Régression

Antoine Nongaillard

But Info – IUT – Université de Lille

R5.AB.12 – Séance 05

Plan

Le principe de régression

D'où vient-elle ?

Elle fut crée par *Sir Francis Galton* en 1885.

Il remarque que les pères dont la taille est supérieure à la moyenne ont des fils plus petits qu'eux. À l'inverse, les pères dont la taille est inférieure à la moyenne ont des fils plus grand qu'eux.

Il écrit un modèle de prédiction de la taille d'un enfant selon la taille de son père via une régression vers la moyenne :

- en entrée : n observations $(x_i, y_i) \in \mathbb{R}^2$ où y_i est le résultat associé à x_i
- on cherche une fonction linéaire $F(x_i)$ approchant ces résultats telle que :

$$F(x_i) = ax_i + a_0 \quad a, a_0 \in \mathbb{R}$$

Le but est de trouver **les bons estimateurs pour les paramètres** a et a_0.

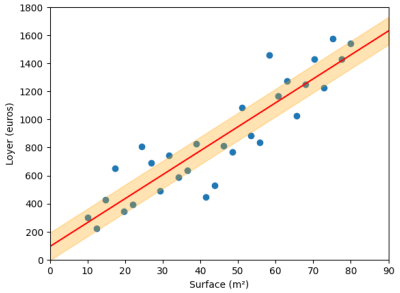
Le principe de régression

Un exemple

Payez-vous trop cher votre loyer ?

Vous avez récupéré sur un site de location une trentaine de prix des locations disponibles, ainsi que la surface associée. On utilise une régression linéaire et un intervalle de confiance.

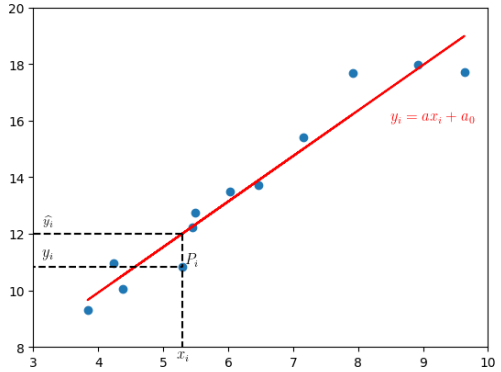
Loyer mensuel (€)	Surface (m^2)
1500	32
2120	65
2500	60
...	...



Calcul d'une régression linéaire

D'un point de vue géométrique

On cherche la droite de régression la plus proche des observations minimisant les erreurs d'estimation : $\epsilon = y_i - \widehat{y}_i$.



Calcul d'une régression linéaire

Méthode des moindres carrés

Pour cela, on utilise souvent la méthode des **moindres carrés** :

$$E(a, a_0) = \sum_{i=1}^{n} [y_i - F(x_i)]^2 = \sum_{i=1}^{n} [y_i - (ax_i + a_0)]^2$$

Après diverses manipulations algébriques impliquant des dérivées partielles :

$$\overline{y} = a\overline{x} + a_0 \qquad a = \frac{S_{x,y}}{S_x}$$

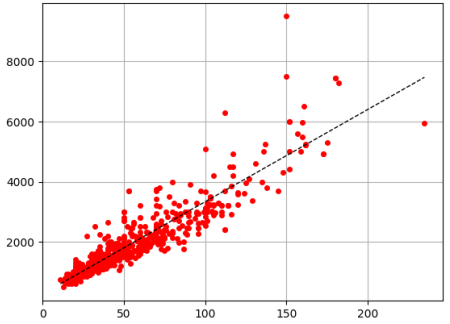
Il existe des conditions nécessaires pour appliquer un tel modèle :

- une relation de corrélation est nécessaire entre les variables explicatives, et les variables à expliquer.
- La variance des distributions associées à différents échantillons de données doivent être les mêmes.
- Les valeurs prises par la variables à expliquer ne sont pas dépendantes les unes des autres.
- La variable à expliquer est distribuée suivant une loi normale.

