Machine Learning (ML)

Chapitre 3 : Évaluation des modèles
Partie I

David TCHOUTA

Académie Française du Numérique
www.frenchtechacademie.fr
Tél/Whatsapp : +33 (0)7 49 62 72 49

April 9, 2022

"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison

Table de matières

Objectifs du chapitre

Dans ce chapitre, nous allons passer en revue les métriques (mesures de performance des modèles) les plus importantes en ML.

À la fin de ce chapitre, vous serez en mesure :

- de manipuler la précision et le rappel (recall)
- de manipuler le F1 score et la matrice de confusion
- de comprendre le problème de surapprentissage (overfitting)
- de manipuler la courbe ROC (Receiver Operating Characteristic) et AUC (Area Under the Curve)
- de réaliser le compromis entre la precision et la recall

Autres métriques 1

Dans le chapitre précédent, nous avons mesuré la qualité du modèle grâce à la précision (accuracy). Par exemple, si nous disposons de 100 points et que le modèle prédit 70 correctement et 30 de manière incorrect, alors on dit que la précision est de 70%.
La précision est intuitif et facile à comprendre mais ce n'est pas toujours la meilleure métrique à observer.
Par exemple, supposons que nous disposons d'un modèle permettant de prédire si les frais de carte de crédit est frauduleux.
Sur 10000 cartes de crédit, nous avons 9900 frais légitimes et 100 frais frauduleux.
Nous pourrions construire un modèle qui prédit simplement que chaque débit est légitime et il obtiendrait 9900/10000 (99%) des prédictions correctes.

Autres métriques 2

Attention !

La précision est une bonne mesure si nos classes sont réparties de manière égale, mais **elle est très trompeuse** si **nos classes sont déséquilibrées**.

Attention !

Il faut toujours être prudent avec la précision. Vous devez connaître la distribution des classes pour savoir comment interpréter sa valeur.

Exercice d'application 1

Accuracy

Supposons que l'on vous demande de construire un modèle pour prédire les spams. Votre ensemble d'apprentissage comprend 1000 e-mails, 950 sont des e-mails légitimes et 50 sont des spams. Vous construisez un modèle qui prédit simplement que tous les e-mails sont légitimes. Quelle est la précision du modèle ?

1. 99%
2. 100%
3. 50%
4. 95%

Matrice de confusion (Confusion Matrix) 1

Comme nous l'avons remarqué dans la partie précédente, nous ne nous intéressons pas seulement au nombre de points de données pour lesquels nous prédisons la bonne classe, mais aussi au nombre de points de données positifs pour lesquels nous prédisons correctement, ainsi qu'au nombre de points de données négatifs pour lesquels nous prédisons correctement.

Nous pouvons voir toutes les notions importantes dans ce que l'on appelle la **confusion matrix**(or Error Matrix or Table of Confusion).

Confusion Matrix 2

La matrice de confusion est un tableau montrant quatre valeurs :

1. Les points de données que nous avions prédits positifs et qui sont en fait positifs
2. Les points de données que nous avions prédits positifs et qui sont en fait négatifs
3. Les points de données que nous avons prédits négatifs et qui sont en fait positifs
4. Les points de données que nous avons prédits négatifs et qui sont en fait négatifs

Le **1er et le 4e** sont les points de données que nous avons **prédits correctement**, le **2e et le 3e** point sont les points de données que nous avons **prédits incorrectement**.

Confusion Matrix 3

Dans notre jeu de données Titanic, nous avons 887 passagers, 342 ont survécu (positif) et 545 n'ont pas survécu (négatif). Le modèle que nous avons construit dans le module précédent a la matrice de confusion suivante :

	Actual positive	**Actual negative**
Predicted positive	233	65
Predicted negative	109	480

Les nombres en bleu représentent le nombre de prédictions correctes. Ainsi, sur les **342 passagers qui ont survécu**, nous en avons prédit **233 correctement** (et **109 incorrectement**). Sur les **545 passagers qui n'ont pas survécu**, nous en avons prédit **480 correctement (et 65 incorrectement)**.

Confusion Matrix 3

Nous pouvons utiliser la matrice de confusion pour calculer la précision. Pour rappel, la précision est le nombre de points de données correctement prédits divisé par le nombre total de points de données :
$(233+480)/(233+65+109+480) = 713/887 = \mathbf{80.38\%}$
Il s'agit en effet de la même valeur que celle obtenue au chapitre précédent.

Attention !

La matrice de confusion décrit pleinement les performances d'un modèle sur un ensemble de données, mais elle est difficile à utiliser pour comparer les modèles.

Exercice d'application 2

Accuracy

Soit la matrice de confusion suivante :

	Actual positive	**Actual negative**
Predicted positive	22	24
Predicted negative	10	44

Quelle est la précision ?

1. 60%
2. 80%
3. 66%
4. 32%

True Positives, True Negatives, False Positives, False Negatives 1

Chaque cellule de la matrice de confusion dispose d'un nom :

1. Un **vrai positif (TP or True Positive)** est un point de données pour lequel nous avons prédit un résultat positif et pour lequel nous avons eu raison (1ere cellule).
2. Un **vrai négatif (True Negative or TN)** est un point de données pour lequel nous avons prédit un résultat négatif et pour lequel nous avons eu raison (4e cellule).
3. Un **faux positif (FP or False Positive)** est un point de données que nous avons prédit positivement et pour lequel nous nous sommes trompés (2e cellule).
4. Un **faux négatif (FN or False Negative)** est un point de données pour lequel nous avons prédit un résultat négatif mais pour lequel nous nous sommes trompés (3e cellule).

True Positives, True Negatives, False Positives, False Negatives 2

Il peut être difficile de s'y retrouver dans ces termes. La façon de s'en souvenir est la suivante : le deuxième mot est la nature de notre prédiction (positive ou négative) et le premier mot indique si cette prédiction est correcte (vraie ou fausse).
Vous verrez souvent la matrice de confusion comme suit :

	Actual positive	**Actual negative**
Predicted positive	TP	FP
Predicted negative	FN	TN

Attention !

Les quatre valeurs de la matrice de confusion (TP, TN, FP, FN) sont utilisées pour calculer plusieurs métriques différentes que nous utiliserons par la suite.

