Rapport partie modélisation:

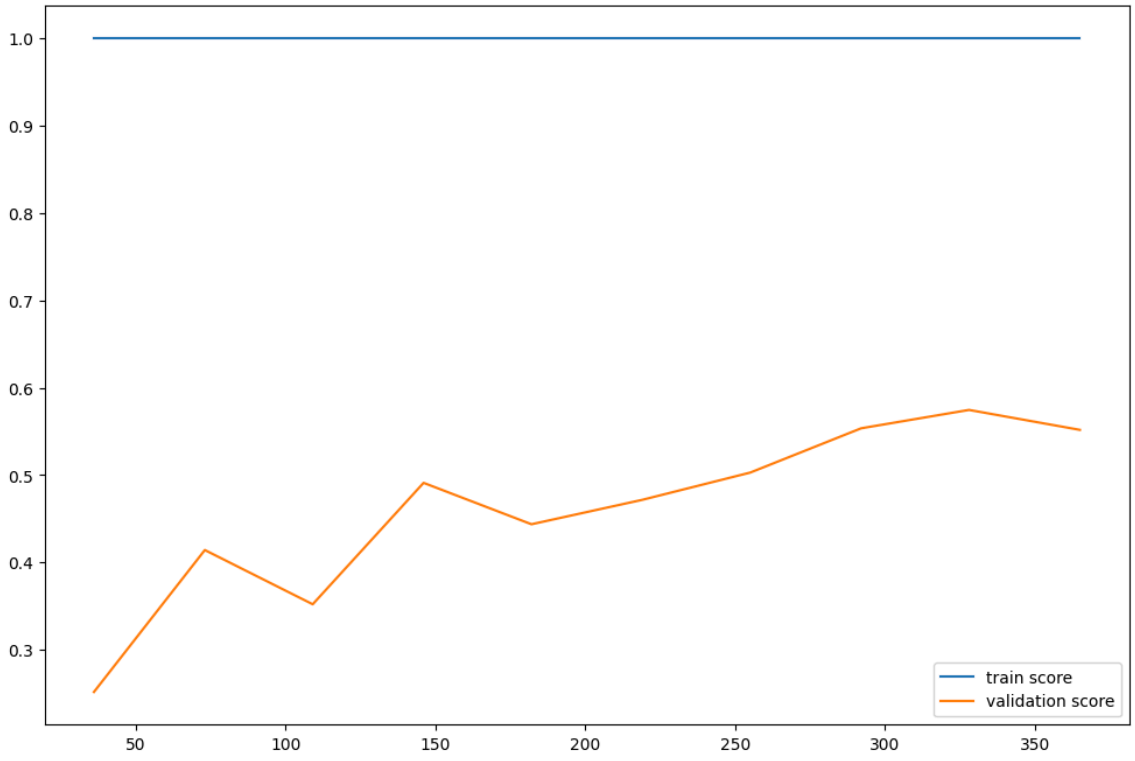
Introduction:

Bienvenue dans cette partie de notre projet dédié au machine learning, où nous explorons le potentiel des données relatives au coronavirus. Après avoir minutieusement analysé un jeu de données issu d'un ensemble de données sur le coronavirus, nous avons identifié les variables les plus pertinentes pour notre étude. Dans les parties précédentes de projet, nous avons également entrepris un prétraitement rigoureux de ce jeu de données dans le but ultime de concevoir un modèle de machine learning performant.
Maintenant, nous franchissons une nouvelle étape en nous lançant dans le développement concret d'un modèle de machine learning. Notre objectif est clair : obtenir les meilleures performances possibles. Pour cela, nous adopterons une approche méthodique.
Tout d'abord, nous entraîneront une gamme variée de modèles pour identifier celui offrant les meilleures performances initiales. Ensuite, nous nous attaquerons à l'optimisation de ce modèle en ajustant ses hyperparamètres. Nous utiliserons pour ce faire une série d'algorithmes, notamment le grid search cv.
Enfin, nous explorerons l'utilisation des courbes de précision et de rappel pour déterminer le seuil optimal, assurant ainsi une performance maximale de notre modèle de machine learning.

