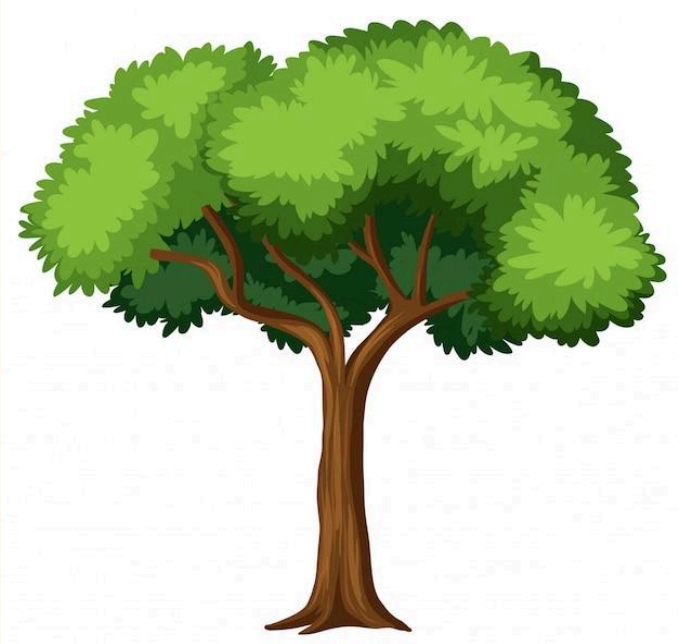
STG_10 T-DEV-810
Jérôme Schlub, Khalil Mazlani, Sacha Mutschler, Daniel Viter

MODÈLE
ARBRE DE DÉCISION

—

Jérôme Schlub



INTRODUCTION

L'objectif du projet est d'utiliser plusieurs modèles capables d'analyser des images radiographiques pour identifier les signes de pneumonie ou non. Pour ce faire, nous avons accès à trois ensembles de données fournis **"datasets"**, qui seront utilisés pour l'entraînement du modèle, la validation et les tests des modèles d'apprentissage automatique.

Modèle choisit “Arbre de décision (DecisionTreeClassifier)”

Dans notre groupe nous avons voulu partir chacun sur un modèle différent, ce qui permet à chacun de prendre la main sur un modèle en particulier, comprendre son fonctionnement, sa logique et de par la suite faire des comparaisons de nos métriques pour évaluer et choisir le modèle le plus performant.

Début de la logique appliqué pour le modèle “DecisionTreeClassifier”

- ## Chargements des images

La première étape consiste à charger l’ensemble du **“datasets”** et d’afficher chacun des trois ensembles dans la console :

```
Train data shape: (2680, 128, 128), Train labels shape: (2680,)

Val data shape: (16, 128, 128), Val labels shape: (16,)

Test data shape: (624, 128, 128), test labels shape: (624,)
```

- ## Prétraitement des données

Ensuite je normalise les images, le fait de mettre les images à une échelle uniforme permettra de converger plus rapidement et de manière plus stable les algorithmes utilisés.

Le fait de modifier le vecteur des images va permettre également de simplifier le traitement des données par les algorithmes.

Quand j’avance, j’aime visualiser les valeurs dans la console, c’est pourquoi je **“print”** à nouveau le **“datasets”** pour m’assurer que tout a bien été traité.

- ## Visualisation des images

Pour prendre en main la librairie **“matplotlib”** j’ai voulu afficher certaines des images du **“datasets”** dans un tableau.

Equilibrage du dataset train

Pour que mon modèle ne soit pas trop biaisé, j'équilibre le **“dataset train”** entre les deux classes, pour se faire je supprime des images en trop dans le dossier **“PNEUMONIA”** pour qu’il y ait un équilibre entre les deux classes.

Utilisation du modèle “Arbre de décision (DecisionTreeClassifier)” et premier entrainement sur le “datasets”

Dans un premier temps, j’utilise le modèle **“DecisionTreeClassifier”** de **“Scikit-Learn”**, qui est un modèle binaire. Son raisonnement se base sur des opérations booléennes, évaluant chaque décision par **“true”** ou **“false”**.

