Régression

Un problème de régression consiste à construire une fonction qui prédit une quantité réelle *Y* en fonction de variables *X*. C'est une façon d'exprimer un lien entre deux quantités en fonction des observations (voir régression). La régression linéaire est le modèle le plus simple et consiste à supposer que la relation est linéaire.

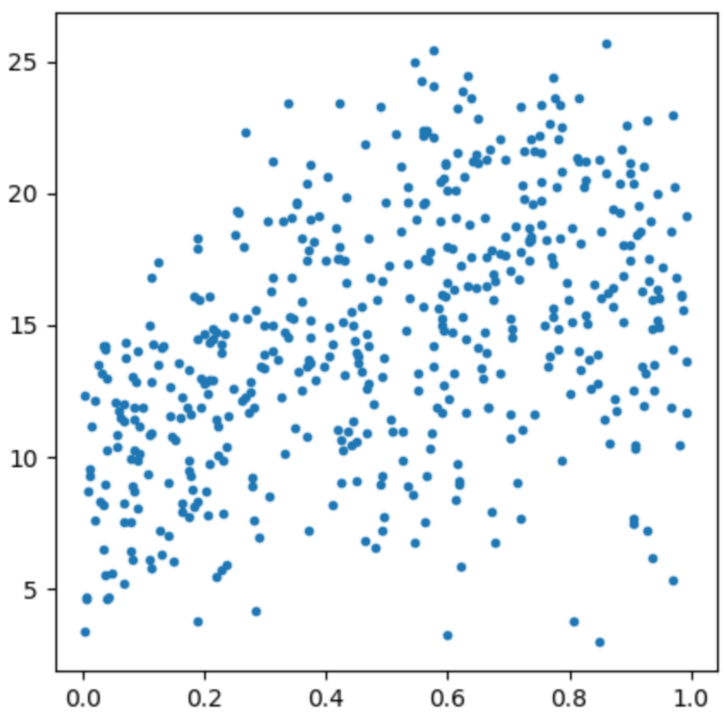
Principe

On commence par générer un jeu de données artificiel.

```
from sklearn.datasets import make_friedman1
X, Y = make_friedman1(n_samples=500, n_features=5)
```

On représente ces données.

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(5, 5))
ax = plt.subplot()
ax.plot(X[:, 0], Y, '.')
```



D'un point de vue géométrique, un problème de régression consiste à trouver la courbe qui s'approche au plus de tous les points. Le plus simple est de supposer que c'est une droite. Dans ce cas, on choisira un modèle de régression linéaire : LinearRegression.

```
from sklearn.linear_model import LinearRegression
reglin = LinearRegression()
reglin.fit(X, Y)
```

L'optimisation du modèle produit une droite dont les coefficients sont :

```
print(reglin.coef_, reglin.intercept_)
```

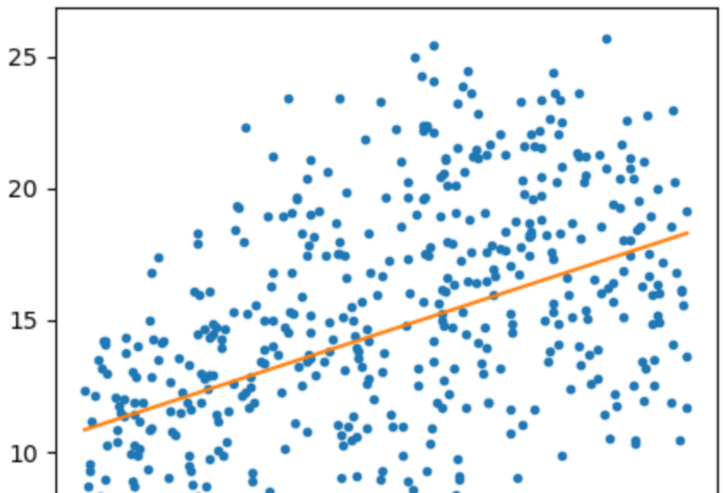
Out:

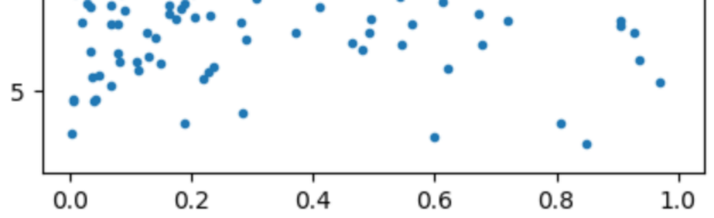
```
[ 7.57903679  7.22417025 -0.23332739  9.80504941  4.93688159] -0.2911721911937004
```

On reprend le premier graphe est on y ajoute la droite qui correspond à la régression linéaire uniquement sur la première dimension.

```
reglin = LinearRegression()
reglin.fit(X[:, :1], Y)

import matplotlib.pyplot as plt
fig = plt.figure(figsize=(5, 5))
ax = plt.subplot()
x = list(sorted(X[:, :1]))
y = reglin.predict(x)
ax.plot(X[:, 0], Y, '.')
ax.plot(x, y)
```





Evaluation

Le critère d'erreur le plus utilisé est l'erreur quadratique. Si y_i est la valeur à prédire, et y_i^* la valeur prédite, l'erreur est :

$$err = \sum_i (y_i - y_i^*)^2$$

La plupart des problèmes sont multidimensionnelles et s'appuie sur de nombreuses variables mais bien souvent la valeur à prédire est réelle. On représente donc le graphique XY où l'axe des abscisses représente la valeur à prédire et l'axe des ordonnées la valeur prédite. On découpe d'abord en train/test.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y)

#################
# On apprend un modèle.

from sklearn.linear_model import LinearRegression
reglin = LinearRegression()
reglin.fit(X_train, y_train)
```

On prédit.

```
pred = reglin.predict(X_test)
```

On calcule l'erreur.

