Intelligence artificielle, apprentissage automatique et big data

Introduction to Machine learning

Case studies in precision agriculture

https://github.com/ylebbah/teaching

Yahia Lebbah, Noureddine Aribi, Lakhdar Loukil, Said Fourour
Laboratoire LITIO, Faculté FSEA, Equipe de formation AP, Faculté SNV
Université Oran1, Algérie

En collaboration avec
Moulay Omar Imane (M2/Info), Helmaoui Aya (M2/info), Touaibia Hiba (L3/AP)

Table des matières

1 Introduction à l'intelligence artificielle

Intelligence Artificielle ?

Résolution de problèmes en s'inspirant de l'intelligence humaine

Domaines de l'IA[1] :

1. **Représentation des connaissances et Raisonnement Automatique** : Comme son nom le suggère, cette branche de l'IA traite le problème de la représentation des connaissances (qui peuvent être incomplètes, incertaines, ou incohérentes) et de la mise en œuvre du raisonnement.
2. **Résolution de problèmes généraux** : L'objectif est de créer des algorithmes généraux pour résoudre des problèmes concrets.
3. **Traitement du langage naturel** : Ce sous-domaine vise à la compréhension, la traduction, ou la production du langage (écrit ou parlé).
4. **Vision artificielle** : Le but de cette discipline est de permettre aux ordinateurs de comprendre les images et la vidéo (par exemple, de reconnaître des visages ou des chiffres).
5. **Robotique** : Cette discipline vise à réaliser des agents physiques qui peuvent agir dans le monde.
6. **Apprentissage machine** : Dans cette branche de l'IA, on essaie de concevoir des programmes qui peuvent s'auto-modifier en fonction de leur expérience.

[1] https://www.labri.fr/perso/meghyn/papers/cours_IA.pdf

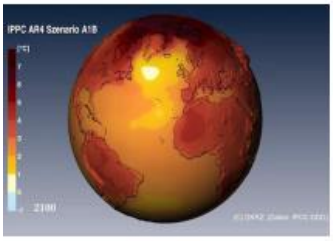
Données = carburant de l'apprentissage automatique [2]

CERN /
Large Hadron Collider
~70 Po/an

Google :
24 PetaOctets/jour

Copernicus :
> 1Po/an

DKRZ (Climat)
500 Po



Square Kilometer Array
1376 Po/an (en 2024)

BIG DATA

kilo/10^3, méga/10^6, giga/10^9, téra/10^{12}, péta/10^{15}, exa/10^{18}, zetta/10^{21} , yotta/10^{24}.

[2] https://perso.ensta-paris.fr/~manzaner/Cours/MI203/cours_ml_intro.pdf

Applications

Anti-Spam (*Classifieur Bayesien*)

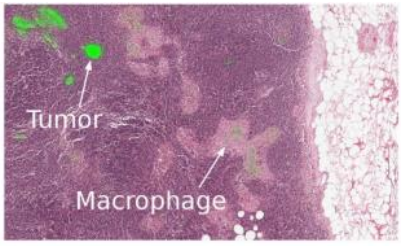
1997 : DeepBlue bat Kasparov
2017: Alpha GO bat Ke Jie
2019: AlphaStar champion de StarCraft

Tri postal automatique (*détection de chiffres manuscrits par réseaux de neurones*)

Recommandation ciblée (*régression logistique*)

Appareil photo avec détection de visages (*boosting*)

Diagnostic médical (*Réseaux de neurones*)



Traduction multi-lingue (Réseaux de neurones)

1.1 Applications en agriculture

[3]

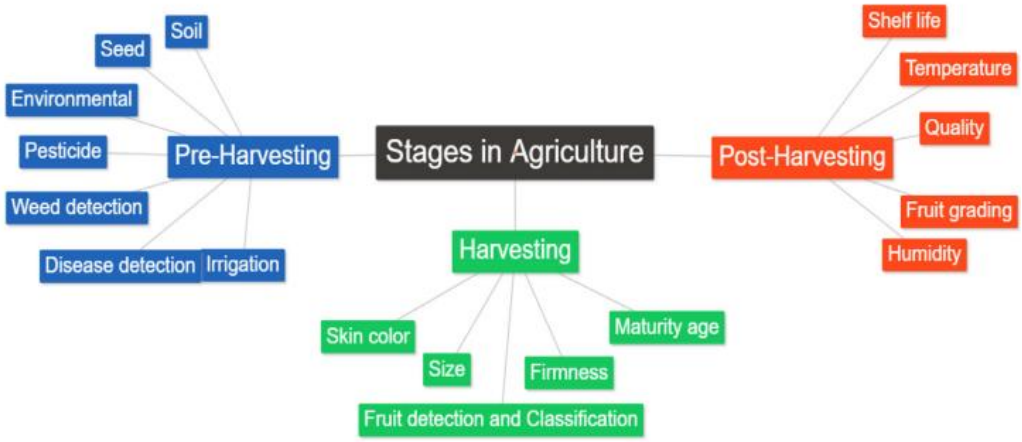
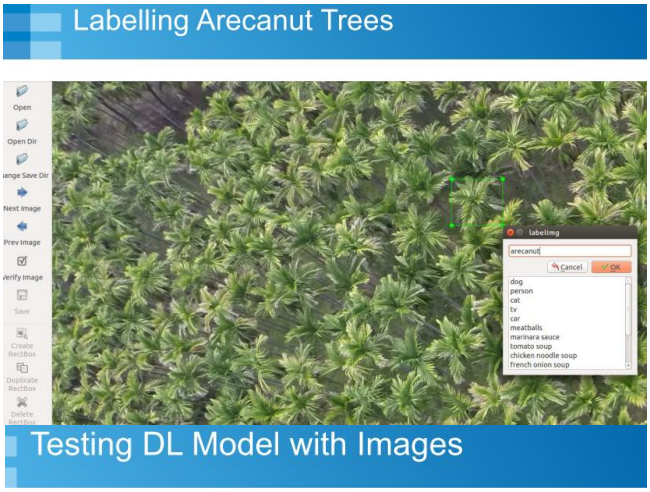


Fig. 2. Important parameters considered in each stage of farming.



[3] https://www.agrotic.org/wp-content/uploads/2018/12/2018_ChaireAgroTIC_DeepLearning_VD2.pdf https://www.inria.fr/sites/default/files/2022-02/livre-blanc-agriculture-numerique-2022_INRIA_BD.pdfhttps://www.itu.int/en/ITU-D/Regional-Presence/AsiaPacific/SiteAssets/Pages/Events/2018/Drones-in-agriculture/asptraining/MachineLearning-Agri.pdf https://www.sciencedirect.com/science/article/pii/S2667318521000106

1.2 Exemples de motivation

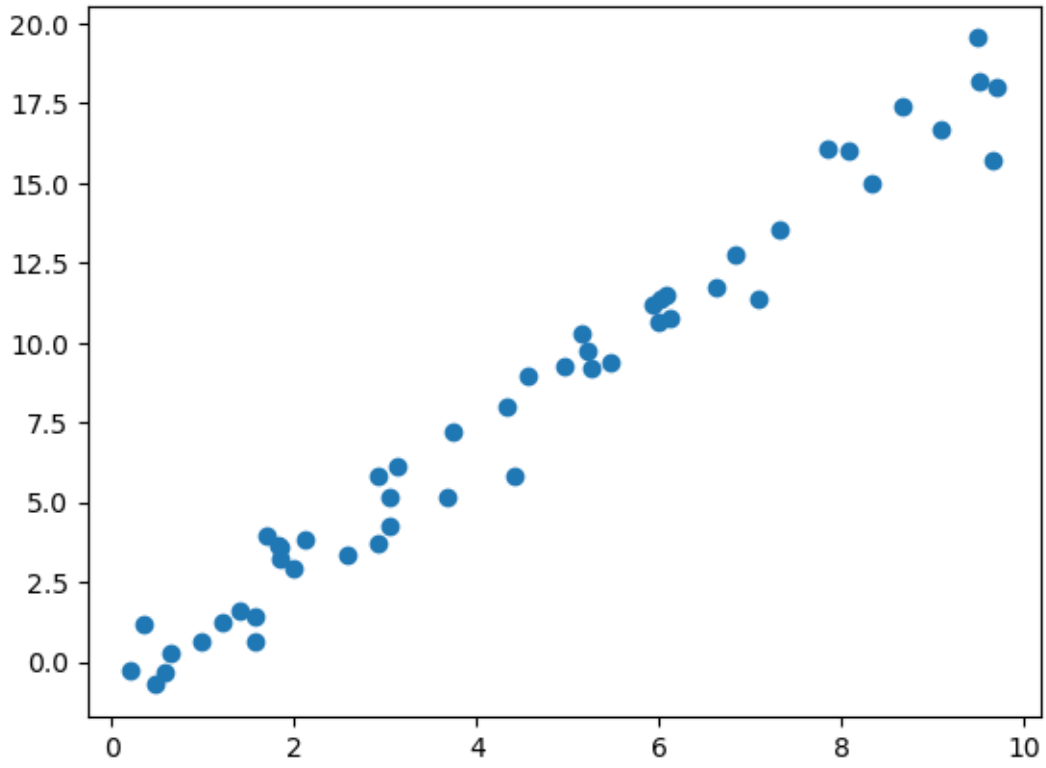
1.2.1 Exemple 1

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

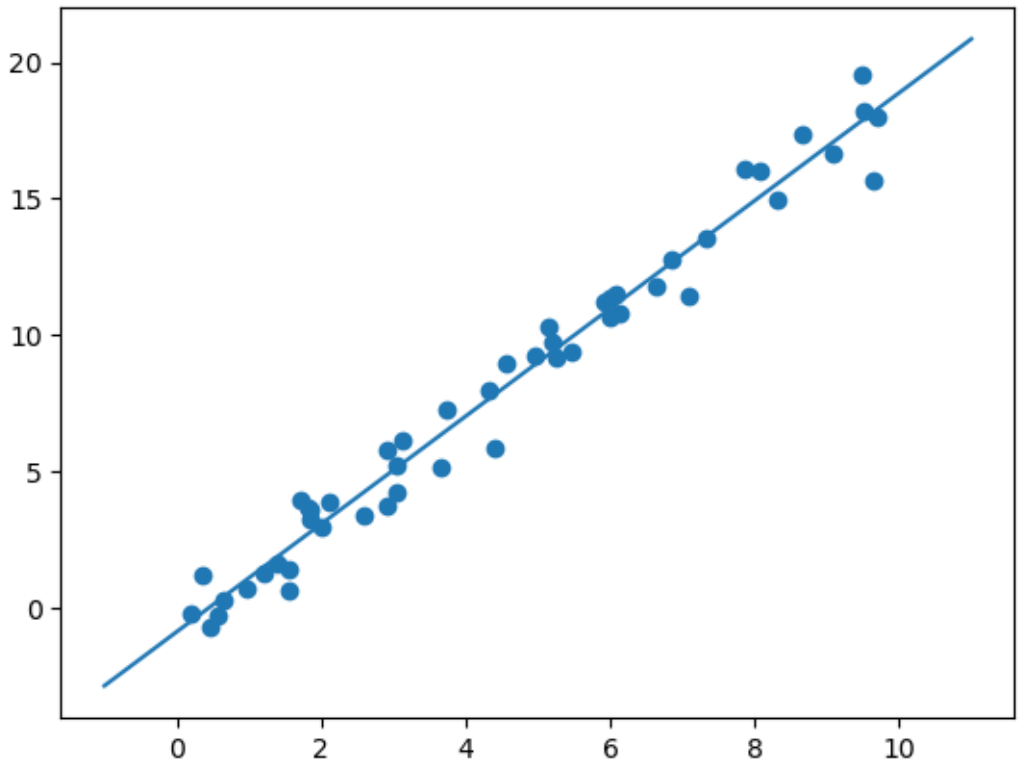
N°	N	R
27	1.9967	2.9215
28	5.1423	10.2882
29	5.9241	11.2099
30	0.4645	_
31	6.0754	11.5123
32	1.7052	3.9485
33	0.6505	0.2652
34	9.4889	19.5424
35	9.6563	15.6929
36	8.0840	15.9898
37	3.0461	5.1793
38	0.9767	0.6544
39	6.8423	12.7764
40	4.4015	5.8155
41	1.2204	1.2211
42	4.9518	9.2607
43	0.3439	1.1657
44	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 !

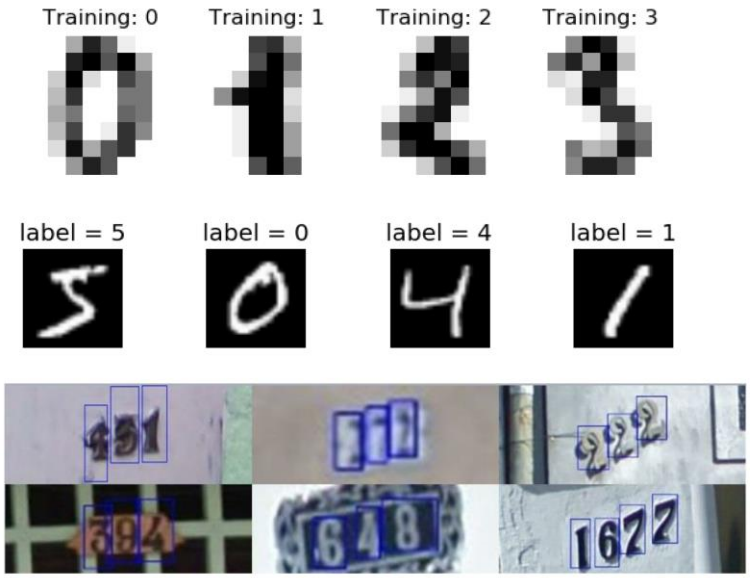


$$\mathbf{R = 1.9776566\ N}$$

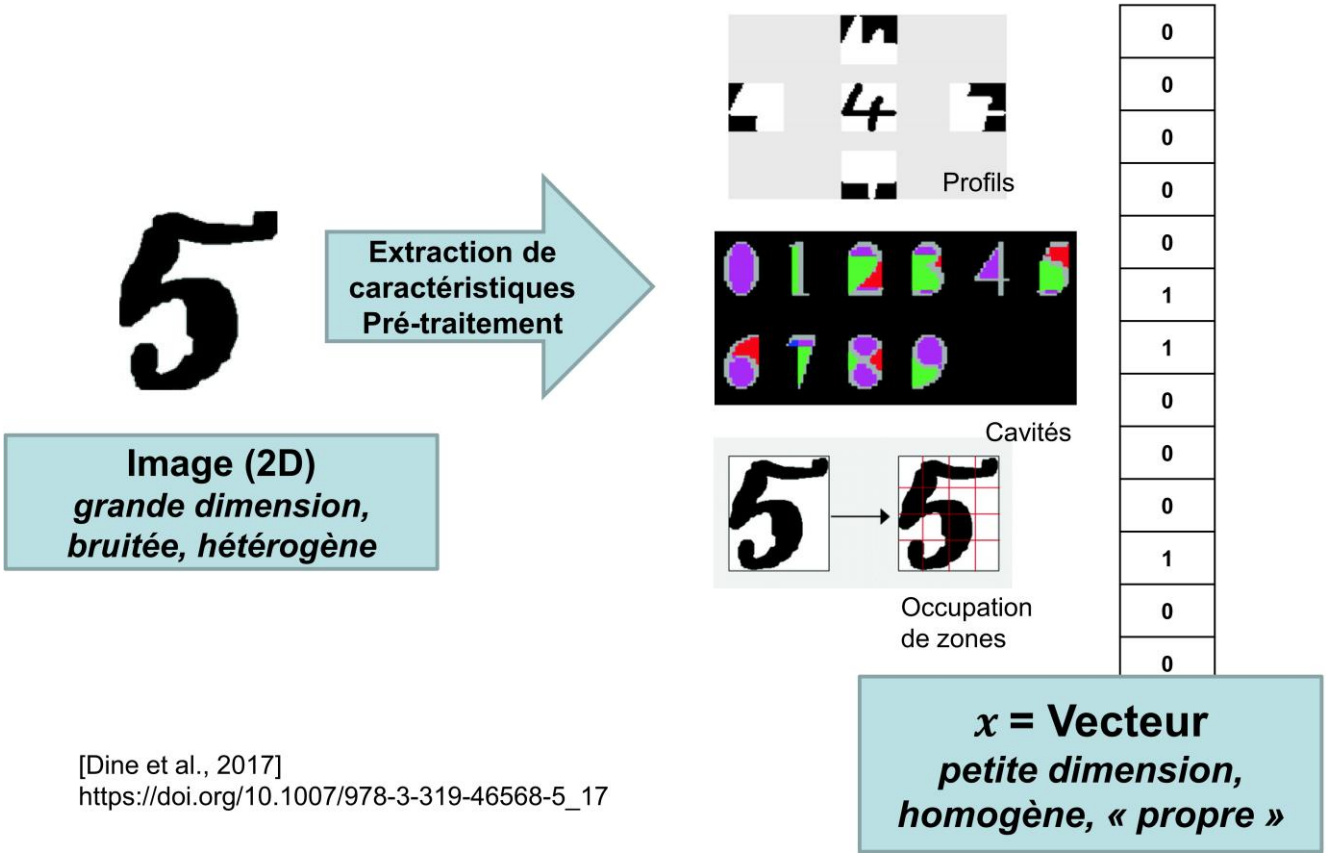
1.2.2 Exemple 2

Etape 1

Etape 2



Etape 3 : formatage des données



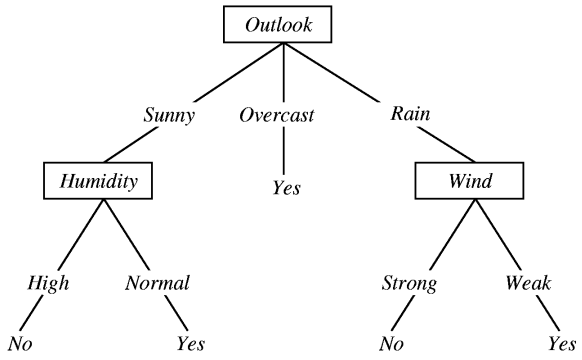
[Dine et al., 2017]
https://doi.org/10.1007/978-3-319-46568-5_17

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

Etape 4 : Choix du modèle

Régression : $\mathbf{y} = \boldsymbol{\alpha_1 x_1} + \boldsymbol{\alpha_2 x_2} + \cdots + \boldsymbol{\alpha_n x_n} + \boldsymbol{\beta}$

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);
```

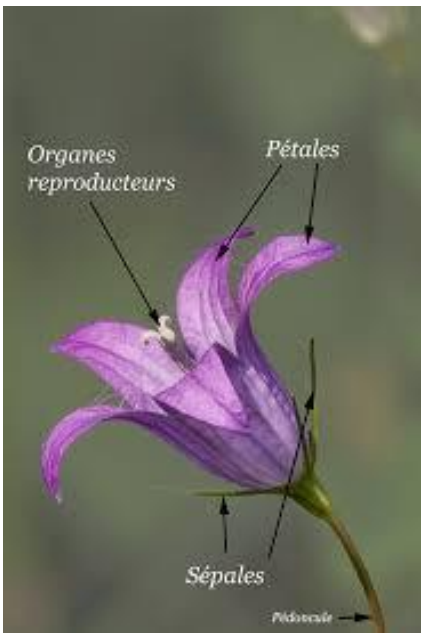
- Précision

```
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')
```

4 Apprentissage supervisé et classification

La classification est la tâche de prédire un attribut de valeur nominale (appelé étiquette de classe) en fonction des valeurs d'autres attributs (appelés variables prédictives). Les objectifs de cet atelier sont les suivants :

1. Fournir des exemples d'utilisation de différentes techniques de classification du package de la bibliothèque **scikit-learn**.
2. Etudier le problème du sur-apprentissage du modèle.

Lisez attentivement les instructions ci-dessous étape par étape. Pour exécuter le code, cliquez sur la cellule correspondante et appuyez simultanément sur les touches SHIFT-ENTER.

4.1 Dataset Vertebrate

Nous utilisons une variante des données sur les vertébrés. Chaque vertébré est classé dans l'une des 5 catégories suivantes : mammifères, reptiles, oiseaux, poissons et amphibiens, sur la base d'un ensemble d'attributs explicatifs (variables prédictives). À l'exception de "nom", le reste des attributs a été converti en une seule représentation binaire. Pour illustrer cela, nous allons d'abord charger les données dans un objet Pandas *DataFrame* et afficher son contenu.

```
import pandas as pd

data = pd.read_csv('vertebrate.csv',header='infer')
data
```

	Name	Warm-blooded	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class
0	human	1	1	0	0	1	0	mammals
1	python	0	0	0	0	0	1	reptiles
2	salmon	0	0	1	0	0	0	fishes
3	whale	1	1	1	0	0	0	mammals
4	frog	0	0	1	0	1	1	amphibians
5	komodo	0	0	0	0	1	0	reptiles
6	bat	1	1	0	1	1	1	mammals
7	pigeon	1	0	0	1	1	0	birds
8	cat	1	1	0	0	1	0	mammals
9	leopard shark	0	1	1	0	0	0	fishes
10	turtle	0	0	1	0	1	0	reptiles
11	penguin	1	0	1	0	1	0	birds
12	porcupine	1	1	0	0	1	1	mammals
13	eel	0	0	1	0	0	0	fishes
14	salamander	0	0	1	0	1	1	amphibians

Étant donné le nombre limité d'exemples d'apprentissage, nous convertissions le problème en une tâche de classification binaire (mammifères vs. non-mammifères). Nous pouvons le faire en remplaçant les étiquettes de classe des instances par des *non-mammifères*, à l'exception de celles qui appartiennent à la classe des mammifères.