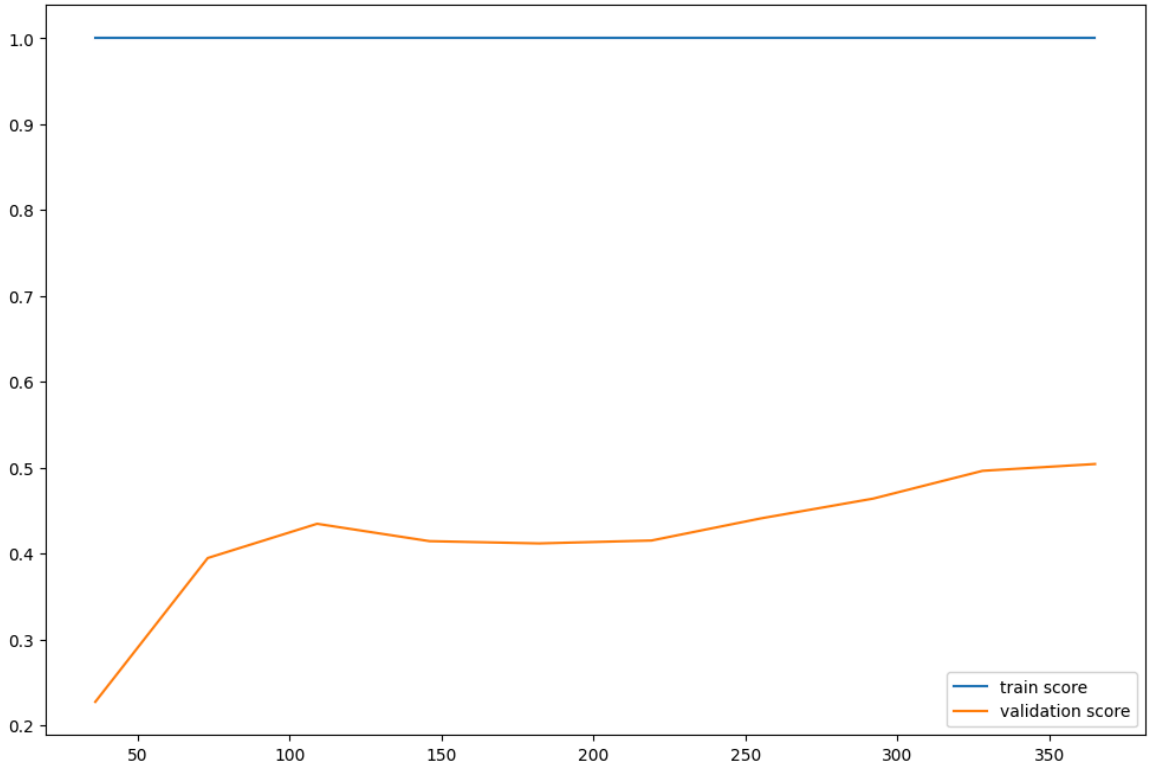
Modélisation et évaluation:
Dans notre projet, nous débutons en créant une liste variée de modèles de machine learning. Nous pouvons inclure autant de modèles que nécessaire pour notre analyse. Dans cette optique, nous envisageons d'intégrer des algorithmes de bagging qui sont random forest et decision tree, un algorithme d'ensemble tel que XG Boost, ainsi qu'un classifieur de machines à vecteurs de support (SVM). Les SVM sont particulièrement adaptés aux ensembles de données de taille moyenne à petite, ce qui les rend pertinents pour notre étude.
En plus de ces choix, nous explorerons également la régression logistique, une méthode classique pour les problèmes de classification, qui offre à la fois une interprétabilité et des performances solides.

Pour chaque combinant ces différents modèles, nous serons en mesure d'évaluer leurs performances respectives et de sélectionner celui qui répond le mieux à notre objectif d'analyse le modèle, nous mettons en place une pipeline qui inclut des étapes de prétraitement telles que la normalisation des données. Par exemple, pour le modèle SVM, nous ajoutons une étape de normalisation entre le prétraitement et le classifieur SVM lui-même, car les SVM sont sensibles à l'échelle des données.
Ensuite, nous évaluons les performances de chaque modèle en utilisant des métriques telles que le score F1 pour la classe positive. Après avoir observé les performances de chaque modèle, nous analysons également les courbes d'apprentissage pour comprendre le comportement des modèles lors de l'apprentissage et de la généralisation.

