13 Introduction à l'apprentissage automatisé

13.1 L'Intelligence Artificielle, qu'est ce que c'est, à quoi ça sert ?

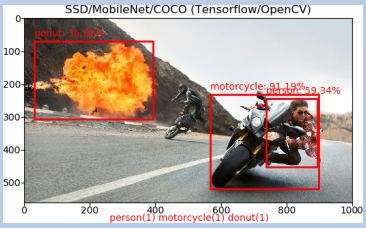
13.1.1 Premières définitions – IA, données

Définition – Intelligence Artificielle

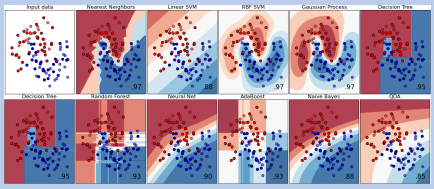
Wikipedia – L'intelligence artificielle (IA) est « l'ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence ».

En première approche, l'IA peut être assimilée à un ensemble d'algorithmes permettant de réaliser des tris, des classifications.

Exemple –



Identification d'objets, en temps réel, dans une vidéo.



Comparaison du résultats d'algorithmes de classification permettant de classifier un nuage de points bleus et rouges de transparences différentes.

On remarquera sur l'exemple de gauche que l'algorithme parvient à identifier sur l'image des zones puis à identifier à quoi elles correspondent. On remarquera aussi que la flamme est identifiée comme étant un donut...

`exploring-opencvs-deep-learning-object-detection-library` & `https://scikit-learn.org/`.

Pour utiliser un algorithme d'IA il est nécessaire de disposer de données... en général beaucoup de données.

13.1 L'Intelligence Artificielle, qu'est ce que c'est, à quoi ça sert ? 1
13.2 Mécanismes d'apprentissages 5
13.3 Algorithmes d'apprentissage 8

- ► Analyser les principes d'intelligence artificielle.
  - Régression et classification, apprentissages supervisé et non supervisé.
  - Phases d'apprentissage et d'inférence.
  - Modèle linéaire monovariable ou multivariable.
- ► Interpréter et vérifier la cohérence des résultats obtenus expérimentalement, analytiquement : Ordre de grandeur. Matrice de confusion (tableau de contingence), sensibilité et spécificité d'un test.
- ► Résoudre un problème en utilisant une solution d'intelligence artificielle :
  - Apprentissage supervisé.
  - Choix des données d'apprentissage.
  - Mise en œuvre des algorithmes (*k* plus proches voisins et régression linéaire multiple).
  - Phases d'apprentissage et d'inférence.

Définition – Données

Les données utilisées par un algorithme d'IA peuvent être sous différentes formes :

- des données quantitatives continues qui peuvent prendre n'importe quelles valeurs dans un ensemble de valeurs (par exemple la température) ;
- des données quantitatives discrètes qui peuvent prendre un nombre limité de valeurs dans un ensemble de valeurs (par exemple nombre de pièces d'un logement) ;
- des données qualitatives (ordinales ou nominales suivant qu'on puissent les classer ou non, par exemple des couleurs, des notes à un test d'opinion ...).

Pour pouvoir traiter ces données, il peut être nécessaire qu'elles soient organisées sous une certaine forme. On peut par exemple identifier :

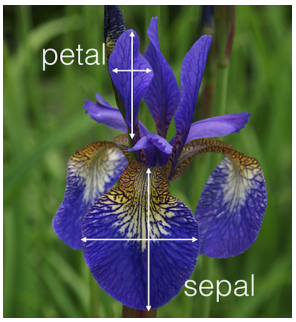
- les données structurées, dans une base de données par exemple ;
- les données semi-structurées, dans un fichier csv, xml ou json par exemple ;
- les données non structurées comme un image, du texte, ou une vidéo.

Exemple –

Le site `https://www.kaggle.com/` partage des sets de données.
On peut par exemple retrouver un dataset semi-strucutrées au format JSON ou CSV des chansons disponibles sur Spotify, sorties entre 1922 et 2021. (`https://www.kaggle.com/yamaerenay/spotify-dataset-19212020-160k-tracks`).
Le site `https://storage.googleapis.com/` propose des sets de photos étiquetées selon 600 classes.

Définition – Observations – Caractéristiques

On appelle observation (ou individu ou objet) une « ligne de donnée » qui va être utilisée par un algorithme d'apprentissage.
Une observation est composée de caractéristiques qui varient pour chacun des observations.