```
data['Class'] = data['Class'].replace(['fishes','birds','amphibians','reptiles'],'non-mammals')
data
```

	Name	Warm-blooded	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class
0	human	1	1	0	0	1	0	mammals
1	python	0	0	0	0	0	1	non-mammals
2	salmon	0	0	1	0	0	0	non-mammals
3	whale	1	1	1	0	0	0	mammals
4	frog	0	0	1	0	1	1	non-mammals
5	komodo	0	0	0	0	1	0	non-mammals
6	bat	1	1	0	1	1	1	mammals
7	pigeon	1	0	0	1	1	0	non-mammals
8	cat	1	1	0	0	1	0	mammals
9	leopard shark	0	1	1	0	0	0	non-mammals
10	turtle	0	0	1	0	1	0	non-mammals
11	penguin	1	0	1	0	1	0	non-mammals
12	porcupine	1	1	0	0	1	1	mammals
13	eel	0	0	1	0	0	0	non-mammals
14	salamander	0	0	1	0	1	1	non-mammals

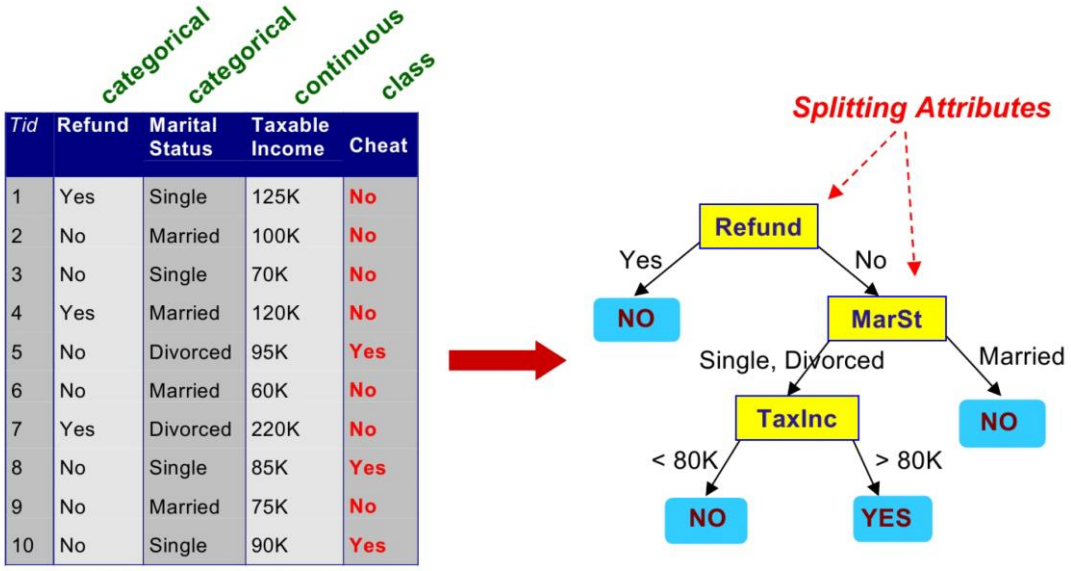
Nous pouvons appliquer la tabulation croisée Pandas pour examiner la relation entre les attributs ***Warm-blooded*** et ***Gives Birth*** par rapport à la classe.

```
pd.crosstab([data['Warm-blooded'],data['Gives Birth']],data['Class'])
```

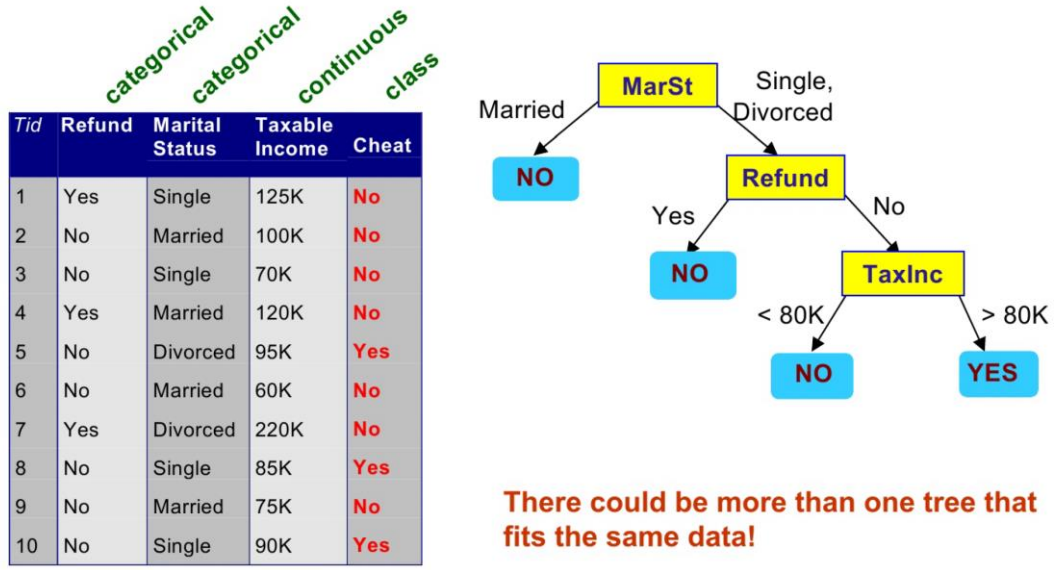
	Class	mammals	non-mammals
Warm-blooded	**Gives Birth**		
0	**0**	0	7
	1	0	1
1	**0**	0	2
	1	5	0

4.2 Principe des arbres de décision

Soit des données d'un établissement financier sur le comportement des clients. Avec les arbres de décision, on peut produire des arbres, notamment les deux ci-dessous (Cheat : triche, Refund : rembourse).



Training Data **Model: Decision Tree**



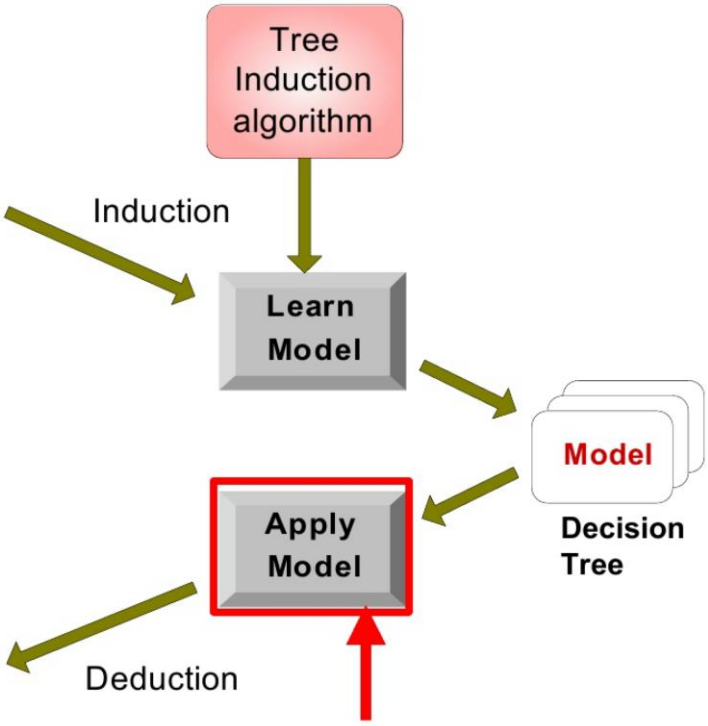
En somme, l'exploitation des arbres de décision peut être résumé comme suit :

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



L'algorithme des arbres de décision consiste en (https://philippe-preux.github.io/Documents/notes-de-cours-de-fouille-de-donnees.pdf) :

Algorithme 1 ID3

Nécessite: 2 paramètres : l'ensemble d'exemples $\mathcal{X}$, l'ensemble d'attributs $\mathcal{A} = \{a_{j\in\{1,\dots p\}}\}$ où p est le nombre d'attributs restants à considérer

Créer un nœud racine

si tous les élements de $\mathcal{X}$ sont positifs **alors**

racine. étiquette $\leftarrow \oplus$

return racine

fin si

si tous les éléments de $\mathcal{X}$ sont négatifs **alors**

racine. étiquette $\leftarrow \ominus$

return racine

fin si

si $\mathcal{A} = \emptyset$ **alors**

racine. étiquette $\leftarrow$ valeur la plus présente de la classe parmi les $\mathcal{X}$

return racine

fin si

$a^* \leftarrow \arg\max_{a\in\mathcal{A}} \text{gain}(\mathcal{X}, a)$

racine. étiquette $\leftarrow a^*$

pour toutes les valeurs v_i de a^* **faire**

ajouter une branche à racine correspondant à la valeur v_i

former $\mathcal{X}_{a^*=vi} \subset \mathcal{X}$ dont l'attribut a^* vaut v_i

si $\mathcal{X}_{a^*=vi} = \emptyset$ **alors**

à l'extrémité de cette branche, mettre une feuille étiquetée avec la valeur la plus présente de la classe parmi les $\mathcal{X}$

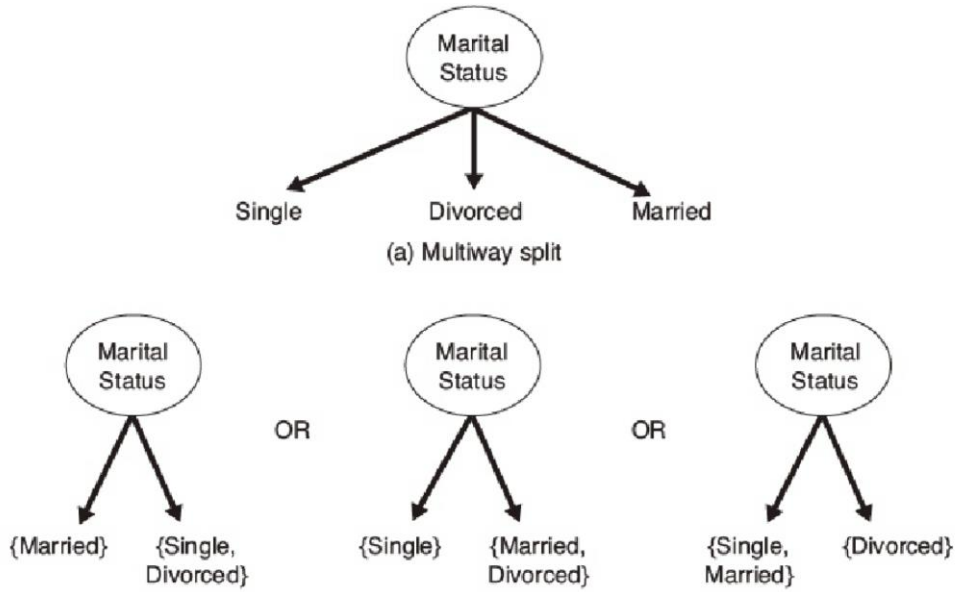
sinon

à l'extrémité de cette branche, mettre ID3 ($\mathcal{X}_{a^*=vi}$, $\mathcal{A} - \{a^*\}$)

fin si

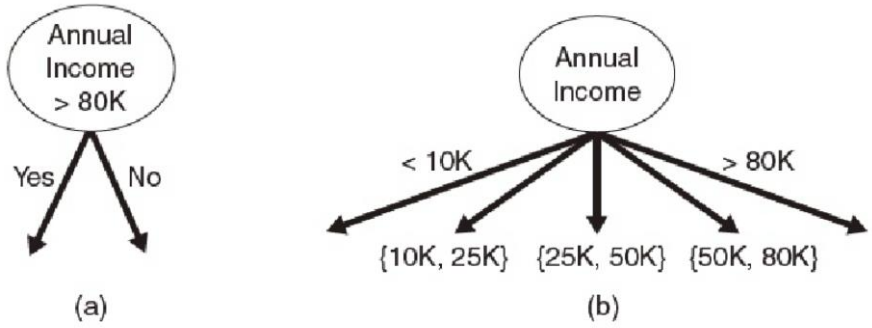
fin pour

return racine



Test conditions for nominal attributes.

Partitionnement d'un noeud de type domaine fini [Tan et al., 2005]



Test condition for continuous attributes.

Partitionnement d'un noeud de type continu [Tan et al., 2005]

4.2.1 Problème du choix de l'attribut

On a besoin de pouvoir évaluer l'homogénéité ou de la pureté (ou impureté) d'un ensemble de données. En physique, on mesure l'homogénéité par l'entropie.

Nous pouvons mesurer l'impureté des données avec l'entropie en maximisant :

$$Gain\ (\mathcal{X}, a_j) = H(\mathcal{X}) - \sum_{v \in\ valeurs(a_j)} \frac{|\mathcal{X}_{a_j=v}|}{|\mathcal{X}|}\ H(\mathcal{X}_{a_j=v})$$

où l'entropie est donnée par :

$$H(\mathcal{X}) = -\sum_{i=1}^{i=n} p_i \log_2 p_i$$

avec

- a_j un des attributs de l'ensemble de données.

- $\mathcal{X}_{a_j=v}$ sont les données qui ont l'attribut *v*.
- *Entropy* mesure l'impureté ou l'hétérogénéité des données S par rapport à l'attribut *y*. Plus la donnée est pure, plus sa valeur est petite.

D'autres mesures [4] sont proposées dans la littérature.

4.2.2 Exemple de déroulement

Référence : https://philippe-preux.github.io/Documents/notes-de-cours-de-fouille-de-donnees.pdf
Soit l'ensemble de données :

Jour	Ciel	Température	Humidité	Vent	Jouer au tennis ?
1	Ensoleillé	Chaude	Élevée	Faible	Non
2	Ensoleillé	Chaude	Élevée	Fort	Non
3	Couvert	Chaude	Élevée	Faible	Oui
4	Pluie	Tiède	Élevée	Faible	Oui
5	Pluie	Fraîche	Normale	Faible	Oui
6	Pluie	Fraîche	Normale	Fort	Non
7	Couvert	Fraîche	Normale	Fort	Oui
8	Ensoleillé	Tiède	Élevée	Faible	Non
9	Ensoleillé	Fraîche	Normale	Faible	Oui
10	Pluie	Tiède	Normale	Faible	Oui
11	Ensoleillé	Tiède	Normale	Fort	Oui
12	Couvert	Tiède	Élevée	Fort	Oui
13	Couvert	Chaud	Normale	Faible	Oui
14	Pluie	Tiède	Élevée	Fort	Non

[4] Parmi les mesures d'impureté utilisées, on peut citer l'indice de GINI :

$$Gini(y,S) = 1 - \sum_{c_i \in dom(y)} \left(\frac{|\sigma_{y=c_i} S|}{|S|} \right)^2$$

En utilisant cette mesure, la sélection de l'attribut se fait en prenant en maximisant la mesure suivante :

$$GiniGain(a_i,S) = Gini(y,S) - \sum_{v_{i,j} \in dom(a_i)} \frac{|\sigma_{a_i=v_{i,j}} S|}{|S|} \cdot Gini(y, \sigma_{a_i=v_{i,j}} S)$$

Soit le calcul de Gain de l'attribut « Vent » en se basant sur l'entropie (*H*) :

Exemple : supposons que $\mathcal{X}$ soit constitué de 14 exemples, 9 positifs et 5 négatifs. Parmi ces exemples, 6 positifs et 2 négatifs prennent la valeur « oui » pour l'attribut a, tandis que les autres exemples prennent la valeur « non » pour cet attribut.

$$\text{Gain } (\mathcal{X}, a) = H(\mathcal{X}) - \sum_{v \in \{\text{oui, non}\}} \frac{|\mathcal{X}_{a=v}|}{|\mathcal{X}|} H(\mathcal{X}_{a=v})$$

$$\text{Gain } (\mathcal{X}, a) = H(\mathcal{X}) - \frac{8}{14} H(\mathcal{X}_{a=\text{oui}}) - \frac{6}{14} H(\mathcal{X}_{a=\text{non}})$$

On a :

$$H(\mathcal{X}_{\text{oui}}) = -(\frac{6}{8} \ln_2 \frac{6}{8} + \frac{2}{8} \ln_2 \frac{2}{8}) \approx 0.811$$

et

$$H(\mathcal{X}_{\text{non}}) = -(\frac{3}{6} \ln_2 \frac{3}{6} + \frac{3}{6} \ln_2 \frac{3}{6}) = 1.0$$

D'où :

$$\text{Gain } (\mathcal{X}, a) \approx 0.940 - \frac{8}{14} 0.811 - \frac{6}{14} 1.00$$

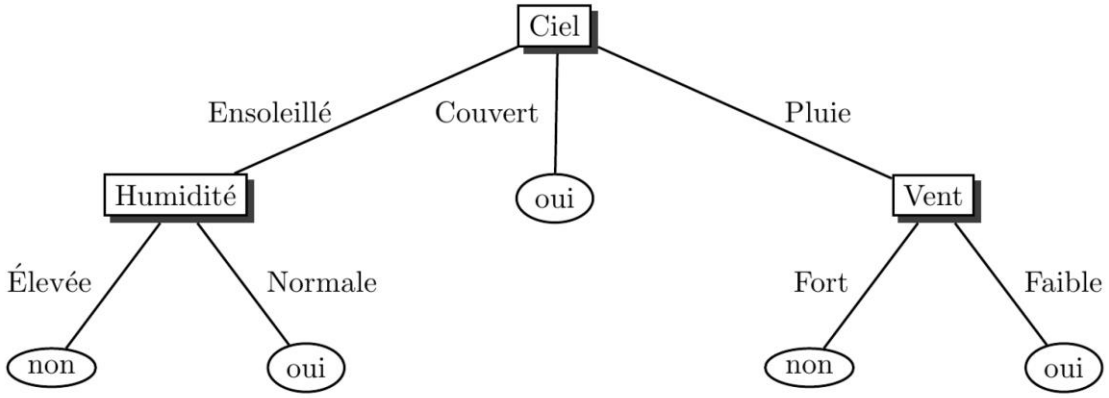
$$\text{Gain } (\mathcal{X}, a) \approx 0.048$$

Le gain d'information en classant le jeu de d'exemples $\mathcal{X}$ par rapport à a est donc : 0.048.

En somme on a les gains des attributs :

Attribut	Gain
Ciel	$0,246$
Humidité	$0,151$
Vent	$0,048$
Température	$0,029$

d'où l'attribut le plus porteur de gain « Ciel », donnant lieu à l'arbre de décision :



4.3 Illustration de la classification avec les Arbres de Décision

Dans cette section, nous appliquons un classifieur d'arbre de décision à l'ensemble des données sur les vertébrés décrit dans la section précédente.

