Pratique de la fouille de données

https://github.com/ylebbah/teaching

Yahia Lebbah
Laboratoire LITIO, Faculté FSEA, Equipe PPCO
Université Oran1, Algérie

Table des matières

1 Préliminaires

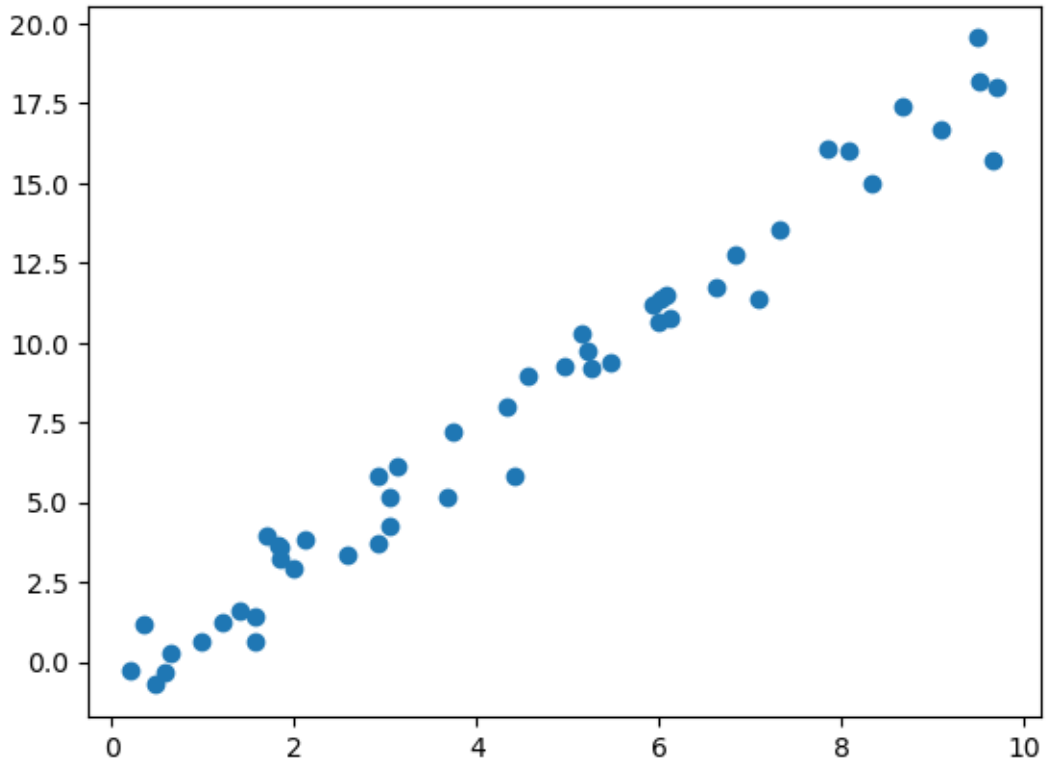
1.1 Exemple 1

N°	N (Azote)	R (Rendement)
1	3.7454	7.2293
2	9.5071	18.1857
3	7.3199	13.5242
4	5.9866	10.6721
5	1.5602	0.6419
6	1.5599	1.4000
7	0.5808	_
8	8.6618	17.3806
9	6.0112	11.3659
10	7.0807	11.3984
11	0.2058	_
12	9.6991	18.0131
13	8.3244	14.9719
14	2.1234	3.8585
15	1.8182	3.6675
16	1.8340	3.5994
17	3.0424	4.2456
18	5.2476	9.1859
19	4.3195	7.9702
20	2.9123	5.8001
21	6.1185	10.7579
22	1.3949	1.6042
23	2.9214	3.7366
24	3.6636	5.1310
25	4.5607	8.9339
26	7.8518	16.0598