Exemple –

Dans le cas de la base de donnée des Iris, présente par exemple dans `scikit-learn`, les données sont composées de 150 observations, chacune composée de 4 caractéristiques : longueur et largeur du sépale ainsi que longueur et largeur du pétale.

Définition – Big data – Wikipedia

Mégadonnées ou données massives. Le big data désigne les ressources d'informations dont les caractéristiques en termes de volume, de vélocité et de variété imposent l'utilisation de technologies et de méthodes analytiques particulières pour générer de la valeur, et qui dépassent en général les capacités d'une seule et unique machine et nécessitent des traitements parallélisés.

Que fait-on avec ces données ? En général les algorithmes vont chercher un lien entre ces données. Dans le cas où il existe un lien connu par le *Data Scientist* on parle d'apprentissage supervisé.

Si ce lien n'est pas connu, on parle d'apprentissage non supervisé ou de clustering.

13.1.2 Apprentissage automatisé – Machine Learning

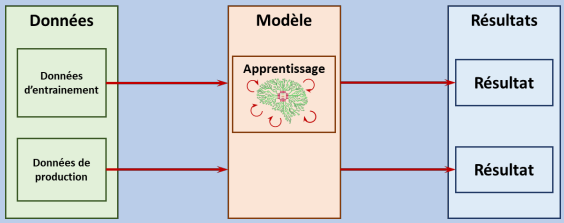
Définition – Apprentissage automatique – Machine learning

L'apprentissage automatique est un champ de l'intelligence artificielle dont l'objectif est d'analyser un grand volume de données afin de déterminer des motifs et de réaliser un modèle prédictif.
L'apprentissage comprend deux phases :

- l'entraînement (ou apprentissage) est une phase d'estimation du modèle à partir de données d'observations ;
- la mise en production du modèle (inférence) est une phase pendant laquelle de nouvelles données sont traitées dans le but d'obtenir le résultat souhaité.

L'entraînement peut être poursuivi même en phase de production.

Exemple –



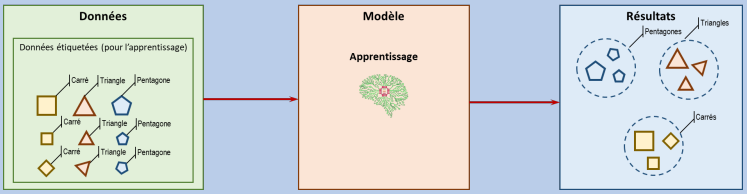
Définition – Apprentissage supervisé – Apprentissage

Tâche d'apprentissage au cours duquel l'algorithme (ou fonction de prédiction) va, à partir d'un ensemble de données **étiquetées**, déterminer un lien entre un ensemble les données et les étiquettes.

Définition – Apprentissage supervisé – Inférence

Tâche au cours duquel l'algorithme (ou fonction de prédiction) va, à partir d'un ensemble de données **non étiquetées**, prédire l'étiquette.

Exemple –



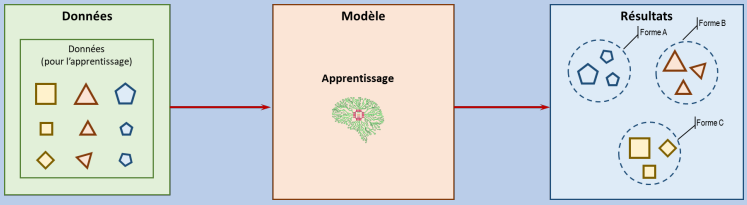
Exemple –

Soit un ensemble d'images en noir et blanc représentant des chiffres de 0 à 9. L'algorithme doit dans un premier temps *apprendre* le lien entre l'image et le chiffre. Dans un second temps, l'algorithme devra déterminer le chiffre en fonction d'une image seule.

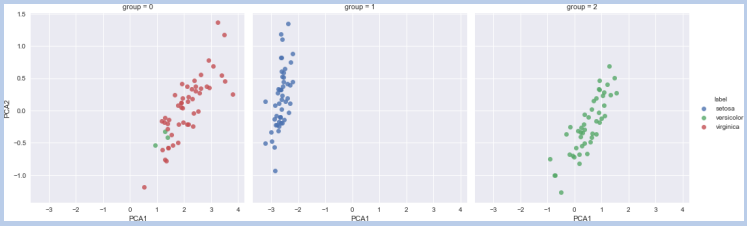
Définition – Apprentissage non supervisé – Clustering

Tâche d'apprentissage au cours duquel l'algorithme (ou fonction de prédiction) va, à partir d'un ensemble de données **non étiquetées**, déterminer un lien entre les données (et les regrouper).

