ML : Linear and polynomial regression

TP1 Apprentissage automatique

Modèle de régression : linéaire et polynomiale

Jérôme Buisine
jerome.buisine@univ-littoral.fr

22 septembre 2022

Durée : 3h

L'objectif de ce TP est l'utilisation de deux modèles d'apprentissage supervisé sur deux base de données différentes. Nous allons apprendre à mettre en place un modèle, l'évaluer avec une ou plusieurs mesures de performances. Nous allons pour cela, utiliser la librairie `scikit-learn`.

1 Ressources

Voici un ensemble de ressources qui peuvent vous être utiles pour ce TP :

— 1. Documentation officielle de l'outil `Git` ;
— 2. Gitlab : interface web pour la gestion de projets versionnés sous `Git` ;
— 3. pyenv : permet la gestion d'environnement Python ;
— 4. matplotlib : librairie Python qui permet un affichage rapide de données ;
— 5. pandas : librairie Python qui permet de lire des données et les traiter ;
— 6. jupyter : un outil de travail permettant une interaction rapide et visuelle avec une console Python ;
— 7. scikit-learn : librairie proposant des outils pour l'apprentissage automatique ;

2 Configuration de l'environnement

Créez un dossier à la racine de votre projet nommé `tp1-linear_and_polynomial_regression`. Dans ce dossier et depuis votre terminal, lancez les commandes suivantes :

```
# spécifie l'environnement virtuel Python à Jupyter
ipython kernel install --user --name=ml-venv
# lance l'application Jupyter
jupyter-lab
```

Remarque : pour chaque section suivante, il vous sera demandé de créer un notebook depuis Jupyter. Faites bien attention à la sélection de l'environnement Python lors de sa création. De plus, il vous sera demandé de bien documenter votre code. Vous pouvez également utiliser des cellules de type `Markdown` plutôt que de type `code`.

3 Régression linéaire

Créez un nouveau notebook nommé « linear_regression.ipynb ». Dans ce notebook nous allons aborder l'utilisation d'un premier modèle d'apprentissage supervisé : la régression linéaire.

3.1 Mise en place de la base de données

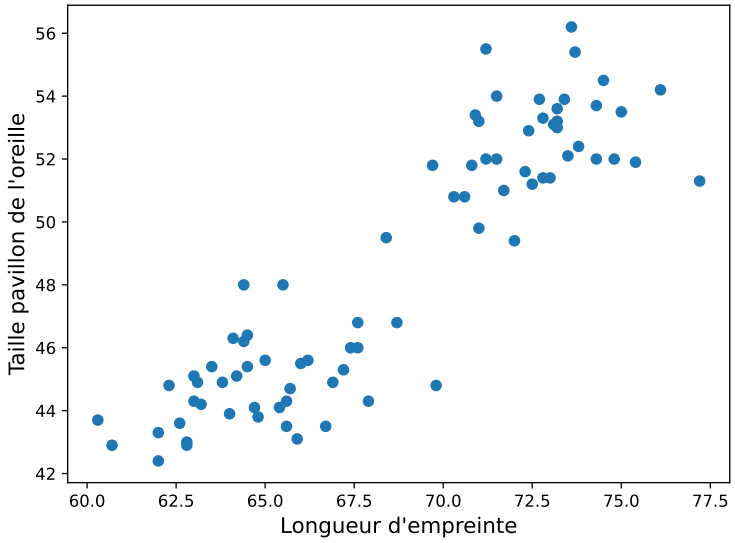
Des chercheurs souhaiteraient analyser un type particulier d'opposum. Ils ont remarqué que ceux qu'ils cherchent à identifier ont pour trait un pavillon d'oreille assez grand. Malheureusement, l'opposum est très discret et ne se montre que très peu.

De ce fait, il souhaiteraient avoir un modèle qui permettrait de donner une estimation du pavillon de l'oreille de l'oppossum à partir d'une empreinte relevée. Ainsi, ils pourront déterminer s'il est utile de continuer à pister ces traces pour trouver l'individu.

