position (i,j). $+1$ si (i,j) est pair, le signe est $+$et -1 si impair, le signe est $-$.

Info : En l'absence de crochet, le signe Σ s'applique à toute l'expression qui le suit jusqu'a rencontrer un signe d'addition ou de soustraction.

La formule de la méthode par élimination de Gauss est :

$$|A| = det \times \prod A[i,i]$$

$\prod A[i,i]$ = Produit des éléments de la diagonale de la matrice triangulaire obtenue. Cela signifie qu'une fois la matrice rendue triangulaire, il suffit de multiplier entre eux, tous les nombres situé sur sa diagonale principale, tel que l'élément de la $L_1 = (1,1)$ et l'élément de la $L_2 = (2,2)$. Donc : $\prod A[i,i] = \prod A[1,1] \times [2,2] \times \ldots \times [n,n]$.

$\det$ = Facteur correctif qui accumule toutes les modifications du déterminant durant les opérations qui vont suivre.

Voici le processus pour obtenir c'est deux éléments :

Étape 1 : Initialisation

Soit A une matrice carré $n \times n$

Soit $det = 1$(la valeur initiale du déterminant)

Étape 2 : Triangulation

Pour chaque colonne de k de 1 à $n - 1$:

1. Trouver le pivot, qui est l'élément idéal : non nul, le plus stable numériquement (c-à-d facilement exploitable dans une opération), et le plus grand dans la colonne k.
2. Échanger de lignes si nécessaire, tel que $L_1 \leftrightarrow L_2$, pour trouver le pivot (cela se traduira par un changement de signe du déterminant)
3. Créer des zéros sous chaque pivots pour obtenir une matrice triangulaire. Pour cela :
    - Pour chaque ligne $i > k$: diviser le point $A[i,k]$ par le point [k,k], tel que $m = A[i,k] / A[k,k]$.
    - Puis multiplier les éléments de la ligne par le résultat et soustraire le tout de la ligne actuelle, tel que : $L_i - m \times L_k$
    - Enfin, remplacer la ligne par les nouveaux éléments

Pour vérifier si deux matrice sont inverse, le produit de ces matrices doit donné la matrice identité qui est égale à 1, tel que :

$$A \times A^{-1} = [1]$$

Cela permet de résoudre des équations de matrice, avec une matrice inconnu X et une matrice B, tel que $A \times X = B$ avec $A^{-1} \times B = X$.

Il faut savoir qu'une matrice carré A est inversible si et seulement si $|A| \neq 0$.

Exemples pour une matrice 1×1 :

Nous avons un joueur qui à marqué 5 buts dans un championnat inconnu, mais a seulement 1 points pour le soulier d'or. On aimerais donc savoir quelle est le coefficient par buts de se championnat. Ainsi :

$A = [5]$

$|A| = 5$

$C^T = (-1)^{i+j} \times |M[i,j]| = 1 \times 1 = [1]$

$$A^{-1} = (1/|A|) \times C^T = (1/5) \times 1 = [0{,}2]$$

Donc, le coefficient dans ce championnat et de 0,2 par buts.

Exemple pour une matrice 2×2 :

$$\begin{bmatrix} 10 & 30 \\ 100 & 200 \end{bmatrix} \times X = \begin{bmatrix} 50 & 150 \\ 300 & 900 \end{bmatrix}$$

Je dois donc trouver la matrice inconnu, qui est égale à A^{-1}, tel que :

$A^{-1} = (1/|A|) \times C^T$

$|A| = a \times d - b \times c = 10 \times 200 - 30 \times 100 = 2000 - 3000 = -1000$

Et C^T la matrice ayant chaque éléments tel que :

$C[i,j] = ((-1))^{i+j} \times |M[i,j]| = C[1,1] = (+1) \times |[10]| = 200$

$C[i,j] = ((-1))^{i+j} \times |M[i,j]| = C[1,2] = (-1) \times |[100]| = -30$

$C[i,j] = ((-1))^{i+j} \times |M[i,j]| = C[2,1] = (-1) \times |[30]| = -100$

$C[i,j] = ((-1))^{i+j} \times |M[i,j]| = C[2,2] = (+1) \times |[200]| = 10$

Est égale à :

$C = \begin{bmatrix} 200 & -100 \\ -30 & 10 \end{bmatrix}$

Et $C^T = \begin{bmatrix} 200 & -30 \\ -100 & 10 \end{bmatrix}$

Ainsi, $A^{-1} = (1/|A|) \times C^T = \left(\frac{1}{-1000}\right) \times \begin{bmatrix} 200 & -30 \\ -100 & 10 \end{bmatrix} = \begin{bmatrix} -0,2 & 0,03 \\ 0,1 & -0,01 \end{bmatrix}$

Soit :

$X[1,1] = (-0,2) \times 50 + 0,03 \times 300 = -1$

$= X[1,2] (-0,2) \times 150 + 0,03 \times 900 = -3$

$X[2,1] = 0,1 \times 50 + (-0,01) \times 300 = 2$

$X[2,2] = 0,1 \times 150 + (-0,01) \times 900 = 6$

Soit, $A^{-1} \times B = X = \begin{bmatrix} -0,2 & 0,03 \\ 0,1 & -0,01 \end{bmatrix} \times \begin{bmatrix} 50 & 150 \\ 300 & 900 \end{bmatrix} = \begin{bmatrix} -1 & -3 \\ 2 & 6 \end{bmatrix}$

Soit $\begin{bmatrix} 10 & 30 \\ 100 & 200 \end{bmatrix} \times \begin{bmatrix} -1 & -3 \\ 2 & 6 \end{bmatrix} = B$

Ainsi B est égale à :

$X[1,1] = 10 \times (-1) + 30 \times 2 = 50$

$X[1,2] = 10 \times (-3) + 30 \times 6 = 150$

$X[2,1] = 100 \times (-1) + 200 \times 2 = 300$

$X[2,2] = 100 \times (-3) + 200 \times 6 = 900$

Donc,

$$\begin{bmatrix} 10 & 30 \\ 100 & 200 \end{bmatrix} \times \begin{bmatrix} -1 & -3 \\ 2 & 6 \end{bmatrix} = \begin{bmatrix} 50 & 150 \\ 300 & 900 \end{bmatrix}$$

Exemple de calcule de régression linéaire multiple :

L'objectif est de trouver le nombre de buts d'Ousmane Démbélé, qui à un niveau de 8 cette saison et jouera 14 matchs.

Autrement dit il faut définir le nombre de but des joueurs sur une saison non seulement par rapport à leur nombre de match mais aussi par rapport à leur niveau, avec des données tel que :

– 5 but pour 10 match avec un niveau 1
– 5 but pour pour 12 match avec un niveau 3
– 5 but pour 15 match avec un niveau 4
– 8 but pour 16 match avec un niveau 4
– 12 but pour 16 match avec un niveau 7
– 17 but pour 17 match avec un niveau 10

L'objectif est de trouver $\hat{Y}$, tel que $\hat{Y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k$

Il faut d'abord trouver le vecteur β,tel que $\beta = \left(X^T X\right)^{-1} X^T Y$:

J'ai déjà quelques matrices tel que :

$$X = \begin{bmatrix} 1 & 10 & 1 \\ 1 & 12 & 3 \\ 1 & 15 & 4 \\ 1 & 16 & 4 \\ 1 & 16 & 7 \\ 1 & 17 & 10 \end{bmatrix} ; \; X^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 10 & 12 & 15 & 16 & 16 & 17 \\ 1 & 3 & 4 & 4 & 7 & 10 \end{bmatrix} \text{ et } Y = \begin{bmatrix} 5 \\ 5 \\ 5 \\ 8 \\ 12 \\ 17 \end{bmatrix}$$

Je commence par trouver A tel que : $X^T X = A$

$A[1,1] = 1 \times 1 + 1 \times 1 + 1 \times 1 + 1 \times 1 + 1 \times 1 + 1 \times 1 = 6$

$A[1,2] = 1 \times 10 + 1 \times 12 + 1 \times 15 + 1 \times 16 + 1 \times 16 + 1 \times 17 = 86$

$A[1,3] = 1 \times 1 + 1 \times 3 + 1 \times 4 + 1 \times 4 + 1 \times 7 + 1 \times 10 = 29$

$A[2,1] = 1 \times 10 + 1 \times 12 + 1 \times 15 + 1 \times 16 + 1 \times 16 + 1 \times 17 = 86$

$A[2,2] = 10 \times 10 + 12 \times 12 + 15 \times 15 + 16 \times 16 + 16 \times 16 + 17 \times 17 = 1270$

$A[2,3] = 10 \times 1 + 12 \times 3 + 15 \times 4 + 16 \times 4 + 16 \times 7 + 17 \times 10 = 452$

$A[3,1] = 1 \times 1 + 3 \times 1 + 4 \times 1 + 4 \times 1 + 7 \times 1 + 10 \times 1 = 29$

$A[3,2] = 10 \times 1 + 12 \times 3 + 15 \times 4 + 16 \times 4 + 16 \times 7 + 17 \times 10 = 452$

$A[3,3] = 1 \times 1 + 3 \times 3 + 4 \times 4 + 4 \times 4 + 7 \times 7 + 10 \times 10 = 191$

$$A = \begin{bmatrix} 6 & 86 & 29 \\ 86 & 1270 & 452 \\ 29 & 452 & 191 \end{bmatrix}$$

Maintenant, il faut trouver l'inverse de cette matrice, tel que $A^{-1} = (1/|A|) \times C^T = \left(X^T X\right)^{-1}$. La méthode la plus adapté pour trouver le déterminant $|A|$ est :

$$|A| = a(ei - fh) - b(di - fg) + c(dh - eg)$$

$$|A| = 6(1270 \times 191 - 452 \times 452) - 86(86 \times 191 - 452 \times 29) + 29(86 \times 452 - 1270 \times 29)$$

$$|A| = 6(38266) - 86(3318) + 29(2042) = 229596 - 285348 + 59218 = 2866$$

Ensuite, il faut trouver la matrice cofacteur C, tel que chaque élément : $C[i,j] = (-1)^{i+j} \times |M[i,j]|$. En sachant que M sera de dimension 2×2, la formule pour trouver son déterminant est :

$$(a \times d - b \times c)$$

Ainsi :

$$C[1,1] = (+1) \times \left|\begin{bmatrix} 1270 & 452 \\ 452 & 191 \end{bmatrix}\right| = 1270 \times 191 - 452 \times 452 = 38266$$

$$C[1,2] = (-1) \times \left|\begin{bmatrix} 86 & 452 \\ 29 & 191 \end{bmatrix}\right| = 86 \times 191 - 452 \times 29 = -3318$$

$$C[1,3] = (+1) \times \left|\begin{bmatrix} 86 & 1270 \\ 29 & 452 \end{bmatrix}\right| = 86 \times 452 - 1270 \times 29 = 2042$$

$$C[2,1] = (-1) \times \left|\begin{bmatrix} 86 & 29 \\ 452 & 191 \end{bmatrix}\right| = 86 \times 191 - 29 \times 452 = -3318$$

$$C[2,2] = (+1) \times \left|\begin{bmatrix} 6 & 29 \\ 29 & 191 \end{bmatrix}\right| = 6 \times 191 - 29 \times 29 = 305$$

$$C[2,3] = (-1) \times \left|\begin{bmatrix} 6 & 86 \\ 29 & 452 \end{bmatrix}\right| = 6 \times 452 - 86 \times 29 = -218$$

$$C[3,1] = (+1) \times \left|\begin{bmatrix} 86 & 29 \\ 1270 & 452 \end{bmatrix}\right| = 86 \times 452 - 29 \times 1270 = 2042$$

$$C[3,2] = (-1) \times \left|\begin{bmatrix} 6 & 29 \\ 86 & 452 \end{bmatrix}\right| = 6 \times 452 - 29 \times 86 = -218$$

$$C[3,3] = (+1) \times \left|\begin{bmatrix} 6 & 86 \\ 86 & 1270 \end{bmatrix}\right| = 6 \times 1270 - 86 \times 86 = 224$$

Ainsi

$$C = \begin{bmatrix} 38266 & -3318 & 2042 \\ -3318 & 305 & -218 \\ 2042 & -218 & 224 \end{bmatrix}$$

$$C^T = \begin{bmatrix} 38266 & -3318 & 2042 \\ -3318 & 305 & -218 \\ 2042 & -218 & 224 \end{bmatrix}$$

Donc,

$$A^{-1} = (1/|A|) \times C^T = (1/(2866)) \times \begin{bmatrix} 38266 & -3318 & 242 \\ -3318 & 305 & -218 \\ 2042 & -218 & 224 \end{bmatrix}$$

$$A^{-1} = 0{,}00035 \times \begin{bmatrix} 38266 & -3318 & 2042 \\ -3318 & 305 & -218 \\ 2042 & -218 & 224 \end{bmatrix} = \begin{bmatrix} 13{,}3931 & -1{,}1613 & 0{,}7147 \\ -1{,}1613 & 0{,}10675 & -0{,}0763 \\ 0{,}7147 & -0{,}0763 & 0{,}0784 \end{bmatrix}$$

$$\left(X^T X\right)^{-1} = \begin{bmatrix} 13{,}3931 & -1{,}1613 & 0{,}7147 \\ -1{,}1613 & 0{,}10675 & -0{,}0763 \\ 0{,}7147 & -0{,}0763 & 0{,}0784 \end{bmatrix}$$

Il faut maintenant que je trouve $X^T Y$, tel que

$$X^T Y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 10 & 12 & 15 & 16 & 16 & 17 \\ 1 & 3 & 4 & 4 & 7 & 10 \end{bmatrix} \times \begin{bmatrix} 5 \\ 5 \\ 5 \\ 8 \\ 12 \\ 17 \end{bmatrix} = B$$

Ainsi :

$$B\,[1{,}1] = 1 \times 5 + 1 \times 5 + 1 \times 5 + 1 \times 8 + 1 \times 12 + 1 \times 17 = 52$$

$$B\,[2{,}1] = 10 \times 5 + 12 \times 5 + 15 \times 5 + 16 \times 8 + 16 \times 12 + 17 \times 17 = 794$$

$$B\,[3{,}1] = 1 \times 5 + 3 \times 5 + 4 \times 5 + 4 \times 8 + 7 \times 12 + 10 \times 17 = 326$$

$$X^T Y = B = \begin{bmatrix} 52 \\ 794 \\ 326 \end{bmatrix}$$

Donc,

$$\beta = \left(X^T X\right)^{-1} X^T Y = \begin{bmatrix} 13{,}3931 & -1{,}1613 & 0{,}7147 \\ -1{,}1613 & 0{,}10675 & -0{,}0763 \\ 0{,}7147 & -0{,}0763 & 0{,}0784 \end{bmatrix} \times \begin{bmatrix} 52 \\ 794 \\ 326 \end{bmatrix}$$

Ainsi :

$$\beta\,[1{,}1] = 13{,}3931 \times 52 + (-1{,}1613) \times 794 + 0{,}7147 \times 326 = 8.2312$$

$$\beta\,[2{,}1] = (-1{,}1613) \times 52 + 0{,}10675 \times 794 + (-0{,}0763) \times 326 = -0.527$$

$$\beta\,[3{,}1] = 0{,}7147 \times 52 + (-0{,}0763) \times 794 + 0{,}0784 \times 326 = 2.1506$$

$$\beta = \begin{bmatrix} 8.2312 \\ -0{,}527 \\ 2.1506 \end{bmatrix}$$

Il faut donc maintenant trouver $\hat{Y}$, tel que :

$$\hat{Y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 = 8{,}2312 + (-0{,}527) \times 14 + 2{,}1506 \times 8 = 18{,}058$$

Donc, le modèle prédit que Dembélé marquera 18 buts cette saison au vu du nombre de match qu'il jouera et de son niveau moyen.

