Techniques d'apprentissage
IFT 712

PROJET DE SESSION : RAPPORT

Félix Dumais (14053686)
Nicolas Fontaine (15203272)
Joëlle Fréchette-Viens (15057894)

Présenté à
Pierre-Marc Jodoin

Faculté des Sciences
Département d'informatique

12 avril 2020

UNIVERSITÉ DE
SHERBROOKE

Table des matières

Table des figures

Liste des tableaux

1 Introduction

Depuis les dernières années, le monde de l'intelligence artificielle est en pleine évolution. De nombreux articles sont publiés annuellement dans différents journaux scientifiques. Ces articles font état des différentes découvertes et avancées dans ce domaine. Récemment, un article publié dans la revue Nature par des chercheurs de Google montrait que l'intelligence artificielle DeepMind venait de surpasser des radiologistes professionnels quant à leur capacité à diagnostiquer les cancers du sein[1]. L'intelligence artificielle, et plus particulièrement, l'apprentissage profond, a démontré des capacités très surprenantes en matière d'analyse d'images médicales. Ces capacités vont du diagnostic d'un patient à partir de radiographie à la segmentation de différents organes dans le corps humain avec des images provenant d'acquisition tomodensitométries. Ainsi, pour le projet du cours IFT712, il a été décidé d'appliquer les techniques d'apprentissage à l'analyse d'images médicales. Une base de données de plus de 5500 radiographies de poumons a été sélectionnée sur Kaggle. Le but du projet est de déterminer les pathologies des patients à partir de ces images et de différents algorithmes utilisant les techniques d'apprentissage. Pour ce faire, 6 différents algorithmes ont été comparés pour déterminer lesquels obtenaient les meilleurs résultats. Les méthodes utilisées pour classifier les images sont les machines à vecteurs de support, les réseaux de neurones multicouches, les méthodes à noyau, les forêts d'arbres décisionnels, les discriminants de Fischer et la régression appliquée à la classification. Ce rapport décrit la démarche utilisée pour ce faire, les résultats obtenus ainsi que la méthode optimale proposée selon les ressources disponibles.

2 Démarche scientifique

2.1 La base de données

La base de données utilisée pour le projet provient du site Kaggle, un site hébergeant une énorme quantité de bases de données spécialement conçues pour entraîner des algorithmes d'apprentissage automatique. Le nom de la base de données est *Random Sample of NIH Chest X-ray Dataset*[2]. Il s'agit de 5606 échantillons d'une base de données de plus de 112 000 de radiographies de torse anonymisées[3]. Cette base de données a été publiée par la *National Institute of Health (NIH)* pour plus de 30 000 patients uniques[4]. La base de données officielle a été étiquetée par des algorithmes de traitement de langage naturel et il est supposé que les étiquettes sont justes dans plus de 90 % des cas. La figure 1 présente une image typique que l'on peut retrouver dans la base de données.

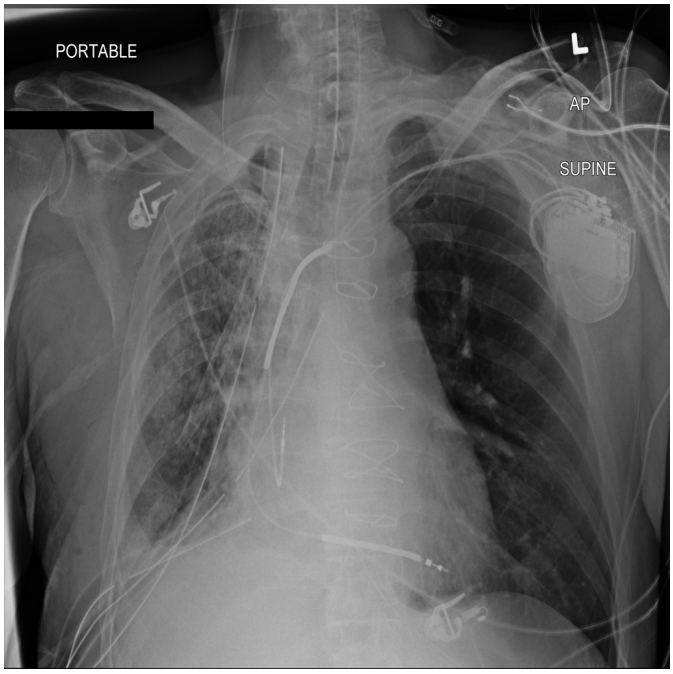


FIGURE 1 – Image typique de la base de données

Le tableau 1 présente un extrait du fichier CSV descriptif de la base de données utilisées pour le projet. Les cibles utilisées lors de l'entraînement sont tirées de ce fichier CSV.

TABLEAU 1 – Extrait du fichier CSV descriptif de la base de données mis sous forme de tableau

Image Index	**Finding Labels**	**Patient Age**	**Patient Gender**
00000013_005.png	Emphysema|Infiltration| Pleural_Thickening|Pneumothorax	060Y	M
00000013_026.png	Cardiomegaly|Emphysema	057Y	M
00000017_001.png	No Finding	077Y	M

Le tableau 1 présente la façon dont l'information sur les images est enregistrée. Ainsi, la première colonne du fichier CSV présente le nom de l'image de la base de données. Ensuite, la seconde colonne présente la ou les pathologies du patient correspondant à l'image. Le terme *No Finding*

signifie qu'aucune pathologie n'a été trouvée chez ce patient. L'âge et le sexe du patient sont aussi indiqués.

En plus des colonnes présentées au tableau 1, le fichier contient aussi la colonne *Follow-up #* correspondant au nombre de suivis faits entre le patient et le médecin, la colonne *Patient ID* correspondant à l'identification du patient et la colonne *View Position* indiquant la position du patient lors de l'acquisition. Enfin, les colonnes *OriginalImageWidth* et *OriginalImageHeight* indiquent la largeur et la hauteur originales de l'image lors de l'acquisition et les colonnes *OriginalImagePixelSpacing_x* et *OriginalImagePixelSpacing_y* présentent les résolutions originales de l'image. Il est à noter que toutes les images ont été rééchantillionnées à 1024 pixels par 1024 pixels par les concepteurs de la base de données et que toutes les images sont en noir et blanc.

2.1.1 Formulation du problème pour le projet

De façon générale, le but du projet est d'utiliser différentes techniques d'apprentissage machine pour être en mesure de dire si une image correspond à un patient malade ou non. Si le patient est malade, les algorithmes doivent être capables de dire quelles sont les pathologies associées à l'image. Au fur et à mesure que le projet avançait, l'équipe s'est rendu compte que le projet choisi avait une complexité intéressante. En effet, il ne s'agit pas de seulement classifier une image selon une cible mutuellement exclusive comme ce serait le cas pour la classification de feuilles d'arbre. Une feuille ne peut appartenir qu'à un seul type d'arbre. Dans le cas présent, un patient peut avoir une ou plusieurs pathologies. Ainsi, une image peut être associée à une ou plusieurs étiquettes. De plus, l'étiquette *No Finding* est mutuellement exclusive de toutes les autres pathologies. Donc, l'équipe a décidé d'aborder le problème de deux façons différentes. Premièrement, il a été décidé d'inclure l'étiquette *No Finding* dans les pathologies potentielles. L'équipe voulait savoir si les algorithmes allaient être en mesure de comprendre que lorsqu'un patient n'était pas malade, toutes les autres étiquettes tombaient automatiquement négatives. La deuxième façon dont le problème a été abordé, a été d'entraîner un premier classifieur à reconnaître si les patients étaient malades ou non. Lorsque le premier classifieur affirme qu'un patient est malade, ce patient est passé à un second algorithme pour déterminer la ou les pathologies (le second classifieur ayant été préalablement entraîné avec des images de patients malades). Le but d'aborder le problème de deux façons différentes était de voir comment les résultats finaux allaient varier. Ces deux façons de faire sont dénotées comme des « types » différents de classifieurs pour la suite du rapport. Finalement, dans un objectif de simplification d'un problème déjà assez complexe, l'équipe a décidé de n'utiliser que les images des patients comme données et la colonne *Finding Labels* comme étiquettes, et de ne pas utiliser les autres informations du tableau.

2.2 Gestion de projet

La gestion des différentes versions du projet s'est faite grâce au logiciel de gestion de versions Git. Il a été décidé de s'inspirer de la méthode *Gitflow* pour utiliser Git, c'est-à-dire que le projet contient une branche *master*, une branche *dev* (ou développement), et plusieurs branches *features* qui ont comme racine la branche *dev*. Chaque membre de l'équipe a travaillé autant que possible sur les branches *feature* et lorsque cette branche était terminée, un *pull request* était fait pour venir la fusionner à la branche *dev*. La figure 2 illustre ce processus.

Également, pour la gestion des tâches entre les membres, l'équipe a décidé d'utiliser le logiciel

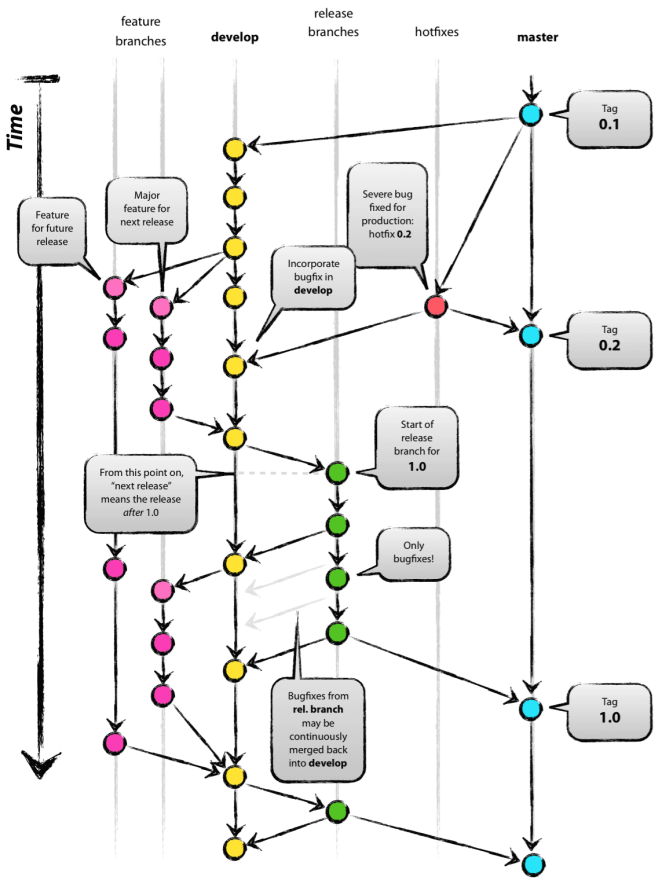


FIGURE 2 – Modèle GitFlow[5]

ZenHub. Il utilise la même méthode de fonctionnement que le logiciel *Trello*, inspiré de la méthode Kanban. La figure 3 présente l'interface *ZenHub*. Il est possible de voir que cette interface est directement intégrée au projet au moyen d'un nouvel onglet dans *GitHub*. L'équipe trouvait cela plus convivial que *Trello*, car il était possible de passer de l'onglet *Code* à l'onglet *ZenHub* en un seul clic à même le site *GitHub*. *ZenHub* s'intègre facilement dans *GitHub* en installant une extension dans *Google Chrome*[6].