Exercice d'application 3

Confusion matrix

Soit la matrice de confusion suivante :

	Actual positive	**Actual negative**
Predicted positive	233	65
Predicted negative	109	480

Complétez par TN, TP, FP, FN

1. Il y a 233 .....
2. Il y a 65 .....
3. Il y a 109 .....
4. Il y a 480 .....

Precision 1

Il y a **deux métriques** généralement observées en **classification** : la **précision** (**precision**, à ne pas confondre avec l'accuracy) et le **recall** (rappel).
En théorie, la **précision fait référence au pourcentage de résultats positifs qui sont pertinents** et le **rappel au pourcentage de cas positifs correctement classés**.

Precision 2

La **précision est le pourcentage de prédictions positives du modèle qui sont correctes**. Nous la définissons comme suit :

	Actual positive	**Actual negative**
Predicted positive	TP	FP
Predicted negative	FN	TN

$$precision = \frac{nombre.de.predictions.positives.qui.sont.correctes}{nombre.de.predictions.positives} = \frac{TP}{TP+FP}$$

Precision 3

Si nous reprenons notre exemple de la matrice de confusion du Titanic :

	Actual positive	**Actual negative**
Predicted positive	233	65
Predicted negative	109	480

La précision est : precision = 233 / (233 + 65) = 0.7819

Exercice d'application 4

Calcul de la précision

Soit la matrice de confusion suivante :

	Actual positive	**Actual negative**
Predicted positive	30	20
Predicted negative	10	40

Quelle est la précision ?

1. 0.5
2. 0.6
3. 0.8
4. 0.65

Recall (Rappel) 1

Le **rappel (recall)** est le **pourcentage de cas positifs que le modèle prédit correctement**. Encore une fois, nous utiliserons la matrice de confusion pour calculer notre résultat :

	Actual positive	**Actual negative**
Predicted positive	TP	FP
Predicted negative	FN	TN

$$recall = \frac{nombre.de.predictions.positives.correctes}{nombre.de.cas.positifs} = \frac{TP}{TP+FN}$$

recall = 233 / (233 + 109) = 0.6813

Recall (Rappel) 2

Attention !

Le **rappel** est une mesure du nombre de cas positifs que le modèle peut rappeler. On l'appelle également **sensitivity** ou **true positive rate**.

Exercice d'application 5

Calcul du recall

Soit la matrice de confusion suivante :

	Actual positive	**Actual negative**
Predicted positive	30	20
Predicted negative	10	40

Quelle est le rappel ?

1. 0.5
2. 0.6
3. 0.75
4. 0.65

Quel compromis entre la precision et le recall ? 1

Nous serons souvent dans une situation où nous devrons choisir entre augmenter le rappel (tout en diminuant la précision) ou augmenter la précision (et diminuer le rappel). Cela dépendra de la situation que nous voudrons maximiser.

Par exemple, supposons que nous construisons un modèle pour prédire si les frais de carte de crédit sont frauduleux. Les cas positifs pour notre modèle sont les frais frauduleux et les cas négatifs sont les frais légitimes.

Considérons les deux scénarios suivants :
1. Si nous prédisons que les frais sont frauduleux, nous la rejetterons. 2. Si nous prédisons que les frais sont frauduleux, nous appelons le client pour confirmer le débit.

Quel compromis entre la precision et le recall ? 2

Dans le cas 1, le client subit un énorme désagrément lorsque le modèle prédit une fraude de manière incorrecte (un faux positif). Dans le cas 2, un faux positif est un inconvénient mineur pour le client.

Plus le nombre de faux positifs est élevé, plus la précision est faible. En raison du coût élevé des faux positifs dans le premier cas, il vaudrait la peine d'avoir un faible rappel afin d'avoir une précision très élevée. Dans le deuxième cas, vous souhaitez un meilleur équilibre entre la précision et le rappel.

Attention !

Il n'y a pas de règle absolue concernant les valeurs de précision et de rappel à atteindre. Cela dépend toujours des données et de l'application.

Exercice d'application 6

Compromis entre la precision et le recall

Supposons un modèle pour prédire le spam. Les cas positifs sont des spams et les cas négatifs sont des mails légitimes. Si nous devons supprimer les e-mails que nous prédisons être des spams, qu'est-ce qui est le plus important à maximiser ?

1. recall
2. precision

Autre métrique : F1 score 1

L'accuracy était une mesure attrayante car il s'agissait d'un seul chiffre. La précision et le rappel étant deux nombres, il n'est pas toujours évident de choisir entre deux modèles si l'un a une meilleure précision et l'autre un meilleur rappel. **Le score F1 est une moyenne de la précision et du rappel**, ce qui nous permet d'obtenir un score unique pour notre modèle.
Voici la formule mathématique pour le score F1 :

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

Autre métrique : F1 score 2

Calculons le score F1 de notre modèle pour le jeu de données Titanic.
Nous allons utiliser les valeurs de précision (0.7819) et de rappel (0.6813) que nous avons calculés précédemment.
Le **score F1** est : 2 * (0.7819) * (0.6813) / (0.7819 + 0.6813) = **0.7281**

Attention !

Le score F1 est la moyenne harmonique des valeurs de précision et de rappel.

Exercice d'application 7

F1 score

Vous avez construit un modèle qui a une précision de 0,8 et un rappel de 0,5. Quelle est l'équation pour le score F1 ?

1. 2 * (0.8 + 0.5) / (0.8 * 0.5)
2. (0.8 + 0.5) / (0.8 * 0.5)
3. 2 * (0.8 * 0.5) / (0.8 + 0.5)
4. (0.8 * 0.5) / (0.8 + 0.5)

Metrics in sklearn 1

Commençons par rappeler notre code du module précédent pour construire un modèle de régression logistique. Le code charge les données du Titanic à partir du fichier csv et le place dans un DataFrame Pandas. Nous créons ensuite une matrice des prédicteurs X et de la variable cible y. Nous créons un modèle de régression logistique et l'ajustons à nos données. Enfin, nous créons une variable y_pred de nos prédictions.

```
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
model = LogisticRegression()
model.fit(X, y)
y_pred = model.predict(X)
```

Metrics in sklearn 2

Maintenant, nous sommes prêts à utiliser nos fonctions de calcul des métriques. Importons-les depuis scikit-learn :

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
```

Chaque fonction prend deux tableaux numpy à une dimension : les valeurs réelles de la cible et les valeurs prédites de la cible. Nous avons les valeurs réelles de la cible et les valeurs prédites de la cible. Nous pouvons donc utiliser les fonctions métriques comme suit :