Mais il est possible qu'il est une baisse de performance, alors :

$$\hat{Y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 = 8{,}2312 + (-0{,}527) \times 14 + 2{,}1506 \times 5 = 11{,}60$$

Donc, si son niveau passe à 5 alors, il devrait marqué 11 buts

Cependant, cela prédit un résultat absolu sans ambiguïté. Alors qu'en réalité la plupart des prédiction son à nuancé. La régression logistique classique apporte cette nuance via une prédiction probabiliste entre les cas possibles.

3. Régression logistique standard (binaire)

Pour contraindre la sortie entre 0 et 1, il faut utiliser une fonction « en S » appelé **sigmoïde ou régression logistique**. Cette fonction transforme n'importe quel nombre réel en une valeur comprise entre 0 et 1. Voici la fonction qui permet cela :

$$P(Y = 1) = \frac{1}{\left(1 + e^{(-z)}\right)}$$

Tel que :

$P(Y = 1)$, la probabilité que l'événement se produisent (classe 1).

e, la base du logarithme naturel

z, la variable dépendante linéaire provenant du modèle de régression, appelée score linéaire : $z = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k$

Ainsi, plus z est proche de $+\infty$, plus $\left(1 + e^{(-z)}\right)$ est proche de 0 et $P(Y = 1)$ proche de 1. Donc l'événement est plus probable de ce produire. Et au contraire, plus z est proche de $-\infty$, plus $\left(1 + e^{(-z)}\right)$ est grand et $P(Y = 1)$ proche de 0, alors l'événement est moins probable. Enfin, si $z = 0$ alors $P\,(Y = 1) = 0{,}5$.

Il est possible de retrouver le score linéaire à partir de la variable probabiliste de la fonction sigmoïde via sa transformation inverse, appelé le Logit. Ça formule est : $logit(P) = ln\left(\frac{P}{1-P}\right) = z$.

Cependant, On n'utilise pas les moindres carrés en régression logistique car cette méthode à pour but de trouver les points les plus courts entre les valeurs prédites et les valeurs réelles expliqués. Alors que l'objectif pour dans une régression linéaire standard est de trouver le chemin le plus court entre les probabilités réelles et les probabilité prédites, qui sont compris entre 0 et 1. Cela est possible avec le maximum de vraisemblance.

Le maximum de vraisemblance (ou MLE en abv. anglais) est une fonction tel que :

$$\mathrm{L}\,(\beta) = \prod P(i)^{y_i} \times (1 - P(i))^{(1-y_i)}$$

Où

$P\,(i) = 1/(1+)e^{-z}$

Pour rappelle, $z = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k$

Toutefois, comme on peut le remarquer, pour trouver les β à partir de la fonction MLE, il faut déjà avoir trouver ces même β pour utiliser la fonction car ils composent z, qui compose $P\,(i)$. Ainsi, pour résoudre ce paradoxe, il faut utilisé une méthode d'optimisation numérique. Les plus courant de ces méthodes sont la descente de gradient et la méthode de Newton-Raphson.

Trouver les coefficients avec la méthode de la descente de gradient

Pour cela, il y a plusieurs étapes :

– 1. Initialiser les coefficients avec des valeurs arbitraires (valeurs nulles ou aléatoire).
– 2. Calcul du gradient, noté ∇ de la fonction de coût, notée J, par rapport au coefficients noté β, tel que :

$$\nabla J(\beta) = \frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)X_i$$

Où,

N est le nombre d'observations

X_i Est le vecteur des variables explicatives pour la i-ème observation. Soit une matrice contenant toutes les caractéristiques d'une observations (hormis variables expliquées) Ne surtout pas oublier l'intercept dans la matrice, qui est la constante pour chaque observation.

y_i Est la variable expliqué de la i-ème observation

p_i Est la probabilité estimé par le modèle sigmoïde pour que la variable $y_i = 1$. Tel que $p_i = \frac{1}{\left(1+e^{(-z)}\right)}$

– 3. Mettre à jour les coefficients tel que :

$$\beta^{(t+1)} = \beta^{(t)} - \eta \times \nabla J\left(\beta^{(t)}\right)$$

Où,

$\beta^{(t)}$ Est l'ancienne valeur (à l'itération *(t)*)

$\beta^{(t+1)}$ Est la nouvelle valeur (après mise à jour)

η Est une valeur entre 0 et 1, qui définit la distance parcouru entre chaque itérations de mise à jour. Si c'est trop petit, il faudra beaucoup d'itérations pour trouver le point et au contraire, si trop grand le point pourrait être sauté. Donc, pour trouver la valeur idéal, il n'y a pas de méthode précise, il faut tester plusieurs valeurs et observer les performances. Pour savoir si c'est le bon, la courbe doit descendre de façon régulière.

Exemple avec la méthode de descente de gradient :

Nous sommes à la 10ème journée de championnat de Ligue 1. Pour chaque match nous avons mesurer trois variables : le moral, la forme physique de l'équipe et la tactique adopté. Leurs valeurs sont compris entre 1 (catastrophique) et 10 (excellent). Voici les résultats des dix derniers matchs du PSG avec les variables expliquées :

1 – moral = 8 ; forme = 4 ; tactique = 6 ; résultat=défaite

2 – moral = 3 ; forme = 3 ; tactique = 7 ; résultat=défaite

3 – moral = 2 ; forme = 6 ; tactique = 8 ; résultat=nul

4 – moral = 3 ; forme = 7 ; tactique = 8 ; résultat=victoire

5 – moral = 6 ; forme =8 ; tactique = 8 ; résultat=victoire

6 – moral = 8 ; forme = 8 ; tactique = 7 ; résultat=victoire

7 – moral = 9 ; forme = 7 ; tactique = 7 ; résultat=victoire

8 – moral = 10 ; forme = 7 ; tactique = 8 ; résultat=victoire

9 – moral = 8 ; forme = 4 ; tactique = 6 ; résultat=nul

10 – moral = 5 ; forme = 6 ; tactique = 8 ; résultat=nul

Soit 2 défaite, 3 nul et 5 victoire.

L'objectif est de connaître la probabilité de victoire du PSG pour son 11ème match, en sachant que le moral de l'équipe est passé à 4, la forme est quant à elle, passer à 7 et enfin la tactique est stable par rapport au dernier match.

Pour cela je doit d'abord commencer par trouver les coefficients idéaux, tel que (toute victoire à comme résultat 1 et tout autre résultat est égale à 0) :

En débutons par l'Initialisation de la valeur des coefficients :

$\beta_0 = 0$

$\beta_1 = 0$

$\beta_2 = 0$

$\beta_3 = 0$

Ensuite, définir les valeurs des X_i, y_i, p_i et $(p_1 - y_1)\, X_i$, (en ajoutant toujours une constante 1 comme première valeur du vecteur pour ajuster, la probabilité), tel que :

$X_1 = [1 \quad 8 \quad 4 \quad 6]$

$y_1 = 0$

$p_1 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(0+0\times8+0\times3+0\times6)}\right)} = 0{,}5$

$(p_1 - y_1)\, X_1 = (0{,}5 - 0) \times [1 \quad 8 \quad 4 \quad 6] = [0{,}5 \quad 4 \quad 2 \quad 3]$

$X_2 = [1 \quad 3 \quad 3 \quad 7]$

$y_2 = 0$

$p_2 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(0+0\times3+0\times3+0\times7)}\right)} = 0{,}5$

$(p_2 - y_2)\, X_2 = (0{,}5 - 0) \times [1 \quad 3 \quad 3 \quad 7] = [0{,}5 \quad 1{,}5 \quad 1{,}5 \quad 3{,}5]$

$X_3 = [1 \quad 2 \quad 6 \quad 8]$

$y_3 = 0$

$p_3 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(0+0\times2+0\times6+0\times8)}\right)} = 0{,}5$

$\left(p_3 - y_3\right) X_3 = (0{,}5 - 0) \times [1 \quad 2 \quad 6 \quad 8] = [0{,}5 \quad 1 \quad 3 \quad 4]$

$X_4 = [1 \quad 3 \quad 7 \quad 8]$

$y_4 = 1$

$p_4 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(0+0\times3+0\times7+0\times8)}\right)} = 0{,}5$

$(p_4 - y_4)\, X_4 = (0{,}5 - 1) \times [1 \quad 3 \quad 7 \quad 8] = [-0{,}5 \quad -1{,}5 \quad -3{,}5 \quad -4]$

$X_5 = [1 \quad 6 \quad 8 \quad 8]$

$y_5 = 1$

$p_5 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(0+0\times6+0\times8+0\times8)}\right)} = 0{,}5$

$\left(p_5 - y_5\right) X_5 = (0{,}5 - 1) \times [1 \quad 6 \quad 8 \quad 8] = [-0{,}5 \quad -3 \quad -4 \quad -4]$

$X_6 = [1 \quad 8 \quad 8 \quad 7]$

$y_6 = 1$

$p_6 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(0+0\times8+0\times8+0\times7)}\right)} = 0{,}5$

$\left(p_6 - y_6\right) X_6 = (0{,}5 - 1) \times [1 \quad 8 \quad 8 \quad 7] = [-0{,}5 \quad -4 \quad -4 \quad -3{,}5]$

$X_7 = [1 \quad 9 \quad 7 \quad 7]$

$y_7 = 1$

$p_7 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(0+0\times9+0\times7+0\times7)}\right)} = 0{,}5$

$\left(p_7 - y_7\right) X_7 = (0{,}5 - 1) \times [1 \quad 9 \quad 7 \quad 7] = [-0{,}5 \quad -4{,}5 \quad -3{,}5 \quad -3{,}5]$

$X_8 = [1 \quad 10 \quad 7 \quad 8]$

$y_8 = 1$

$p_8 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(0+0\times10+0\times7+0\times8)}\right)} = 0{,}5$

$\left(p_8 - y_8\right) X_8 = (0{,}5 - 1) \times [[1 \quad 10 \quad 7 \quad 8]] = [-0{,}5 \quad -5 \quad -3{,}5 \quad -4]$

$X_9 = [1 \quad 8 \quad 4 \quad 6]$

$y_9 = 0$

$p_9 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(0+0\times8+0\times4+0\times6)}\right)} = 0{,}5$

$\left(p_9 - y_9\right) X_9 = (0{,}5 - 0) \times [1 \quad 8 \quad 4 \quad 6] = [0{,}5 \quad 4 \quad 2 \quad 3]$

$X_{10} = [1 \quad 5 \quad 6 \quad 8]$

$y_{10} = 0$

$$p_{10} = \frac{1}{\left(1+e^{(-z)}\right)} == \frac{1}{\left(1+e^{(0+0\times5+0\times6+0\times8)}\right)} = 0{,}5$$

$$\left(p_{10} - y_{10}\right) X_{10} = (0{,}5 - 0) \times [1 \quad 5 \quad 6 \quad 8] = [0{,}5 \quad 2{,}5 \quad 3 \quad 4]$$

Puis trouver la somme des $(p_1 - y_1)\, X_1$, tel que :

$$\sum_{i=1}^{10} (p_i - y_i) X_i = [0{,}5 \quad 4 \quad 2 \quad 3] + [0{,}5 \quad 1{,}5 \quad 1{,}5 \quad 3{,}5] + [0{,}5 \quad 1 \quad 3 \quad 4] + [-0{,}5 \quad -1{,}5 \quad -3{,}5 \quad -4] + [-0{,}5 \quad -3 \quad -4 \quad -4] + [-0{,}5 \quad -4 \quad -4 \quad -3{,}5] + [-0{,}5 \quad -4{,}5 \quad -3{,}5 \quad -3{,}5] + [-0{,}5 \quad -5 \quad -3{,}5 \quad -4] + [0{,}5 \quad 4 \quad 2 \quad 3] + [0{,}5 \quad 2{,}5 \quad 3 \quad 4] = [0 \quad -5 \quad -7 \quad -1{,}5]$$

$$\mathrm{J}\nabla(\beta) = \frac{1}{N}\sum_{i=1}^{N} \left(p_i - y_i\right) X_i = 0{,}1 \times [0 \quad -5 \quad -7 \quad -1{,}5] = [0 \quad -0{,}5 \quad -0{,}7 \quad -0{,}15]$$