2.3 Choix de design

Le code permettant d'entraîner les différents modèles comporte une structure à plusieurs dossiers. Premièrement, toutes les données sont mises dans le dossier *data*. À l'intérieur du répertoire, les images, en format PNG, sont enregistrées dans le dossier *sample/images*. Les cibles des pathologies se trouvent dans le fichier CSV *sample_labels.csv*.

Le coeur du code se trouve dans le dossier *src*. Le fichier *main_ift712.py* contient le script qui est exécuté lorsqu'on lance le programme. Ce script appelle les différentes classes et les différentes méthodes pour déterminer le meilleur classifieur. Le fichier *DataHandler.py* contient la classe *DataHandler*. Cette classe a pour but de s'occuper de la gestion des données. Elle sert à télécharger les données et les transformer dans des formats manipulables par les autres méthodes. Elle réduit aussi la taille des images à 32 pixels par 32 pixels, car sinon les algorithmes mettent trop de temps à s'exécuter. La classe *Metrics* contenue dans le fichier *Metrics.py* contient différentes méthodes

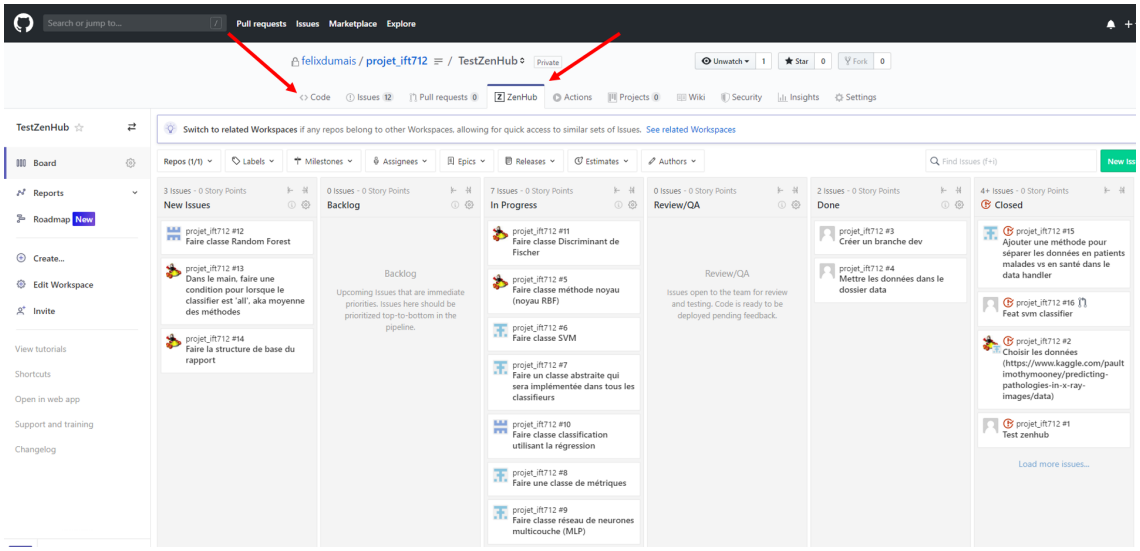


FIGURE 3 – Interface ZenHub

permettant de calculer les métriques utiles à l'analyse des résultats. Le fichier *utils.py* contient une fonction utilisée dans plusieurs autres fichiers.

Tous les classifieurs se trouvent dans le dossier *src/models*. Chacun est implémenté par une classe qui lui est propre. Chaque classe implémentant les classifieurs hérite de la classe *Classifier* qui est dans le fichier *Classifier.py*. Le but de faire de l'héritage était d'avoir sensiblement la même structure pour chacun des classifieurs et ainsi faciliter leur utilisation.

Finalement, le fichier *.gitignore* et le fichier *README.md* sont placés à la racine du projet. Le fichier *README.md* donne des informations quant à l'utilisation du code tandis que le fichier *.gitignore* permet de ne pas suivre certains fichiers par git.

2.3.1 Utilisation du fichier *main_ift712.py*

Le fichier *main_ift712.py* est le fichier qui est exécuté pour entraîner les différents modèles. Il est possible d'exécuter le script avec des lignes de commande dans un terminal grâce à la fonction *argument_parser()*. Les entrants de la fonction sont le choix du modèle à entraîner, la grosseur de l'ensemble d'entraînement, si l'on désire faire une *cross validation* du modèle ou non, le type de classifieur et l'argument *verbose* pour avoir un retour du programme.

Les modèles choisis pour le projet sont les machines à vecteurs de support, le discriminant de Fisher, un réseau de neurones multicouches, un modèle utilisant un noyau RBF, les forêts d'arbres décisionnels et un modèle utilisant la régression logistique. Ces modèles sont décrits à la section 2.4. Il est également possible de spécifier *all* dans les choix de modèles. Ceci permet d'entraîner tous les modèles et de les combiner par vote majoritaire. Pour ce qui est de la grosseur de l'ensemble d'entraînement, l'utilisateur doit choisir une valeur entre 0 et 1 pour spécifier le pourcentage de l'ensemble de données utilisé pour l'entraînement. Finalement, le type de classifieur revient à la façon dont l'équipe a choisi de résoudre le problème. Si l'utilisateur spécifie 1, un seul classifieur est entraîné en considérant les patients sains comme une pathologie. Si l'utilisateur spécifie 2, deux classifieurs sont entraînés selon la méthode décrite à la section 2.1.1.

Une fois que l'utilisateur lance le programme, tous ces paramètres sont enregistrés dans des variables. Ensuite, une instance de la classe *DataHandler* est déclarée. Lors de l'instanciation de la classe, les images sont téléchargées, réduites à une dimension et mises dans un tableau *numpy* où la première dimension correspond aux indices des images. Aussi, la méthode *_import_csv* de la classe *DataHandler* lit le fichier CSV. La méthode trouve chaque pathologie unique du fichier et crée un tableau *numpy* avec comme première dimension les indices des images et comme deuxième dimension les pathologies associées à chacune. Ce tableau ne comporte que des valeurs de 0 ou 1, indiquant l'appartenance ou non d'une image à une pathologie.

Avant d'entraîner le modèle, la fonction *main* appelle les méthodes *plot_data* et *show_samples* de la classe *DataHandler*. La figure 4 présente quelques échantillons de la base de données utilisés pour entraîner les données, la figure 5 présente la distribution des patients sains et des patients malades et la figure 6 présente la distribution de toutes les pathologies chez les patients. Il est à noter que dans la figure 6, la somme du nombre de pathologies est supérieure au nombre d'images puisqu'il peut y avoir plus d'une pathologie par patient.