```
print("accuracy:", accuracy_score(y, y_pred)) ⟹ 0.804960
print("precision:", precision_score(y, y_pred)) ⟹ 0.773462
print("recall:", recall_score(y, y_pred)) ⟹ 0.698830
print("f1 score:", f1_score(y, y_pred)) ⟹ 0.734254
```

Metrics in sklearn 3

Nous constatons que l'accuracy est de 80%, ce qui signifie que 80% des prédictions du modèle sont correctes. La précision est de 78%, ce qui correspond au pourcentage de prédictions positives du modèle qui sont correctes. Le rappel est de 68%, ce qui correspond au pourcentage de cas positifs que le modèle a prédit correctement. Le score F1 est de 73%, qui est une moyenne de la précision et du rappel.

Attention !

Avec un seul modèle, les métriques ne nous disent pas grand-chose. Pour certains problèmes, une valeur de 60% est bonne, et pour d'autres, une valeur de 90% est bonne, selon la difficulté du problème. Nous utiliserons les métriques pour comparer différents modèles et choisir le meilleur.

Exercice d'application 8

Accuracy, precision, recall and F1 score

Supposons que nous ayons un tableau numpy 2d de X variables dépendantes et un tableau numpy 1D y de la variable cible. Réorganisez le code afin de construire un modèle sur les données et d'afficher la précision, le rappel et le score F1 dans cet ordre.

1. print("recall:", recall_score(y, ypred))
2. print("F1 score:", f1_score(y, ypred))
3. ypred = model.predict(X)
4. model = LogisticRegression()
5. print("precision:", precision_score(y, ypred))
6. model.fit(X,y)

Confusion Matrix in Sklearn 1

Scikit-learn possède une fonction permettant de calculer la matrice de confusion que nous pouvons utiliser pour obtenir les quatre valeurs de la matrice de confusion (vrais positifs, faux positifs, faux négatifs et vrais négatifs). En supposant que y représente nos vraies valeurs cibles et que y_pred représente les valeurs prédites, nous pouvons utiliser la fonction confusion_matrix() comme suit :

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y, y_pred))
```

```
[[475  70]
 [103 239]]
```

Attention !

Ne pas oublier d'importer la métrique avant de l'utiliser.

Confusion Matrix in Sklearn 2

Scikit-learn inverse la matrice de confusion pour montrer les négatifs (Actual et Predicted) en premier :

	Predicted negative	**Predicted positive**
Actual negative	475	70
Actual positive	103	239

Au lieu de :

	Actual positive	**Actual negative**
Predicted positive	239	70
Predicted negative	103	475

Confusion Matrix in Sklearn 3

Attention !

Comme les valeurs cibles négatives correspondent à 0 et les positives à 1, scikit-learn les a classées dans cet ordre. Assurez-vous de bien vérifier que vous interprétez les valeurs correctement.

Exercice d'application 9

Confusion matrix

Soit la matrice de confusion suivante :

4	1
3	2

Complétez :

1. True positives : .....
2. False positives : .....
3. False negatives : .....
4. True negatives : .....

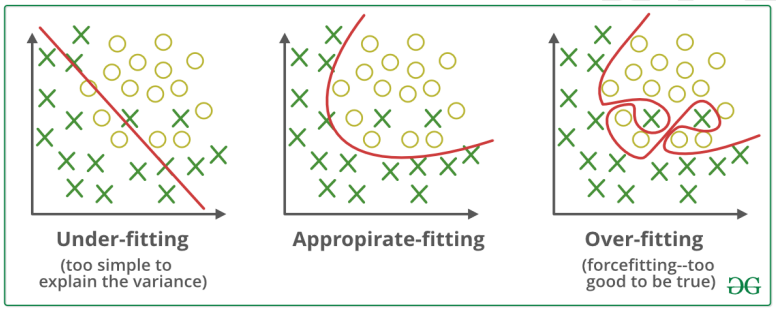
Overfitting ou surapprentissage 1

Definition

Jusqu'à présent, nous avons construit un modèle avec toutes nos données, puis nous avons observé ses performances sur ces mêmes données. Cela gonfle artificiellement nos chiffres puisque notre modèle a pu voir les réponses au quiz avant que nous lui donnions le quiz. Cela peut conduire à ce que nous appelons **l'overfitting (le surapprentissage)**. On parle **surapprentissage** (d'ajustement excessif) **lorsque nous obtenons de bons résultats sur les données que le modèle a déjà vues, mais que nous n'obtenons pas de bons résultats sur les nouvelles données**.

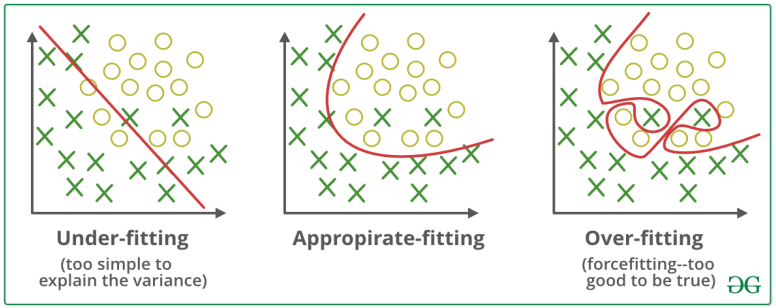
Overfitting ou surapprentissage 2

On peut voir un modèle surajusté ou surentraîné (overfit) comme suit:



Dans le graphique 3, la ligne est trop serrée et tente de placer chaque point de données du bon côté de la ligne, mais elle ne tient pas compte de l'essence des données.

Overfitting ou surapprentissage 3



Dans le graphique 3, vous pouvez voir que nous avons fait un assez bon travail pour obtenir les points jaunes à droite et les points verts à gauche, mais cela ne reflète pas ce qui se passe. Un seul point aberrant pourrait totalement perturber l'emplacement de la ligne. Alors que le modèle obtiendrait un excellent score sur les données qu'il a déjà vues, il est peu probable qu'il soit performant sur les nouvelles données.

Overfitting ou surapprentissage 4

Attention !

Plus nos données comporteront des variables dépendantes, plus nous serons enclins au surapprentissage (overfitting).

