

Cahier thématique

Regression lineaire avec cross validation et évaluation de performance

**Manipulation 1**: Utilisation d’un algorithme de régression linéaire avec plusieurs prédicteurs

**Préparé par :** Hafed Benteftifa  
  
© Hafed Benteftifa et Nesrine Zemirli 2015-2016

Ce document ne peut être utilisé dans le cadre d’une formation, publication papier, site internet ou tout support sans mon accord express.

Aucune reproduction, même partielle, ne peut être faite de ce document et de l'ensemble de son contenu : textes, images, etc. sans mon autorisation express. Pour toutes informations, communiquer avec moi sur [info@degenio.com](mailto:info@degenio.com).

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Changement** |
| 10 juin 2016 | 1.0 | Version initiale |
|  |  |  |
|  |  |  |

**Manipulation 1**: Utilisation d’un algorithme de régression linéaire avec plusieurs prédicteurs

**Objectif**

Prédire le prix de vente d'une maison comme une fonction des attributs de la maison par la mise en place et l’exploitation de l’algorithme de régression linéaire.

**Préliminaire**

* Anaconda ou pycharm est disponible.

**Approche**

1. On utilisera le dataset connu comme le Boston-house-price dataset. Celui-ci contient 506 instances. Les features sont au nombre de 14 dont la valeur ou prix median de la maison qui sera la valeur cible. Les attributs sont des réels.

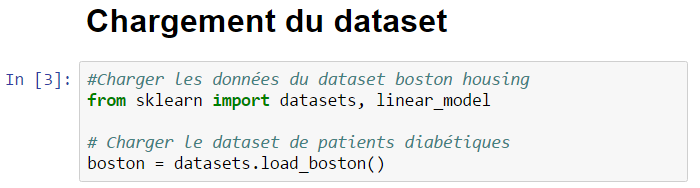
Les étapes que l'on va suivre sont les suivantes

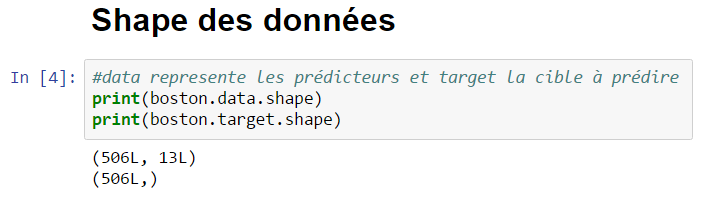
* Créer le modele
  + Étape 1: obtenir le data
  + Étape 2: faire un pré-traitement des données
  + Étape 3: Définir les features
* Faire l'apprentissage du modele
  + Étape 4: choisir un algorithme ML et procéder à l'apprentissage
* Évaluer le Score et tester le modele
  + Étape 5: Passer au test et faire la prédiction

**Démarche**

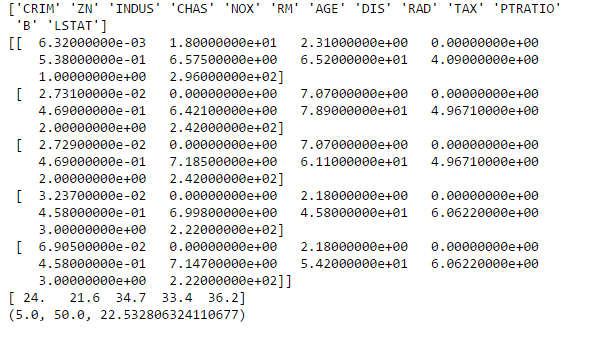
Étape 1: obtenir le data

1. On importe le dataset Boston-house-price.



1. On procède à l'exploration du dataset. En premier, on détermine le shape du dataset que ce soit au niveau des prédicteurs ou du target, soit:  
    