Dans ma compréhension,j’ai l’impression que l’on utilise les modèles de la même manière :

- fit -> pour entraîner le modèle
- score -> pour évaluer le modèle
- predict -> pour utiliser le modèle

C’est surtout au niveau de la compréhension et de l’utilisation des hyperparamètres que les choses diffèrent.

- ## Evaluer mon modèle

Pour évaluer mon modèle je vais utiliser **“classification_report”**, **“confusion_matrix”**, **“accuracy_score”**, de **“sklearn.metrics”**. Ces métriques vont me permettre d’avoir beaucoup plus de données pour évaluer mon modèle qu’avec un classique **“score”**.

- **accuracy_score** : Permet de calculer le pourcentage de prédictions exactes par rapport au total.
- **confusion_matrix** : Permet d'afficher un tableau comparant les vraies valeurs aux valeurs prédites.
- **classification_report** : Fournit des métriques plus précises pour analyser le modèle
    - **Précision :** Indique le pourcentage de prédictions correctes parmi les prédictions positives faites par le modèle. Une haute précision signifie que peu de faux positifs sont produits.

- **Rappel (Recall) :** Indique le pourcentage de vrais positifs correctement identifiés parmi tous les positifs réels. Un haut rappel signifie que le modèle capture la majorité des vrais positifs.
- **Score F1 :** La moyenne harmonique de la précision et du rappel. Il combine les deux mesures en un seul score pour donner une idée globale de la performance du modèle, surtout utile lorsque les classes sont déséquilibrées.

Optimisation des hyperparamètres de manière progressive

Pour optimiser les hyperparamètres je pars sur l'utilisation de **"GridSearchCV"**, cet outil est surpuissant, il va me permettre de lui fournir un paramètre ou un dictionnaire de paramètres, d'entraîner le modèle avec chacune des combinaisons, d'évaluer les performances de chaque modèle et ensuite sélectionner le modèle ayant eu le meilleur résultat.

L'optimisation des hyperparamètres va permettre à mon modèle d'être potentiellement plus performant, de faire moins **"d'overfitting (surapprentissage)"** ou **"underfitting (sous-apprentissage)"**.

Selon mes lectures trouvées sur internet un hyperparamètre qui peut être très utile pour le modèle **"DecisionTreeClassifier"** est **"max_depth"**.

Le paramètre **"max_depth"** contrôle la profondeur maximale d'un arbre de décision, aidant à équilibrer entre un modèle trop complexe **"overfitting"** et un modèle trop simple **"underfitting"**.

Pour optimiser les hyperparamètres de manière progressive je crée un dictionnaire **"Python"** ou je vais passer plusieurs valeurs à **"max_depth"**.

Dans les paramètres de **"GridSearchCV"** je passe le modèle, le dictionnaire de mes hyperparamètres, une valeur à la **"cross_validation"**, je pars sur une valeur de 5 qui a l'air d'être très souvent utilisée. Les derniers paramètres vont me permettre de spécifier les cœurs du **CPU** à utiliser pour les calculs et afficher des détails lors de la progression, chose qui peut-être très utile pour surveiller l'exécution du modèle.

Je tiens à préciser que j'ai utilisé plusieurs méthodes de **"cross_validation"**, le classique qui est **"K-fold Cross-Validation",** et aussi **"StratifiedKFold"**.

- **K-fold Cross-Validation :** On divise le datasets de manière équitable, le dernier sous-ensemble est utilisé pour la validation.
- **StratifiedKFold** : Même logique que la **"cross-validation"** précédante, mais celle-ci équilibre également les **"classes"** dans chacun des **"folds"**.

Une fois la recherche des meilleurs hyperparamètres effectuée par **"GridSearchCV"** on peut **"print"** la méthode suivante **"best_params_"** pour connaître la meilleure combinaison

d’hyperparamètres. On peut utiliser également **“best_score”** pour connaître le meilleur pourcentage de prédictions correctes.