Exercice d'application 10

Overfitting : choisir les bonnes réponses

On parle d'overfitting lorsque :

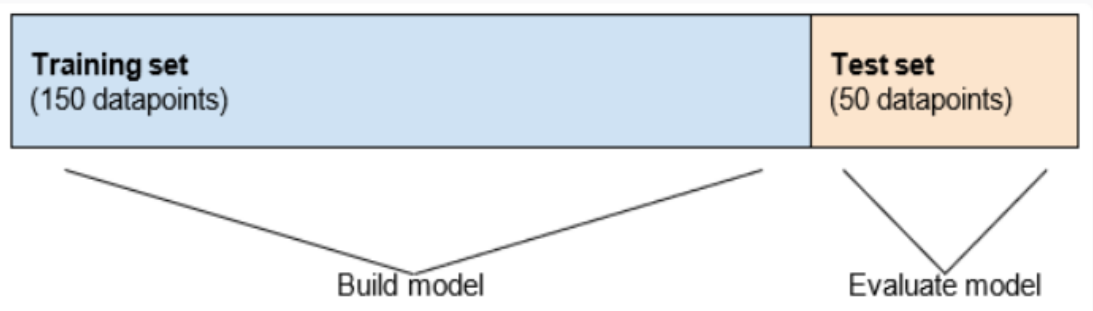
1. nous obtenons de meilleures performances (lorsqu'on réalise des prédictions) sur les nouvelles données .
2. nous avons de piètres performances sur les données que le modèle avait déjà vues.
3. nous avons de meilleures performances sur les données que le modèle avait déjà vues.
4. nous n'avons de pas de meilleures performances sur les nouvelles données.

Ensemble d'entraînement et ensemble de test 1

Pour donner à un modèle une évaluation non biaisée, nous aimerions savoir comment notre modèle se comporteraient sur des données qu'elles n'ont pas encore vues.
En effet, notre modèle fera des prédictions sur des données dont nous ne connaissons pas la réponse. Nous aimerions donc évaluer la performance de notre modèle sur de nouvelles données, et pas seulement sur les données qu'il a déjà vues. Pour simuler la réalisation de prédictions sur de nouvelles données non vues, nous pouvons **diviser notre ensemble de données en un ensemble d'apprentissage et un ensemble de test**. **L'ensemble d'apprentissage est utilisé pour construire les modèles. L'ensemble de test est utilisé pour évaluer les modèles**. Nous divisons nos données avant de construire le modèle, ainsi le modèle n'a aucune connaissance de l'ensemble de test et nous lui donnerons une évaluation bien meilleure.

Ensemble d'entraînement et ensemble de test 2

Si notre ensemble de données contient 200 points de données, la répartition en un **ensemble d'apprentissage (training set) et un ensemble de test (test set)** peut se présenter comme suit :



Répartition entre ensemble d'entraînement et ensemble de test

Attention !

Une répartition standard consiste à placer 70 à 80% de nos données dans l'ensemble d'apprentissage et 20 à 30% dans l'ensemble de test. En utilisant moins de données dans l'ensemble d'apprentissage, notre modèle n'aura pas autant de données pour apprendre, nous voulons donc lui en donner le plus possible tout en en laissant suffisamment pour l'évaluation.

Exercice d'application 11

Training set and test set

Lequel permet d'évaluer le modèle ?

1. training set
2. test set
3. l'ensemble des données
4. la base de données

Training set and test set in Sklearn 1

Scikit-learn dispose d'une fonction intégrée pour **diviser les données en un ensemble d'entraînement et un ensemble de test**.

En supposant que nous ayons un tableau numpy bidimensionnel X de nos prédicteurs et un tableau numpy unidimensionnel y de la cible, nous pouvons utiliser **la fonction train_test_split()**. Elle placera **aléatoirement chaque point de données** soit dans l'ensemble **d'entraînement**, soit dans l'ensemble **de test**. **Par défaut, l'ensemble d'apprentissage représente 75% des données et l'ensemble de test représente les 25% restants des données**.

Nous pouvons utiliser l'attribut shape pour voir les tailles de nos ensembles de données.

Training set and test set in Sklearn 2

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
X_train, X_test, y_train, y_test = train_test_split(X, y)
print("whole dataset:", X.shape, y.shape) ⟹ (887, 6) (887,)
print("training set:", X_train.shape, y_train.shape) ⟹ (665, 6)
(665,)
print("test set:", X_test.shape, y_test.shape) ⟹ (222, 6) (222,)
```

Training set and test set in Sklearn 3

Nous pouvons voir que sur les 887 points de données de notre ensemble de données, 665 sont dans notre ensemble d'apprentissage et 222 dans l'ensemble de test. Chaque point de données de notre ensemble de données est utilisé exactement une fois, soit dans l'ensemble d'apprentissage, soit dans l'ensemble de test. Notez que nous avons 6 prédicteurs dans notre ensemble de données, donc nous avons toujours 6 prédicteurs dans notre ensemble d'entraînement et notre ensemble de test.

Attention !

Nous pouvons modifier la taille de notre ensemble d'apprentissage en utilisant le paramètre **train_size**. Par exemple, **train_test_split(X, y, train_size=0.6)** place **60% des données dans l'ensemble d'apprentissage et 40% dans l'ensemble de test**.

Exercice d'application 12

Training set and test set in Sklearn

Nous avons un tableau numpy 2d X de 100 points de données et 4 prédicteurss et un tableau 1d y de 100 valeurs cibles. Quel est le résultat du programme ci-dessous ?

```
X_train, X_test, y_train, y_test = train_test_split(X, y)

print(X_train.shape, y_train.shape)
```

(..... , ...) (..... ,)

Construction d'un modèle dans Sklearn 1

Maintenant que nous savons comment diviser nos données en un ensemble d'apprentissage et un ensemble de test, nous devons modifier la façon dont nous construisons et évaluons le modèle. Toute la construction du modèle se fait avec l'ensemble d'apprentissage et toute l'évaluation se fait avec l'ensemble de test.

Dans le chapitre précédent, nous avons construit un modèle et l'avons évalué sur le même ensemble de données. Maintenant, nous construisons le modèle en utilisant l'ensemble d'entraînement:

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

Construction d'un modèle dans Sklearn 2

Évaluons le modèle sur l'ensemble de test :
print(model.score(X_test, y_test))
En fait, toutes les métriques que nous calculons dans les parties précédentes doivent être calculées sur l'ensemble de test :
evaluating the model
y_pred = model.predict(X_test)
print("accuracy:", accuracy_score(y_test, y_pred)) $\Longrightarrow$ 0.833333
print("precision:", precision_score(y_test, y_pred)) $\Longrightarrow$ 0.793650
print("recall:", recall_score(y_test, y_pred)) $\Longrightarrow$ 0.675675
print("f1 score:", f1_score(y_test, y_pred)) $\Longrightarrow$ 0.729927

Attention !

Les valeurs des métriques peuvent changer d'une exécution à une autre.

Construction d'un modèle dans Sklearn 3

Nos valeurs d'accuracy, de précision, de rappel et de score F1 sont en fait très similaires aux valeurs obtenues lorsque nous avons utilisé l'ensemble des données. C'est un signe que notre modèle n'est pas surajusté (n'est pas overfitting).

Attention !

Si vous exécutez le code, vous remarquerez que vous obtenez des scores différents à chaque fois. Cela s'explique par le fait que la répartition entraînement-test est effectuée de manière aléatoire et que les scores seront différents selon les points qui se trouvent dans l'ensemble de formation et dans l'ensemble de test. Nous verrons, lors de la leçon sur la validation croisée, que nous disposons de moyens plus précis pour mesurer ces scores.

Exercice d'application 13

Training set and test set

Quelle est l'instruction correcte ?

1. model.fit(X_test, y_test) – print(model.score(X_train, y_train))
2. model.fit(X_train, y_train) – print(model.score(X_train, y_train))
3. model.fit(X, y) – print(model.score(X_test, y_test))
4. model.fit(X_train, y_train) – print(model.score(X_test, y_test))
5. model.fit(X_train, y_test) print(model.score(X_train, y_test))

Utilisation d'un état aléatoire (random state or seed) 1

Comme nous l'avons remarqué dans la partie précédente, lorsque nous divisons aléatoirement les données en un ensemble d'apprentissage et un ensemble de test, nous nous retrouvons avec des points de données différents dans chaque ensemble à chaque fois que nous exécutons le code. C'est le résultat du caractère aléatoire, et nous avons besoin qu'il soit aléatoire pour qu'il soit efficace, mais cela peut parfois rendre difficile le test du code.

Random state or seed 2

Par exemple, chaque fois que nous exécutons le code suivant, nous obtenons des résultats différents :