```
from sklearn import tree

Y = data['Class']
X = data.drop(['Name','Class'],axis=1)

clf = tree.DecisionTreeClassifier(criterion='entropy',max_depth=3)
clf = clf.fit(X, Y)
```

Les commandes précédentes extrairont les attributs de prédiction (X) et de classe cible (Y) de l'ensemble de données de vertébrés et créeront un objet classifieur de type arbre de décision en utilisant l'entropie comme mesure d'**impureté** pour le critère de division. La classe d'arbre de décision dans la bibliothèque Python **sklearn** prend également en charge l'utilisation de « **gini** » comme mesure d'impureté. Le classifieur ci-dessus est également contraint de générer des arbres avec une *profondeur* maximale égale à 3. Ensuite, le classifieur est entraîné sur les données étiquetées à l'aide de la fonction *fit()*.

Nous pouvons tracer l'arbre de décision résultant obtenu après l'apprentissage du classifieur. Pour ce faire, vous devez d'abord installer à la fois **graphviz** (http://www.graphviz.org) et son interface Python appelée **pydotplus** (http://pydotplus.readthedocs.io/).

Télécharger et installer **graphviz** (Ajouter au PATH), puis lancer Anaconda Prompt et taper[5] :

```
pip install Graphviz
pip install pydotplus
```

Note : Si vous rencontrez ce message d'erreur :

"twisted 18.7.0 requires PyHamcrest>=1.9.0, which is not installed."

alors il faut télécharger

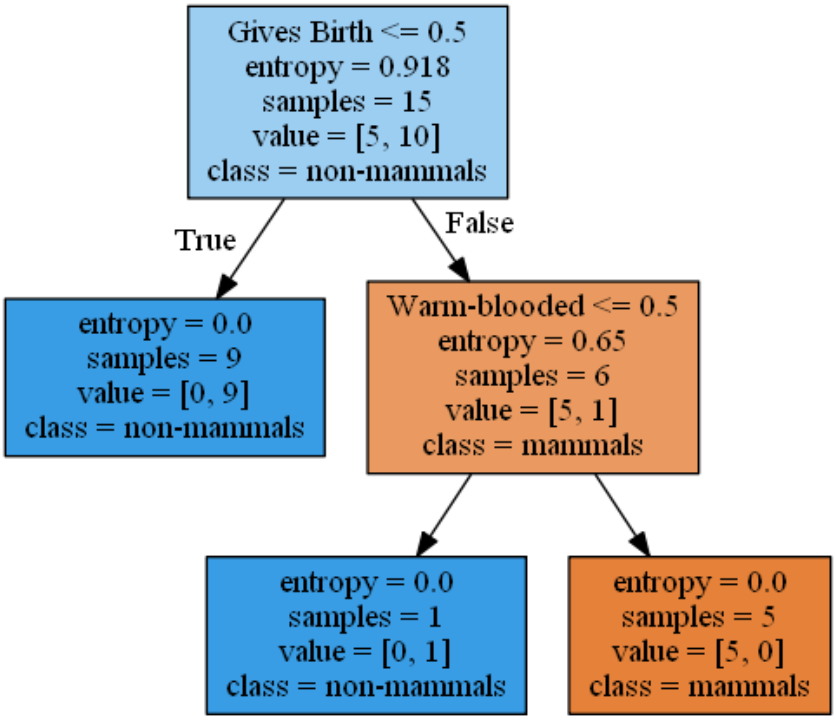
https://download.lfd.uci.edu/pythonlibs/r4tycu3t/PyHamcrest-2.0.2-py3-none-any.whl

Puis taper (dans Anaconda Prompt) :

```
pip install PyHamcrest-2.0.2-py3-none-any.whl
```

```
import pydotplus
from IPython.display import Image
dot_data = tree.export_graphviz(clf, feature_names=X.columns,
class_names=['mammals','non-mammals'], filled=True,
                        out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

[5] Si la version du PIP est ancienne, vous devriez envisager de mettre à niveau via la commande (à partir de Anaconda Prompt) : python -m pip install --upgrade pip



Ensuite, supposons que nous appliquions l'arbre de décision pour classer les exemples de test suivants.

```
testData = [['gila monster',0,0,0,0,1,1,'non-mammals'],
        ['platypus',1,0,0,0,1,1,'mammals'],
        ['owl',1,0,0,1,1,0,'non-mammals'],
        ['dolphin',1,1,1,0,0,0,'mammals']]
testData = pd.DataFrame(testData, columns=data.columns)
testData
```

	Name	Warm-blooded	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class
0	gila monster	0	0	0	0	1	1	non-mammals
1	platypus	1	0	0	0	1	1	mammals
2	owl	1	0	0	1	1	0	non-mammals
3	dolphin	1	1	1	0	0	0	mammals

Nous extrayons d'abord les attributs du prédicteur et de classe cible des données de test, puis appliquons le classifieur d'arbre de décision pour prédire leurs classes.

```
testY = testData['Class']
testX = testData.drop(['Name','Class'],axis=1)

predY = clf.predict(testX)
predictions = pd.concat([testData['Name'],pd.Series(predY,name='Predicted Class')], axis=1)
predictions
```

	Name	Predicted Class
0	gila monster	non-mammals
1	platypus	non-mammals
2	owl	non-mammals
3	dolphin	mammals

À l'exception de platypus, qui est un mammifère pondeur, le classifieur prédit correctement l'étiquette de classe des exemples de test. Nous pouvons calculer la précision du classifieur sur les données de test comme le montre l'exemple ci-dessous.

```
from sklearn.metrics import accuracy_score
print('Accuracy on test data is %.2f' % (accuracy_score(testY, predY)))
```

```
Accuracy on test data is 0.75
```

4.4 Surapprentissage du modèle (Model Overfitting)

Pour illustrer le problème du surapprentissage du modèle, nous considérons un ensemble de données bidimensionnel contenant 1500 instances étiquetées, chacune étant affectée à l'une des deux classes, 0 ou 1. Les instances de chaque classe sont générées comme suit :

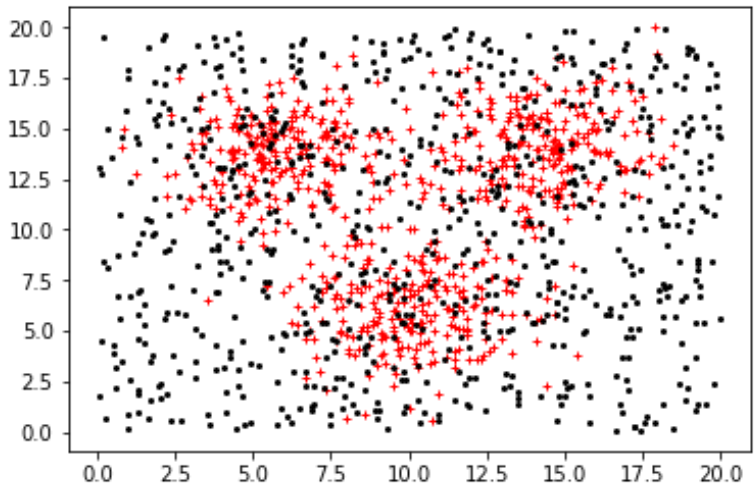
1. Les instances de la classe 1 sont générées à partir d'un mélange de 3 distributions gaussiennes, centrées en [6,14], [10,6] et [14 14], respectivement.
2. Les instances de la classe 0 sont générées à partir d'une distribution uniforme dans une région carrée, dont les côtés ont une longueur égale à 20.

Pour plus de simplicité, les deux classes ont un nombre égal d'instances étiquetées. Le code pour générer et tracer les données est indiqué ci-dessous. Toutes les instances de la classe 1 sont affichées en rouge tandis que celles de la classe 0 sont affichées en noir.

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.random import random

%matplotlib inline

N = 1500

mean1 = [6, 14]
mean2 = [10, 6]
mean3 = [14, 14]
cov = [[3.5, 0], [0, 3.5]]  # diagonal covariance
np.random.seed(50)
X = np.random.multivariate_normal(mean1, cov, int(N/6))
X = np.concatenate((X, np.random.multivariate_normal(mean2, cov, int(N/6))))
X = np.concatenate((X, np.random.multivariate_normal(mean3, cov, int(N/6))))
X = np.concatenate((X, 20*np.random.rand(int(N/2),2)))
Y = np.concatenate((np.ones(int(N/2)),np.zeros(int(N/2))))

plt.plot(X[:int(N/2),0],X[:int(N/2),1],'r+',X[int(N/2):,0],X[int(N/2):,1],'k.',ms=4)
```

```
[<matplotlib.lines.Line2D at 0x1893e207f60>,
```

```
<matplotlib.lines.Line2D at 0x1893e2161d0>]
```

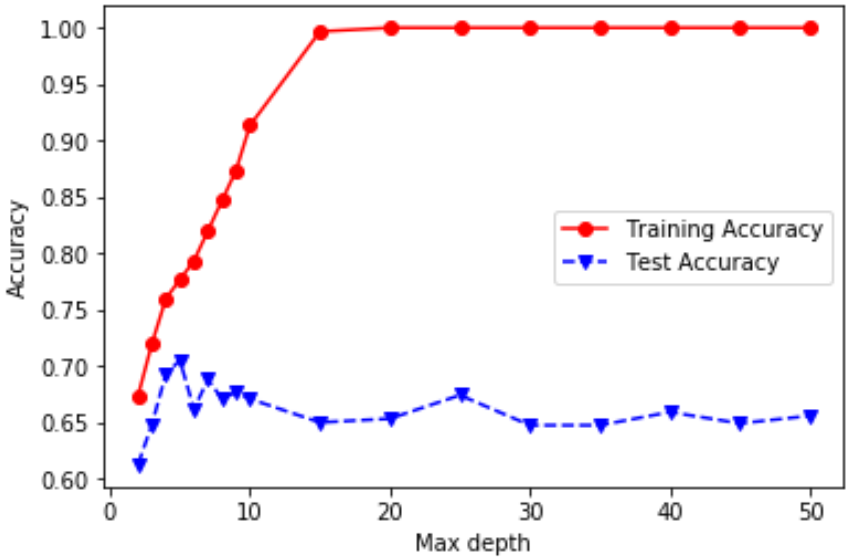


Dans cet exemple, nous réservons 80 % des données étiquetées pour l'apprentissage et les 20 % restantes pour les tests. Nous adaptons ensuite des arbres de décision de différentes profondeurs maximales (de 2 à 50) à l'ensemble d'apprentissage et traçons leurs précisions respectives lorsqu'elles sont appliquées aux ensembles d'apprentissage et de test.

```
##########################################
# Training and Test set creation
##########################################

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.8, random_state=1)

from sklearn import tree
from sklearn.metrics import accuracy_score

##########################################
# Model fitting and evaluation
##########################################

maxdepths = [2,3,4,5,6,7,8,9,10,15,20,25,30,35,40,45,50]

trainAcc = np.zeros(len(maxdepths))
testAcc = np.zeros(len(maxdepths))

index = 0
for depth in maxdepths:
    clf = tree.DecisionTreeClassifier(max_depth=depth)
    clf = clf.fit(X_train, Y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    trainAcc[index] = accuracy_score(Y_train, Y_predTrain)
    testAcc[index] = accuracy_score(Y_test, Y_predTest)
    index += 1
```

```
#########################################
# Plot of training and test accuracies
#########################################

plt.plot(maxdepths,trainAcc,'ro-',maxdepths,testAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Max depth')
plt.ylabel('Accuracy')
```

```
Text(0,0.5,'Accuracy')
```



Le graphique ci-dessus montre que la précision de l'apprentissage continuera de s'améliorer à mesure que la profondeur maximale de l'arbre augmente (c'est-à-dire que le modèle devient plus complexe). Cependant, la précision du test s'améliore initialement jusqu'à une profondeur maximale de 5, avant de diminuer progressivement en raison du surapprentissage du modèle.

4.5 Techniques de classification alternatives

Outre la classification par arbre de décision, la bibliothèque Python ***sklearn*** prend également en charge d'autres techniques de classification. Dans cette section, nous fournissons des exemples pour illustrer comment appliquer le classifieur du (k) plus proche voisin (**k-nearest neighbor**), les classifieurs linéaires (régression logistique et séparateurs à vaste marge (SVM)), ainsi que les méthodes d'ensemble (**boosting**, **bagging** et **random forest**) aux données bidimensionnelles donné dans la section précédente.

4.6 Classifieur K-Nearest neighbor

Dans cette approche, l'étiquette de classe d'une instance de test est prédite sur la base de la classe majoritaire de ses k instances d'apprentissage les plus proches.

4.6.1 Principe de K-Nearest neighbor

Référence : https://eduscol.education.fr/document/30061/download

L'algorithme de k plus proches voisins ne nécessite pas de phase d'apprentissage à proprement parler, il faut juste stocker le jeu de données d'apprentissage.
Soit un ensemble E contenant n données labellisées : $E = \{(y_i, \vec{x}_i)\}$ avec i compris entre 1 et n, où y_i correspond à la classe (le label) de la donnée i et où le vecteur $\vec{x}_i$ de dimension p ($\vec{x}_i = (x_{1i}, x_{2i}, ..., x_{pi})$) représente les variables prédictrices de la donnée i. Soit une donnée u qui n'appartient pas à E et qui ne possède pas de label (u est uniquement caractérisée par un vecteur $\vec{x}_u$ de dimension p). Soit d une fonction qui renvoie la distance entre la donnée u et une donnée quelconque appartenant à E. Soit un entier k inférieur ou égal à n. Voici le principe de l'algorithme de k plus proches voisins :

- ▷ On calcule les distances entre la donnée u et chaque donnée appartenant à E à l'aide de la fonction d
- ▷ On retient les k données du jeu de données E les plus proches de u
- ▷ On attribue à u la classe qui est la plus fréquente parmi les k données les plus proches.

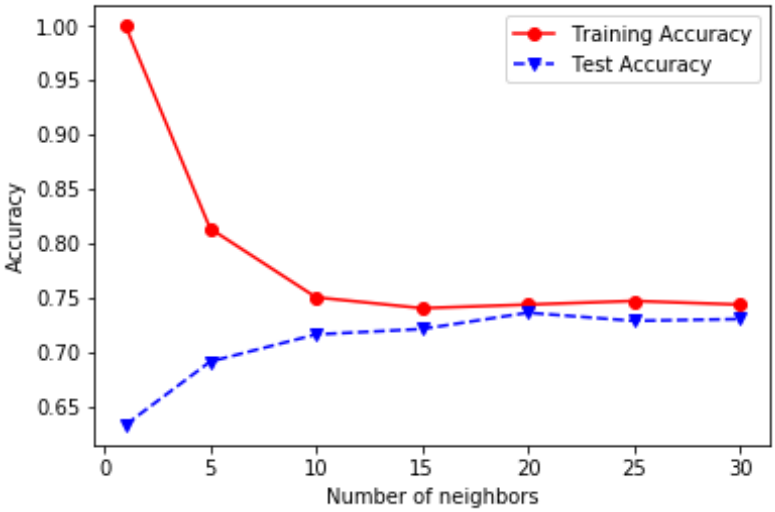
Par défaut, nous pouvons utiliser la distance euclidienne (qui équivaut à la distance de Minkowski avec un facteur exposant égal à p=2) :

$$\text{Minkowski distance}(x, y) = \left[\sum_{i=1}^{N} |x_i - y_i|^p\right]^{\frac{1}{p}}$$

4.6.2 Mise en pratique

```
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
%matplotlib inline

numNeighbors = [1, 5, 10, 15, 20, 25, 30]
trainAcc = []
testAcc = []

for k in numNeighbors:
    clf = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2)
    clf.fit(X_train, Y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    trainAcc.append(accuracy_score(Y_train, Y_predTrain))
    testAcc.append(accuracy_score(Y_test, Y_predTest))

plt.plot(numNeighbors, trainAcc, 'ro-', numNeighbors, testAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
```

```
Text(0,0.5,'Accuracy')
```



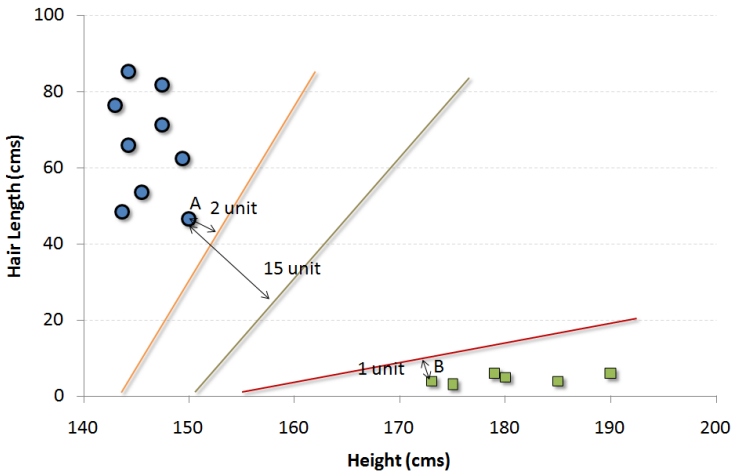
Notez que les classifieurs linéaires fonctionnent mal sur les données car les véritables limites de décision entre les classes sont non linéaires pour l'ensemble de données bidimensionnel donné.

4.7 Les séparateurs à vastes marges non linéaires

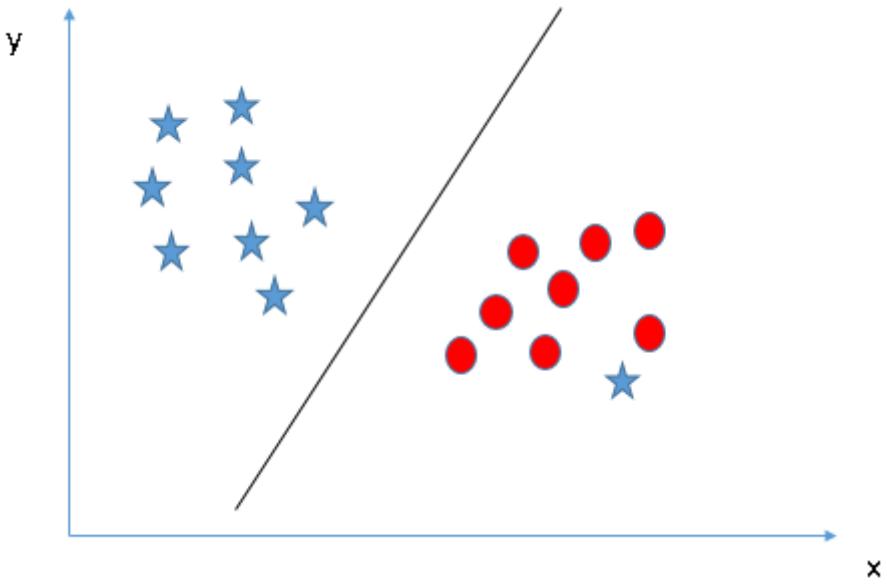
Le code ci-dessous montre un exemple d'utilisation d'un séparateur à vastes marges non linéaire avec un noyau de fonction de base radiale gaussienne pour s'adapter à l'ensemble de données bidimensionnel.

4.7.1 Principe

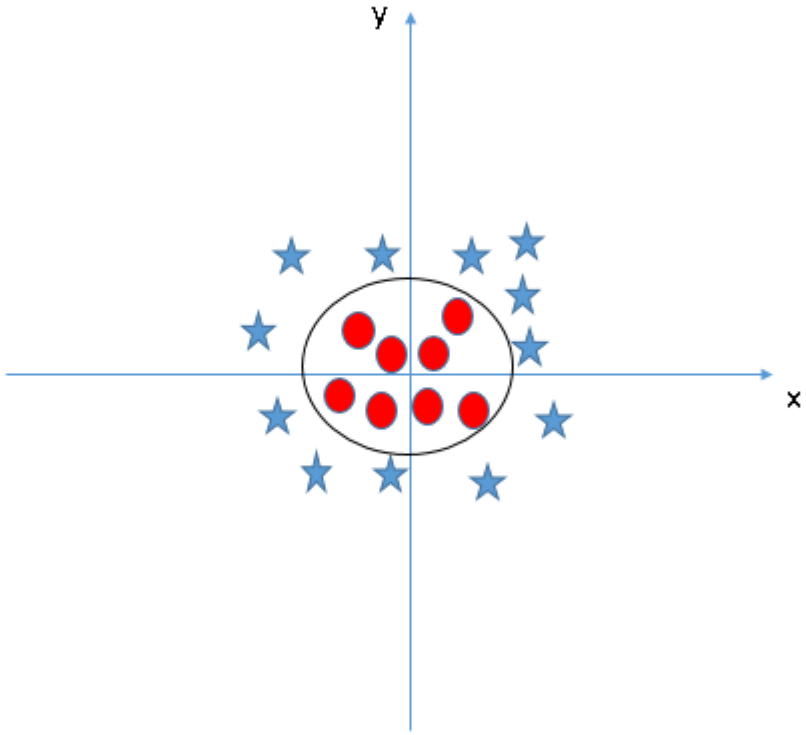
Cas 1 :



Cas 2 :



Cas 3 :



4.7.2 Mise en pratique

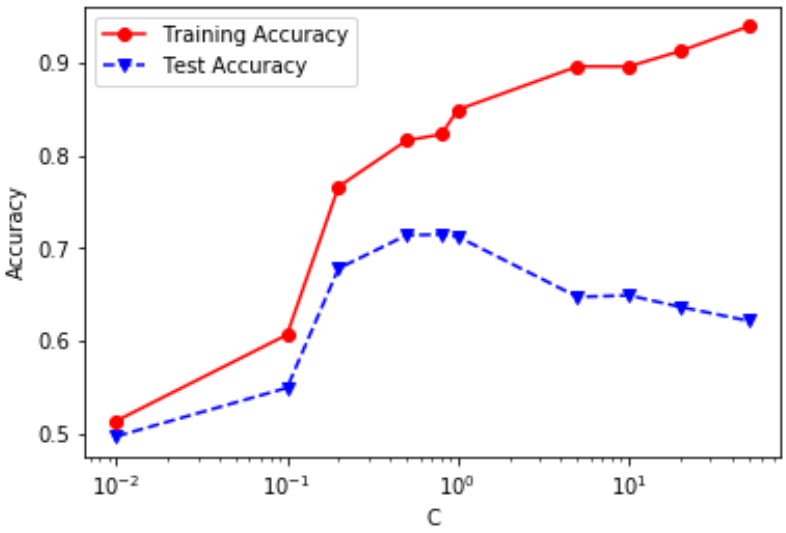
Soit C la paramètre de régularisation[6].

```
from sklearn.svm import SVC

C = [0.01, 0.1, 0.2, 0.5, 0.8, 1, 5, 10, 20, 50]
SVMtrainAcc = []
SVMtestAcc = []

for param in C:
   clf = SVC(C=param,kernel='rbf',gamma='auto')
   clf.fit(X_train, Y_train)
   Y_predTrain = clf.predict(X_train)
   Y_predTest = clf.predict(X_test)
   SVMtrainAcc.append(accuracy_score(Y_train, Y_predTrain))
   SVMtestAcc.append(accuracy_score(Y_test, Y_predTest))

plt.plot(C, SVMtrainAcc, 'ro-', C, SVMtestAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('C')
```

[6] Régularisation : Cette technique clé de machine learning vise à limiter le « surapprentissage » (overfitting) et à contrôler l'erreur de type variance pour aboutir à de meilleures performances. Lors de l'apprentissage d'un modèle, la régularisation permet d'imposer une contrainte pour favoriser les modèles simples au détriment des modèles complexes. Autrement dit, cela permet de réduire l'erreur de type variance et d'améliorer la généralisation de la solution. Il existe de nombreuses formes de régularisation, qui dépendent de l'objectif recherché et des hypothèses fixées sur le problème.

