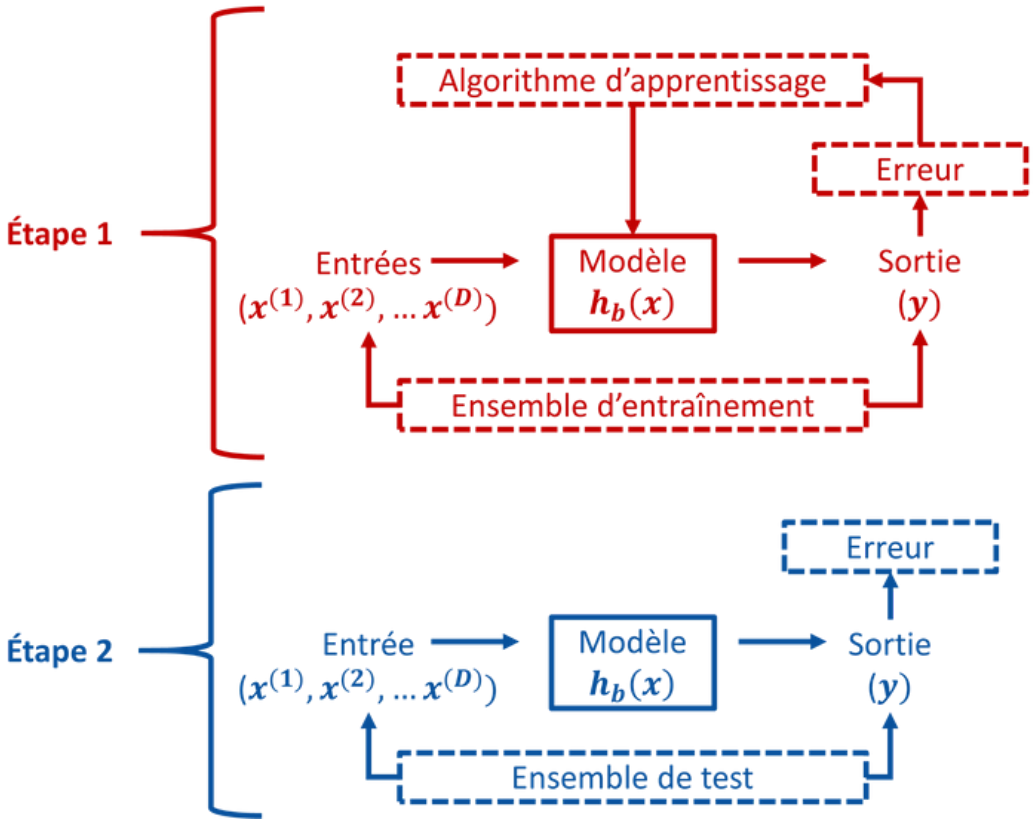
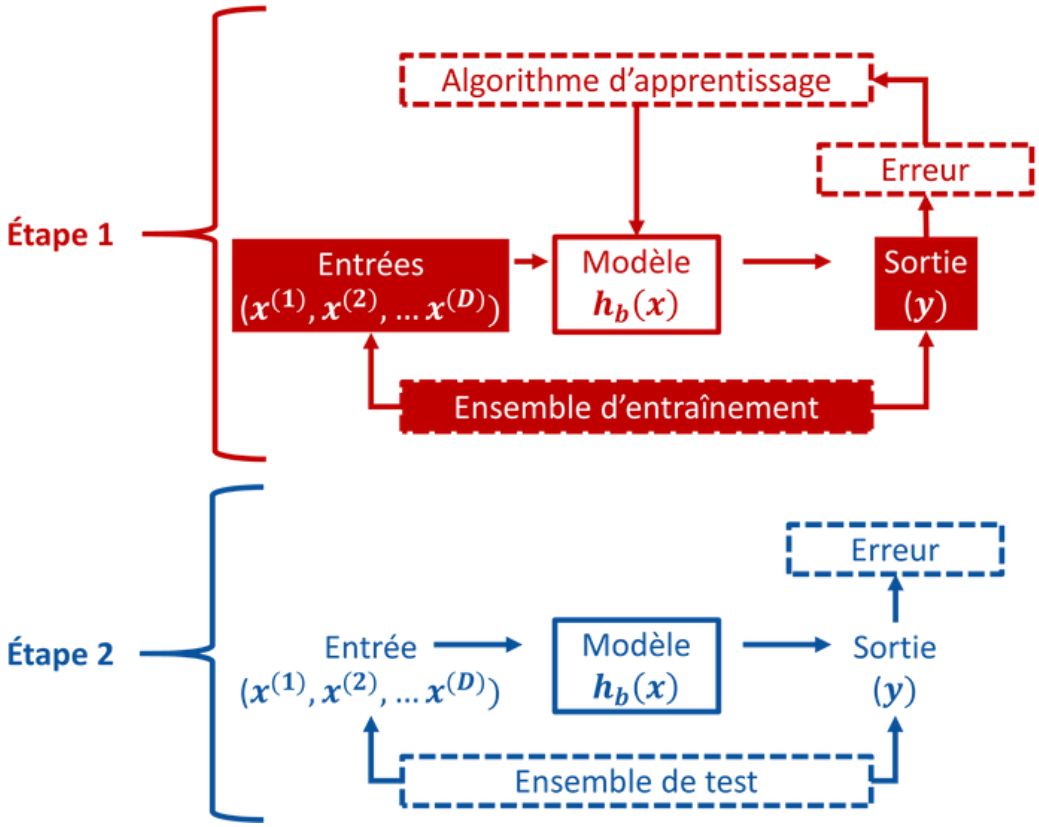
COURS 2

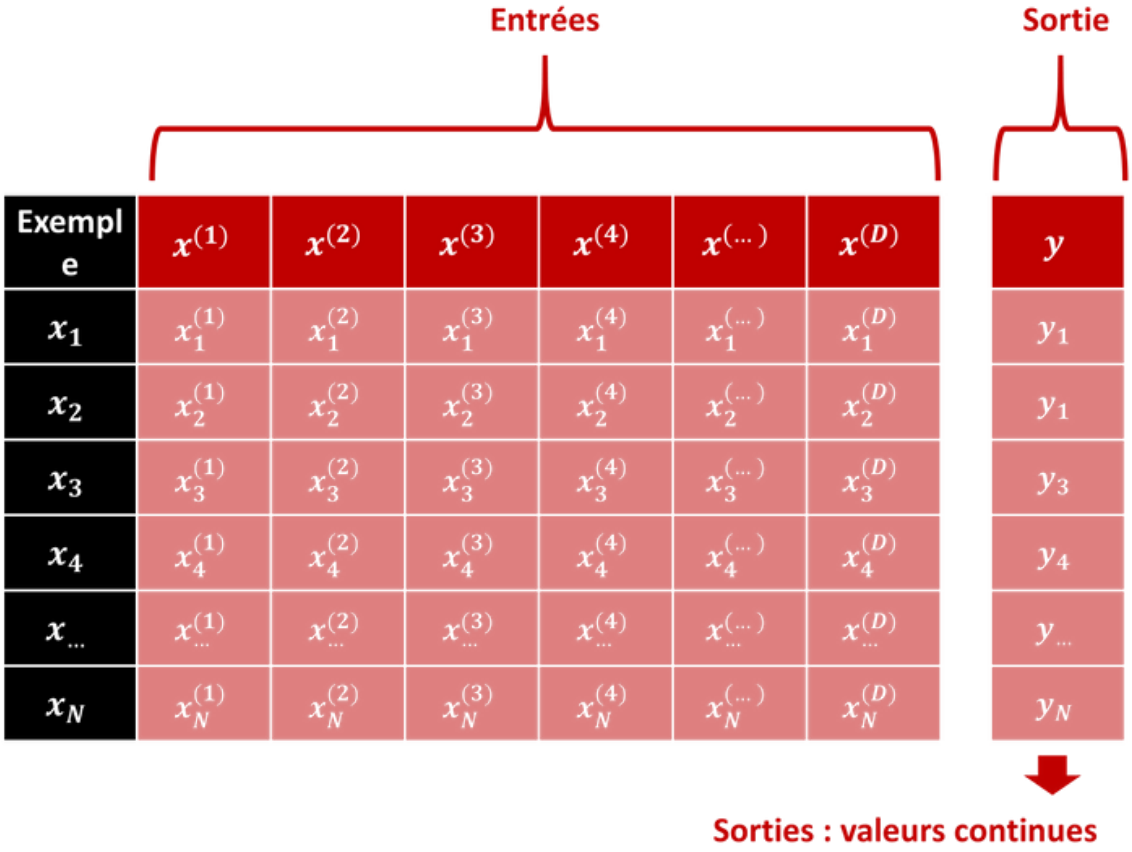
APPRENTISSAGE MACHINE SUPERVISÉ : RÉGRESSION

1. CONCEPTS FONDAMENTAUX

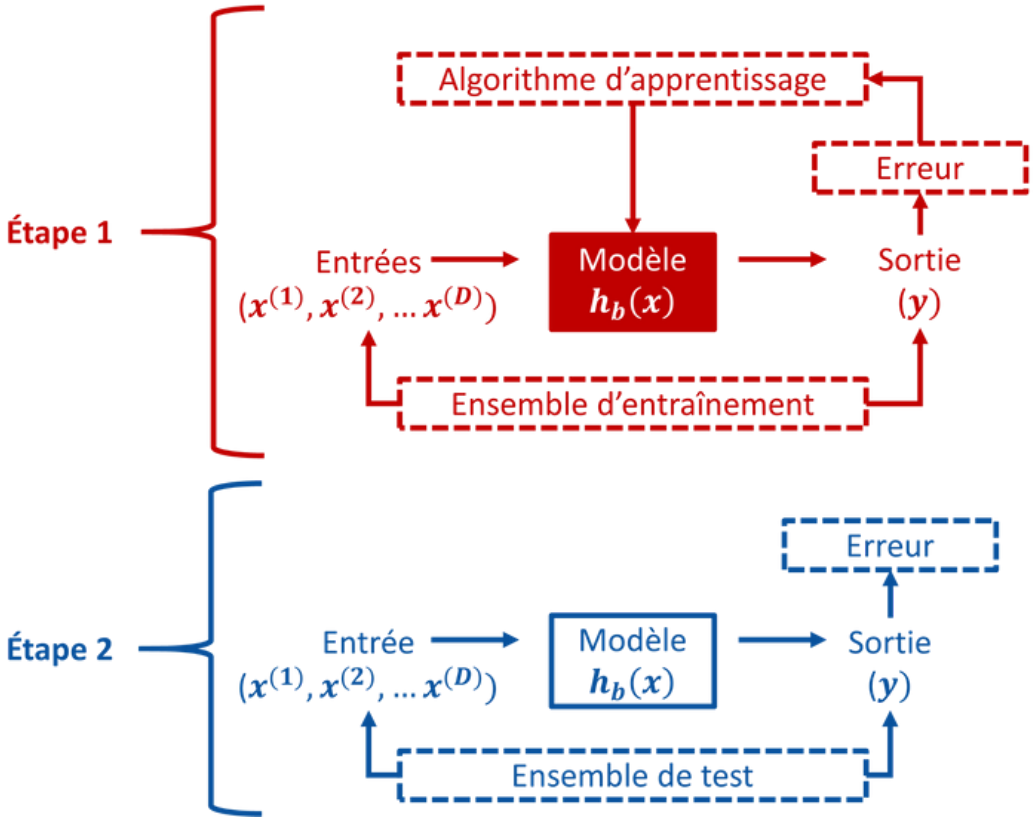



Étape 1
Algorithme d'apprentissage
Erreur
Entrées
$(x^{(1)}, x^{(2)}, \dots x^{(D)})$
Modèle
$h_b(x)$
Sortie
(y)
Ensemble d'entraînement
Étape 2
Erreur
Entrée
$(x^{(1)}, x^{(2)}, \dots x^{(D)})$
Modèle
$h_b(x)$
Sortie
(y)
Ensemble de test

