Rapport 2

Ekoué Mawuéna Octave Adama†, Auguste Aka Tiemele†, Deoth Guei †
Département d'informatique et de génie logiciel, Université Laval

Résumé

A la suite de notre premier rapport qui traitait l'analyse et le pré-traitement de données de polypharmacies dans le but de déterminer les combinaisons de médicaments qui peuvent conduire à une hospitalisation , nous nous concentrons cette fois sur le traitement de données ainsi que l'implémentation des algorithmes pour la détections de nos combinaisons de médicaments nocives. Plus précisément, dans cette partie du projet nous utiliserons les données pré-traitées , nos premières intuitions sur les attributs importants pour l'entraînement de nos modèles , un traitement de données pour l'application de nos algorithmes, l'évaluation des modèles implementés , les difficultés rencontrées ainsi qu'une rétrospective sur le projet. Les algorithmes adaptés pour la réalisation de notre système sont les réseaux de neurones, les règles d'association basées sur la détection de patrons fréquents et les forets aléatoires.

Keywords : Traitement de données, polypharmacie , réseaux de neurones ,forets aléatoires ,règles d'associations ,

1. Introduction

La polypharmacie est une pratique courante dans le traitement de diverses maladies. Cependant, la combinaison inappropriée de certains médicaments peut entraîner des conséquences graves, telles que l'hospitalisation des patients. Dans ce contexte, notre projet s'est concentré sur l'analyse des données de polypharmacie et la détection des combinaisons de médicaments nocives à l'aide d'algorithmes tels que les réseaux de neurones, les règles d'associations basées sur la détection de patrons fréquents et les forêts aléatoires. Dans ce rapport, nous nous concentrons plus particulièrement sur le traitement des données, la mise en œuvre de ces algorithmes, l'évaluation des modèles, les difficultés rencontrées et une rétrospective sur l'ensemble du projet.

2. Alogorithme Random Forest

2.1. Retour sur le pré-traitement et améliorations

De prime abord nous avons entraîner un classifieur Random Forest avec notre jeu de données pré-traitées initiales, en excluant les attributs "timestamp" et "patient_id". Nous avons obtenu des résultats catastrophiques en test. **Precision :**0.373626 ; **Error :**0.359 ; **F1-score :** 0.159251 **Recall :**0.10119 ; **Accuracy :**0.641 ; **ROC_AU score :**0.507674

Après analyse de ces résultats, il est clair que notre modèle ne fonctionne pas sur notre jeu de données, ce qui nous a conduit à remettre en question la représentativité des données. Pour résoudre ce problème, nous avons changé la représentation en considérant le nombre de fois que chaque médicament a été prescrit à un patient pour déterminer si ce patient a été au moins une fois hospitalisé ou non (0, 1). Cela signifie que nous avons regroupé nos observations en sommant celles qui ont un identifiant de patient identique.

Département d'informatique et de génie logiciel, Université Laval *

Cette nouvelle représentation nous a permis de mieux discriminer notre jeu de données. Cependant, cela introduit un biais lié à la fréquence d'utilisation des médicaments. En somme nous avons transformé nos données en un jeu de données ne contenant que le nombre de patients sous polypharmacie (soit 1,4 million), parmi lesquels 77,3% ont été hospitalisés au moins une fois et 22,7% n'ont jamais été hospitalisés, comme l'illustre la figure 1.

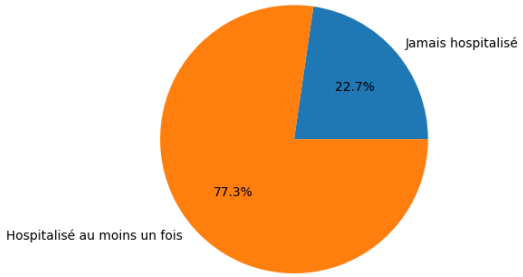

FIGURE 1 – Nombre total de patients en polypharmacie

2.2. **Présentation de l'algorithme Random Forest adapté à notre problème.**

Pour trouver les combinaisons nocives, nous avons utilisé l'algorithme d'arbres de décisions Random Forest dans notre processus de décision. Cet algorithme est robuste au bruit et résout le problème de sur-ajustement des arbres de décision. Cependant, il nécessite dans certains cas une grande quantité de mémoire pour fonctionner correctement.
Pour retrouver nos combinaisons nocives, nous avons mis en place un système d'analyse et de décision avec un classifieur Random Forest. Le système est illustré à la figure 2.

Pour notre algorithme de Random Forest utilisé, nous avons fixé le nombre d'arbres à 100, la fonction de qualité 'gini', le nombre d'attributs à la racine carrée du nombre d'attributs dans notre jeu de données, et nous avons équilibré les classes lorsque le jeu de données était déséquilibré.