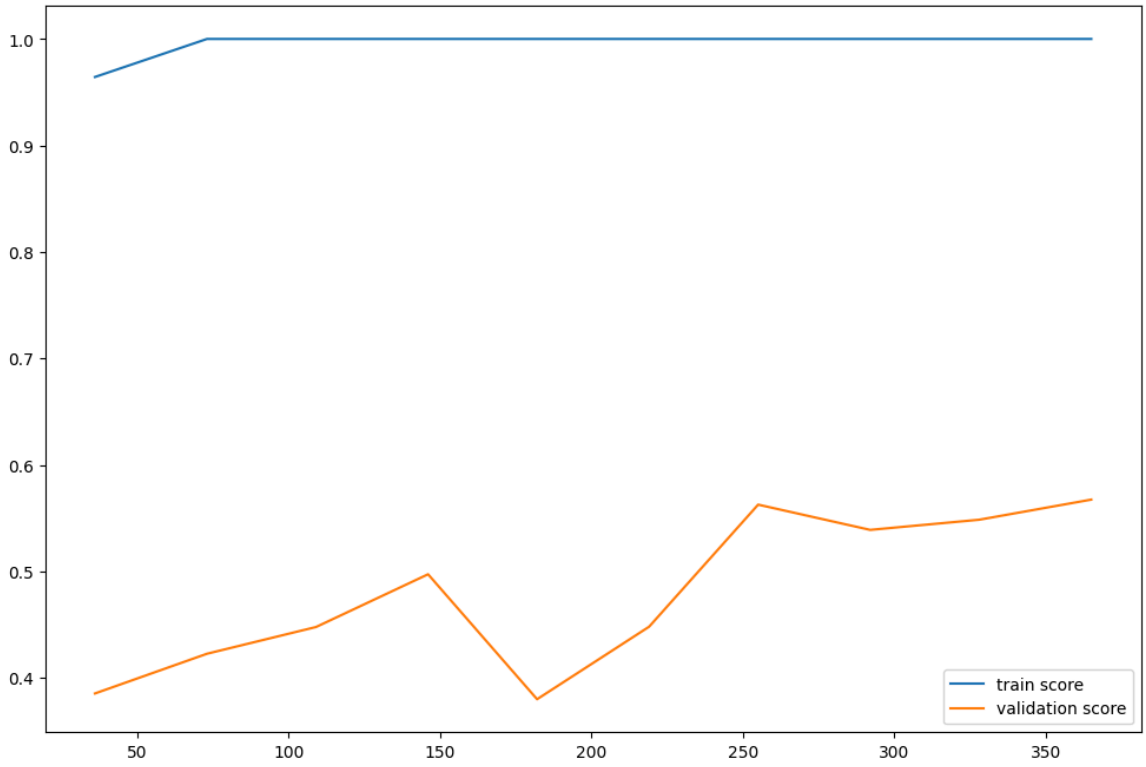
Random Forest::
- Précision : 0.56
- Recall : 0.31
- F1-score : 0.40
- Accuracy : 0.86
- temps d'exécution: 1m 52s



Decision Tree:
- Précision : 0.36
- Recall : 0.31
- F1-score : 0.33
- Accuracy : 0.82
- temps d'exécution: 1m