Exemple –



Classification d'iris en utilisant un algorithme d'apprentissage non supervisé.

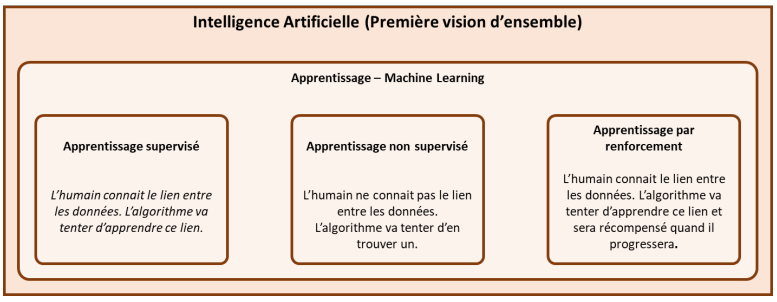


On remarque (lorsque l'image est en couleur) que sur des mesures sur 150 iris, l'algorithme a réussi à retrouver les 3 différentes espèces, sans les connaître, à 3 erreurs près.

Définition – Apprentissage par renforcement

Si au cours de l'apprentissage supervisé, un mécanisme de récompense est mis en œuvre pour améliorer les performances du modèle, on parle d'apprentissage par renforcement.

On peut donc faire une synthèse des méthodes d'apprentissage dont nous avons pris connaissance (il en existe d'autres).



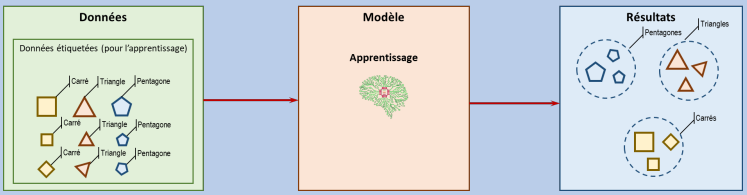
Comment situer le deep learning parmi ces méthodes d'apprentissage ? – Pour moi, l'utilisation du terme « deep learning » apparaît lorsqu'on commence à utiliser des réseaux de neurones... mais les réseaux de neurones peuvent être utilisés dans chacune des méthodes d'apprentissage sus-citées...

13.1.3 Encore des définitions...

Définition – Classification

En apprentissage automatique, on appelle problème de classification les problèmes ou les étiquettes sont discrètes.

Exemple –



Dans le cas où nous souhaitons regrouper par forme, il s'agit d'un problème de classification. Les classes discrètes sont les formes (triangles, carrés, pentagones ...). Il serait aussi possible d'opérer un regroupement par couleur. Les classes seraient donc différentes.

Définition – Régression

En apprentissage automatique, on appelle problème de régression les problèmes ou les étiquettes sont continues.

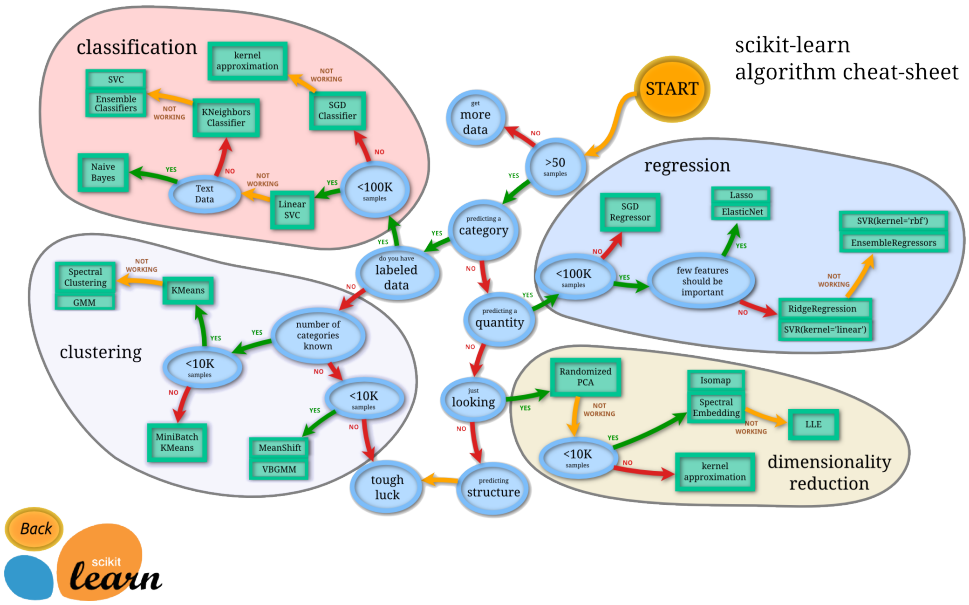
Exemple –

Dans le cas où nous souhaiterions que l'algorithme détermine l'aire ou le périmètre de la forme, les étiquette seraient alors continues sur $\mathbb{R}_*^+$.