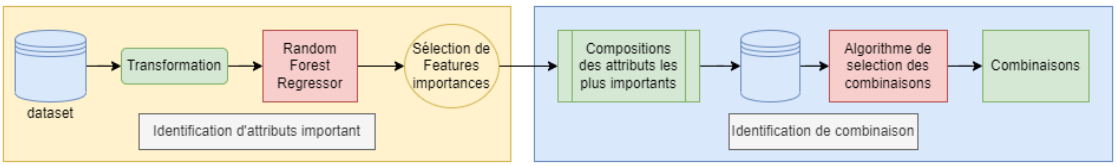

FIGURE 2 – Processus d'analyse et de décisions pour extraire les combinaisons nocives.

2.3. **Elaboration du systeme et detection de medicament important**

L'élaboration de ce système a été faite comme suit. Tout d'abord nous avons remarqué [figure 1] qu'il y a 22.7% des 1.4 millions de patients qui ont eu des prescriptions de certains médicaments mais n'ont jamais été hospitalisés. Cette donnée nous permet de penser qu'il y a des prescriptions de médicaments qui ont une influence faible sur le fait d'être hospitalisé

ou non. Cependant, cela ne signifie pas que nous n'avons pas de biais lié à la fréquence de prescription de médicaments chez les patients, comme la prescription de médicaments populaires.

Nous avons échantillonné notre nouveau jeu de données de façon aléatoire pour l'entraînement et le test, puis avons entraîné un Random Forest sur les données d'entraînement. Nous avons obtenu de bon résultat sur notre jeu de test. Voici ces resultat, **Precision :**0.928955 ; **Error :**0.234009 ; **F1-score :**0.83303 **Recall :**0.755061 ; **Accuracy :**0.765991 ; **ROC_AU score :**0.779148 et la matrice de confusion, a la figure 3.

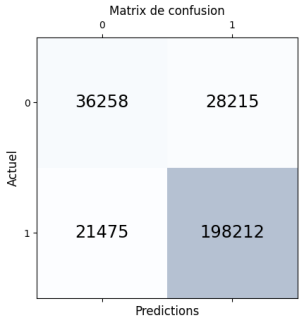

Figure 3 – Matrix de confusion Random forest sur jeu donnée transformée

Nous avons obtenu de bon résultats, mais il convient de prendre ce résultat avec des pincettes. En effet, ce résultat ne fonctionne qu'avec une représentation volontairement biaisée de notre jeu de données initial, qui identifiait de façon indépendante nos observations. Contrairement à cette représentation, notre nouvelle représentation identifie nos observations de façon globale sur l'historique d'un patient, c'est-à-dire qu'il faut disposer de toutes les prescriptions d'un patient depuis la création de notre jeu de données pour obtenir une bonne prédiction. Néanmoins, cette représentation nous permet d'identifier les attributs (dans notre cas, les médicaments) qui participent fortement à l'issue d'une hospitalisation chez un patient en polypharmacie. Pour identifier ces attributs, nous avons utilisé la formule de diminution d'impureté suivante :

$$\Delta I = \frac{N_t}{N * (impurity - \frac{N_{tr}}{N_t} * right_impurity - \frac{N_{tl}}{N_t} * left_impurity)}$$

, où N est le nombre total d'observations, N_t est le nombre d'observations sur un nœud, N_{tr} est le nombre d'observations sur le nœud enfant de droite et N_{tl} est le nombre d'observations sur le nœud enfant de gauche. Cette formule peut être résumée comme l'importance qu'un attribut a de sépare nos données dans les différents noeuds d'arbres de décisions. Le résultat d'importance est présenté sur la figure 4.

Pour la sélection de nos attributs, nous avons retenu ceux qui sont au-dessus de la médiane, à savoir les 'Drug_1' 'Drug_4' 'Drug_5' 'Drug_7' 'Drug_11' 'Drug_12' 'Drug_16' et 'Drug_17'. Comme mentionné précédemment, notre mesure d'impureté utilise le nombre d'observations qu'un attribut parvient à discriminer. Par conséquent, notre mesure d'impureté est très sensible aux attributs qui ont un grand nombre d'observation, même si la discrétisation est médiocre. Pour remédier à cette faiblesse, nous avons utilisé un algorithme de sélection d'attributs basé sur permutation d'attribut important. Cet algorithme consiste