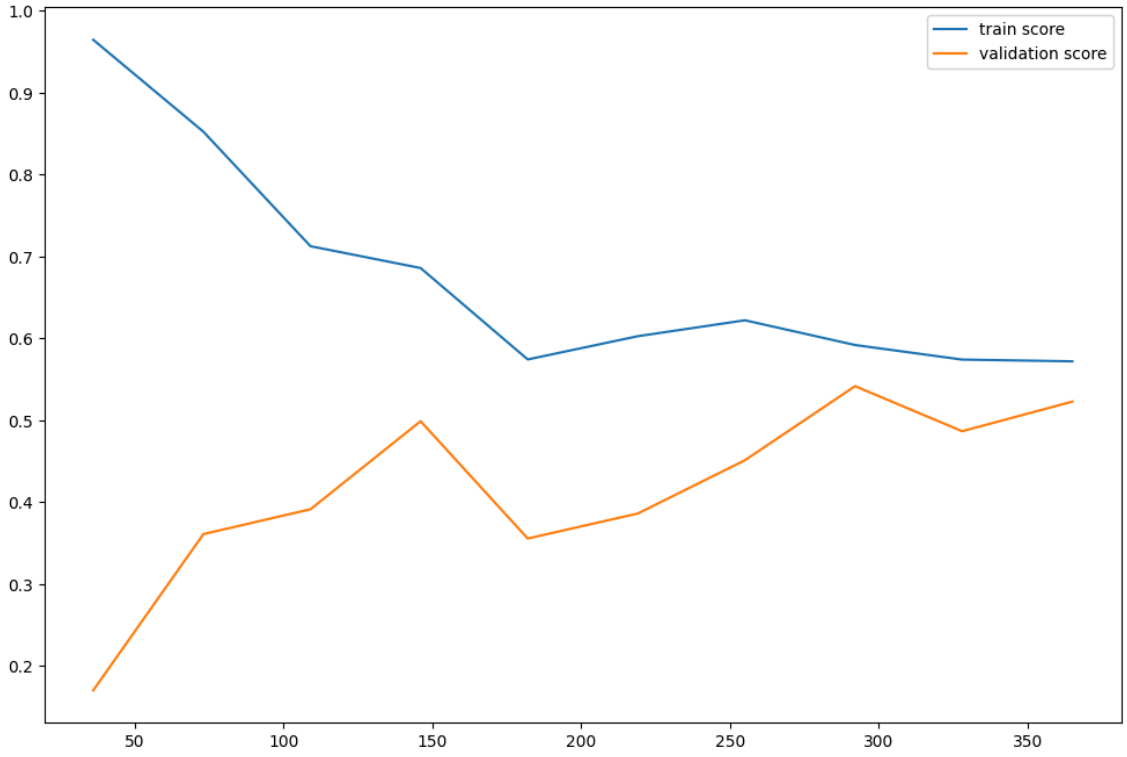


XGBoost:
- Précision : 0.64
- Recall : 0.44
- F1-score : 0.52
- Accuracy : 0.88
- temps d'exécution: 1m 48s