Ensuite je pars sur la même logique pour la suite, c’est à dire que je passe en paramètre de mon modèle le meilleur hyperparamètre trouvé par **“GridSearchCV”** et je crée un nouveau dictionnaire Python pour le passer à **“GridSearchCV”**.

L’utilisation de **“best_params_”** et de **“best_score”** va me faciliter la vie pour choisir à nouveau le meilleur hyperparamètre pour ce second dictionnaire **Python**.

Je progresse ainsi de suite jusqu’à optimiser 4 hyperparamètres à mon modèle.

Les hyperparamètres que j’utilise :

- **max_depth** : déjà expliqué
- **criterion** : Le paramètre **“criterion”** détermine la fonction utilisée pour mesurer la qualité d'une division dans un arbre de décision, influençant ainsi la manière dont l'arbre choisit les points de séparation.
- **min_samples_split** : Le paramètre **“min_samples_split”** spécifie le nombre minimum d'échantillons requis pour diviser un nœud dans un arbre de décision, empêchant ainsi l'arbre de se diviser trop souvent et aidant à contrôler la complexité du modèle.
- **min_samples_leaf** : Le paramètre **“min_samples_leaf”** spécifie le nombre minimum d'échantillons qu'une feuille doit contenir dans un arbre de décision, aidant à prévenir la création de feuilles avec trop peu de données et à contrôler la complexité du modèle.

Une fois que j’ai optimisé les meilleurs hyperparamètres pour mon modèle je le stocke dans une variable pour me permettre de l’utiliser par la suite et utiliser la méthode **“predict”** pour la prédiction des étiquettes (**labels**) sur l’ensemble du **“dataset test”**.

Là aussi j’utilise **“classification_report”**, **“confusion_matrix”**, **“accuracy_score”**, de **“sklearn.metrics”** pour évaluer les métriques de mon modèle.

Optimisation des hyperparamètres de manière globale

Pour l'optimisation des hyperparamètres de manière globale, c'est un peu moins complexe que de le faire d'une manière progressive.

Dans mes recherches et mes lectures sur internet que j'ai pu lire, il suggère généralement de passer entre 5 à 10 hyperparamètres pour améliorer son modèle. Dans mon cas, j'ai passé au total 4 hyperparamètres, pourquoi ? Parce que je souhaite comprendre ce que je fais, comment on utilise et fonctionne les hyperparamètres, en mettre trop d'un coup aurait l'effet inverse pour ma part, je préfère privilégier la compréhension de ce que je fais à des taux de résultats élevés.

Je créer un dictionnaire **Python** en passant tous les hyperparamètres :

```
param_grid = {
    'max_depth': [None, 10, 20, 30, 40, 50],
    'criterion': ['gini', 'entropy'],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4, 6, 8, 10]
}
```

Finalement, je passe les 3 premiers hyperparamètres du dictionnaire, passer les 4 directement me demanderait énormément de temps :

Fitting 5 folds for each of 216 candidates, totalling 1080 fits

[CV] END criterion=gini, max_depth=None, min_samples_leaf=1, min_samples_split=2; total time= 23.9s

[CV] END criterion=gini, max_depth=None, min_samples_leaf=1, min_samples_split=2; total time= 25.3s

[CV] END criterion=gini, max_depth=None, min_samples_leaf=1, min_samples_split=2; total time= 21.5s