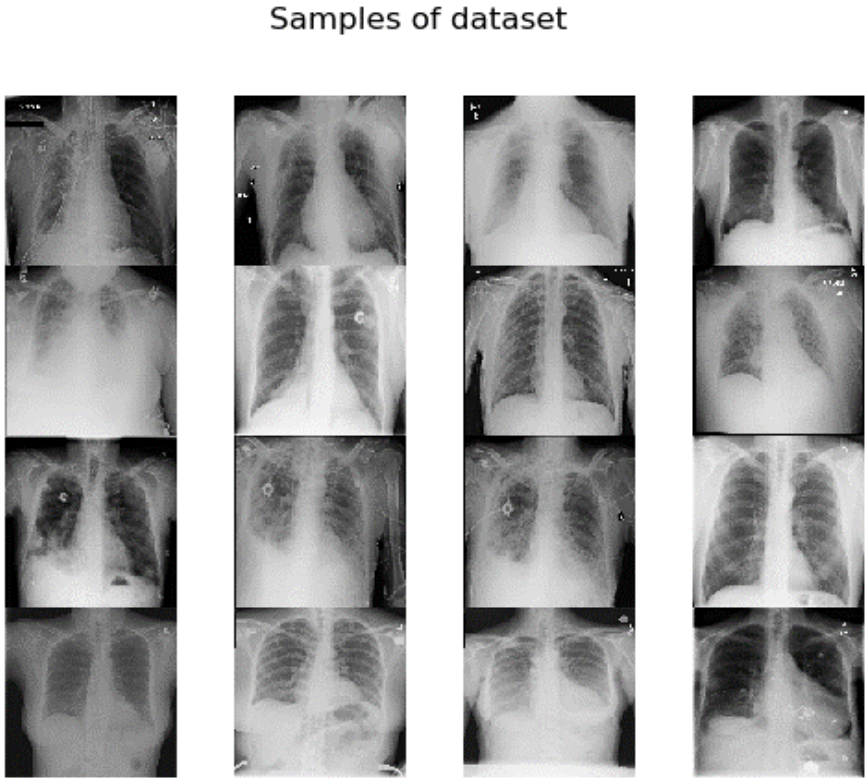


Figure 4 – Affichage d'un échantillon de la base de données

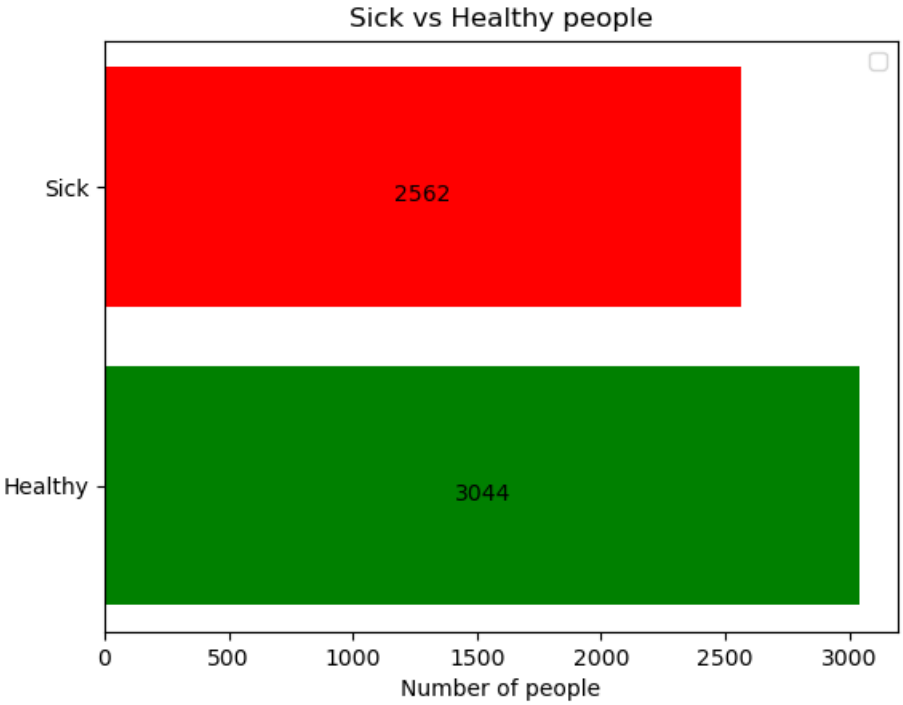


FIGURE 5 – Distribution du nombre de patients sains et malades

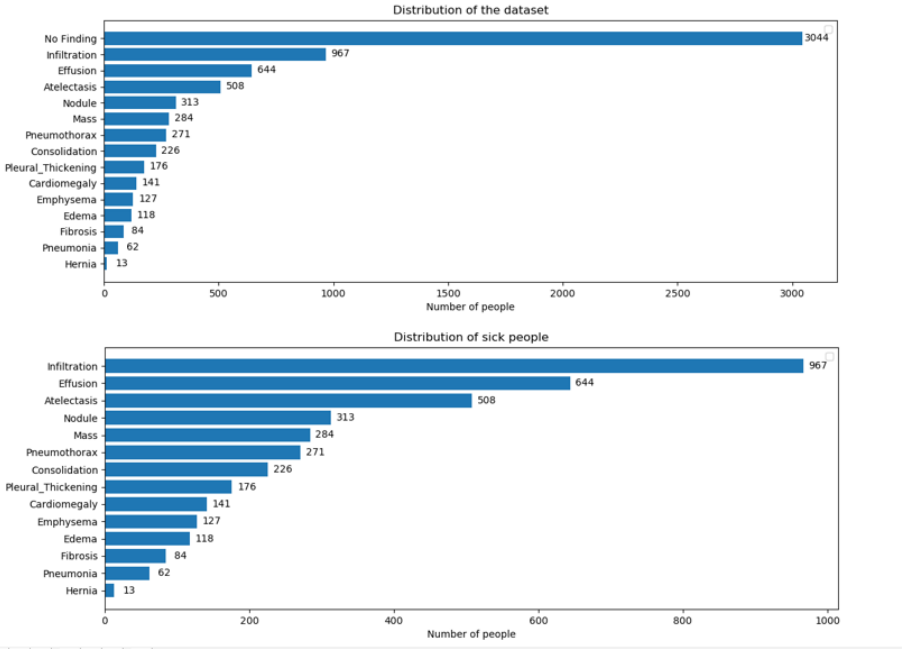


FIGURE 6 – Distribution des pathologies chez les patients. En haut de la figure : affichage incluant les patients sains. En bas de la figure : affichage excluant les patients sains.

Finalement, une instance de la classe *Trainer* est déclarée pour entraîner le ou les modèles choisis.

2.4 Méthodes utilisées

Toutes les méthodes ont été implémentées avec l'interface de programmation *scikit-learn*. Comme le projet était fondamentalement un problème multiclasses et multicibles, le problème a été transformé en un problème de relevance binaire, c'est-à-dire que l'équipe implémentait un classifieur binaire sur chacune des classes possibles. Chaque classifieur était indépendant des autres. Les gestions de la relevance binaire s'est faite avec la classe *sklearn.multiclass.OneVsRestClassifier*. La validation croisée k-fois a été effectuée avec la classe *sklearn.model_selection.GridSearchCV* qui permet de tester plusieurs hyperparamètres de façon parallèle.

2.4.1 Discriminant de Fisher

La méthode du discriminant de Fisher est une sous-classe des méthodes de classification linéaire. Elle fait l'hypothèse que les données sont réparties selon une gaussienne. L'idée est de projeter les données sur une droite de façon à ce que la différence entre la moyenne de la projection des données de chaque classe soit maximale[7]. Des développements mathématiques tels que ceux vus en classe permettent de réduire le problème à une simple inversion de matrice. Il s'agit donc aussi d'une méthode de type « closed form ». La classe *LinearDiscriminantAnalysis* de la librairie *sklearn.discriminant_analysis* a été utilisée pour implémenter la méthode.

2.4.2 Régression logistique

La régression logistique est une version améliorée de la méthode du perceptron, où la fonction d'activation, ou la non-linéarité est changée d'une fonction signe à une fonction sigmoïdale. Comme la fonction sigmoïde envoie sur un espace continu de zéro à un ([0,1]), on dit que la sortie d'une machine utilisant la régression logistique est en fait la probabilité qu'une donnée d'entrée appartienne à la première classe. Aussi, un réseau logistique, contrairement au perceptron, continue de s'entraîner même si les données sont toutes bien classées puisque la valeur du gradient n'est jamais exactement nulle. La classe *LogisticRegression* de la librairie *sklearn.linear_model* a été utilisée pour implémenter la méthode.

2.4.3 Réseau de neurones multicouches

Les réseaux de neurones multicouches sont la suite logique de la régression logistique. Cependant, la principale différence est que les réseaux de neurones multicouches implémentent une ou plusieurs couches cachées contrairement à la régression logistique. Le principal avantage des couches cachées est de permettre au modèle d'apprendre des fonctions de séparation non linéaire. En effet, la régression logistique ne permet que d'apprendre des modèles linéaires. L'apprentissage des modèles se fait de manière itérative à l'aide d'une propagation avant et d'une rétropropagation du gradient. La propagation avant permet de calculer la perte avec une fonction de coût. Dans le cas présent, la fonction de coût est l'entropie croisée. Ensuite, les paramètres du modèle sont modifiés selon le gradient de la fonction de coût. La classe *MLPClassifier* de la librairie *sklearn.neural_network* a été utilisée pour implémenter les réseaux de neurones multicouches.

2.4.4 Méthode à noyau

Les méthodes à noyau optimisent la classification de données à partir de la représentation duale de la fonction d'erreur. Elles impliquent le calcul de la matrice de Gram, dont chaque élément est un noyau. Plusieurs types de noyaux existent ; pour ce projet, les *radial base function* (noyaux RBF) ont été choisis puisque c'est une des méthodes vues en classe. Il s'agit d'une méthode de type « closed form » qui ne suppose pas que les données sont réparties de façon gaussienne, mais qui espère néanmoins qu'une segmentation des données via l'application d'un noyau RBF permet de rendre les données linéairement séparables. La classe *GaussianProcessClassifier* de la librairie *sklearn.gaussian_process* a été utilisée pour implémenter la méthode.

2.4.5 Machine à vecteurs de support

Les machines à vecteur de support (SVM) sont la suite logique des méthodes à noyau. En effet, ils requièrent le calcul d'une matrice de Gram et la classification des données se fait à partir de la représentation duale de la fonction d'erreur. Cependant, les SVM font appel à la notion de marge. Cette marge correspond à la plus petite distance entre la surface de séparation et les données d'entraînement. Les vecteurs de support correspondent aux données qui tombent sur cette marge. Tout comme les méthodes à noyau, la prédiction d'une donnée d'un ensemble de test se fait en calculant une distance noyau. Contrairement aux méthodes à noyau, cette distance n'est pas calculée entre la nouvelle donnée et toutes les données de l'ensemble d'entraînement, mais seulement entre la nouvelle donnée et les vecteurs de support. Le grand avantage est que moins de mémoire est nécessaire. L'implémentation des SVM s'est faite avec la classe *SVC* de la librairie *sklearn.svm*.

2.4.6 Forêt d'arbres décisionnels

La méthode de la forêt d'arbres décisionnels est basée sur une combinaison de type vote majoritaire. Plusieurs arbres décisionnels sont d'abord générés de façon à ce que les erreurs produites par ces deniers soient le moins corrélées. Puisque les arbres décisionnels sont des modèles ayant une forte capacité, il est alors possible de combiner leurs résultats avec la méthode de *Bootstrap AGGregatING* afin d'abaisser leur variance et obtenir de meilleures performances de prédiction. La classe *RandomForestClassifier* de la librairie *sklearn.ensemble* a été utilisée pour implémenter la méthode.

2.5 Métriques étudiées

La performance globale du modèle a été évaluée avec cinq métriques, soient le Kappa de Cohen, la mesure F1, la justesse, la précision et le rappel. Également, les courbes précision-rappel et les courbes ROC ont été déterminées. Une métrique a été calculée pour chacune des classes et une métrique moyenne a été calculée pour la performance globale du modèle. Chacune des métriques est calculée à partir de la matrice de confusion obtenue avec les prédictions faites par les classifieurs (voir la figure 7). La matrice de confusion présente les résultats obtenus entre les cibles réelles et les prédictions faites par les classifieurs.

