Monitoring Regression

October 3, 2023

/BF/

1. Linear regression

1.1. Definition

La régression linéaire est un des cas d'école basique d'analyse statistique, cherchant à modéliser une réponse/un résultat en fonction d'une ou plusieurs variables.

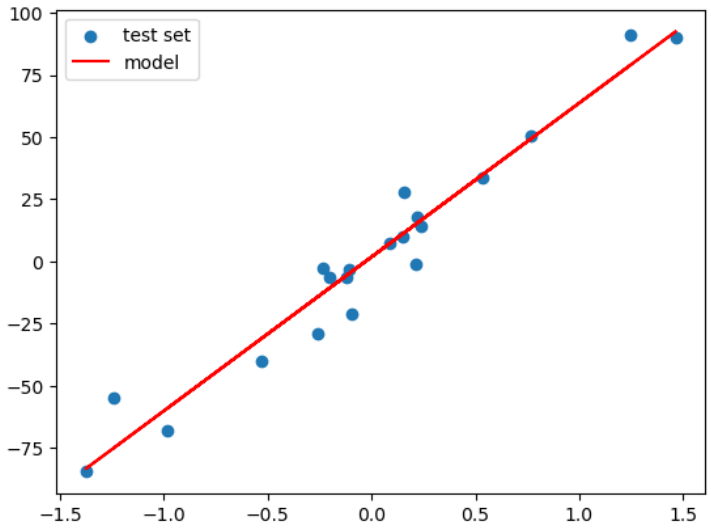

Figure 1: Illustration d'une régression linéaire.

Dans cet exemple simple, on cherche à obtenir une fonction linéaire de type $y = a \times x + b$ permettant d'estimer, pour chaque point i la réponse y_i en fonction de x_i.

Un modèle n'étant jamais parfait, il existe un écart entre la réponse calculée $\hat{y}_i$ et la réponse mesurée y_i , écart aussi appelé résidu ε_i, défini alors comme:

$$\varepsilon_i = y_i - \hat{y}_i$$

Ce résidu provient de deux phénomènes : l'erreur expérimentale pure de la mesure de la réponse, et l'écart entre le modèle et le comportement réel du système, appelé le manque d'ajustement (Lack of fit). Ce manque d'ajustement est lui-même issu de deux composantes, dont nous reparlerons plus loin :

- Le biais : l'écart entre la prédiction et la réponse (notion métrologique de justesse).
- La variance : la variabilité des résultats du modèle pour un point donné (notion métrologique de précision).

L'application la plus courante de la régression linéaire utilise la méthode des moindres carrés, cherchant les paramètres pour lesquels l'erreur (l'écart) est la plus petite possible, ce qui peut se résumer par la minimisation de la fonction de coût f :

$$\text{Find} \min_{a,b} f(a,b), \text{for } f(a,b) = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

pour n points de données.

La qualité du modèle obtenu peut être appréciée à l'aide du coefficient de détermination R^2, définit comme:

$$R^2 = \frac{\sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

avec $\bar{y}$: valeur moyenne des réponses.

Une autre métrique fréquemment utilisée est le RMSE, pour *Root Mean Square Error*, définit comme :

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2}$$

1.2. `from sklearn.linear_model import LinearRegression`

La librairie scikit-learn contient un module `linear_model` proposant plusieurs modèles, dont la méthode des moindres carrés (*Ordinary Least Square -OLS-*), décrite en 1.1.

Il est ainsi possible de créer simplement une instance d'un modèle, possédant plusieurs attributs et méthodes, dont les plus importantes sont :

- `model.fit(X, y)` : Ajuste le modèle en fonction du jeu de données entré en argument.
- `model.predict(X)` : Prédire l'image de X avec le modèle ajusté.
- `model.score(X,y)` : Retourne le R^2 du modèle sur les données en argument.

Un exemple valant plus ou moins $6,022.10^{23}$ définitions :

```
# model
model = LinearRegression()
model.fit(x_train, y_train)
print('score', model.score(x_train, y_train))
plt.scatter(x_train, y_train)
y2 = model.predict(x_train)
plt.plot (x_train, y2, 'r')
```