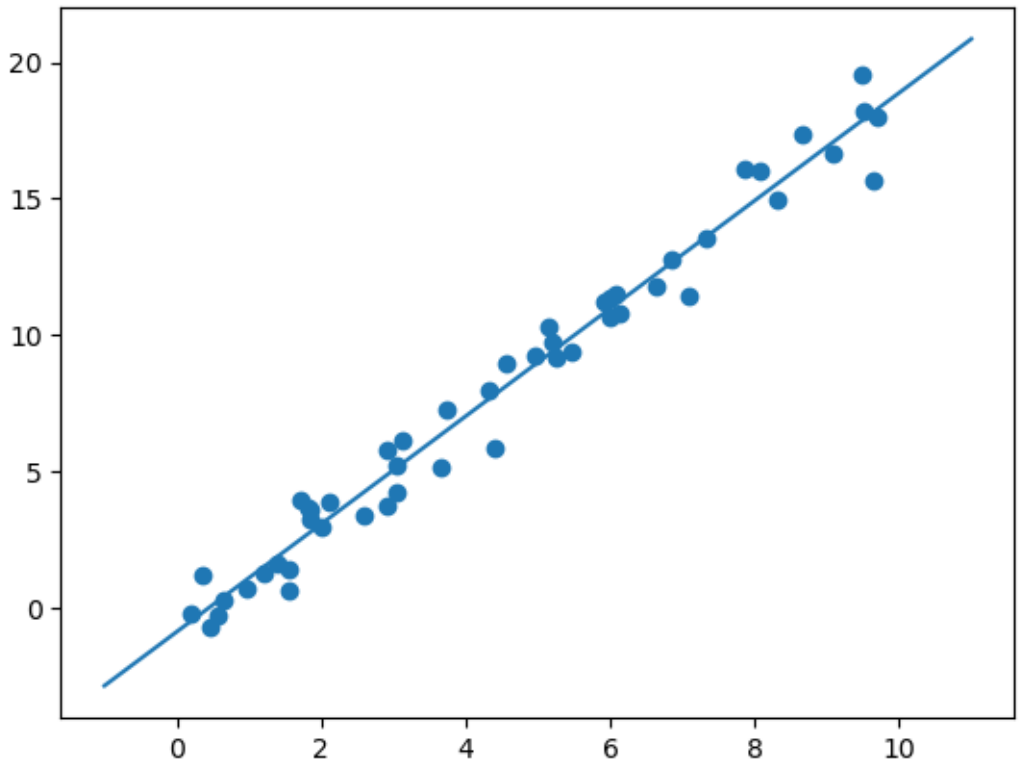
N°	N	R
27	1.9967	2.9215
28	5.1423	10.2882
29	5.9241	11.2099
30	0.4645	_
31	6.0754	11.5123
32	1.7052	3.9485
33	0.6505	0.2652
34	9.4889	19.5424
35	9.6563	15.6929
36	8.0840	15.9898
37	3.0461	5.1793
38	0.9767	0.6544
39	6.8423	12.7764
40	4.4015	5.8155
41	1.2204	1.2211
42	4.9518	9.2607
43	0.3439	1.1657
44	9.0932	16.6681
45	2.5878	3.3671
46	6.6252	11.7487
47	3.1171	6.1496
48	5.2007	9.7301
49	5.4671	9.4044
50	1.8485	3.2104
51	5.4671	9.4044
52	1.8485	3.2104
	.....	.....

Peut-on approximer ces données par un modèle ?

Visualisons ces données 2D !



Idée ... calculer la droite médiane du nuage des points !