Calcul d'une régression linéaire

Voici un exemple basique mais complet :

```
my_data = house_data[house_data['loyer']<10000]
my_data_x = my_data['surface'].values
my_data_y = my_data['loyer'].values
# mise en forme des données (après filtrage)
x = np.array(my_data_x).reshape(-1,1)
y = np.array(my_data_y)
# création du modèle
from sklearn.linear_model import LinearRegression
LM = LinearRegression()
LM.fit(x, y)
# tracé des points et de la droite de régression
X = np.linspace(x.min(), x.max(), 3)
Y = LM.predict(X.reshape(-1,1))
plt.plot(my_data_x, my_data_y, 'ro')
plt.plot(X,Y, "black", linestyle="--")
```



Régression linéaire multiple

Méthode des moindres carrés

La contrainte linéaire s'exprime de manière vectorielle par : $\widehat{y} = x^T\theta$ avec $x^T = (x_1, \ldots, x_N)$ et $\widehat{y}^T = (\widehat{y_1}, \ldots, \widehat{y_N})$ et N le nombre d'observations.

La **régression linéaire multiple** est une généralisation immédiate : la fonction F que l'on souhaite estimer dépend de plusieurs variables.

Les observations peuvent s'écrire : $(X_i = (x_{i1}, x_{i2}, \ldots, x_{im}) \in \mathbb{R}^m, y_i \in \mathbb{R})$.
La fonction à estimer sera de la forme : $F(x_{i1}, x_{i2}, \ldots, x_{im}) = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \cdots + \theta_m x_{im}$

L'objectif est d'estimer $\theta = (\theta_0, \theta_1, \theta_2, \ldots, \theta_m)$ afin de minimiser la distance entre l'observation y_i et le modèle $\widehat{y}_i$.

- distance euclidienne : $\| \widehat{y}_i - y_i \|_2 = \| x_i^T\theta - y_i \|_2$
- calcul de l'erreur empirique : $E = \sum_{i=1}^{N} \| x_i^T\theta - y_i \|_2$
- La solution est exacte [1] : $\widehat{\theta} = (X^TX)^{-1}X^Ty$

1. https://eli.thegreenplace.net/2014/derivation-of-the-normal-equation-for-linear-regression

Calcul d'une régression linéaire

Les distances les plus courantes

Un grand nombre de variation à cette méthode existent, basées sur des notions de distance différentes :

- Euclidian : $\sqrt{\sum(x-y)^2}$
- Manhattan : $\sum(|x-y|)$
- Chebyshev : $\max(|x-y|)$
- Minkowski p : $\sum(|x-y|^p)^{\frac{1}{p}}$
- ...

La méthode des moindres carrés présente néanmoins des inconvénients :

- la matrice X^TX doit être réversible
- très chronophage dès que le nombre d'observations N et de variables m sont élevés

Calcul d'une régression linéaire

Méthode de la descente du gradient

Il existe des algorithmes pour approximer la solution : l'un des plus connus est la *descente du gradient*. L'algorithme suit les étapes suivantes :

1. `initialisation aléatoire de` $\theta = (\theta_0, \theta_1, \ldots, \theta_m)$
2. `pour chaque` θ_i `faire:` $\theta_i = \theta_i + \alpha\frac{\partial E}{\partial \theta_i}$
3. `si pas de convergence, reprendre à l'étape 2, sinon Fin.`

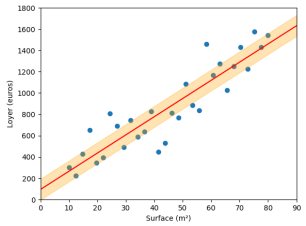
À chaque itération, les valeurs des θ_i sont recalculés en fonction de l'erreur actuelle E et du facteur α, jusqu'à convergence.

En python :

```
X = np.matrix([np.ones(house_data.shape[0]), house_data['surface'].values]).T
y = np.matrix(house_data['loyer']).T
# calcul exact du paramètre theta
theta = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
```


En pratique

On obtient deux valeurs paramétrant notre modèle : $loyer = 25.2\ surface + 144.42$

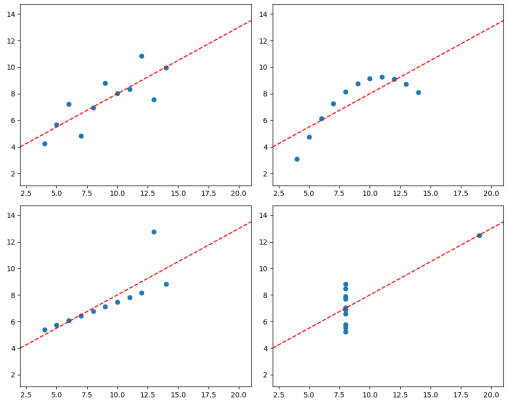


À partir de là, on entre dans la phase d'exploitation du modèle.