```
plt.xscale('log')
plt.ylabel('Accuracy')
```

```
Text(0,0.5,'Accuracy')
```



Observez que le SVM non linéaire peut atteindre une précision de test plus élevée par rapport au SVM linéaire.

4.8 Classifieurs linéaires : régression logistique

Les classifieurs linéaires tels que la régression logistique et le séparateur à vaste marge (SVM) construisent un hyperplan de séparation linéaire pour distinguer les instances de différentes classes.

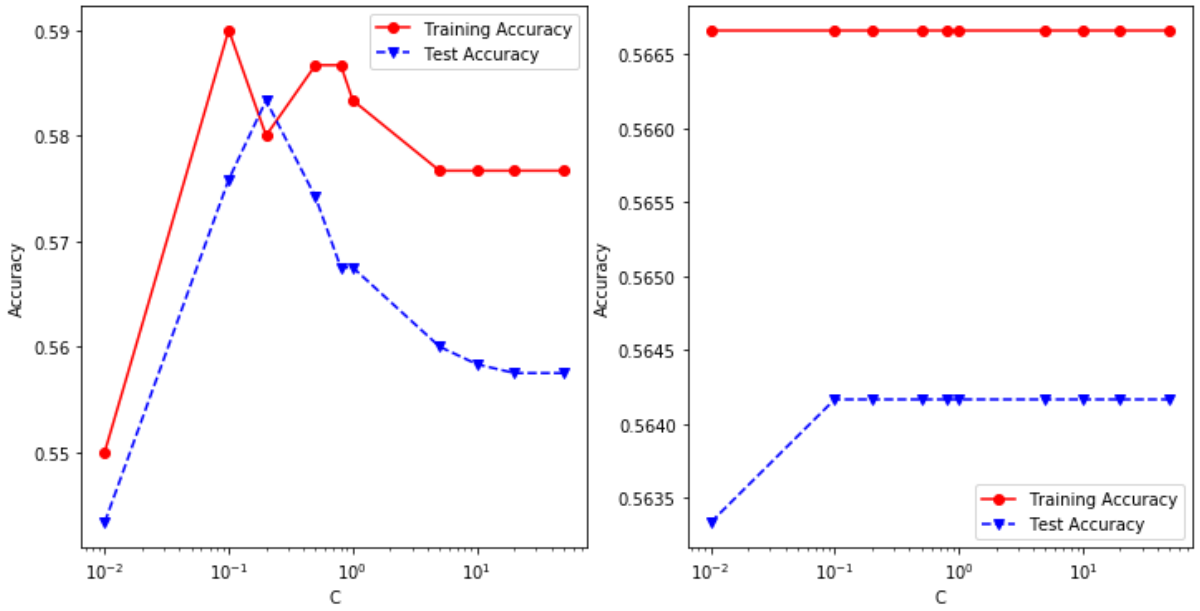
4.8.1 Principe de la régression logistique

La régression logistique est un **cas particulier d'analyse de régression** et est utilisée lorsque la **variable dépendante (classe) est nominalement échelonnée**. C'est le cas, par exemple, de la variable décision d'achat avec les deux valeurs "achète un produit" et "n'achète pas de produit".

4.8.2 Mise en pratique

```
from sklearn import linear_model
from sklearn.svm import SVC

C = [0.01, 0.1, 0.2, 0.5, 0.8, 1, 5, 10, 20, 50]
LRtrainAcc = []
LRtestAcc = []
SVMtrainAcc = []
SVMtestAcc = []

for param in C:
    clf = linear_model.LogisticRegression(C=param)
    clf.fit(X_train, Y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    LRtrainAcc.append(accuracy_score(Y_train, Y_predTrain))
    LRtestAcc.append(accuracy_score(Y_test, Y_predTest))

    clf = SVC(C=param,kernel='linear')
    clf.fit(X_train, Y_train)
```

```
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    SVMtrainAcc.append(accuracy_score(Y_train, Y_predTrain))
    SVMtestAcc.append(accuracy_score(Y_test, Y_predTest))

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,6))
ax1.plot(C, LRtrainAcc, 'ro-', C, LRtestAcc,'bv--')
ax1.legend(['Training Accuracy','Test Accuracy'])
ax1.set_xlabel('C')
ax1.set_xscale('log')
ax1.set_ylabel('Accuracy')

ax2.plot(C, SVMtrainAcc, 'ro-', C, SVMtestAcc,'bv--')
ax2.legend(['Training Accuracy','Test Accuracy'])
ax2.set_xlabel('C')
ax2.set_xscale('log')
ax2.set_ylabel('Accuracy')
```



4.9 Méthodes ensemblistes

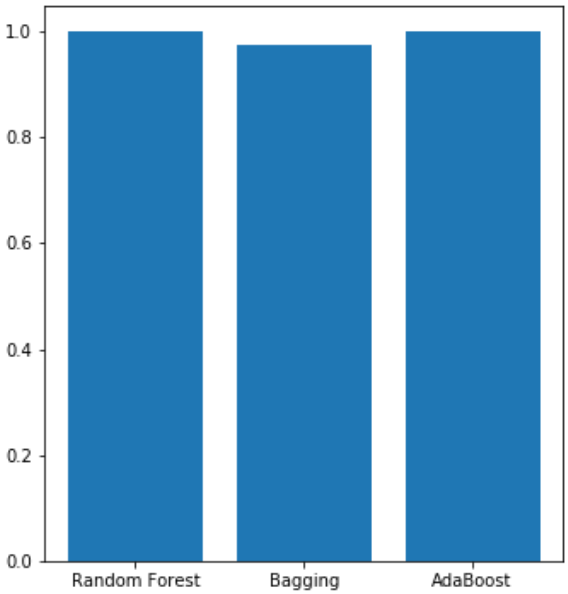
L'**apprentissage ensembliste** utilise plusieurs algorithmes d'apprentissage pour obtenir de meilleures prédictions. Un classifieur ensembliste construit un ensemble de classificateurs de base à partir des données d'apprentissage et effectue une classification en votant sur les prédictions faites par chaque classifieur de base. Nous considérons 3 types de classifieurs ensemblistes dans cet exemple : **bagging**, **boosting** et **random forest**.

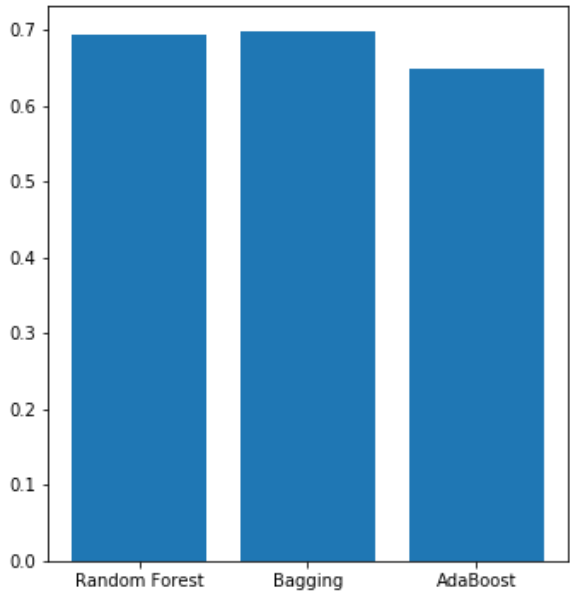
Dans l'exemple ci-dessous, nous ajustons 500 classifieurs de base à l'ensemble de données bidimensionnel à l'aide de chaque méthode ensembliste. Le classifieur de base correspond à un arbre de décision avec une profondeur maximale égale à 10.

```
from sklearn import ensemble
from sklearn.tree import DecisionTreeClassifier

numBaseClassifiers = 500
maxdepth = 10
trainAcc = []
```

```
testAcc = []

clf = ensemble.RandomForestClassifier(n_estimators=numBaseClassifiers)
clf.fit(X_train, Y_train)
Y_predTrain = clf.predict(X_train)
Y_predTest = clf.predict(X_test)
trainAcc.append(accuracy_score(Y_train, Y_predTrain))
testAcc.append(accuracy_score(Y_test, Y_predTest))

clf =
ensemble.BaggingClassifier(DecisionTreeClassifier(max_depth=maxdepth),n_estimators=numB
aseClassifiers)
clf.fit(X_train, Y_train)
Y_predTrain = clf.predict(X_train)
Y_predTest = clf.predict(X_test)
trainAcc.append(accuracy_score(Y_train, Y_predTrain))
testAcc.append(accuracy_score(Y_test, Y_predTest))

clf =
ensemble.AdaBoostClassifier(DecisionTreeClassifier(max_depth=maxdepth),n_estimators=nu
mBaseClassifiers)
clf.fit(X_train, Y_train)
Y_predTrain = clf.predict(X_train)
Y_predTest = clf.predict(X_test)
trainAcc.append(accuracy_score(Y_train, Y_predTrain))
testAcc.append(accuracy_score(Y_test, Y_predTest))

methods = ['Random Forest', 'Bagging', 'AdaBoost']
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,6))
ax1.bar([1.5,2.5,3.5], trainAcc)
ax1.set_xticks([1.5,2.5,3.5])
ax1.set_xticklabels(methods)
ax2.bar([1.5,2.5,3.5], testAcc)
ax2.set_xticks([1.5,2.5,3.5])
ax2.set_xticklabels(methods)
```

```
[Text(0,0,'Random Forest'), Text(0,0,'Bagging'), Text(0,0,'AdaBoost')]
```





Cette section fournit plusieurs exemples d'utilisation de la bibliothèque Python **sklearn** pour créer des modèles de classification à partir de données d'entrée données. Nous illustrons également le problème du sur-apprentissage du modèle et montrons comment appliquer différentes méthodes de classification à l'ensemble de données donné.

5 Apprentissage non-supervisé : Clustering

L'atelier suivant contient des exemples Python pour résoudre des problèmes de classification. L'analyse de cluster cherche à partitionner les données d'entrée en groupes d'instances étroitement liées afin que les instances qui appartiennent au même cluster soient plus similaires les unes aux autres qu'aux instances qui appartiennent à d'autres clusters. Dans cet atelier, nous fournirons des exemples d'utilisation de différentes techniques de clustering fournies par le package de la bibliothèque **scikit-learn**.

Lisez attentivement les instructions étape par étape ci-dessous. Pour exécuter le code, cliquez sur la cellule correspondante et appuyez simultanément sur les touches SHIFT-ENTER.

5.1 Clustering des K-moyennes (K-means)

L'algorithme de clustering k-means représente chaque cluster par son **centroïde** (centre de gravité) de cluster correspondant. L'algorithme partitionnerait les données d'entrée en k clusters disjoints en appliquant de manière itérative les deux étapes suivantes :

1. Formez k clusters en attribuant à chaque instance son centroïde le plus proche.
2. Recalculez le centroïde de chaque cluster.

- **Step-1:** Select the number K to decide the number of clusters.
- **Step-2:** Select random K points or centroids. (It can be other from the input dataset).
- **Step-3:** Assign each data point to their closest centroid, which will form the predefined K clusters.
- **Step-4:** Calculate the variance and place a new centroid of each cluster.
- **Step-5:** Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.
- **Step-6:** If any reassignment occurs, then go to step-4 else go to FINISH.
- **Step-7**: The model is ready.

Dans cette section, nous effectuons un regroupement de k-moyennes sur un exemple de jeu de données de classement de films. Nous créons d'abord l'ensemble de données comme suit.

	user	Film1	Film2	Film3	Film4
0	Jamal	5	5	2	1
1	Maroua	4	5	3	2
2	Bouali	4	4	4	3
3	lisa	2	2	4	5
4	Lechachi	1	2	3	4
5	Harir	2	1	5	5

```
import pandas as pd
ratings =
[['Jamal',5,5,2,1],['Maroua',4,5,3,2],['Bouali',4,4,4,3],['Lina',2,2,4,5],['Lechachi',1,2,3,4],['Harir',
2,1,5,5]]
titles = ['user','Film1','Film2','Film3','Film4']
movies = pd.DataFrame(ratings,columns=titles)
```

```
movies
```

Dans cet exemple de jeu de données, les 3 premiers utilisateurs ont aimé les films d'action (Film1 et Film2) tandis que les 3 derniers utilisateurs ont apprécié les films d'horreur (Film3 et Film4). Notre objectif est d'appliquer le clustering k-means sur les utilisateurs pour identifier des groupes d'utilisateurs ayant des préférences de film similaires.
L'exemple ci-dessous montre comment appliquer le regroupement des k-moyennes (avec k=2) sur les données de classement des films. Nous devons d'abord supprimer la colonne "user" avant d'appliquer l'algorithme de clustering. L'affectation de cluster pour chaque utilisateur est affichée sous la forme d'un objet dataframe.

```
from sklearn import cluster

data = movies.drop('user',axis=1)
k_means = cluster.KMeans(n_clusters=2, max_iter=50, random_state=1)
k_means.fit(data)
labels = k_means.labels_
pd.DataFrame(labels, index=movies.user, columns=['Cluster ID'])
```

	Cluster ID
user	
Jamal	1
Maroua	1
Bouali	1
lisa	0
Lechachi	0
Harir	0

L'algorithme de clustering k-means affecte les trois premiers utilisateurs à un cluster et les trois derniers utilisateurs au second cluster. Les résultats sont conformes à nos attentes. Nous pouvons également afficher le **centroïde** pour chacun des deux clusters.

```
centroids = k_means.cluster_centers_
pd.DataFrame(centroids,columns=data.columns)
```

	Film1	Film2	Film3	Film4
0	1.666667	1.666667	4.0	4.666667
1	4.333333	4.666667	3.0	2.000000

```
import numpy as np

testData = np.array([[4,5,1,2],[3,2,4,4],[2,3,4,1],[3,2,3,3],[5,4,1,4]])
labels = k_means.predict(testData)
labels = labels.reshape(-1,1)
usernames = np.array(['Name1','Name2','Name3','Name4','Name5']).reshape(-1,1)
cols = movies.columns.tolist()
cols.append('Cluster ID')
```

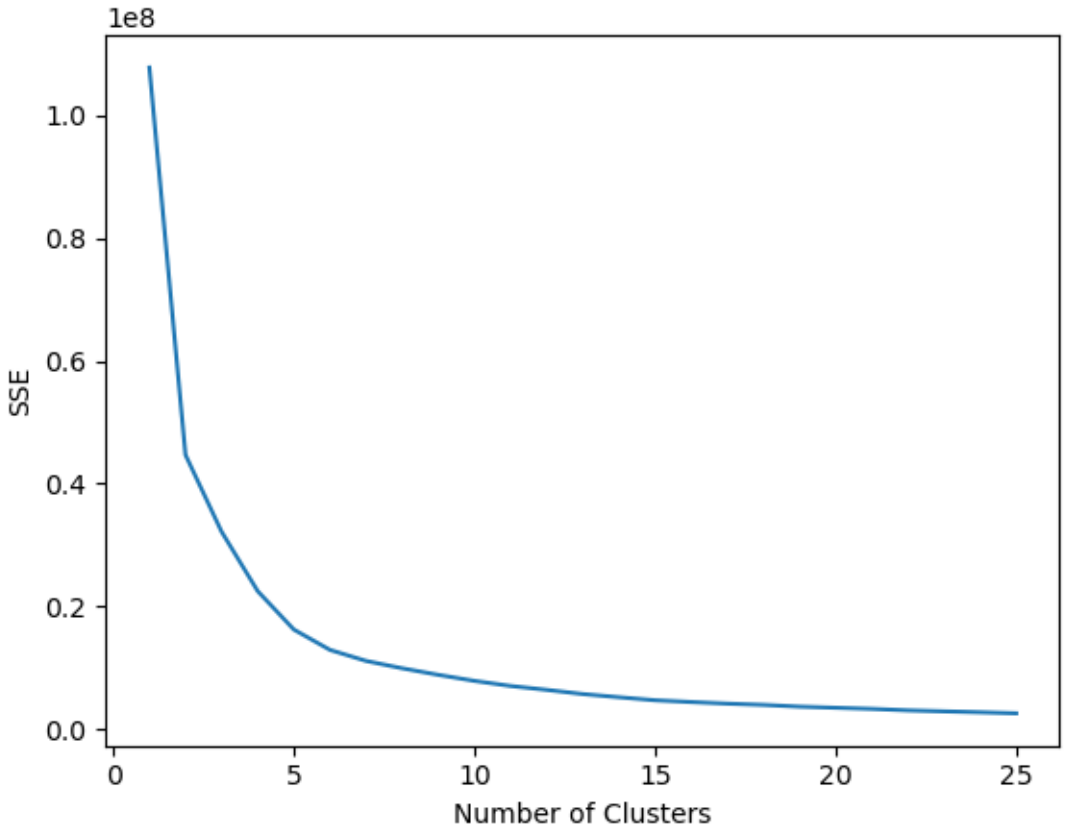
```
newusers = pd.DataFrame(np.concatenate((usernames, testData, labels),
axis=1),columns=cols)
newusers
```

	user	Film1	Film2	Film3	Film4	Cluster ID
0	paul	4	5	1	2	1
1	kim	3	2	4	4	0
2	liz	2	3	4	1	1
3	tom	3	2	3	3	0
4	bill	5	4	1	4	1

Pour déterminer le nombre de clusters dans les données, nous pouvons appliquer des k-moyennes avec un nombre variable de clusters de 1 à 6 et calculer leurs erreurs de somme des carrés (SSE) correspondantes, comme indiqué dans l'exemple ci-dessous. Le « coude » dans le graphique de l'SSE en fonction du nombre de clusters peut être utilisé pour estimer le nombre de clusters.

On remarque sur ce graphique, la forme d'un bras où le point le plus haut représente l'épaule et le point où K vaut 25 représente l'autre extrémité : la main. Le nombre optimal de clusters est le point représentant le **coude.** Ici le coude peut être représenté par K valant 3 à 10. C'est **le nombre optimal de clusters**. Généralement, le point du coude est celui du nombre de clusters à partir duquel la variance ne se réduit plus significativement. En effet, la "chute" de la courbe de variance (distortion) entre 3 et 10 clusters est significativement grande.
Le fait de chercher le point représentant le coude, a donné nom à cette méthode : **La méthode Elbow** (coude en anglais).
Finalement, le choix (au vu de ce graphique), entre 10 clusters, reste un peu flou et à votre discrétion. Le choix se fera en fonction de votre jeu de données et ce que vous cherchez à accomplir. Finalement, Il n'y a pas de solution unique à un problème de clustering.