1. CONCEPTS FONDAMENTAUX



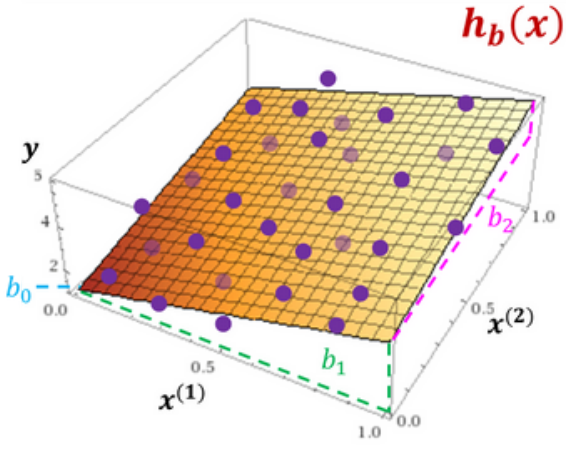
1. CONCEPTS FONDAMENTAUX



1. CONCEPTS FONDAMENTAUX

Exemple de **modèle**: régression linéaire
(plusieurs caractéristiques)

$$h_b(x) = b_0 + b_1 x^{(1)} + b_2 x^{(2)}$$

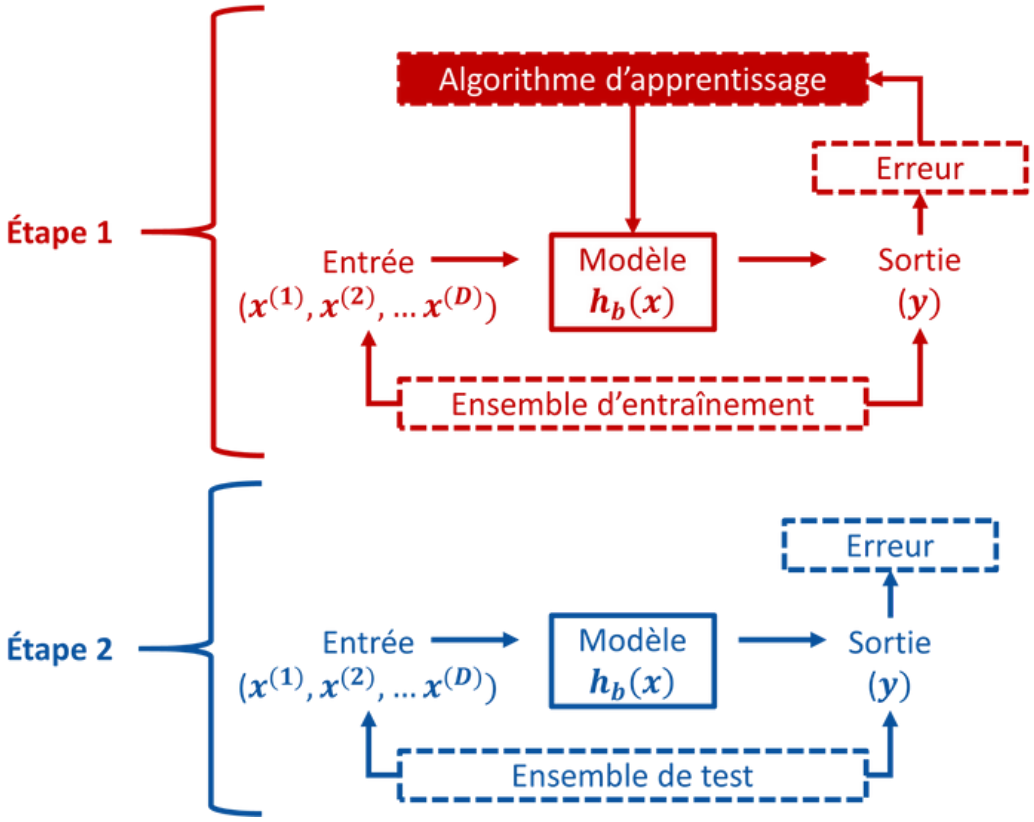


Le modèle peut aussi contenir des termes polynomiaux

$$h_b(x) = b_0 + b_1 x^{(1)} + b_2 x^{(2)} + b_3 (x^{(2)})^2 + b_4 x^{(1)} x^{(2)}$$

Ordre supérieur à 1 **Interaction**

1. CONCEPTS FONDAMENTAUX



1. CONCEPTS FONDAMENTAUX

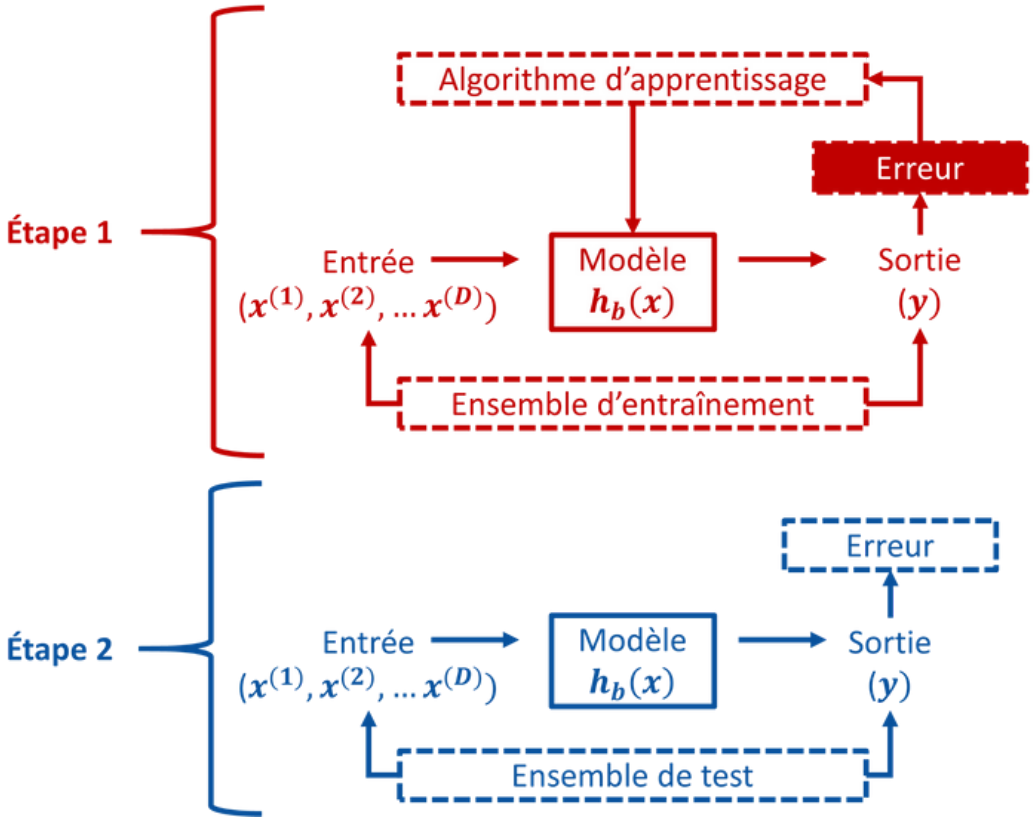
L'objectif de l'algorithme d'apprentissage est de minimiser l'erreur sur l'ensemble de test.

Pour ce faire, il doit trouver les valeurs des paramètres b_0, b_1, etc., qui permettent de minimiser l'erreur de généralisation.

Minimiser l'erreur de généralisation implique de trouver le meilleur compromis entre le biais et la variance du modèle.

Rappel:

1. CONCEPTS FONDAMENTAUX



1. CONCEPTS FONDAMENTAUX

La fonction de coût correspond à l'erreur de prédiction dans l'ensemble d'entraînement.

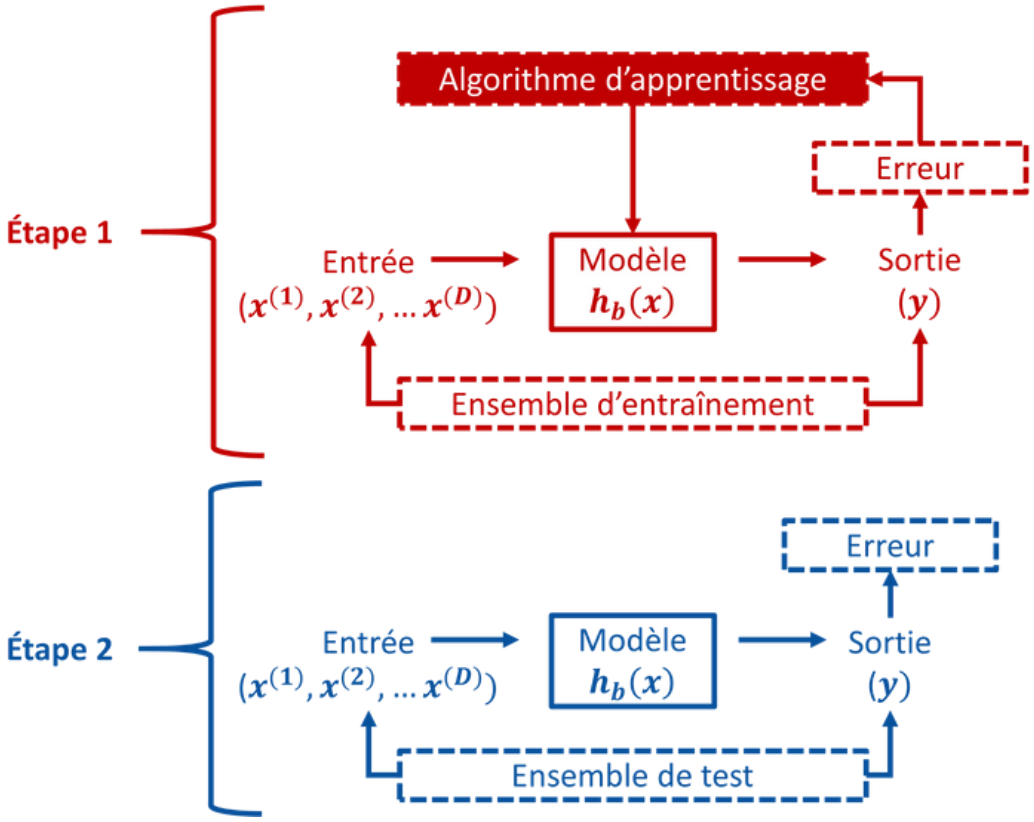
En régression linéaire, on utilise généralement la fonction de coût suivante:

$$\sum_{i=1}^{N} \left(y_i - h(\boldsymbol{x}_i)\right)^2$$

À retenir :

- **On utilise les carrés des différences entre les valeurs prédites par le modèle et les valeurs réelles parce que sinon, les différences brutes positives et négatives s'annuleraient.**