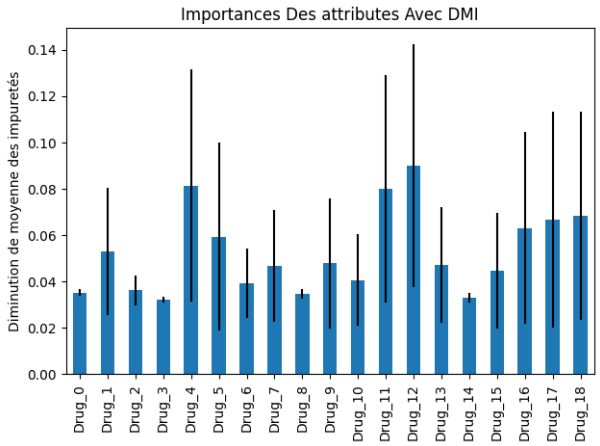

FIGURE 4 – Feature importance avec diminution de la moyenne d'impureté

à utiliser notre modèle de Random Forest entraîné précédemment, puis à échantillonner de manière aléatoire nos attributs et nos données du jeu d'entraînement. Nous évaluons ensuite ces résultats avec un processus de validation croisée et vérifions si le score de précision diminue en supprimant et permutant chaque attribut. Nous identifions ensuite les attributs qui diminuent le moins le score de précision. Après l'exécution de cet algorithme, nous obtenons les résultats suivants figure x.

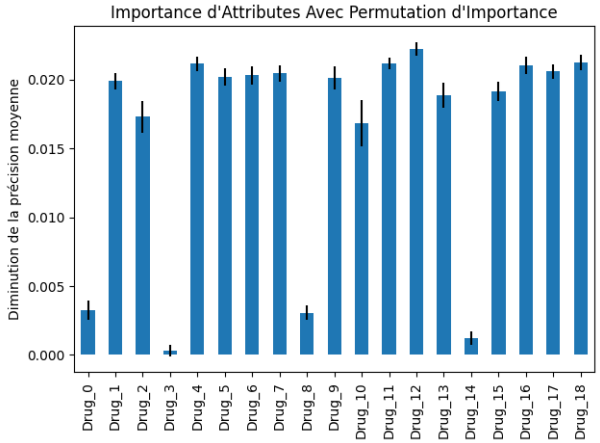

FIGURE 5 – Permutation de l'importance des caractéristiques avec une diminution de la précision moyenne.

Comme pour le premier essai, nous avons gardé les attributs avec un score au-dessus de la médiane. Nous avons ainsi identifié les attributs qui participent le plus à la séparation de nos données transformées ('Drug_1', 'Drug_4', 'Drug_5', 'Drug_6' 'Drug_7', 'Drug_9', 'Drug_10', 'Drug_11', 'Drug_12', 'Drug_13', 'Drug_15', 'Drug_16', 'Drug_17' et 'Drug_18'). Cependant, il est prudent de vérifier si en ne gardant que ces attributs, nous sommes toujours en mesure de classifier nos données. Nous avons donc entraîné à nouveau un classifieur Random Forest sur notre jeu d'entraînement en utilisant uniquement ces attributs, et nous avons obtenu un résultat toujours positif. Voici ces résultats, **Precision :**0.888155 ;

Error :0.181971 ; **F1-score :**0.88142 **Recall :**0.874785 ; **Accuracy :**0.818029 ; **ROC_AU score :**0.7497.

2.3.1. **Identification de nos combinaisons nocives**

À partir de ces résultats précèdent, nous pouvons analyser notre jeu de données transformé en regardant les prescriptions chez les patients n'ayant jamais été hospitalisés par rapport à ceux ayant été hospitalisés au moins une fois, comme présenté dans la figure 6.