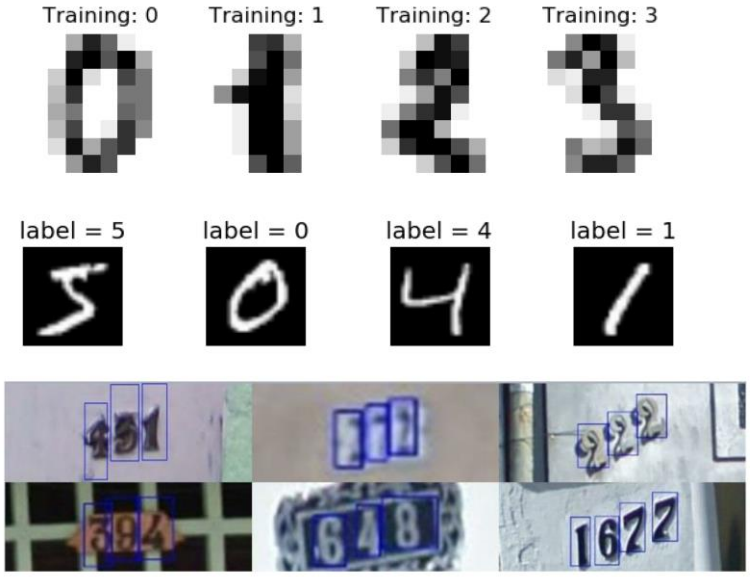


$$\mathbf{R = 1.9776566\ N}$$

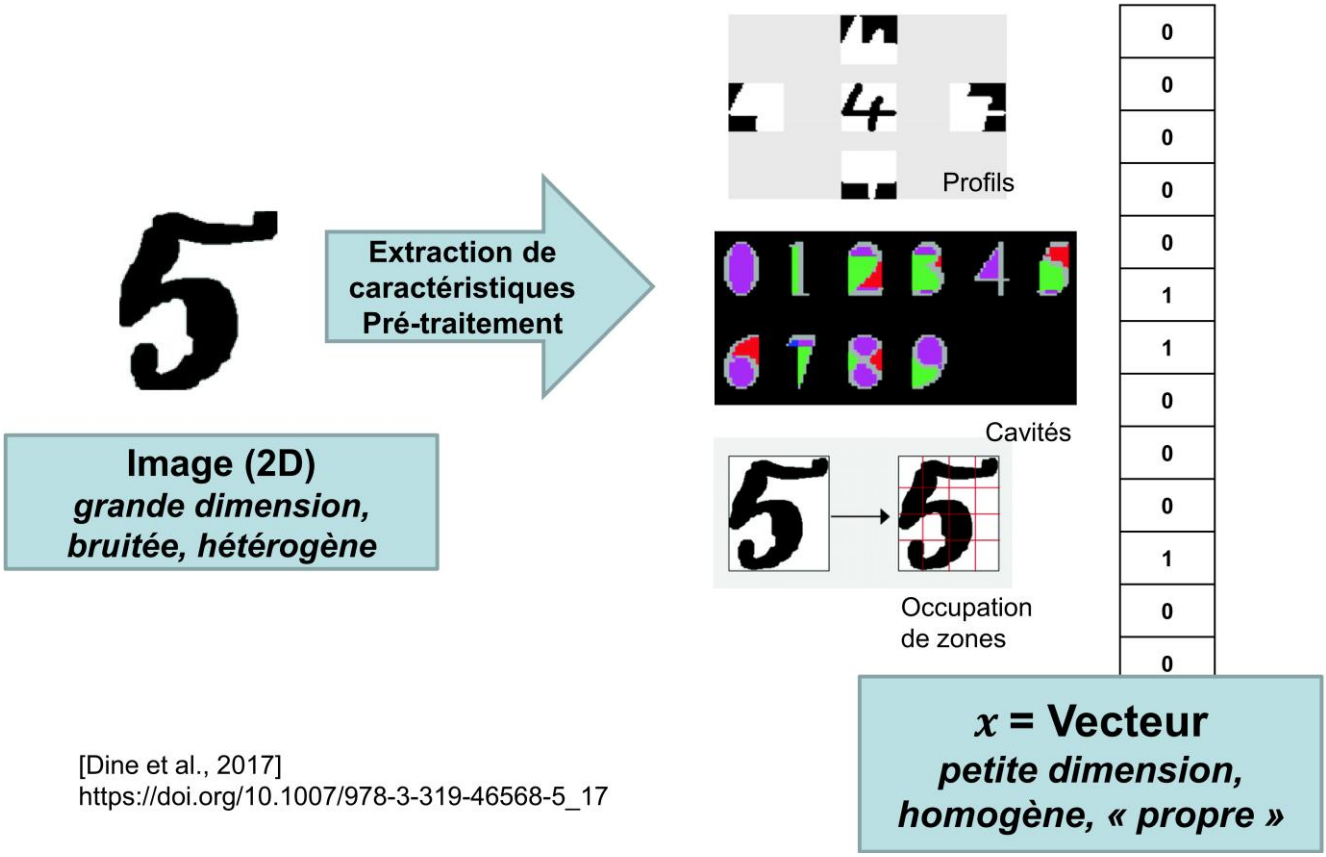
1.2 Exemple 2

Etape 1

Etape 2



Etape 3 : formatage des données



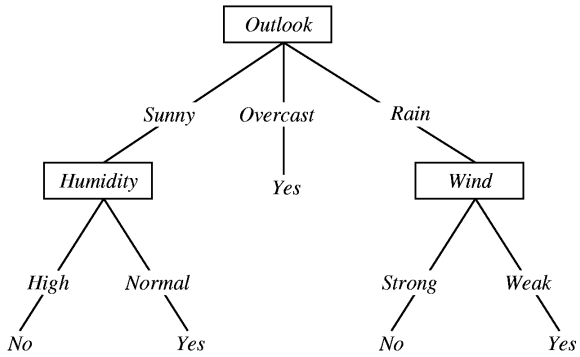
[Dine et al., 2017]
https://doi.org/10.1007/978-3-319-46568-5_17

Attribut 1	Attribut 2	….	Attribut n	Classe
…	…	…	…	…
…	…	…	…	…

Etape 4 : Choix du modèle

Régression : $y = \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n + \beta$

Arbre de décision :



SVM : …

Naive-bayes : …

…

Deep-learning (Réseaux de neurones) : …

1.3 Environnement de travail

Pour faciliter la mise en pratique de l'atelier il est nécessaire de :

- Disposer d'un laptop sous windows ou linux.
- Installer un environnement de travail sous python, de préférence Anaconda https://www.anaconda.com/download