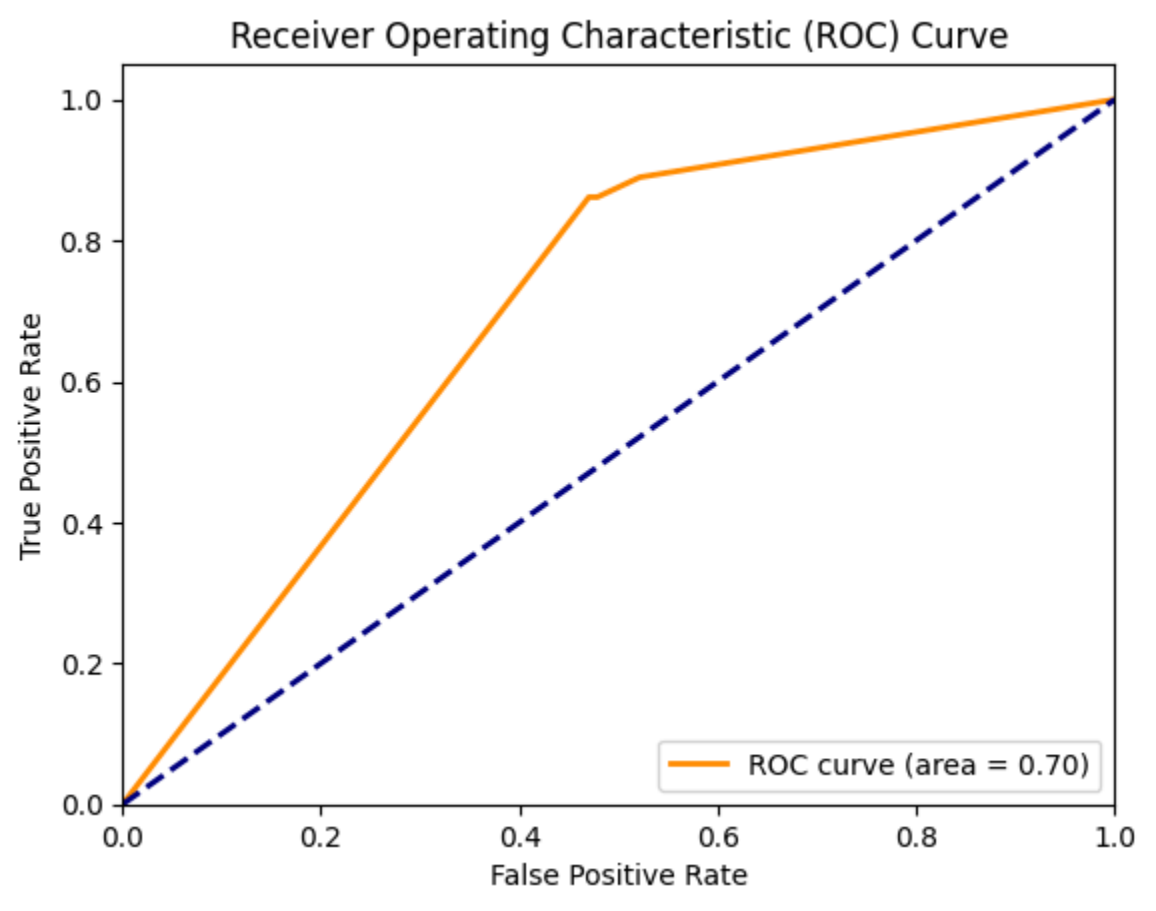
Il me faudrait plus ou moins 7 heures et 30 minutes (moyenne du temps d'un fold = 25 secondes * le nombre d'itérations soit 1080 fits), je passe du coup sur une logique un peu différente. Je

sépare le **“dataset” “train”** à l’aide de la fonction **“train_test_split”**, de cette manière, je fais en sorte de prendre seulement 30 % de ce **“dataset”** pour entraîner mon modèle, ce qui va me permettre de passer les 4 hyperparamètres d’un coup et d'entraîner mon modèle directement avec toutes les combinaisons voulues. J’ai à ce moment précis la combinaison des meilleurs hyperparamètres que je peux donner à mon modèle.

Une fois en possession du meilleur modèle, je l’utilise à l’aide de la fonction **“predict”** pour l’évaluer sur le données de **“test”**.

La courbe de ROC

J'apprécie la courbe de ROC pour sa simplicité de compréhension, elle va me permettre très rapidement de détecter si j'ai un taux de positif élevé et un taux de faux positif faible, en fait plus la courbe se rapproche du coin supérieur gauche, meilleur est le modèle. Je vais vous montrer la courbe de ROC pour mon modèle ayant la meilleure optimisation à l'aide de mes hyperparamètres.