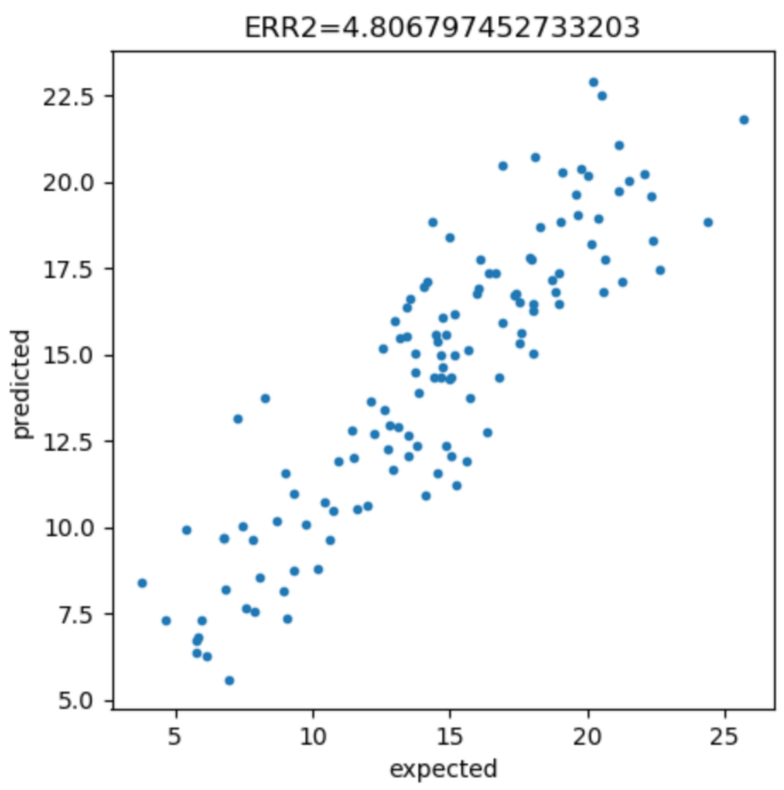
```
from sklearn.metrics import mean_squared_error
err = mean_squared_error(y_test, pred)
print(err)
```

Out:

```
4.806797452733203
```

On dessine.

```
fig = plt.figure(figsize=(5, 5))
ax = plt.subplot()
ax.plot(y_test, pred, '.')
ax.set_xlabel("expected")
ax.set_ylabel("predicted")
ax.set_title("ERR2={0}".format(err))
```



Comparer deux modèles

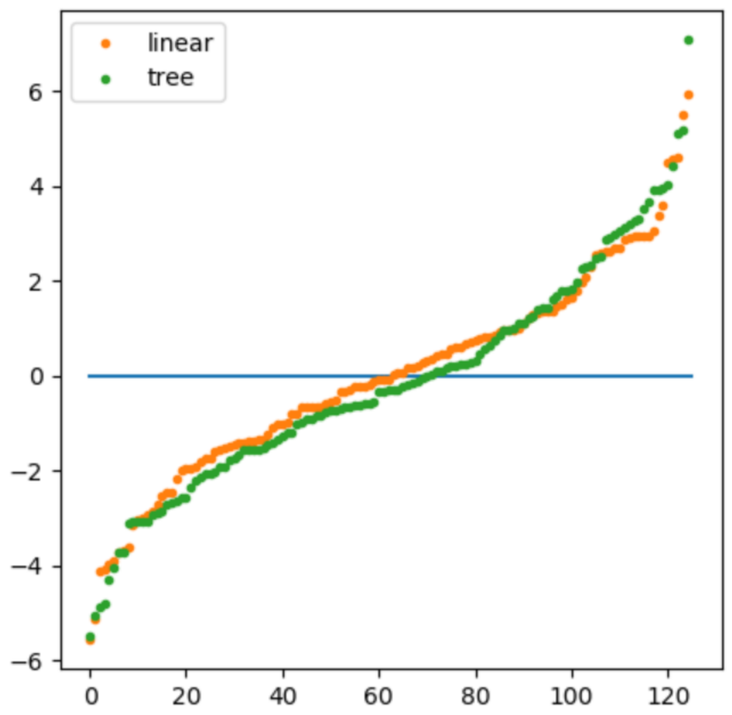
On peut bien évidemment les comparer numériquement. Graphiquement, les nuages de points s'entremêlent et deviennent peu visible. Pour y remédier, on trie les erreurs par ordre croissant.

```
from sklearn.tree import DecisionTreeRegressor
regtree = DecisionTreeRegressor()
regtree.fit(X_train, y_train)
pred_tree = regtree.predict(X_test)
```

On dessine.

```
y1 = list(sorted(pred - y_test))
y2 = list(sorted(pred_tree - y_test))
fig = plt.figure(figsize=(5, 5))
ax = plt.subplot()
ax.plot([0, len(y1)], [0, 0], '-')
ax.plot(y1, '.', label="linear")
ax.plot(y2, '.', label="tree")
ax.legend()


plt.show()
```



Total running time of the script: (0 minutes 0.328 seconds)

Download Python source code: plot_regression.py

Download Jupyter notebook: plot_regression.ipynb

Mis à jour le 2018-03-07.