Ensuite, mettre à jour le gradient de la fonction de coût par rapport au coefficient, en commençons avec les valeurs initialisations du vecteur des coefficients actuels , tel que :

$$\beta^{(1)} = \beta^0 - \eta \times \nabla \mathrm{J}\left(\beta^0\right) = [0 \quad 0 \quad 0 \quad 0] - 0{,}1 \times [0 \quad -0{,}5 \quad -0{,}7 \quad -0{,}15]$$

$$\beta^{(1)} = [0 \quad 0{,}05 \quad 0{,}07 \quad 0{,}015]$$

Puis recalculer le gradient, on commençons par p_i, tel que :

$$p_1 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(-0{,}77)}\right)} = 0{,}684$$

$$p_2 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(-0{,}465)}\right)} = 0{,}614$$

$$p_3 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(-0{,}64)}\right)} = 0{,}655$$

$$p_4 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(-0{,}76)}\right)} = 0{,}681$$

$$p_5 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(-0{,}98)}\right)} = 0{,}727$$

$$p_6 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(-1{,}065)}\right)} = 0{,}744$$

$$p_7 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(-1{,}045)}\right)} = 0{,}739$$

$$p_8 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(-1{,}11)}\right)} = 0{,}752$$

$$p_9 = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(-0{,}77)}\right)} = 0{,}684$$

$$p_{10} = \frac{1}{\left(1+e^{(-z)}\right)} = \frac{1}{\left(1+e^{(-0{,}79)}\right)} = 0{,}688$$

En passant par la somme des $(p_i - y_i)X_i$, tel que :

$\sum_{i=1}^{10}(p_i - y_i)X_i = [2{,}968 \quad 8{,}104 \quad 5{,}249 \quad 13{,}84]$

Et trouver les nouveaux gradients, tel que :

$\nabla J(\beta) = \frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)X_i = \frac{1}{10} \times [2{,}968 \quad 8{,}104 \quad 5{,}249 \quad 13{,}84] = [0{,}2968 \quad 0{,}8104 \quad 0{,}5249 \quad 1{,}384]$

Et enfin, calculer les coefficients :

$\beta^{(2)} = \beta^1 - \eta \times \nabla J(\beta^1) = [0 \quad 0{,}05 \quad 0{,}07 \quad 0{,}015] - 0{,}10 \times [0{,}2968 \quad 0{,}8104 \quad 0{,}5249 \quad 1{,}384]$

$\beta^{(2)} = [0 \quad 0{,}05 \quad 0{,}07 \quad 0{,}015] - [0{,}2968 \quad 0{,}8104 \quad 0{,}5249 \quad 1{,}384]$

Le coût augmente à la multiplication par η, donc on doit modifier celui-ci au profit d'un nombre plus petit, tel que :

$\beta^{(2)} = \beta^1 - \eta \times \nabla J(\beta^1) = [0 \quad -0{,}5 \quad -0{,}7 \quad -0{,}15] - 0{,}01 \times [0{,}2968 \quad 0{,}8104 \quad 0{,}5249 \quad 1{,}384]$

$\beta^{(2)} = [0 \quad 0{,}05 \quad 0{,}07 \quad 0{,}015] - [0{,}002968 \quad 0{,}008104 \quad 0{,}005249 \quad 0{,}01384]$

$\beta^{(2)} = [0{,}002968 \quad 0{,}041896 \quad 0{,}064751 \quad 0{,}00116]$

Je décide de m'arrêter là car j'estime que pour un exemple le nombre de pas est déjà suffisamment bas.

Donc, je peux maintenant estimer la probabilité que le PSG à de gagner son prochain match, tel que :

$z = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 = 0{,}002968 + 0{,}041896 \times 4 + 0{,}064751 \times 7 + 0{,}00116 \times 8 = 0{,}6321$

$$P(1) = \frac{1}{1 + e^{(-0{,}6321)}} = 0{,}6525$$

Ainsi, pour conclure le PSG à 65,25% de chance de remporter son prochain match selon le modèle.

L'objectif de la méthode de Newton-Raphson

La méthode de Newton-Raphson sert à trouver des coefficients qui minimise la négative de la log vraisemblance, (la fonction de coût est noté J) plus rapidement que le gradient, tel que :

$$J(\beta) = -\sum_{i=1}^{n}\left[y_i log\left(p_i\right) + (1 - y_i) log(1 - p_i)\right]$$

Où

$p_i = \sigma(z_i) = \frac{1}{1+e^{-z_i}}$ Est la probabilité prédite

$z_i = \beta^T x_i$ Est le score linéaire pour l'observation i, autrement dit, deux vecteurs, contenant respectivement les coefficients et les variables explicatives.

y_i Est la valeur expliqué compris dans l'intervalle [0;1]

Les étapes pour trouver les coefficients avec la méthode de Newton-Raphson sont :

– 1. Une initialisation : comme pour la méthode de descente de gradient
– 2. Calcule du gradient et de la Hessienne :

2.1 Deux méthodes équivalent pour calculer le gradient :

Pour trouver le gradient du coefficient β_j :

$$\frac{\partial J}{\partial \beta_j} = -\sum_{\mathrm{i}=1}^{\mathrm{n}} (y_i - p_i)\, x_{ij}$$

Où :

y_i, x_i et p_i signifient la même chose que dans la formule de la fonction de coût,

x_{ij} est un élément de la matrice X correspondant au coefficient β_j.

X est la matrice de dimension $n \times p$, tel que n le nombre d'observations de l'expérience et p le nombre de variables explicatives pour chaque observations. (ce p ne correspond pas du tout à ce qui suit)

Pour trouver tout le gradient rapidement de tout les coefficients, sous forme vectorielle :

$$\nabla \mathrm{J}\,(\beta) = X^T (p - y)$$

Où :

X est la matrice identiques que dans la première méthode.

$p = [p_1 \quad p_2 \quad \dots \quad p_n]^T$ Est le transposé des probabilités prédites

$y = [y_1 \quad y_2 \quad \dots \quad y_n]^T$ Est le vecteur des variables expliquées

2.2 Calculer la Hessienne : Deux méthodes aussi équivalent

À partir de la première méthode de calcule du gradient, on trouve la Hessienne, tel que :

$$\frac{\partial^2 J}{\partial \beta_j \partial \beta_k} = \sum_{\mathrm{i}=1}^{\mathrm{n}} p_i (1 - p_i)\, x_{ij} x_{ik}$$

Où :

x_{ij} et x_{ik} sont des éléments de la matrice X correspondant respectivement au coefficients β_j et β_k.

X et p_i sont identiques que lors du calcule de gradient.

À partir de la forme matricielle du gradient, on trouve la Hessienne tel que :

$$\mathrm{H}(\beta) = X^T \mathrm{DX}$$

Où :

D est une matrice diagonale de taille $n \times n$ (signifie que pour cette matrice carré seule les éléments sur la diagonale son non vide) avec :

$$D_{ii} = p_i(1 -)p_i$$

X toujours identiques à la matrice utilisé lors du calcule du gradient

3. – mise à jour des coefficients (ou paramètres) :

La fonction de mise à jour des coefficients permet de trouver les coefficients à partir du gradient et des paramètres d'initialisation mais contrairement à la méthode de descente de gradient, la distance à chaque itération n'est pas arbitraire mais provient de la fonction Hessienne, tel que :

$$\beta^{(t+1)} = \beta^{(t)} - H^{-1} \times \nabla J\beta^{(t)}$$

Cette méthode permet par conséquent d'optimiser la distance parcouru pour chaque itération.

À l'instar de la méthode de la descente de gradient, l'itération se poursuit jusqu'à ce que la norme euclidienne du gradient soit inférieur à une valeur arbitraire, tel que :

$$||\nabla J\beta^t|| < \varepsilon$$

Où ε est généralement :

$\varepsilon = 10^{-3}$ (convergence standard)

$\varepsilon = 10^{-6}$ (convergence précise)

$\varepsilon = 10^{-8}$ (convergence très précise)

Et où la norme euclidienne d'un vecteur, tel que $\begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_p \end{bmatrix}$ est :

$$||\nabla J\beta^t|| < \varepsilon = \sqrt{g_1^2 + g_2^2 + \ldots + g_p^2}$$

Exemple avec la méthode de Newton-Raphson :

Contexte identique que l'exemple donnée lors de la partie sur la méthode de descente de gradient. Rappel :

> Nous sommes à la 10ème journée de championnat de Ligue 1. Pour chaque match nous avons mesurer trois variables : le moral, la forme physique de l'équipe et la tactique adopté. Leurs valeurs sont compris entre 1 (catastrophique) et 10 (excellent). Voici les résultats des dix derniers matchs du PSG avec les variables expliquées :
>
> 1 – moral = 8 ; forme = 4 ; tactique = 6 ; résultat=défaite

2 – moral = 3 ; forme = 3 ; tactique = 7 ; résultat=défaite

3 – moral = 2 ; forme = 6 ; tactique = 8 ; résultat=nul

4 – moral = 3 ; forme = 7 ; tactique = 8 ; résultat=victoire

5 – moral = 6 ; forme =8 ; tactique = 8 ; résultat=victoire

6 – moral = 8 ; forme = 8 ; tactique = 7 ; résultat=victoire

7 – moral = 9 ; forme = 7 ; tactique = 7 ; résultat=victoire

8 – moral = 10 ; forme = 7 ; tactique = 8 ; résultat=victoire

9 – moral = 8 ; forme = 4 ; tactique = 6 ; résultat=nul

10 – moral = 5 ; forme = 6 ; tactique = 8 ; résultat=nul

Soit 2 défaite, 3 nul et 5 victoire.

L'objectif est de connaître la probabilité de victoire du PSG pour son 11ème match, en sachant que le moral de l'équipe est passé à 4, la forme est quant passer à 7 et enfin la tactique est stable par rapport au dernier match.

Pour cela je doit d'abord commencer par trouver les coefficients idéaux, tel que (toute victoire à comme résultat 1 et tout autre résultat est égale à 0) :

Les valeurs initiales des coefficients sont :

$\beta_0 = 0$

$\beta_1 = 0$

$\beta_2 = 0$

$\beta_3 = 0$

Ensuite nous cherchons le gradient vectorielle on commençons par définir la matrice X et son transposé, telle que :

$$X = \begin{bmatrix} 1 & 8 & 4 & 6 \\ 1 & 3 & 3 & 7 \\ 1 & 2 & 6 & 8 \\ 1 & 3 & 7 & 8 \\ 1 & 6 & 8 & 8 \\ 1 & 8 & 8 & 7 \\ 1 & 9 & 7 & 7 \\ 1 & 10 & 7 & 8 \\ 1 & 8 & 4 & 6 \\ 1 & 5 & 6 & 8 \end{bmatrix} \text{ Et } X^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 8 & 3 & 2 & 3 & 6 & 8 & 9 & 10 & 8 & 5 \\ 4 & 3 & 6 & 7 & 8 & 8 & 7 & 7 & 4 & 6 \\ 6 & 7 & 8 & 8 & 8 & 7 & 7 & 8 & 6 & 8 \end{bmatrix}$$

Puis définir le vecteur y et p , tel que :

$$y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \text{ Et } p = \begin{bmatrix} 0{,}5 \\ 0{,}5 \\ 0{,}5 \\ 0{,}5 \\ 0{,}5 \\ 0{,}5 \\ 0{,}5 \\ 0{,}5 \\ 0{,}5 \\ 0{,}5 \end{bmatrix}$$

(le calcule des probabilités du vecteur p est le même que dans la méthode de descente de gradient)

Je peux donc calculer les gradients des coefficients initiaux, tel que :

$$(p - y) = \begin{bmatrix} 0{,}5 \\ 0{,}5 \\ 0{,}5 \\ -0{,}5 \\ -0{,}5 \\ -0{,}5 \\ -0{,}5 \\ -0{,}5 \\ 0{,}5 \\ 0{,}5 \end{bmatrix}$$

$$\nabla \mathrm{J}(\beta) = X^T(p - y) = \begin{bmatrix} 0 \\ -5 \\ -7 \\ -1.5 \end{bmatrix}$$

Maintenant, que j'ai le gradient, je dois calculer l'Hessienne, tel que :

$$D = \begin{bmatrix} 0{,}25 & & & & & & & & & \\ & 0{,}25 & & & & & & & & \\ & & 0{,}25 & & & & & & & \\ & & & 0{,}25 & & & & & & \\ & & & & 0{,}25 & & & & & \\ & & & & & 0{,}25 & & & & \\ & & & & & & 0{,}25 & & & \\ & & & & & & & 0{,}25 & & \\ & & & & & & & & 0{,}25 & \\ & & & & & & & & & 0{,}25 \end{bmatrix}$$

$$\mathrm{H}(\beta) = X^T \mathrm{DX} = \begin{bmatrix} 2{,}5 & 15{,}5 & 15 & 18{,}25 \\ 15{,}5 & 114 & 95{,}25 & 111 \\ 15 & 95{,}25 & 97 & 111{,}5 \\ 18{,}25 & 111 & 111{,}5 & 134{,}75 \end{bmatrix}$$

Je peux enfin faire la mise à jour des coefficients, tel que :

$$H^{-1} = \begin{bmatrix} 1{,}1980 & -0{,}1010 & -0{,}1410 & -0{,}0490 \\ -0{,}1010 & 0{,}0490 & -0{,}0350 & 0{,}0110 \\ -0{,}1410 & -0{,}0350 & 0{,}2360 & -0{,}1660 \\ -0{,}0490 & 0{,}01110 & -0{,}1660 & 0{,}1540 \end{bmatrix}$$

$$\beta^{(1)} = \beta^{(0)} - H^{-1} \times \nabla J\beta^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1{,}1980 & -0{,}1010 & -0{,}1410 & -0{,}0490 \\ -0{,}1010 & 0{,}0490 & -0{,}0350 & 0{,}0110 \\ -0{,}1410 & -0{,}0350 & 0{,}2360 & -0{,}1660 \\ -0{,}0490 & 0{,}01110 & -0{,}1660 & 0{,}1540 \end{bmatrix} \times \begin{bmatrix} 0 \\ -5 \\ -7 \\ -1.5 \end{bmatrix}$$

$$\beta^{(1)} = \beta^{(0)} - H^{-1} \times \nabla J\beta^{(0)} = \begin{bmatrix} -1.5655 \\ 0{,}0165 \\ 1{,}2280 \\ -0{,}8760 \end{bmatrix}$$

Je m'arrête ici pour l'exemple mais l'idéale est d'itérer les calculent et la mise à jour en repartant sur la base de ce dernier $\beta^{(t+1)}$.

