Joey Lévesque

111-219-341

Sam Chandavong
111-026-119

Corentin Labelle
111-132-133

Analyse et traitement de données massives

GLO-7027

Projet de session rapport 2

Travail présenté à

Richard Khoury

Département d’informatique et de génie logiciel

Université Laval

Hiver 2023

Sam Chandavong, Corentin Labelle, Joey Lévesque

Abstract

Ce rapport présentera les algorithmes de traitements de données qui nous auront permis d'analyser les 30 millions d'entrées du jeu de données des admissions des patients ainsi que de leurs prescriptions à un temps donné. L'analyse inclura aussi les méthodes que nous avons utilisées afin d'augmenter le taux de précision de l'algorithme dans une discussion sur les tests exécutés. Dans notre contexte, l'arbre de décision XGBoost aura été l'algorithme le plus promettant. Notre modèle aura atteint les meilleurs résultats avec des entrées dans le jeu de données mélangées et avec le retrait de l'identité des patients pour en faire des variables indépendantes et identiquement distribuées. Malgré la précision de 80%, le ROC (Receiver Operating Characteristic) de 66% nous permet de conclure que le modèle a appris quelque chose.

1 Analyse des algorithmes utilisés

1.1 ANALYSE EN COMPOSANTES PRINCIPALES

Nous avons tenté d'appliquer une analyse en composantes principales pour tenter de réduire le nombre de médicaments. Que ce soit sur le jeu de données entier ou sur différentes partitions (en ne gardant, par exemple, que les hospitalisations ou que les polypharmacies), les analyses en composantes principales ont montré que tous les médicaments étaient pertinents dans l'explication de la variance. Cette analyse ne nous permet donc pas de diminuer le nombre de dimensions. Par la même occasion, nous avons essayé via une étape de prétraitement de combiner des entrées rapprochées temporellement (moins de 3 jours, moins de 7 jours, moins de 14 jours, etc.) à l'hôpital afin de regrouper les médicaments donnés à différentes périodes dans le temps. Les résultats obtenus n'étaient pas satisfaisants puisque cela créait une entrée dans le jeu de données où beaucoup de médicaments (voir tous les médicaments) ont été prescrits ce qui compliquerait la recherche de polypharmacies.

1.2 ALGORITHME À PRIORI

L'objectif d'appliquer l'algorithme à priori était de cibler des combinaisons de médicaments étant régulièrement prescrits ensemble. L'algorithme a tout d'abord été appliqué au jeu de données en entier afin de cibler des combinaisons de médicaments régulièrement présents, indépendamment d'une hospitalisation. Cela pourrait, par exemple, nous permettre de combiner un ou plusieurs médicaments en un seul attribut et ainsi réduire le nombre total d'attributs. Nous avons établi un seuil de 10%, considérant que des combinaisons ayant une fréquence inférieure à ce seuil ne devraient pas être traitées différemment. Sans surprise, chaque médicament a été identifié comme étant un 'itemset' fréquent (leurs supports est d'environ 0,22). Cependant, aucune combinaison de médicament n'a été identifiée.

	support	itemsets
0	0.226268	(drug_0)
1	0.226525	(drug_1)
2	0.226013	(drug_2)
3	0.225887	(drug_3)
4	0.226242	(drug_4)
5	0.226151	(drug_5)
6	0.226427	(drug_6)
7	0.226069	(drug_7)
8	0.226657	(drug_8)
9	0.226576	(drug_9)
10	0.226403	(drug_10)
11	0.226470	(drug_11)
12	0.226232	(drug_12)
13	0.226180	(drug_13)
14	0.226129	(drug_14)
15	0.226040	(drug_15)
16	0.225780	(drug_16)
17	0.226022	(drug_17)
18	0.226103	(drug_18)

Figure 1.2.1 Support des médicaments individuels

L'algorithme à priori a ensuite été appliqué à une partition regroupant les prescriptions prises au moment d'une hospitalisation. Un patron fréquent dans ce jeu de données pourrait être une première indication d'une combinaison de médicament potentiellement nocive. Cependant, avec un seuil de 10%, aucune combinaison de médicament ne fut identifiée comme étant significativement plus élevée que les autres.