13.2 Mécanismes d'apprentissages

13.2.1 Classification des algorithmes d'apprentissage

La bibliothèque `scikit-learn` propose une classification de divers algorithmes en fonction des apprentissages automatiques (hors réseaux de neurones).



13.2.2 Validation du modèle

Une fois un algorithme d'apprentissage choisi, on se pose la question de la validation du modèle. Quels critères et outils vont nous permettre de considérer que notre apprentissage est « bon » ?

Lors d'un problème de classification, il est par exemple possible de déterminer les écarts entre les valeurs prédites par l'algorithme et les valeurs cibles.

Critères de validation des problèmes de classification

Définition – Valeur prédictive positive

Valeur prédictive positive (*accuracy classification score*) :

$$\text{Justesse} = \frac{\text{Nombre de prédictions vraies}}{\text{Nombre de prédictions totales}}.$$

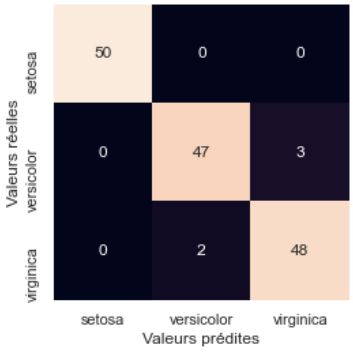
```python
import sklearn.metrics as skm

# X(numpy.ndarray) : données d'entrées
# Y(numpy.ndarray) : données cibles correspondantes
# Y_pred(numpy.ndarray) : données prédites par l'algorithme de classification

print(skm.accuracy_score(Y, Y_pred))
```

Définition – Matrices de confusion

La matrice de confusion est une métrique permettant de déterminer la qualité d'une classification. En abscisses sont indiquées le valeurs réelles, et on ordonnée les valeurs prédites.

Exemple –

Dans l'exemple ci-contre, nous cherchons à classifier des iris, grâce à un algorithme, selon 3 familles : les setosa, les versicolor et les virginica.
Sur la diagonale, sont retrouvées les iris dont l'espèce a été correctement prédite. En revanche, 3 versicolor ont été classées parmi les virginica et 2 virginica ont été classifiées dans les versicolor.



Remarque

La matrice de confusion présentée est dite non-normalisée. Si la répartition des étiquettes n'est pas uniforme (en fonction des classes), il est probable qu'il y ait davantage d'erreurs pour les classes ayant beaucoup d'étiquettes. Pour palier ce problème on peut utiliser des matrices de confusions normalisées (on divise alors chaque terme de la matrice par la somme des éléments de la même ligne).

Critères de validation des problèmes de régression

Définition – Erreur moyenne quadratique

Appelée également *mean squared error*, il s'agit d'une moyenne d'écart au carré :

$$msq = \frac{1}{N}\sum_{i=1}^{n} \left\| Y_i - \hat{Y}_i \right\|^2$$

en notant Y_i la valeur réelle (étiquetée) et $\hat{Y}_i$ la valeur estimée par l'algorithme.

13.2.3 Séparation des données/Gestion des données ?

Lorsque l'on souhaite disposer d'un modèle défini par un algorithme d'IA, il faut en premier lieu disposer de données.

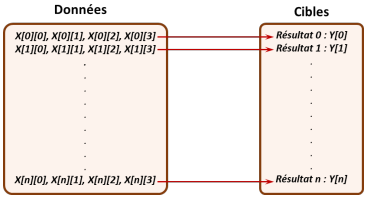
Types de données – Apprentissage supervisé

En apprentissage supervisé, il est nécessaire de connaître des données d'entrées ainsi que les sorties correspondantes (appelées aussi `cibles` ou `target`).

Dans le cadre d'une programmation `Python` avec la bibliothèque `scikit-learn` les données d'entées et de sorties peuvent être implémentées en utilisant des `numpy.ndarray`.

Ainsi, si les données d'entrées, stockées dans la variable `data` sont composées de `n` observations elles mêmes composées de `p` catégories, le shape de `data` sera `(n,p)`.