Il est maintenant temps de calculer la probabilité que le PSG gagne, tel que :

$z = 0{,}0885$

$P\,(1) = \frac{1}{1+e^{(-0{,}0885)}} = 0{,}52{,}2$

Donc, la probabilité que le PSG gagne sont prochain match est de 52,2% avec cette méthode

Cependant, le problème de la régression logistique standard est qu'il y a un risque de sur-apprentissage du modèle, c'est-à-dire que le modèle sera très bon pour prédire les événements qui contiennent des valeurs explicatives déjà vues dans les données d'entraînement en plus de prévoir les anomalies spécifiques propres à ces données mais à contrario sera très mauvais pour prédire en dehors de ce champs. Par conséquent il nous reste une solution si l'on souhaite que notre modèle de prédiction soit pertinent sur un large champ de données en entrée : c'est de lui fournir justement des données qui recouvre tout le champs sur lequel il opère. Mais pour éviter cette tâches coûteuse, voir impossible en fait entrer en jeu la régression logistique régularisée.

4. La régression logistique régularisée

La régression logistique régularisée est un modèle qui régularise les coefficients. Cela signifie qu'elle ajoute une pénalité à la fonction de coût que le modèle cherche à minimiser. Cette pénalité varie en fonction des coefficients du modèle. Ce qui permet d'avoir un modèle qui peut prédire des événements encore jamais vues à partir de ses données tout en étant vraisemblable. Il y a deux méthodes principaux de régularisation des coefficients : L1 (Lasso) et L2 (Ridge).

Pour la suite du documents, je passerais d'une notation β courante en statistique à une notation θ souvent utilisé en apprentissage automatique (ML).

La fonction de coût de base pour la régression logistique (sans régularisation) est la log-loss :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\, log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right]$$

Où :

m Est le nombre d'observations

$y^{(i)}$ Est la valeur expliquée de l'observation (i) qui est soit de 0 soit de 1

$x^{(i)}$ Un vecteur des valeurs explicatives de l'observation (i)

θ Est le vecteur de coefficients comme β

h_θ Est la fonction de prédiction du modèle tel que :

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

La fonction de coût avec régularisation L1 :

$$J_{L1}(\theta) = J(\theta) + \frac{\lambda}{m}\sum_{j=1}^{n}\left|\theta_j\right|$$

Où :

$\left|\theta_j\right|$ Est la valeur absolue de θ_j. C'est à dire sans aucun signe (- ou +)

λ Est un hyper-paramètre non déterminé, qui contrôle l'intensité de la régularisation. Il pondère l'importance relatif de deux objectifs contradictoires :

– Minimiser l'erreur sur les données d'entraînement avec la log-loss
– Minimiser la complexité du modèle avec la pénalité

Alors pour déterminer la valeur de l'hyper-paramètre, il faut :

1. Définir une plage de valeur à tester, généralement une échelle algorithmique tel que : $\lambda = [0{,}0001;\ 0{,}001;\ 0{,}01;\ 0{,}1;\ 1;\ 10;\ 100;\ 1000]$

2. Diviser les observations en deux groupes : données d'entraînement avec un proportion 70-80% et de test avec une proportion 20-30%. Cette proportion dépend de la taille des données et de la complexité du modèle. Moins il y a de données et plus il faudra tendre à équilibrer les deux groupes et plus la complexité est élevé plus il faudra surchargé les données d'entraînement. En règles générale :
   - Observations < 1000 : Privilégiez du 70%/30% voir 60%/40%
   - 1000 < Observations < 10 000 : Privilégiez du 80%/20%
   - 10 000 < Observations : plutôt du 80%/20% ou même du 90%/10%
3. Diviser les observations d'entraînements en k plis (ou groupe) équilibré, idéalement avec des lots par plis entre 5 et 10. Plus k est grand et plus l'estimation est stable mais c'est plus long.
4. Pour chaque valeur de la plage de λ, séparer les k-1 plis pour l'entraînement et le plis k restant pour la validation.
5. Pour chaque valeur de la plage de λ, entraîner le modèle sur les k-1 plis. Cela signifie minimiser la fonction de coût avec régularisation du modèle pour chaque valeur λ.
6. Pour chaque valeur de la plage de λ, évaluer la performance sur le plis de validation. Cela veut dire que pour chaque valeur λ :
   a. Faire des prédictions sur les données du plis de validations (non utilisé pour l'entraînement).
   b. Calculer des métriques de performances. Il y a plusieurs types de métriques d'évaluation, voici les plus courants :
      - Mesurer le taux de d'événement prédit comme positifs qui ont vraiment étaient positifs. C'est l'évaluation de la précision de la prédiction (ou accuracy) la formule est :

$$Prés ion = (VP + VN)\,/(VP + VN + FP + FN)$$

Où VP = vrais positifs, VN = vrais Négatifs, FP = Faux positifs et FN = faux négatifs, et où le seuil est arbitraire mais par défaut il est définit à 0,5

      - Mesurer la moyenne harmonique entre la précision et le rappel, (nommé score F1). Autrement dit, cela permet de mesurer un compromis entre le taux de précision de la prédiction positives du modèle et le taux de réussite positives du modèle. Utile dans le cas où il y a un déséquilibre numérique entre les classes. La formule est :

$$F1 = 2 \times ((précision \times rappel)\,/(précision + rappel))$$

Où rappel est la mesure du taux de bonne prédiction positives que le modèle à a son actif parmi toutes les observations positives réelles , tel que :

$$Rappel = VP/(VP + FN)$$

      - Mesurer l'incertitude des prédictions probabilistes, en pénalisant davantage les prédictions confidentes mais incorrectes. Une prédiction confidente est une prédiction où le modèle est sûr de lui. Par exemple il prédit une issue à 99% etc. La formule est :

$$log - loss = -\left(\frac{1}{N}\right) \times \sum [y_i \times log\,(p_i) + (1 - y_i) \times log(1 - p_i)]$$

Où

N est le nombre d'observations

y_i est la valeur réelles (0 ou 1) pour l'observation i

p_i Est la probabilité prédites par le modèle pour l'observation i

- Montrer les vrais/faux positifs/négatifs avec une matrice de confusion, tel qu'une matrice 2×2 avec les colonnes pour les valeurs de prédictions et les lignes pour les valeurs réelles, tout deux diviser en valeurs positifs et négatifs.
- Montrer les différences entre les taux de vrais positifs et de faux positifs avec la Courbe ROC et mesurer la capacité de discrimination du modèle avec la courbe AUC.

c. Comparer les métriques des différentes valeurs λ, puis choisir celles avec les meilleurs performances.
d. Entraîner le modèle sur l'intégralité du jeu d'entraînement de la même façon qu'avec les plis. Pour rappel cela veut dire : minimiser la fonction de coût avec régularisation du modèle avec le λ.
e. Tester (ou évaluer) les performances du λ avec l'intégralité jeu de test, toujours de la même façon que lors de l'utilisation des plis.

La fonction de coût avec régularisation L2 :

$$J_{L2}(\theta) = J(\theta) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Où les paramètres ont la même signification que dans la fonction L1.

À la différence de L1, L2 ne supprime pas les coefficients même en cas de forte pénalité.

Attention la pénalité ne doit pas s'appliquer à l'intercept. Cela pourrait biaiser le modèle.

Cependant, actuellement tout mes données ont le même poids, qu'il soit récolté d'il y a dix ans ou d'aujourd'hui. Cela peut être préjudiciable pour certaines prédictions. Pour résoudre ce problème nous ajouterons une pondération temporelle, qui donnera plus de poids au événements récents.

Cas pratique :

Il y a certaines caractéristiques communes des matchs de la ligue 1 sur 10 ans, de la saison 15-16 à la saison 24-25. Ces caractéristiques sont entre autres :

- le nom des équipes de chaque rencontres
- l'issue réelles de la rencontre
- l'heure et la date du début de la rencontre
- Les dernières côte enregistrer du match de tout les issues du marché 1N2 d'un bookmaker
- Les premiers côte enregistrer du match de tout les issues du marché 1N2 d'un bookmaker

– Les dates et heures de toutes les cotes

Le but est de prédire l'issue des prochaines rencontres de Ligue 1 avec des performances supérieur à l'aléatoire.

Pour l'instant, je prends en compte seulement les côtes comme variables du résultat finale. Donc, pour réaliser cette tâche, j'ai fait le choix du modèle de régression logistique avec régulation L2. Cela permettra de prédire l'issue du match à partir de côtes qui ne sont pas présent dans les données d'entraînement du modèle.

Je fais le choix de la library scikit-learn de python, qui est l'outil idéal pour répondre à ce besoin dans notre cas, car facile à prendre en main et complet.

Il y aura trois modèles : un pour la prédiction des victoires à domicile, un autre pour les match nul, puis un dernier pour les victoires à l'extérieur.

Il faut savoir que par défaut, l'étape "d " lors de l'évaluation de performance de l'hyper paramètres se fait automatiquement, avec les fonctions de cette library.

Étant, donner que notre objectif est de trouver les values bets, nous choisirons la meilleur valeur C, en fonction de celui qui offre la meilleur performance log-loss, car elle permet :

– L'évaluation de la précision du niveau de confiance qu'accorde le modèle à ses prédictions. Cela se fait avec log-loss. Donc, plus le log-loss est faible, plus on peut accorder de la confiance au prédiction dont le modèle à confiance. Un log-loss d'un modèle à trois issues aléatoire et équilibré est d'environ 1.0986.

Et comme critère de comparaison secondaire, nous choisissons :

– L'évaluation du taux de réussite positives du modèle et son taux de précision positives. Cela se fait avec le score F1 et plus le score et élever, plus les taux sont élever. Le score F1 d'un modèle à trois issues aléatoire et équilibré est d'environ 0.33.
– L'évaluation du taux de discrimination entre les classes du modèle. Cela se fait avec la courbe AUC. Plus le taux est proche de 50 % plus la discrimination est aléatoire, à contrario, plus c'est proche de 1, plus c'est parfait, et si c'est inférieur à 50 % cela veut dire que c'est pire que discriminer aléatoirement. Le taux AUC d'un modèle à trois issues aléatoire et équilibré est d'environ 0.5.

Donc, après évaluation des modèles, voici la valeur des trois métriques et l'interprétation qu'on en fait :

– Victoire à domicile :
    - Le log loss est d'environ 0.62, cela signifie que la précision des probabilités du modèle sont supérieur à un modèle aléatoire. Après pondération des classes cela reste à peu près constant.
    - Le score F1 est d'environ 0,50. Ce qui signifie que le modèle à un score supérieur à un modèle aléatoire. Par contre après pondération des classes, nous passons à 0,64
    - Le taux AUC est de 70 % environ. Ce qui montre une assez bonne discrimination du modèle, pour choisir la bonne classe par rapport à l'aléatoire.

De manière générale, le modèle à de bonnes performances dans les critères choisit par rapport à un modèle aléatoire.

– Nul :
    - Le log loss est d'environ 0.52. Cela montre que la calibration des probabilité est très bonne par rapport à un modèle aléatoire. Après pondération, il est à 0,68
    - Le score F1 est de 0. Ce qui indique clairement une incapacité à prédire un match nul. Soit un modèle pire qu'un modèle aléatoire. Et 0,32 après pondération.
    - Le taux AUC est d'environ de 50%, qui qui montre une capacité de discrimination entre les classes d'un niveau aléatoire.

  Donc, de manière générale ce que l'on peut dire du modèle, est que, bien que sa calibration probabiliste soit très bonne, il est inopérant dans la réalité, car sa capacité de distinguer les matchs nuls, des match gagnants par unes des deux équipes est de niveau aléatoire. Et surtout il est incapable de prédire un match nul.

– Victoire à l'extérieur :
    - Le logs-loss est de 0.6 à peu près. Ce qui indique une bonne calibration moyenne des probabilités par rapport à un modèle aléatoire. Lègère augmentation à 0,64 après pondération.
    - Le score F1 est de 0,25. Ce qui montre de grandes difficulté à prédire les victoires à l'extérieur. C'est pire qu'un modèle aléatoire. Mais 0,52 après pondération.
    - Le taux AUC est de 66% environ. Signifie qu'il y a une bonne capacité de discrimination générale du modèle par rapport à un modèle aléatoire.

  Ainsi, il y a généralement une discrimination des classes et une calibration moyenne des probabilité plutôt bonne mais des difficultés à prédire les victoire à l'extérieur.

Pour conclure, les différents modèles ont tous une calibrations des probabilités supérieur à un modèle aléatoire à l'instar des capacités de discriminations des classes qui est aussi généralement bonne hormis pour le modèle de prédiction des matchs nul qui est moyen sans pondération. Contrairement, au capacité de prédiction qui sont majoritairement mauvais par rapport à un modèle aléatoire, à l'exception du modèle de prédiction de victoire à domicile.

Donc le score qualité moyen

– sans pondération : est de 0,361
    - Pour victoire domicile seulement elle est de 0,468
    - Pour nul seulement elle est de 0,263
    - Pour victoire extérieur seulement elle est de 0,352
– Avec pondération : est de 0,452
    - Pour victoire domicile seulement elle est de 0,538
    - Pour nul seulement elle est de 0,351
    - Pour victoire extérieur seulement elle est de 0,469

Le prochain objectif sera donc, d'élever le taux de prédiction de la classe de prédilection. Pour cela un modèle multi-classe sera tester. Car, cela permettra au modèle unique d'avoir le contexte qu'il lui faut pour être plus précis dans ses prédictions, sans devoir ajouté de features

particulier dans un premier temps. C’est ce qui est adapté à un problème à trois issues comme tel. Ce type de modèle est une régression logistique multinomiale.

5. La régression logistique multinomiale régularisé L2

Un modèle de régression logistique multinomiale à pour but de prédire une variable qualitative nominal parmi un ensemble de plus de deux classes. Concrètement, pour faire cela, le principe de probabilité complémentaire de la régression logistique binaire est généraliser à chaque classes du modèle multinomiale. Pour rappel, le principe de probabilité complémentaire pour le cas binaire permet de trouver la probabilité d'une classe (complémentaire) à partir de la probabilité d'une autre classe. Donc, dans le cas d'un problème multinomiale à K classes, on généralise ce principe. Au lieu d'avoir un seul vecteur de paramètres θ, nous allons avoir K vecteurs de paramètres, tel que : $\theta_1, \theta_2, \ldots, \theta_k$. Chaque vecteur θ_k est responsable de calculer la probabilité que l'instance X appartienne à la classe K par rapport à une classe de référence (qui est à définir).