```
from sklearn.model_selection import train_test_split
X = [[1, 1], [2, 2], [3, 3], [4, 4]]
y = [0, 0, 1, 1]
X_train, X_test, y_train, y_test = train_test_split(X, y)
print('X_train', X_train)
print('X_test', X_test)
```

Random state or seed 3

Pour obtenir **la même répartition à chaque fois**, nous pouvons utiliser l'attribut **random_state**. Nous choisissons un **nombre arbitraire à lui donner**, et **à chaque fois que nous exécutons le code, nous obtenons la même répartition**.

```
from sklearn.model_selection import train_test_split
X = [[1, 1], [2, 2], [3, 3], [4, 4]]
y = [0, 0, 1, 1]
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=2)
print('X_train', X_train)
print('X_test', X_test)
```

Exercice d'application 14

Random state or seed

Nous utilisons un paramètre random_state pour nous assurer que nous obtenons la même répartition aléatoire chaque fois que le même code est exécuté :

1. True
2. False

Seuil de régression logistique 1

Si vous vous souvenez de la section 2, nous avons parlé du **compromis entre la précision et le rappel**. Avec un **modèle de régression logistique**, nous disposons d'un moyen facile de passer de la précision au recall. Le modèle de régression logistique ne **renvoie** pas seulement une **prédiction**, mais aussi une **valeur de probabilité** comprise entre 0 et 1. En général, si la valeur est $>=$**0.5**, nous prédisons que l**e passager a survécu**, et si la valeur est $<$**0.5**, le passager **n'a pas survécu**. Cependant, nous pourrions choisir n'importe quel seuil entre 0 et 1.

Seuil de régression logistique 2

Si nous **augmentons le seuil**, nous aurons **moins de prédictions positives**, mais ces dernières auront plus de chances d'être correctes. Cela s**ignifie que la précision sera plus élevée et le rappel plus faible**. D'un autre côté, si **nous abaissons le seuil**, nous aurons **plus de prédictions positives**, et nous aurons donc plus de chances d'attraper tous les cas positifs. Cela signifie que **le rappel sera plus élevé et la précision plus faible**.

Attention !

Chaque choix de seuil est un modèle différent. **Une courbe ROC (Receiver operating characteristic) est un graphique montrant tous les modèles possibles et leurs performances**.

Exercice d'application 15

Seuil de regression logistique

Si nous choisissons un seuil de 0,75, laquelle des affirmations suivantes est vraie concernant nos prédictions et notre précision ?

1. plus de prédictions négatives / la précision sera plus faible
2. les prédictions plus négatives / la précision sera plus élevée
3. plus de prédictions positives / la précision sera plus élevée

Sensitivity and specificity 1

Definition

Une courbe ROC est un graphique de la sensibilité par rapport à la spécificité. Ces valeurs présentent le même compromis que la précision et le rappel.

Revenons sur la matrice de confusion, car nous l'utiliserons pour définir la sensibilité et la spécificité :

	Actual positive	**Actual negative**
Predicted positive	TP	FP
Predicted negative	FN	TN

La sensibilité,encore appelé le rappel (recall), c'est le taux de vrais positifs. Rappelons qu'elle est calculée comme suit :

$$recall = sensitivity = \frac{nombre.de.predictions.positives.correctes}{nombre.de.cas.positifs} = \frac{TP}{TP+FN}$$

Sensitivity and specificity 2

La spécificité (specificity) est le taux de vrais négatifs (true negative rate) :

$$specificity = \frac{nombre.de.predictions.negatives.correctes}{nombre.de.cas.negatifs} = \frac{TN}{TN+FP}$$

Nous avons effectué un train test split (diviser nos données entre train et test) sur notre jeu de données Titanic et obtenu la matrice de confusion suivante :

	Actual positive	**Actual negative**
Predicted positive	61	21
Predicted negative	35	105

Nous avons 96 cas positifs et 126 cas négatifs dans notre ensemble de test.

Sensitivity and specificity 3

Calculons la sensibilité et la spécificité :
sensibility $= 61/96 = 0.6354$
specificity $= 105/126 = 0.8333$
L'objectif est de maximiser ces deux valeurs, bien qu'en général, le fait d'augmenter l'une dimunie l'autre. Selon la situation, on mettra davantage l'accent sur la sensibilité ou la spécificité.

Attention !

Alors que nous examinons généralement les valeurs de précision et de rappel, pour **la représentation graphique**, la norme est d'utiliser **la sensibilité et la spécificité**. Il est possible de construire une courbe précision-rappel, mais ce n'est pas une pratique courante.

Exercice d'application 16

Sensitivity and specificity

Soit la matrice de confusion suivante:

	Actual positive	**Actual negative**
Predicted positive	30	20
Predicted negative	10	40

Choisir la bonne réponse de la sensibilité et de la spécificité :

1. 0.22 / 0.40
2. 0.3 / 0.2
3. 0.75 / 0.666

Sensitivity and specificity in Sklearn 1

Scikit-learn n'a pas défini de fonctions pour la sensibilité et la spécificité, mais nous pouvons le faire nous-mêmes. La sensibilité est la même que le rappel, elle est donc facile à définir :

```
from sklearn.metrics import recall_score
sensitivity_score = recall_score
print(sensitivity_score(y_test, y_pred))
```

Maintenant, pour définir la spécificité, si nous réalisons que c'est aussi le rappel de la classe négative, nous pouvons obtenir la valeur de la fonction **sklearn precision_recall_fscore_support()**.