2 Python par l'exemple

2.1 Indentation

Le langage Python exige une indentation stricte pour déclarer les blocs du programme. Exemple :

```
for i in [1, 2, 3, 4, 5]:
    print(i) # first line in "for i" block
    for j in [1, 2, 3, 4, 5]:
        print(j) # first line in "for j" block
        print(i + j) # last line in "for j" block
    print(i) # last line in "for i" block
print("done looping")
```

2.2 Fonctions

```
def double(x):
    """this is where you put an optional docstring that explains what the function does.
    for example, this function multiplies its input by 2"""
    return x * 2
```

Le langage Python est du premier ordre ; on peut introduire des fonctions en argument. Exemple :

```
def apply_to_one(f):
    """calls the function f with 1 as its argument"""
    return f(1)
```

et par la suite, on peut écrire :

```
my_double = double # refers to the previously dened function
x = apply_to_one(my_double) # equals 2
print(x)
```

2.3 Expressions : Chaines de caractères, exp. logiques

Les chaines sont avec cotes ou en double-cotes :

```
single_quoted_string = 'data science'
double_quoted_string = "data science"
tab_string = "\t" # represents the tab character
len(tab_string) # is 1
multi_line_string = """This is the first line
    and this is the second line
    and this is the third line"""
print(multi_line_string)
```

Logique

```
one_is_less_than_two = 1 < 2 # equals True
true_equals_false = True == False # equals False
x = None
print(x == None) # prints True, but is not Pythonic
print(x is None) # prints True, and is Pythonic
```

2.4 Exception

```
try:
    print(0/0)
except ZeroDivisionError:
    print("cannot divide by zero")
```

2.5 Listes, tuples, comprehension, ensemble

Listes

```
integer_list = [1, 2, 3]
heterogeneous_list = ["string", 0.1, True]
list_of_lists = [ integer_list, heterogeneous_list, [] ]
list_length = len(integer_list) # equals 3
list_sum = sum(integer_list) # equals 6
x = range(10) # is the list [0, 1, ..., 9]
zero = x[0] # equals 0, lists are 0-indexed
one = x[1] # equals 1
nine = x[-1] # equals 9, 'Pythonic' for last element
eight = x[-2] # equals 8, 'Pythonic' for next-to-last element
first_three = x[:3] # [-1, 1, 2]
three_to_end = x[3:] # [3, 4, ..., 9]
one_to_four = x[1:5] # [1, 2, 3, 4]
last_three = x[-3:] # [7, 8, 9]
without_first_and_last = x[1:-1] # [1, 2, ..., 8]
copy_of_x = x[:] # [-1, 1, 2, ..., 9]
1 in [1, 2, 3] # True
0 in [1, 2, 3] # False
x = [1, 2, 3]
x.extend([4, 5, 6]) # x is now [1,2,3,4,5,6]
x = [1, 2, 3]
y = x + [4, 5, 6] # y is [1, 2, 3, 4, 5, 6]; x is unchanged
x = [1, 2, 3]
x.append(0) # x is now [1, 2, 3, 0]
y = x[-1] # equals 0
z = len(x) # equals 4
x, y = [1, 2] # now x is 1, y is 2
_, y = [1, 2] # now y == 2, didn't care about the first element
```

Tuples

```
my_list = [1, 2]
my_tuple = (1, 2)
other_tuple = 3, 4
my_list[1] = 3 # my_list is now [1, 3]
try:
    my_tuple[1] = 3
except TypeError:
    print("cannot modify a tuple")
def sum_and_product(x, y):
    return (x + y),(x * y)
sp = sum_and_product(2, 3) # equals (5, 6)
s, p = sum_and_product(5, 10) # s is 15, p is 50
x, y = 1, 2 # now x is 1, y is 2
x, y = y, x # Pythonic way to swap variables; now x is 2, y is 1
print(x + y)
```

Liste en compréhension

```
even_numbers = [x for x in range(5) if x % 2 == 0] # [0, 2, 4]
squares = [x * x for x in range(5)] # [0, 1, 4, 9, 16]
even_squares = [x * x for x in even_numbers] # [0, 4, 16]
square_dict = { x : x * x for x in range(5) } # { 0:0, 1:1, 2:4, 3:9, 4:16 }
```

```
square_set = { x * x for x in [1, -1] } # { 1 }
pairs = [(x, y)
   for x in range(10)
   for y in range(10)] # 100 pairs (0,0) (0,1) ... (9,8), (9,9)
increasing_pairs = [(x, y) # only pairs with x < y,
   for x in range(10) # range(lo, hi) equals
   for y in range(x + 1, 10)] # [lo, lo + 1, ..., hi - 1]
```