2. On affiche quelques statistiques du dataset ainsi qu'une partie du data, soit:

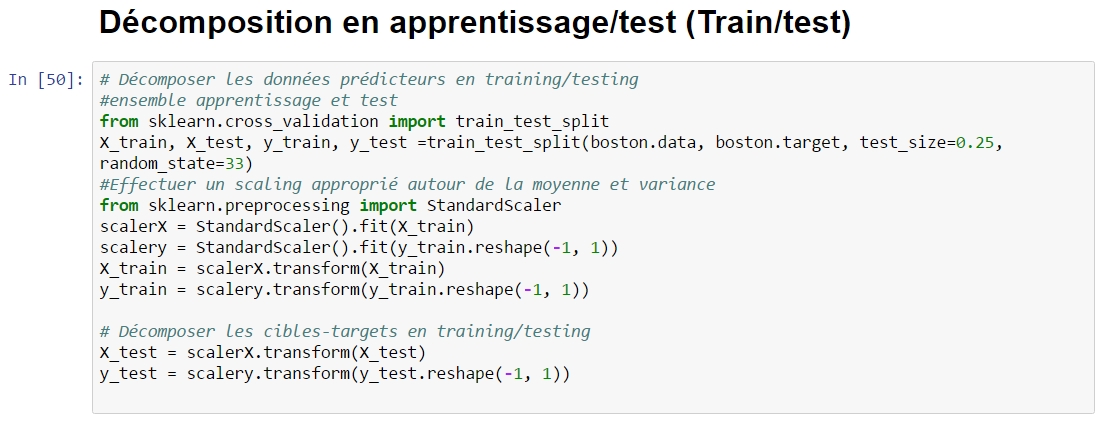




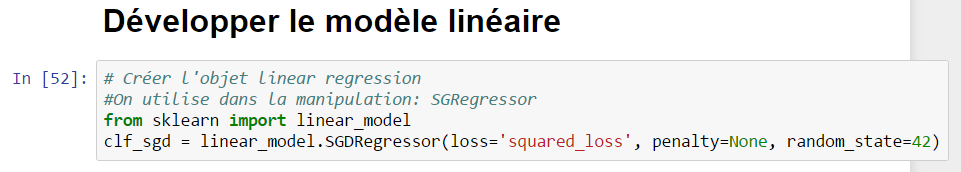
1. Dans le cas présent, tous les prédicteurs seront inclus dans le modèle



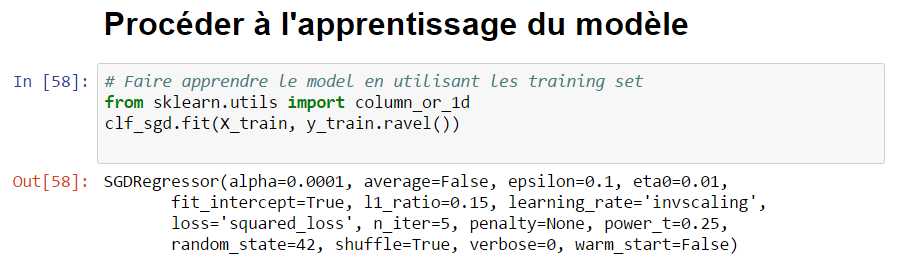
1. On partage les données disponibles entre ensemble d'apprentissage et test. On remarque aussi que l'on a procédé à une mise à l'échelle des ensemble d'apprentissage et de test comme le montre la figure suivante:



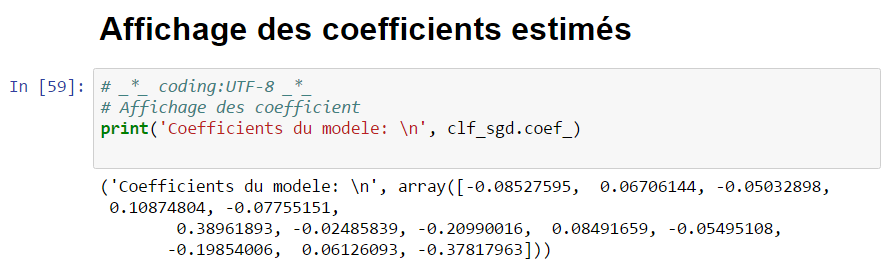
1. On passe ensuite à l'instantiation du modèle de regression linéaire. On utilisera la version SGDRegressor disponible au niveau de scikit-learn.