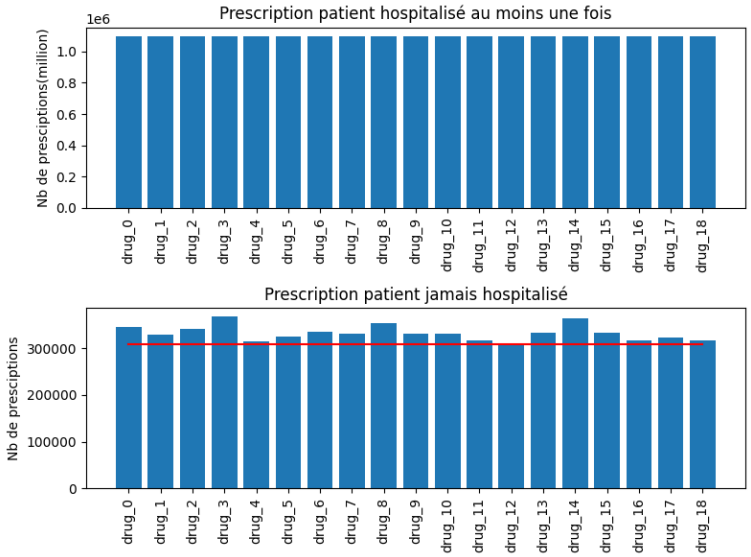

Figure 6

Nous remarquons, avec le graphe à la figure 6, et avec les 14 attributs importants détectés, que le nombre d'utilisations des médicaments chez les patients qui ont été hospitalisés au moins une fois est identique, tandis que le nombre d'apparitions des médicaments chez les patients qui n'ont jamais été hospitalisés est distribué différemment. Nous constatons également que les médicaments les moins utilisés chez les clients qui n'ont jamais été hospitalisés sont ceux qui ont été identifiés par notre algorithme de permutation d'importance de caractéristiques..

Cependant, cela ne nous dit rien sur le fait que l'utilisation de ces 14 médicaments sont nocives. Mais Nous pouvons nous demander si ces médicaments sont prescrits en faible quantité à des patients à faible risque d'hospitalisation et en polypharmacie. Dans ce cas, combiner un ou plusieurs de ces médicaments peut être nocif. Nous avons donc examiner toutes les observations contenant la prescription de ces médicaments pour voir si le taux de personnes hospitalisées est élevé, puis identifier nos combinaisons nocives.

Dans la deuxième partie de notre processus, nous avons cherché nos combinaisons néfastes parmi nos attributs sélectionnés. Étant donné la pertinence des 14 attributs après notre analyse, nous allons tenter d'identifier les combinaisons néfastes en mettant en place un algorithme permettant de tester toutes les combinaisons possibles pour chaque catégorie de

polypharmacie. Notre algorithme suit les étapes suivantes :

— Créer un ensemble de combinaisons de n médicaments parmi les 14 médicaments identifiés.

— Pour chaque combinaison, récupérer dans notre jeu de données originales (sans modification), les observations où tous les médicaments dans notre combinaison sont à 1 et les autres à 0, puis ensuite récupérer le champ hospit et calculer un score de précision.

— Ensuite, garder la combinaison avec le meilleur score de précision

Nous avons choisi la précision comme score car, dans ce problème, nous avons que des observations d'une combinaison dans le jeu de données et l'hypothèse que toutes ces observations sont nocives, c'est-à-dire avec hospit à 1. Pour plus de clarté voici le pseudocode de l'algorithme en détail :

1- initialiser best_current_combi à None
2- initialiser best_precision_score à -inf
3- pour chaque combinaison de n nombres dans medicaments_important, faire :
  3.2- initialiser diff à medicaments - combinaison
  3.5- initialiser query_where à compose_query_where_with(combinaison, diff)
  3.6- exécuter la requête SQL "Select hospit From dataset query_where"
  3.7- initialiser la liste y à partir des résultats de la requête SQL
  3.8- initialiser y_true avec les éléments de y convertis en entiers
  3.9- initialiser y_pred avec des tableaux de 1 ayant la même forme que y_true
  3.10- initialiser precision_socre à la valeur de la précision entre y_true et y_pred
  3.11- si precision_score est supérieur ou égal à best_precision_score, faire :
    3.11.1- mettre à jour best_score avec la valeur de score
    3.11.2- mettre à jour best_current_combi avec la valeur de combi
4- ajouter {'combinaison' : best_current_combi, 'score' : best_score} à best_combinaisons

Nous avons exécuté l'algorithme ci-dessus pour trouver le nombre de n=(5, 6, 7, 8, 9, 10, 11, 12, 13, 14) médicaments parmis nos 14 médicaments.

3. **Découvertes de règles d'associations**

Pour mettre en place cet algorithme, il y a des hyperparamètres de base à spécifier, tels que le support et la confiance. L'importance du support vient du fait qu'on est censé avoir suffisamment d'instances d'une combinaison avant d'être plus ou moins sûr de son statut nocif ou non. L'importance de la confiance vient du fait que l'on doit savoir combien de fois on peut être sûr que le statut de la combinaison est nocif ou pas, à partir de toutes les instances de la combinaison qui ont été observées.

Étant donné que nous avons un grand ensemble de données avec seulement deux étiquettes possibles et un déséquilibre en faveur des 0, la recherche exhaustive de motifs fréquents sur

ce grand jeu de données s'annonce très difficile.

Nous avons donc décidé de diviser les données en deux en fonction de leurs étiquettes, c'est-à-dire les instances avec des hospitalisations à 0 d'un côté et celles avec des hospitalisations à 1 de l'autre.