```
from sklearn.metrics import precision_recall_fscore_support
print(precision_recall_fscore_support(y_test, y_pred))
(array([0.81118881, 0.70886076]), array([0.83453237, 0.6746988
]), array([0.82269504, 0.69135802]), array([139, 83], dtype=int64))
```

Sensitivity and specificity in Sklearn 2

Le deuxième tableau est le rappel, nous pouvons donc ignorer les trois autres tableaux. Il y a deux valeurs. La première est le rappel de la classe négative et la seconde est le rappel de la classe positive. La deuxième valeur est le rappel standard ou la valeur de sensibilité, et vous pouvez voir que la valeur correspond à ce que nous avons obtenu ci-dessus. La première valeur est la spécificité. Écrivons donc une fonction pour obtenir cette valeur.

```
def specificity_score(y_true, y_pred):
    p, r, f, s = precision_recall_fscore_support(y_true, y_pred)
    return r[0]
print(specificity_score(y_test, y_pred)) ⟹ 0.83453237
```

Attention !

La sensibilité est identique au rappel (ou rappel de la classe positive) et la spécificité est le rappel de la classe négative.

Sensitivity and specificity in Sklearn 3

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score,
precision_recall_fscore_support
sensitivity_score = recall_score
def specificity_score(y_true, y_pred):
    p, r, f, s = precision_recall_fscore_support(y_true, y_pred)
    return r[0]
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
```

Sensitivity and specificity in Sklearn 4

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=5)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("sensitivity:", sensitivity_score(y_test, y_pred))
print("specificity:", specificity_score(y_test, y_pred))
```

Exercice d'application 17

Sensitivity and specificity in Sklearn

Quelle valeur obtenons-nous avec le code suivant ? p, r, f, s = precision_recall_fscore_support(y_test, y_pred)
print(r[1])

1. precision
2. specificity
3. sensibility

Seuil de régression logistique dans Sklearn 1

Lorsqu'on utilise la méthode de **predict()** de Scikit-learn, les valeurs **0 et 1 de la prédiction vous sont données**. Cependant, en coulisses, le modèle de régression logistique obtient une valeur de probabilité entre 0 et 1 pour chaque point de données, puis arrondit à 0 ou 1. Si nous voulons choisir un seuil différent de 0,5, nous aurons besoin de ces **valeurs de probabilité**.

Seuil de régression logistique dans Sklearn 2

Nous pouvons utiliser **la fonction predict_proba** pour les obtenir :

```
model.predict_proba(X_test)
```

Le résultat est un tableau numpy avec 2 valeurs pour chaque point de données (par exemple [0.78, 0.22]). Vous remarquerez que la somme des deux valeurs est égale à 1. La première valeur est la probabilité que le point de données soit dans la classe 0 (n'a pas survécu) et la seconde est la probabilité que le point de données soit dans la classe 1 (a survécu).

Seuil de régression logistique dans Sklearn 3

Nous n'avons besoin que de la deuxième colonne de ce résultat, que nous pouvons extraire grâce à la syntaxe suivante :
model.predict_proba(X_test)[:, 1]
Maintenant, nous voulons simplement comparer ces valeurs de probabilité avec notre seuil. Supposons que nous voulons un seuil de 0.75. Nous comparons le tableau ci-dessus à 0.75. Cela nous donnera un tableau de valeurs Vrai/Faux qui sera notre tableau de valeurs cibles prédites.
$y\_pred = model.predict\_proba(X\_test)[:, 1] > 0.75$
Un seuil de 0,75 signifie que nous devons être plus confiants pour faire une prédiction positive. Cela se traduit par moins de prédictions positives et plus de prédictions négatives.

Seuil de régression logistique dans Sklearn 4

Nous pouvons maintenant utiliser n'importe quelle métrique scikit-learn en utilisant y_test comme valeurs réelles et y_pred comme valeurs prédites :

```
print("precision:", precision_score(y_test, y_pred))
print("recall:", recall_score(y_test, y_pred))
```

Seuil de régression logistique dans Sklearn 5

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score
from sklearn.model_selection import train_test_split
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
```

Seuil de régression logistique dans Sklearn 6

```
X_train, X_test, y_train, y_test = train_test_split(X, y)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)[:, 1] > 0.75
print("precision:", precision_score(y_test, y_pred))
print("recall:", recall_score(y_test, y_pred))
```

Attention !

En fixant le seuil à 0,5, on obtient le modèle de régression logistique original. Toute autre valeur de seuil donne un autre modèle.

Exercice d'application 18

Seuil de régression logistique dans Sklearn

Laquelle des propositions suivantes nous donne un tableau des probabilités prédites (chaque valeur sera la probabilité que le point de données appartienne à la classe positive) ?

1. model.predict_proba(X_test)[1]
2. model.predict_proba(X_test)
3. model.predict_proba(X_test)[:, 0]
4. model.predict_proba(X_test)[:, 1]
5. model.predict_proba(X_test)[0]

Construction du ROC 1

La courbe ROC est un graphique de la spécificité par rapport à la sensibilité. Soit un modèle de régression logistique, calculons la spécificité et la sensibilité pour chaque seuil possible. Chaque probabilité prédite est un seuil. Si nous avons 5 points de données avec les probabilités prédites suivantes : 0.3, 0.4, 0.6, 0.7, 0.8, nous utiliserons chacune de ces 5 valeurs comme seuil.

Notez que nous traçons la sensibilité par rapport à la 1-spécificité (uniquement la deuxième colonne). Il n'y a pas de raison particulière de procéder de cette façon, si ce n'est que c'est la norme.
Commençons par examiner le code permettant de **construire la courbe ROC**. Scikit-learn possède une fonction **roc_curve()** que nous pouvons utiliser. Cette fonction prend les vraies valeurs de la variable cible et les probabilités prédites par notre modèle.