Ainsi, la fonction de coût du modèle de régression logistique multinomiale régularisée L2 (appelé aussi fonction de perte logistique multinomiale) est :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left(y_k^{(i)} log\left(h_\theta\left(x^{(i)}\right)_k\right)\right) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Où :

K (grand) est le nombre de classes,

k (petit) est la classes actuelles lors de l'itération

$y_k^{(i)}$ Est un scalaire qui à soit une valeur de 1 ou soit de 0. C'est ce qui est appelé, encodage « One-Hot » de la vérité terrain.

$h_\theta\left(x^{(i)}\right)_k$ est une fonction softmax et non plus une fonction sigmoïde. Elle renvoie la probabilité prédite que l'observation $x^{(i)}$ appartienne à la classe k, tel que :

$$h_\theta(x)_k = \frac{e^{\theta_k^T x}}{\sum_{j=1}^{K} e^{\theta_j^T x}}$$

Où x est le vecteur de l'observation à l'instar de $x^{(i)}$.

Et, elle garantie que la somme des probabilités de toutes les classes soit égale à 1, tel que :

$$\sum_{k=1}^{K} h_\theta\left(x^{(i)}\right)_k = 1$$

Ainsi, il y a deux méthodes de calcule de la probabilité d'une classe :

– Le modèle Logit (avec classe de référence) se calcule en plusieurs étapes :
  - 1. Choisir arbitrairement une classe de référence (ou classe de base).
  - 2. Modéliser la log vraisemblance (log-odds) de chaque classe par rapport à cette classe de référence. Tel que pour une classe J quelconque, on a :

$$log\left(\frac{P(y = K|x)}{P(y = K|x)}\right) = \theta_j^T x$$

Où J est un nombre allant de 1 à $K-1$

- 3. Calculer les probabilités des différentes classes tel que :
  - Pour une classe J :

$$P(y=j\,|x) = \frac{e^{\theta_j^T x}}{1+\sum_{k=1}^{K-1} e^{\theta_k^T x}}$$

  - Pour la classe de référence K :

$$P(y=K\,|x) = \frac{1}{1+\sum_{k=1}^{K-1} e^{\theta_k^T x}}$$

– La formulation Softmax (sans classe de référence) : C'est la formulation la plus courante en apprentissage automatique et c'est celle utiliser par scikit-learn. Voici ses étapes :

- 1. Attribuer un vecteur θ_k à chaque classe k (de 1 à k), sans en désigner une comme référence.
- 2. Calculer les probabilité des classes, tel que :

$$P\,(y=j|x) = \frac{e^{\theta_j^T x}}{\sum_{j=1}^{K} e^{\theta_j^T x}}$$

Toutefois, cette formule à un problème. Elle est surdéterminé. C'est à dire que l'ajout d'un vecteur constant C à tous les vecteurs de paramètres θ_k provoquent une constance des probabilités. Par conséquent, la solution est de fixer une contrainte aux paramètres. La contrainte la plus courante est de fixer les paramètres d'une classe à zéro. En pratique, cela consiste à choisir une classe K est à amener son vecteur de paramètres à 0, tel que : $\theta_k = 0$.

Donc, lors de l'attribution des vecteurs de paramètres à chaque classe il faut aussi veuillez à attribuer un vecteur de paramètres 0 à une classe préalablement choisit.

Ainsi, même la méthode de calcule softmax, qui est censé être sans classe de référence en à finalement une.

Cas pratique :

Avec le même contexte que le précédent cas pratique et en tenant compte de ses conclusions, l'objectif sera ici, de passer d'un modèle binomiale à un modèle régularisé L2 trois classes, tel qu'un vecteur de classes [Victoire domicile, Nul, Victoire extérieur]. Et les mêmes métriques seront à surveiller. Le résultat attendu est une augmentation du score F1.

Toutefois, il faut savoir que l'évaluation change un peu pour toutes ces métriques. En effet, ils étaient tous basés toutes ou parties sur l'évaluation de la classe positive. Et comme, avec la multinomiale il n'y a plus de classes définie comme positives alors :

– Log loss : lui ne change pas
– F1 : calcule de manière classique le F1 pour chaque classe puis plusieurs posssibilités pour produire un score :
  - Macro : Moyenne des moyennes F1 sur toutes les classes.

    - Weighted : moyenne des F1 à l'instar du macro sauf qu'elles est pondérer par le nombre d'observations de chaque classes. Plus une classe sera présente, plus elle aura de poids.
    - Micro : calcule un score F1 global pour l'ensemble des classes, en regroupant les paramètres de chaque classes et leurs appliquer la fonction F1.
- ROC AUC : à globalement deux stratégies :
    - One-vs-rest : Calcule séparer et faire la moyenne
    - One-vs-one : calculer par pair comme si c'était des binaire puis faire la moyenne

Puisque l'objectif est d'augmenter la performance globale du modèle, nous nous tournons logiquement vers la moyenne.

Voici les résultat du modèle :

- Le log loss : est à environ 0,96 en moyenne et avec pondération, elle est d'environ 1. Ce qui montre une perte de calibration en moyenne des probabilités très forte par rapport aux derniers modèles dont les log-loss étaient de 0,6 ; 0,52 et 0,62, ce qui donne une moyenne de 0,58.
- Le score F1 : est d'environ 0,41 en moyenne et de 0,467 avec pondération. Cela illustre une prédiction moyenne des classes en nette amélioration comparer aux précédents modèles, dont les scores F1 sont de 0,5 ; 0 ; 0,25 et dont la moyenne est de 0,25.
- Le taux AUC : est d'environ 65% en moyenne. Cela signifie, une amélioration (modeste mais présente) de la discrimination générale entre les classes par rapport aux modèles précédent, qui ont un taux de 58% ; 25% et de 0%. Ce qui leurs fait une moyenne de 62%.

Conclusion, l'objectif à était atteint, cet-à-dire une amélioration du taux de prédiction des classes et au passage une amélioration aussi du taux AUC. Par contre, je constate une nette baisse de la calibration du modèle, mais toujours dans une zone acceptable. Ainsi le score qualité baisse :

- Avec pondération : à environ 0,28.
- Sans pondération : à environ 0,27

Toutefois, il ne ne faut pas oublier que l'objectif est de « prédire les issues des prochaines rencontres de Ligue 1 ». Les métriques observés sont sur l'ensemble des données d'observations des 10 dernières années. Mais j'ai besoin de bonnes performances uniquement sur cette saison.

Ainsi, je pense que le modèle sera plus performant s'il n'a pas une vue d'ensemble généraliser et uniforme de ces données mais qu'il peut déceler les variations d'une saison à une autre, en donnant plus d'importance au bruits récents mais sans éclipser les observations plus anciennes, qui offrent tout de même un contexte précieux. La pondération temporelle permet cela.

6. La pondération temporelle dans la régression logistique régularisée L2

La pondération temporelle, permet de varier le poids d'une observation dans les données d'entraînement en fonction de son ancienneté. Généralement plus une observation est vielle, moins elle à de poids. Il y a deux façons de calculer ce poids :

– La décroissance temporelle : utiliser pour suivre une évolution continue. Voici la formule :

$$\omega_i = e^{-\alpha\left(t_{max}-t_i\right)}$$

Où :

t_{max} est le temps le plus récent (temps de la dernière observations d'entraînement)

t_i est le temps de l'observation i

α est le taux décroissance du poids, plus c'est petit et plus la croissance et lente et au contraire plus c'est grand et plus la croissante est rapide. Par exemple un taux de 0,1 est une croissance plutôt lente.

Cependant, si le taux de décroissance du poids est peu rapide cela entraîne une pollution du modèle par les observations anciennes et donc peu pertinentes mais au contraire si le le taux et trop rapide cela rend les infos anciens peu pertinent pour le modèle alors qu'ils sont utile. Donc, une valeur optimal, notamment avec la méthode d'optimisation bayésienne.

– La fenêtre glissante : est utiliser pour marquer une évolution abrupte dans les données. Cela consiste à attribuer un poids arbitraire, tel que ω aux observations récentes et $\omega - 1$ aux plus anciens. Par exemple $\omega = 1$ aux plus récents et $\omega = 0$ aux plus anciens.

Dans la pratique un modèle peut combiner les deux approchent, car les deux cas de figure peuvent être présent dans un même jeu.

Ce poids se multiplie à la fonction de perte individuelle de chaque observation, pour produire une fonction de coût multinomiale régularisée avec L2 et pondération temporelle, tel que :

$$J_{L2}(\theta) - \frac{1}{m}\sum_{i=1}^{m} w_i \sum_{k=1}^{K}\left[y_k^{(i)} log\left(h_\theta\left(x^{(i)}\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Cas pratique :

En partant du contexte du dernier cas pratique, j'ai un problème. Le modèle s'entraîne et s'évalue sur des données trier de manière aléatoire. Sauf que notre objectif est de prédire les prochains matchs de Ligue 1. Alors, nous devons faire en sorte qu'il s'entraîne avec les données à sa disposition tout en apprenant les tendances actuelles. Toutefois les testes doivent être effectués uniquement sur les données récents. Bien sûr les données d'entraînement ne doivent pas être utilisées pour les tests pour éviter l'apprentissage par cœur.

Alors concrètement, j'agirais sur deux points du modèle précédent :

– Méthode Rolling : Les observations doivent être ordonnées du plus anciens au plus récents, puis diviser ses données en train et test avec un ratio 80/20 tout en ayant les

plus récents dans les tests. Ensuite effectué les plis, tout en ayant les observations les plus récents comme tests des plis. Donc, concrètement :

- Premier split train/test de 80/20 :
  - Train : 2015-2016 à 2022-2023
  - Tests : 2023-2024 à 2024-2025
- Sur le groupe train : une saison par plis et la saison suivantes pour tester ce plis

– Méthode d'optimisation bayésienne :

- 1. Définir une intervalle de alpha [0 ;1]
- 2. Effectuer avec le lambda et C une cross validation sur les deux

Problème nous ne pouvons pas faire d'optimisation bayésienne avec la library utilisé, on a dû implémenter la library optuna, spécialisé dans l'optimisation d'hyper-paramètres.

Voici les résultats du modèle avec pondération et itérations de validations croisée de max 35 :

– Log loss = 0,93 et de même avec pondération de classe
– Score F1 = 0,43 et de même avec la pondération de classe
– Taux AUC = 0,68 contre 0,63

Donc le score qualité est d'environ 0,29

Toutes les métriques montrent donc, que ce modèle à de meilleurs performances. La prochaine section expliquera la métrique de "score qualité".

7. Calcule du taux de confiance accorder au modèle

F1 étant la métrique qui évalue la capacité de prédiction du modèle et log loss celui qui évalue la confiance du modèle dans ces prédictions, alors un modèle parfait est un modèle ayant à la fois une capacité de prédiction parfaite, et donc un score F1 de 1 et à la fois un niveau de confiance absolue en ses prédictions, c'est-à-dire un log-loss de 0.

L'augmentation du log-loss ou la baisse du score F1 entraîne une baisse de qualité du modèle. Alors, j'évalue un modèle parfait, tel que :

$$Q = F1 - logLoss = 1 - 0 = 1$$

C'est le score qualité, noté Q, le plus élevé, puisque si un des deux paramètres n'est pas à leurs niveau de perfection, Q sera toujours inférieur à 1. Alors un modèle de mauvaise qualité c'est à dire qui ne rempli pas ses fonctions, c'est-à-dire d'avoir de meilleur performance qu'un jeu aléatoire est :

$$Q = F1 - logLoss = 0{,}33 - 1{,}0986 = -0{,}7686$$

Où :

$F1$ Aléatoire est $\frac{1}{K}$,

$logLoss$ Aléatoire est $-log\left(\frac{1}{k}\right)$,

tel que k étant le nombre de classe.

Alors pour avoir un ratio facilement interprétable, nous normalisons la log-los, tel que :

$$score_{log} = 1 - \frac{logLoss}{-log\left(\frac{1}{k}\right)}$$

Où $score_{log}$ est le score parfait

Alors la qualité du modèle peut être représenté par le score qualité Q, tel que :

$$Q = \frac{\left(score_{log} + scoref1\right)}{2}$$

Où $Q \leq 0$ signifie médiocre et $Q = 1$ signifie : parfait.

Alors le score Q actuelle est :

$score_{log} = 1 - \frac{logLoss}{-log\left(\frac{1}{k}\right)} = 1 - \frac{0{,}93}{1{,}0986} = 0{,}1535$

$Q = \frac{\left(score_{log} + scoref1\right)}{2} = \frac{0{,}1535 + 0{,}4291}{2} = 0{,}2913$

Maintenant l'objectif est d'accroire le score qualité pour atteindre la barre des 0,5. Pour cela, nous explorerons l'ajout de variables expliquées comme les dates, en particulier des intervalles de temps entre l'événement et la fin de saison.

8. Accroître la performance

J'ai plusieurs variables qui caractérisent un match et dont l'issue du match dépend :

– Nom de l'équipe à domicile
– Nom de l'équipe à l'extérieur
– Intervalle de jours entre le match et la fin de saison
– Intervalle de jours entre le match et la publications des côtes

Toutefois, j'ajouterais seulement les intervalles aux variables expliquées, dont voici la qualité du modèle :

$$Q = \frac{\left(score_{log} + scoref1\right)}{2} = \frac{0{,}1561 + 0{,}3753}{2} = 0{,}2657$$

Mais je constate que le score qualité à baisser, ce qui pourrait bien signifiait que ces variables brouille la ligne de discrimination et sont très fortement corrélé au variables déjà en place. Pour explorer cette piste, je modifierais la régularisation qui est actuellement L2 au profit de la régularisation L1, car cela permet de supprimer certaines variables impertinent et/ou redondant. Ainsi, cela donne un score qualité du modèle à :

$$Q = \frac{\left(score_{log} + scoref1\right)}{2} = \frac{0{,}1028 + 0{,}3922}{2} = 0{,}2475$$

Cela est dû au fonctionnement de la régularisation L1, qui élimine certaines variables est donc à tendance à privilégier le taux de prédiction à la calibration.