La donnée cible sera quant à elle un vecteur de `n` valeurs.



Exemple –

Données d'entrées : matrice `X` de `n` observations avec 4 caractéristiques.
Données de sortie : vecteur `Y` de `n` résultats.
Dans le cadre d'une classification, où `r` résultats sont possibles. On peut attribuer une valeur (entière) entre 1 et `r` à chacun des résultats, notamment si ceux-ci sont des données qualitatives.

1: https://scikit-learn.org/stable/modules/preprocessing.html

Normalisation des données [1]

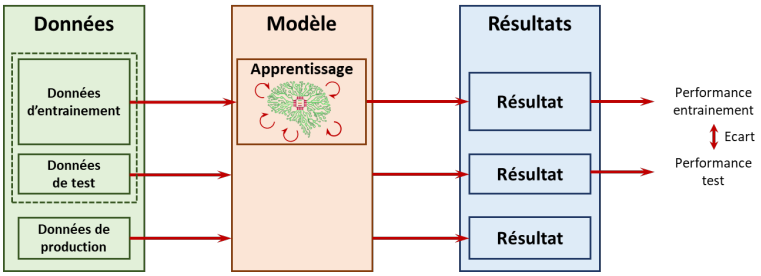
Séparation des données

Lors du démarrage d'un apprentissage supervisé, on dispose d'un jeu de données comprenant les données d'entrée et les cibles correspondantes. Il est d'usage de séparer ces données en deux parties :

- les données d'entraînement permettant... d'entraîner le modèle. Dans la pratique on utilise entre 60 et 80 % des données de base ;
- les données de te test, permettant de valider l'apprentissage (ou le modèle), dans la pratique entre 20 et 40 % des données de base.

On commence donc par réaliser un apprentissage sur les données d'entraînement. Cet entraînement produit des résultats. Connaissant les cibles correspondant aux données d'entraînement, on peut donc en déduire une performance du modèle sur le traitement des données d'entraînement.

On utilise alors le modèle en lui donnant les données de test. De même, on peut donc en déduire une performance du modèle sur le traitement des données de test.



Se pose alors le problème de comment séparer les données. En effet, les données de base pouvant potentiellement être triées (ordonnées par rapport à une des caractéristiques), et sélectionner une partie pourrait créer un biais dans l'apprentissage.

`scikit-learn` permet de séparer aléatoirement des données en fixant le pourcentage de données de validation.

```
from sklearn.model_selection import train_test_split
# X(numpy.ndarray) : données d'entrées
# Y(numpy.ndarray) : données cibles correspondantes
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33)
```

On perçoit donc que la qualité de l'apprentissage peut dépendre du choix utilisé lors de la séparation des données. De plus, certains choix de paramètres permettant d'affiner le modèle sont aussi liés aux données d'entraînement sélectionnées. Une des solutions pour résoudre ce problème est d'avoir recours à la validation croisée[2].

2: https://scikit-learn.org/stable/modules/cross_validation.html

13.3 Algorithmes d'apprentissage

13.3.1 Algorithme des k plus proches voisins

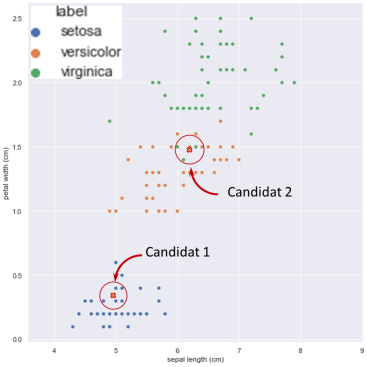
Présentation

L'algorithme des k plus proches voisins (aussi appelé k-nearest neighbors algorithm – k-NN) permet de réaliser des opérations de régression et de classification sur un ensemble de données.

Pour une première approche, cette méthode permet d'estimer la classe d'une nouvelle donnée en déterminant la norme entre cette nouvelle donnée et l'ensemble des données du jeu d'entraînement. Parmi les k plus proches voisins, on recherche la classe majoritaire et on attribue cette classe à la nouvelle donnée.

Exemple –

Dans le cas ci-contre, on cherche à classifier les iris selon 3 catégories (setosa, versicolor et virginca). Les données étiquetées sont représentées dans le plan longueur du sépale en abscisse et largeur du pétale en ordonnée.
Soient les candidats 1 et 2, deux iris dont ont connaît les caractéristiques mais pas l'espèce. Les 5 plus proches voisins du candidat 1 sont des setosa. Il est donc probable que le candidat 1 soit un setosa aussi.
Concernant le candidat 2, parmi les 5 plus proches voisins, 3 sont des virginica et 2 sont des versicolor. On peut faire l'hypothèse que c'est un virginica.