Vous allez à votre niveau intervenir pour concevoir un modèle qui puisse répondre à cette demande. Heureusement, une base de données est disponible pour proposer un tel modèle. Pour cela, nous allons utiliser le descripteur `footlgth`, la taille d'empreinte de l'opposum. Il vous faudra prédire le pavillon d'oreille soit le descripteur `earconch`.

Tâche 1 : Exploitez donc la même base de données du TP précédent (base opposum). Préparez cette base de données de manière à ce qu'elle soit correctement nettoyée.

Puis affichez les données des descripteurs ciblés :



3.2 Utilisation d'une régression linéaire

Un modèle de régression linéaire permet d'apprendre un modèle de la forme :

$$y = \alpha x + \beta$$

3.3 La régression linéaire comme un problème d'optimisation

Pour bien comprendre le fonctionnement du modèle, nous allons nous-même identifier le meilleur modèle.

Obtenir le meilleur modèle revient à trouver les meilleurs paramètres α et β pour résoudre l'équation

$y = \alpha x + \beta$. On cherche au final à minimiser la fonction de coût. Ce qui revient à résoudre le problème de minimisation suivant :

$$\underset{\alpha,\beta \in \mathbb{R}^2}{\arg\min} \frac{1}{N} \sum_{i=1}^{N} (\alpha x_i + \beta - y_i)^2$$

Pour rappel, $\alpha x_i + \beta = f(x_i)$, ce qui revient à minimiser notre *MSE*.

Tâche 2 : Créez la fonction `predict_model` qui prend en paramètres deux réels α, β et des données à prédire. Cette fonction retournera les données prédites relativement aux paramètres de régression d'entrée.

Nous allons calculer la performance de ce genre de modèle à partir d'une métrique de performance d'erreur. Il s'agit généralement d'une fonction de perte (également appelée fonction de coût).

Nous déterminons l'erreur quadratique moyenne (*Mean Squared Error*) comme fonction de coût pour évaluer notre modèle sur les données :

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - h(x_i))^2$$

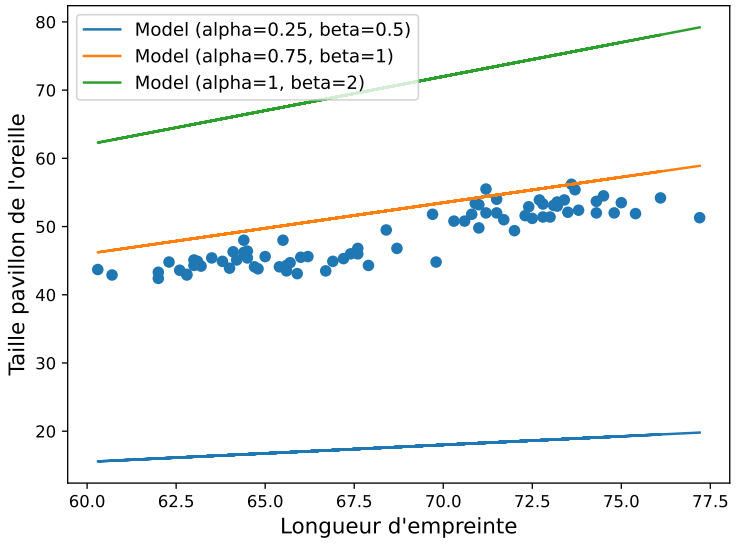
où y_i est la valeur observée et $h(x_i)$ la valeur prédite par le modèle.

Tâche 3 : Créez la fonction *mse* qui vous permet de calculer l'erreur quadratique moyenne des prédictions d'un modèle proposé.

Par exemple, pour des valeurs de α et β définies, nous pouvons obtenir les erreurs *MSE* suivantes :

```
Model (alpha=0.25, beta=0.5): 964.068
Model (alpha=0.75, beta=1): 18.656
Model (alpha=1, beta=2): 489.946
```

De plus, on pourra remarquer que les paramètres proposés pour ces 3 modèles ne sont pas les meilleurs :



Tâche 4 : Vérifiez que votre fonction `predict_model` vous fournit bien les erreurs *MSE* attendues.