Ensembles

```
s = set()
s.add(1) # s is now { 1 }
s.add(2) # s is now { 1, 2 }
s.add(2) # s is still { 1, 2 }
x = len(s) # equals 2
y = 2 in s # equals True
z = 3 in s # equals False
stopwords_list = ["a","an","at"] + hundreds_of_other_words + ["yet", "you"]
"zip" in stopwords_list # False, but have to check every element
stopwords_set = set(stopwords_list)
"zip" in stopwords_set # very fast to check
item_list = [1, 2, 3, 1, 2, 3]
num_items = len(item_list) # 6
item_set = set(item_list) # {1, 2, 3}
num_distinct_items = len(item_set) # 3
distinct_item_list = list(item_set) # [1, 2, 3]
```

2.6 Dictionnaires

```
from collections import defaultdict
empty_dict = {} # Pythonic
empty_dict2 = dict() # less Pythonic
grades = { "Jaber" : 80, "Tarik" : 95 } # dictionary literal
jaber_grade = grades["Jaber"] # equals 80
try:
   karim_grade = grades["Karim"]
except KeyError:
   print("no grade for Karim!")
jaber_has_grade = "Jaber" in grades # True
karim_has_grade = "Karim" in grades # False
jaber_grade = grades.get("Jaber", 0) # equals 80
karim_grade = grades.get("Kariom", 0) # equals 0
no_ones_grade = grades.get("No One") # default default is None
grades["Tarik"] = 99 # replaces the old value
grades["Karim"] = 100 # adds a third entry
num_students = len(grades) # equals 3
tweet = {
   "user" : "jabergrus",
   "text" : "Data Science is Awesome",
   "retweet_count" : 100,
   "hashtags" : ["#data", "#science", "#datascience", "#awesome", "#yolo"]
}
tweet_keys = tweet.keys() # list of keys
```

```
tweet_values = tweet.values() # list of values

tweet_items = tweet.items() # list of (key, value) tuples
"user" in tweet_keys # True, but uses a slow list in
"user" in tweet # more Pythonic, uses faster dict in
"jabergrus" in tweet_values # True
dd_list = defaultdict(list) # list() produces an empty list
dd_list[2].append(1) # now dd_list contains {2: [1]}
dd_dict = defaultdict(dict) # dict() produces an empty dict
dd_dict["Jaber"]["City"] = "Oran" # { "Jaber" : { "City" : Oran"}}
dd_pair = defaultdict(lambda: [0, 0])
dd_pair[2][1] = 1 # now dd_pair contains {2: [0,1]}
```

2.7 Alternatives et boucles

```
if 1 > 2:
   message = "if only 1 were greater than two..."
elif 1 > 3:
   message = "elif stands for 'else if'"
else:
   message = "when all else fails use else (if you want to)"
parity = "even" if x % 2 == 0 else "odd"
x = 0
while x < 10:
   print(x, "is less than 10")
   x += 1
for x in range(10):
   print(x, "is less than 10")
   for x in range(10):
      if x == 3:          continue # go immediately to the next iteration
   if x == 5:
      break # quit the loop entirely
print(x)
```

2.8 Tri

```
x = [4,1,2,3]
y = sorted(x) # is [1,2,3,4], x is unchanged
print(x.sort()) # now x is [1,2,3,4]
# sort the list by absolute value from largest to smallest
x = sorted([-4,1,-2,3], key=abs, reverse=True) # is [-4,3,-2,1]
print(x)
```

2.9 Programmation Objet

```
# by convention, we give classes PascalCase names
class Set:
   # these are the member functions
   # every one takes a first parameter "self" (another convention)
   # that refers to the particular Set object being used
   def __init__(self, values=None):
      """This is the constructor.
      It gets called when you create a new Set.
      You would use it like
      """
```

```
        self.dict = {} # each instance of Set has its own dict property
        # which is what we'll use to track memberships
        if values is not None:
            for value in values:
                self.add(value)
    def __repr__(self):
        """this is the string representation of a Set object
        if you type it at the Python prompt or pass it to str()"""
        return "Set: " + str(self.dict.keys())
    # we'll represent membership by being a key in self.dict with value True
    def add(self, value):
        self.dict[value] = True
    # value is in the Set if it's a key in the dictionary
    def contains(self, value):
        return value in self.dict
    def remove(self, value):
        del self.dict[value]
```

Exemples :

```
s = Set([1,2,3])
s.add(4)
print(s.contains(4)) # True
s.remove(3)
print(s.contains(3)) # False
```