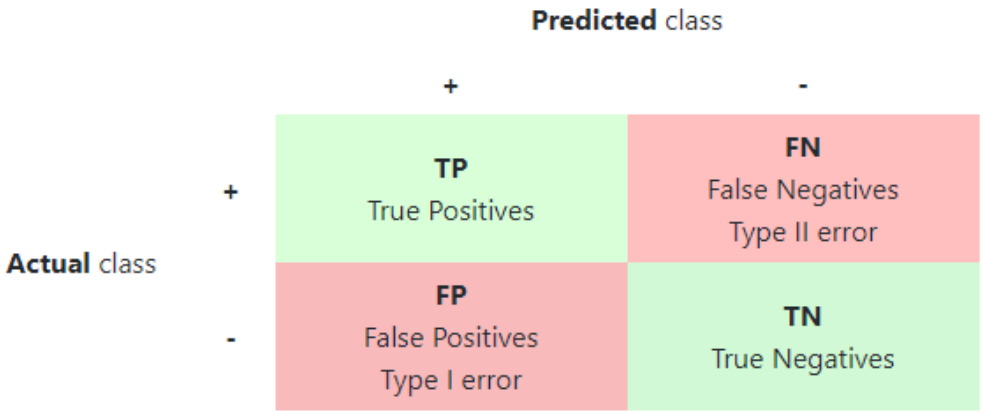


FIGURE 7 – Matrice de confusion [8]

Pour les prochaines sections, les symboles de la figure 7 seront utilisés pour écrire les équations. Ainsi,

TP = nombre de vrais positifs obtenus (*true positives* ou TP en anglais)
TN = nombre de vrais négatifs obtenus (*true negatives* ou TN en anglais)
FP = nombre de faux positifs obtenus (*false positives* ou FP en anglais)
FN = nombre de faux négatifs obtenus (*false negatives* ou FP en anglais)

2.5.1 Justesse

La justesse est une métrique très utilisée pour connaître la performance d'un modèle en techniques d'apprentissage. Cependant, l'un des désavantages de la justesse est qu'elle apporte peu d'information lorsqu'il n'y a pas d'équilibre entre les valeurs positives et négatives. L'équation 1 permet de calculer la justesse.

$$ACC = \frac{TP + TN}{TP + FN + FP + TN} \tag{1}$$

2.5.2 Kappa de Cohen

L'une des métriques qui est intéressante de connaître lorsqu'il y a un manque d'équilibre entre les valeurs positives et les valeurs négatives est le Kappa de Cohen. Il s'agit d'une métrique qui permet de connaître le niveau d'accord entre deux «correcteurs». Dans le cas présent, les correcteurs sont les valeurs prédites et les valeurs cibles.

Pour calculer le Kappa de Cohen il faut premièrement calculer la probabilité d'obtenir un vrai positif de manière aléatoire avec l'équation 2.

$$p_{TP} = \frac{TP + FN}{TP + FN + FP + TN} \times \frac{TP + FP}{TP + FN + FP + TN} \tag{2}$$

Ensuite il faut calculer la probabilité d'obtenir un vrai négatif de manière aléatoire avec l'équation 3.

$$p_{TN} = \frac{FP + TN}{TP + FN + FP + TN} \times \frac{FN + TN}{TP + FN + FP + TN} \tag{3}$$

Les équations 2 et 3 permettent de calculer la probabilité globale d'entente entre les cibles et les prédictions données à l'équation 4.

$$p_e = p_{TP} + p_{TN} \tag{4}$$

Finalement, il faut utiliser la justesse de l'équation 1 pour faire le test du Kappa de Cohen. Dans le cas présent, la justesse sera notée p_o. Ainsi, le Kappa de Cohen se calcule avec l'équation 5.

$$\kappa = \frac{p_o - p_e}{1 - p_e} \tag{5}$$

2.5.3 Précision

La précision est une métrique permettant de calculer la capacité d'un classifieur à ne pas étiqueter comme positives les valeurs négatives. L'équation 6 donne l'équation de la précision.

$$Pr = \frac{TP}{TP + FP} \tag{6}$$

2.5.4 Rappel

Le rappel, ou taux de vrais positifs est une métrique complémentaire à la précision qui indique la capacité d'un classifieur à bien étiqueter les valeurs positives. L'équation 7 donne l'équation du rappel.

$$Re = \frac{TP}{TP + FN} \tag{7}$$

2.5.5 Mesure F_1

La mesure F_1 est une métrique qui combine la précision et le rappel. Elle peut être interprétée comme la moyenne pondérée de la précision et du rappel. L'équation 8 donne la formule de la mesureF_1

$$F_1 = \frac{2 \times Pr \times Re}{Pr + Re} \tag{8}$$

2.5.6 Courbe ROC et précision-rappel

La courbe ROC (ou *receiver operating characteristic curve* en anglais) s'obtient en calculant le taux de faux positifs et le rappel (eq. 7) en faisant varier le seuil décisionnel d'un classifieur binaire. Le taux de faux positifs se calculent avec l'équation 9 et la figure 8 présente une courbe ROC.

$$FPR = \frac{FP}{FP + TN} \tag{9}$$

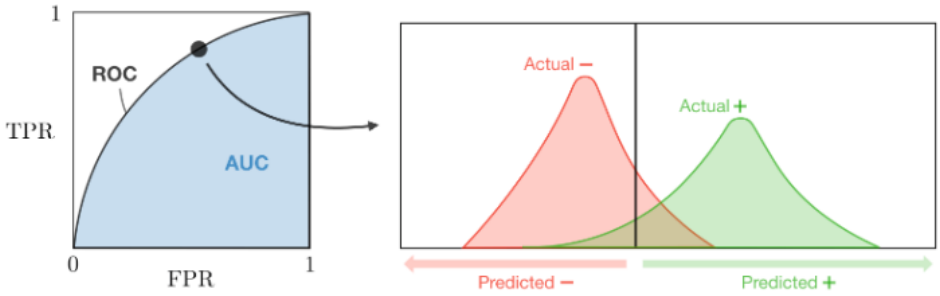


FIGURE 8 – À gauche : courbe ROC, à droite : distribution des valeurs positives et négatives avec le seuil décisionnel [8]

Plus l'aire sous la courbe d'une courbe ROC tend vers 1, plus le modèle à la capacité de bien séparer les données positives et négatives.

La courbe précision-rappel est une autre courbe permettant de juger de la capacité d'un modèle lorsqu'il y a un manque d'équilibre entre les valeurs positives et négatives. Elle s'obtient en calculant le rappel et la précision en faisant varier le seuil décisionnel d'un classifieur binaire. La figure 9 présente une courbe précision-rappel. Comme pour la courbe ROC, plus l'aire sous la courbe tend vers 1, plus le modèle a la capacité de bien séparer les données.

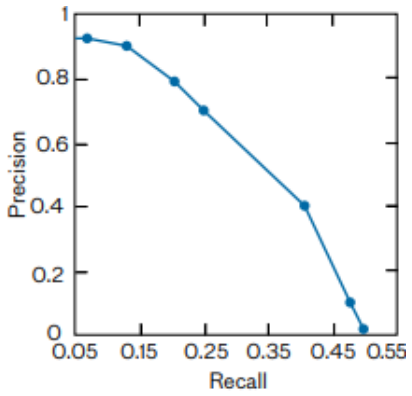


FIGURE 9 – Courbe précision-rappel [9]

3 Présentation et analyse des résultats

3.1 Résultats des modèles

Cette section présente les résultats obtenus lors de la validation croisée des différents modèles. Comme discuté à la section 2.4, le code implémente la fonction *GridSearchCV* de la la librairie *sklearn.model_selection* pour effectuer cette étape. Cette méthode est appelée dans la méthode *_research_hyperparameter* de chacune des classes implémentant un classifieur. Il a été décidé de faire trois «fold» pour la validation croisée, car avec cinq, l'algorithme prenait beaucoup trop de temps pour calculer les hyperparamètres. Également, tous les paramètres de chacun des classifieurs ont été optimisés sur un classifieur de type 1 avec des images rééchantillonnées à 32 pixels par 32 pixels. Si l'équipe avait eu accès à du matériel informatique plus puissant, probablement qu'elle aurait entraîné les algorithmes sur des images moins compressées, sur les 2 types de classifieurs et avec plus de «folds». Cependant, comme la validation croisée prenait plusieurs heures pour chacun des modèles, ce sacrifice a dû être fait.

Également, l'ensemble d'entraînement utilisé consistait à 85 % des 5606 images de la base de données.

3.1.1 Discriminant de Fisher

Le solutionneur *svd* fut choisi étant donné la grande taille des données. Tout autre solutionneur causait des erreurs de mémoire. Les valeurs optimales des hyperparamètres *tol* et *n_components* sont respectivement 0,01 et 12.

La méthode du discriminant de Fisher prend très peu de temps à s'entraîner ; quelques minutes tout au plus. Les tableaux 4 et 5 présentent les résultats obtenus. Globalement, il appert que cette méthode a permis de détecter des cas positifs dans 10 classes sur 15 et 9 classes sur 15 pour les classificateurs de catégorie 1 et 2 respectivement. Par contre, avec une précision globale de 0,38/0,31, il faut admettre que la plupart de ces cas positifs sont en réalité des faux positifs. Avec un rappel de 0,35/0,37, seul le tiers des patients sont diagnostiqués correctement. Il n'y a pas de différence significative entre les résultats des deux types de classificateurs.

3.1.2 Régression logistique

La méthode implémentée dans *scikit-learn* pour effectuer de la régression logistique possède plusieurs hyperparamètres optionnels à optimiser. Pour le projet, quelques-uns de ces paramètres ont été optimisés à l'aide d'une validation croisée. Les principaux paramètres optimisés sont C qui représente l'inverse du pouvoir de régularisation, *solver* qui représente le solveur utiliser pour faire l'optimisation et *class_weight* qui permet d'attribuer un certain poids aux différentes classes si certaines sont moins bien représentées dans les données.

Les paramètres optimaux retournés par la validation croisée sont 1 pour C, *liblinear* pour le solveur, et *balanced* pour les poids attribués aux classes qui ajustent ces poids selon l'inverse de leur fréquence dans les données d'entraînement. Malgré cela, les résultats obtenus pour la régression logistique sont plutôt faibles comparés aux autres méthodes. Les résultats (présents aux tableaux 6 et 7) montrent que pour certaines classes, le Kappa de Cohen est négatif, ce qui veut dire que

le modèle performe moins bien qu'un simple choix aléatoire pour ces classes. Le modèle n'a aussi pas réussi à classer 4 des 15 classes présentes. Cela semble être dû à une sous-représentation de ces classes dans les données puisque ce sont 4 des 5 classes les moins présentes. Il ne semble pas y avoir de gain de performance notable entre les types 1 et 2 de classeurs et le temps de calcul des deux types était très similaire, les deux prenaient quelques minutes seulement à rouler.