Ensuite, étant donné que l'étiquette la plus importante pour nous ici est l'étiquette 1, nous allons nous concentrer d'abord sur ce groupe d'instances.

Malgré la division des données en deux, on se rend compte que le nombre de données demeure toujours conséquent pour les deux groupes.

La technique mise en place pour quand même pouvoir trouver les motifs fréquents est de diviser nos données en instances avec l'étiquette 1 par lots de 10 000, puis d'essayer de découvrir les motifs fréquents de tous les lots avant de vérifier si ces motifs sont également fréquents dans au moins la moitié de tous les lots.

Les motifs fréquents des données des instances avec l'étiquette 1 trouvés seront ensuite utilisés pour trouver la fréquence de leur contrepartie dans les données des instances avec l'étiquette 0. On pourra ainsi conclure, par comparaison, si notre combinaison est nocive ou pas (ce dernier processus peut être assimilé à la confiance).

3.1. **Les défis de l'implémentation sont les suivants :**

La recherche exhaustive dans un grand jeu de données. En effet, même en procédant par lots, on ne peut pas choisir des combinaisons fréquentes dans un lot et conclure sur le statut de la combinaison, car un motif fréquent dans un lot peut être finalement non fréquent dans la base de données initiale.

Le choix du support : Ce choix est compliqué, non seulement dans le sens où l'on ne sait pas à quelle fréquence exacte une combinaison est fréquente, mais aussi la limite entre deux supports très proches, par exemple 0,03 et 0,04, donne de nombreuses combinaisons par lot et aucune combinaison par lot, respectivement.

Le choix de la confiance : Comment savoir que nous sommes sûrs d'avoir suffisamment d'hospitalisations à 1 pour conclure qu'une combinaison est nocive, sachant que théoriquement, le nombre d'instances de la combinaison avec les hospitalisations égales à 1 risque d'être plus petit que celui avec les hospitalisations égales à 0 en raison du déséquilibre des données.

Finalement, les choix du support et de la confiance peuvent différer en fonction du nombre de médicaments dans une combinaison, car plus le nombre de médicaments est grand, plus il est difficile de trouver la combinaison dans les données.

4. **Techniques d'apprentissage par réseaux de neurones**

4.1. **Préparation des données**

Les données que nous allons utiliser dans cette partie sont issue du pré-traitement décrit dans le premier rapport, puis retransformer de sorte à avoir une ligne par patient. Pour se faire, nous sommons simplement les lignes regroupé par patients. Après cela, nous mettons à 1 tous les nombres supérieurs à 0. Cela veut dire que les données nous disent par ligne tous les médicaments qui ont une fois été prescrit au patient et si ce patient a ne serait-ce qu'une fois été hospitalisé, son hospit sera à 1. Nous avons maintenant 1420809 lignes. Cette

transformation occasionnera certainement la découverte de fausses combinaisons nocives, mais elle permettra quand même de ne rater aucune combinaison nocive. Nous décrirons plus bas comment éliminer les fausses combinaisons nocives.

4.2. **Description de l'algorithme**

L'idée est d'entraîner un réseau de neurone de classification plus ou moins précis, grâce auquel, nous allons avoir les ''hospits=1" les mieux classés, c'est-à-dire les hospits à 1 que le réseau arrive à prédire avec une certitude élevée. Pour ce faire nous allons définir un seul de probabilité selon la performance de notre algorithme, et sélectionné les combinaisons classées comme hospit avec une certitude supérieure au seuil. De ces données bien classées comme hospit à 1 avec une grande certitude par notre modèle, nous allons avoir plusieurs combinaisons probablement nocives. Pour épurer cette liste, nous allons dans un premier temps éliminer les doublons puis supprimer les combinaisons nocives composées de d'autres combinaisons nocives plus petites. Nous partons avec l'hypothèse selon laquelle si une combinaison de A, B, C, D, E est nocive, toute autre combinaison contenant A, B, C, D, E et d'autre médicaments sera également nocive, d'où nous supprimons les combinaison nocive contenant d'autres combinaisons nocives, car elles ne sont pas informatives. Après cette étape, par le biais de ''importance de permutation"(que nous allons détailler plus loin), nous éliminerons les combinaisons contenant des médicaments avec une faible importance. De cette manière, nous avons de bonne chance d'éliminer les faux positives sans perdre de l'information importante.