	Support	itemsets
0	0.291076	(drug_0)
1	0.301033	(drug_1)
2	0.293298	(drug_2)
3	0.277398	(drug_3)
4	0.308430	(drug_4)
5	0.300674	(drug_5)
6	0.301072	(drug_6)
7	0.295476	(drug_7)
8	0.288834	(drug_8)
9	0.299553	(drug_9)
10	0.298202	(drug_10)
11	0.308594	(drug_11)
12	0.311130	(drug_12)
13	0.295114	(drug_13)
14	0.282762	(drug_14)
15	0.298973	(drug_15)
16	0.308077	(drug_16)
17	0.305025	(drug_17)
18	0.311031	(drug_18)

Figure 1.2.2 Support du regroupement des prescriptions prises au moment d'une hospitalisation

Nous avons ensuite utilisé l'algorithme sur une partition regroupant toutes les polypharmacies. Cette partition devrait augmenter les chances d'identifier des 'itemset' fréquentes. En effet, en retirant les prescriptions ne contenant aucune ou peu de médicaments, on augmente la proportion de combinaison de médicaments dans le jeu de donnée analysé. Encore une fois, avec un seuil de 10%, aucune combinaison de médicament ne fut identifiée.

	support	itemsets
0	0.414524	(drug_0)
1	0.414470	(drug_1)
2	0.413968	(drug_2)
3	0.413884	(drug_3)
4	0.414416	(drug_4)
5	0.414316	(drug_5)
6	0.414673	(drug_6)
7	0.414109	(drug_7)
8	0.415241	(drug_8)
9	0.415142	(drug_9)
10	0.414509	(drug_10)
11	0.414653	(drug_11)
12	0.414607	(drug_12)
13	0.414076	(drug_13)
14	0.414067	(drug_14)
15	0.413909	(drug_15)
16	0.413646	(drug_16)
17	0.413935	(drug_17)
18	0.413806	(drug_18)

Figure 1.2.3 Support des polypharmacies.

Finalement, l'algorithme à priori a été appliqué à une partition ne regroupant que les prescriptions d'une polypharmacie prises par un patient ayant été hospitalisé. Chaque médicament individuellement est présent dans 0,40 à 0,44 de ces prescriptions. Toutes les combinaisons de deux médicaments ont été identifiées (avec un support de 0,20) et toutes les combinaisons de trois médicaments ont été identifiées (avec un support d'environ 0,10). Aucune combinaison n'est significativement plus présente que d'autres.

En résumé, les médicaments et combinaisons de médicaments semblent équitablement répartis dans le jeu de données et ses partitions. Bien qu'on ne puisse cibler des « itemsets » fréquents intéressants, on pourrait potentiellement utiliser ces résultats pour corroborer les résultats obtenus avec d'autres algorithmes.

1.3 ARBRE DE DÉCISION

Pour implémenter le modèle d'un arbre de décision, nous avons utilisé le modèle DecisionTreeClassifier de la librairie sklearn.tree. Un arbre de décision est un modèle intéressant puisqu'il est facilement interprétable. On peut avoir accès à l'importance des attributs lors de la prise de décision du modèle. Cette fonctionnalité est pertinente dans notre cas, puisqu'au-delà d'un simple modèle permettant de prédire des hospitalisations, nous voulons comprendre la cause de ces hospitalisations et cibler les médicaments ou combinaisons de médicaments dangereux.

Le modèle prenait en entrée une liste de 19 booléens indiquant si le médicament est présent (1) ou absente (0) de la prescription. La cible indique s'il y a eu hospitalisation (1) ou non (0). Les données ont été séparées en données d'entraînement (70% des données totales) et en données de test (30%

des données totales) grâce à la fonction train_test_split de Scikit-learn. Les entrées du jeu de données ont aussi été mélangées afin qu'elles soient indépendantes et identiquement distribuées. Voici les résultats obtenus suite à l'entraînement :

```
Confusion Matrix
[[7 002 678  166 755]
 [1 770 327  126 346]]
Report
              precision    recall  f1-score   support

           0       0.80      0.98      0.88   7169433
           1       0.43      0.07      0.12   1896673

    accuracy                           0.79   9066106
   macro avg       0.61      0.52      0.50   9066106
weighted avg       0.72      0.79      0.72   9066106

Feature Importance
[0.04689528 0.05421395 0.0515581  0.04730032 0.03834274 0.03864629
 0.06929651 0.04627949 0.04889284 0.05623883 0.04138018 0.07280518
 0.05673414 0.04365583 0.03446564 0.04832778 0.04917793 0.05272977
 0.1030592 ]
70.98493242263794  secondes écoulées.
```