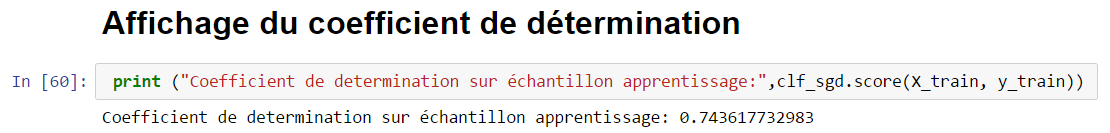
1. On procède ensuite à l'apprentissage du modèle en utilisant l'ensemble d'apprentissage, soit:

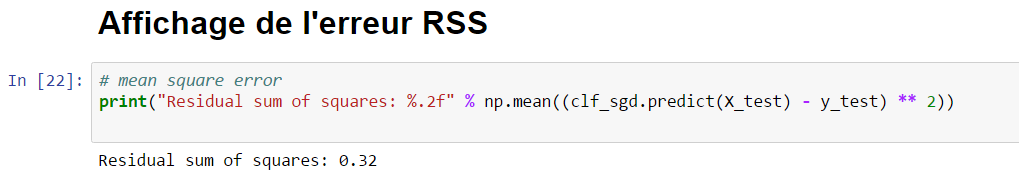


1. Les coefficients estimés du modèle sont ensuite affichés.

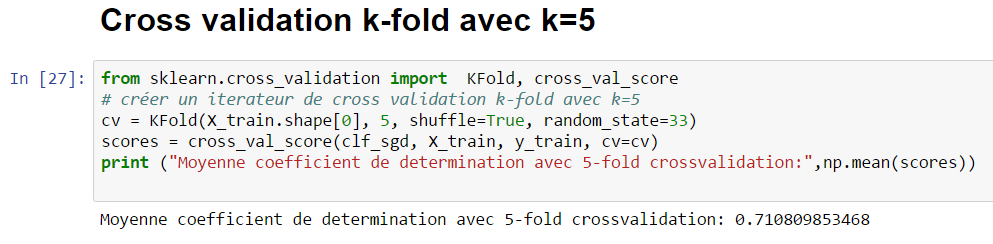


1. Le coefficient de determination ainsi que l'erreur RSS sont affichés. On Remarque que le coefficient de determination est assez faible.

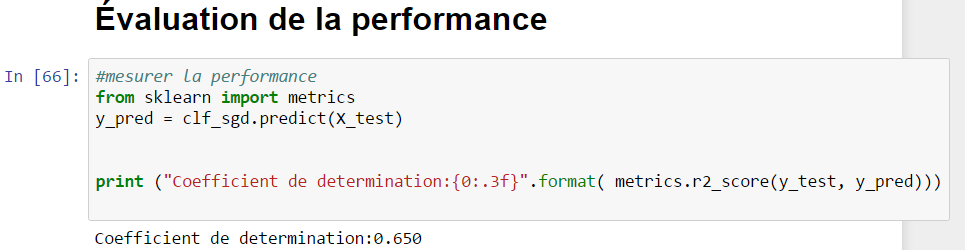




1. Pour l'exemple, on procède à une cross-validation k-fold avec k=5. On commence par le premier LR estimator SGRegressor



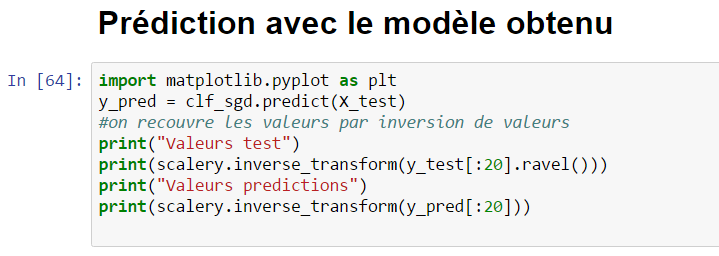
1. On évalue maintenant la performance de l’algorithme utilisé comme suit :



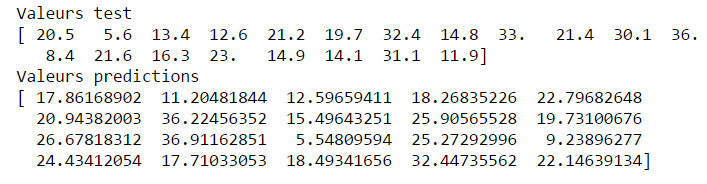
ce qui donne en sortie:

Coefficient de determination:0.650

1. On termine notre exploration du modèle en procédant à une prédiction en utilisant l'échantillon de test.



ce qui donne:



**Annexe**

Prédicteur linéaire SGRegressor : **Stochastic Gradient Descent (SGD)**

A simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) [Support Vector Machines](http://en.wikipedia.org/wiki/Support_vector_machine) and [Logistic Regression](http://en.wikipedia.org/wiki/Logistic_regression). Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.