```
import matplotlib.pyplot as plt
%matplotlib inline

numClusters = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25]
SSE = []
for k in numClusters:
    k_means = cluster.KMeans(n_clusters=k)
    k_means.fit(data)
    SSE.append(k_means.inertia_)

plt.plot(numClusters, SSE)
plt.xlabel('Number of Clusters')
plt.ylabel('SSE')
```

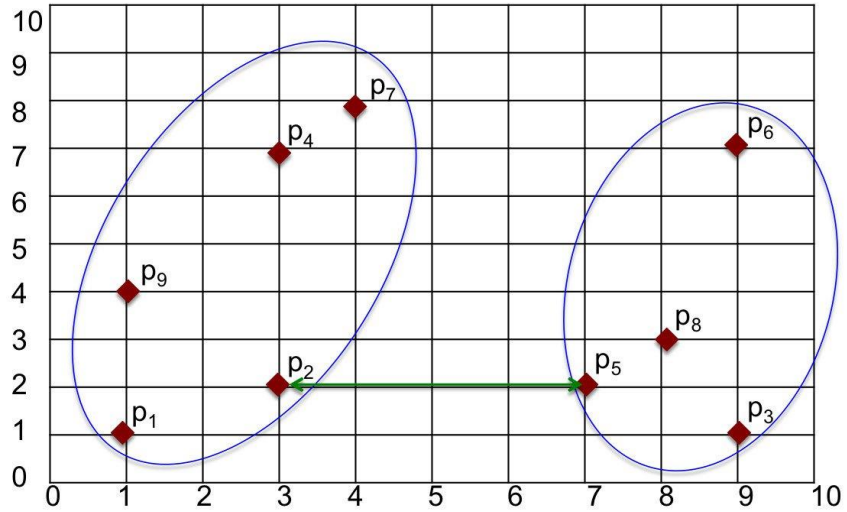


5.2 Clustering hiérarchique

Cette section présente des exemples d'application du regroupement hiérarchique à l'ensemble de données sur les vertébrés utilisé dans l'atelier 6 (Classification). Plus précisément, nous illustrons les résultats de l'utilisation de 3 algorithmes de clustering hiérarchique fournis par la bibliothèque Python **scipy** : (1) lien unique (MIN), (2) lien complet (MAX) et (3) moyenne de groupe. D'autres algorithmes de clustering hiérarchique fournis par la bibliothèque incluent la méthode des centroïdes et la méthode de Ward.

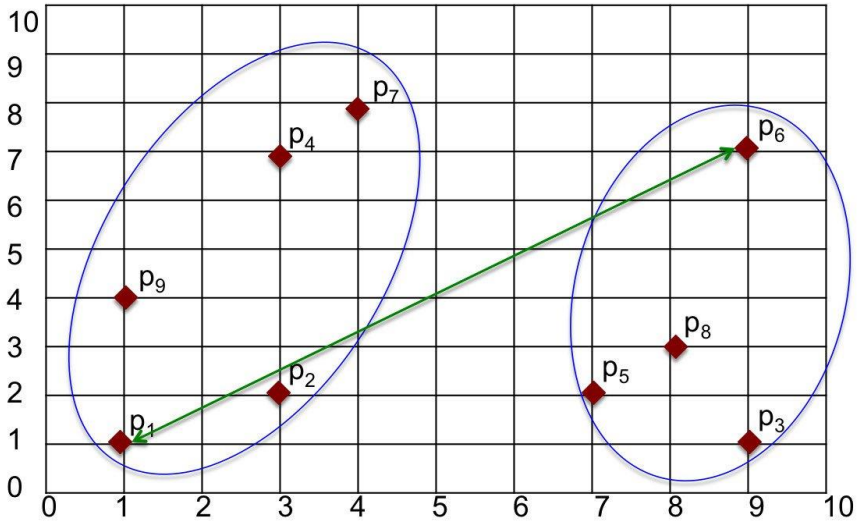
- Step 1: Compute the proximity matrix using a particular distance metric
- Step 2: Each data point is assigned to a cluster
- Step 3: Merge the clusters based on a metric for the similarity between clusters
- Step 4: Update the distance matrix
- Step 5: Repeat Step 3 and Step 4 until only a single cluster remains

Min (Single) Linkage



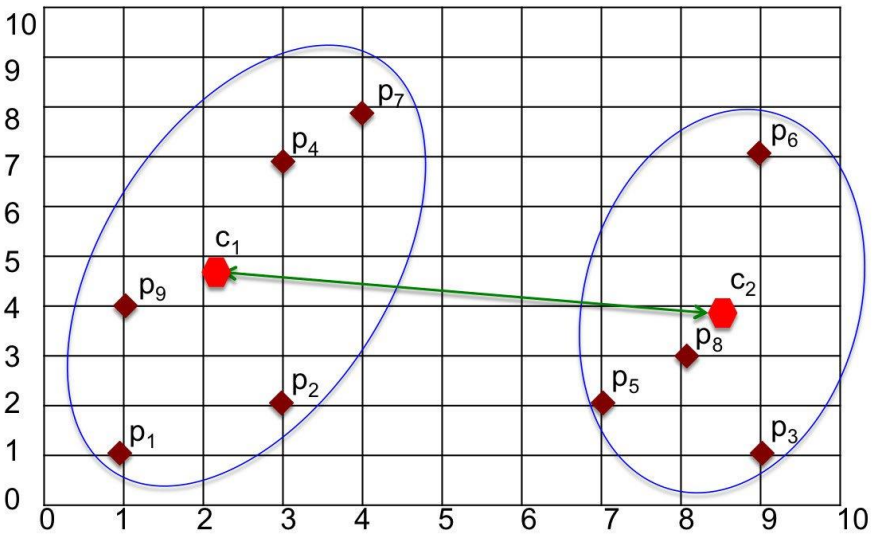
Min Linkage Method

Max (Complete) Linkage



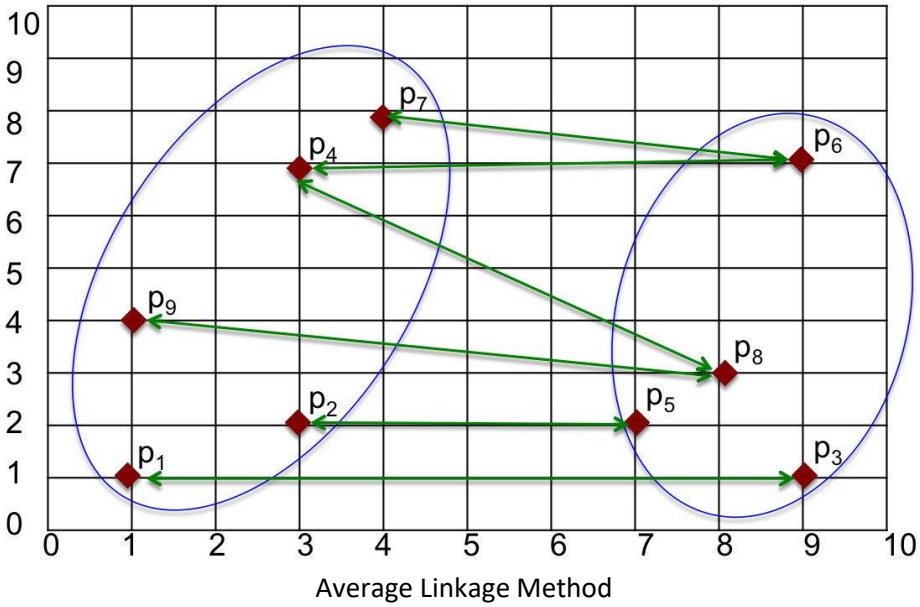
Max Linkage Method

Centroid Linkage



Centroid Linkage Method

Average Linkage

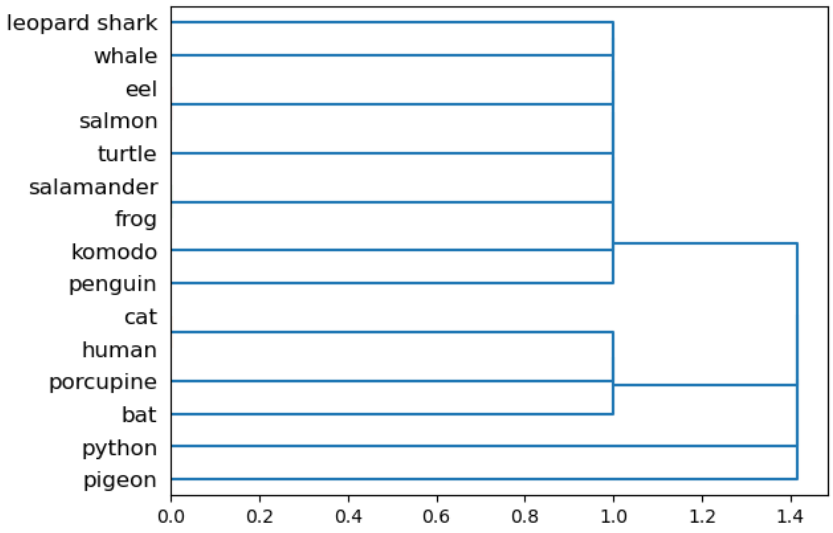



```
import pandas as pd

data = pd.read_csv('D:\ylebbah/tex/enseignement/poly-ml-ia-bigd-
ap/vertebrate.csv',header='infer')
data
```

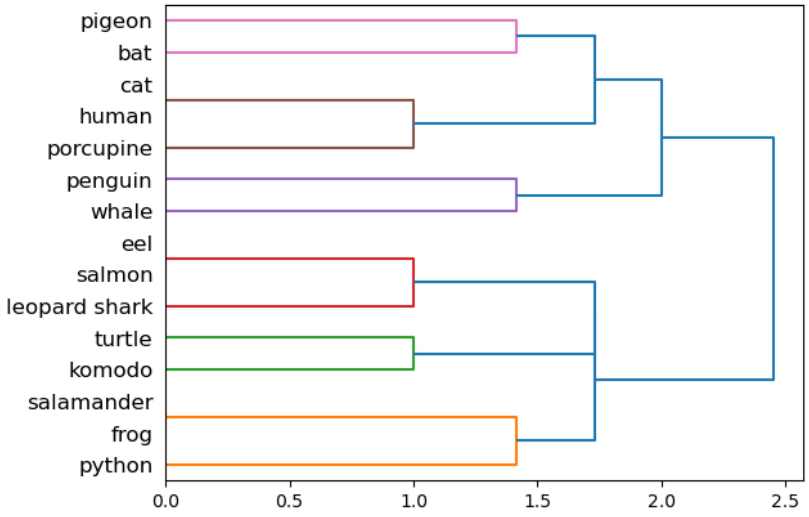
	Name	Warm-blooded	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class
0	human	1	1	0	0	1	0	mammals
1	python	0	0	0	0	0	1	reptiles
2	salmon	0	0	1	0	0	0	fishes
3	whale	1	1	1	0	0	0	mammals
4	frog	0	0	1	0	1	1	amphibians
5	komodo	0	0	0	0	1	0	reptiles
6	bat	1	1	0	1	1	1	mammals
7	pigeon	1	0	0	1	1	0	birds
8	cat	1	1	0	0	1	0	mammals
9	leopard shark	0	1	1	0	0	0	fishes
10	turtle	0	0	1	0	1	0	reptiles
11	penguin	1	0	1	0	1	0	birds
12	porcupine	1	1	0	0	1	1	mammals
13	eel	0	0	1	0	0	0	fishes
14	salamander	0	0	1	0	1	1	amphibians

```
from scipy.cluster import hierarchy
```

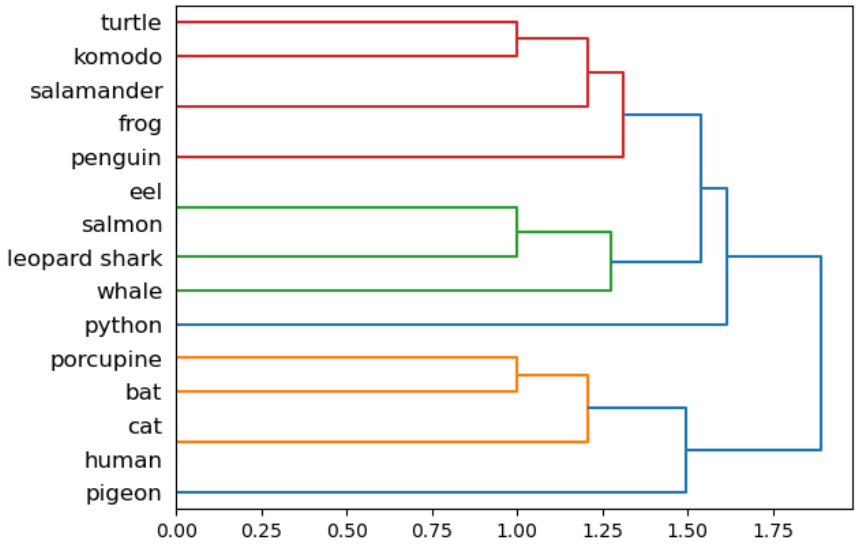
```
import matplotlib.pyplot as plt
%matplotlib inline

names = data['Name']
Y = data['Class']
X = data.drop(['Name','Class'],axis=1)
Z = hierarchy.linkage(X.values, 'single')
dn = hierarchy.dendrogram(Z, labels=names.tolist(),orientation='right')
```




```
Z = hierarchy.linkage(X.values, 'complete')
dn = hierarchy.dendrogram(Z,labels=names.tolist(),orientation='right')
```




```
Z = hierarchy.linkage(X.values, 'average')
dn = hierarchy.dendrogram(Z,labels=names.tolist(),orientation='right')
```



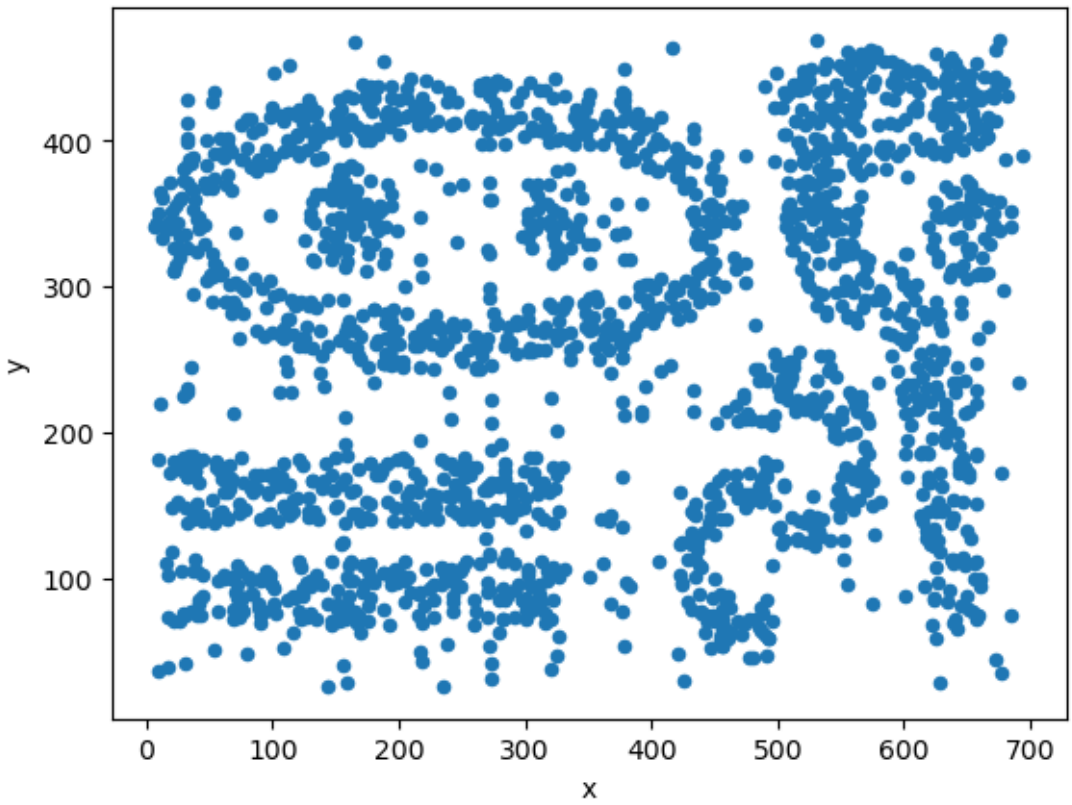
5.3 Clustering basé sur la densité

Le clustering basé sur la densité identifie les clusters individuels comme des régions à haute densité qui sont séparées par des régions à faible densité. **DBScan** est l'un des algorithmes de clustering basés sur la densité les plus populaires. Dans DBScan, les points de données sont classés en 3 types --- points centraux (**core** points), points frontières (**border** points) et points de bruit (**noise** points) --- en fonction de la densité de leur voisinage local. La densité locale de voisinage est définie selon 2 paramètres : le rayon de la taille du voisinage (**eps**) et le nombre minimum de points dans le voisinage (**min_samples**).

Pour cette approche, nous utiliserons un jeu de données bruité en 2 dimensions créé à l'origine par Karypis et al. [*George Karypis, Eui-Hong Han, and Vipin Kumar. CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling. IEEE Computer 32(8): 68-75, 1999.*] pour évaluer leur algorithme CHAMELEON proposé. L'exemple de code ci-dessous chargera et tracera la distribution des données.

```
import pandas as pd
data = pd.read_csv('D:\ylebbah/tex/enseignement/poly-ml-ia-bigd-ap/chameleon.data',
delimiter=' ', names=['x','y'])
data.plot.scatter(x='x',y='y')
```

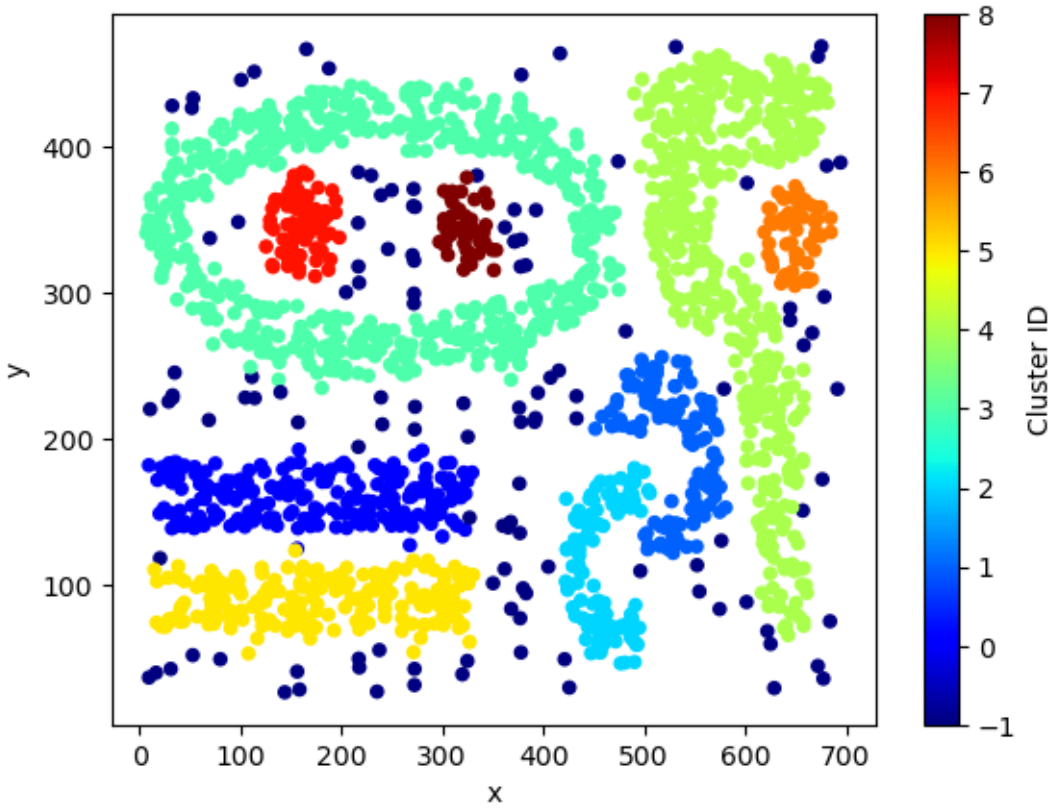
```
<Axes: xlabel='x', ylabel='y'>
```



Nous appliquons l'algorithme de clustering DBScan sur les données en définissant le rayon de voisinage (eps) à 15.5 et le nombre minimum de points (min_samples) à 5. Les clusters sont attribués à des identifiants compris entre 0 et 8 tandis que les points de bruit sont attribués à un cluster ID est égal à -1.

```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=15.5, min_samples=5).fit(data)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = pd.DataFrame(db.labels_,columns=['Cluster ID'])
result = pd.concat((data,labels), axis=1)
result.plot.scatter(x='x',y='y',c='Cluster ID', colormap='jet')
```

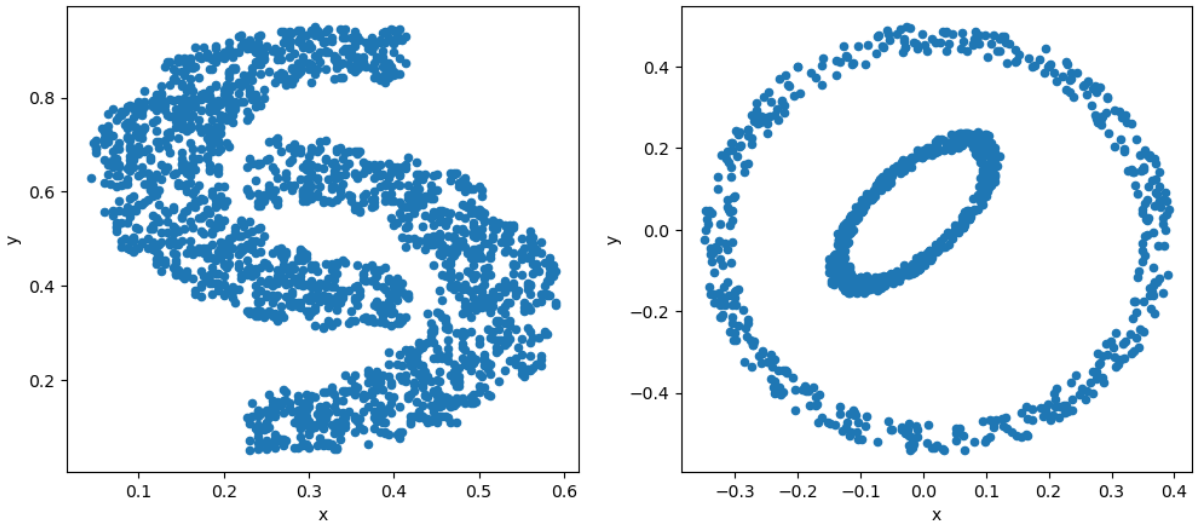
```
<Axes: xlabel='x', ylabel='y'>
```



5.4 Clustering spectral

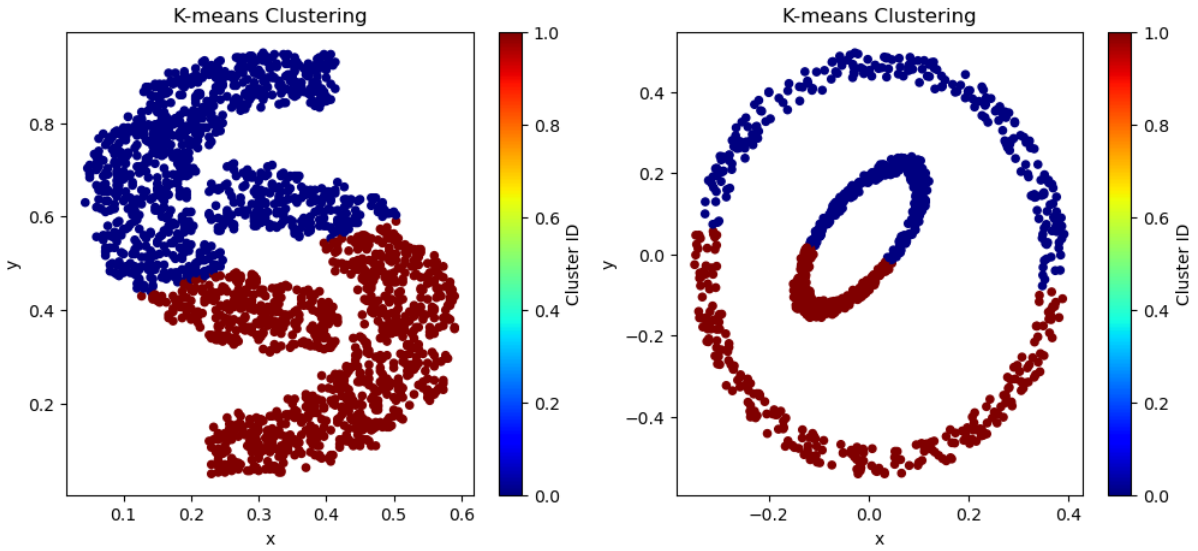
L'une des principales limitations de l'algorithme de clustering k-means est sa tendance à rechercher des clusters de forme globulaire. Ainsi, cela ne fonctionne pas lorsqu'il est appliqué à des ensembles de données avec des clusters de forme arbitraire ou lorsque les centroïdes de cluster se chevauchent. Le clustering spectral peut surmonter cette limitation en exploitant les propriétés du graphe de similarité pour surmonter ces limitations. Pour illustrer cela, considérons les ensembles de données bidimensionnels suivants.