Figure 1.3.1 Matrice de confusion sans échantillonnage

La librairie scikit-learn permet de visualiser un rapport suite à l'entraînement d'un modèle. Ce modèle n'obtient pas de résultats convaincants et a rapidement été mis de côté. Avec la matrice de confusion, on remarque rapidement que le modèle classe la grande majorité des prescriptions (8 773 005) comme étant des prescriptions ne menant pas à des hospitalisations. Cela peut être expliqué par le fait que le jeu de données original contient beaucoup plus d'exemples de non-hospitalisation que d'hospitalisation. Environ 80% des entrées du jeu de données correspondent à des non-hospitalisations. Un tel débalancement de classe peut introduire un biais dans un modèle. Par exemple, le modèle pourrait classer toutes les prescriptions comme étant des prescriptions ne menant pas à des hospitalisations, et il obtiendrait de résultats respectables, sans n'avoir rien appris. Une solution pour pallier ce problème est de suréchantillonner la classe minoritaire, ou de sous-échantillonner la classe majoritaire. L'avantage de suréchantillonner la classe minoritaire est que nous ne perdons aucune information présente dans le jeu de données. En revanche, cela implique d'augmenter le nombre total d'échantillons lors de l'entraînement du modèle. À l'inverse, le sous-échantillonnage permet de diminuer notre nombre de prescriptions totales, ce qui facilite l'entraînement. Nous avons entraîné le modèle en utilisant ces deux méthodes afin de compenser le biais causé par le débalancement des classes. Dans les deux cas, nous voulions obtenir le même nombre d'hospitalisations que de non-hospitalisation.

```
Confusion Matrix
[[1249737  646936]
 [ 813418 1083255]]
Report
              precision    recall  f1-score   support

           0       0.61      0.66      0.63   1896673
           1       0.63      0.57      0.60   1896673

    accuracy                           0.62   3793346
   macro avg       0.62      0.62      0.61   3793346
weighted avg       0.62      0.62      0.61   3793346

Feature Importance
[0.04571993 0.05554232 0.05180329 0.0454326  0.06360088 0.03879685
 0.05814013 0.0422439  0.0484814  0.06693058 0.04395286 0.04787391
 0.09661985 0.04231718 0.03701303 0.04598712 0.0717869  0.04740794
 0.05034932]
30.174609899520874  secondes écoulées.
```

Figure 1.3.2 Matrice de confusion avec sous-échantillonnage

```
Confusion Matrix avec sur-échantillonnage
[[4596238 2573195]
 [2590002 4579431]]
Report
              precision    recall  f1-score   support

           0       0.64      0.64      0.64   7169433
           1       0.64      0.64      0.64   7169433

    accuracy                           0.64  14338866
   macro avg       0.64      0.64      0.64  14338866
weighted avg       0.64      0.64      0.64  14338866

Feature Importance
[0.03948602 0.05341953 0.04849481 0.04021831 0.03915463 0.03757081
 0.0757131  0.04801926 0.0439114  0.0520342  0.03896431 0.0830187
 0.06343297 0.04116783 0.03551536 0.04488505 0.05338387 0.04881006
 0.11279977]
130.1767613887787  secondes écoulées.
```

Figure 1.3.3 Matrice de confusion avec suréchantillonnage

Le suréchantillonnage et le sous-échantillonnage ne permettent pas d'obtenir de meilleurs résultats que lors de l'utilisation de l'entièreté du jeu de données. Nous avons testé ce même modèle avec différentes valeurs de 'max_depth' (la profondeur maximale de l'arbre), mais les meilleurs résultats étaient obtenus lors que ce paramètre était à 'None' (les nœuds sont étendus jusqu'à ce que toutes les feuilles soient pures). Un simple arbre de décision semble limité dans notre cas, et une solution combinant plusieurs arbres de décision devrait obtenir de meilleurs résultats.