1. CONCEPTS FONDAMENTAUX



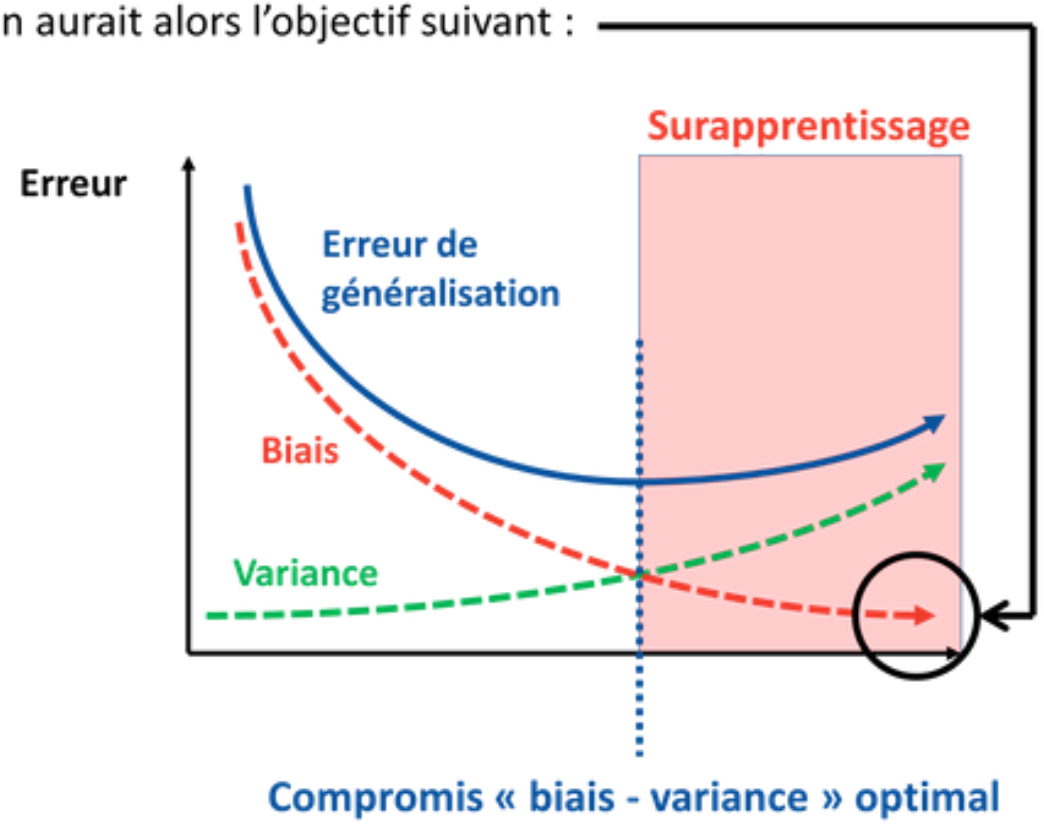
1. CONCEPTS FONDAMENTAUX

Le premier terme de la fonction de perte correspond donc à la fonction de coût suivante :

$$\sum_{i=1}^{N} \left(y_i - h(\boldsymbol{x}_i)\right)^2$$

Si on minimisait uniquement cette fonction de coût, on minimiserait le biais, sans égard pour la variance du modèle.

On aurait alors l'objectif suivant :



2. HYPERPARAMÈTRES ET RÉGULARISATION

2. HYPERPARAMÈTRES ET RÉGULARISATION

Pour tenir compte de la variabilité du modèle, on ajoute une fonction de **régularisation**.

Cette fonction a pour objectif de réduire la complexité du modèle.

Un modèle moins complexe aura moins de flexibilité pour accommoder tous les points de l'ensemble d'entraînement.

Donc, un modèle moins complexe va être :

- Plus représentatif de la tendance générale à l'intérieur de l'ensemble d'entraînement.
- Moins représentatif du bruit à l'intérieur de l'ensemble d'entraînement.

Pour ce faire, elle prend en entrées les valeurs des paramètres du modèle et envoie en sortie une valeur proportionnelle à la taille des paramètres.

On en trouve généralement trois « saveurs » :

- L2 : Ridge
- L1 : Lasso
- L2 + L1 : ElasticNet

2. HYPERPARAMÈTRES ET RÉGULARISATION

2.1. Régularisation L2 : Ridge

Dans tous les cas de régularisation, on ajoute un biais au modèle en pénalisant la taille des paramètres.

Dans la régularisation *Ridge*, la valeur de cette pénalité correspond à la somme de chacun des paramètres au carré (généralement, on n'inclut pas b_0).

On a donc la fonction de régularisation *Ridge* suivante :

$$\lambda \sum_{j=1}^{p} (b_j)^2$$

où p correspond au nombre de paramètres
et où λ est le poids que l'on souhaite accorder à cette pénalité.

λ est un **hyperparamètre**. Ceci veut dire qu'il n'est pas estimé automatiquement par l'algorithme d'apprentissage, mais qu'il doit plutôt être fixé par le chercheur.

Plus la valeur de λ est élevée, plus l'importance de la pénalité sera grande par rapport à l'importance de la fonction de coût.

- Plus la valeur de λ est élevée, moins le modèle est complexe.
- Plus la valeur de λ est élevée, plus le biais du modèle est élevé.
- Plus la valeur de λ est élevée, plus la variance du modèle est faible.

2. HYPERPARAMÈTRES ET RÉGULARISATION

2.1. Régularisation L2 : Ridge

Voici alors la fonction de perte complète :

$$J(\boldsymbol{b}) = \frac{1}{2n}\left[\sum_{i=1}^{N}\left(y_i - h(\boldsymbol{x}_i)\right)^2 + \lambda \sum_{j=1}^{p}\left(b_j\right)^2\right]$$

Notons que :

- **On calcule la moyenne parce que ça donne une mesure plus intuitive (mais ça ne change rien à l'estimation des paramètres).**
- **On divise par 2 simplement parce que ça simplifie certains calculs mathématiques utilisant la fonction de perte.**

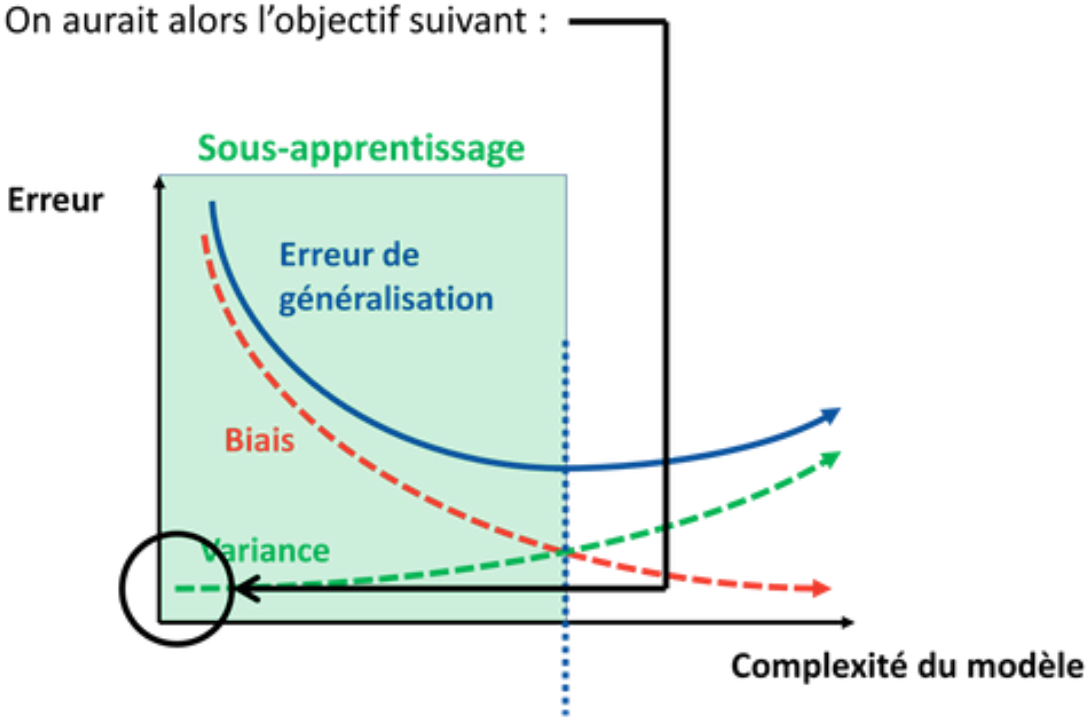
2. HYPERPARAMÈTRES ET RÉGULARISATION

2.1. Régularisation L2 : Ridge

$$J(\boldsymbol{b}) = \frac{1}{2n}\left[\sum_{i=1}^{N}\left(y_i - h(\boldsymbol{x}_i)\right)^2 + \lambda \sum_{j=1}^{p}\left(b_j\right)^2\right]$$

Si on minimisait uniquement la fonction de régularisation, on minimiserait la variance du modèle, sans égard pour le biais.

On aurait alors l'objectif suivant :

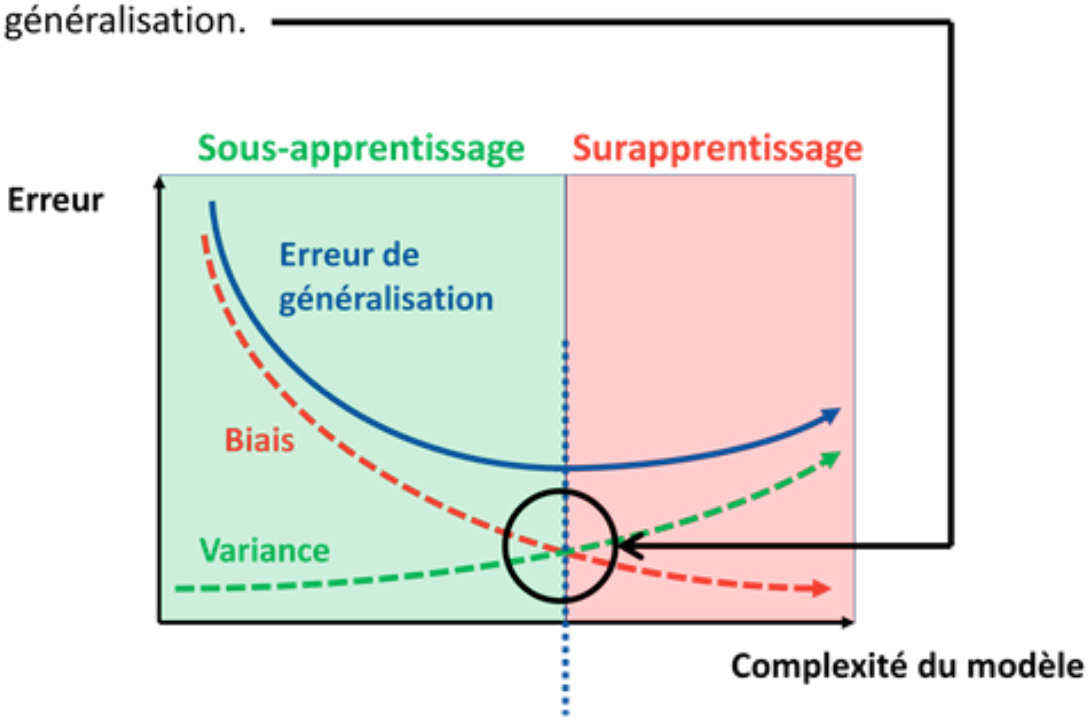


2. HYPERPARAMÈTRES ET RÉGULARISATION

2.1. Régularisation L2 : Ridge

$$J(\boldsymbol{b}) = \frac{1}{2n}\left[\sum_{i=1}^{N}\left(y_i - h(\boldsymbol{x}_i)\right)^2 + \lambda \sum_{j=1}^{p}\left(b_j\right)^2\right]$$

En créant une tension entre la minimisation du biais (fonction de coût) et la minimisation de la variance (fonction de régularisation), on obtient une **fonction de perte** ayant pour objectif de minimiser l'erreur de généralisation.