On parlera un peu plus tard de ce résultat, je tiens à souligner que j'ai tracé la courbe de ROC pour chaque étape de l'optimisation de mes hyperparamètres, je m'explique. A l'aide de **"GridSearchCV"** j'ai pu facilement trouver la meilleure optimisation, mais dans ma quête de compréhension et de performance j'ai voulu visualiser la courbe de **ROC** pour chaque étape. Que j'utilise la **"cross-validation"** de base **"K-fold Cross-Validation"** ou **"StratifiedKFold"** j'obtiens toujours les mêmes résultats lors de la recherche des meilleurs hyperparamètres à l'aide de **"GridSearchCV"**.

- Pour **“max_depth”** = 'max_depth': 10
    - Quand je trace la **courbe de ROC**, je passe uniquement **“max_depth”** à 10, j'obtiens une **“area”** de 0.68

- Pour **“max_depth”** et **“criterion”** = 'criterion': 'gini', 'max_depth': 10
    - Quand je trace **la courbe de ROC**, je passe **“max_depth”** à 10 et **“criterion”** à **“gini”,** j'obtiens également une **“area”** de 0.68

- Pour **“max_depth”** et **“criterion”** et **“min_samples_split”** = 'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 5
    - Quand je trace **la courbe de ROC**, je passe **“max_depth”** à 10, **“criterion”** à **“gini”** et **“min_samples_split”** à 5, j'obtiens une **“area”** de 0.69

- Pour **“max_depth”** et **“criterion”** et **“min_samples_split”** et **“min_samples_leaf”** : 'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5
    - Quand je trace **la courbe de ROC**, je passe **“max_depth”** à 10, **“criterion”** à **“gini”, “min_samples_split”** à 5 et **“min_samples_leaf”** à 5, j'obtiens une **“area”** de 0.70

De prime abord ce que je constate c’est que malgré l’optimisation des hyperparamètres je n’obtiens pas un score beaucoup plus élevé, ma première réflexion est de me dire que mon modèle n’est sans doute pas approprié.

Pour en revenir à l’annonce de la courbe de ROC sur le meilleur modèle, voici ma synthèse :

- La courbe en pointillé représente une performance aléatoire, ma courbe étant au dessus on peut déjà se dire que mon modèle surpasse le hasard, ce qui est tout de même un bon premier point.
- Cela étant dit, on peut partir sur une analyse un peu plus profonde. Mon modèle obtient un taux de vrai positif assez élevé, malheureusement cela s’accompagne également avec

un taux de faux positif notable. Il y a pour moi un compromis entre la capacité à identifier correctement les vrais positifs et le nombre de faux positifs, ce qui fait que mon modèle n'est pas le plus performant.

Première conclusion

Si je fais une conclusion brute, sans trop prendre en compte les données analytiques, je peux conclure par :

- Rajouter d'autres hyperparamètres ?
    - Avant cela il faut tout d'abord que je me documente et me renseigne énormément pour comprendre en profondeur l'utilisation des hyperparamètres
- Tester d'autres combinaisons d'hyperparamètres ?
    - Je ne suis pas sûr d'avoir le temps
- Peut-être que mon modèle n'est pas adapté à la situation, je devrais tester d'autres algorithmes de classification pour les comparer

Les rapports lors des optimisations des meilleurs hyperparamètres à l'aide de GridSearchCV

Rapport arbre de décision avec hyperparamètre (max_depth)

Fitting 5 folds for each of 6 candidates, totalling 30 fits

[CV] END .....max_depth=None; total time= 24.9s

…

Le meilleur score trouvé: 0.8611940298507463

Meilleurs paramètres trouvé: {'max_depth': 10}

—

```
Résultat avec le modèle et le paramètre (max_depth)

Test - Accuracy : 0.780448717948718
Test - Confusion Matrix :
[[133 101]
 [ 36 354]]
Test - Classification Report :
              precision    recall  f1-score   support

           0       0.79      0.57      0.66       234
           1       0.78      0.91      0.84       390

    accuracy                           0.78       624
   macro avg       0.78      0.74      0.75       624
weighted avg       0.78      0.78      0.77       624
```

Rapport arbre de décision avec hyperparamètre (max_depth) et la cross-validation (StratifiedKFold(n_splits=5))

Fitting 5 folds for each of 6 candidates, totalling 30 fits

[CV] END .....max_depth=None; total time= 24.3s

...