2.10 Préparation à l'apprentissage machine

Visualisation

Nous utilisons la librairie matplotib pour différentes visualisations.

```
from matplotlib import pyplot as plt
years = [1950, 1960, 1970, 1980, 1990, 2000, 2010]
gdp = [300.2, 543.3, 1075.9, 2862.5, 5979.6, 10289.7, 14958.3]
# create a line chart, years on x-axis, gdp on y-axis
plt.plot(years, gdp, color='green', marker='o', linestyle='solid')
# add a title
plt.title("Nominal GDP")
# add a label to the y-axis
plt.ylabel("Billions of $")
plt.show()
```

Courbes bars

```
from collections import Counter
grades = [83,95,91,87,70,0,85,82,100,67,73,77,0]
decile = lambda grade: grade // 10 * 10
histogram = Counter(decile(grade) for grade in grades)
plt.bar([x - 4 for x in histogram.keys()], # shift each bar to the left by 4
    histogram.values(), # give each bar its correct height
    8) # give each bar a width of 8
plt.axis([-5, 105, 0, 5]) # x-axis from -5 to 105,
# y-axis from 0 to 5
plt.xticks([10 * i for i in range(11)]) # x-axis labels at 0, 10, ..., 100
```

```
plt.xlabel("Decile")
plt.ylabel("# of Students")
plt.title("Distribution of Exam 1 Grades")
plt.show()
```

Courbes

```
variance = [1, 2, 4, 8, 16, 32, 64, 128, 256]
bias_squared = [256, 128, 64, 32, 16, 8, 4, 2, 1]
total_error = [x + y for x, y in zip(variance, bias_squared)]
xs = [i for i, _ in enumerate(variance)]
# we can make multiple calls to plt.plot
# to show multiple series on the same chart
plt.plot(xs, variance, 'g-', label='variance') # green solid line
plt.plot(xs, bias_squared, 'r-.', label='bias^2') # red dot-dashed line
plt.plot(xs, total_error, 'b:', label='total error') # blue dotted line
# because we've assigned labels to each series
# we can get a legend for free
# loc=9 means "top center"
plt.legend(loc=9)
plt.xlabel("model complexity")
plt.title("The Bias-Variance Tradeo")
plt.show()
```

Scatterplots

```
test_1_grades = [ 99, 90, 85, 97, 80]
test_2_grades = [100, 85, 60, 90, 70]
plt.scatter(test_1_grades, test_2_grades)
plt.title("Axes Aren't Comparable")
plt.xlabel("test 1 grade")
plt.ylabel("test 2 grade")
plt.show()
```

Exploration des données

Exploration des données en 1D Soit la génération d'un vecteur 1D :

```
import random
random.seed(0)
# uniform between -100 and 100
uniform = [200 * random.random() - 100 for _ in range(10000)]
```

Soit les trois fonctions pour uniformiser la visualisation des données :

```
import math
def bucketize(point, bucket_size):
    """floor the point to the next lower multiple of bucket_size"""
    return bucket_size * math.floor(point / bucket_size)
def make_histogram(points, bucket_size):
```

```
    """buckets the points and counts how many in each bucket"""
    return Counter(bucketize(point, bucket_size) for point in points)
def plot_histogram(points, bucket_size, title=""):
    histogram = make_histogram(points, bucket_size)
    plt.bar(histogram.keys(), histogram.values(), width=bucket_size)
    plt.title(title)
    plt.show()
```

On peut maintenant visualiser nos données 1D comme suit :

```
plot_histogram(uniform, 10, "Uniform Histogram")
```

3 Introduction aux méthodes d'apprentissage et à SciKit-Learn

3.1 Principe

L'objectif de la fouille de données (et de l'apprentissage machine (machine learning)) est la construction d'un modèle sur un ensemble de données. Le volet "apprentissage" vient du fait que le modèle contient des paramètres qui doivent être fixés, ou plus précisément "appris". Une fois ces paramètres "appris" sur des données observées en entrée, le modèle peut être utilisé pour appréhender de nouvelles données. Il y a deux grandes catégories d'apprentissage :

- **Apprentissage supervisé :** L'objectif est de dresser un modèle entre les attributs observés en entrée, et un attribut disponible particulier appelé étiquette ou classe. Une fois ce modèle est construit, on peut l'utiliser pour trouver l'étiquette d'une nouvelle donnée. Deux grandes méthodes non supervisées sont développées : (1) méthodes par régression, quand l'étiquette est un attribut continu; (2) méthodes par classification quand l'étiquette est une valeur discrète.
- **Apprentissage non-supervisé :** Ici, on veut construire un modèle sans connaissance à priori d'une quelconque étiquette ou classe sur les données disponibles. Parmi ces méthodes, on peut citer : (1) Le clustering qui vise à partitionner les données en plusieurs groupes; (2) méthodes de réduction des dimensions, qui visent à représenter les données d'une façon plus compacte; (3) méthodes orientées motifs, qui vont repérer des régularités appelées motifs, dans les données.