```
import pandas as pd
data1 = pd.read_csv('D:\ylebbah/tex/enseignement/poly-ml-ia-bigd-ap/2d_data.txt',
delimiter=' ', names=['x','y'])
data2 = pd.read_csv('D:\ylebbah/tex/enseignement/poly-ml-ia-bigd-ap/elliptical.txt',
delimiter=' ', names=['x','y'])
fig, (ax1,ax2) = plt.subplots(nrows=1, ncols=2, figsize=(12,5))
data1.plot.scatter(x='x',y='y',ax=ax1)
data2.plot.scatter(x='x',y='y',ax=ax2)
```



Ci-dessous, nous démontrons les résultats de l'application des k-moyennes aux ensembles de données (avec k=2).

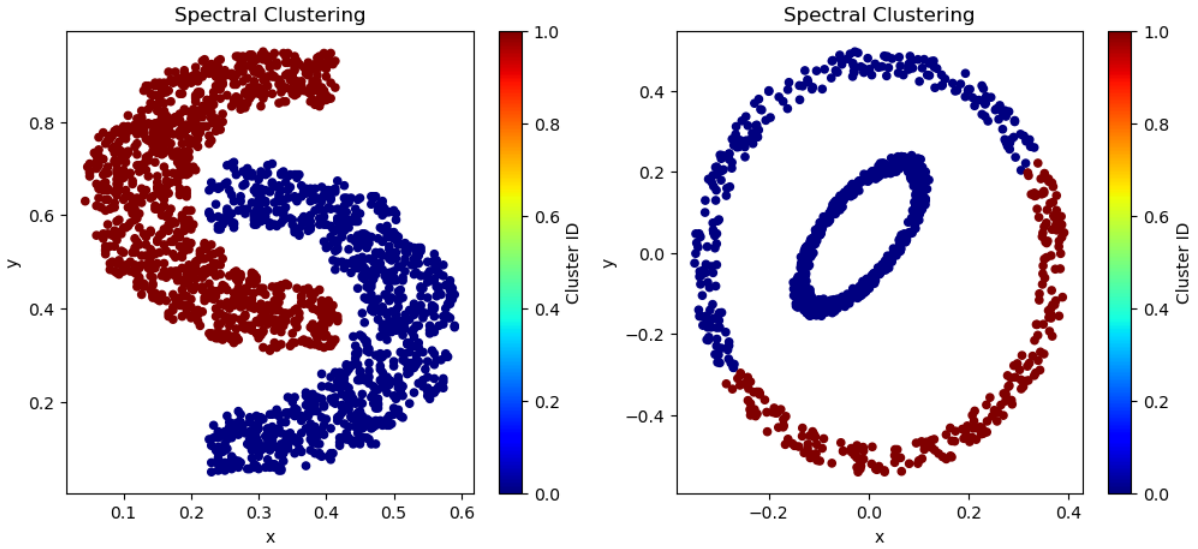
```
from sklearn import cluster

k_means = cluster.KMeans(n_clusters=2, max_iter=50, random_state=1)
k_means.fit(data1)
labels1 = pd.DataFrame(k_means.labels_,columns=['Cluster ID'])
result1 = pd.concat((data1,labels1), axis=1)

k_means2 = cluster.KMeans(n_clusters=2, max_iter=50, random_state=1)
k_means2.fit(data2)
labels2 = pd.DataFrame(k_means2.labels_,columns=['Cluster ID'])
result2 = pd.concat((data2,labels2), axis=1)

fig, (ax1,ax2) = plt.subplots(nrows=1, ncols=2, figsize=(12,5))
result1.plot.scatter(x='x',y='y',c='Cluster ID',colormap='jet',ax=ax1)
ax1.set_title('K-means Clustering')
result2.plot.scatter(x='x',y='y',c='Cluster ID',colormap='jet',ax=ax2)
ax2.set_title('K-means Clustering')
```



Les graphiques ci-dessus montrent les mauvaises performances du clustering k-means. Ensuite, nous appliquons le *clustering spectral* aux ensembles de données. Le clustering spectral ***convertit les données en un graphe de similarité*** et applique l'algorithme de partitionnement de graphe découpé normalisé pour générer les clusters. Dans l'exemple ci-dessous, nous utilisons la fonction de base radiale gaussienne comme mesure d'affinité (similarité). Les utilisateurs doivent ajuster la valeur du paramètre du noyau (gamma) afin d'obtenir les clusters appropriés pour l'ensemble de données donné.

```
from sklearn import cluster
import pandas as pd

spectral = cluster.SpectralClustering(n_clusters=2,random_state=1,affinity='rbf',gamma=5000)
spectral.fit(data1)
labels1 = pd.DataFrame(spectral.labels_,columns=['Cluster ID'])
result1 = pd.concat((data1,labels1), axis=1)

spectral2 = cluster.SpectralClustering(n_clusters=2,random_state=1,affinity='rbf',gamma=100)
spectral2.fit(data2)
labels2 = pd.DataFrame(spectral2.labels_,columns=['Cluster ID'])
result2 = pd.concat((data2,labels2), axis=1)

fig, (ax1,ax2) = plt.subplots(nrows=1, ncols=2, figsize=(12,5))
result1.plot.scatter(x='x',y='y',c='Cluster ID',colormap='jet',ax=ax1)
ax1.set_title('Spectral Clustering')
result2.plot.scatter(x='x',y='y',c='Cluster ID',colormap='jet',ax=ax2)
ax2.set_title('Spectral Clustering')
```