Référence : http://scikit-learn.org/stable/modules/sgd.html

**Pénalité l2**

SGD supports the following penalties:

* penalty="l2": L2 norm penalty on coef\_.
* penalty="l1": L1 norm penalty on coef\_.
* penalty="elasticnet": Convex combination of L2 and L1; (1 - l1\_ratio) \* L2 + l1\_ratio \* L1.

Référence : http://scikit-learn.org/stable/modules/sgd.html

**Références**

<http://online.cambridgecoding.com/notebooks/cca_admin/getting-started-with-regression-and-decision-trees>

Code

#Charger les données du dataset boston housing

from sklearn import datasets, linear\_model

# Charger le dataset de maison boston

boston = datasets.load\_boston()

#data represente les prédicteurs et target la cible à prédire

print(boston.data.shape)

print(boston.target.shape)

import numpy as np

#afficher les noms des colonnes features

print(boston.feature\_names)

#données sur les 5 premiers enregistrements predicteurs

print(boston.data[:5,:10] )

#données sur les 5 premiers enregistrements cibles

print(boston.target[:5,] )

#max, mean et moyenne

print(min(boston.target ),max(boston.target ), np.mean(boston.target))

# Décomposer les données prédicteurs en training/testing

#ensemble apprentissage et test

from sklearn.cross\_validation import train\_test\_split

X\_train, X\_test, y\_train, y\_test =train\_test\_split(boston.data, boston.target, test\_size=0.25,

random\_state=33)

#Effectuer un scaling approprié autour de la moyenne et variance

from sklearn.preprocessing import StandardScaler

scalerX = StandardScaler().fit(X\_train)

scalery = StandardScaler().fit(y\_train.reshape(-1, 1))

X\_train = scalerX.transform(X\_train)

y\_train = scalery.transform(y\_train.reshape(-1, 1))

# Décomposer les cibles-targets en training/testing

X\_test = scalerX.transform(X\_test)

y\_test = scalery.transform(y\_test.reshape(-1, 1))

# Créer l'objet linear regression

#On utilise dans la manipulation: SGRegressor

from sklearn import linear\_model

clf\_sgd = linear\_model.SGDRegressor(loss='squared\_loss', penalty=None, random\_state=42)

# Faire apprendre le model en utilisant les training set

from sklearn.utils import column\_or\_1d

clf\_sgd.fit(X\_train, y\_train.ravel())

# \_\*\_ coding:UTF-8 \_\*\_

# Affichage des coefficient

print('Coefficients du modele: \n', clf\_sgd.coef\_)

print ("Coefficient de determination sur échantillon apprentissage:",clf\_sgd.score(X\_train, y\_train))

# mean square error

print("Residual sum of squares: %.2f" % np.mean((clf\_sgd.predict(X\_test) - y\_test) \*\* 2))

from sklearn.cross\_validation import KFold, cross\_val\_score

# créer un iterateur de cross validation k-fold avec k=5

cv = KFold(X\_train.shape[0], 5, shuffle=True, random\_state=33)

scores = cross\_val\_score(clf\_sgd, X\_train, y\_train, cv=cv)

print ("Moyenne coefficient de determination avec 5-fold crossvalidation:",np.mean(scores))

#mesurer la performance

from sklearn import metrics

y\_pred = clf\_sgd.predict(X\_test)

print ("Coefficient de determination:{0:.3f}".format( metrics.r2\_score(y\_test, y\_pred)))

import matplotlib.pyplot as plt

y\_pred = clf\_sgd.predict(X\_test)

#on recouvre les valeurs par inversion de valeurs

print("Valeurs test")

print(scalery.inverse\_transform(y\_test[:20].ravel()))

print("Valeurs predictions")

print(scalery.inverse\_transform(y\_pred[:20]))