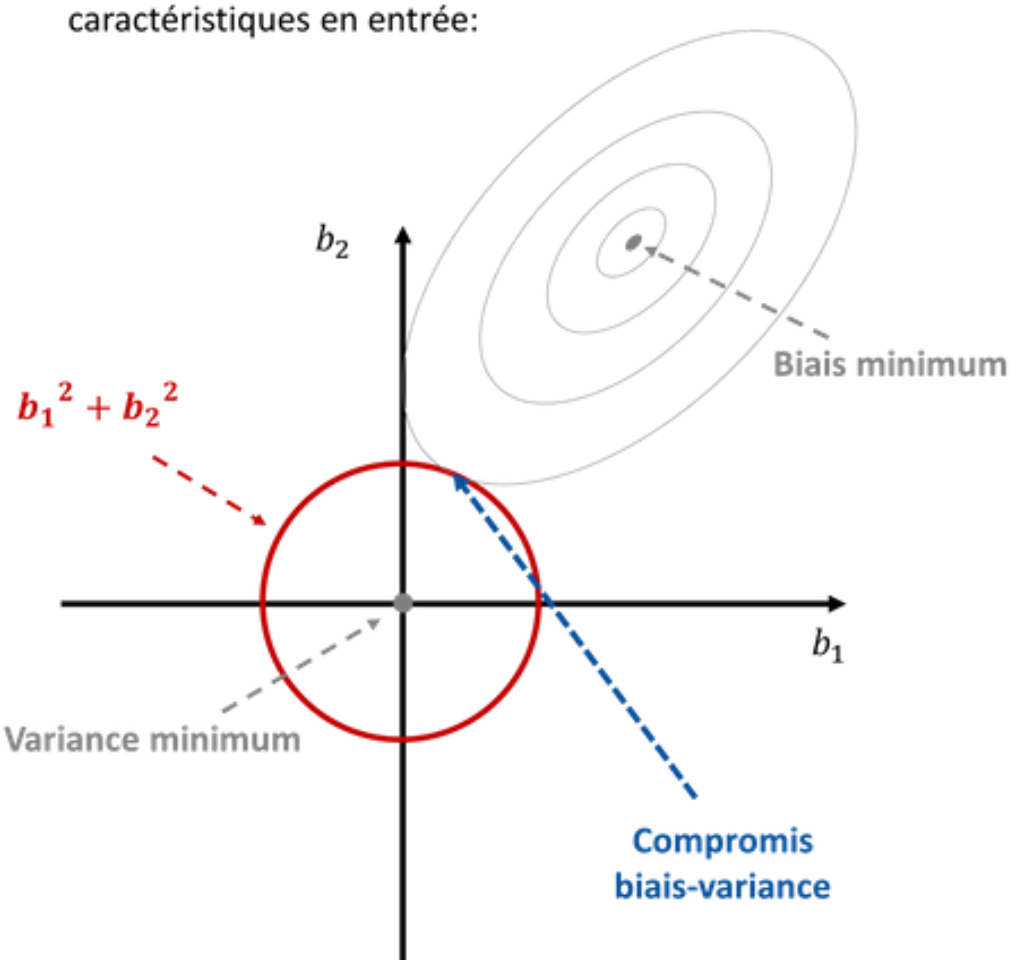


2. HYPERPARAMÈTRES ET RÉGULARISATION

2.1. Régularisation L2 : Ridge

$$J(\boldsymbol{b}) = \frac{1}{2n}\left[\sum_{i=1}^{N}\left(y_i - h(\boldsymbol{x}_i)\right)^2 + \lambda \sum_{j=1}^{p}\left(b_j\right)^2\right]$$

Illustrons le compromis dans un cas où on a deux caractéristiques en entrée:



2. HYPERPARAMÈTRES ET RÉGULARISATION

2.2. Régularisation L1 : Lasso

$$J(\boldsymbol{b}) = \frac{1}{2n}\left[\sum_{i=1}^{N}\left(y_i - h(\boldsymbol{x}_i)\right)^2 + \lambda \sum_{j=1}^{p}\left(b_j\right)^2\right]$$

La régularisation L2 (Ridge) permet ainsi de simplifier le modèle.

Toutefois, elle ne permet pas de réduire la valeur d'un paramètre à « 0 ».

Si l'on souhaite que la valeur de certains paramètres soient à « 0 », par exemple parce que l'on souhaite éliminer du modèle les variables les moins importantes, on utilisera plutôt une régularisation de type L1 (Lasso).

Régularisation L1 : Lasso

$$\lambda \sum_{j=1}^{p}\left|b_j\right|$$

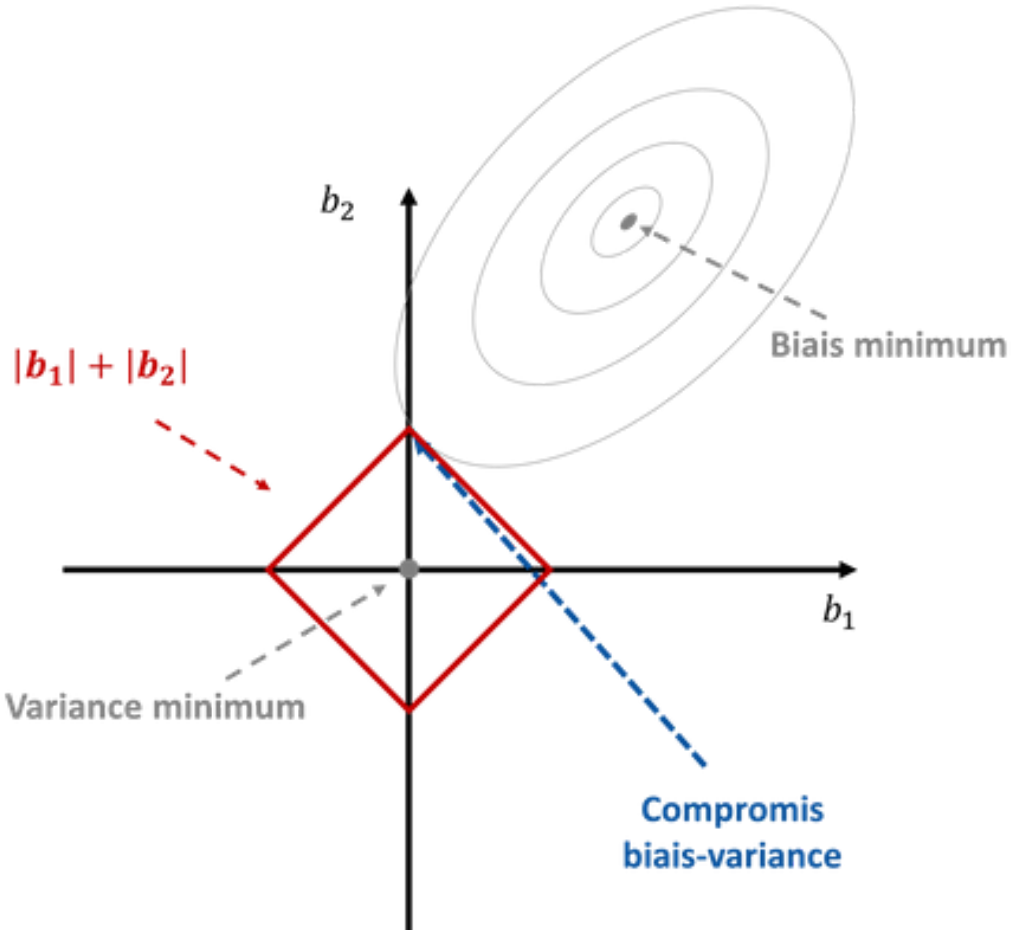
Le principe est le même que pour la régression L2, mais plutôt que d'additionner les carrés des valeurs des paramètres, on additionne les valeurs absolues.

2. HYPERPARAMÈTRES ET RÉGULARISATION

2.2. Régularisation L1 : Lasso

$$J(\boldsymbol{b}) = \frac{1}{2n}\left[\sum_{i=1}^{N}\left(y_i - h(\boldsymbol{x}_i)\right)^2 + \lambda \sum_{j=1}^{p}|b_j|\right]$$

Illustrons le compromis biais-variance avec la régularisation L1, dans un cas où on a deux caractéristiques en entrée:



2. HYPERPARAMÈTRES ET RÉGULARISATION

2.3. ElasticNet

Avantage de la régularisation L2 : conserve tous les paramètres, ce qui permet de modéliser des systèmes complexes.

Avantage de la régularisation L1 : permet de construire des modèles plus parcimonieux (plus creux).

Il est possible de combiner les deux méthodes. Cette méthode de régularisation est nommée *ElasticNet*

- L'idée est de combinée L1 et L2 tel que :

$$\lambda \sum_{j=1}^{p} \left(b_j\right)^2 + (1-\lambda) \sum_{j=1}^{p} \left|b_j\right|$$

- On a donc la fonction de perte suivante:

$$J(\boldsymbol{b}) = \frac{1}{2n}\left[\sum_{i=1}^{N} \left(y_i - h(\boldsymbol{x}_i)\right)^2 + \lambda \sum_{j=1}^{p} \left(b_j\right)^2 + (1-\lambda) \sum_{j=1}^{p} \left|b_j\right|\right]$$

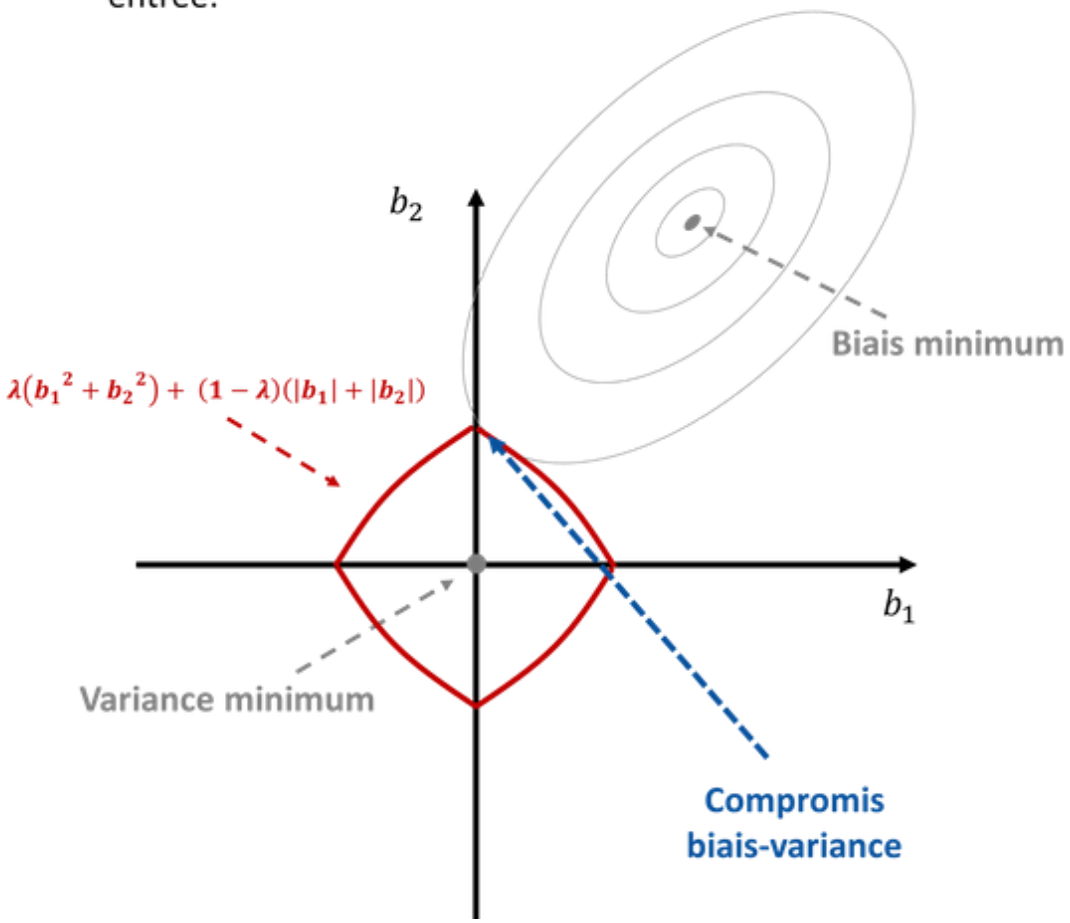
- Notez que dans *scikit-learn*, ElasticNet est défini ainsi:

$$\frac{\lambda}{2}(1 - l1\_ratio) \sum_{j=1}^{p} \left(b_j\right)^2 + \lambda \cdot l1\_ratio \sum_{j=1}^{p} \left|b_j\right|$$

2. HYPERPARAMÈTRES ET RÉGULARISATION

2.3. ElasticNet

Illustrons le compromis biais-variance avec le compromis ElasticNet, dans un cas où on a deux caractéristiques en entrée:

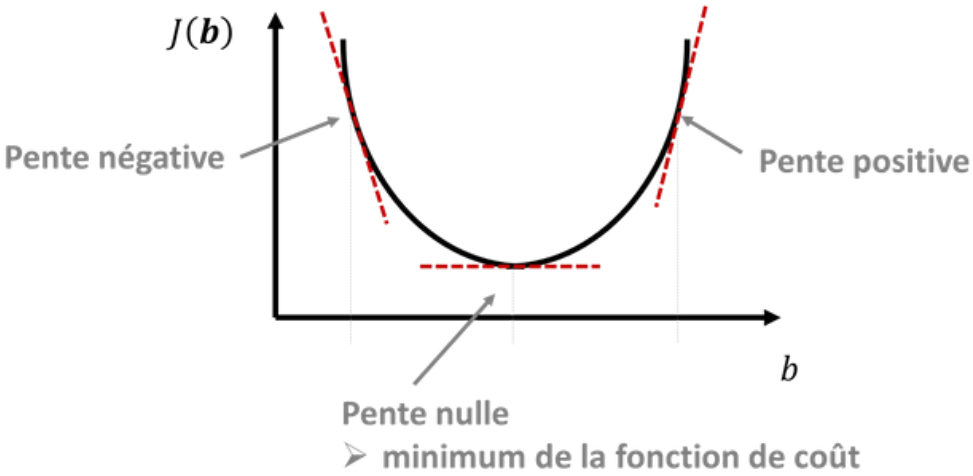


3. MÉTHODES ANALYTIQUES ET ITÉRATIVES

3. MÉTHODES ANALYTIQUES ET ITÉRATIVES

- Si l'on est à la valeur minimum de la fonction de perte, la « dérivée première » de celle-ci aura la valeur « 0 ».