```
Text(0.5, 1.0, 'Spectral Clustering')
```



5.5 Résumé

Cet atelier illustre des exemples d'utilisation de différentes implémentations Python d'algorithmes de clustering. Des algorithmes tels que k-means, clustering spectral et DBScan sont conçus pour créer des partitions disjointes des données, tandis que les algorithmes à lien unique, à lien complet et à moyenne de groupe sont conçus pour générer une hiérarchie de partitions de cluster.

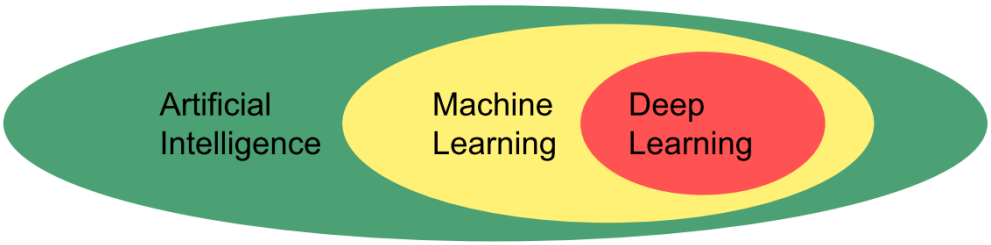
6 Deep learning

https://graphics.stanford.edu/courses/cs468-17-spring/LectureSlides/L10%20-%20intro_to_deep_learning.pdf

What is Deep Learning?

Deep learning allows computational models that are composed of **multiple processing layers to learn representations of data** with **multiple levels of abstraction**.

Deep Learning by Y. LeCun et al. Nature 2015



//////////////////////////////////

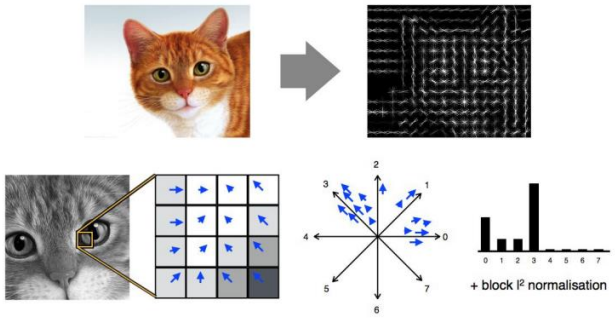
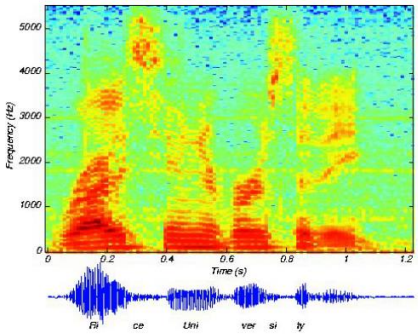


Image: HoG



Audio: Spectrogram

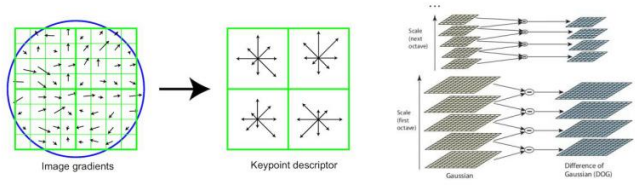
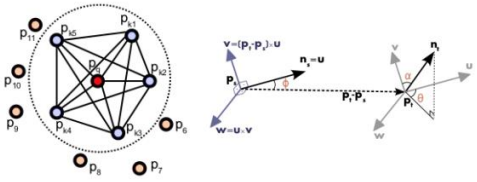
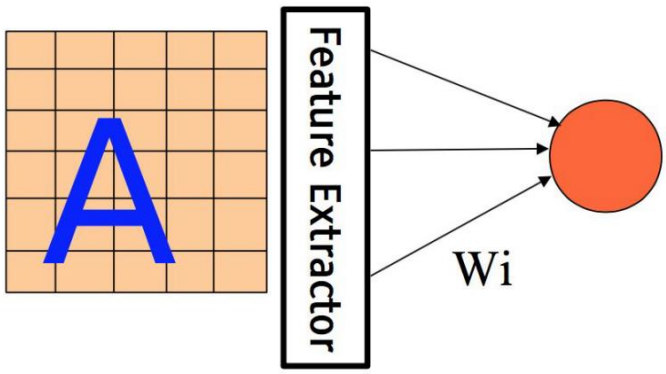


Image: SIFT



Point Cloud: PFH



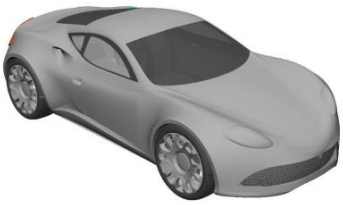
Linear Regression
SVM
Decision Trees
Random Forest
...

$$y = sign\left(\sum_{i=1}^{N} W_i F_i(X) + b\right)$$

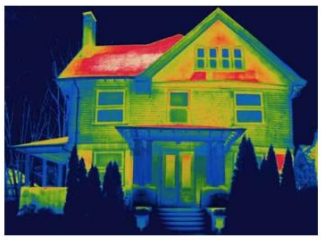
////////////////////////////////////

Image

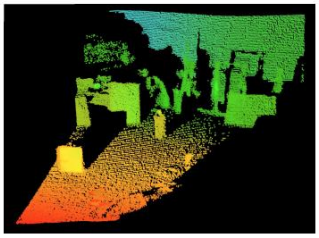
Video



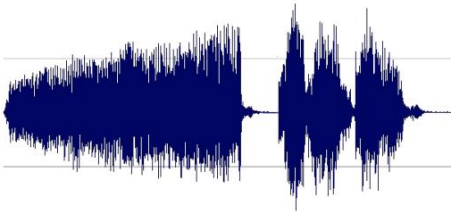
3D CAD Model



Thermal Infrared



Depth Scan



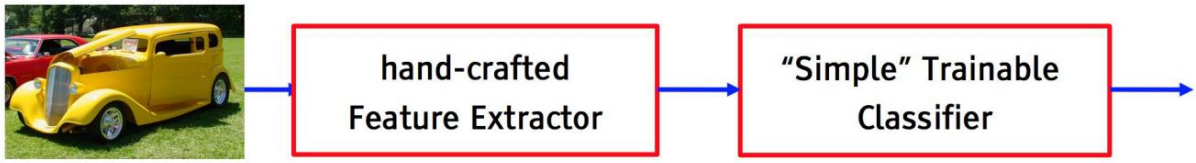
Audio

Can we automatically learn “good” feature representations?

////////////////////////////////

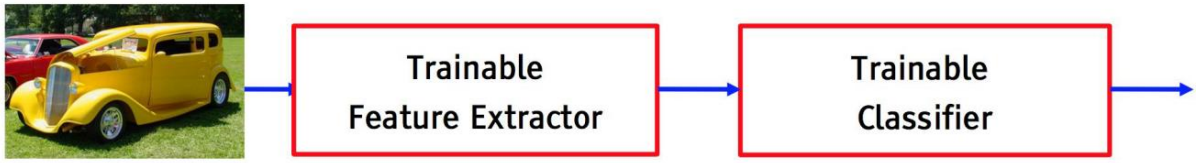
The traditional model of pattern recognition (since the late 50's)

- Fixed/engineered features (or fixed kernel) + trainable classifier



End-to-end learning / Feature learning / Deep learning

- Trainable features (or kernel) + trainable classifier



////////////////////////////////////

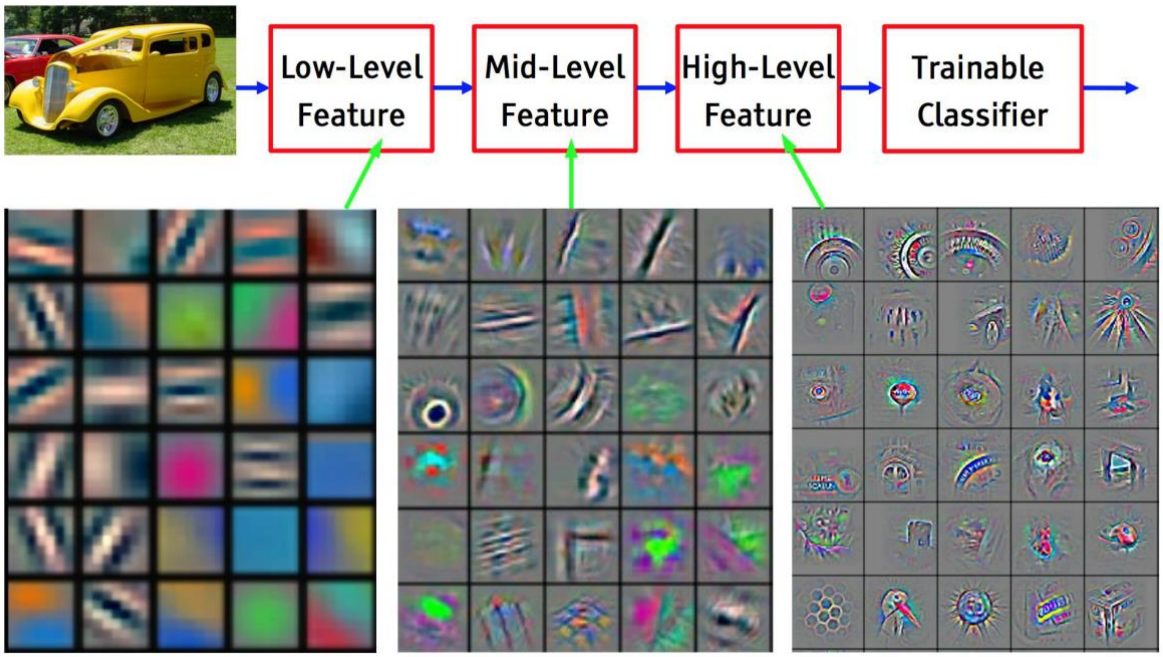
Traditional Pattern Recognition: Fixed/Handcrafted Feature Extractor

Mainstream Modern Pattern Recognition: Unsupervised mid-level features

Deep Learning: Representations are hierarchical and trained

//

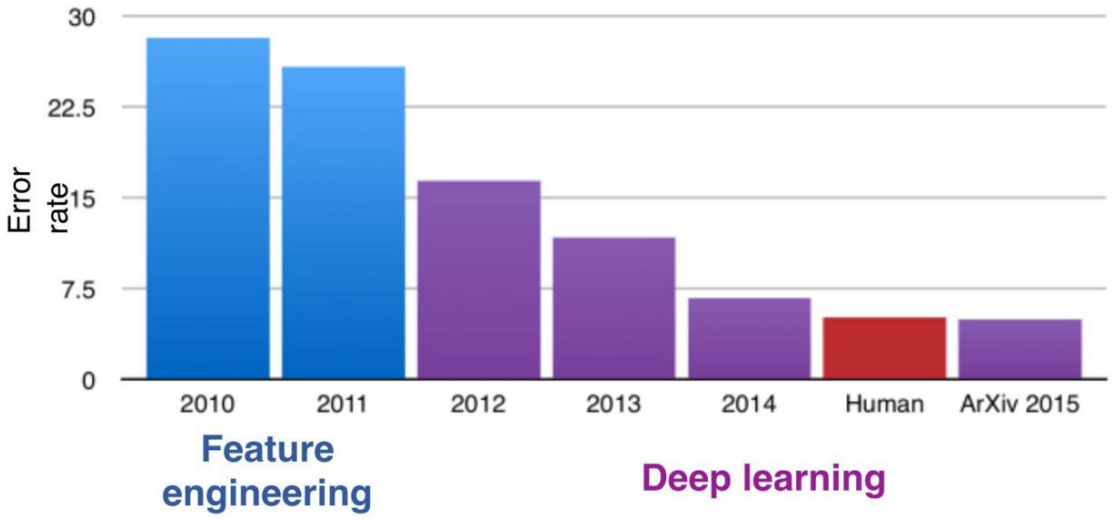
It's deep if it has more than one stage of non-linear feature transformation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

//

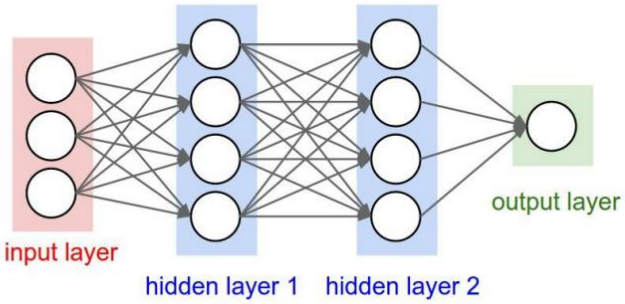
ImageNet 1000 class image classification accuracy



//

A Simple Neural Network

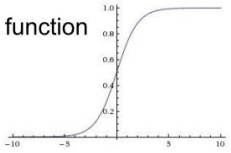
Use recent three days' average temperature to predict tomorrow's average temperature.

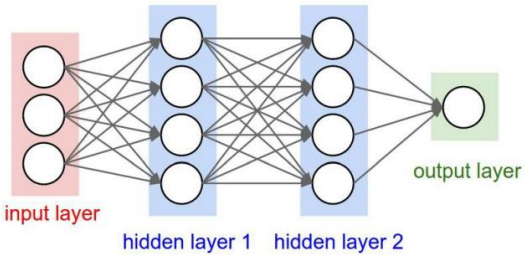


//

A Simple Neural Network

Sigmoid function



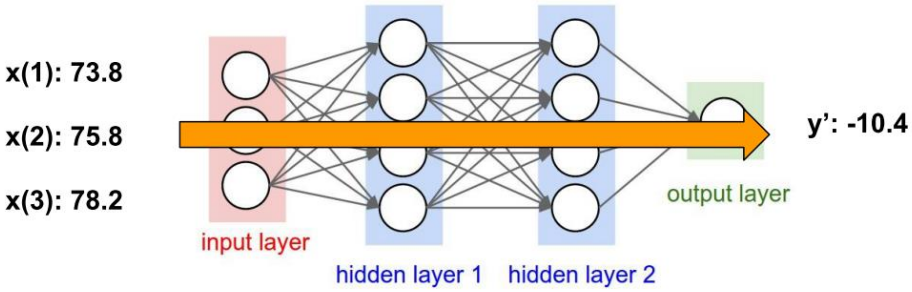


W1, b1, W2, b2, W3, b3 are network parameters that need to be learned.

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

//

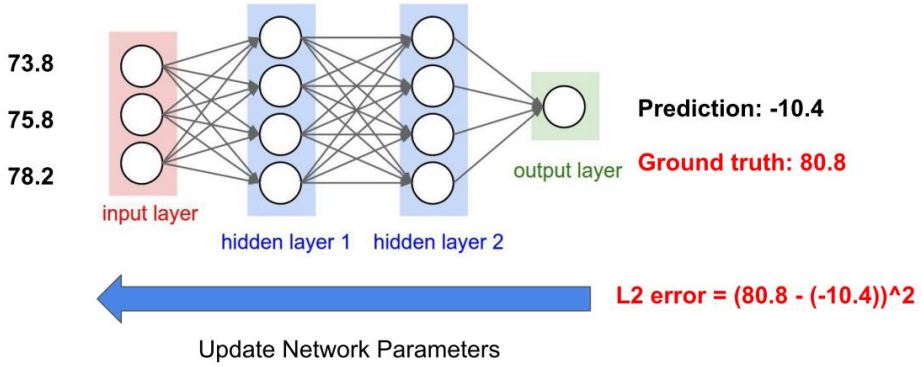
Neural Network: Forward Pass



$$y' = W_3 f(W_2 f(W_1 x + b_1) + b_2) + b_3)$$

//

Neural Network: Backward Pass



Minimize: $L(x, y; W, b) = \sum_{i=1}^{N} (W_3 f(W_2 f(W_1 x_i + b_1) + b_2) + b_3) - y_i)^2$

Given N training pairs: $\{x_i, y_i\}_{i=1}^{N}$

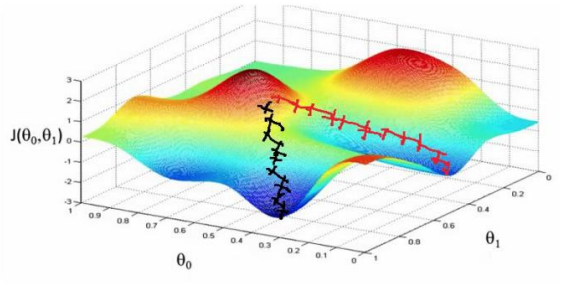
//

Neural Network: Backward Pass

Minimize: $L(x, y; W, b) = \sum_{i=1}^{N} (W_3 f(W_2 f(W_1 x_i + b_1) + b_2) + b_3) - y_i)^2$

Given N training pairs: $\{x_i, y_i\}_{i=1}^{N}$

Non-convex optimization :(
Use gradient descent!

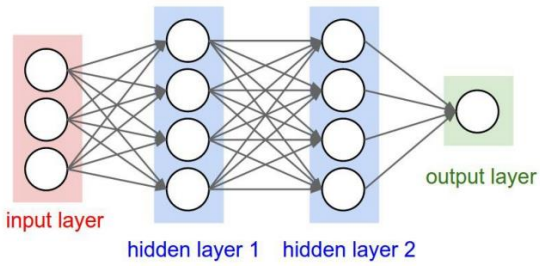


Parameter update example:

$$W_3 = W_3 - \eta \frac{\partial L}{\partial W_3}$$

//

A Simple Neural Network



Model: Multi-Layer Perceptron (MLP) $y' = W_3 f(W_2 f(W_1 x + b_1) + b_2) + b_3)$

Loss function: L2 loss $l(y, y') = (y - y')^2$

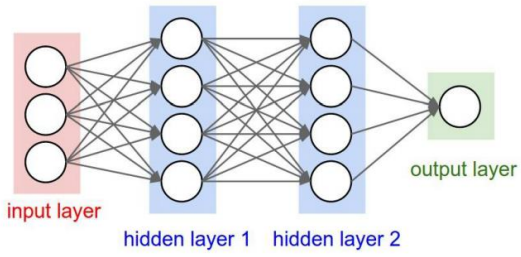
Optimization: Gradient descent $W = W - \eta \frac{\partial L}{\partial W}$

//

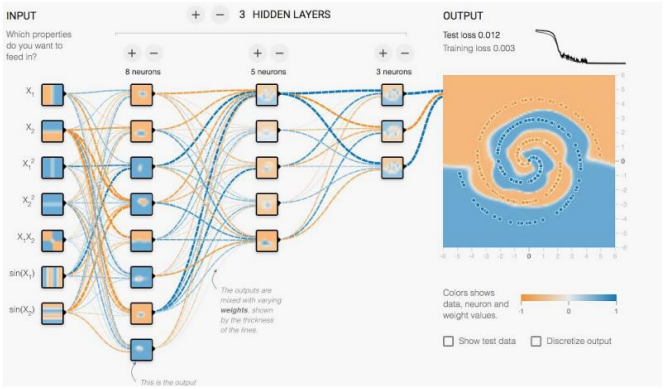
Multi-Layer Perceptron

Fully Connected

Non-linear Op



http://playground.tensorflow.org/

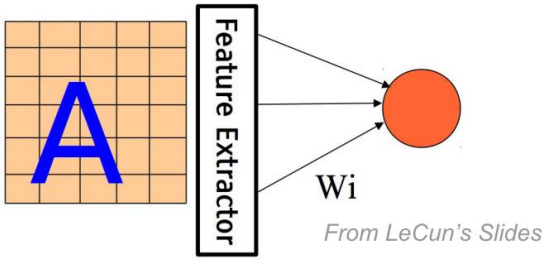


////////////////////

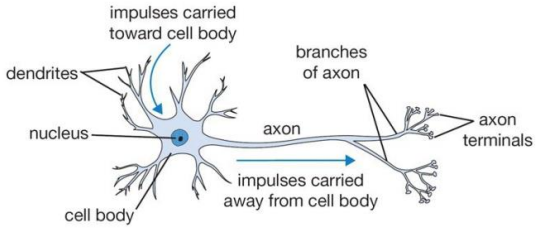
Fully Connected

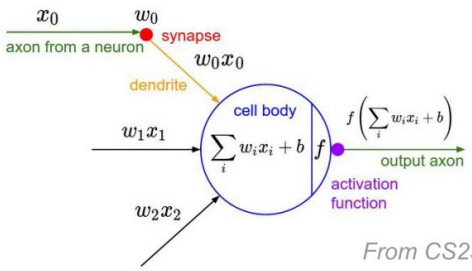
- The first learning machine: the **Perceptron** Built at Cornell in 1960
- The Perceptron was a (binary) linear classifier on top of a simple feature extractor

$$y = sign\left(\sum_{i=1}^{N} W_i F_i(X) + b\right)$$



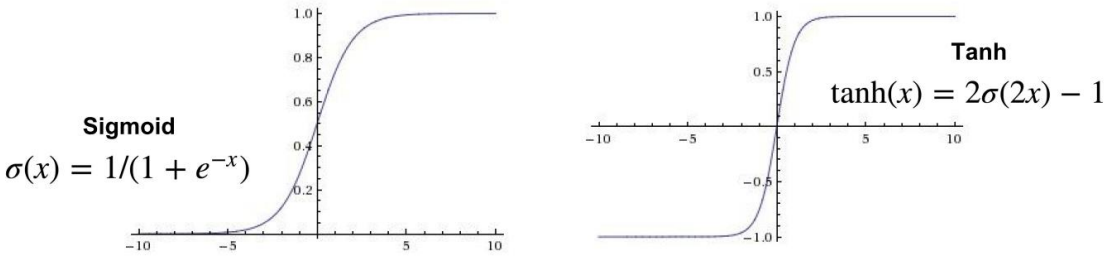
From LeCun's Slides





From CS231N

Non-linear Op

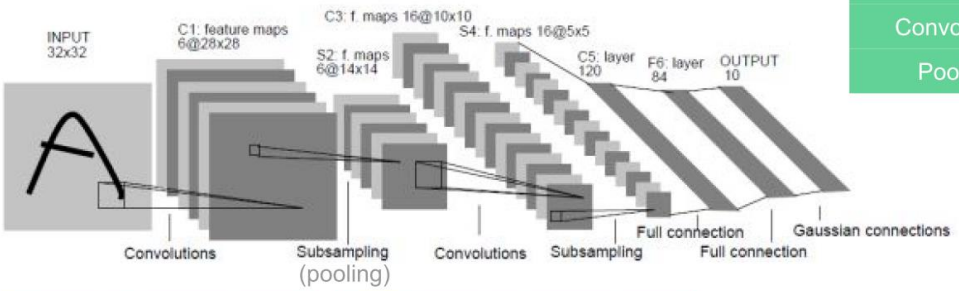


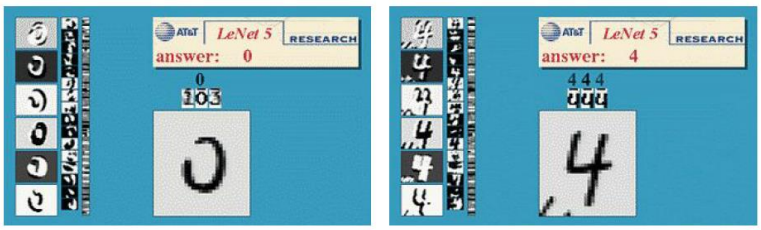
Major drawbacks: Sigmoids saturate and kill gradients

From CS231N

Convolutional Neural Network : LeNet (1998 by LeCun et al.)

Fully Connected
Non-linear Op
Convolution
Pooling



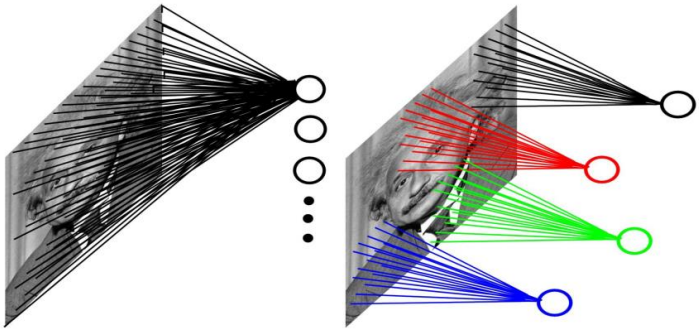


One of the first successful applications of CNN.

Convolution

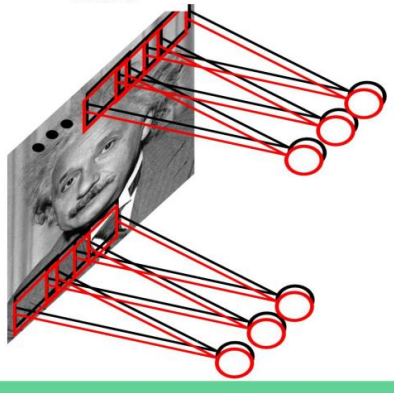
Fully Connected NN in high dimension

- Example: 200x200 image
  - Fully-connected, 400,000 hidden units = 16 billion parameters
  - Locally-connected, 400,000 hidden units 10x10 fields = 40 million params
  - Local connections capture local dependencies



Shared Weights & Convolutions: Exploiting Stationarity

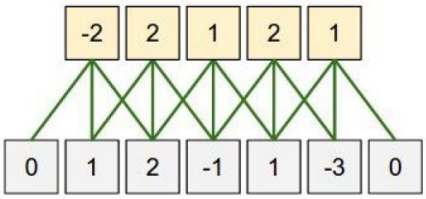
- Example: 200x200 image
  - 400,000 hidden units with 10x10 fields = 1000 params
  - 10 feature maps of size 200x200, 10 filters of size 10x10



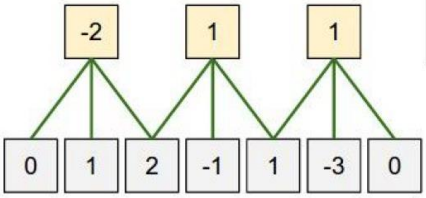
Slide from LeCun

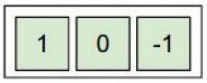
Convolution

Stride 1

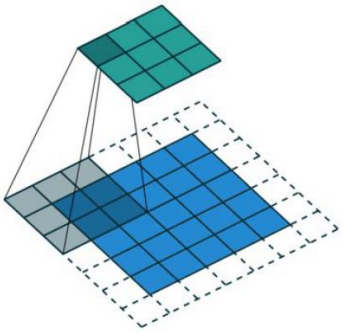


Stride 2

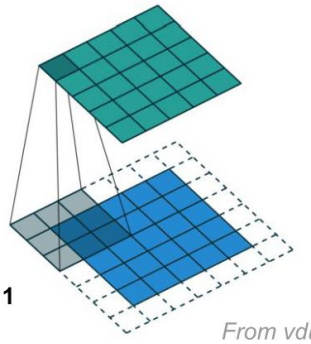




From CS231N



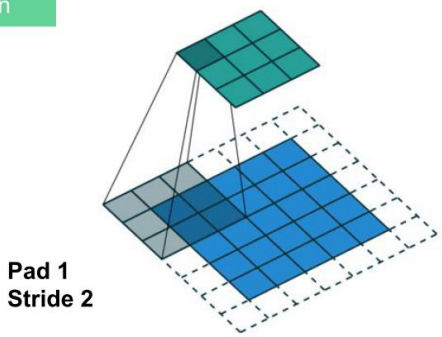
Pad 1
Stride 2



Pad 1
Stride 1

From vdumoulin/conv_arithmetic

Convolution

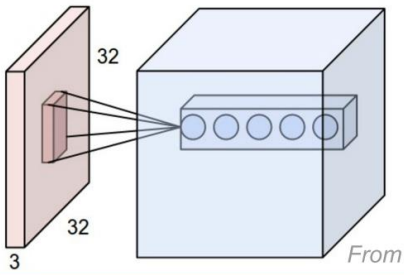


5x5 RGB Image
5x5x3 array

3x3 kernel, 2 output channels, pad 1, stride 2
weights: 2x3x3x3 array
bias: 2x1 array

Output
3x3x2 array

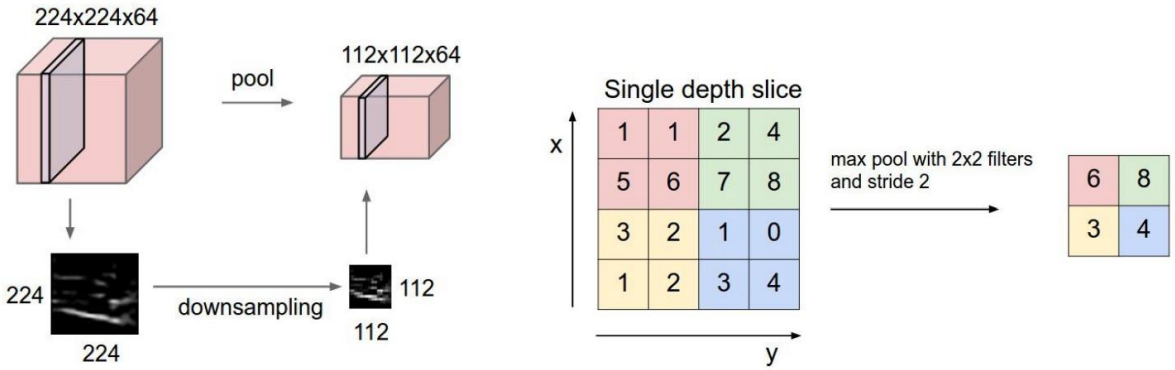
H' = (H - K)/stride_H + 1
= (7-3)/2 + 1 = 3



From CS231N

Pooling

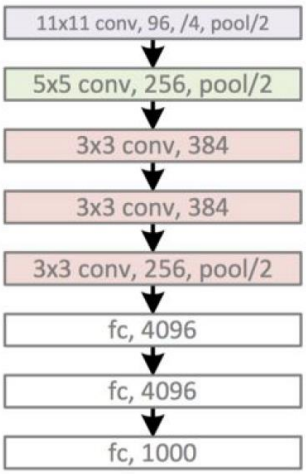
Pooling layer (usually inserted in between conv layers) is used to reduce spatial size of the input, thus reduce number of parameters and overfitting.

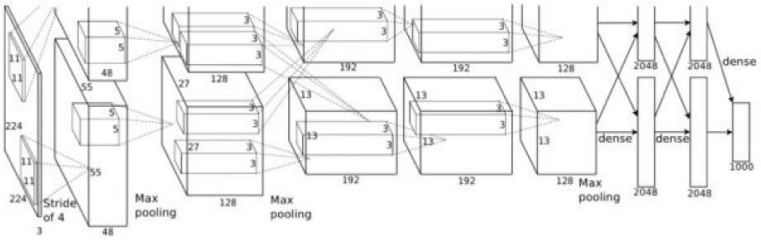


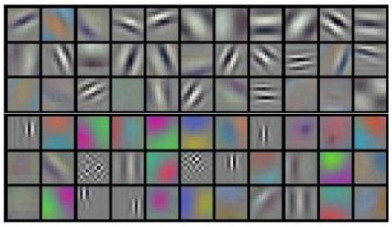
Discarding pooling layers has been found to be important in training good generative models, such as variational autoencoders (VAEs) or generative adversarial networks (GANs).
It seems likely that future architectures will feature very few to no pooling layers.

From CS231N

AlexNet (2012 by Krizhevsky et al.)



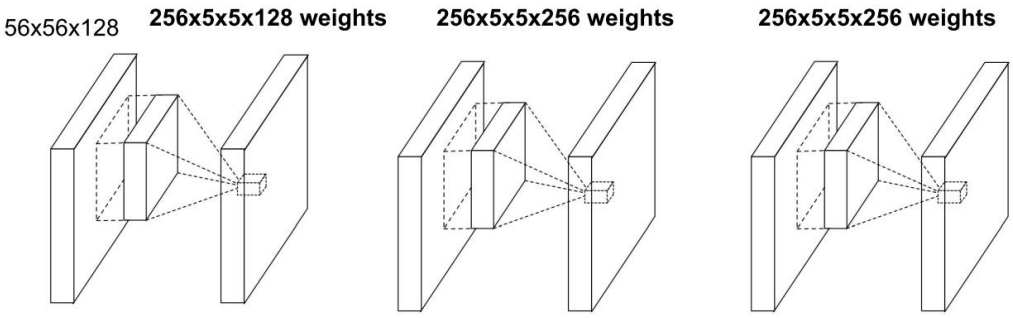




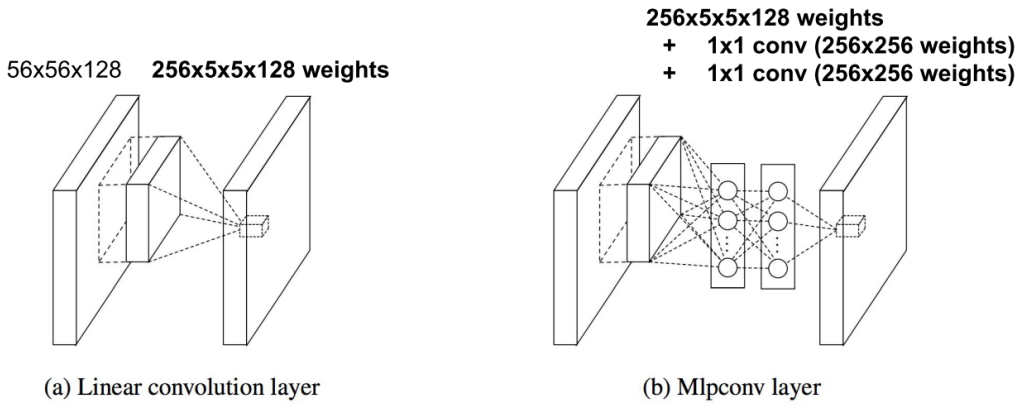
The first work that popularized Convolutional Networks in Computer Vision

What's different?

Network in Network (2013 by Min Lin et al.)



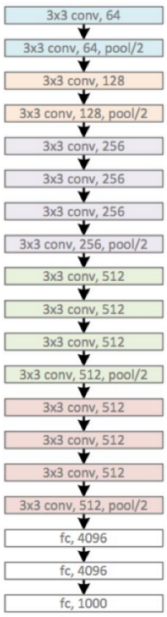
Network in Network (2013 by Min Lin et al.)



(a) Linear convolution layer (b) Mlpconv layer

1x1 convolution: MLP in each pixel's channels
Use very little parameters for large model capacity.

VGG (2014 by Simonyan and Zisserman) The runner-up in ILSVRC 2014



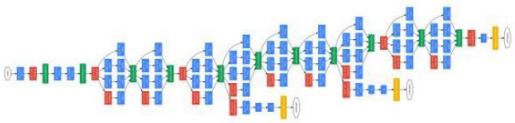
Karen Simonyan, Andrew Zisserman: **Very Deep** Convolutional Networks for Large-Scale Image Recognition.

- Its main contribution was in showing that the depth of the network is a critical component for good performance.
- Their final best network contains 16 CONV/FC layers and, appealingly, features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end.

-- quoted from CS231N

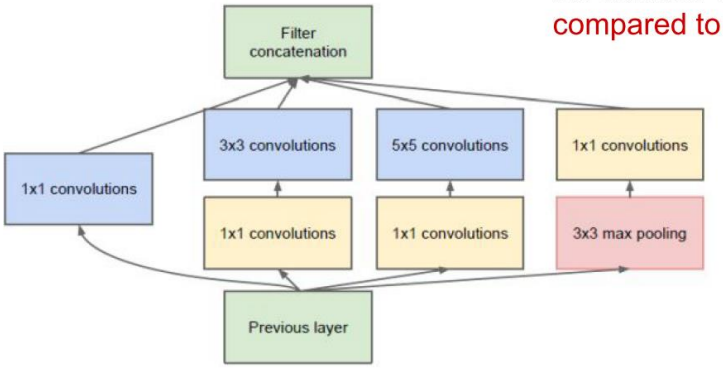
GoogleNet (2015 by Szegedy et al.)

The winner in ILSVRC 2014



Its main contribution was the development of an Inception Module and the using Average Pooling instead of Fully Connected layers at the top of the ConvNet, which dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).

-- edited from CS231N

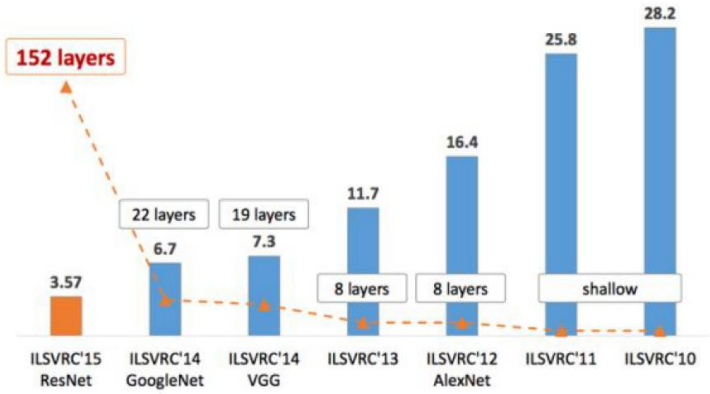


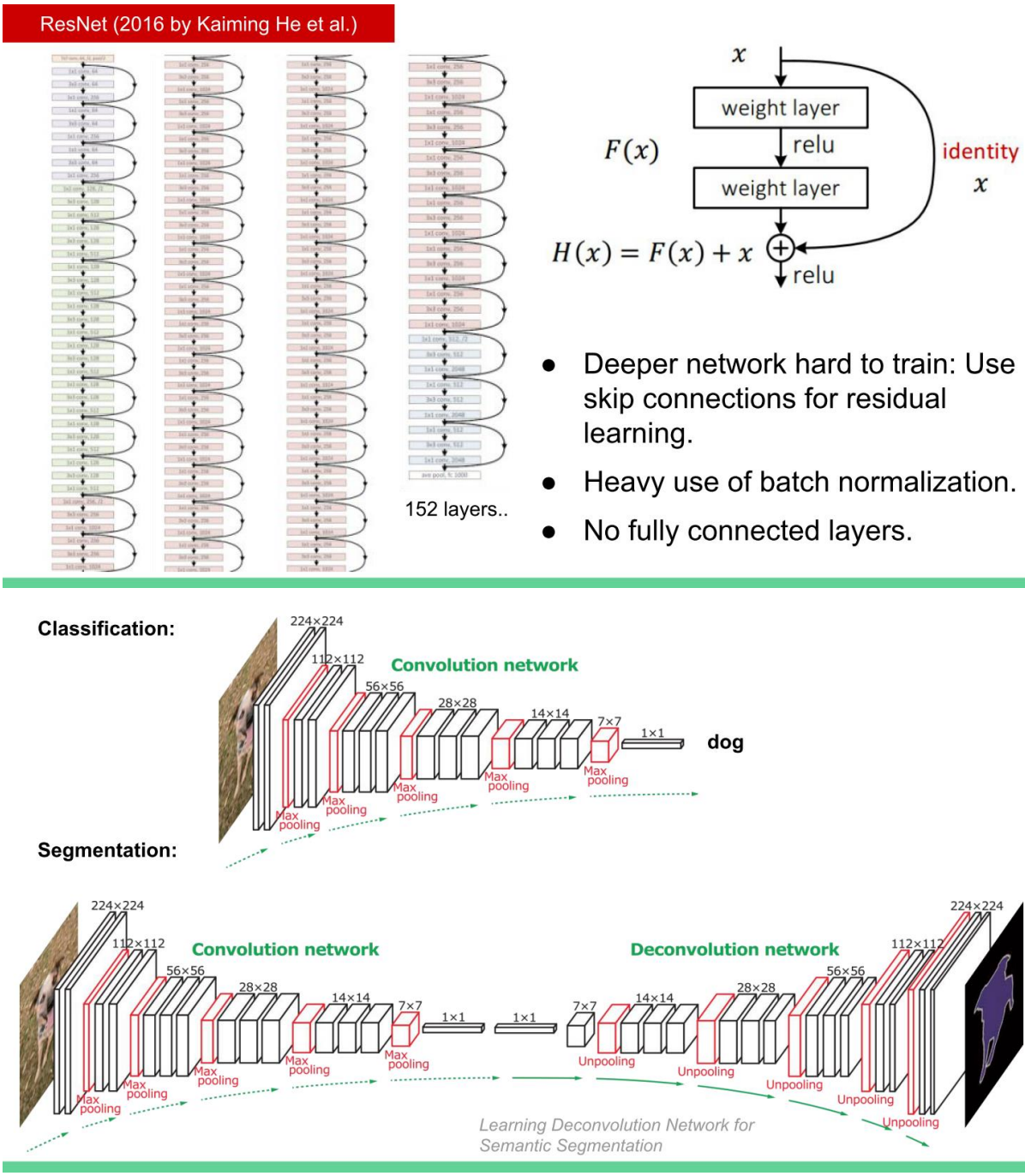
An Inception Module: a new building block..

Tip on ConvNets:
Usually, most computation is spent on convolutions, while most space is spent on fully connected layers.

ResNet (2016 by Kaiming He et al.)

The winner in ILSVRC 2015





6.1 Deep-learning (in details, forthcoming)

Ressources :

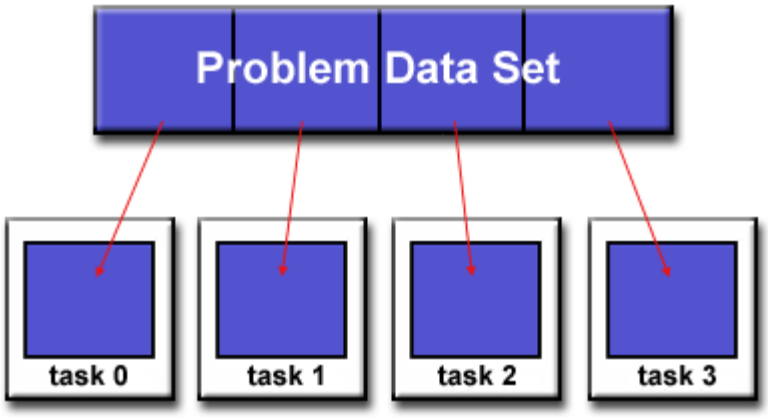
- https://www.csd.uoc.gr/~hy539/schedule.html

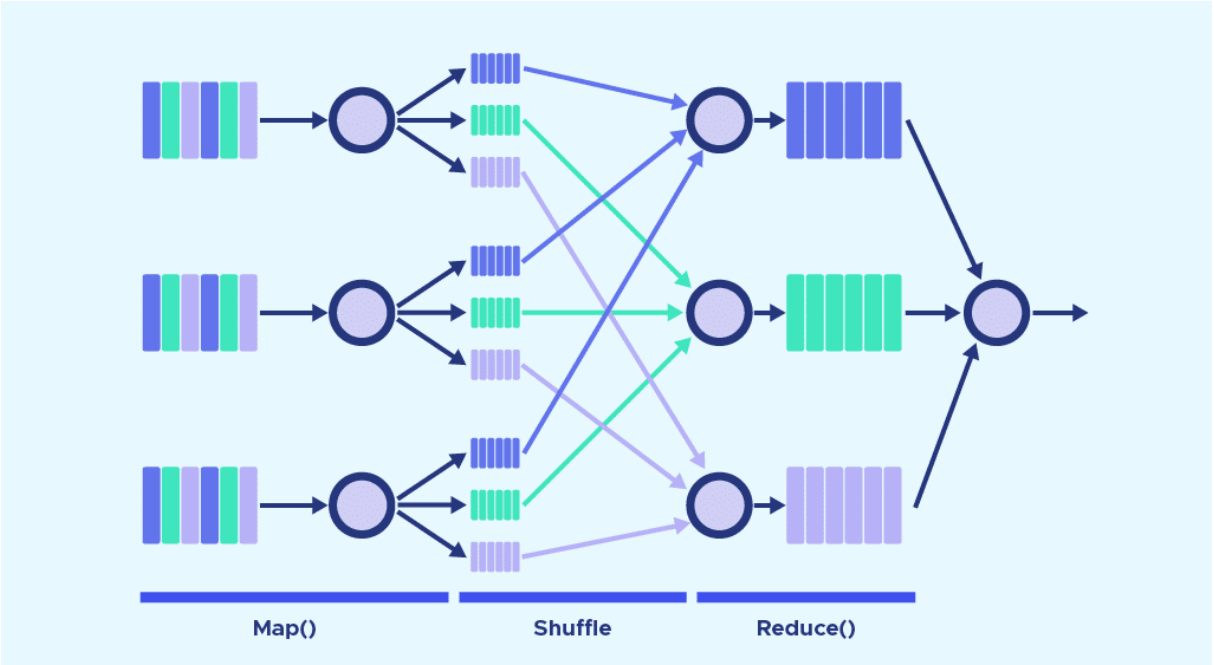
7 Introduction au traitement de données massives (big data)

Références :
https://pnavaro.github.io/big-data/05-MapReduce.html
https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial
https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/
https://www.softcover.io/read/f0b3ea72/introbigdata_book/mapreduce_chapter
https://datascientest.com/en/mapreduce-how-to-use-it-for-big-data

7.1 Map Reduce paradigm

When solving a problem on a massive, the idea is to decompose the problem into subproblems solved in parallel tasks.





MapReduce is the programming model of the Hadoop framework. It enables the analysis of massive volumes of Big Data through parallel processing. Discover everything you need to know: overview, functioning, alternatives, benefits, training...

The massive volumes of Big Data offer numerous opportunities for businesses. However, it can be challenging to process this data quickly and efficiently on traditional systems. It's necessary to turn to new solutions specifically designed for this purpose.

The MapReduce programming model is one of these solutions. It was originally created by Google to analyze the results of its search engine. Over time, this tool has become extremely popular for its ability to break down terabytes of data to process them in parallel. This approach allows for faster results.

The MapReduce paradigm consists in three steps:

1. **The MAP function**
   The input data is divided into smaller blocks, and each of these blocks is assigned to a "mapper" for processing.
   Let's take an example of a file containing 100 records to process. It's possible to use 100 mappers simultaneously to process each record separately. However, multiple records can also be assigned to each mapper.
   In reality, the Hadoop framework automatically decides how many mappers to use. This decision depends on the size of the data to be processed and the available memory blocks on each server.
   The Map function receives the input from the disk in the form of "key/value" pairs. These pairs are processed, and another set of intermediate key/value pairs is produced.
2. **The REDUCE function**
   After all the mappers have completed their processing tasks, the framework shuffles and organizes the results. It then sends them to the "reducers." It's worth noting that a reducer cannot start if a mapper is still active.
   The Reduce function also receives inputs in the form of key/value pairs. All the values produced by the map with the same key are assigned to a single reducer. The reducer is responsible for aggregating the values for that key. Reduce then produces a final output, still in the form of key/value pairs. However, the types of keys and values can vary depending on the use case. All inputs and outputs are stored in HDFS. It's important to mention that the map function is essential for filtering and sorting the initial data, while the reduce function is optional.
3. Combine, Partition and **Shuffling**
   The purpose of the **Shuffling function** is to associate all the values of a similar key.
   There are two intermediate steps between Map and Reduce called Combine and Partition.
   The Combine process is optional. A "combiner" is a "reducer" executed individually on each mapper server. It further reduces the data on each mapper in a simplified form. This helps **simplify shuffling** and organization because the volume of data to be organized is reduced.
   The Partition step, on the other hand, translates the key-value pairs produced by the mappers into another set of key-value pairs before sending them to the reducer. This process decides how **the** data should be presented to the reducer and assigns them to a specific reducer.

MapReduce Job – Logical View

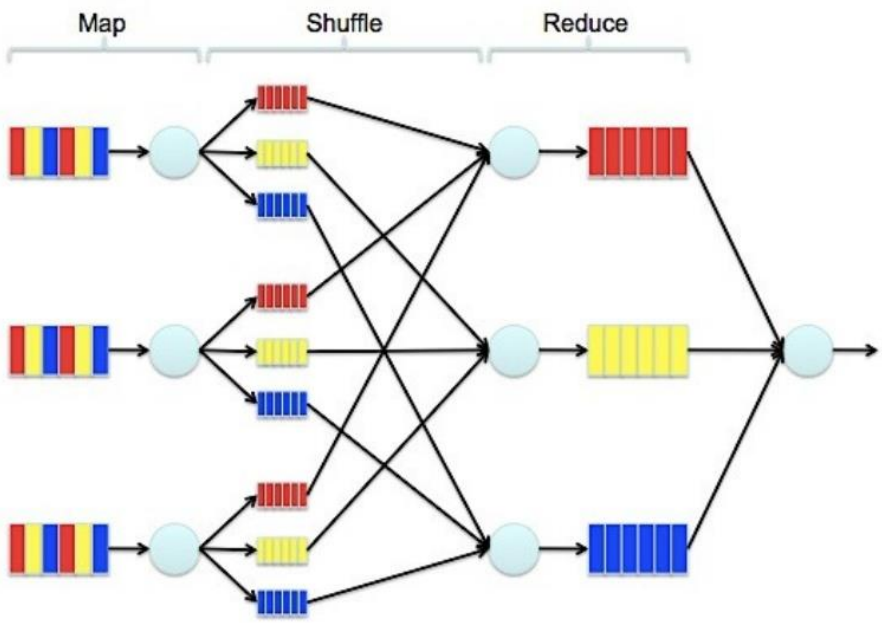


Image from - http://mm-tom.s3.amazonaws.com/blog/MapReduce.png

The `map(func, seq)` Python function applies the function func to all the elements of the sequence seq. It returns a new list with the elements changed by func.