Pour construire le modèle de fouille en vue dans SciKit-Learn, on procède comme suit :

1) **Choix type de modèle** : Repérer le type de modèle adapté aux données disponibles.
2) **Fixer le modèle** : Fixer les paramètres usuels du modèle.
3) **Formatage des données** : Mettre en forme les données dans le format d'entrée de l'algorithme de fouille.
4) **Génération du modèle** : Appliquer l'algorithme de fouille.
5) **Application du modèle** : Appliquer le modèle sur de nouvelles données.

Nous donnons ci-dessous les différents types de fouille sur des exemples.

3.2 Fouille par régression

Soit un dataset sur deux attributs.

```
import matplotlib.pyplot as plt
import numpy as np
rng = np.random.RandomState(42)
x = 10 * rng.rand(50)
y = 2 * x - 1 + rng.randn(50)
plt.scatter(x, y);
```

Procédons à l'utilisation de la régression sur nos données :

- Choix type de modèle :

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

- Fixer le modèle : Dès lors que nous avons choisi une régression, cette dernière nécessite certaines options comme par exemple : (1) veut on matcher toutes les données ? (2) veut on travailler sur une forme normalisée ? (3) y-t-il nécessité de prétraiter les données ? (4) quel degré de régularisation ? Dans notre exemple, on voudrait que l'option \code{fit_intercept} soit vraie.

```
model = LinearRegression(fit_intercept=True)
```

- Formatage des données : Les données sont souvent dans une forme tabulaire en 2D, et les étiquette/sortie souvent un vecteur. Ici \code{y} est déjà bien formaté. Par contre les données en entrée ne sont pas encore mis en forme tabulaire. On procèdera comme suit en ajoutant une dimension supplémentaire :

```
X = x[:, np.newaxis]
X.shape
```

- Génération du modèle : Nous appliquons maintenant le modèle :

```
model.fit(X, y)
```

- Suite à cette exécution, le modèle est généré avec toutes ses données propres au modèle, comme par exemple les coefficients de la régression.

```
model.coef_
```

- Application du modèle : Dès lors que c'est une méthode supervisée, on pourra prédire avec le modèle généré l'étiquette d'une nouvelle donnée :

```
xfit = np.linspace(-1, 11)
Xfit = xfit[:, np.newaxis]
yfit = model.predict(Xfit)
```

- On peut visualiser le tout comme suit :

```
plt.scatter(x, y)
plt.plot(xfit, yfit);
```

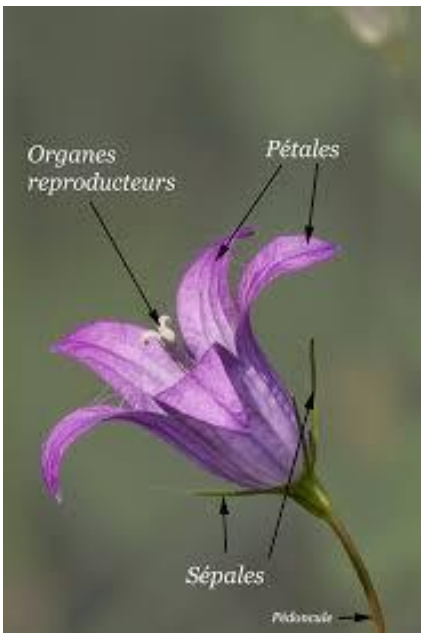
- Précision

```
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error,
mean_squared_error
mean_absolute_error(y,yfit)
```

3.3 Apprentissage supervisée sur les données Iris

Scikit-Learn est une librairie Python qui contient les algorithmes implémentés de fouille. La forme standard de représentation des données en vue d'un traitement par un algorithme de fouille, est la forme tabulaire en deux dimensions. seaborn est une librairie de visualisation, qui contient aussi notamment des exemples de données.
Soit par exemple le dataset Iris :

```
import seaborn as sns
iris = sns.load_dataset('iris')
iris.head()
```



Si on veut éliminer la classe :

```
X_iris = iris.drop('species', axis=1)
X_iris.shape
```

Et si on s'intéresse uniquement à la classe :

```
y_iris = iris['species']
y_iris.shape
```

On se pose ici une question supplémentaire : de combien notre modèle par régression est bon pour prédire la classe (type) ?
Nous utiliserons cette fois-ici une autre méthode celle du classifieur bayesien naïf.
Pour évaluer les performances de notre modèle, nous décomposons nos données en deux sous-ensembles : sous-ensemble d'apprentissage, un sous-ensemble de test.

```
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X_iris, y_iris, random_state=1)
```