Quoi qu'il en soit, il semble que j'arrive au bout de l'exploration de la régression logistique et il ne me permettent pas d'atteindre un niveau de score qualité satisfaisant. Même s'il est possible de combiner les deux types de régularisation avec la méthode elasticNet, cela ne me permettra pas d'avoir c'est écart positif du score qualité du modèle.

Il faut donc, explorer d'autres types de modèles. Notamment un modèle capable d'apprendre de ces erreurs dans un apprentissage constant et pourquoi pas faire émerger d'autres variables explicatives etc a partir de cette apprentissage. Le type de modèle qui se rapproche le plus de cela est le gradient boosting.

9. Le gradient boosting

Le gradient boosting est une technique d'apprentissage ensembliste (ensemble learning) qui combine plusieurs modèles faibles, comme des arbres de décision peu profonds, pour former un modèle fort. Son idée centrale est d'ajouter itérativement des modèles qui corrigent les erreurs résiduelles du modèle actuel.

Voici, les étapes de mise en œuvre du gradient Boosting :

1. Initialisation :

   On commence avec un modèle constant (c-à-d que peut importe la variables d'entrées cela donne toujours les mêmes résultats), tel que :

   $$F_0(x) = \arg\min_{\Upsilon} \sum_{i=1}^{n} L(y_i, \gamma)$$

   Où :

   L Est une fonction de perte différenciable

   γ Est une valeur constante initial

   n Est le nombre d'observation dans le dataset

2. Boucle sur itérations $m = 1$ à M

   Pour chaque itération :

   A. Calculer les résidus (pseudo-résidus)

      Pour chaque observation i, on calcule le gradient négatif de la perte (d'où l'intérêt d'avoir une fonction de perte différenciable, tel que :

      $$r_{im} = -\left[\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}\right]$$

      Où :

      r_{im} Est le résidu de l'observation i à l'itération m.

      F_{m-1} est la prédiction actuel du modèle (la confiance de la prédiction)

      Exemple pour une log loss :

      $$L(y, F) = log\left(1 + e^{-yF}\right) \Rightarrow \frac{\partial L}{\partial F} = \frac{-y \cdot e^{-yF}}{1 + e^{-yF}} \Rightarrow r_{im} = \left[\frac{-y_i}{1 + e^{y_i F - 1(x_i)}}\right]$$

   B. Un modèle faible apprend sur les résidus

      On entraîne un modèle $h_m(x)$, comme un arbre de décision sur les données (x_i, r_{im}). Ce modèle apprend à prédire les résidus.

   C. Mettre à jour le modèle

$$F_m(x) = F_{m-1}(x) + \nu \cdot h_m(x)$$

Où ν est le taux d'apprentissage (learning rate), typiquement entre 0,01 et 0,1.

3. Modèle final

Après M itérations :

$$F_M(x) = F_0(x) + \nu \sum_{m=1}^{M} h_m(x)$$

Plusieurs points à noté :

– Plus le nombre d'itérations M est grand, plus il y a un risque de sur-apprentissage.
– Un faible taux d'apprentissage ν donne de meilleur performance de généralisation du modèle mais cela nécessite plus d'itérations.

Une fonction de perte différenciable est une fonction dont on peut calculer la dérivée en tout point (ou presque). La dérivée nous donne la pente de la fonction à un point donné, tel qu'une fonction $L(y, \hat{y})$ est différenciable si pour tout point $\hat{y}$, la limite suivante existe :

$$\frac{\partial L}{\partial y} = \lim_{h \to 0} \frac{L(y, \hat{y} + h) - L(y, \hat{y})}{h}$$

Où h représente un petit incrément infinitésimale, qui tend vers 0.

Formule d'une fonction de perte différenciable, bien connu, qui est le log loss :

$$L(y, \hat{y}) = -[y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})]$$

Dont la dérivée est :

$$\frac{\partial L}{\partial \hat{y}} = \frac{y}{\hat{y}} + \frac{1 - y}{1 - y}$$

Il y a aussi l'erreur quadratique (MSE).

Un arbre de décision est un outil d'apprentissage automatique supervisé (= les réponses apprises sont fournies par un superviseur), qui peut être utilisé pour des problèmes à la fois de classification (prédire une catégorie) et à la fois de régression (prédire une valeur continue). Comme son nom l'indique, sa structure ressemble à un arbre (ou organigramme de décision). Autrement dit, à chaque intersection il répond à une question et en fonction de la réponse il prend la décision de tourner à gauche ou à droite et ainsi de suite jusqu'à arriver à une extrémité de l'arbre. Chaque ramifications inférieurs est plus pur que la précédente.

Voici les composants d'un arbre de décision :

– Nœud racine (root node) : Le noeud supérieur qui contient l'ensemble complet des données du dataset.
– Nœuds de décision (decision nodes/internal nodes) : Les nœuds où le dataset est divisé. Ils représentent un test sur une caractéristique (par exemple, "Côtes < 2 ?").

– Branches (edges) : Le résultat du test au précédent nœud de décision. Elles mènent au nœud enfant suivant.
– Nœuds feuilles (leaf nodes / terminal nodes) : Les nœuds finaux qui ne se divisent plus. Ils contiennent la prédiction finale (la classe majoritaire pour la classification, la valeur moyenne pour la régression).

Voici les étapes pour construire un arbre de décision :

1. Commencer par la racine en obtenant l'ensemble des données
2. Sélectionner la meilleur caractéristique pour diviser les données. C'est l'étape la plus importante. Le critère de sélection est basé sur la maximisation de "l'impureté" ou du "gain informationnelle".
3. Diviser les données en sous-ensembles en fonctions des valeurs possibles de cette caractéristique.
4. Répétez récursivement les étapes 2 et 3 pour chaque sous-ensemble jusqu'à ce qu'une "condition d'arrêt" soit atteinte.

Une **condition d'arrêt** est une condition qui spécifie à l'algorithme de ne plus diviser les nœuds une fois qu'elle est vérifier. Voici les conditions d'arrêt courantes :

– Le nœud devient "pur" (tout les échantillons appartiennent à la même classe).
– La profondeur maximal de l'arbre est atteinte.
– Le nombre d'échantillons dans un nœud est inférieur à un seuil minimum.
– Le "gain d'information" d'une division est trop faible.

Alors concrètement, pour sélectionner les meilleurs caractéristiques en fonctions des critères évoquer à l'étape 2, il faut d'abord comprendre les concepts :

– D'entropie (ID3) : qui mesure le désordre ou l'incertitude dans un ensemble de données. Elle est maximale quand les classes sont équitablement réparties, et nulle quand tous les éléments sont de la même classe, tel que :

$$Entropie\,(S) = -\sum_{i=1}^{c} p_i \, log_2(p_i)$$

Où :

c Est le nombre de classes

p_i est la proportion des éléments de la classe i dans l'ensemble S.

– De gain d'information (IG) : qui mesure la réduction de l'entropie après une division basée sur une caractéristique A. On choisit la caractéristique qui maximise le gain d'information, tel que :

$$Gain\,(S,A) = Entropie\,(S) - \sum_{v \in Valeurs(A)} \frac{|S_v|}{|S|}$$

Où :

$Valeurs(A)$ = toutes les valeurs possibles de la caractéristique A (ex : Oui ou Non).

S_v Est le nombre d'éléments du sous ensemble S pour lequel la caractéristique A a la valeur v.

$\frac{|S_v|}{|S|}$ est le poids (la proportion) du sous ensemble S_v par rapport à l'ensemble S.

À noté, que plus le gain est élevé, plus la division basée sur la caractéristique A réduit le désordre.

– D'impureté de Gini (CART – arbres de classification et de régression) : qui mesure la probabilité qu'un élément choisi au hasard dans l'ensemble S soit mal classé si on lui attribue une classe aléatoire selon la distribution (proportion) des classes dans S. Ainsi, comme l'entropie est maximale pour une distribution uniforme, l'impureté de Gini est tel que :

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2$$

Donc, pour évaluer une division, on calcule l'impureté de Gini pour chaque sous-ensemble créé, puis on fait la moyenne pondérée. On choisit la caractéristique qui minimise cette impureté moyenne.

Cas pratique :

Avec le même contexte que le cas pratique précédent, nous reformatons le modèle, pour passer à un gradient boosting tout en conservant les mêmes variables.

Plusieurs choses à définir :

– Le taux d'apprentissage est d'intervalle [0,01 ; 1].
– Tester plusieurs valeurs des conditions d'arrêt suivant :
  - profondeur de l'arbre (commence à 3 puis incrémenter si besoin)
  - Nombre d'arbres (commence à 100 puis incrémenter si besoin). Il est corrélé au taux d'apprentissage.
  - Taille minimal des feuilles (commencer à 1 puis varier) plus la taille est bas et plus il y a un risque de sur apprentissage.
  - Taille minimum d'une branche pour qu'elle puissent être subdiviser. (commencer à 2 puis varier)
  - La fraction du jeu d'entraînement utilisé pour construire chaque arbre. Cela doit être une saison par arbre.

Toutes ces hyper-paramètres et paramètres seront optimiser avec la méthode bayésienne. Sans oublier la pondération temporelle qui contient un hyper-paramètre dans son équation.

On peut aussi ajouter la condition d'arrêt qui permet de stopper après x itérations sans amélioration de performance, en testant sur des données du jeu de test préalablement choisit.

Résultat :

Le score qualité du modèle est de 0,3263. Avec :

– Un log-loss d'environ 0,955
– Et un score F1 d'à peu près 0,522.

Toutefois, nous avons remarqué (de nouveau) un problème au quel nous avons fait face dès les premier modèle de régression logistique et que nous n'avons pas encore résolu. Ce problème est la difficulté pour les modèles de prédire les classes rares, et la classe rare ici est le ‘’draw’’. En effet le score F1 de celle-ci était de 0.

Ainsi, pour parer à ce problème nous avons ajouter une pondération liée à la proportion des classes. Autrement dit, plus une classe est sous représenter dans les observations, et plus elle à de poids. Cela à permis de donner de l'importance à cette classe. Cette implémentation change beaucoup le résultat, car avant d'effectuer cette pondération le score qualité était aux alentours de 0,25. Ce qui n'ont faisait pas un meilleur modèle que les précédents. Donc, si l'on avait ajouté cette implémentation au précédent modèles, on aurait sans doute obtenue un score qualité avoisinant les même valeur.

À noté que cette pondération sera ajouté au modèle antérieur et ultérieur.

Pour améliorer encore plus significativement les performances, ont peut ajouté de la régularisation au modèle et d'autres optimisation que nous verrons. Ce modèle s'appelle le XGBoost.

10. Le XGBoost

Le modèle XGBoost est une version optimisée du modèle de gradient boosting. Elle intègre des fonctions supplémentaires de régularisation et d'optimisation.

Ainsi, nous avons la fonction objective que l'on cherche à minimiser, tel que :

$$Obj(\theta) = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Où :

L est une fonction de perte

$\Omega(f_k)$ Un terme de régularisation pour l'arbre k

n est le nombre d'observations

K est le nombre d'arbres

Et pour un problème de classification multi-classe (ce qui nous intéresse) la fonction de perte devient un softmax logloss tel que :

$$L(y_i, \hat{y}_i) = -\sum_{k=1}^{K} 1\{y_i = k\} log \frac{e^{\hat{y}_{ik}}}{\sum_{k'} e^{\hat{y}_{ik}'}}$$

Où les prédictions $\hat{y}_i$ devinent des vecteurs $\hat{y}_i = (\hat{y}_{i1},...,\hat{y}_{iK})$ pour K classes.

La régularisation ici, s'applique au poids des feuilles et non au coefficients directement. Ainsi, la régularisation L2 pour un arbre f est définie tel que :

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2$$

Où :

T est le nombre de feuilles

ω_j^2 est le score de la feuille j

γ est le coût de complexité

Et la régularisation L1 pour un arbre f est additionner au L2, définie tel que :

$$\Omega(f) = \gamma T + \alpha \sum_{j=1}^{T} \left|\omega_j^2\right| + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2$$

Alors, pour utiliser seulement la régularisation L2 il faut définir λ comme nul.

Cela s'additionne donc à l'apprentissage additif, tel que :

$$Obj^{(t)} = \sum_{i=1}^{n} L\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t)$$

Pour rappel, l'ajout d'arbre à chaque itération est pour corriger les résidus, tel que :

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

L'optimisation de la perte de la fonction objective se fait en trois étapes :

1. D'abord le calcule des dérivées :
    1. Le calcule de la dérivée première (qui est la dérivée gradient) avec :

    $$g_i = \partial_{\hat{y}^{(t-1)}} L\left(y_i, \hat{y}_i^{(t-1)}\right)$$

    Pour un problème de classification multi-classe chaque dérivée gradient est calculé pour chaque classe k, tel que :

    $$g_{ik} = \frac{\partial L\left(y_i, \hat{y}_i^{(t-1)}\right)}{\partial_{\hat{y}_{ik}^{(t-1}}}$$

    2. Le calcule de la dérivée seconde (qui est la dérivée hessienne) avec :

    $$h_i = \partial^2_{\hat{y}_i^{(t-1)}} L\left(y_i, \hat{y}_i^{(t-1)}\right)$$

    De même pour un problème de classification multi-classe de la dérivée hessienne :

    $$h_{ik} = \frac{\partial^2 L\left(y_i, \hat{y}_i^{(t-1)}\right)}{\partial\left(\hat{y}_{ik}^{(t-1)}\right)^2}$$

2. Faire une approximation de Taylor d'ordre 2 de la fonction de perte pour obtenir une fonction quadratique, tel que :

$$Obj^{(t)} \approx \sum_{i=1}^{n} \left[L\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t)$$

3. Optimiser cette fonction en supprimant les constantes pour trouver son minimum analytique.

$$Obj^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t)'''$$

Concernant les conditions d'arrêt, le score optimal d'une feuille ω_j^* dans une structure d'arbre fixée peut se calculer tel que :

$$\omega_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

Où

I_j Est l'ensemble des indices des observations qui appartiennent à la feuille j dans l'arbre.

Ainsi la valeur minimal de la fonction objective pour cette structure est :

$$Obj^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{\left(\sum_{i\in I_j} g_i\right)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T$$

Alors pour une division (split) de données qui sépare I en I_L et I_R, le gain est :

$$Gain = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma$$

Ainsi, le split ce fait seulement si $Gain > 0$.