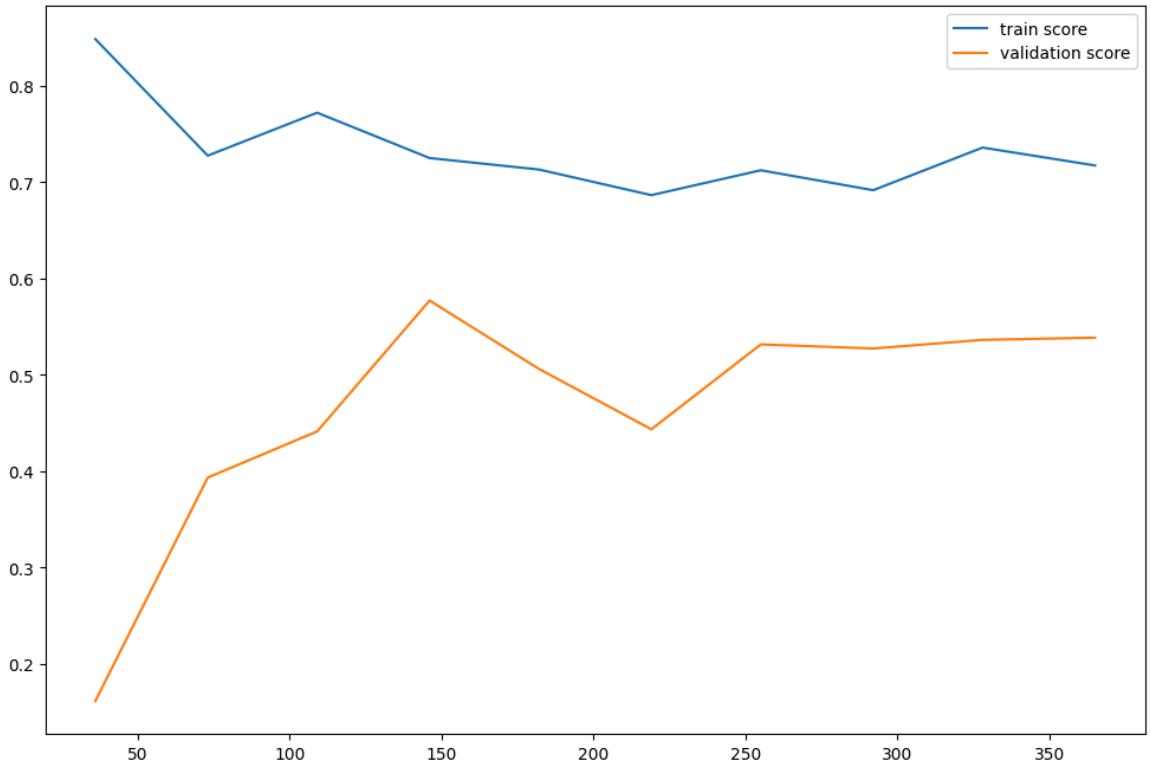


Régression logistique:

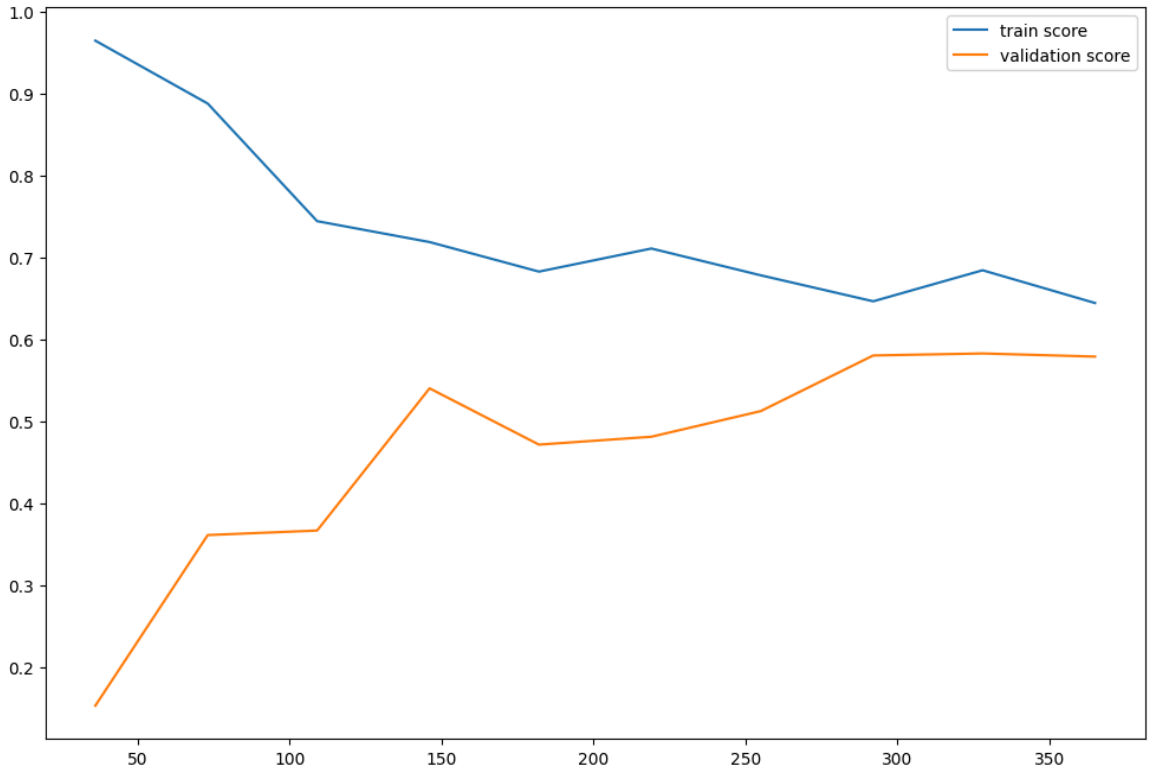
- Précision : 0.67
- Recall : 0.38
- F1-score : 0.48
- Accuracy : 0.88
- temps d'exécution: 3m 47s



KNN:
- Précision : 0.53
- Recall : 0.50
- F1-score : 0.52
- Accuracy : 0.86
- temps d'exécution: 1m 16s