Nous appliquons notre protocole d'apprentissage en 5 étapes :

```
from sklearn.naive_bayes import GaussianNB # 1. choose model class
model = GaussianNB()                       # 2. instantiate model
model.fit(Xtrain, ytrain)                  # 3. fit model to data
y_model = model.predict(Xtest)
```

Nous pouvons maintenant faire appel à une méthode qui calcule la précision de notre prédiction :

```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```

3.4 Apprentissage non supervisée : réduction des dimensions

Nous ferons appel à une PCA pour réduire les 4 dimensions en deux comme convenu, via les 5 étapes :

```
from sklearn.decomposition import PCA  # 1. Choose the model class
model = PCA(n_components=2)     # 2. Instantiate the model with hyperparameters
model.fit(X_iris)              # 3. Fit to data. Notice y is not specified!
X_2D = model.transform(X_iris)   # 4. Transform the data to two dimensions
```

On pourra visualiser les résultats :

```
iris['PCA1'] = X_2D[:, 0]
iris['PCA2'] = X_2D[:, 1]
sns.lmplot(x = "PCA1", y = "PCA2", data=iris, fit_reg=False);
```

3.5 Apprentissage non supervisée : clustering

L'objectif du clustering est de partitionner les données dans des groupes distincts. Au lieu de faire appel au classique k-means, nous ferons plutôt appel à une toute autre méthode, celle GMM (Gaussian mixture model). Appliquons les 5 étapes :

```
from sklearn.mixture import GMM      # 1. Choose the model class
model = GMM(n_components=3,
        covariance_type='full')  # 2. Instantiate the model w/ hyperparameters
model.fit(X_iris)                # 3. Fit to data. Notice y is not specified!
y_gmm = model.predict(X_iris)       # 4. Determine cluster labels
```

Visualisons :

```
iris['cluster'] = y_gmm
sns.lmplot(x = "PCA1", y = "PCA2", data=iris, hue='species',
        col='cluster', fit_reg=False);
```

Remarquons que les deux premiers clusters sont nets, alors que le dernier contient un bruit .. à commenter.

3.6 Application à des chiffres écrits à la main

3.6.1 Chargement et visualisation des données

Nous importons les images :

```
from sklearn.datasets import load_digits
digits = load_digits()
digits.images.shape
```

Il y a 1 797 échantillons, où chaque image est de taille 8x8. Visualisons les 100 premières images :

```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(10, 10, figsize=(8, 8),
            subplot_kw={'xticks':[], 'yticks':[]},
            gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i, ax in enumerate(axes.flat):
```

```
  ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')
  ax.text(0.05, 0.05, str(digits.target[i]),
          transform=ax.transAxes, color='green')
```

Chargeons la matrice des données et le vecteur classe :

```
X = digits.data
X.shape
y = digits.target
y.shape
```

3.6.2 Réduction des dimensions

Comme nous avons 64 dimensions, il est insurmontable de visualiser les points. Nous réduirons les dimensions à 2 en faisant appel à la méthode \code{Isomap} (manifold learning).

```
from sklearn.manifold import Isomap
iso = Isomap(n_components=2)
iso.fit(digits.data)
data_projected = iso.transform(digits.data)
data_projected.shape
```

Visualisons :

```
plt.scatter(data_projected[:, 0], data_projected[:, 1], c=digits.target,
  edgecolor='none', alpha=0.5,
  cmap=plt.cm.get_cmap('Spectral', 10))
plt.colorbar(label='digit label', ticks=range(10))
plt.clim(-0.5, 9.5);
```

On voit bien qu'il y a un partitionnement très visible en 10 groupes !

3.6.3 Classification

Faisons appel au classifieur bayésien naïf :

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0)
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(Xtrain, ytrain)
y_model = model.predict(Xtest)
```

Evaluons la précision :

```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```

Pour évaluer pourquoi il y a 20\% d'échec on peut faire appel à la méthode "confusion matrix" :

```
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(ytest, y_model)
sns.heatmap(mat, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value');
```

Cette matrice comptabilise le nombre d'échec

On peut encore mettre en évidence les chiffres mal prédits :

```
fig, axes = plt.subplots(10, 10, figsize=(8, 8),
  subplot_kw={'xticks':[], 'yticks':[]},
   gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i, ax in enumerate(axes.flat):
  ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')
  ax.text(0.05, 0.05, str(y[i]),
            transform=ax.transAxes,
            color='green' if (ytest[i] == y_model[i]) else 'red')
```