Output :

```
score 0.9732695284776092
```

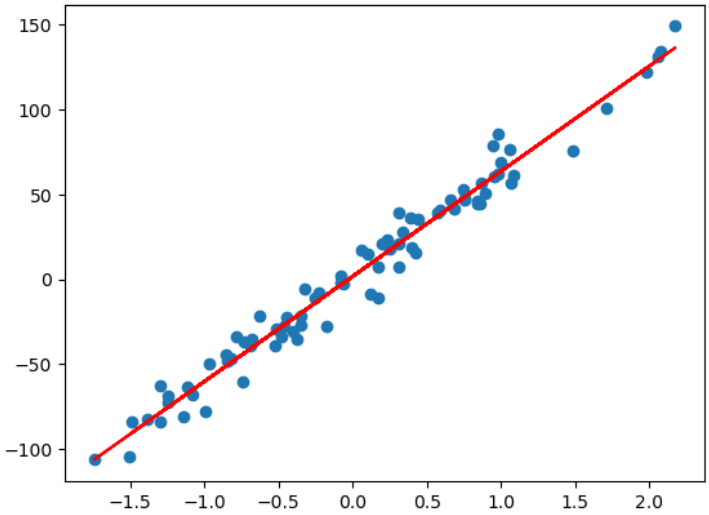

Figure 2: Tracé résultant du code précédent.

Afin d'entraîner et de valider un modèle avec une méthodologie correcte, scikit-learn permet de séparer nos données en un jeu de données d'entraînement (*train set*) et un jeu de test (*test set*).

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

1.3. Polynomial features

Si l'on cherche à modéliser un phénomène plus complexe, il est possible d'obtenir un meilleur résultat en utilisant des features d'ordre supérieur.

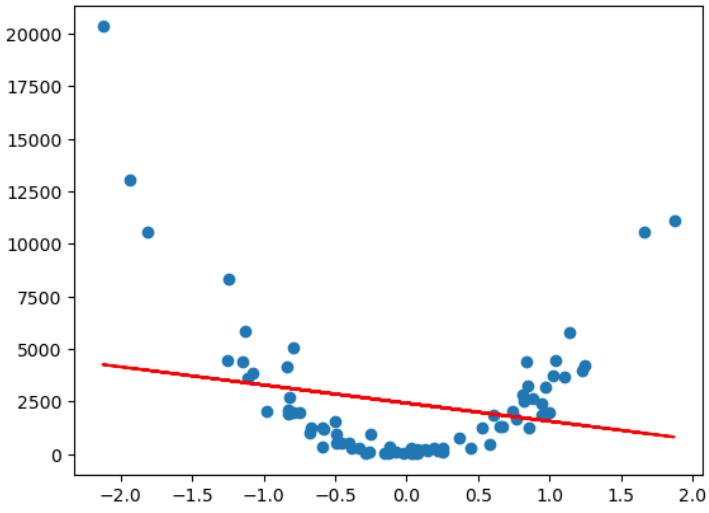

Figure 3: Un phénomène difficile à modéliser avec une fonction affine.

La classe `Polynomial Features` contenue dans le module `preprocessing` permet de rajouter des caractéristiques polynomiales et d'interaction aux données d'entrées. Attention toutefois,

avec un trop grand nombre de caractéristiques polynomiales arrive le risque d'*overfitting* (voir Chapitre 2).

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2)
x_train = poly.fit_transform(xtrain)
x_test = poly.fit_transform(xtest)
model.fit(x_train, y_train)
```

Résultat :

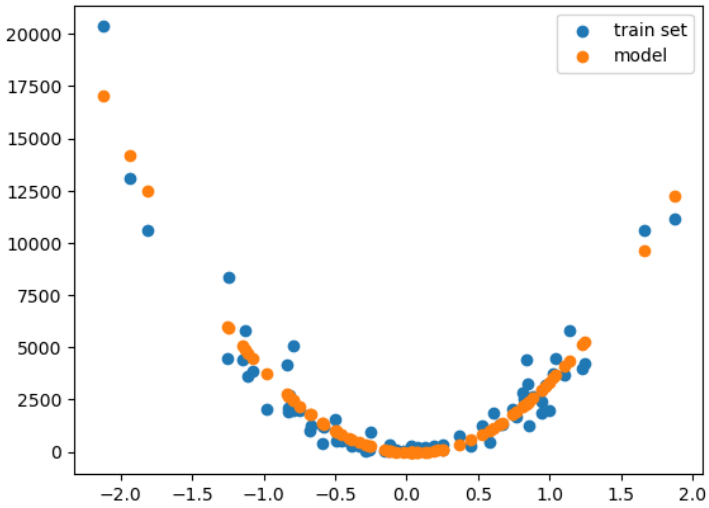

Figure 4: Un meilleur modèle obtenu à l'aide de Polynomial Features.

Un moyen plus fluide d'utiliser Polynomial Features est de l'intégrer dans un pipeline. Un exemple :