La valeur donnée par la validation croisée pour l'inverse du pouvoir de régularisation correspond à la valeur minimale de l'intervalle testé pour la validation. Il est donc possible que la valeur optimale soit plus faible que 1, ce serait une avenue à explorer si jamais le modèle devait être poussé plus loin.

3.1.3 Réseau de neurones multicouches

Comme il a été discuté à la section 2.4.3, la classe *MLPClassifier* de la librairie *sklearn.nural_network* a été utilisée pour implémenter le réseau de neurones multicouches. Les paramètres qui ont été optimisés sont le paramètre *alpha* qui est le terme de régularisation L_2 et le paramètre *hidden_layer_sizes* qui définit le nombre de couches cachées et le nombre de neurones par couche. Les valeurs d'*alpha* testées étaient : $[1 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2}, 1 \times 10^{-1}, 1 \times 10^{0}]$. Les valeurs de *hidden_layer_sizes* testées variaient entre une seule couche cachée de 10 neurones à trois couches cachées de 1000 neurones.

Ainsi, les hyperparamètres qui optimisaient au mieux le réseau de neurones étaient 3 couches cachées de 10 neurones et un terme de régularisation de 0,01.

Lorsque l'on entraîne l'algorithme avec les hyperparamètres trouvés dans la validation croisée, les résultats du classifieur de type 1 sont présentés au tableau 2 et ceux du classifieur de type 2 sont présentés au tableau 3. Il est possible de voir que pour l'ensemble des métriques les résultats sont plus satisfaisants pour un classifieur de type 2. Il faut cependant faire attention avec ces résultats, car ils incluent la capacité du modèle à séparer les patients malades et les patients sains. Pour avoir une meilleure idée de la performance, il peut être intéressant d'aller voir les tableaux 8 et 9 en annexe (section 5). Ces tableaux montrent que les résultats obtenus avec le réseau de neurones multicouches sont plus que décevants. Ils nous montrent qu'il n'est même pas capable de classifier les maladies. Cela peut être dû au fait que les images utilisées ont été compressées à 32 x 32. Également, l'équipe suppose qu'il y a eu un problème avec la validation croisée pour le réseau de neurones multicouches. En effet, le réseau de neurones a réussi à classifier un peu les patients malades ou non, mais il n'était pas du tout en mesure de trouver les maladies des patients malades.

3.1.4 Méthode à noyau

La fonction de *scikit-learn* utilisée pour faire la méthode à noyau optimise automatiquement les paramètres du noyau fourni. Les autres hyperparamètres choisis sont *n_restarts_optimizer*, i.e. le nombre de fois que les paramètres du noyau sont optimisés (optimisé à 2), ainsi que *max_iter_predict*, i.e. le nombre d'itérations utilisées lors de la prédiction (optimisé à 10). Ce dernier paramètre aurait pu être plus élevé au prix d'un temps de calcul plus élevé, mais puisque la méthode prend déjà plusieurs heures pour arriver à des résultats, la valeur de 10 fut conservée.

Les résultats du tableau 10 montrent que la méthode à noyau de type 1 n'arrive qu'à détecter

les patients sains, et avec une précision de seulement 0,66. Ce n'est pas très efficace. Les résultats du tableau 11 montrent que la méthode à noyau de type 2 est légèrement meilleure, car elle détecte aussi les cas d'infiltration avec une précision de 0,57. C'est donc dire que plus de la moitié des cas d'infiltration dénotés positifs sont des vrais positifs. Cependant, le rappel de 0,08 pour cette maladie montre que beaucoup de cas ne sont pas détectés par la méthode. Le classificateur de type 2 est donc un peu meilleur que le 1, mais pas de beaucoup. La bonne performance de la méthode à noyau en termes de métriques globales n'est due qu'au fait que les données sont presque toujours associées à l'étiquette « No finding », qui est une étiquette très fréquente.

3.1.5 Machine à vecteurs de support

Comme il a été discuté à la section 2.4.5, la classe *SVC* de la librairie *sklearn.svm* a été utilisée pour implémenter les machines à vecteurs de support. En implémentant la classe OneVsRestClassifier, un classifieur binaire était mis pour chaque étiquette possible. Les paramètres qui ont été optimisés sont le paramètre C qui est le terme de régularisation L_2, le paramètre *degree* qui contrôle le degré d'un noyau polynomial et le paramètre *gamma* qui contrôle l'influence d'une nouvelle donnée d'entraînement sur la frontière de séparation des données. Également, même si un noyau n'est pas un hyperparamètre, il a été décidé de tester plusieurs noyaux dans la validation croisée. Ainsi, le noyau RBF, polynomial et linéaire ont été testé. L'hyperparamètre *gamma* et C sont des coefficients applicables aux trois types de noyaux. Le coefficient *degree* ne s'applique que lorsque le noyau est polynomial. Les valeurs de C testées étaient : [1, 10, 100]. Les valeurs de *gamma* testées étaient : $[1 \times 10^{-6}, 1 \times 10^{-5}, 1 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2}]$. Finalement, les degrés du noyau polynomial variaient entre 3 et 5 inclusivement.

Ainsi, les hyperparamètres qui optimisaient au mieux la machine à vecteurs de support étaient pour un noyau linéaire. Le terme de régularisation C était 100 et le paramètre *gamma* était 0,01. La validation croisée a pris plus de 9 heures à calculer pour cette méthode.

Lorsque l'on entraîne l'algorithme avec les hyperparamètres trouvés dans la validation croisée, les résultats du classifieur de type 1 sont présentés au tableau 2 et ceux du classifieur de type 2 sont présentés au tableau 3. Il est possible de voir que pour l'ensemble des métriques les résultats sont plus satisfaisants pour un classifieur de type 1. Il faut cependant faire attention avec ces résultats, car ils incluent la capacité du modèle à séparer les patients malades et les patients sains. Pour avoir une meilleure idée de la performance, il peut être intéressant d'aller voir les tableaux 12 et 13 en annexe (section 5). Il est possible de voir que la justesse concernant la capacité du classifieur à séparer les patients malades ou non est au mieux de 0,59 pour le classifieur de type 1.

3.1.6 Forêt d'arbres décisionnels

Encore une fois, la méthode de *scikit-learn* pour les forêts d'arbres décisionnels possède beaucoup de paramètres optionnels. Les principaux paramètres optimisés sont le nombre d'arbres dans une forêt *n_estimators*, la mesure d'impureté *criterion*, la profondeur maximale *max_depth*, la variable dictant la méthode de *bootstrapping* est utilisée dans l'entraînement *bootstrap* et le type de poids attribué aux différentes classes *class_weight*.

Les paramètres optimaux retournés par la validation croisée sont 50 pour le nombre d'arbres dans une forêt, *gini* pour l'impureté, 1 pour la profondeur maximale et *True* pour le Bootstrap-

ping. Ces paramètres donnent des résultats plutôt mauvais en comparaison avec ceux des autres méthodes, ce sont par contre ceux-ci qui sont utilisés pour la méthode du vote majoritaire et les courbes ROC et précision/rappel. Les résultats présentés aux tableaux 14 et 15 utilisent un nombre maximal de 150 arbres par forêt et une profondeur maximale de 10, puisque ces paramètres semblaient meilleurs pour le projet. Avec ces paramètres, la forêt d'arbres décisionnels devient la meilleure méthode du projet, réussissant à prédire les maladies les plus présentes jusqu'à environ 33% du temps. Par contre, le modèle semble avoir beaucoup de difficulté avec les classes qui sont moins présentes dans les données d'entraînement.

3.2 Comparaison des résultats de modèles

Les métriques de chaque méthode sont comparées dans les tableaux 2 et 3. Sans surprise, la justesse des résultats est très élevée (autour de 90%), puisque les patients n'ont pour la majorité qu'une seule classe d'appartenance. Ainsi, chaque fois qu'un modèle prédit qu'un patient **n'a pas** une pathologie, la justesse des résultats augmente. Ce n'est donc pas une bonne métrique pour évaluer la qualité des modèles. Il faut plutôt s'en remettre à la précision et au rappel. À ce niveau, il faut admettre que la performance globale des différentes méthodes n'est pas très bonne ; le meilleur modèle en termes de précision, celui des noyaux RBF, n'a raison que sur 66% des étiquettes positives, et le meilleur modèle en termes de rappel, celui de la régression logistique, n'identifie correctement l'étiquette du patient que dans 45% des cas. Il faut aussi noter que puisque les données étiquetées *No Finding* comptent comme des résultats positifs et qu'elles représentent plus de la moitié des cas, les métriques de tous les modèles sont très dépendantes des résultats obtenus pour cette classe. Aussi, on remarque que les maladies très peu fréquentes telles que l'hernie et la pneumonie n'ont été détectées par aucun modèle. Avec respectivement 13 et 62 échantillons de ces maladies sur des milliers de cas, il aurait été surprenant que les modèles arrivent à les détecter. Ces classes peuvent pratiquement être considérées comme du bruit parmi les données, donc elles auraient pu être éliminées pour voir si cela améliore les performances des modèles.