Construction du ROC 2

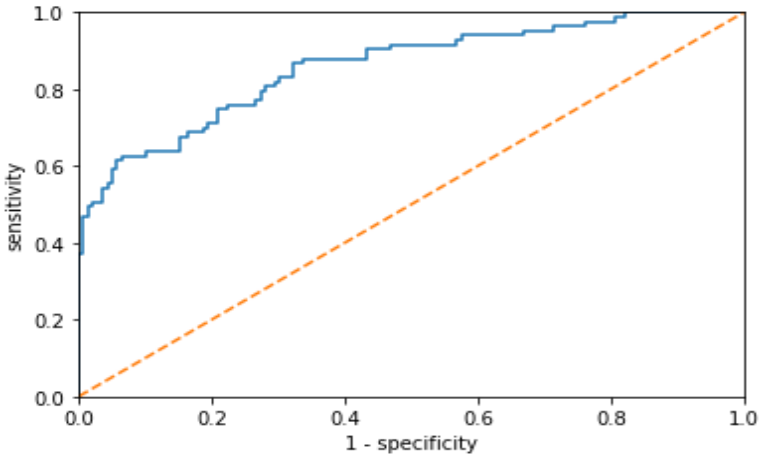
Nous utilisons d'abord la méthode predict_proba() sur le modèle pour obtenir les probabilités. Ensuite, nous appelons la fonction roc_curve. La **fonction roc_curve() renvoie un tableau des taux de faux positifs, un tableau des taux de vrais positifs et les seuils**. Le taux de faux positifs est égal à 1-spécificité (axe des x) et le taux de vrais positifs est un autre terme pour la sensibilité (axe des y). Les valeurs de seuil ne seront pas nécessaires dans le graphique.

Ci-dessous le code pour tracer la courbe ROC dans matplotlib. Notez que nous avons également du code pour tracer une ligne diagonale. Cela peut nous aider à voir visuellement à quelle distance se trouve notre modèle par rapport à un modèle qui prédit de manière aléatoire.