1.4 XGBoost

Le dernier algorithme utilisé dans notre projet et qui a eu les résultats les plus prometteurs a été utilisé via la librairie XGBoost [1]. L'inspiration initiale nous est venue d'une publication du site Kaggle [2]. Ce site étant une plateforme en ligne hébergeant des compétitions de sciences des données. Cette librairie a déjà été utilisée dans un modèle pour prédire une situation similaire dans

un domaine de la santé [3]. Le modèle que nous avons utilisé est le “Gradient-Boosted Decision Tree (GBDT)” qui est souvent utilisé pour les problèmes de classification et d’apprentissage supervisé. Cela se prêtait très bien à notre situation puisque nos données sont étiquetées (nous connaissons les hospitalisations). L’idée est d’utiliser un ensemble d’arbres de décision. Chaque arbre fait une prédiction, et on obtient une prédiction finale en combinant toutes les prédictions des arbres. La descente de gradient permet d’optimiser les paramètres de chaque arbre. Le gradient est dérivé d’une fonction de perte. Dans notre cas, la fonction que nous avons utilisée est la fonction de perte logarithmique.

Lors de l’entraînement, nous avons adapté le jeu de données. Nous avons converti les timestamps en nombre de jours à partir du premier jour du jeu de données puisque l’algorithme n'est pas en mesure de bien comprendre le timestamp, nous avons aussi enlevé le concept d’id et mélanger les entrées dans le jeu de données afin de créer des variables indépendantes et identiquement distribuées. Mélanger les données nous a permis d’éliminer l’historique des patients qui créait un biais dans notre modèle.

L’entrée de notre modèle a été modifiée afin de trouver les attributs du modèle qui nous permettaient d’avoir les meilleures performances. La cible était toujours l’hospitalisation (ou la non-hospitalisation). Lors des différents tests, la principale métrique utilisée était le ROC (Receiver Operating Characteristic) qui sera expliquée dans la section 2.

La première méthode utilisée était l’approche la plus simple, soit d’avoir en entrée un vecteur binaire de 19 valeurs indiquant si le médicament est présent ou non. Avec cette approche, nous avons obtenu un ROC de 0.66588. La deuxième méthode utilisée prenait en compte les hospitalisations antérieures, c’est-à-dire si cette prescription a déjà causé une hospitalisation par le passé. Avec cette méthode, nous avons obtenu un ROC de 0,66424. La troisième méthode utilisée prenait en compte les modifications dans les prescriptions. Plutôt que d’avoir la prescription actuelle en entrée, nous avons modifié les prescriptions pour prendre en compte les changements dans le temps. Si un médicament était ajouté à une prescription, une valeur de 1 lui était attribuée. Si un médicament n’était pas modifié, une valeur de 0 lui était attribué. Finalement, si un médicament était retiré, une valeur de –1 lui était attribuée. Avec cette méthode, nous avons obtenu un ROC de 0.53652. La quatrième méthode utilisée prenant en compte l’effet des médicaments dans le temps. Cette méthode permettait de représenter le fait qu’un médicament introduit il y a un long moment aura un moins grand effet. On a utilisé une fonction de décroissance exponentielle ($2^{(-t/h)}$) ou t est le nombre de jours écoulés lors de la prescription de la ligne en cours depuis la première prescription du jeu de données et h une valeur définie pour voir comment le modèle réagit à ce phénomène nous avons essayé plusieurs valeurs de h pour ce modèle. À l’extrême, pour une petite valeur de h, ce modèle convergera vers le modèle le plus simple (le modèle ne prenant que les médicaments actuels). Avec cette méthode, nous avons obtenu un ROC entre 0.658 et 0.664. De la même façon, nous avons aussi essayé avec une formule de la courbe de Bell avec l’équation $2^{((t-d)^2/h)}$ ou t et h sont les mêmes valeurs que la fonction précédente et d (le délai) est une valeur fournie dans l’équation. Modifier n’importe lesquelles des valeurs ne changent peu ou pas le résultat.

De toutes les méthodes testées, celle qui obtenait les meilleurs résultats est la première, soit l’approche la plus simple.