Le meilleur score trouvé: 0.8611940298507463

Meilleurs paramètres trouvé: {'max_depth': 10}

—

```
Résultat avec le modèle et le paramètre (max_depth) et la cross-validation (StratifiedKFold(n_splits=5))

Test - Accuracy : 0.780448717948718
Test - Confusion Matrix :
[[133 101]
 [ 36 354]]
Test - Classification Report :
              precision    recall  f1-score   support

           0       0.79      0.57      0.66       234
           1       0.78      0.91      0.84       390

    accuracy                           0.78       624
   macro avg       0.78      0.74      0.75       624
weighted avg       0.78      0.78      0.77       624
```

Rapport arbre de décision avec hyperparamètres (max_depth, criterion)

Fitting 5 folds for each of 12 candidates, totalling 60 fits

[CV] END .....criterion=gini, max_depth=None; total time= 36.0s

...

Le meilleur score trouvé: 0.8611940298507463

Meilleurs paramètres trouvé: {'criterion': 'gini', 'max_depth': 10}

—

```
Résultat avec le modèle et les paramètres (max_depth, criterion)

Test - Accuracy : 0.780448717948718
Test - Confusion Matrix :
[[133 101]
 [ 36 354]]
Test - Classification Report :
              precision    recall  f1-score   support

           0       0.79      0.57      0.66       234
           1       0.78      0.91      0.84       390

    accuracy                           0.78       624
   macro avg       0.78      0.74      0.75       624
weighted avg       0.78      0.78      0.77       624
```

Rapport arbre de décision avec hyperparamètres (max_depth, criterion) et la cross-validation (StratifiedKFold(n_splits=5))

Fitting 5 folds for each of 12 candidates, totalling 60 fits

[CV] END .....criterion=gini, max_depth=None; total time= 25.0s

...

Le meilleur score trouvé: 0.8611940298507463

Meilleurs paramètres trouvé: {'criterion': 'gini', 'max_depth': 10}

—

```
Résultat avec le modèle et les paramètres (max_depth, criterion) et la cross-validation (StratifiedKFold(n_splits=5))

Test - Accuracy : 0.780448717948718
Test - Confusion Matrix :
[[133 101]
 [ 36 354]]
Test - Classification Report :
              precision    recall  f1-score   support

           0       0.79      0.57      0.66       234
           1       0.78      0.91      0.84       390

    accuracy                           0.78       624
   macro avg       0.78      0.74      0.75       624
weighted avg       0.78      0.78      0.77       624
```

Rapport arbre de décision avec hyperparamètres (max_depth, criterion, min_samples_split)

Fitting 5 folds for each of 36 candidates, totalling 180 fits

[CV] END criterion=gini, max_depth=None, min_samples_split=2; total time= 26.6s

...

Le meilleur score trouvé: 0.8638059701492538

Meilleurs paramètres trouvé: {'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 5}

—

```
Résultat avec le modèle et les paramètres (max_depth, criterion, min_samples_split)

Test - Accuracy : 0.780448717948718
Test - Confusion Matrix :
[[132 102]
 [ 35 355]]
Test - Classification Report :
              precision    recall  f1-score   support

           0       0.79      0.56      0.66       234
           1       0.78      0.91      0.84       390

    accuracy                           0.78       624
   macro avg       0.78      0.74      0.75       624
weighted avg       0.78      0.78      0.77       624
```

Rapport arbre de décision avec hyperparamètres (max_depth, criterion, min_samples_split) et la cross-validation (StratifiedKFold(n_splits=5))

Fitting 5 folds for each of 36 candidates, totalling 180 fits

[CV] END criterion=gini, max_depth=None, min_samples_split=2; total time= 25.7s

...