Construction du ROC 3

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_curve
import matplotlib.pyplot as plt
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred_proba = model.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba[:,1])
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], linestyle='–')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('1 - specificity') v plt.ylabel('sensitivity')
plt.show()
```

Construction du ROC 4



Attention !

Comme nous n'utilisons pas les valeurs seuils pour construire le graphique, celui-ci ne nous indique pas quel seuil donnerait chacun des modèles possibles.

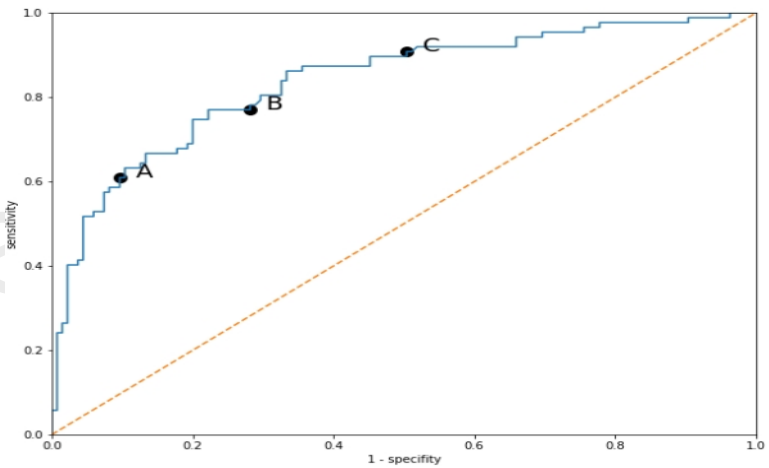
Exercice d'application 19

Construction du ROC

Écrire un programme qui permet de construire la courbe ROC avec scikit-learn.

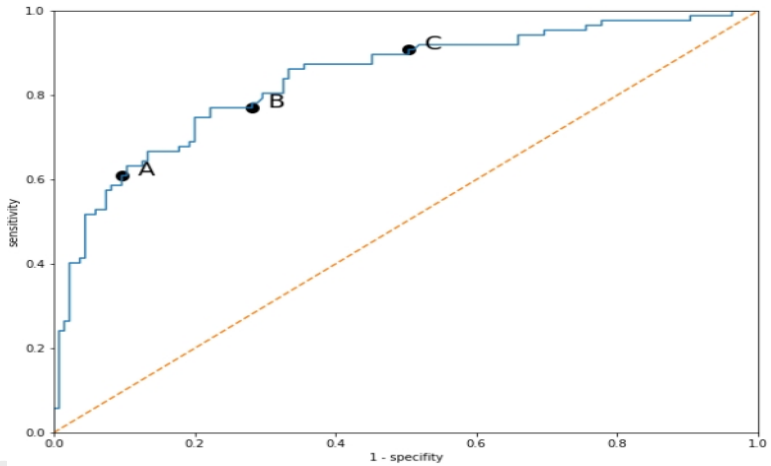
Interprétation de la courbe ROC 1

La courbe ROC montre la performance, non pas d'un seul modèle, mais de plusieurs modèles. Chaque choix de seuil est un modèle différent.
Soit courbe ROC avec les points mis en évidence ci-dessous :



Chaque point A, B et C correspond à un modèle avec un seuil différent.

Interprétation de la courbe ROC 2



Le modèle A a une sensibilité de 0,6 et une spécificité de 0,9 (rappelons que le graphique indique une spécificité de 1).
Le modèle B a une sensibilité de 0,8 et une spécificité de 0,7.
Le modèle C a une sensibilité de 0,9 et une spécificité de 0,5.

Interprétation de la courbe ROC 3

La façon de choisir entre ces modèles dépendra des spécificités de notre situation.

Attention !

Plus la courbe se rapproche du coin supérieur gauche, meilleures sont les performances. La courbe ne doit jamais tomber en dessous de la ligne diagonale, car cela signifierait que les performances sont inférieures à celles d'un modèle aléatoire.

Exercice d'application 19

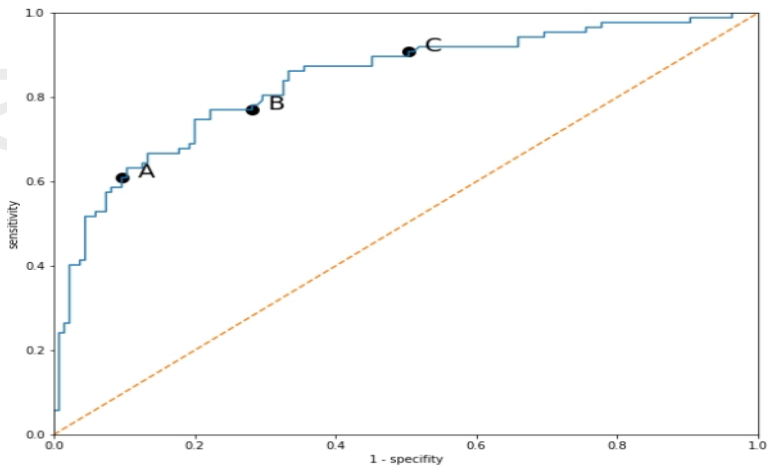
Interprétation de la courbe ROC

Dans quel coin du tracé de la courbe ROC se situerait le modèle idéal ?

1. Nord-Ouest
2. Nord-Est
3. Sud-Ouest
4. Nord-Ouest

Choix de la courbe ROC 1

Lorsque nous sommes prêts à finaliser notre modèle, nous devons choisir un seuil unique que nous utiliserons pour faire nos prédictions. La courbe ROC est un moyen de nous aider à choisir le seuil idéal pour notre problème.
Regardons à nouveau notre courbe ROC avec trois points mis en évidence :



Choix de la courbe ROC 2

Si nous sommes dans une situation où **il est plus important que toutes nos prédictions positives soient correctes que nous attrapions tous les cas positifs (ce qui signifie que nous prédisons correctement la plupart des cas négatifs), nous devrions choisir le modèle avec une spécificité plus élevée (modèle A)**.

Si nous sommes dans une situation où **il est important d'attraper le plus grand nombre possible de cas positifs, nous devrions choisir le modèle ayant la plus grande sensibilité (modèle C)**.

Choix de la courbe ROC 3

Si nous voulons un équilibre entre la sensibilité et la spécificité, nous devons choisir le modèle B.

Attention !

Il peut être difficile de garder la trace de tous ces termes. Même les experts doivent les consulter à nouveau pour s'assurer qu'ils interprètent correctement les valeurs.

Exercice d'application 20

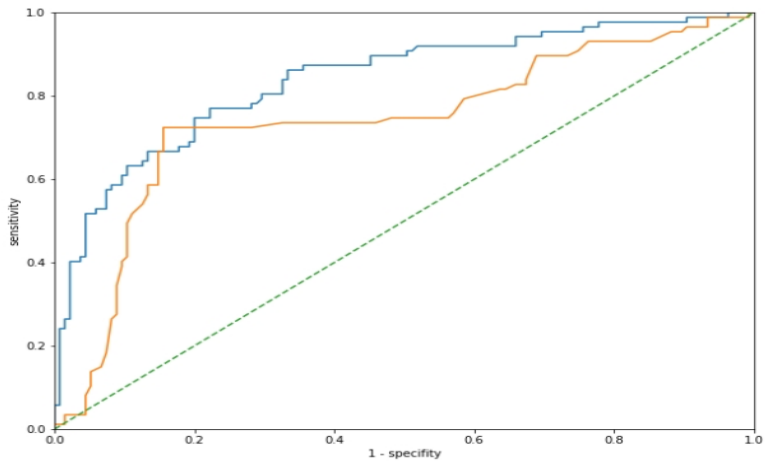
Choix du modèle grâce à la courbe ROC

Supposons que nous ayons un modèle permettant de prédire les fraudes à la carte de crédit. Si nous détectons des frais frauduleux sur le compte d'une personne, nous allons désactiver sa carte de crédit. Nous voulons donc nous assurer que notre prédiction est exacte lorsqu'elle est positive. Lequel des trois modèles du tracé ROC est préféré dans ce cas ?

1. modèle B
2. modèle A
3. modèle C

Aire sous la courbe 1

Nous utiliserons parfois la courbe ROC pour comparer deux modèles différents. Voici une comparaison des courbes ROC de deux modèles.



Vous pouvez constater que **la courbe bleue est plus performante que la courbe orange** puisque la **ligne bleue est presque toujours supérieure à la ligne orange**.

Aire sous la courbe 2

Pour obtenir une mesure empirique de ce résultat, nous calculons l'aire sous la courbe, également appelée **AUC (Area Under the Curve)**. Il s'agit de l'aire sous la courbe ROC. Il s'agit d'une **valeur comprise entre 0 et 1**, **plus elle est élevée, mieux c'est**.

Comme le ROC est un graphique de tous les différents modèles de régression logistique avec différents seuils, l'AUC ne mesure pas la performance d'un seul modèle. Elle donne un sens général de la performance du modèle de régression logistique. Pour obtenir un modèle unique, vous devez encore trouver le seuil optimal pour votre problème.

Aire sous la courbe 3

Scikit-learn grâce à la **fonction roc_auc_score()** permet de **calculer l'aire sous la courbe** :

(roc_auc_score(y_test, y_pred_proba[:,1])

Voici les valeurs des deux courbes (bleue et orange) :

Blue AUC: 0.8379
Orange AUC: 0.7385

Nous constatons empiriquement ($0.8379 > 0.7385$) que la courbe bleue est la plus performante.

Aire sous la courbe 4

Nous pouvons utiliser la fonction roc_auc_score() pour calculer le score AUC d'un modèle de régression logistique sur les données du Titanic. Nous construisons deux modèles de régression logistique, le modèle 1 avec 6 prédicteurs et le modèle 2 avec seulement les prédicteurs Pclass et male (variable booléenne que nous avons créée). **Nous constatons que le score AUC du modèle 1 est plus élevé à chaque exécution**.

Aire sous la courbe 5

```
from sklearn.metrics import roc_auc_score
model1 = LogisticRegression()
model1.fit(X_train, y_train)
y_pred_proba1 = model1.predict_proba(X_test)
print("model 1 AUC score:", roc_auc_score(y_test, y_pred_proba1[:,
1]))
model2 = LogisticRegression()
model2.fit(X_train[:, 0:2], y_train)
y_pred_proba2 = model2.predict_proba(X_test[:, 0:2])
print("model 1 AUC score:", roc_auc_score(y_test, y_pred_proba2[:,
1]))
```

Aire sous la courbe 6

Attention !

Il est important de noter que cette métrique nous indique la performance générale d'un modèle de régression logistique sur nos données. **Comme une courbe ROC montre la performance de plusieurs modèles, l'AUC ne mesure pas la performance d'un seul modèle**.

Exercice d'application 21

AUC

Complétez le programme ci-dessous pour calculer le score AUC d'un modèle de régression logistique. Supposez que X_train, X_test, y_train, y_test ont déjà été créés.

```
model = ..... ()
model. ..... (X_train, y_train)
y_pred_proba = model.predict.proba( ..... )
print(roc_auc_score( ..... , y_pred_proba[:, 1]))
```