Tableau 2 – Métriques globales des différentes méthodes avec un classifieur de type 1

Méthode	**Kappa de Cohen**	**Mesure F_1**	**Justesse**	**Précision**	**Rappel**
Discriminant de Fisher	0,31	0,36	0,90	0,37	0,35
Régression logistique	0,20	0,29	0,82	0,22	0,45
Réseau de neurones multicouches	0,22	0,29	0,87	0,26	0,34
Méthode à noyau	0,41	0,44	0,93	0,66	0,33
Machine à vecteurs de support	0,26	0,33	0,87	0,29	0,37
Forêt d'arbres décisionnels	0,40	0,44	0,92	0,53	0,38
Vote majoritaire	0,34	0,40	0,90	0,39	0,41

Parmi les maladies les plus fréquentes, soit l'infiltration (967 cas), l'effusion (644 cas) et l'atelectaris (508 cas), seules la première et la troisième sont détectées par presque tous les modèles. Il est curieux de noter que seule une minorité de modèles arrive à détecter l'effusion. Les membres de l'équipe ne sont pas des experts en médecine, mais il se pourrait que cette maladie soit uniquement

Tableau 3 – Métriques globales des différentes méthodes avec un classifieur de type 2

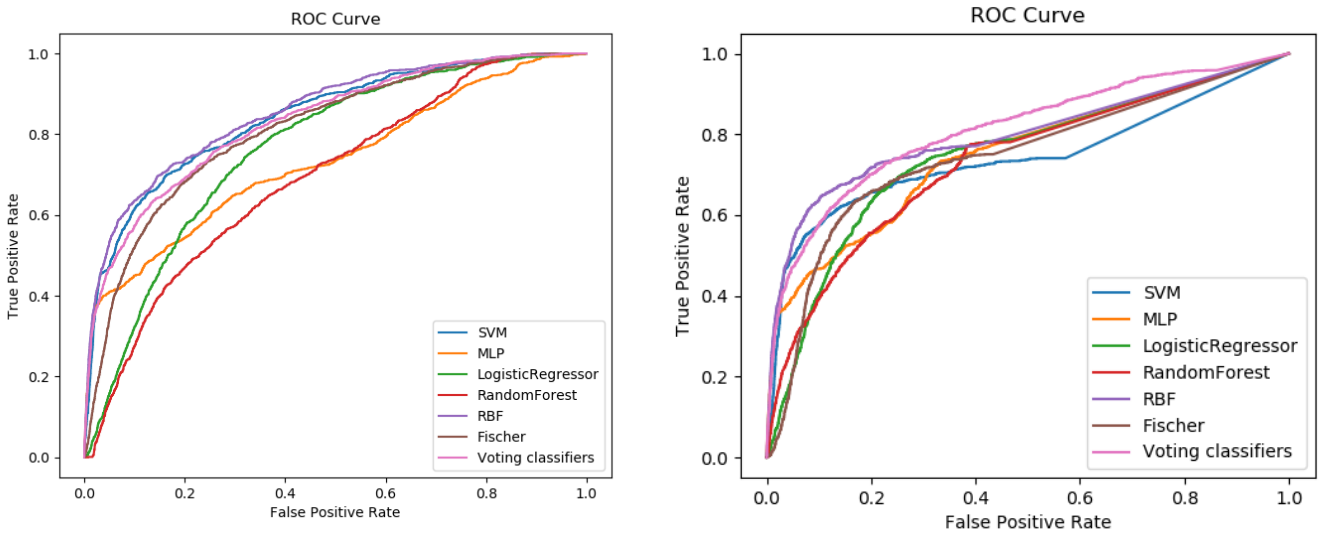
Méthode	**Kappa de Cohen**	**Mesure F_1**	**Justesse**	**Précision**	**Rappel**
Discriminant de Fisher	0,27	0,34	0,88	0,31	0,37
Régression logistique	0,25	0,32	0,86	0,27	0,40
Réseau de neurones multicouches	0,46	0,50	0,93	0,56	0,45
Méthode à noyau	0,42	0,45	0,93	0,66	0,34
Machine à vecteurs de support	0,18	0,26	0,86	0,22	0,30
Forêt d'arbres décisionnels	0,41	0,45	0,92	0,54	0,39
Vote majoritaire	0,37	0,42	0,91	0,44	0,39

détectable à partir de petits détails dans les radiographies et que sa détection soit compromise par la réduction des images à des fichiers de 32×32 pixels. Ce changement de résolution affecte assurément la qualité de tous les résultats ; si l'équipe avait eu accès à des machines plus puissantes, il aurait été intéressant de voir si la performance des algorithmes sur les images originales de 1024×1024 pixels permettrait de mieux détecter cette maladie.

Il est intéressant de constater que dans le cadre de ce projet, le vote majoritaire n'est pas la meilleure des méthodes selon les métriques. En effet, au chapitre de la mesure F_1 et du Kappa de Cohen, le vote majoritaire se classe 3^e, derrière la méthode à noyaux ($1^{\text{ère}}$) et la forêt d'arbres décisionnels (2^e). Cela pourrait s'expliquer par le grand nombre de classes ainsi que la tendance générale des modèles à classer les données dans la catégorie *No Finding*. En effet, dès que 3 modèles classent un patient dans cette catégorie, il faut absolument que les 3 autres modèles soient d'accord sur une maladie précise pour que l'image soit classée autrement. Puisque deux de nos modèles (méthode à noyaux, réseau de neurones multicouches) classent la majorité des données dans la catégorie *No Finding*, il n'est pas surprenant que le vote majoritaire n'arrive pas à faire mieux.

En supposant que les résultats du vote majoritaire représentent une sorte de moyenne des résultats de toutes les méthodes pour chaque type de classifieur, on peut conclure que le classifieur de type 2 permet d'obtenir de meilleurs résultats que celui de type 1 car toutes les métriques à l'exception du rappel sont meilleures. Ceci est logique, puisque séparer le problème en deux étapes rétablit un certain équilibre entre les classes de données. Comme montré à la figure 5, plus de la moitié des patients n'ont aucune maladie. Faire un premier filtrage de ces données permet par la suite aux pathologies d'avoir un plus grand poids dans la deuxième étape de l'entraînement. Il faut toutefois noter que les hyperparamètres utilisés pour les deux types de modèles proviennent d'une optimisation faite avec GridSearch sur les classifieurs de type 1.

À la lumière de ces facteurs, il y aurait plusieurs façons de déclarer une méthode meilleure que les autres. Certains modèles (méthode à noyaux, réseau de neurones multicouches) sont frileux sur les diagnostics et ont tendance à placer les patients dans la catégorie *No Finding* la plupart du temps. Ils ne déclarent donc pas de gens malades sans raison, mais déclarent beaucoup de gens sains qui ne le sont pas nécessairement. D'autres modèles (discriminant de Fisher, SVM) vont détecter adéquatement des cas de la plupart des maladies, mais pas de façon précise ni avec un bon rappel.

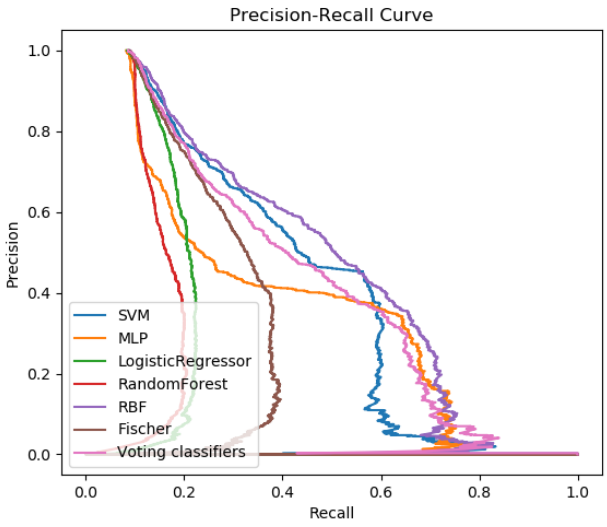


(a) (b)

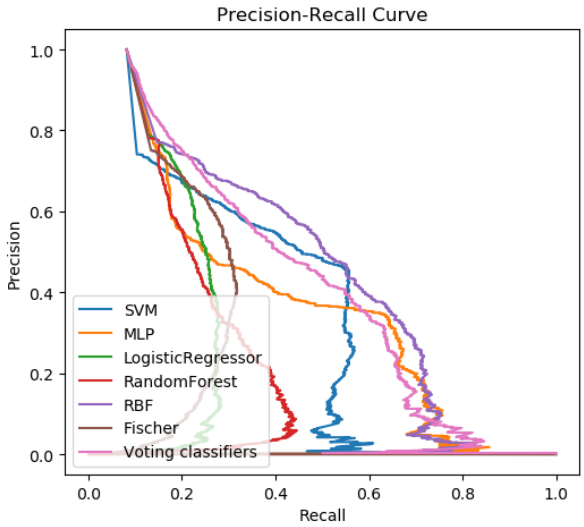
FIGURE 10 – Courbe ROC pour les classifieurs de type 1 (a) et de type 2 (b).

Finalement, les autres modèles (régression logistique et forêt d'arbres décisionnels) détectent peu de maladies différentes, mais sont plus fiables au niveau de la précision et du rappel. Ces deux dernières méthodes seraient techniquement les plus utiles de toutes pour un médecin dans la vie réelle, car elles pourraient permettre un premier filtrage des images en ce qui a trait aux maladies les plus fréquentes. Cependant, tel que visible sur les figures 10 et 11, leurs performances sont loin d'être idéales et il faudrait encore les peaufiner grandement avant de les utiliser. D'un point de vue purement basé sur ces courbes, c'est la méthode à noyau qui serait le meilleur choix pour classer ces données parmi les méthodes implémentées.

Puisque l'ensemble des données provient d'un « dataset » Kaggle et non pas d'une compétition, il n'a pas été possible de mettre les résultats sur le site. Par contre, des notebooks avec des résultats d'autres utilisateurs ainsi que leur démarche sont disponibles. Les résultats ont donc été comparés avec ceux du notebook *Train Simple XRay CNN* de K. Scott Mader, i.e. celui le mieux coté. Celui-ci a éliminé la catégorie *No Finding* ainsi que les maladies les moins nombreuses, et a réajusté la fréquence et le nombre absolu d'images de chaque maladie. Il arrive à détecter chaque maladie avec une précision allant de 2,6% pour la fibrose (maladie rare) jusqu'à 37,2% pour l'infiltration (maladie fréquente). Ces résultats sont comparables à ceux obtenus par la forêt d'arbres décisionnels dans ce projet, à la différence que cette dernière méthode n'arrivait pas à détecter les maladies plus rares.



(a)



(b)

FIGURE 11 – Courbe précision-rappel pour les classifieurs de type 1 (a) et de type 2 (b).

4 Conclusion

En résumé, six méthodes ont été implémentées pour classer des images de radiographies de patients selon la ou les maladies qu'ils présentent. Deux types de classement ont été effectués : un classement général et une autre version où les patients sains sont d'abord filtrés par un premier classifieur, puis les patients considérés malades sont reclassés par maladie dans un deuxième classifieur. Ce deuxième type de méthode semble plus efficace que le premier. Parmi toutes les méthodes, la méthode à noyau utilisant un noyau RBF est celle qui obtient les meilleures métriques, mais elle est malgré cela peu performante. D'autres méthodes, telles que la forêt d'arbres décisionnels, sont meilleures pour détecter un plus grand spectre de maladies, au détriment d'une bonne performance globale.