Nous allons maintenant chercher les paramètres optimaux de notre modèle de régression.

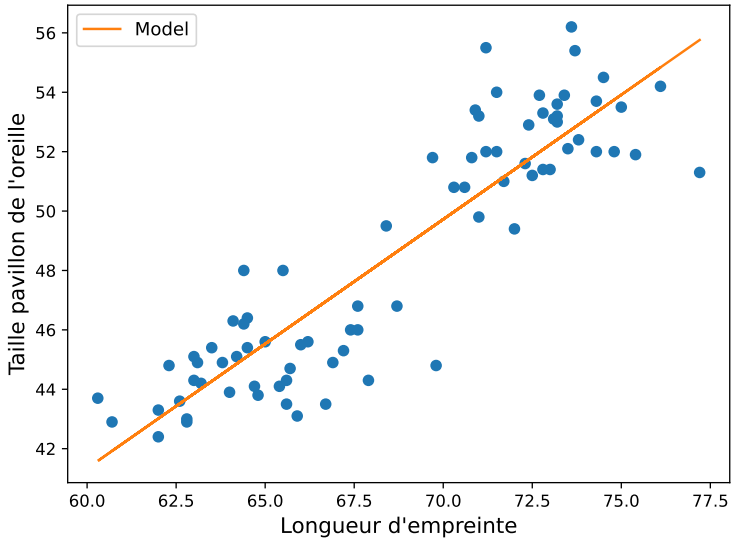
Avec $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$ et $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$, la solution de ce problème est donnée par :

$$\alpha = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

$$\beta = \bar{y} - \alpha\bar{x} \quad .$$

Tâche 5 : Calculez les coefficients α et β de la fonction linéaire correspondant le mieux à votre jeu de données dans le sens des moindres carrés (erreur *MSE*). Vous pouvez vérifier que l'erreur quadratique moyenne obtenue est plus faible que pour les autres α et β que vous avez essayé. Vous devriez obtenir une erreur proche de ≈ 3.262.

Vous devriez obtenir un modèle proposant cette régression :



Remarque : on constate que le modèle propose une approximation des données réelles. Il faut garder à l'esprit que ce type de modèle n'est pas forcément le meilleur prédicteur, mais peut rester suffisant pour l'utilisation et la tâche demandée.

3.4 Utilisation de scikit-learn

Vous allez maintenant utiliser la librairie Python `scikit-learn` pour créer notre modèle de régression linéaire (c'est tout de suite plus rapide...) :

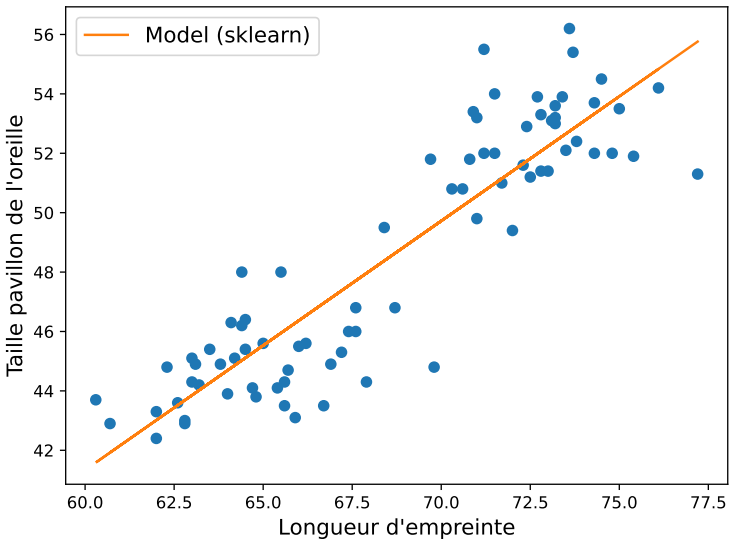
```
from sklearn.linear_model import LinearRegression
```

Tâche 6 : En utilisant la documentation de la librairie, utilisez les fonctions `fit` pour l'apprentissage et `predict` pour prédire les réponses du modèle sur les données.