4.3. **Architecture du réseau**

L'architecture du réseau de neurones utilisé dans notre modèle est une architecture de type séquentielle de TensorFlow Keras. Elle est constituée de 4 couches de neurones pleinement connectées qui sont empilées les unes sur les autres. La première couche est une couche d'aplatissement (Flatten) qui prend en entrée un tenseur de dimension (19, 1) correspondant à un vecteur de 19 caractéristiques (ou features) de notre jeu de données. Cette couche transforme le tenseur d'entrée en un vecteur de dimension 19. Les 3 couches suivantes sont des couches de neurones Dense avec une fonction d'activation sigmoïde. Les deux premières couches ont chacune 256 neurones, tandis que la troisième couche est constituée de 2 neurones pour la classification binaire. La dernière couche est une couche Dense à 1 neurone avec une fonction d'activation sigmoïde qui produit la sortie finale, c'est-à-dire la prédiction de la classe de sortie.

Il est possible de rencontrer un problème de vanishing gradient avec la fonction d'activation sigmoïde, surtout dans les réseaux de neurones profonds. Cependant, pour l'architecture de réseau de neurones que nous avons décrite, le problème de vanishing gradient est peu probable. La raison en est que notre réseau de neurones n'est pas très profond (il n'a que quatre couches), et que la fonction d'activation sigmoïde est souvent plus stable dans les réseaux peu profonds. De plus, la taille relativement petite de chaque couche (256, 256, 2 et 1 neurones) est également susceptible de réduire le risque de vanishing gradient.

4.4. **Entraînement, tests et ajustement des paramètres**

La fonction de perte utilisée pour entraîner le modèle est la binary cross-entropy (ou entropie croisée binaire) et l'optimiseur utilisé est Adam. Le modèle est entraîné avec un batch_size de 512 et un nombre d'époques de 100. De plus, la méthode de rappel (callback) LearningRateScheduler est utilisée pour ajuster le taux d'apprentissage (learning rate) de l'optimiseur Adam au cours de l'entraînement, afin d'améliorer la convergence du modèle.

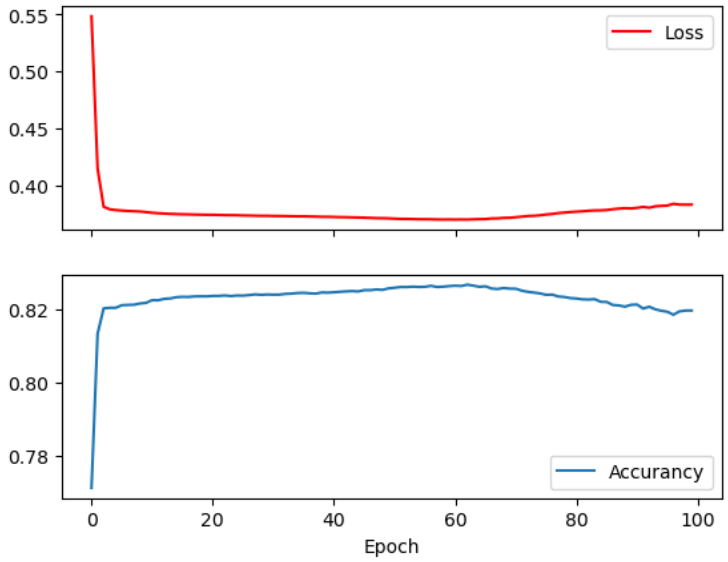

Figure 7 – Précision et perte en fonction du nombre d'époque

On constate sur la figure 7 qu'a partir de l'époque 30, le modèle n'apprend plus vraiment. La précision se stabilise, ainsi que la perte. A partir de 60, la précision décline et la perte commence a augmenter. Nous allons donc choisir comme nombre d'époque 35 pour éviter le sur-apprentissage et refaire l'entraînement. 70% des données servent a l'entraînement et 30% sert au test. Après l'entraînement et le test, les performances de notre modèle(appelons la version 2.0) ce présentent comme suit :

Exactitude : 0.82
Précision : 0.85
Rappel : 0.92

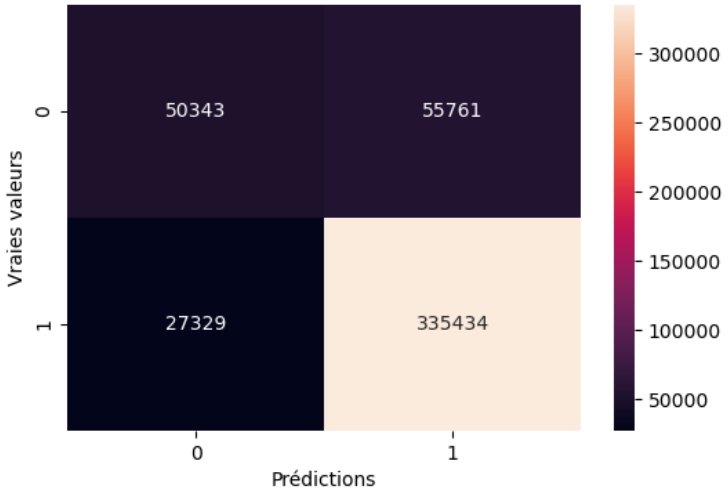

Figure 8 – Matrice de confusion

Rappelons que nos données sont dé-balancées (0.77 de la classe 1 et 0.23 de la classe 0). Pour amélioré les performance de notre modèle, nous allons tenter de regler ce problème et