- La « dérivée première » de la fonction de perte correspond à la pente de la fonction de perte au point ***b*** évalué.



3. MÉTHODES ANALYTIQUES ET ITÉRATIVES

3.1. Méthodes analytiques

On calcule la solution en une seule étape avec un algorithme.

Ex. équation normale :

- On utilise le fait que l'on cherche *b* pour que la dérivée de la fonction de coût soit égale à 0 :

$$\frac{\partial}{\partial(b)}\big(J(b)\big) = 0$$

- On utilise le calcul matriciel et on trouve :

$$\boldsymbol{b} = \left(\boldsymbol{X}^T\boldsymbol{X} + \boldsymbol{\lambda}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

$$b = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} \quad X = \begin{pmatrix} 1 & x_1^{(1)} & x_1^{(2)} & x_1^{(\dots)} & x_1^{(p)} \\ 1 & x_2^{(1)} & x_2^{(2)} & x_2^{(\dots)} & x_2^{(p)} \\ 1 & x_{\dots}^{(1)} & x_{\dots}^{(2)} & x_{\dots}^{(\dots)} & x_{\dots}^{(p)} \\ 1 & x_N^{(1)} & x_N^{(2)} & x_N^{(\dots)} & x_N^{(p)} \end{pmatrix} \quad y = \begin{pmatrix} y_0 \\ y_1 \\ y_{\dots} \\ y_N \end{pmatrix}$$

À retenir:

- Une seule étape.
- Toutefois, on doit inverser une matrice, ce qui peut être très lourd au niveau computationnel.
- Si on a beaucoup de caractéristiques (> 10 000), ça peut être très long.

3. MÉTHODES ANALYTIQUES ET ITÉRATIVES

3.2. Méthodes itératives

On essaie de s'approcher de la solution à chaque étape avec une heuristique.

Ex. descente de gradient :

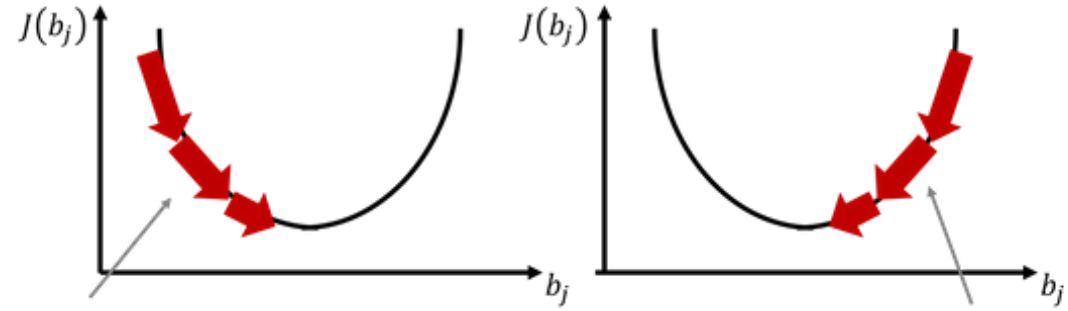
- On utilise le fait que l'on cherche *b* pour que la dérivée de la fonction de coût soit égale à 0 (comme pour a méthode analytique) :

$$\frac{\partial}{\partial(b)}\big(J(b)\big) = 0$$

- On utilise la valeur de la pente pour les valeurs de paramètres actuelles.

$$b_j := b_j - \alpha \frac{\partial}{\partial(b_j)}\big(J(b)\big)$$

- **On déplace la valeur du paramètre…**
  - **…dans la direction opposée à la pente.**
  - **…de manière proportionnelle à la pente.**



Pente négative
- **On augmente la valeur de b_j**

Pente positive
- **On diminue la valeur de b_j**

3. MÉTHODES ANALYTIQUES ET ITÉRATIVES

3.2. Méthodes itératives

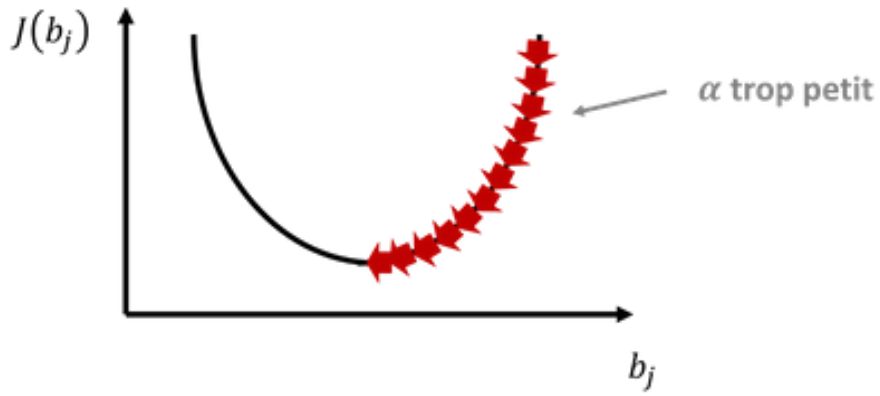
Ex. descente de gradient :

- On utilise la valeur de la pente pour les valeurs de paramètres actuelles.

$$b_j \coloneqq b_j - \boldsymbol{\alpha} \frac{\partial}{\partial(b_j)} \big(J(b)\big)$$

- On déplace la valeur du paramètre…
  - …dans la direction opposée à la pente.
  - …de manière proportionnelle à la pente.

- « α » est un hyperparamètre qui fixe la taille de la modification apportée au paramètre à chaque étape de la méthode itérative.
  - **Si α est trop petit, l'apprentissage sera très long.**
  - Si α est trop grand, l'apprentissage ne convergera pas vers une solution.



3. MÉTHODES ANALYTIQUES ET ITÉRATIVES

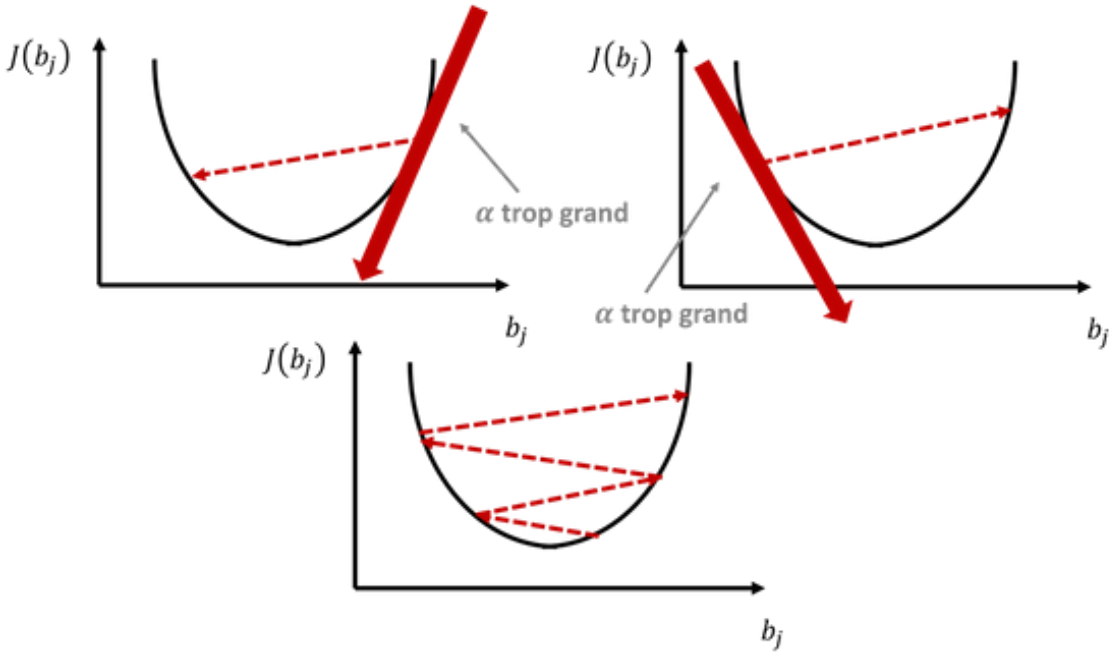
3.2. Méthodes itératives

Ex. descente de gradient :

- On utilise la valeur de la pente pour les valeurs de paramètres actuelles.

$$b_j := b_j - \alpha \frac{\partial}{\partial(b_j)}\big(J(b)\big)$$

- « α » est un hyperparamètre qui fixe la taille de la modification apportée au paramètre à chaque étape de la méthode itérative.
  - Si α est trop petit, l'apprentissage sera très long.
  - **Si α est trop grand, l'apprentissage ne convergera pas vers une solution.**



In [1]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

import pandas as pd
import numpy as np

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------


from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------

data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()

features_cols = ['PSQ_SS', 'PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITT
T', 'STAIYTT', 'AGE', 'SEXE', 'SES']

X = data.loc[:, features_cols]
y = data['WHODASTT']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_st
ate=42)

scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

# ------------------------------------------------------------------
------------
# ÉTAPE 4 : entraîner le modèle (ensemble "Entraînement")
# ------------------------------------------------------------------
------------

model = LinearRegression()

model.fit(X_train, y_train)

y_train_pred = model.predict(X_train)

mse_train = mean_squared_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)

print("lr.coef_: {}".format(model.coef_))
print("\nlr.intercept_: {}".format(model.intercept_))

print("\n\nTraining mse: ", mse_train)
print("\nTraining r2: ", r2_train)