2 Tests effectués

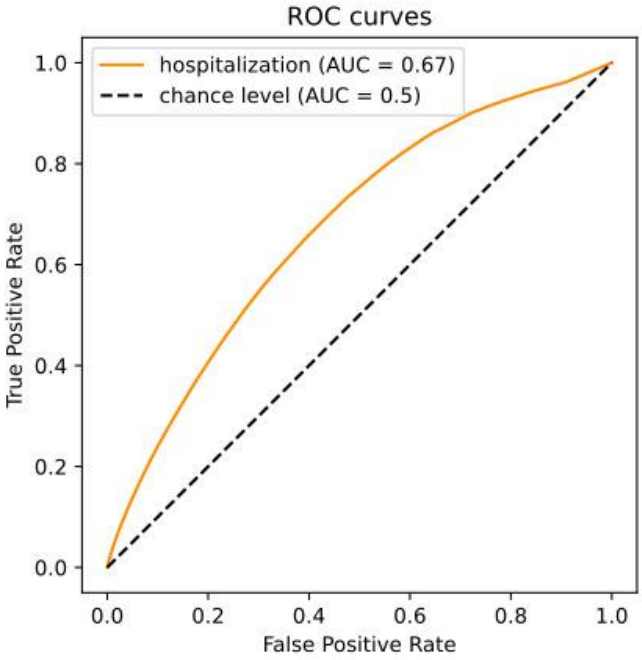

Figure 2.1 Courbe ROC

La courbe ROC a été utilisée pour évaluer notre modèle. La courbe ROC est une courbe de performance d'un modèle de classification binaire. Selon cette figure, on remarque que notre modèle est meilleur que la chance, ce qui confirme qu'il a fait un certain apprentissage. Cette courbe est aussi une référence pour choisir le seuil du modèle. Le seuil du modèle est ce qui permet de convertir une prédiction en décision. Par exemple, si le modèle prédit une hospitalisation à 52% et que le seuil est à 50%, alors la décision sera une hospitalisation. Cependant, si on augmente seuil à une valeur supérieure à 52%, la décision sera une non-hospitalisation. La valeur du seuil est donc très importante et nous permet de contrôler les performances de notre modèle par rapport au nombre de faux positif ou faux négatif. Nous avons finalement établi un seuil de 80%, autrement dit, le modèle prédit une hospitalisation lorsqu'il obtient une probabilité de 80% ou plus. Ce seuil correspond à la proportion d'hospitalisation dans le jeu de donnée entier et offre de bonnes performances.

Établir un seuil nous permet aussi d'obtenir des statistiques à propos des performances du modèle. Ce modèle obtient une exactitude de 0,7905, une précision de 0,7418, un rappel de 0,00014 et un F1 score de 0,00029. La valeur de l'exactitude n'est pas excellente. Puisque nous avons établi un seuil à 80%, le fait d'obtenir une exactitude de 79% signifie que l'on fait aussi bien que la chance. On peut aussi remarquer que la courbe ROC vient confirmer que le modèle a fait un certain apprentissage. On obtient une valeur très faible pour le rappel, ce qui indique que notre modèle prédit beaucoup de non négatifs.

C'est le point faible de ce modèle. Une recherche de paramètre plus poussée par exemple via l'hyperparamètre qui sera discuté dans la rétrospective du projet pourrait être effectuée afin d'augmenter le rappel, quitte à ce que cela affecte négativement l'exactitude ou la précision.

3 Études de cas

En annexe, il est possible de voir un exemple d'arbre généré par l'algorithme XGBoost (parmi tous les arbres faisant partie du modèle). Cet arbre permet de classifier une prescription, c'est-à-dire de prédire si elle mènera à une hospitalisation. À noter que le premier médicament correspond à l'indice 0. Comme étude de cas, prenons l'entrée 22 774 992 du jeu de données et parcourons le graphe. La prescription de cette entrée, en représentation binaire, est 1000110111010000100. Le premier nœud correspond au médicament 18. Si le médicament est présent, on emprunte le chemin de droite. Dans notre cas, le médicament 18 n'est pas présent dans la prescription, on emprunte donc le chemin de gauche. Le deuxième nœud correspond au médicament 6. Ce médicament n'est pas présent dans la prescription, on emprunte donc le chemin de gauche. Le nœud suivant correspond au médicament 11. Ce médicament est présent dans notre prescription, on emprunte donc le chemin de droite. Le nœud suivant correspond au médicament 15. Ce médicament n'est pas présent dans notre prescription, on arrive donc à une feuille ayant une valeur de –0,0920503512. Cette valeur est retournée par notre modèle. Dans le cas d'une classification binaire, cette valeur correspond au score brut de la classe 1 (hospitalisation). Cette valeur peut être convertie en probabilité en utilisant la fonction logistique. La valeur de –0,0920503512 correspond à une probabilité d'appartenance à la classe 1 de 0,477. La prédiction de cet arbre sera prise en compte lors de la prédiction finale du modèle.