```
from sklearn.pipeline import make_pipeline
model3= make_pipeline(PolynomialFeatures(2),LinearRegression())
model3.fit(x2_train, y2_train)
```

2. Dilemme biais/variance

Comme évoqué précédemment, un modèle comporte deux types d'erreurs : le biais et la variance (la justesse et la précision pour nos amis métrologue).

Par exemple, le modèle illustré en Figure 3 présente un grand biais. On peut parler d'*underfit*, le modèle n'est *pas assez ajusté*. Dans le Chapitre 1.3, nous avons vu qu'introduire des caractéristiques polynomiales permet de réduire ce biais. Pour ce faire, il a fallu augmenter la complexité du modèle, et donc la variance. Augmenter la variance du modèle est un bon moyen de réduire le biais, mais peut mener à l'excès inverse : *l'overfitting*, le modèle devient *trop* spécialisé, comme illustré sur la Figure 5.

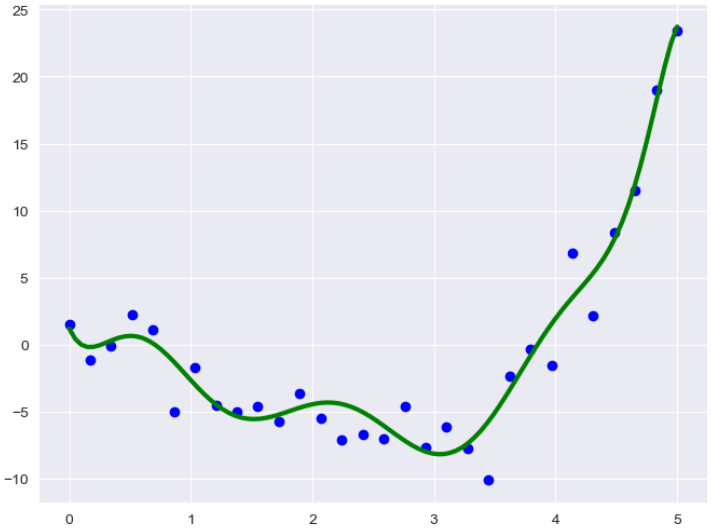

Figure 5: Un exemple d'*overfitting* : un modèle avec un faible biais mais une forte variance.

En augmentant la complexité du modèle (et donc la variance), nous avons réduit le biais, mais pas forcément l'erreur totale du modèle. Il y a donc un optimum à estimer entre biais et variance, selon la complexité du phénomène, comme illustré en Figure 6.

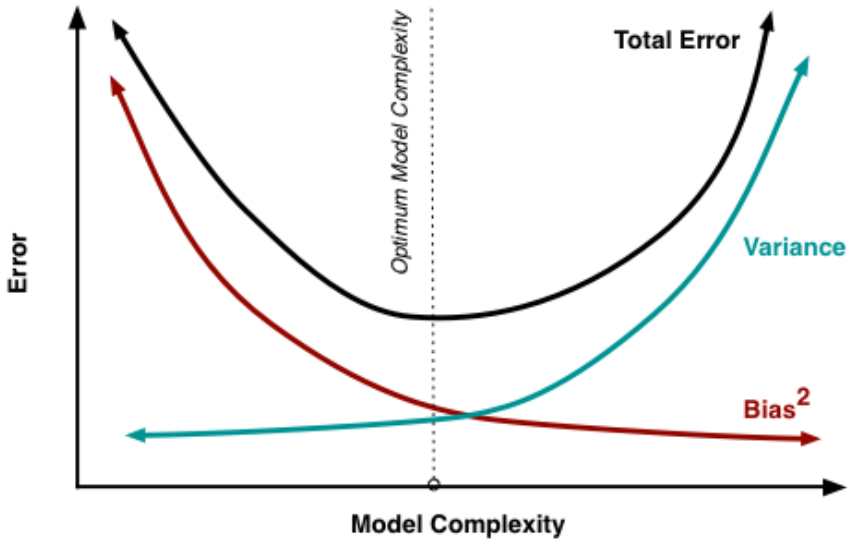

Figure 6: Avec une complexité trop grande, le modèle *overfit*, avec peu de complexité, le modèle *underfit*. Honteusement volé du superbe travail de Scott Fortmann-Roe[1] .

[1]Understanding the Bias-Variance Tradeoff, Scott Fortmann-Roe, 2012