# ------------------------------------------------------------------
------------
# ÉTAPE 5 : vérifier la généralisabilité des résultats (ensemble "T
est")
# ------------------------------------------------------------------
------------

y_test_pred = model.predict(X_test)

mse_test = mean_squared_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)

print("\n\nTest mse: ", mse_test)
print("\nTest r2: ", r2_test)
```

```
lr.coef_: [-0.01841002  0.66643157 -0.00954378  0.09255
818  0.09841242  0.03537588
  0.05306677 -0.03699934 -0.03070391]

lr.intercept_: 2.738351254480287


Training mse:  1.1467944606927987

Training r2:  0.3036194681962495


Test mse:  1.2990942585698801

Test r2:  0.16689421958822104
```

In [2]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

import pandas as pd
import numpy as np

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------

data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()

features_cols = ['PSQ_SS', 'PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITT
T', 'STAIYTT', 'AGE', 'SEXE', 'SES']

X = data.loc[:, features_cols]
y = data['WHODASTT']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_st
ate=40)
```

```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

# ------------------------------------------------------------------
------------
# ÉTAPE 4 : entraîner le modèle (ensemble "Entraînement")
# ------------------------------------------------------------------
------------

#model = LinearRegression()
#model = Ridge(alpha = 10)
model = Lasso(alpha = .1)
#model = ElasticNet(alpha=.01, l1_ratio = .05)

model.fit(X_train, y_train)

y_train_pred = model.predict(X_train)

mse_train = mean_squared_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)


print("lr.coef_: {}".format(model.coef_))
print("\nlr.intercept_: {}".format(model.intercept_))

print("\nTraining mse: ", mse_train)
print("\nTraining r2: ", r2_train)


# ------------------------------------------------------------------
------------
# ÉTAPE 5 : vérifier la généralisabilité des résultats (ensemble "T
est")
# ------------------------------------------------------------------
------------

y_test_pred = model.predict(X_test)

mse_test = mean_squared_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)

print("\nTest mse: ", mse_test)
print("\nTest r2: ", r2_test)
```

```
lr.coef_: [ 0.          0.53731972  0.01206909  0.
0.04733905  0.01432853
  0.         -0.         -0.        ]

lr.intercept_: 2.686977299874552

Training mse:  1.1006509971686493

Training r2:  0.27688106434608284

Test mse:  1.5002477010472157

Test r2:  0.22707546660129452
```

In [3]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

# Importer les librairies utiles
import pandas as pd
import numpy as np

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

# Importer une fonction qui nous permette de construire aléatoireme
nt les ensembles "Entraînement" et "Test"
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

# Importer le modèle de régression linéaire
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------

# Importons un ensemble de données
data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()

features_cols = ['PSQ_SS', 'PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITT
T', 'STAIYTT', 'AGE', 'SEXE', 'SES']

X = data.loc[:, features_cols]
y = data['WHODASTT']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.2, random_state = 42)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

# ------------------------------------------------------------------
------------
# ÉTAPE 4 : entraîner le modèle (ensemble "Entraînement")
# ------------------------------------------------------------------
------------

model = LinearRegression()
model.fit(X_train, y_train)

y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

mse_train = mean_squared_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)

print("\n\nalgorithm: vanilla")
print("\nTraining mse: ", mse_train)
print("\nTraining r2: ", r2_train)
print("\nTest mse: ", mse_test)
print("\nTest r2: ", r2_test)

mse_test_min = mse_test

for i in range(7):
    model = Ridge(alpha = 0.001*(10**i), max_iter=100000)
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
    mse_train = mean_squared_error(y_train, y_train_pred)
    r2_train = r2_score(y_train, y_train_pred)
    mse_test = mean_squared_error(y_test, y_test_pred)
    r2_test = r2_score(y_test, y_test_pred)

    if mse_test < mse_test_min:
        mse_test_min = mse_test
        print("\n\nalgorithm: Ridge")
        print("\nalpha: ", 0.001*(10**i))
        print("\nTraining mse: ", mse_train)
        print("\nTraining r2: ", r2_train)
        print("\nTest mse: ", mse_test)
        print("\nTest r2: ", r2_test)

for i in range(7):
    model = Lasso(alpha = 0.001*(10**i), max_iter=100000)
```

```
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
    mse_train = mean_squared_error(y_train, y_train_pred)
    r2_train = r2_score(y_train, y_train_pred)
    mse_test = mean_squared_error(y_test, y_test_pred)
    r2_test = r2_score(y_test, y_test_pred)

    if mse_test < mse_test_min:
        mse_test_min = mse_test
        print("\n\nalgorithm: Lasso")
        print("\nalpha: ", 0.001*(10**i))
        print("\nTraining mse: ", mse_train)
        print("\nTraining r2: ", r2_train)
        print("\nTest mse: ", mse_test)
        print("\nTest r2: ", r2_test)

for i in range(7):
    for j in range(7):
        model = ElasticNet(alpha=0.001*(10**i), l1_ratio = 0.001*(1
0**i), max_iter=100000)
        model.fit(X_train, y_train)
        by_train_pred = model.predict(X_train)
        y_test_pred = model.predict(X_test)
        mse_train = mean_squared_error(y_train, y_train_pred)
        r2_train = r2_score(y_train, y_train_pred)
        mse_test = mean_squared_error(y_test, y_test_pred)
        r2_test = r2_score(y_test, y_test_pred)


        if mse_test < mse_test_min:
            mse_test_min = mse_test
            print("\n\nalgorithm: ElasticNet")
            print("\nalpha: ", 0.001*(10**i))
            print("\nl1_ratio: ", 0.001*(10**j))
            print("\nTraining mse: ", mse_train)
            print("\nTraining r2: ", r2_train)
            print("\nTest mse: ", mse_test)
            print("\nTest r2: ", r2_test)

print("\n\n Algorithm finished")
```

```
algorithm: vanilla
Training mse:  1.1630841733813415
Training r2:  0.2884153779557871
Test mse:  1.283353999826951
Test r2:  0.17599644573815298

algorithm: Ridge
alpha:  0.001
Training mse:  1.1630841733876482
Training r2:  0.2884153779519286
Test mse:  1.28335311331578
Test r2:  0.17599701494069242

algorithm: Ridge
alpha:  0.01
Training mse:  1.1630841740119362
Training r2:  0.2884153775699839
Test mse:  1.283345135792988
Test r2:  0.17600213707155865

algorithm: Ridge
alpha:  0.1
Training mse:  1.163084236381323
Training r2:  0.2884153394118717
Test mse:  1.2832654671681114
Test r2:  0.17605328993356728

algorithm: Ridge
alpha:  1.0
```

```
Training mse:  1.1630904144198373

Training r2:  0.28841155963625287

Test mse:  1.2824793484092842

Test r2:  0.17655803348167098


algorithm: Ridge

alpha:  10.0

Training mse:  1.1636541329623664

Training r2:  0.28806667191857493

Test mse:  1.275589278570987

Test r2:  0.18098194304725834


algorithm: Ridge

alpha:  100.0

Training mse:  1.1917339727037002

Training r2:  0.27088719118390947

Test mse:  1.2573619506692244

Test r2:  0.1926851698870653


 Algorithm finished
```

4. VALIDATION

4. VALIDATION

Dans le dernier exemple, on a donc utilisé notre ensemble de test pour sélectionner les meilleures valeurs de nos hyperparamètres.

Or, ceci est strictement interdit!

En effet, on a alors sélectionné notre modèle final en nous basant, entre autres, sur l'ensemble de test.

Ce faisant, l'ensemble de test n'est plus un ensemble de données valide pour tester l'erreur de généralisation de notre modèle.

Quand on doit utiliser les données pour sélectionner les valeurs de nos hyperparamètres, on doit utiliser un troisième ensemble de données, soit un **ensemble de « validation »**.

1. Pour chaque ensemble d'hyperparamètres possible, on entraîne notre modèle sur **l'ensemble d'entraînement**.
2. On utilise **l'ensemble de validation** pour vérifier l'erreur de généralisation des différents modèles entraînés.
3. L'ensemble de valeurs des hyperparamètres qui permet de minimiser l'erreur dans **l'ensemble de validation** est sélectionné pour le modèle final.
4. Le modèle final est entraîné sur toutes les données provenant des **ensembles « entraînement + validation »**.
5. On mesure l'erreur de généralisation finale sur **l'ensemble de test**.

Faisons à nouveau le dernier exemple dans scikit-learn, mais maintenant avec trois ensembles de données :

1. Un ensemble d’entraînement.

2. Un ensemble de validation.

3. Un ensemble de test.

In [4]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

import pandas as pd
import numpy as np

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------

data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()

features_cols = ['PSQ_SS', 'PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITT
T', 'STAIYTT', 'AGE', 'SEXE', 'SES']

X = data.loc[:, features_cols]
y = data['WHODASTT']


X_train_full, X_test, y_train_full, y_test = train_test_split(X, y,
test_size = 0.2, random_state = 0)
```

```
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full,
y_train_full, test_size = 0.2, random_state = 0)


scaler = StandardScaler()

X_train_full = scaler.fit_transform(X_train_full)
X_train = scaler.fit_transform(X_train)
X_valid = scaler.fit_transform(X_valid)
X_test = scaler.fit_transform(X_test)

# ----------------------------------------------------------------
------------
# ÉTAPE 4 : entraîner le modèle (ensemble "Entraînement")
# ----------------------------------------------------------------
------------

# Linear regression : vanilla

model = LinearRegression()
model.fit(X_train, y_train)

y_train_pred = model.predict(X_train)
y_valid_pred = model.predict(X_valid)

mse_train = mean_squared_error(y_train, y_train_pred)
mse_valid = mean_squared_error(y_valid, y_valid_pred)


mse_valid_min = mse_valid

print("\n\nalgorithm: vanilla")
print("\nTraining mse: ", mse_train)
print("\nValidation mse: ", mse_valid)

# Linear regression : Ridge

for i in range(7):

    model = Ridge(alpha = 0.001*(10**i), max_iter=100000)
    model.fit(X_train, y_train)

    y_train_pred = model.predict(X_train)
    y_valid_pred = model.predict(X_valid)

    mse_train = mean_squared_error(y_train, y_train_pred)
    mse_valid = mean_squared_error(y_valid, y_valid_pred)

    if mse_valid < mse_valid_min:
        mse_valid_min = mse_valid
        print("\n\nalgorithm: Ridge")
        print("\nalpha: ", 0.001*(10**i))
        print("\nTraining mse: ", mse_train)
        print("\nValidation mse: ", mse_valid)
```

```
# Linear regression : Lasso

for i in range(7):

    model = Lasso(alpha = 0.001*(10**i), max_iter=100000)
    model.fit(X_train, y_train)

    y_train_pred = model.predict(X_train)
    y_valid_pred = model.predict(X_valid)

    mse_train = mean_squared_error(y_train, y_train_pred)
    mse_valid = mean_squared_error(y_valid, y_valid_pred)

    if mse_valid < mse_valid_min:
        mse_valid_min = mse_valid
        print("\n\nalgorithm: Lasso")
        print("\nalpha: ", 0.001*(10**i))
        print("\nTraining mse: ", mse_train)
        print("\nValidation mse: ", mse_valid)

# Linear regression : ElasticNet

for i in range(7):
    for j in range(7):

        model = ElasticNet(alpha=0.001*(10**i), l1_ratio = 0.001*(1
0**i), max_iter=100000)
        model.fit(X_train, y_train)

        y_train_pred = model.predict(X_train)
        y_valid_pred = model.predict(X_valid)

        mse_train = mean_squared_error(y_train, y_train_pred)
        mse_valid = mean_squared_error(y_valid, y_valid_pred)


        if mse_valid < mse_valid_min:
            mse_valid_min = mse_valid
            print("\n\nalgorithm: ElasticNet")
            print("\nalpha: ", 0.001*(10**i))
            print("\nl1_ratio: ", 0.001*(10**j))
            print("\nTraining mse: ", mse_train)
            print("\nValidation mse: ", mse_valid)

print("\n\n Algorithm finished")
```

```
algorithm: vanilla

Training mse:  1.1175284511510861

Validation mse:  1.336501087193961


 Algorithm finished
```

In [5]:

```
# ----------------------------------------------------------------
------------
# ÉTAPE 5 : vérifier la généralisabilité des résultats (ensemble "T
est")
# ----------------------------------------------------------------
------------

model = LinearRegression()
model.fit(X_train_full, y_train_full)

y_train_full_pred = model.predict(X_train_full)
y_test_pred = model.predict(X_test)

mse_train_full = mean_squared_error(y_train_full, y_train_full_pred
)
r2_train_full = r2_score(y_train_full, y_train_full_pred)

mse_test = mean_squared_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)

print("\nTraining mse: ", mse_train_full)
print("\nTraining r2: ", r2_train_full)

print("\nTest mse: ", mse_test)
print("\nTest r2: ", r2_test)

print("\nmodel.coef_: {}".format(model.coef_))
print("\nmodel.intercept_: {}".format(model.intercept_))
```

```
Training mse:  1.140982939946361

Training r2:  0.29749336618889777

Test mse:  1.531917471677924

Test r2:  -0.42952797181145086

model.coef_: [-0.01917877  0.64293176  0.07429414  0.04
972269  0.08556524  0.04095564
  0.05801186 -0.02191542 -0.03891519]

model.intercept_: 2.8771043771111113
```