SVM:
- Précision : 0.67
- Recall : 0.38
- F1-score : 0.48
- Accuracy : 0.88
- temps d'exécution: 54s



Nous constatons que le modèle SVM présente des performances compétitives, mais également la régression logistique, XGBoost, qui affichent des métriques similaires à celles des KNN et les courbes d'apprentissage et de test convergent à partir d'un certain seuil, ce qui indique que notre modèle peut généraliser et ne souffre pas de surajustement, contrairement aux autres modèles de bagging. Sur cette base, nous décidons de nous concentrer sur l'optimisation des modèles SVM, régression logistique, KNN et XGBoost en utilisant Randomized Search CV pour trouver les meilleurs hyperparamètres.
Nous excluons les modèles de bagging en raison de leur propension à ouverfit les données, ce qui est problématique étant donné la petite taille de notre ensemble de données depuis le début. De plus, la normalisation des données est difficile à réaliser pour ces modèles dans ce contexte.

L’optimisation de SVM:
Lorsqu' on a un domaine d'hyper paramètres excessivement grand, on a utilisé Randomized Search CV à la place de Grid Search CV. Ce que va faire cet optimiseur, c'est de chercher de façon aléatoire différentes combinaisons dans tout le domaine des hyperparamètres que nous avons défini.
Pour ce faire, nous commençons par ajouter quelques hyperparamètres dans notre dictionnaire. Nous ajoutons des hyperparamètres pour la sélection des variables avec le SelectKBest, dans lequel nous testons différentes valeurs pour le nombre de variables sélectionnées, allant de 4 à 100. Ensuite, nous ajoutons des hyperparamètres pour le pre-processing des données avec le Polynomial Features, où nous testons différents degrés (2, 3, 4 et5), a côté des hyperparamètres présidents concernant le svm qui sont le gamma (0.01,0.01,0.001,0.0001) et la constante de régularisation C(10,100,1000).
En utilisant Randomized Search CV, nous faisons passer notre modèle et notre dictionnaire d'hyper paramètres, ainsi que le nombre d'itérations, c'est-à-dire combien de fois l'algorithme va effectuer une recherche aléatoire avec toutes nos combinaisons d'hyper paramètres. Dans cet exemple, nous testons 40 configurations au hasard. Après avoir entraîné notre grille, nous imprimons les résultats, et nous obtenons une grande amélioration: recall = précision = 0.50, accuracy = 0.86.

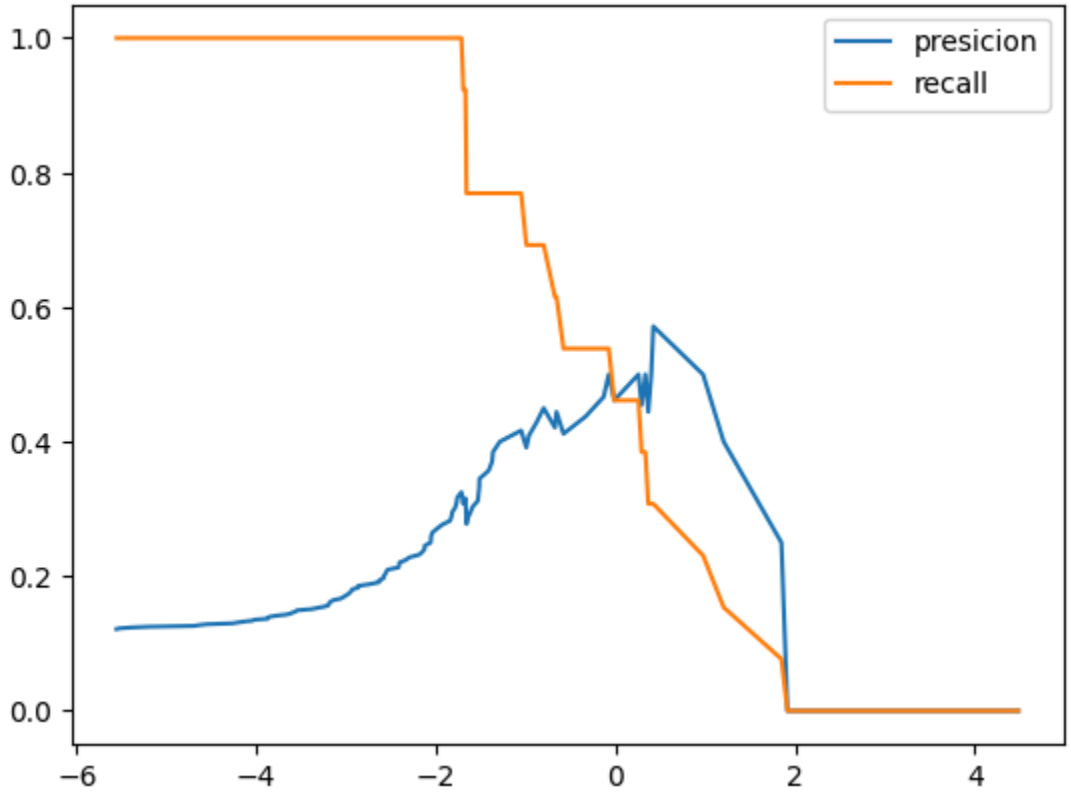
pour les autres modèles:
Pour les autres modèles, nous avons appliqué la même stratégie que celle utilisée pour le SVM, mais nous n'avons pas obtenu de bons résultats. Par conséquent, nous finalisons le modèle basé sur celui du SVM.