Pour ce qui est des ensembles explicatives avec valeurs manquantes (I_M), il y a deux scénarios à tester, celui qui offre le meilleur gain est choisie comme direction par défaut pour ce nœud spécifique :

– Scénario 1 : Envoyer I_M vers la branche de gauche (I_L), tel que :

$$Gain = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L \cup I_M} g_i\right)^2}{\sum_{i\in I_L \cup I_M} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma$$

– Scenario 2 : Envoyer I_M vers la branche droite (I_R), tel que :

$$Gain = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R \cup I_M} g_i\right)^2}{\sum_{i\in I_R \cup I_M} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma$$

Et dans le cas où le modèle est entraîné sur des données sans valeurs manquantes mais en rencontre lors de la prédiction, il existe une direction par défaut globale. En général, il enverra les valeurs manquantes vers la branche droite.

À noté que la gestion des valeurs manquantes comporte un risque. En effet, cela peut engendrer des nœuds qui contient que des données manquantes et ainsi nuire à l'interopérabilité et à la performance.

Cas pratique :

Régulariser le modèle avec L2 tout en conservant la suppression des données manquantes et en optimisant la perte sur la fonction objective. Puis dans un second temps conserver les données manquantes et comparer les performances.

Cette fonction prend plusieurs paramètres :

– N_estimators : qu'on à déja vu. C'est le nombre d'itérations. On gardera le même intervallle de recherche entre 50 et 300.
– Max_depth : aussi déja vu. C'est la prondeur max des arbres comme conditions d'arrêt. On gardera le même intervalle si il n'y pas de contre-indication, c'est à dire [1 ; 5].
– Learning rate : déja vue. On laisse le même intervalle de recherche.
– Objective : pour définir la fonction de perte. Notre fonction de perte est la softmax. Donc, multi :softprob.
– Booster : définit la technique de boosting. Rester sur celui par défaut.
– Gamma : paramètre de régularisation pour les arbres. Cela définit un seuil de gain minimal pour effectuer un split. Laisser la valeur par défaut.
– Early_stopping_rounds : définit une limite d'itération sans amélioration de performance avant l'arrêt. Commencer par une intervalle de 5 à 10.
– Définir lambda et et alpha
– Conserver la pondération temporelle et de classes

Résultat :

Pas d'amélioration significatives des résultats :

– nous tournons autour d'un score qualité de 0,33,
– Avec un log-loss d'à peu près 1
– Et environ un score F1 de 0,56.

Un des problèmes de tout les modèles que j'ai tester jusqu'à maintenant est qu'ils considèrent chaque observations comme indépendantes, malgré la pondération temporelle que j'ai implémenté. Même en ayant utilisé des features pondérées par le temps, ces modèles ne capture pas les séquences des matchs. Or, certaines tendances dépendent de l'historique récent : par exemple, après une défaite, une équipe peut avoir plus de chances de perdre le match suivant, même si sa probabilité globale est favorable. Pour modéliser ce type de dépendance temporelle, les réseaux de neurones récurrents, et plus spécifiquement les LSTM sont plus appropriés.

11. Modèle de Long Short-Term Memory (LSTM)

Un LSTM est un type de réseau de neurones récurrents (RNN) spécialement conçu pour traiter des données séquentielles et capturer des dépendances à long terme. Contrairement, aux RNN standards qui souffrent du problème de "vanishing gradient", les LSTM peuvent mémoriser des informations sur de longues périodes.

Le problème du **vanishing gradient** est une difficulté récurrente dans les modèles RNN. Il survient lorsque, au cours de l'apprentissage, le modèle ajuste les gradients en fonction des pertes observées afin de minimiser la fonction objectif. À mesure des itérations, les coefficients multiplicateurs appliquées aux gradients deviennent de plus en plus petits, jusqu'à tendre vers des valeurs presque nulles. À l'inverse, il peut également se produire un phénomène opposé appelé exploding gradients, où les valeurs gradients deviennent excessivement grandes.

Pour remédier au problème du vanishing gradient, le modèle LSTM a été conçu grâce à l'introduction de portes dans ses cellules, il parvient à maintenir un niveau d'information quasi constant au fil du temps. Ces portes régulent la propagation des gradients en agissant de manière ciblée sur les données unitaires, plutôt que sur l'ensemble du réseau, ce qui stabilise l'apprentissage.

Toutefois, cela ne supprime pas totalement le problème, mais prolonge considérablement la capacité de mémoire du réseau. Je montrerais par la suite que cette caractéristique peut présenter certains avantages.

Un modèle LSTM est composé d'une couche contenant un ensemble fixe de cellules fonctionnant en parallèle. Chacune de ces cellules prend en compte l'ensemble des features explicatives (X) à chaque pas de temps et produit une sortie scalaire représentant une analyse spécialisée. Ces sorties sont ensuite combinées pour former le vecteur d'état caché. De plus chaque cellule est réutilisée séquentiellement sur toute la longueur de la séquence de temps, et est composé :

– D'un état mémoire (cell state), noté C_t, qui est l'information capturer par la cellule à un instant t, en plus de l'ensemble des informations précédemment capturées, tel que :

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Où :

$\odot$ représente le produit élément par élément (Hadamard product).

f_t : est la probabilité de pertinence des informations, destinées à déterminer ce qui dois être oublier dans les anciennes informations (appelée porte de l'oublie).

C_{t-1} : est la vecteur d'informations qui précédent celui en cours.

i_t : est la probabilité de pertinence de l'information, destinée à déterminer ce qui dois être apprise dans les nouvelles informations (appelé porte d'entrée).

$\tilde{C}_t$: est vecteur d'information représentant les nouvelles valeurs candidates pour l'état de la cellule, calculées à partir des entrées actuelles.

– D'un état caché (hidden state), noté h_t, qui est l'information que renvoie la cellule à un instant t, basé sur C_t. Autrement dit, l'interprétation produit en fonction de l'état C_t, tel que :

$$h_t = o_t \odot tanh(C_t)$$

Où :

o_t : est la probabilité de pertinence de l'information, destiné à déterminé l'information à révélé (appelé porte de sortie).

$tanh$: est la tangente hyperbolique

– De trois types de portes (gates), qui contrôle le flux d'information, tel que :

- La porte de l'oubli (Forget gate) : détermine si l'information h_{t-1} doit être oubliée ou garder selon la probabilité. Autrement dit, elle pondère h_{t-1} e fonction de sa pertinence, calculer tel que :

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Où :

x_t : est un vecteur d'entrée au temps t, (soit l'information brute courante).

h_{t-1} : état caché précédent

σ : fonction sigmoïde

W_f : matrices de poids à initialiser

b_f : est le biais à initialiser

$[h_{t-1}, x_t]$: est la concaténation des deux vecteurs

- Porte d'entrée (Input gate) : détermine si l'information $\tilde{C}_t$ doit être stocker ou jeter selon la probabilité. Autrement dit, elle pondère $\tilde{C}_t$ selon sa pertinence, tel que :

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

Et :

$$\tilde{C}_t = tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right)$$

Donc :

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

- Porte de sortie (Output gate) : détermine quelle partie de l'état de la $tanh(C_t)$ doit être transmis à h_t, tel que :

$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right)$$

Donc :

$$h_t = o_t \odot tanh(C_t)$$

La tangente hyperbolique (tanh) est la représentation d'une intensité normalisé entre -1 et 1 et préserve le signe de son entrée, tout en ayant des gradients forts et une moyenne nul. Voici la définition mathématique :

$$tanh(x) = \frac{\left(e^{x} - e^{(-x)}\right)}{e^{x} + e^{(-x)}}$$

Ou équivalent avec sigmoïde :

$$tanh(x) = 2 \times \sigma(2x) - 1$$

Ainsi, $tanh$ est préférer à la sigmoïde pour $\tilde{C}_t$, car elle permet de représenter au mieux les deux types d'état : affirmation et négation d'une question fermé.

Cas pratique :

L'objectif de ce cas pratique est de prédire l'issue d'un match à partir des caractéristiques des deux équipes. Chaque équipe est décrite à travers l'historique de ses rencontres, incluant pour chaque match :

– Le résultat du match, (Y)
– Les cotes d'ouverture et de clôture pour chaque issue possible,
– L'intervalle entre la date d'émission des cotes et le début du match,
– L'intervalle entre la date du match et la fin de saison,
– Le cosinus/sinus pour représenté les cycles calendaires,
– L'historique de la compétition disputée. (incluant l'intervalle entre la date du match et le dernier match de la compétition).
– Ainsi que l'historique des deux équipes impliqué dans la rencontre

Étant donné que le principal atout du modèle LSTM réside dans sa capacité à exploiter les séquences temporelles, il n'est plus possible de supprimer les observations incomplètes, comme cela était fait dans les modèles précédents, car cela casserait la continuité des séries.

Pour palier ce problème, j'introduirais des variables indicatrices (features binaires, 0 ou 1) signalant la présence ou l'absence de côtes, et remplaceras les valeurs manquantes par la moyenne de la médiane des côtes pour l'issue spécifique des deux équipes d'une rencontre dans une compétition donnée, plutôt que d'éliminer l'observation. La médiane permet de faire comprendre au modèle que l'information est manquante mais qu'on suppose qu'elle est similaire au autres valeurs observé.

Ainsi, la structure du jeu de données doit être entièrement repensée. Alors qu'elle était auparavant organisée par compétition et par saison, elle doit désormais être regroupée par équipes afin de préserver la cohérence temporelle des séquences.

Concernant le problème du vanishing gradient, il n'est pas particulièrement critique dans notre cas, car les informations anciennes ne sont pas déterminantes en soi, c'est la récurrence temporelle qui importe. Si certaines informations passées sont "oubliées", c'est qu'elles sont

trop anciennes ou peu représentatives de la dynamique actuelle des équipes et donc qu'elles méritent donc d'être oubliées.

Le processus d'entraînement du modèle se déroulera comme suit :

1. Récupération des données : Collecter l'historique des rencontres des dix dernières saisons pour toutes les équipes et compétitions concernées.
2. Préparation des matrices : construire une matrice de données pour chaque équipe et chaque compétition, en intégrant les variables indicatrices pour les valeurs manquantes.
3. Représentation vectorielle : Générer, pour chaque équipe et chaque compétition, des vecteurs h_t de taille fixe (par exemple 64), résumant l'état des performances à un instant t. Ces vecteurs serviront à prédire le résultat du match suivant. Les hyper-paramètres (taille du vecteur, etc.) seront optimisés via une approche bayésienne.
4. Construction des features de match : Pour chaque match, concaténer les trois vecteurs correspondantes aux deux équipes participantes et à la compétition disputée.
5. Découpage des données : Séparer les données en ensembles d'entraînement, de la validation et de test
6. Optimisation des batchs et epoch : Diviser chaque matrice d'équipe en plusieurs lots (par exemple 50 à 100 matchs par batch) afin d'accélérer l'entraînement et de réduire la consommation de ressources. Et atteindre un nombre d'entraînement optimal.
7. Pondération de classes : pondérer les classes en fonction de leurs fréquences, de sortes que le poids soit définit proportionnellement à leurs poids inverses.
8. Entraînement progressif : Entraîner le modèle lot par lot sur l'ensemble d'entraînement, ajustant les hyper-paramètres selon les performances observées sur l'ensemble de validation.
9. Évaluation : évaluer la performance finale du modèle sur l'ensemble de test.

Résultat :

Résultat impossible à mesurer car le modèle prédit à chaque fois la classe majoritaire. Cela est peut être dû au nombre élevé de features par rapport au nombres d'observations. En effet le nombres de features est aux alentours de 200-300 tandis que les observations sont entre 10 et 15 000. Le modèle n'arrive donc peut être pas a trouver de récurrences suffisamment stables pour être généralisé. Je mets par conséquent ce type de modèle sur le côté et passe à un modèle ensembliste, pour tenter de combler les défauts individuelles des différents modèle que j'ai vue jusqu'à maintenant.

12. L’apprentissage ensembliste

L’apprentissage ensembliste consiste à combiner les prédictions de plusieurs modèles afin d’obtenir un résultat plus performant qu’un modèle individuel.
Pour cela, on introduit un méta-modèle (ou meta-learner), chargé d’apprendre comment pondérer ou sélectionner les prédictions des modèles de base pour produire la sortie finale.

Les principales étapes du processus sont les suivantes :

1. Division du jeu de données
   Le jeu de données est divisé en plusieurs parties, notamment un ensemble destiné à l’entraînement du méta-modèle et un autre réservé à son évaluation finale.

2. Entraînement du méta modèle
   Le méta modèle apprend à généraliser les performances des modèles de base à partir de leurs prédictions sur la première partie du jeu de données.
   Cet apprentissage est souvent réalisé à l’aide d’une validation croisée afin d’éviter le sur-apprentissage et d’assurer une bonne capacité de généralisation.

3. Évaluation du méta-modèle
   Le méta-modèle est ensuite testé sur la seconde partie du jeu de données, restée indépendante, afin d’évaluer sa performance réelle.

4. Mise à jour continue du méta-modèle
5. Plutôt que de ré-entraîner régulièrement les modèles de base, il est possible de mettre à jour uniquement le méta-modèle au fur et à mesure que de nouvelles données apparaissent.
   Cela permet d’adapter le système aux changements de contexte ou distribution des données tout en réduisant les coûts de ré-entraînement.

Cas d’usage :
Les modèles de bases sont des modèles de régression logistique binaire, dont chacun est chargé de prédire un issue de match parmi les suivantes :
– victoire domicile,
– nul,
– victoire extérieur,
À partir des features suivantes :
– cote domicile à la l’ouverture et à la fermeture,
– cote nul à la l’ouverture et à la fermeture,
– cote domicile à la l’ouverture et à la fermeture.
J’applique une pondération de classe à chacun de ses modèles pour palier au déséquilibre de classe.
L’hyper-paramètre lambda idéal est trouvé via une Grid recherche pour maximiser les performance.