- Maintenant qu'on a notre paramètre θ, on a trouvé la droite qui colle le mieux à nos données.
- On peut effectuer des prédictions sur de nouvelles données : c'est-à-dire prédire le loyer en fonction de la surface qu'on nous donne en entrée, en appliquant directement la formule du modèle ci-dessus.

Le quartet d'Anscombe

Il permet de montrer visuellement que pour 4 jeux de données très différents, on obtient la même droite de régression !



Si l'on n'examine pas assez les données, si on ne mesure pas de la bonne manière l'erreur de son modèle, on peut facilement arriver à des aberrations de modélisation.

Mais ensuite...

Que se passe-t-il si :

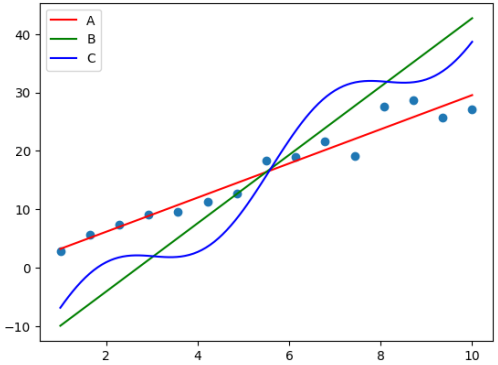
- on change l'hypothèse de linéarité (une droite) et qu'on en prend une autre (un polynôme du second degré par exemple) ?
- on teste le modèle avec d'autres types d'erreurs que la distance euclidienne ?
- on ajoute des caractéristiques supplémentaires en entrée (dimensions) ?
- au fur et à mesure que la surface augmente, les données ont l'air d'être de plus en plus "éparses" ? Comment intégrer ce comportement dans la modélisation ?

Trouver la configuration la plus pertinente revient, du coup, à tester les différents modèles qui en découlent et trouver celui qui convient le mieux en termes de performance.

Plan

Mesurer l'erreur d'un modèle

Voici 3 modèles A,B et C entraînés différemment dont la somme des erreurs est nulle, alors que les 3 modèles sont des performances très différentes.



De nombreux indicateurs ont été crées afin d'évaluer les erreurs commises et comparer les modèles, même si la somme des erreurs n'est pas utilisable.

Mesurer l'erreur d'un modèle

Il existe différentes mesures :

- l'**erreur moyenne absolue** ou *Mean Absolute Error (MAE)* :

$$MAE = \frac{1}{N}\sum |y - \hat{y}|$$

- l'**erreur quadratique moyenne** ou *Mean Square Error (MSE)* renforce l'importance des erreurs fortes. Elle peut être comprise comme la variance des erreurs.

$$MSE = \frac{1}{N}\sum (y - \hat{y})^2$$

- la **racine de l'erreur quadratique moyenne** ou *Root Mean Square Error (RMSE)*. Elle peut être comprise comme l'écart-type des erreurs.

$$RMSE = \sqrt{\frac{1}{N}\sum (y - \hat{y})^2}$$

Mesurer l'erreur d'un modèle

Deux indicateurs permettent de calculer la partie de la variance due au modèle crée :

- le **coefficient de détermination**, noté R^2 :

$$R^2 = 1 - \frac{\sum(y - \widehat{y})^2}{\sum(y - \overline{y})^2}$$

- la **variance expliquée** ou *Explained Variance Score (EVS)* :

$$EVS = 1 - \frac{S(y - \widehat{y})}{S(y)}$$

D'autres indicateurs existent comme :