Ce projet a permis de constater que l'implémentation de méthodes d'apprentissage dans un contexte concret est plutôt complexe, surtout dans le cas d'un problème multiclasse ayant des données déséquilibrées. Il aurait été intéressant d'ajouter l'âge et le sexe des patients dans les données afin de voir si les résultats auraient été meilleurs. Dans tous les cas, les méthodes implémentées ne sont pas assez mûres pour remplacer un médecin !

5 Annexes

Tableau 4 – Résultats du discriminant de Fisher, type 1

Labels	**Kappa de Cohen**	**Mesure F1**	**Justesse**	**Précision**	**Rappel**
Emphysema	-0.0183	0.0000	0.9631	0.0000	0.0000
Infiltration	0.0733	0.2033	0.7669	0.2551	0.1689
Pleural_Thickening	0.0110	0.0400	0.9429	0.0455	0.0357
Pneumothorax	0.0098	0.0533	0.9156	0.0606	0.0476
Cardiomegaly	0.0226	0.0465	0.9512	0.0385	0.0588
No Finding	0.1708	0.6386	0.5922	0.6299	0.6474
Atelectasis	0.0055	0.0794	0.8621	0.0877	0.0725
Edema	-0.0132	0.0000	0.9727	0.0000	0.0000
Effusion	0.0708	0.1647	0.8312	0.1687	0.1609
Consolidation	-0.0043	0.0317	0.9275	0.0400	0.0263
Mass	0.0484	0.0941	0.9084	0.0769	0.1212
Nodule	0.0521	0.0964	0.9108	0.1250	0.0784
Fibrosis	-0.0131	0.0000	0.9703	0.0000	0.0000
Pneumonia	-0.0082	0.0000	0.9834	0.0000	0.0000
Hernia	nan	0.0000	1.0000	0.0000	0.0000

Tableau 5 – Résultats du discriminant de Fisher, type 2

Labels	Kappa de Cohen	Mesure F1	Justesse	Précision	Rappel
Emphysema	-0.0218	0.0000	0.9501	0.0000	0.0000
Infiltration	0.0449	0.2083	0.7289	0.2143	0.2027
Pleural_Thickening	-0.0374	0.0000	0.9275	0.0000	0.0000
Pneumothorax	0.0067	0.0625	0.8930	0.0556	0.0714
Cardiomegaly	0.0061	0.0345	0.9334	0.0244	0.0588
No Finding	0.1708	0.6386	0.5922	0.6299	0.6474
Atelectasis	0.0501	0.1384	0.8371	0.1222	0.1594
Edema	-0.0230	0.0000	0.9489	0.0000	0.0000
Effusion	0.0958	0.2039	0.8050	0.1765	0.2414
Consolidation	0.0571	0.1075	0.9013	0.0909	0.1316
Mass	0.0667	0.1165	0.8918	0.0857	0.1818
Nodule	0.0222	0.0877	0.8763	0.0794	0.0980
Fibrosis	-0.0229	0.0000	0.9548	0.0000	0.0000
Pneumonia	-0.0115	0.0000	0.9762	0.0000	0.0000
Hernia	0.0000	0.0000	0.9988	0.0000	0.0000

Tableau 6 – Résultats de la régression logistique, type 1

Labels	Kappa de Cohen	Mesure F1	Justesse	Précision	Rappel
Emphysema	-0.0268	0.0000	0.9013	0.0000	0.0000
Infiltration	0.0625	0.2762	0.6385	0.2132	0.3919
Pleural_Thickening	-0.0203	0.0323	0.8573	0.0208	0.0714
Pneumothorax	-0.0083	0.0649	0.8288	0.0446	0.1190
Cardiomegaly	0.0318	0.0638	0.8954	0.0390	0.1765
No Finding	0.1994	0.6362	0.6029	0.6489	0.6239
Atelectasis	0.0726	0.1871	0.7313	0.1244	0.3768
Edema	0.0436	0.0714	0.8763	0.0408	0.2857
Effusion	0.1266	0.2579	0.7194	0.1775	0.4713
Consolidation	0.0758	0.1392	0.8383	0.0917	0.2895
Mass	0.0140	0.0788	0.7776	0.0471	0.2424
Nodule	0.0334	0.1206	0.7919	0.0811	0.2353
Fibrosis	-0.0300	0.0000	0.9275	0.0000	0.0000
Pneumonia	-0.0157	0.0000	0.9489	0.0000	0.0000
Hernia	0.0000	0.0000	0.9952	0.0000	0.0000

Tableau 7 – Résultats de la régression logistique, type 2

Labels	Kappa de Cohen	Mesure F1	Justesse	Précision	Rappel
Emphysema	-0.0249	0.0000	0.9275	0.0000	0.0000
Infiltration	0.0449	0.2201	0.7134	0.2112	0.2297
Pleural_Thickening	-0.0221	0.0235	0.9013	0.0175	0.0357
Pneumothorax	-0.0160	0.0500	0.8644	0.0385	0.0714
Cardiomegaly	-0.0047	0.0267	0.9132	0.0172	0.0588
No Finding	0.1994	0.6362	0.6029	0.6489	0.6239
Atelectasis	0.1551	0.2449	0.8240	0.1890	0.3478
Edema	0.0204	0.0476	0.9049	0.0286	0.1429
Effusion	0.1093	0.2317	0.7634	0.1744	0.3448
Consolidation	0.1101	0.1690	0.8597	0.1154	0.3158
Mass	0.0512	0.1069	0.8609	0.0714	0.2121
Nodule	0.0354	0.1069	0.8609	0.0875	0.1373
Fibrosis	0.0310	0.0526	0.9572	0.0476	0.0588
Pneumonia	-0.0151	0.0000	0.9560	0.0000	0.0000
Hernia	0.0000	0.0000	0.9929	0.0000	0.0000

Tableau 8 – Résultats du réseau de neurones multicouches, type 1

Labels	Kappa de Cohen	Mesure F1	Justesse	Précision	Rappel
Emphysema	0.0000	0.0304	0.0155	0.0155	1.0000
Infiltration	0.0000	0.0000	0.8240	0.0000	0.0000
Pleural_Thickening	0.0000	0.0000	0.9667	0.0000	0.0000
Pneumothorax	0.0000	0.0000	0.9501	0.0000	0.0000
Cardiomegaly	0.0000	0.0000	0.9798	0.0000	0.0000
No Finding	0.2340	0.6802	0.6266	0.6498	0.7137
Atelectasis	0.0000	0.0000	0.9180	0.0000	0.0000
Edema	0.0000	0.0000	0.9834	0.0000	0.0000
Effusion	0.0000	0.0000	0.8966	0.0000	0.0000
Consolidation	0.0000	0.0000	0.9548	0.0000	0.0000
Mass	0.0000	0.0000	0.9608	0.0000	0.0000
Nodule	0.0000	0.0000	0.9394	0.0000	0.0000
Fibrosis	0.0000	0.0000	0.9798	0.0000	0.0000
Pneumonia	0.0000	0.0000	0.9905	0.0000	0.0000
Hernia	nan	0.0000	1.0000	0.0000	0.0000

Tableau 9 – Résultats du réseau de neurones multicouches, type 2

Labels	Kappa de Cohen	Mesure F1	Justesse	Précision	Rappel
Emphysema	0.0000	0.0000	0.9845	0.0000	0.0000
Infiltration	0.0000	0.0000	0.8240	0.0000	0.0000
Pleural_Thickening	0.0000	0.0000	0.9667	0.0000	0.0000
Pneumothorax	0.0000	0.0000	0.9501	0.0000	0.0000
Cardiomegaly	0.0000	0.0000	0.9798	0.0000	0.0000
No Finding	-0.0065	0.7117	0.5529	0.5550	0.9915
Atelectasis	0.0000	0.0000	0.9180	0.0000	0.0000
Edema	0.0000	0.0000	0.9834	0.0000	0.0000
Effusion	0.0000	0.0000	0.8966	0.0000	0.0000
Consolidation	0.0000	0.0000	0.9548	0.0000	0.0000
Mass	0.0000	0.0000	0.9608	0.0000	0.0000
Nodule	0.0000	0.0000	0.9394	0.0000	0.0000
Fibrosis	0.0000	0.0000	0.9798	0.0000	0.0000
Pneumonia	0.0000	0.0000	0.9905	0.0000	0.0000
Hernia	nan	0.0000	1.0000	0.0000	0.0000

Tableau 10 – Résultats de la méthode à noyaux, type 1

Labels	Kappa de Cohen	Mesure F1	Justesse	Précision	Rappel
Emphysema	0.0000	0.0000	0.9845	0.0000	0.0000
Infiltration	0.0000	0.0000	0.8240	0.0000	0.0000
Pleural_Thickening	0.0000	0.0000	0.9667	0.0000	0.0000
Pneumothorax	0.0000	0.0000	0.9501	0.0000	0.0000
Cardiomegaly	0.0000	0.0000	0.9798	0.0000	0.0000
No Finding	0.2708	0.6952	0.6445	0.6647	0.7286
Atelectasis	0.0000	0.0000	0.9180	0.0000	0.0000
Edema	0.0000	0.0000	0.9834	0.0000	0.0000
Effusion	0.0000	0.0000	0.8966	0.0000	0.0000
Consolidation	0.0000	0.0000	0.9548	0.0000	0.0000
Mass	0.0000	0.0000	0.9608	0.0000	0.0000
Nodule	0.0000	0.0000	0.9394	0.0000	0.0000
Fibrosis	0.0000	0.0000	0.9798	0.0000	0.0000
Pneumonia	0.0000	0.0000	0.9905	0.0000	0.0000
Hernia	nan	0.0000	1.0000	0.0000	0.0000