L'algorithme

Prenons comme exemple des données sous la forme `[x,y,c]` où `x` et `y` représentent des coordonnées et `c` la classe d'appartenance.

Première étape – Calculer la distance Il faut tout d'abord définir une fonction de distance. On peut par exemple utiliser la distance euclidienne. Pour deux vecteurs x_1 et x_2 de taille N, $d = \sqrt{\sum_{i=1}^{n} (x_{1,i} - x_{2,i})^2}$.

```
import math as m
def distance(x1:list, x2:list) -> float :
    distance = 0.
    for i in range(len(x1)):
        distance += (x1[i]-x2[i])**2
    return m.sqrt(distance)
```

Deuxième étape – Détermination des plus proches voisins Pour cette étape, on dispose de données pour l'entraînement `data_train` et d'une donnée à tester `data_test`. Il va falloir calculer la distance entre la donnée à tester et les données d'entraînement puis trier les données.

Enfin, on retournera les k lignes les plus proches de `data_test`.

```
def get_voisins(data_train:list, data_test, k:int) -> list :
    distances =[]
    for ligne in data_train :
        d = distance(ligne,data_test)
        distances.append([ligne,d])
    # Tri de la liste suivant la seconde colonne (distances)
    distances.sort(key=lambda t : t[1])
    voisins = []
    for i in range(k) :
        voisins.append(distance[i][0])
    return voisins
```

Troisième étape – Prédiction Une fois qu'on connaît les k plus proches voisins, on regarde quelle est la classe majoritaire parmi ces voisins.

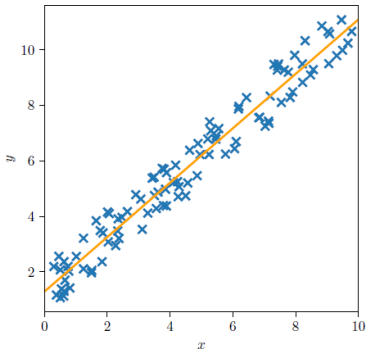
```python
def prediction(data_train:list, data_test, k:int) -> list :
    voisins = get_voisins(data_train, data_test, k)
    sorties = [ligne[-1] for ligne in voisins]
    predict = max(set(sorties), key=sorties.count)
    return predict
```

Prédiction avec `scikit-learn` Un exemple rapide pour réaliser une classification k-NN avec `scikit-learn`.

```python
import numpy as np
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier

iris_X, iris_y = datasets.load_iris(return_X_y=True)


# A random permutation, to split the data randomly
np.random.seed(0)

# Chargement des données et séparation des données
indices = np.random.permutation(len(iris_X))
iris_X_train = iris_X[indices[:-10]]
iris_y_train = iris_y[indices[:-10]]
iris_X_test = iris_X[indices[-10:]]
iris_y_test = iris_y[indices[-10:]]

# Création du classifieur
knn = KNeighborsClassifier() # KNeighborsClassifier(n_neighbors=k) pour
    choisir k
# Entraînement du classifieur
knn.fit(iris_X_train, iris_y_train)
# Prédiction sur le jeu de test (à comparer avec iris_y_test)
knn.predict(iris_X_test)
```

13.3.2 Régression univariée [3]

3: Chloé-Agathe Azencott



Dans le cadre de la régression linéaire unvariée, on dispose de n observations : $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_i, y_i), ..., (x_n, y_n)\}$ et des étiquettes y_i associées. La régression linéaire consiste à choisir une fonction de prédiction f de la forme, $\forall x \in \mathbb{R}, f(x) = ax+b$ avec$(a, b) \in \mathbb{R}^2$. Il va donc falloir minimiser l'erreur entre les prédictions et la base de données d'entraînement. La minimisation se note :

$$\left(\hat{a}, \hat{b}\right) = \arg \min_{(a,b)\in\mathbb{R}^2} \frac{1}{n} \sum_{i=1}^{n} (y_i - (ax_i + b))^2 .$$

(Méthode des moindres carrés)

On montre que si $\sum_{i=1}^{n} x_i^2 \neq \left(\sum_{i=1}^{n} x_i\right)^2$ le problème à une unique solution. Sinon le système est indéterminé et il y a une infinité de solutions.

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```

Bibliographie

1 Éric Biernat et Michel Lutz. *Data science : fondamentaux et études de cas.* Eyrolles.