Indication : il vous sera nécessaire de redimensionner vos données d'entrée correctement. Pour cela, vous pouvez utiliser `numpy` :

```
import numpy as np
# X_data les données d'entrée avec un descripteur
X_train_reshaped = np.array(X_train).reshape(-1, 1)
```

Affichez la droite de régression obtenue à partir des données d'apprentissage :



Utilisez la fonction de score proposée par scikit-learn pour calculer de nouveau les performances du modèle :

```
from sklearn.metrics import mean_squared_error
```

Remarque : nous allons dans la suite du TP principalement utiliser la librairie de `scikit-learn`.

4 Régression polynomiale

4.1 Mise en place de la base de données

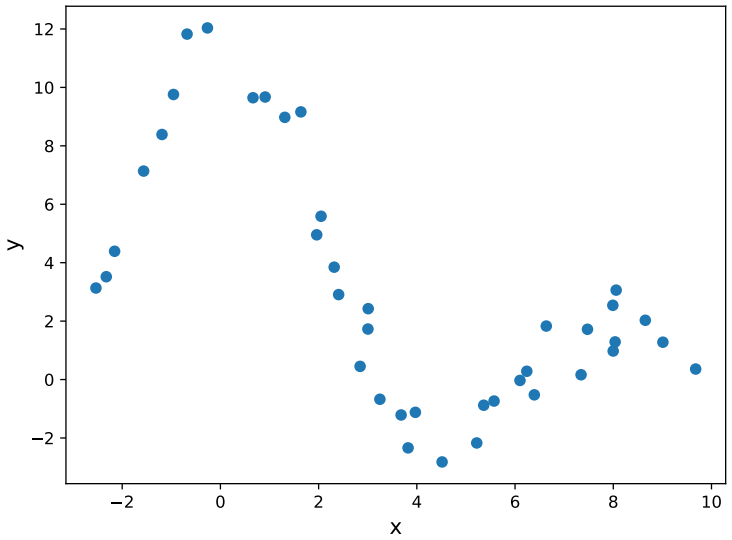
Nous allons travailler sur un nouvel ensemble de données. Nous allons chercher à créer un modèle qui approche au mieux la fonction suivante :

$$f(x) = 12 * \frac{sin(x)}{x + \epsilon}$$

Tâche 7 : Définir la fonction qui prend en entrée un ensemble de valeurs (un vecteur) et qui retourne pour chaque valeur x, sa valeur $f(x)$. Vous pouvez utiliser `numpy` pour simplifier votre fonction au maximum. La valeur ϵ évite la division par 0. Elle est disponible avec le package `numpy` : `np.finfo(np.float32).eps`.

Tâche 8 : Ajouter un bruit aléatoire de moyenne 0 et d'écart-type 1 à chaque point de la fonction. Générer un ensemble de 40 données à partir de la fonction f dans l'intervalle $[-3, 10]$.

Vous devez obtenir une base de données qui ressemble à :



4.2 Séparation des données

La différence entre une donnée prédite par le modèle et la donnée réelle est appelée *erreur résiduelle* ou *erreur d'apprentissage*. Au contraire, lorsque nous évaluons notre modèle sur une base de test indépendante, l'erreur résultante est appelée *erreur de prédiction* ou *erreur de test*.

On apprend généralement notre modèle sur des données d'apprentissage et l'on évalue ses performances sur la base de test dans l'objectif d'éviter le surapprentissage. Ce sont les performances de notre modèle sur cette base de test qui témoigneront de sa capacité à **généraliser**.

Tâche 9 : Nous allons proposer de découper notre base de données. Pour cela, nous allons utiliser la librairie `scikit-learn` qui permet de séparer rapidement les données.