In [6]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

import pandas as pd
import numpy as np

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import SGDRegressor

from sklearn.metrics import mean_squared_error

# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------

# Importons un ensemble de données
data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()

features_cols = ['PSQ_SS', 'PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITT
T', 'STAIYTT', 'AGE', 'SEXE', 'SES']

X = data.loc[:, features_cols]
y = data['WHODASTT']


X_train_full, X_test, y_train_full, y_test = train_test_split(X, y,
test_size = 0.2, random_state = 42)
```

```
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full,
y_train_full, test_size = 0.2, random_state = 42)

scaler = StandardScaler()

X_train_full = scaler.fit_transform(X_train_full)
X_train = scaler.fit_transform(X_train)
X_valid = scaler.fit_transform(X_valid)
X_test = scaler.fit_transform(X_test)

# ------------------------------------------------------------------
------------
# ÉTAPE 4 : entraîner le modèle (ensemble "Entraînement")
# ------------------------------------------------------------------
------------


# Linear regression : vanilla

model = LinearRegression()
model.fit(X_train, y_train)

y_train_pred = model.predict(X_train)
y_valid_pred = model.predict(X_valid)

mse_train = mean_squared_error(y_train, y_train_pred)
mse_valid = mean_squared_error(y_valid, y_valid_pred)


mse_valid_min = mse_valid

print("\n\nalgorithm: vanilla")
print("\nTraining mse: ", mse_train)
print("\nValidation mse: ", mse_valid)

# Linear regression : Ridge

for i in range(7):

    model = SGDRegressor(alpha = 0.0001*(10**i), max_iter=100000, l
earning_rate='constant', eta0=.100)
    model.fit(X_train, y_train)

    y_train_pred = model.predict(X_train)
    y_valid_pred = model.predict(X_valid)

    mse_train = mean_squared_error(y_train, y_train_pred)
    mse_valid = mean_squared_error(y_valid, y_valid_pred)

    if mse_valid < mse_valid_min:
        mse_valid_min = mse_valid
        print("\n\nalgorithm: Ridge")
        print("\nalpha: ", 0.001*(10**i))
        print("\nTraining mse: ", mse_train)
```

```
        print("\nValidation mse: ", mse_valid)

print("\n\n Algorithm finished")
```

```
algorithm: vanilla

Training mse:  1.1650665053318559

Validation mse:  1.1983958143628217


 Algorithm finished
```

4. VALIDATION

4.1. Validation croisée

La procédure utilisée dans le dernier exemple était maintenant valide.

Toutefois, on se rend compte que le nombre d'exemples disponibles pour entraîner initialement le modèle et sélectionner les hyperparamètres descend rapidement.

Une méthode permettant de palier ce problème est la **validation croisée**.

La validation croisée permet de **réaliser l'entraînement et la validation avec le même ensemble de données**.

- Ainsi, toutes ces données contribuent à l'estimation des paramètres **et** à l'estimation des hyperparamètres.

4. VALIDATION

4.1. Validation croisée

Version avec **« k-plis »** *(k-fold)*.

- Pour expliquer la méthode, prenons directement un exemple. Considérons qu'on a un échantillon de 100 données.

4. VALIDATION

4.1. Validation croisée

Version avec **« k-plis »** *(k-fold)*.

- Pour expliquer la méthode, prenons directement un exemple. Considérons qu'on a un échantillon de 100 données.

On recommande généralement entre 5 et 10 plis.

- Pour les cas où on a très peu de données, on peu utiliser la version où **k = n** (*leave one out*).

  - Cependant, comme tous les sous-groupes de l'échantillon sont alors très similaires (chacun possède n-1 observations), le modèle obtenu risque d'être très représentatif de cet échantillon, mais de se généraliser plutôt mal à d'autres échantillons.

  - On augmente ainsi la variance des modèles qui seraient obtenus à l'aide de différents échantillons.

4. VALIDATION

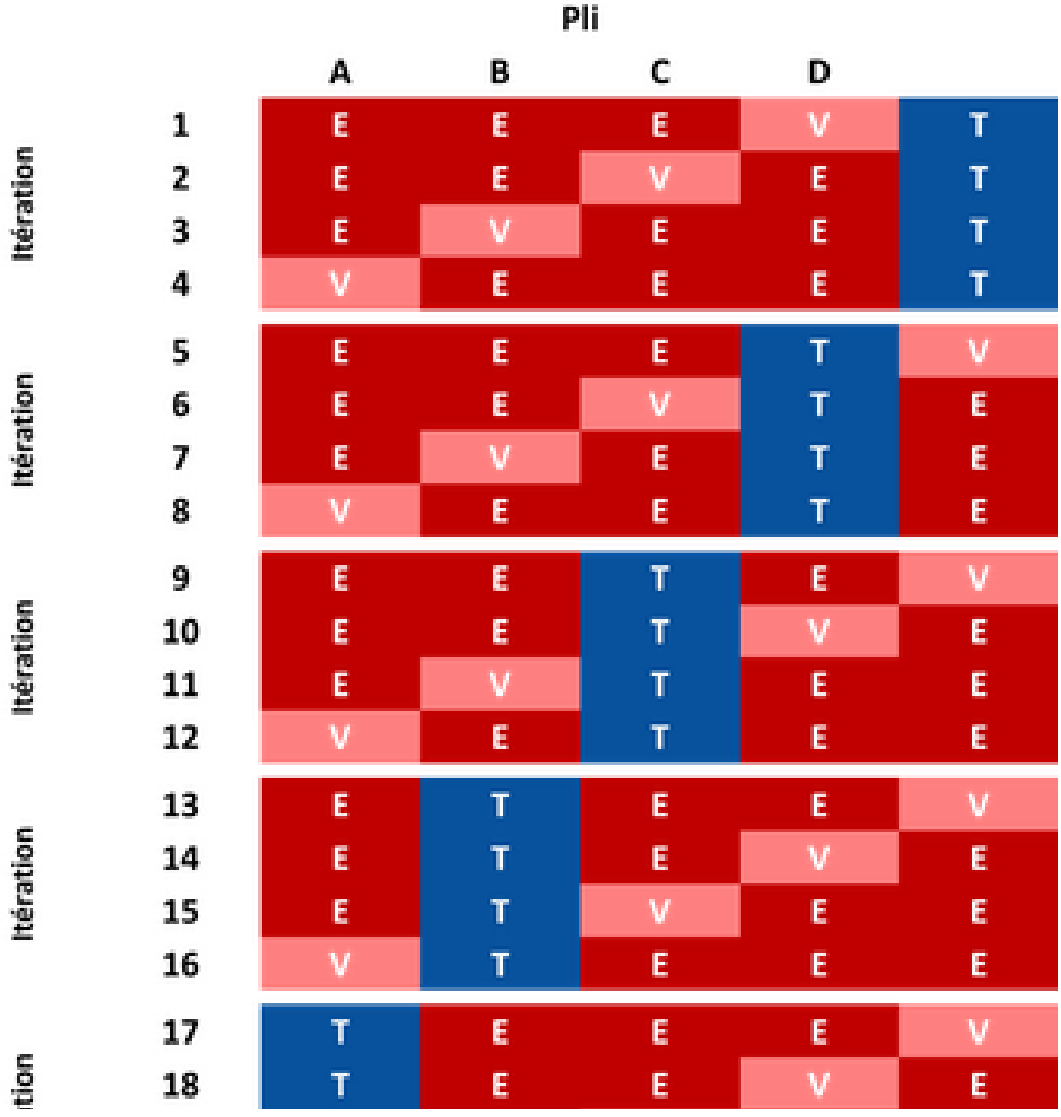
4.2. Validation croisée nichée

4. VALIDATION

4.2. Validation croisée nichée

Dans la validation croisée **nichée**, à l'intérieur de chaque itération :

- Seules les donnée d'entraînement (E) servent à ajuster les paramètres.
- Seules les données de validation (V) servent à ajuster les hyperparamètres.
- Les données test (T) ne sont utilisées pour aucune décision et sont conservées seulement pour l'évaluation finale.



Faisons à nouveau le même exemple avec scikit-learn,
mais maintenant avec de la validation croisée à 5 plis.

In [7]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

import pandas as pd
import numpy as np

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

from sklearn.model_selection import train_test_split

from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet

from sklearn.model_selection import cross_val_score

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------

data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()

features_cols = ['PSQ_SS', 'PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITT
T', 'STAIYTT', 'AGE', 'SEXE', 'SES']

X = data.loc[:, features_cols]
y = data['WHODASTT']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.2, random_state = 42)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

print(np.mean(X_train[:,0]))
print(np.mean(X_train[:,0]))


# ------------------------------------------------------------------
------------
# ÉTAPE 4 : entraîner le modèle (ensemble "Entraînement")
# ------------------------------------------------------------------
------------

# Linear regression : vanilla

model = LinearRegression()

r2_scores = cross_val_score(model, X_train, y_train, cv=5)

r2_max = np.mean(r2_scores)

print("\n\nalgorithm: vanilla")
print("\nMoyenne r2: ", r2_max)

# Linear regression : Ridge

for i in range(7):

    model = Ridge(alpha = 0.001*(10**i), max_iter=100000)

    r2_scores = cross_val_score(model, X_train, y_train, cv=5)

    if np.mean(r2_scores) > r2_max:
        r2_max = np.mean(r2_scores)
        print("\n\nalgorithm: Ridge")
        print("\nalpha: ", 0.001*(10**i))
        print("\nMoyenne r2: ", r2_max)

# Linear regression : Lasso

for i in range(7):

    model = Lasso(alpha = 0.001*(10**i), max_iter=100000)

    r2_scores = cross_val_score(model, X_train, y_train, cv=5)

    if np.mean(r2_scores) > r2_max:
        r2_max = np.mean(r2_scores)
        print("\n\nAlgorithme: Lasso")
        print("\nalpha: ", 0.001*(10**i))
```

```
        print("\nMoyenne r2: ", r2_max)

# Linear regression : ElasticNet

for i in range(7):
    for j in range(7):

        model = ElasticNet(alpha=0.001*(10**i), l1_ratio = 0.001*(1
0**i), max_iter=100000)

        r2_scores = cross_val_score(model, X_train, y_train, cv=5)

        if np.mean(r2_scores) > r2_max:
            r2_max = np.mean(r2_scores)
            print("\n\nAlgorithm: ElasticNet")
            print("\nalpha: ", 0.001*(10**i))
            print("\nl1_ratio: ", 0.001*(10**j))
            print("\nMoyenne r2: ", r2_max)

print("\n\n Algorithm finished")
```

-6.878149378149117e-17
-6.878149378149117e-17

algorithm: vanilla

Moyenne r2: 0.2568900458282938

algorithm: Ridge

alpha: 0.001

Moyenne r2: 0.2568901755047025

algorithm: Ridge

alpha: 0.01

Moyenne r2: 0.2568913419443709

algorithm: Ridge

alpha: 0.1

Moyenne r2: 0.2569029422703751

algorithm: Ridge

alpha: 1.0

Moyenne r2: 0.2570126198480135

algorithm: Ridge

alpha: 10.0

Moyenne r2: 0.2575499123039169

Algorithme: Lasso

alpha: 0.01

Moyenne r2: 0.26116564900440914

Algorithme: Lasso

alpha: 0.1

```
Moyenne r2:  0.26532740344768324


 Algorithm finished
```

In [8]:

```
# ------------------------------------------------------------------
------------
# ÉTAPE 5 : vérifier la généralisabilité des résultats (ensemble "T
est")
# ------------------------------------------------------------------
------------

alpha_final = .1

model = Lasso(alpha = alpha_final)
model.fit(X_train, y_train)

y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

r2_train = r2_score(y_train, y_train_pred)

r2_test = r2_score(y_test, y_test_pred)

print("\nTraining r2: ", r2_train)
print("\nTest r2: ", r2_test)

print("\nmodel.coef_: {}".format(model.coef_))
print("\nmodel.intercept_: {}".format(model.intercept_))
```

```
Training r2:  0.2738562589584391

Test r2:  0.16058111528906427

model.coef_: [ 0.         0.5705757  0.         0.
0.0269514  0.
  0.        -0.        -0.       ]

model.intercept_: 2.7542087542053872
```