Cela donne les performances suivantes pour chacun des modèles :

Model	LogLoss	Score_log	Score_F1	Q
Home	0.685982	0.593126	0.536986	0.565056
Draw	0.688959	0.592081	0.419540	0.505810
Away	0.679807	0.595307	0.400000	0.497653

Ensuite, j'applique par dessus un méta modèle qui aura pour rôle de définir et choisir la prédiction d'un modèle de base qui aura le plus de chances réelles de se produire dans un cas de figure particulier. Ce choix se fera en prenant en compte les features suivantes :

– Probabilité de l'issue : victoire à domicile du modèle correspondant
– Probabilité de l'issue : nul du modèle correspondant
– Probabilité de l'issue : victoire à l'extérieur du modèle correspondant

Ainsi, j'opte pour la régression logistique multi-classe et tout comme les modèles de bases j'applique la pondération de classe et la recherche de l'hyper-paramètre lambda via grid.
En plus de cela j'ajoute une méthode pour préservés le ratio générales des classes après pondérations au niveau des folds d'entraînement et de validations. Cela permet de palier au manques potentiel de donner. Puisque le dataset est divisé plusieurs fois.

Malgré cela, les performances sont assez mitigé, en particulier la log-loss qui n'est pas mieux que l'aléatoire :

Model	LogLoss	Score_log	Score_F1	Q
Meta	1.095798	0.477145	0.391517	0.434331

Cela démontre une faible discrimination du modèle et une mauvaise calibration, causé, sans doute par un signal faible des features (que 3 features et non exclusives) et un biais sur les modèles de bases (victoire à domicile est de base un modèle plus performant).

Pour résoudre ces problématiques

1. je calibre les probabilités des modèles de bases pour donner un meilleur signal aux features,
2. Normalise les probabilités des modèles, pour donner du contexte et limité les biais.
3. Générer des observations réaliste par rapport à ma distribution pour palier au faible nombre dans certains jeu, car la pondération de classe ne suffit plus.

Au passage, j'ai aussi normaliser les côtes, en passant des côtes brutes aux probabilité implicites de l'issue.

Ainsi, voici les résultats des modèles de bases :

Model	LogLoss	Score_log	Score_F1	Q
Home	0.645531	0.607707	0.642202	0.624954
Draw	0.694797	0.590041	0.386364	0.488202
Away	0.696052	0.589604	0.356164	0.472884

Et voici les résultats du méta-modèle :

Model	LogLoss	Score_log	Score_F1	Q
Meta	0.982227	0.504483	0.501805	0.503144

J'ai décider de me focaliser sur l'amélioration de la log loss du méta-modèle, car c'est la métrique principale qui permet de mieux mesurer la satisfaction de mon objectif. J'ai donc, choisis de calibrer ses probabilités et de modifier Q pour qu'ils prennent en compte, non plus Score_F1 mais l'Average_precision_score.

Toutefois, il faut savoir que j'utiliserais les côtes de clôture comme features pour paramétrer le modèle, mais qu'en réalité lors de la prédiction, je n'utiliserais que rarement les côtes de clôture. J'utiliserais à la place des côtes intermédiaire entre la côtes d'ouverture et fermeture. Ainsi, j'ajouterais une pondération temporelle aux côtes, relatif à l'écart du début du match. Cela permettra au modèle de prendre en compte cette situation et tenter de prédire du mieux possible en fonction de celle-ci.

Les 3 feaures de pondération temporelle sont :

– Delta : qui permet de prendre en compte le mouvement absolue de la côte
– ratio : prend en compte le mouvement relatif de la côte
– Time_to_match : ajoute du contexte au deux features

J'ajoute aussi deux autres types de features :

– Les features de confiance
– Les logits

Puis je calibre le modèle, en utilisons la méthode qui convient. Pour finir, j'ai décider de ne plus entraîner les modèles par compétitions mais de tout assembler pour entraîner des gros modèles avec toutes les données disponible. Cela à considérablement améliorer la calibration visible sur la courbe de calibration, même si elle n'a pas modifier la log-loss.

Voici les résultat du méta modèle :

Model	LogLoss	Score_log	Score AP.	Score_F1	Q
Meta	0.97792	0.505582	0.495751	0.387529	0.500666

Pour rappel, le score qualité (Q) prend maintenant en compte le score d'average precision à la place du score F1 pour une meilleur représentativité de l'objectif.

13. Calibration du modèle de régression

J'ai plusieurs modèle de régression logistique probabiliste. La calibration sert à calibrer les probabilités des modèles. C'est à dire ramener les probabilités prédites proche des probabilités réelles.

Pour cela, il y a plusieurs étapes :

1. Définir la métrique d'évaluation du niveau de calibration. Il y a deux type de métrique :
    a. Courbe de calibration : pour un diagnostic visuel

        L'avantage est qu'il est facile à comprendre et à interpréter mais il est subjectif.

    b. Score Brier/Log-Loss : qui sont des métriques quantitatives
        i. Score Brier : l'avantage est qu'il est comparable facilement et qu'il punit sévèrement les erreurs de confiance. Cependant, il est moins sensible aux probabilité extrême, c'est à dire qu'il pénalise moins les erreurs de prédictions très confiantes. De plus il dépend de la prévalence (proportion) des classes dans le jeu.
        ii. Log-Loss : L'avantage est que contrairement au score Brier, il est très sensible aux probabilités et est moins influencé par la proportion des classes.
2. Diviser les données en 3 jeux :
    a. entraînement : servira à entraînement comme habituellement
    b. validation : servira à ajuster les hyper-paramètres en fonction des métriques choisit
    c. Test : servira à tester les performances finales
3. Calibrer les hyper-paramètres avec des méthodes d'optimisations d'hyper-paramètres que l'on ne présente plus, tel que : Grid Search, Random Search, l'optimisation Bayésienne ou encore la validation croisée.

Toutefois, c'est toutes les étapes que l'on déjà pour chaque modèle lors des différentes optimisation que l'on propose. Je peux éventuellement ajouter la métrique visuelle.

Finalement ce que j'ai fais :

– Augmenter artificiellement le nombre d'observations de la classe "nul" avec SMOTE, car elle est fortement inférieur au deux autres.
– Calibrer avec la méthode sigmoïd (Platt scaling)

Le log loss toujours élevé, je vais tenter de fournir une représentation relative des équipes et des compétitions aux modèles, pour leurs permettre peut être de mieux discriminer et ainsi améliorer leur log-loss.

14. Embeddings

L'embedding (ou « prolongement lexical » en français) est une technique de représentation des données qui transforme des objets discrets (comme des mots, des images, des entitées) en vecteurs continus dans un espace de dimension réduite. De sortes à ce que des objets similaires aient des représentations vectorielles proches dans un espace d'embedding.

Cela a résolu le problème de la représentations one-hot, qui consiste à représenter un objet avec une taille fixe qui contient un seul 1 et le reste 0. Chaque objet avait le 1 placer à des coordonnées spécifique du vecteur fixe. Cela posait en effet, plusieurs problèmes :

– Plus il y avait d'objets à représenter et plus la taille du vecteur augmenter,
– Aucune similarité entre deux vecteurs,
– Par conséquent, cela ne capturé pas de sémantique.

La fonction d'embedding est une fonction E qui map un mot w à un vecteur v, tel que :

$$E(w) = V \in R^d$$

Où R^d est un espace vectoriel de dimension d et $d \ll V$ (signifie que V est très inférieur à d. Ce qui règle le problème de la taille.

Similarité cosinus : La similarité entre deux vecteurs d'embedding se mesure par :

$$similarité(u, V) = \frac{(u \cdot V)}{(\|u\| \ \|V\|)}$$

Où :

$\|u\|$ et $\|V\|$ sont la norme (ou longueur) de leur vecteur respectif, tel que pour un vecteur $u = [u_1, u_2, \ldots, u_n]$ dans R^n. Elles se mesure selon plusieurs normes :

– La norme euclidienne (L_2), qui est la plus courante :

$$\|u\|_2 = \sqrt{\left(u_1^2 + u_2^2 + \ldots + u_n^2\right)}$$

– La norme L_1 (Norme Manhattan) :

$$\|u\|_1 = |u_1| + |u_2| + \ldots + |u_n|$$

Où $|u_n|$ est la valeur absolue de u.

– La norme L_p (généralisation) :

$$\|u\|_p = \left(|u_1|^p + |u_2|^p + \ldots |u_n|^p\right)^{\left(\frac{1}{p}\right)}$$

Où $1 \leq p \leq +\infty$ ici, un paramètre réel, qui détermine le type de norme et influence comment on mesure la magnitude du vecteur.

– La norme infini (L_∞) ou norme max :

$$\|u\|_\infty = \max\left(|u_1|, |u_2|, \ldots, |u_n|\right)$$

Où $\|u\|_\infty$ est la plus grande valeur de l'ensemble u

On choisit la norme en fonction de plusieurs facteurs :

– Norme L_2 si la magnitude est importante. Elle permet les vecteurs similaires. Ex : système de recommandation, qui recommande des articles similaires à ceux acheter et aimés. La magnitude est importante car l'intensité des notes est importante pour évaluer et comparer le niveau de satisfaction des utilisateurs.
– Norme L_1 si les données sont creuses. Elle permet de représenter la présence ou l'absence d'objets spécifiques dans les vecteurs, car l'absence est représenter par 0 (données creuses) et la présence par 1. Ex : permet de dire si un mot contient la représentation d'autres mots, tel que : smartphone, qui est un vecteurs dans lequel les vecteurs des smart et phone se superpose.
– Norme L_∞ si le contrôle de qualité est rechercher : permet de capturer la valeur la plus anormales. Ex : plusieurs capteurs sur une machine, elle détecte le capteur qui diverge le plus des autres.
– Norme L_p pour les cas spécifique (c'est une norme générale) : permet pr exemple, une recommandation personnalisé avec pondération.

Donc, la formule de base est tel que pour un mot W_i, son embedding e_i est obtenu par :

$$e_i = W \times x_i$$

Où :

$W \in R^{(d \times V)}$ est la matrice d'embedding

$x_i = R^V$ est le vecteur one-hot du mot

Ainsi, les méthodes d'apprentissages des embeddings pour des données texte, sont :

– Word2Vec (Mikolov et al., 2013) :
  - architecture Skip-gram. Le principe est de prédire les mots à partir d'un mot cible. Dont la fonction objective est :

$$J(\theta) = \frac{1}{T}\sum \log P\left(w_{\{t+j\}} | w_t\right)$$

Où :

T est le nombre total de mots dans le corpus

t, la position de chaque mot

j, est la position d'un mot par rapport à un mot référence

  - Architecture CBOW (continuous Bag of Words), dont le principe est de prédire un mot à partir de son contexte.

– GloVe (Global Vectors) : est une approche basée sur la factorisation matricielle, dont la fonction objective est :

$$J(\theta) = \sum f(X_{ij})\left(w_i \cdot w_j + b_i + b_j - \log x_{ij}\right)^2$$

Où X_{ij} est la fréquence de co-occurrence des mots i et j.

Il y'a d'autres méthodes pour d'autres types de données :

– Node2Vec pour les données en Graph
– CNN embedding pour les données d'images
– Les séries temporelles pour des besoins de temporalité

Concernant, la méthode Word2Vec et GloVe, il faut savoir que cette première est utile dans le cas :

– D'un corpus de taille petite à moyenne,
– Pour des besoin de similarité locale/contextuelle,
– Des ressources computationnelle limitées,
– Apprentissage incrémental nécessaire,
– Besoin de bonne relations syntaxiques (c-à-d dire que l'organisation des mots doit être grammaticalement correcte).

GloVe quant à lui est utile dans le cas :

– De très grands corpus,
– Pour un besoin de capture statistiques globales,
– Importance des co-occurrences à longue distance
– Bonne relations sémantiques (c-à-d que l'organisation des mots/phrases doit avoir du sens)
– Stabilité des résultats.

Architecture CBOW de la méthode Word2Vec est à utiliser dans le cas :

– De petit ou moyen corpus,
– De données bruitées,
– De performance computationnelle critique,
– De mots fréquents dominants,
– pour des tâches de classification et d'analyse de sentiments (comme les émotions, opinions et tout ce qui est abstrait)

Et l'architecture Skip-gram de la méthode Word2Vec doit être utiliser dans le cas :

– D'un corpus large,
– De mots rares importants,
– Où la précision sémantique est prioritaire,
– Où les ressources computationnelles sont suffisantes
– Et pour des des tâches de similarité, recherche, recommandation.

Les hyper-paramètres à optimiser, sont :

– La dimension d : typiquement entre 50 et 300,
– La taille de fenâtre contextuelle : 2-10
– Et le nombre d'époques : 5-50.

Cas d'usage :

J'ai plusieurs features par observations/matchs, qui permettent de prédire l'issue d'un match. L'objectif ici est d'affiner ce résultat en incluant une représentation des équipes prenant parti au match et de la compétition disputée. Ces représentations se feront par embedding.

Ainsi, avec pytorch :

1. Je créer une matrice d’équipes et une autre de compétitions, contenant toutes les embeddings , dont chaque ligne correspond à un vecteur d’embedding.
2. J’entraîne le modèle d’embeddings en amont des modèles de prédictions.

Au final aucune amélioration exploitable n’a été constaté. Je vais donc essayé un apprentissage par backpropagation.

15. Apprentissage bout en bout avec entraînement conjoint

End-to-end learning with join training, en anglais, signifie que les modèles d'embedding, les trois modèles de base et le méta-modèle seront entraîner simultanément dans un pipeline entièrement différenciable où la log-loss finale est rétro-propagée à travers l'ensemble des composants. L'ensemble constitue un unique réseau neuronal multi-branches.

Composants initiaux :

– Modèle d'embedding
– Modèles binaires de régression logistique de base :
    - M1 : Home
    - M2 : Draw
    - M3 : Away
– Méta-modèle de régression logistique multi-classe

Le processus se déroule ainsi :

1. Définir les hyper-paramètres pour chacun des modèles.
2. Modèle d'embedding : calcule des embeddings à partir des ids des équipes et compétitions, pour prédire le résultat de chaque matchs
3. Modèles de base : produisent des prédictions en utilisant à la fois les features brutes et les ambeddings.
4. Méta-modèle : génère des prédictions finales à partir des features brutes, des embeddings et des probabilités des modèles de base.
5. Calculer la log-loss de l'ensemble de la séquence.
6. Rétro-propagation : ajuster tous les poids à partir de la log-loss calculée.

Répéter les étapes 2 à 6 jusqu'à convergence, puis ajuster de nouveaux hyper-paramètres et relancer le processus pour optimiser encore les performances.