Nous avons aussi accès à l'importance des médicaments dans la décision de la polypharmacie (voir annexe). On remarque que les médicaments importants sont les médicaments 12 et 18. Cet arbre corrobore cette information puisque la racine est la présence ou non du médicament 18.

Prenons par la suite un cas de polypharmacie erroné. C'est-à-dire un cas où notre algorithme a déterminé avec un degré de confiance suffisant qu'une combinaison de médicaments particulière devrait mener à une hospitalisation, mais cela n'est pas le cas. Il existe beaucoup de cas du genre dans notre jeu de données et certaines raisons peuvent expliquer cela comme mentionner dans le rapport 1. Un exemple évident est simplement une personne ayant une santé plus résistante que la moyenne. Prenons par exemple un de ces cas, la ligne 39466 dans le jeu de données est un cas où la combinaison des médicaments en représentation binaire 0010010010101100111 est une des combinaisons qui devrait mener à une hospitalisation, mais ce n'est pas le cas.

Dans les cas de succès, c'est-à-dire un cas de polypharmacie correctement déterminé, il y a plusieurs exemples contenus dans notre jeu de données. Par exemple la ligne 22258 contient une polypharmacie véridique de la combinaison des médicaments 1000110000011001111 qui a mené à une hospitalisation. Ce patient est probablement une personne ayant une santé dans la moyenne et donc est bien représenté dans notre modèle.

4 Version finale du système

La version finale de notre projet aura été une implémentation de XGBoost. Afin de faciliter l'entraînement sur notre jeu de données modifiées, nous avons donc opté pour limiter nos tests sur un jeu de données beaucoup plus petit de 100 000 entrées. Cela nous permettait d'effectuer des entraînements plus rapides et d'accélérer la recherche des paramètres du modèle (particulièrement pour le taux d'apprentissage, la valeur optimale étant de 0,1). De plus, lors de chaque entraînement, les entrées dans le jeu de données étaient mélangées selon dix "random seeds" (agencement aléatoire prédéfini) différents, afin de s'assurer que les performances des différents modèles soient constantes. Seule la version finale du modèle a été entraînée sur le jeu de données total, puisque l'entraînement

sur une telle quantité de données était long. Néanmoins, l'utilisation de l'entièreté des entrées a toutefois pu avoir comme résultat d'aplanir la courbe ROC ("Smooth the ROC curve"), ce qui rend la courbe plus régulière et moins saccadée ce qui facilite la lecture de nos données. Il est à noter que considérant le nombre de combinaisons de médicaments (2 ^ 19) versus le nombre de lignes du jeu de données (environ 30 millions), le fléau de la dimensionnalité ne nous impacte peu et nous pouvons avoir une bonne confiance dans notre algorithme.

Une des intuitions que nous avons eue au cours de la session et qui a eu des résultats très prometteurs est de mélanger les entrées dans le jeu de données. Même si cela a pu nous sembler contre-intuitif initialement, le fait de mélanger les entrées nous a permis d'avoir de meilleurs résultats puisque cela enlevait le biais que pouvait avoir notre modèle si le même id de patient était répété plusieurs lignes de suite.

Une intuition que nous avons eue et qui a malheureusement seulement eu une légère amélioration consistait à enlever la première ligne de chaque patient et d'ajouter l'hospitalisation précédente. La logique derrière cela était qu'un patient arrivant pour la première fois à l'hôpital ne serait pas là pour une polypharmacie.

Comme mentionné plus haut, nous avons aussi fait des expérimentations sur les effets des médicaments par les effets par jour ainsi que l'effet des médicaments par le délai des effets par jours et dans les deux cas les résultats n'ont pas été concluants. L'hypothèse que nous avons est que l'ajout d'attributs ou la complexification d'attributs n'engendre pas nécessairement d'information pertinente supplémentaire pour le modèle. On considère même que cela plus à titre de bruit dans notre jeu de données.