de faire une autre version de notre modèle. Pour régler le problème de dé-balancement de nos classes , nous avons le choix entre le sous-échantillonnage (under-sampling) et le sur-échantillonage (over-sampling). Le sous-échantillonnage consiste à supprimer aléatoirement certaines observations de la classe majoritaire pour atteindre un équilibre entre les classes. En supprimant, nous pourrions perdre de l'information importante, ce que nous ne voulons pas. Nous avons donc choisi le sur-échantillonage qui duplique aléatoirement certaines des observations de la classe minoritaire jusqu'à atteindre un équilibre entre les classes.

Après l'entraînement et le test, les performance de notre nouveau modèle se présente comme suit :

Exactitude : 0.73
Précision : 0.8
Rappel : 0.77

Comme on peut le voir, ce modèle performe moins que le précédent. Nous pensions que résoudre le dé-balancement via l'over-sampling serait profitable mais visiblement, ce n'est pas le cas. Cela peut s'expliquer par le fait que le sur-échantillonnage augmente le bruit dans nos données. Étant donné que le sous-échantillonnage n'est pas une option du fait qu'il pourrait nous faire perdre de l'information, nous allons conserver notre version précédente de notre modèle. Dans la suite du travail, le modèle considéré sera le modèle précédente (la version 2.0)

4.5. **Recherche des combinaison nocives**

Nous allons maintenant nous appliquer à détecter les combinaisons nocives. Pour se faire, nous allons dans un premier temps classifier l'ensemble du jeux de données selon notre modèle, puis sélectionner les prescription de hospit = 1 dont le modèle est le plus sûr possible. Compte tenu de la distribution de la probabilité des prédictions, nous avons choisi le seuil de certitude de 0.99. Une fois ces prescriptions sélectionnées, nous éliminons les doublons puis nous supprimons les combinaisons contenant d'autres combinaisons.

Nous allons maintenant utiliser "Permutation importance" pour déterminer l'importance de chaque feature dans notre modèle de machine learning. Le processus consiste à entraîner un modèle sur les données d'entraînement, puis à évaluer les performances du modèle lorsqu'on retire chaque feature un par un. Lorsqu'un feature est retiré, le modèle est entraîné de nouveau sur les données restantes et les performances sont évaluées sur les données de test. La différence entre les performances du modèle original et celles du modèle avec le feature retiré est alors calculée. Ce processus est répété pour chaque feature, permettant ainsi de mesurer l'importance relative de chaque feature dans le modèle. Les neufs premiers features les plus importants retenus par cette méthode sont : **drug_1, drug_4, drug_5, drug_7, drug_11, drug_12, drug_16, drug_17, drug_18** Les combinaisons constitué de ces features sont nos combinaisons nocives. Nous allons donc sélectionner toute les combinaison où au moins 5 de ces médicaments sont présents.

5. **Choix du système final**

Le potentiel système base sur la découvertes de règles d'association nécessite trop de considération a prendre en compte et le nombre de combinaison des médicaments dans chaque lot est presque également reparti.De ce fait il devient complique computationellement de vérifier la fréquence d'une combinaison dans chaque lot afin de fournir une exploitabilité correcte.De ce fait ce fait ,ce système ne pourra être utilise comme système final.

Quant aux réseaux de neurones , il génère beaucoup trop de combinaisons et en crée qui n'appartiennent pas vraiment a notre jeu de données. On obtient cependant des scores plus ou moins satisfaisants mais l'interpretabilité risque d'être difficile.

Pour des raisons de meilleurs explicabilités et résultats ,notre système basé sur le random forest est choisi comme le système final. Les résultats de ce système seront donc les résultats qui seront importants pour la suite de notre travail.