On préserve ici un tiers des données pour la validation du modèle :

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,
                                                    test_size=0.33,
                                                    random_state=12)
```

4.3 Modèle de régression polynomiale

Un modèle de régression polynomiale s'écrit sous la forme :

$$y = \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_n x^n + \beta$$

avec n le degré souhaité du polynôme.

— Avec $n = 0$, quel type de modèle avons-nous ?

— Avec $n = 1$, quel type de modèle avons-nous ?

En l'état, nous ne savons pas quel degré de polynôme il nous faut utiliser pour disposer du meilleur modèle.

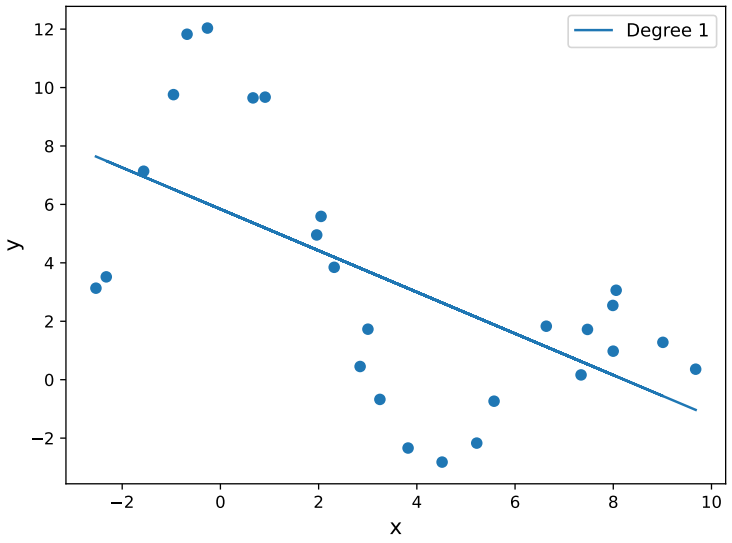
La première étape va consister à créer le modèle lui-même. Pour cela, nous allons utiliser un outil de transformation nommé `PolynomialFeatures` et proposé par `scikit-learn`. Il permet de générer une

nouvelle matrice de caractéristiques composée de toutes les combinaisons polynomiales des caractéristiques avec un degré inférieur ou égal au degré spécifié.

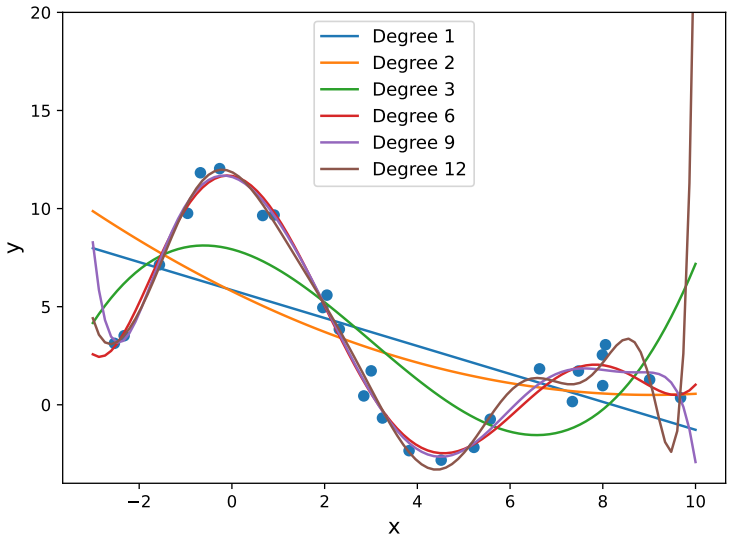
Tâche 10 : À l'aide de la librairie `scikit-learn` et sa fonction `make_pipeline`, créer un estimateur composé à la fois de la transformation des données et du modèle. On choisira comme modèle un `Ridge`.

Remarque : le modèle `Ridge` résout un modèle de régression où la fonction de perte est la fonction linéaire des moindres carrés (minimise la somme des carrés résiduels) et une régularisation (nommée norme l2) qui vise à réduire la complexité du modèle. le modèle `LinearRegression` minimise uniquement la somme des carrés résiduels (pour les curieux : régularisation l2).

Tâche 11 : Utiliser votre pipeline pour définir un modèle de degré 1. Faites apprendre ce modèle sur les données d'apprentissage et affichez ses performances :



Tâche 12 : Faites de même pour les degrés $n \in [1, 2, 3, 6, 9, 12]$. Elle permet de générer des données dans un intervalle donné. Vous devriez obtenir comme modèles :



Indication : pour afficher correctement votre modèle, vous pouvez utiliser la fonction `numpy` suivante `np.linspace(-3, 10, 100)`. En prédisant à partir de ces nouvelles données, vous obtiendrez un affichage plus précis et lisible de votre modèle.

Nous allons maintenant calculer l'erreur d'un modèle. Nous allons ici utiliser la somme des erreurs résiduelles au carrés et non l'erreur moyenne quadratique. La fonction d'erreur *RSS* est définie telle que :

$$RSS = \sum_{i=1}^{N} (y_i - f(x_i))^2$$

Tâche 13 : Créez cette nouvelle fonction d'erreur et évaluez l'erreur obtenue sur les données d'apprentissage pour les modèles de différents degrés.

Vous devriez par exemple, obtenir ce genre d'erreurs :