Finalisation de modèle

Il est très intéressant de finaliser la création de notre modèle en observant les courbes de précision et de rappel, et en définissant un seuil de décision pour notre modèle. Pour cela, nous allons utiliser la fonction **precision_recall_curve** du module metrics de sklearn. Cette fonction nous permet de visualiser la précision et le rappel de notre modèle en fonction d'un seuil de décision que nous allons définir.

La plupart des modèles que nous développons en apprentissage automatique, comme le modèle de support vectoriel que nous sommes en train de développer, utilisent une fonction de décision pour calculer la probabilité d'une prédiction. En fonction d'un seuil de décision, nous décidons si une prédiction est positive ou négative. Par exemple, si la probabilité calculée est supérieure à 0,5, nous prédisons que la classe est 1, sinon, nous prédisons que la classe est 0. Ce seuil peut être ajusté selon nos besoins une fois que le modèle est entraîné.

Pour visualiser l'effet du seuil de décision sur la performance de notre modèle, nous allons utiliser la fonction **precision_recall_curve**. Cette fonction nous donne les valeurs de précision et de rappel pour différents seuils de décision. Nous pouvons ensuite tracer ces valeurs sur un graphique, avec le seuil de décision en abscisse et la précision et le rappel en ordonnée.



En général, il existe un compromis entre la précision et le rappel. Plus la précision augmente, plus le rappel diminue, et vice versa. Nous devons trouver un équilibre qui convient à nos besoins, en fonction de notre cahier des charges. Par exemple, nous pourrions sacrifier un peu de précision pour obtenir un meilleur rappel, ou vice versa.

Pour trouver le seuil de décision optimal qui maximise à la fois la précision et le rappel, nous pouvons utiliser le score F1, qui est une mesure combinée de la précision et du rappel. Nous pouvons ajuster le seuil de décision pour maximiser le score F1.

Enfin, nous pouvons créer une fonction finale qui prend en paramètre notre modèle entraîné et un seuil de décision, et qui retourne les prédictions du modèle en fonction de ce seuil. Nous pouvons utiliser cette fonction pour évaluer la performance de notre modèle avec différents seuils de décision.

```
def final_model(model, x,threshold=0):
    return model.decision_function(x)>threshold
```

En utilisant cette approche, nous pouvons ajuster notre modèle pour obtenir les meilleures performances possibles en fonction de nos besoins spécifiques.

le test de modèle final:
Lorsque nous choisissons le seuil égal à -1,1, nous obtenons les résultats suivants :
- F1-score : **0.540540540540406**
- Recall :**0.7692307692307693**, ce qui est supérieur à notre objectif de 0.7.

Ainsi, nous pouvons conclure que notre approche est plutôt efficace et produit de bons résultats.

ou est les résultats?

Parmi trois cas positifs, notre modèle parvient à en prédire correctement deux. De plus, il réussit à prédire correctement les cas négatifs. En conclusion, on peut affirmer que notre modèle final est bien construit et capable de généraliser les données.