Tableau 11 – Résultats de la méthode à noyaux, type 2

Labels	Kappa de Cohen	Mesure F1	Justesse	Précision	Rappel
Emphysema	0.0000	0.0000	0.9845	0.0000	0.0000
Infiltration	0.1028	0.1420	0.8276	0.5714	0.0811
Pleural_Thickening	0.0000	0.0000	0.9667	0.0000	0.0000
Pneumothorax	0.0000	0.0000	0.9501	0.0000	0.0000
Cardiomegaly	0.0000	0.0000	0.9798	0.0000	0.0000
No Finding	0.2708	0.6952	0.6445	0.6647	0.7286
Atelectasis	0.0000	0.0000	0.9180	0.0000	0.0000
Edema	0.0000	0.0000	0.9834	0.0000	0.0000
Effusion	-0.0069	0.0000	0.8930	0.0000	0.0000
Consolidation	0.0000	0.0000	0.9548	0.0000	0.0000
Mass	0.0000	0.0000	0.9608	0.0000	0.0000
Nodule	0.0000	0.0000	0.9394	0.0000	0.0000
Fibrosis	0.0000	0.0000	0.9798	0.0000	0.0000
Pneumonia	0.0000	0.0000	0.9905	0.0000	0.0000
Hernia	nan	0.0000	1.0000	0.0000	0.0000

Tableau 12 – Résultats de la machine à vecteurs de support, type 1

Labels	Kappa de Cohen	Mesure F1	Justesse	Précision	Rappel
Emphysema	-0.0231	0.0000	0.9429	0.0000	0.0000
Infiltration	0.0292	0.1725	0.7491	0.2056	0.1486
Pleural_Thickening	0.0167	0.0563	0.9203	0.0465	0.0714
Pneumothorax	0.0349	0.0935	0.8847	0.0769	0.1190
Cardiomegaly	0.0445	0.0714	0.9382	0.0513	0.1176
No Finding	0.1730	0.6400	0.5933	0.6307	0.6496
Atelectasis	-0.0068	0.0925	0.8133	0.0769	0.1159
Edema	0.0164	0.0400	0.9429	0.0278	0.0714
Effusion	0.0894	0.1970	0.8062	0.1724	0.2299
Consolidation	0.0787	0.1321	0.8906	0.1029	0.1842
Mass	0.0568	0.1135	0.8514	0.0741	0.2424
Nodule	0.0189	0.1000	0.8288	0.0734	0.1569
Fibrosis	-0.0186	0.0000	0.9631	0.0000	0.0000
Pneumonia	-0.0096	0.0000	0.9810	0.0000	0.0000
Hernia	0.0000	0.0000	0.9976	0.0000	0.0000

Tableau 13 – Résultats de la machine à vecteurs de support, type 2

Labels	Kappa de Cohen	Mesure F1	Justesse	Précision	Rappel
Emphysema	-0.0223	0.0000	0.9477	0.0000	0.0000
Infiltration	-0.0210	0.2161	0.5945	0.1638	0.3176
Pleural_Thickening	-0.0374	0.0000	0.9275	0.0000	0.0000
Pneumothorax	0.0477	0.1020	0.8954	0.0893	0.1190
Cardiomegaly	0.0226	0.0465	0.9512	0.0385	0.0588
No Finding	-0.0168	0.4851	0.4851	0.5469	0.4359
Atelectasis	0.0547	0.1546	0.8050	0.1200	0.2174
Edema	0.0462	0.0702	0.9370	0.0465	0.1429
Effusion	0.0734	0.1967	0.7669	0.1529	0.2759
Consolidation	0.0465	0.0941	0.9084	0.0851	0.1053
Mass	-0.0264	0.0325	0.8585	0.0222	0.0606
Nodule	0.0124	0.0876	0.8514	0.0698	0.1176
Fibrosis	-0.0193	0.0000	0.9620	0.0000	0.0000
Pneumonia	-0.0096	0.0000	0.9810	0.0000	0.0000
Hernia	0.0000	0.0000	0.9976	0.0000	0.0000

Tableau 14 – Résultats de la forêt d'arbres décisionnels, type 1

Labels	Kappa de Cohen	Mesure F1	Justesse	Précision	Rappel
Emphysema	0.0000	0.0000	0.9845	0.0000	0.0000
Infiltration	0.1611	0.2712	0.7955	0.3636	0.2162
Pleural_Thickening	0.0000	0.0000	0.9667	0.0000	0.0000
Pneumothorax	0.0296	0.0417	0.9453	0.1667	0.0238
Cardiomegaly	-0.0093	0.0000	0.9738	0.0000	0.0000
No Finding	0.2736	0.6908	0.6445	0.6693	0.7137
Atelectasis	0.0572	0.1121	0.8870	0.1579	0.0870
Edema	0.0962	0.1143	0.9631	0.0952	0.1429
Effusion	0.1849	0.2550	0.8680	0.3065	0.2184
Consolidation	0.0287	0.0635	0.9298	0.0800	0.0526
Mass	0.0000	0.0000	0.9608	0.0000	0.0000
Nodule	0.0000	0.0000	0.9394	0.0000	0.0000
Fibrosis	0.0000	0.0000	0.9798	0.0000	0.0000
Pneumonia	0.0000	0.0000	0.9905	0.0000	0.0000
Hernia	nan	0.0000	1.0000	0.0000	0.0000

Tableau 15 – Résultats de la forêt d'arbres décisionnels, type 2

Labels	Kappa de Cohen	Mesure F1	Justesse	Précision	Rappel
Emphysema	0.0000	0.0000	0.9845	0.0000	0.0000
Infiltration	0.1673	0.2985	0.7765	0.3333	0.2703
Pleural_Thickening	0.0000	0.0000	0.9667	0.0000	0.0000
Pneumothorax	0.0351	0.0435	0.9477	0.2500	0.0238
Cardiomegaly	0.0000	0.0000	0.9798	0.0000	0.0000
No Finding	0.2927	0.6997	0.6540	0.6766	0.7244
Atelectasis	0.0526	0.0889	0.9025	0.1905	0.0580
Edema	0.1283	0.1429	0.9715	0.1429	0.1429
Effusion	0.1203	0.1931	0.8609	0.2414	0.1609
Consolidation	-0.0087	0.0000	0.9501	0.0000	0.0000
Mass	0.0000	0.0000	0.9608	0.0000	0.0000
Nodule	0.0000	0.0000	0.9394	0.0000	0.0000
Fibrosis	0.0000	0.0000	0.9798	0.0000	0.0000
Pneumonia	0.0000	0.0000	0.9905	0.0000	0.0000
Hernia	nan	0.0000	1.0000	0.0000	0.0000

Tableau 16 – Résultats du vote majoritaire, type 1

Labels	Kappa de Cohen	Mesure F1	Justesse	Précision	Rappel
Emphysema	-0.0163	0.0000	0.9679	0.0000	0.0000
Infiltration	0.0872	0.2374	0.7479	0.2538	0.2230
Pleural_Thickening	0.0051	0.0370	0.9382	0.0385	0.0357
Pneumothorax	0.0145	0.0556	0.9191	0.0667	0.0476
Cardiomegaly	0.0241	0.0476	0.9524	0.0400	0.0588
No Finding	0.2385	0.6968	0.6326	0.6443	0.7585
Atelectasis	0.0468	0.1148	0.8716	0.1321	0.1014
Edema	0.0290	0.0500	0.9548	0.0385	0.0714
Effusion	0.0735	0.1705	0.8264	0.1685	0.1724
Consolidation	0.0354	0.0789	0.9168	0.0789	0.0789
Mass	0.0532	0.0976	0.9120	0.0816	0.1212
Nodule	0.0032	0.0612	0.8906	0.0638	0.0588
Fibrosis	-0.0061	0.0000	0.9762	0.0000	0.0000
Pneumonia	-0.0111	0.0000	0.9774	0.0000	0.0000
Hernia	0.0000	0.0000	0.9988	0.0000	0.0000

Tableau 17 – Résultats du vote majoritaire, type 2

Labels	**Kappa de Cohen**	**Mesure F1**	**Justesse**	**Précision**	**Rappel**
Emphysema	-0.0099	0.0000	0.9774	0.0000	0.0000
Infiltration	0.0522	0.1818	0.7646	0.2340	0.1486
Pleural_Thickening	-0.0178	0.0000	0.9548	0.0000	0.0000
Pneumothorax	0.0272	0.0615	0.9275	0.0870	0.0476
Cardiomegaly	-0.0170	0.0000	0.9655	0.0000	0.0000
No Finding	0.2249	0.6846	0.6243	0.6423	0.7329
Atelectasis	0.0872	0.1563	0.8716	0.1695	0.1449
Edema	0.0557	0.0714	0.9691	0.0714	0.0714
Effusion	0.1390	0.2286	0.8395	0.2273	0.2299
Consolidation	0.0810	0.1176	0.9287	0.1333	0.1053
Mass	0.0601	0.0952	0.9322	0.1000	0.0909
Nodule	0.0028	0.0303	0.9239	0.0667	0.0196
Fibrosis	0.0000	0.0000	0.9798	0.0000	0.0000
Pneumonia	-0.0052	0.0000	0.9869	0.0000	0.0000
Hernia	0.0000	0.0000	0.9988	0.0000	0.0000

6 Références

[1] S. M. M. et al., "International evaluation of an ai system for breast cancer screening," *Nature*, January 2020.

[2] "Random sample of nih chest x-ray dataset." `https://www.kaggle.com/nih-chest-xrays/sample`. Consulté le 2020-02-11.

[3] "Nih chest x-rays." `https://www.kaggle.com/nih-chest-xrays/data`. Consulté le 2020-02-11.

[4] X. W. et al., "Chestx-ray8 : Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," *IEEE*, 2017.

[5] "Introducing gitflow." `https://datasift.github.io/gitflow/IntroducingGitFlow.html`. Consulté le 2020-04-05.

[6] "Zenhub for github." `https://chrome.google.com/webstore/detail/zenhub-for-github/ogcgkffhplmphkaahpmffcafajaocjbd`. Consulté le 2020-02-11.

[7] P.-M. Jodoin, "Ift603-712 - techniques d'apprentissage." Notes de cours et vidéos.

[8] "Machine learning tips and tricks cheatsheet." `https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-machine-learning-tips-and-tricks`. Consulté le 2020-04-08.

[9] H. E. V. et al., "Shared perception for autonomous systems," *Lincoln Laboratory Journal*, 2017.