- l'**erreur maximale** est surtout utilisée pour garantir que tous les résultats sont acceptables
- le **taux d'erreur moyen** ou *Mean Average Percentage Error (MAPE)* est utile si la variable cible comprend à la fois des valeurs faibles et fortes : c'est indépendant de l'amplitude
- l'**erreur quadratique logarithmique moyenne** ou *Mean Squared Logarithmic Error (MSLE)* est insensible à l'échelle de la variable cible : pratique si la variable évolue de manière exponentielle

En pratique

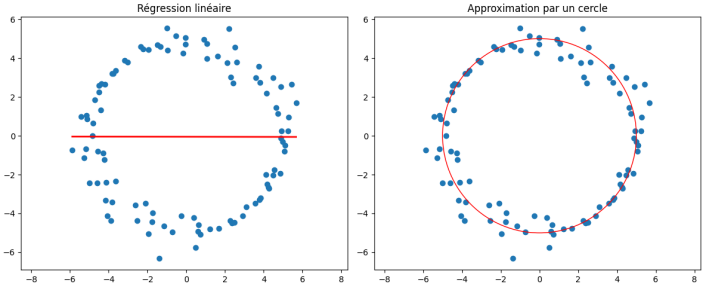
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# data
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([15, 11, 2, 8, 25, 32])
# model
LM = LinearRegression()
LM.fit(x, y)
# affichage des coefficients calculés
print("intercept: " + LM.intercept_ + " slope: " + LM.coef_)
# une première prédiction
x_test=np.array([20,40]).reshape(-1,1)
y_pred = LM.predict(x_test)
# calcul des scores
r2 = LM.score(x, y)
mse = mean_squared_error(y, LM.predict(x), squared=True)
rmse = mean_squared_error(y, LM.predict(x), squared=False)
mae = mean_absolute_error(y, LM.predict(x))
```


Plan

Contraintes et conséquences du choix

L'objectif du *Machine Learning* (ML) est d'identifier un modèle, approximation de la réalité pour effectuer des prédictions. Qui dit approximation, dit perte d'information et donc mesure d'erreurs.

Le modèle statistique sous-jacent subit une contrainte de forme, indépendante des données : si on fait une régression linéaire, on aura forcément un modèle sous forme d'une droite, ce qui contraint sa forme.



Les différents types de régression

Il existe différents types de régression :

- la régression linéaire simple
- la régression linéaire multiple (plusieurs dimensions)
- la régression Ridge (régularisation, contraintes sur les coefficients)
- la régression Lasso (régularisation, contraintes sur les domaines de valeurs)
- la régression polynomiale
- ...

Il existe encore d'autres type de régression mais qui servent à faire de la classification :

- la régression logistique binaire
- la régression logistique polytomique

Mais surtout, il existe encore bien d'autres approches pour faire des modèles prédictifs !

La régression polynomiale

C'est une généralisation de la régression linéaire multiple.

Si on considère deux variables x_1 et x_2, l'équation de degré 2 cherchée est la suivante :

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_1^2 + a_4 x_1 x_2 + a_5 x_2^2$$

Si chaque variable ou produit de variables est remplacé par une nouvelle variable z, on obtient :

$$y = a_0 + a_1 z_1 + a_2 z_2 + a_3 z_3 + a_4 z_4 + a_5 z_5$$

La régression polynomiale est donc une régression linéaire dans ce nouvel espace. Les algorithmes vus précédemment fonctionnent et sont tout indiqués.

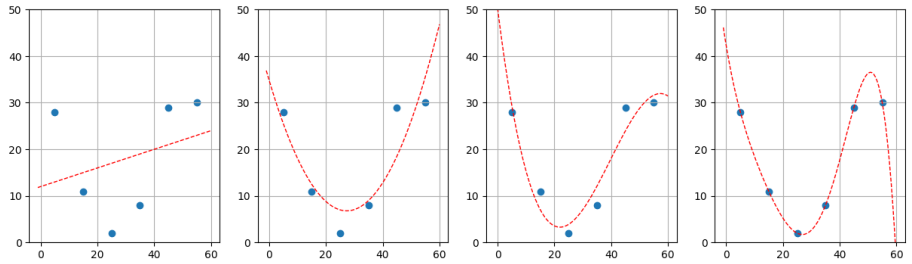
En pratique