Après la mise en place de notre système avec Random Forest, nous avons obtenu les résultats suivants dans le tableau 1, qui est un récapitulatif des combinaisons trouvées. Le tableau inclut le score de précision obtenu lors de la recherche de notre combinaison, le nombre de patients qui ont été hospitalisés plusieurs fois par rapport à ceux qui n'ont jamais été hospitalisés avec cette combinaison, ainsi que le nombre de fois où la combinaison a été prescrite. Lorsque nous examinons les résultats du tableau 1, nous pouvons constater que

#	**Combinaison**	**Précision**	**# hospitalisé/jamais hospitalisé**	**# prescriptions**
1	0 1 0 0 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1	80%	2/4	10
2	0 1 0 0 1 1 1 1 0 1 1 1 1 0 0 1 1 1 0	85.7%	5/0	7
3	0 1 0 0 1 1 1 0 0 0 1 1 1 1 0 1 1 1 0	100%	3/0	3
4	0 0 0 0 1 1 1 0 0 0 1 1 1 1 0 1 1 1 0	100%	2/0	2
5	0 1 0 0 1 0 1 1 0 1 1 1 0 0 0 0 1 1 0	100%	1/0	1
6	0 1 0 0 1 1 0 0 0 0 1 1 0 0 0 1 1 1 0	100%	3/0	3
7	0 1 0 0 1 1 0 0 0 0 0 1 0 1 0 0 1 1 0	87.5%	4/0	8
8	0 1 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0	87.5%	38/8	70
9	0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0	78%	101/23	200

TABLE 1 – Tableau récapitulatif des combinaisons de médicaments avec leur précision, le nombre de patients et le nombre de prescriptions correspondant

les combinaisons les plus importantes se situent entre les lignes 5 et 9. En effet, en observant les autres résultats, nous avons l'impression d'un effet d'accumulation de médicaments dans une combinaison déjà nocive. Pour simplifier l'analyse, nous allons donc nous concentrer sur les combinaisons de la ligne 5 à la ligne 9.

Par exemple, pour le patient_id :283694, nous avons remarqué que sur 12 prescriptions, la moitié d'entre elles incluaient la combinaison de la ligne 9, et le patient a été hospitalisé à chaque fois. Cependant, pour l'autre moitié des prescriptions qui n'incluaient pas cette combinaison nocive, le patient n'a pas été hospitalisé. Cette différence peut s'expliquer par des historiques de médicaments différents entre les utilisateurs.

6. **Retropesctive sur le projet.**

À l'origine, nous pensions que l'exploration et le prétraitement de nos données seraient simples, comme souvent présenté dans les tutoriels en ligne. Cependant, nous avons rapidement compris la complexité associée à la manipulation d'un ensemble de données réelles. La dimensionnalité importante de nos données ainsi que la binarité de nos attributs ont ajouté une complexité supplémentaire. Malheureusement, cela nous a conduit à nous concentrer davantage sur les prescriptions plutôt que sur les clients, ce qui a eu un impact négatif sur les résultats. Nous avons réalisé que le fait de nous concentrer sur les patients était la meilleure approche, car cela a permis d'obtenir des traitements plus efficaces et de meilleurs résultats. En ce qui concerne l'implémentation des algorithmes, nous pensions que le traitement de nos données serait suffisant pour obtenir des scores élevés dès le premier essai. Cependant, nous avons rapidement compris que cela ne serait pas aussi simple. Les résultats de nos premiers tests ont été décevants et nous avons dû effectuer plusieurs itérations avant d'obtenir des résultats satisfaisants.

Une alternative supplémentaire pour le traitement de nos données pourrait être de réaliser un groupement, qui permettrait de regrouper les clients en plusieurs groupes selon le nombre de fois où ils ont été hospitalisés. Nous pourrions ainsi nous concentrer sur les clients

les plus fréquemment hospitalisés et rechercher dans leur espace les combinaisons nocives. Cette réduction de l'espace de recherche pourrait nous permettre de mettre en place des algorithmes de recherche plus exhaustifs, mais également plus fiables.

7. **Conclusion**

En conclusion, ce projet de détection de polypharmacies nocives a été une aventure riche en enseignements. Nous avons exploré plusieurs approches, y compris les règles d'associations basées sur les patrons fréquents, les réseaux de neurones et les arbres de décision. Bien que nous ayons rencontré des difficultés avec certaines de ces approches, nous avons finalement réussi à identifier 9 combinaisons de médicaments qui peuvent conduire à une hospitalisation.

Cependant, nous avons également réalisé qu'il reste encore beaucoup à faire dans ce domaine de recherche, en particulier en ce qui concerne la qualité des données et la difficulté de l'interprétation des modèles de réseaux de neurones. Nous avons également appris que l'exploration de données est une étape cruciale dans la création d'un modèle prédictif efficace.

Dans l'ensemble, ce projet nous a donné une occasion précieuse d'apprendre et de grandir en tant que scientifiques de données. Nous espérons que nos résultats et nos expériences pourront aider à orienter les futures recherches sur la détection de polypharmacies nocives.