```
Degree 1: RSS of 293.5614299719814
Degree 2: RSS of 278.0674612643563
Degree 3: RSS of 179.71183511362062
Degree 6: RSS of 12.494652936224401
Degree 9: RSS of 11.163922829584456
Degree 12: RSS of 7.552538251342144
```

Tâche 14 : Que pouvez-vous en conclure ?

4.4 Validation du modèle

Après avoir appris sur la base d'apprentissage, il est important de vérifier les performances du modèle sur de nouvelles données.

Tâche 15 : Évaluez maintenant l'erreur obtenue sur les données de **test** pour les modèles de différents degrés.

Vous devriez par exemple, obtenir ce genre d'erreurs :

```
Degree 1: RSS of 110.59983095941676
Degree 2: RSS of 102.26283597799494
Degree 3: RSS of 49.72837170007651
Degree 6: RSS of 14.03292237285887
Degree 9: RSS of 14.906695677849315
Degree 12: RSS of 23.034153103566755
```

Tâche 16 : Que peut-on remarquer ?

Nous avons dans ce TP utilisé pour le moment deux métriques d'erreur : *RSS* et *MSE*. Une autre mesure bien connue pour les modèles de régression est le coefficient de détermination (aussi appelé R-Squared et noté R^2). Il est définit par :

$$R^2 = 1 - \frac{RSS}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

avec $\bar{y}$ la moyenne des observations.

La librairie `scikit-learn` propose également cette métrique. Il est également possible de récupérer la mesure R^2 directement depuis notre modèle à partir de sa méthode score.

Tâche 17 : Calculez avec cette nouvelle mesure les performances des modèles à la fois sur la base d'apprentissage et de test.

```
[Train]
Degree 1: R^2 of 0.37502913454266606
Degree 2: R^2 of 0.4080146634437124
Degree 3: R^2 of 0.6174066152539013
Degree 6: R^2 of 0.9733997955389219
Degree 9: R^2 of 0.9762328228426663
Degree 12: R^2 of 0.9839211971143776

[Test]
Degree 1: R^2 of 0.4025324445806352
Degree 2: R^2 of 0.4475694393742471
Degree 3: R^2 of 0.7313640679475149
Degree 6: R^2 of 0.9241932311037783
Degree 9: R^2 of 0.9194730502861882
Degree 12: R^2 of 0.8755679911392156
```

5 Remise des travaux

N'oubliez pas de réaliser un commit de vos travaux. Taguez également votre projet avec le tag « `tp1` » et soumettez-le sur le serveur Gitlab. Il fera office de rendu.