```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
# data
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([15, 11, 2, 8, 25, 32])
pf = PolynomialFeatures(degree=2)
x_ = pf.fit_transform(x)
# model
PM = LinearRegression().fit(x_, y)
# prédiction
x_test = np.array([20,40]).reshape(-1,1)
x_test_ = pf.fit_transform(x_test)
y_pred = PM.predict(x_test_)
# résultat et scores
print("intercept: " + PM.intercept_ + " coefficients: " + PM.coef_)
R2 = PM.score(x_, y)
MSE = mean_squared_error(y, LM.predict(x), squared=True)
RMSE = mean_squared_error(y, LM.predict(x), squared=False)
MAE = mean_absolute_error(y, LM.predict(x))
```


Overfitting et *Underfitting*

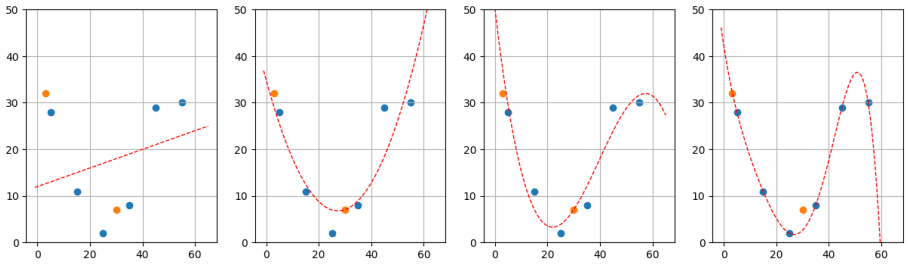
Le choix du degré du polynôme n'est pas chose facile. Trop faible, et on tombe dans l'**underfitting**, trop élevé et on tombe dans l'**overfitting**.

On rappelle qu'**il existe toujours un polynôme de degré n passant par n points**.



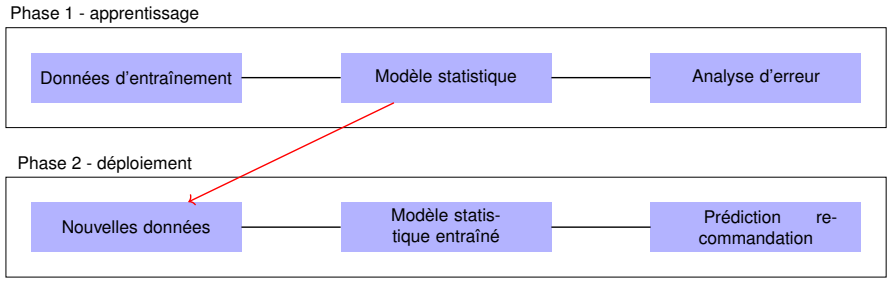
Overfitting et *Underfitting*

La technique classique pour décider du meilleur modèle, consiste à ne pas utiliser toutes les observations pour faire l'apprentissage. C'est en voyant la performance de différents modèles sur ces données de test que l'on décidera du meilleur modèle.



Les bases de l'apprentissage automatique

Le machine learning n'est pas une baguette magique !
Un *data scientist* passe la majorité de son temps à la sélection, au nettoyage des données plutôt qu'à la modélisation. L'apprentissage se base sur un ensemble de données d'entraînement (*training set*)



Gestion des données

Évaluer un modèle revient à mesurer l'écart entre la prédiction et la réalité.

Si on évalue un modèle à partir des mêmes données qui ont servi à l'entraînement, l'erreur sera minimum sur ces données, mais sera aussi toujours plus élevée sur des données que le modèle ne connaît pas.

Il faut donc répartir les données disponibles pour l'entraînement et l'évaluation :

- training set (80%)
- testing set (20%)

Des fonctions prédéfinies existent pour cela :

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
```

Si le jeu de données est volumineux, utiliser l'ensemble des données pour l'entraînement de votre modèle pour se révéler chronophage. Le *sampling* est l'échantillonnage des données : il doit être représentatif de toutes les données, sinon on introduit un biais. Une récupération aléatoire suffit à éviter ce biais !

```
sample = np.random.randint(data_size, size=int(data_size*0.1) )
sampled_data = data.iloc[sample]
```