En soi notre algorithme n'a eu besoin que de très peu de prétraitement, les données dans leur état initial semblaient nous donner les meilleurs résultats sans devoir les modifier.

5. Rétrospective du projet

Nous avons validé notre hypothèse de l'utilité de XGBoost. En effet, dans le premier rapport, nous observons déjà l'utilité d'un arbre de décision et notre choix de librairie qui est au final une collection d'arbres de décision répond amplement à notre besoin.

Un point surprenant à prendre en compte est l'historique du patient. Il nous semblait a priori que d'avoir toutes ces prescriptions en ordre allait aider l'algorithme, mais au contraire, les résultats les plus prometteurs ont été obtenus en mélangeant les entrées dans le jeu de données comme mentionnées plus haut.

De la même façon, l'approche du « less is more » a été plus prometteuse que ce que l'on aurait pu penser. En effet, plus nous essayions d'ajouter ou de modifier nos attributs, plus le ROC score diminuait.

Si le projet était à refaire, deux éléments nous paraissent intéressants à modifier. Le premier aurait consisté à trouver d'autres manières de limiter le bruit lors de l'étape de prétraitement dans notre jeu de données. La deuxième aurait été de tester avec plus d'algorithmes différents pour comparer chacun des résultats de ceux-ci. L'utilisation de l'hyperparamétrage pour trouver de nouveaux attributs aurait aussi été une excellente façon d'optimiser notre algorithme. Si nous avions eu plus de

temps, l'utilisation d'un réseau de neurones profond aurait aussi pu être un algorithme qui nous aurait donné des résultats prometteurs considérant la performance des réseaux à gérer les données binaires. Toutefois, il faut considérer qu'il est très difficile de déterminer comment un réseau obtient son résultat ce qui s'applique très mal à un contexte médical [4].

[1]: XGBoost. 2022. URL: https://xgboost.readthedocs.io/en/stable/#

[2]: Kaggle. 2022. URL: https://www.kaggle.com/code/cdeotte/xgboost-starter-0-793

[3]: National library of medecine. 2020. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7931945/

[4]: IBM. 2021. URL: https://www.ibm.com/topics/neural-networks

6. Annexe

6.1 Exemple d’arbre de décision

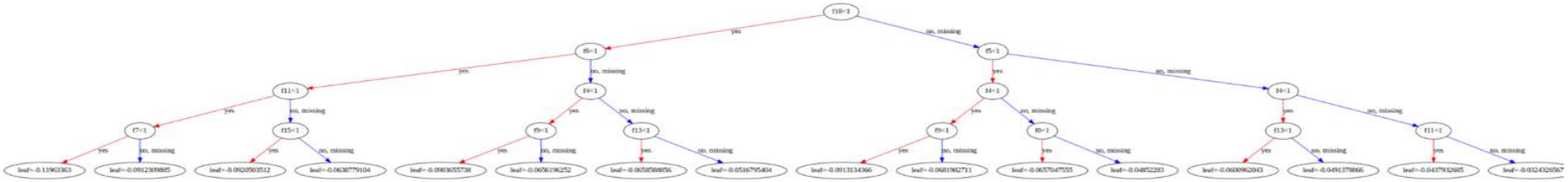

Figure 6.1.1 Exemple d’arbre de décision

6.2 Importance des attributs

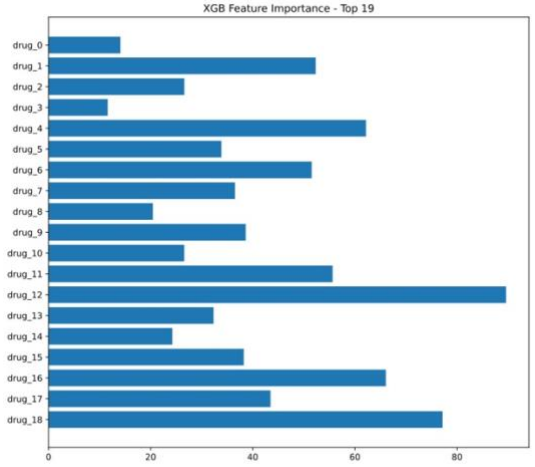

Figure 6.1.2 Importance des attributs