Le meilleur score trouvé: 0.8638059701492538

Meilleurs paramètres trouvé: {'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 5}

—

```
Test - Accuracy : 0.780448717948718
Test - Confusion Matrix :
[[132 102]
 [ 35 355]]
Test - Classification Report :
              precision    recall  f1-score   support

           0       0.79      0.56      0.66       234
           1       0.78      0.91      0.84       390

    accuracy                           0.78       624
   macro avg       0.78      0.74      0.75       624
weighted avg       0.78      0.78      0.77       624
```

Rapport arbre de décision avec hyperparamètres (max_depth, criterion, min_samples_split, min_samples_leaf)

Taille du dataset d'entraînement réduit : (804, 16384), (804,)

Fitting 5 folds for each of 216 candidates, totalling 1080 fits

[CV] END criterion=gini, max_depth=None, min_samples_leaf=1, min_samples_split=2; total time= 4.3s

...

Le meilleur score trouvé: 0.814642857142857

Meilleurs paramètres trouvé: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5}

—

```
Test - Accuracy : 0.7147435897435898
Test - Confusion Matrix :
[[107 127]
 [ 51 339]]
Test - Classification Report :
              precision    recall  f1-score   support

           0       0.68      0.46      0.55       234
           1       0.73      0.87      0.79       390

    accuracy                           0.71       624
   macro avg       0.70      0.66      0.67       624
weighted avg       0.71      0.71      0.70       624
```

Rapport arbre de décision avec hyperparamètres (max_depth, criterion, min_samples_split, min_samples_leaf) et la cross-validation (StratifiedKFold(n_splits=5))

Taille du dataset d'entraînement réduit : (804, 16384), (804,)

Fitting 5 folds for each of 216 candidates, totalling 1080 fits

[CV] END criterion=gini, max_depth=None, min_samples_leaf=1, min_samples_split=2; total time= 4.6s

...

Le meilleur score trouvé: 0.814642857142857

Meilleurs paramètres trouvé: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5}

—

```
Test - Accuracy : 0.7147435897435898
Test - Confusion Matrix :
[[107 127]
 [ 51 339]]
Test - Classification Report :
              precision    recall  f1-score   support

           0       0.68      0.46      0.55       234
           1       0.73      0.87      0.79       390

    accuracy                           0.71       624
   macro avg       0.70      0.66      0.67       624
weighted avg       0.71      0.71      0.70       624
```

Conclusion sur le modèle à l'aide de toutes ces métriques

Les ajustement successifs des hyperparamètres **"criterion"**, **"max_depth"**, et **"min_samples_split"** donnent des résultats très similaires, avec une **"accuracy"** de **0.78**. Une de mes premières conclusions est que finalement le modèle ne bénéficie pas beaucoup de tous ces ajustements et pourrait être proche de ses limites de performance pour la tâche que je lui donne.

Lorsque je souhaite passer le dernier hyperparamètre **"min_samples_split"** j'ajuste le **"dataset train"** à **30%** pour que la tâche effectuée par **"GridSearchCV"** soit plus rapide, en effet si je ne réduis pas le **"dataset"** la tâche quand à elle prendrait +7 heures 30 minutes pour l'optimisation globale des hyperparamètres. Ce que je constate c'est que réduire autant ce **"dataset"** entraîne une baisse significative de la performance, avec une **"accuracy"** à **0.71**. Je peux donc conclure que la quantité de données d'entraînement est importante pour la performance du modèle.

"L'accuracy" de **0.78** indique une performance correcte mais loin d'être excellente. On voit que le "**recall"** pour la classe 1, c'est-à-dire des personnes ayant une pneumonie est élevée de **0.91**. Cependant le "**recall"** pour la classe 0, donc les personnes n'ayant pas de pneumonie est faible de **0.57**, ce qui veut dire que le modèle manque beaucoup de cas sans pneumonie.

Je peux donc conclure en disant que le modèle **"DecisionTreeClassifier"** n'est pas un modèle adapté pour cette tâche.