In [9]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ------------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ------------------------------------------------------------------
------------

import pandas as pd
import numpy as np

# ------------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ------------------------------------------------------------------
------------

from sklearn.model_selection import train_test_split

from sklearn import preprocessing

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler


from sklearn.metrics import mean_squared_error

# ------------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ------------------------------------------------------------------
------------

data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()

features_cols = ['PSQ_SS', 'PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITT
T', 'STAIYTT', 'AGE', 'SEXE', 'SES']

X = data.loc[:, features_cols]
y = data['WHODASTT']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.2, random_state = 42)

# ----------------------------------------------------------------
------------
# ÉTAPE 4 : entraîner le modèle (ensemble "Entraînement")
# ----------------------------------------------------------------
------------

# Linear regression : vanilla

k_folds = 10

# vanilla

lr = Pipeline(steps = [('scaler', StandardScaler()), ('model', Line
arRegression())])
lr_score = cross_val_score(lr, X_train, y_train, cv=k_folds)
print('\n\nVanilla')
print('\nScore r2 = ', np.mean(lr_score))

# Ridge

model = Ridge(max_iter = 100000)
pipe = Pipeline(steps = [('scaler', StandardScaler()), ('model', mo
del)])
params = {'model__alpha':[.0001, .001, .01, .1, 1, 10, 100, 1000, 1
0000]}
grid = GridSearchCV(estimator=pipe, param_grid=params, cv=k_folds)
grid_result = grid.fit(X_train, y_train)
best_params = grid_result.best_params_
best_score = grid_result.best_score_
print('\n\nRidge')
print('\nMeilleur paramètre: ', best_params)
print('\nScore r2 = ', best_score)


# Lasso

model = Lasso()
pipe = Pipeline(steps = [('scaler', StandardScaler()), ('model', mo
del)])
params = {'model__alpha':[.0001, .001, .01, .1, 1, 10, 100, 1000, 1
0000]}
grid = GridSearchCV(estimator=pipe, param_grid=params, cv=k_folds)
grid_result = grid.fit(X_train, y_train)
best_params = grid_result.best_params_
best_score = grid_result.best_score_
print('\nLasso')
print('\nMeilleur paramètre: ', best_params)
print('\nScore r2 = ', best_score)

# ElasticNet
```

```
model = ElasticNet(max_iter=100000)
pipe = Pipeline(steps = [('scaler', StandardScaler()), ('model', mo
del)])
params = {'model__alpha':[.0001, .001, .01, .1, 1, 10, 100, 1000, 1
0000],\
         'model__l1_ratio':[.0001, .001, .01, .1, 1, 10, 100, 1000,
10000]}
grid = GridSearchCV(estimator=pipe, param_grid=params, cv=k_folds)
grid_result = grid.fit(X_train, y_train)
best_params = grid_result.best_params_
best_score = grid_result.best_score_
print('\nElasticNet')
print('\nMeilleur paramètre: ', best_params)
print('\nScore r2 = ', best_score)
```

```
Vanilla

Score r2 =  0.22533068684915597


Ridge

Meilleur paramètre:  {'model__alpha': 10}

Score r2 =  0.2267699909925829

Lasso

Meilleur paramètre:  {'model__alpha': 0.1}

Score r2 =  0.2425834037291083

ElasticNet

Meilleur paramètre:  {'model__alpha': 0.0001, 'model__l
1_ratio': 1000}

Score r2 =  0.2483676562070778
```

In [10]:

```
# ------------------------------------------------------------------
------------
# ÉTAPE 5 : vérifier la généralisabilité des résultats (ensemble "T
est")
# ------------------------------------------------------------------
------------

alpha_final = .1

model = Lasso(alpha = alpha_final)
model.fit(X_train_full, y_train_full)

y_train_full_pred = model.predict(X_train_full)
y_test_pred = model.predict(X_test)

r2_train_full = r2_score(y_train_full, y_train_full_pred)

r2_test = r2_score(y_test, y_test_pred)

print("\nTraining r2: ", r2_train_full)
print("\nTest r2: ", r2_test)

print("\nmodel.coef_: {}".format(model.coef_))
print("\nmodel.intercept_: {}".format(model.intercept_))
```

```
Training r2:  0.2738562589584391

Test r2:  -26.791044602653088

model.coef_: [ 0.          0.5705757   0.          0.
0.0269514   0.
  0.         -0.         -0.        ]

model.intercept_: 2.7542087542053872
```

Faisons à nouveau le même exemple, mais maintenant avec de la validation croisée nichée :

- Entrainement/validation : 4 plis.
- « Entraînement + Validation » / test : 5 plis.

In [11]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

import pandas as pd
import numpy as np

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

from sklearn.model_selection import train_test_split

from sklearn import preprocessing

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean_squared_error

# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------

data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()

features_cols = ['PSQ_SS', 'PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITT
T', 'STAIYTT', 'AGE', 'SEXE', 'SES']

X = data.loc[:, features_cols]
y = data['WHODASTT']
```

```
# ------------------------------------------------------------------
------------
# ÉTAPE 4 : entraîner le modèle (ensemble "Entraînement")
# ------------------------------------------------------------------
------------

k_folds = 5

# Ridge

model = Ridge(max_iter = 100000)
pipe = Pipeline(steps = [('scaler', StandardScaler()), ('model', mo
del)])
params = {'model__alpha':[.0001, .001, .01, .1, 1, 10, 100, 1000, 1
0000]}
grid = GridSearchCV(estimator=pipe, param_grid=params, cv=k_folds)
grid_result = grid.fit(X, y)
best_params = grid_result.best_params_
best_score = grid_result.best_score_
print('\n\nRidge')
print('\nMeilleur paramètre: ', best_params)
print('\nScore r2 = ', best_score)

outer_cv = cross_val_score(grid, X, y, cv = 5)

print('\nTest r2 = ', np.mean(outer_cv))

# Lasso

model = Lasso()
pipe = Pipeline(steps = [('scaler', StandardScaler()), ('model', mo
del)])
params = {'model__alpha':[.0001, .001, .01, .1, 1, 10, 100, 1000, 1
0000]}
grid = GridSearchCV(estimator=pipe, param_grid=params, cv=k_folds)
grid_result = grid.fit(X_train, y_train)
best_params = grid_result.best_params_
best_score = grid_result.best_score_
print('\n\nLasso')
print('\nMeilleur paramètre: ', best_params)
print('\nScore r2 = ', best_score)

outer_cv = cross_val_score(grid, X, y, cv = 5)

print('\nTest r2 = ', np.mean(outer_cv))

# ElasticNet

model = ElasticNet()
pipe = Pipeline(steps = [('scaler', StandardScaler()), ('model', mo
del)])
params = {'model__alpha':[.0001, .001, .01, .1, 1, 10, 100, 1000, 1
0000],\
          'model__l1_ratio':[.0001, .001, .01, .1, 1, 10, 100, 1000,
10000]}
```

```
grid = GridSearchCV(estimator=pipe, param_grid=params, cv=k_folds)
grid_result = grid.fit(X_train, y_train)
best_params = grid_result.best_params_
best_score = grid_result.best_score_
print('\n\nElasticNet')
print('\nMeilleur paramètre: ', best_params)
print('\nScore r2 = ', best_score)

outer_cv = cross_val_score(grid, X, y, cv = 5)

print('\nTest r2 = ', np.mean(outer_cv))
```

```
Ridge

Meilleur paramètre:  {'model__alpha': 100}

Score r2 =  0.11714939013664122

Test r2 =  0.09502921599882315


Lasso

Meilleur paramètre:  {'model__alpha': 0.1}

Score r2 =  0.26586655375768675

Test r2 =  0.12546046134417047


ElasticNet

Meilleur paramètre:  {'model__alpha': 0.0001, 'model__l
1_ratio': 1000}

Score r2 =  0.2734796835644329

Test r2 =  0.1302080201240685
```

In [12]:

```
# ------------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ------------------------------------------------------------------
------------

# Importer les librairies utiles
import pandas as pd
import numpy as np

# ------------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ------------------------------------------------------------------
------------

# Importer les fonctions de prétraitement
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

# Importer une fonction qui nous permette de construire aléatoireme
nt les ensembles "Entraînement" et "Test"
from sklearn.model_selection import train_test_split

# Importer le modèle de régression logistique de sklearn
from sklearn.linear_model import LogisticRegression

# Importer la fonction de validation croisée
from sklearn.model_selection import cross_val_score

# Importer la fonction permettant d'afficher le rapport de classifi
cation
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score


# ------------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ------------------------------------------------------------------
------------

# Importons un ensemble de données
data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()

features_cols = ['PSQ_SS', 'PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITT
T', 'STAIYTT', 'AGE', 'SEXE', 'SES']
```

```
X = data.loc[:, features_cols]
y = data['WHODASTTB']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.2, random_state = 42)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

# ------------------------------------------------------------------
------------
# ÉTAPE 4 : définir et entraîner le modèle
# ------------------------------------------------------------------
------------

# Définir le modèle
model = LogisticRegression(solver='lbfgs')

# Entraîner le modèle
model.fit(X_train, y_train)

print('\n\n\n\nPHASE ENTRAÎNEMENT\n\n')

y_train_pred = model.predict(X_train)

cfm = confusion_matrix(y_train, y_train_pred)

accuracy  = accuracy_score(y_train, y_train_pred)
precision = precision_score(y_train, y_train_pred)
recall    = recall_score(y_train, y_train_pred)
f1        = f1_score(y_train, y_train_pred)
auc       = roc_auc_score(y_train, y_train_pred)

print('Confusion matrix: \n\n', cfm, '\n\n')

print('Justesse: \n\n', accuracy, '\n\n')
print('Précision: \n\n', precision, '\n\n')
print('Rappel: \n\n', recall, '\n\n')
print('Score F1: \n\n', f1, '\n\n')
print('AUC: \n\n', auc, '\n\n')


# ------------------------------------------------------------------
------------
# ÉTAPE 5 : vérifier la généralisabilité des résultats (ensemble "T
est")
# ------------------------------------------------------------------
------------

print('\n\n\n\nPHASE TEST\n\n')

y_test_pred = model.predict(X_test)
```

```
cfm = confusion_matrix(y_test, y_test_pred)

accuracy  = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred)
recall    = recall_score(y_test, y_test_pred)
f1        = f1_score(y_test, y_test_pred)
auc       = roc_auc_score(y_test, y_test_pred)

print('Confusion matrix: \n\n', cfm, '\n\n')

print('Justesse: \n\n', accuracy, '\n\n')
print('Précision: \n\n', precision, '\n\n')
print('Rappel: \n\n', recall, '\n\n')
print('Score F1: \n\n', f1, '\n\n')
print('AUC: \n\n', auc, '\n\n')
```

PHASE ENTRAÏNEMENT

Confusion matrix:

```
 [[ 97  45]
 [ 49 106]]
```

Justesse:

0.6835016835016835

Précision:

0.7019867549668874

Rappel:

0.6838709677419355

Score F1:

0.6928104575163399

AUC:

0.6834847796456156

PHASE TEST

Confusion matrix:

```
 [[23 18]
 [12 22]]
```

Justesse:

0.6

Précision:

0.55

Rappel:

0.6470588235294118

Score F1:

0.5945945945945946

AUC:

0.6040172166427547