```
def f(x):
   return x * x

rdd = [2, 6, -3, 7]
res = map(f, rdd )
res  # Res is an iterator
```

```
print(*res)
```

```
from operator import mul
rdd1, rdd2 = [2, 6, -3, 7], [1, -4, 5, 3]
res = map(mul, rdd1, rdd2 ) # element wise sum of rdd1 and rdd2
```

```
print(*res)
```

The function `reduce(func, seq)` continually applies the function func() to the sequence seq and return a single value. For example, reduce(f, [1, 2, 3, 4, 5]) calculates f(f(f(f(1,2),3),4),5).

```
from functools import reduce
from operator import add
rdd = list(range(1,6))
reduce(add, rdd) # computes ((((1+2)+3)+4)+5)
```

7.2 Exercise

If the generator of random variable X is discrete with probability mass function $x_1 \mapsto p_1, x_2 \mapsto p_2, \ldots, x_n \mapsto p_n$ then

$$\mathrm{Var}(X) = (\sum_{i=1}^{n} p_i x_i^2) - \mu^2,$$

Where μ is the average value, i.e.

$$\mu = \sum_{i=1}^{n} p_i x_i.$$

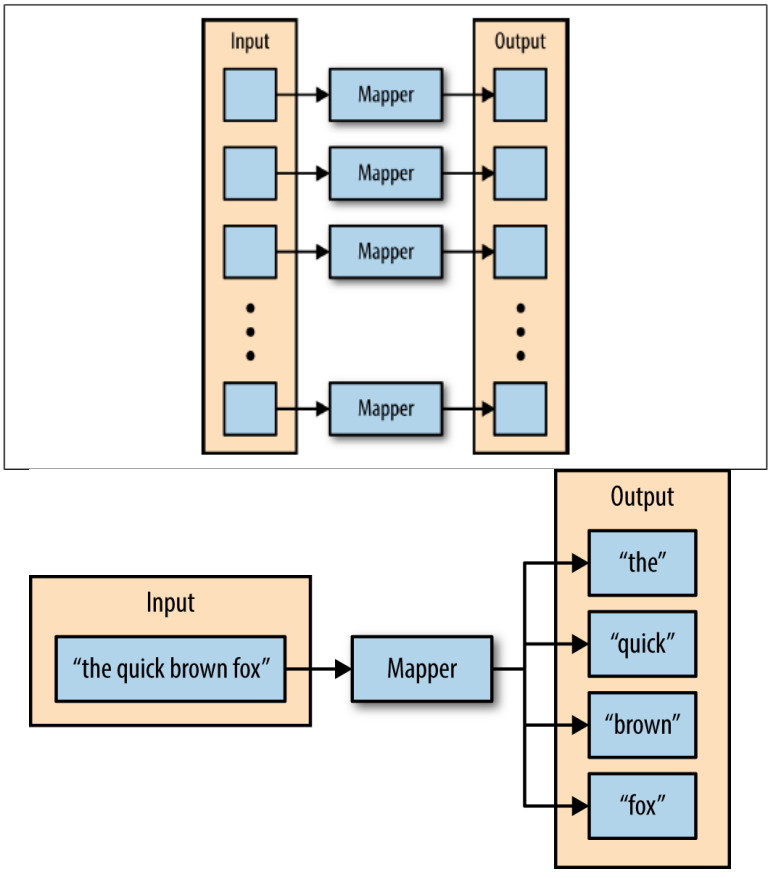
Example:

```
numRows = 100
numSamplesPerRow = 500
x = numpy.random.rand(numRows, numSamplesPerRow)
print(x)
```

1. Write functions to compute the average value and variance.
2. Write functions to compute the average value and variance using `map` and `reduce`.

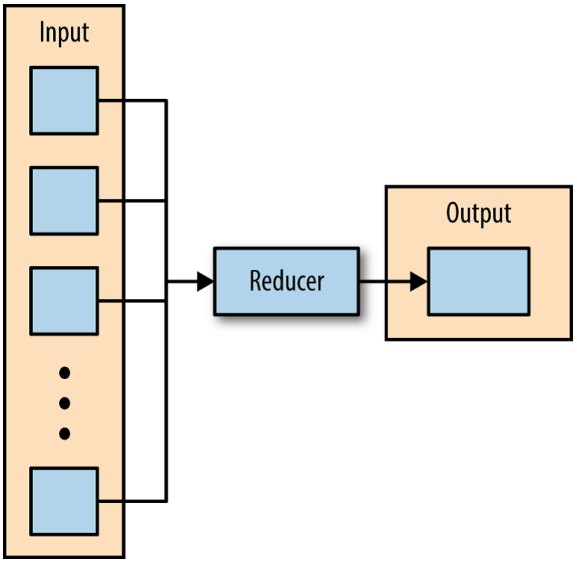
7.3 Writing An Hadoop MapReduce Programs In Python

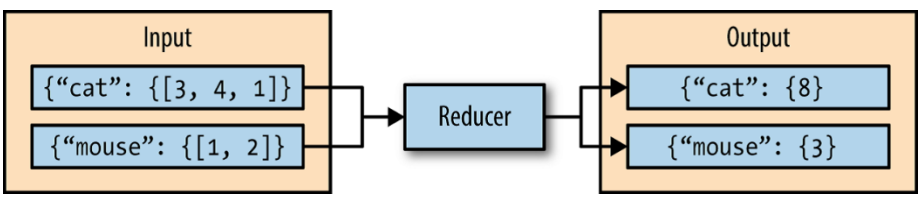
Save the following code in the file `mapper.py`. It will read data from `STDIN`, split it into words and output a list of lines mapping words to their (intermediate) counts to `STDOUT`. The Map script will not compute an (intermediate) sum of a word's occurrences though. Instead, it will output `<word> 1` tuples immediately – even though a specific word might occur multiple times in the input. In our case we let the subsequent Reduce step do the final sum count.



```
#!/usr/bin/env python
"""mapper.py"""

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print('%s\t%s' % (word, 1))
```

Save the following code in the file `reducer.py`. It will read the results of `mapper.py` from `STDIN` (so the output format of `mapper.py` and the expected input format of `reducer.py` must match) and sum the occurrences of each word to a final count, and then output its results to `STDOUT`.






```
#!/usr/bin/env python
```

```
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print('%s\t%s' % (current_word, current_count))
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print('%s\t%s' % (current_word, current_count))
```

I recommend to test your `mapper.py` and `reducer.py` scripts locally before using them in a MapReduce job. Here are some ideas on how to test the functionality of the Map and Reduce scripts.

```
echo foo foo quux labs foo bar quux | python mapper.py | sort | python reducer.py
```

7.4 Running the Python Code on Hadoop plateformes

…

8 Cas d'études

8.1 Case study in agriculture: Suitable crop for suitable soil

Source : **https://www.kaggle.com/code/dhamur/machine-learning-in-agriculture/notebook**

This is dataset which is used to recommend the crop for the suitable soil. This will be very useful in crop production (Agriculture) without looses based on soli ph, rainfall, humidity and other chemical components present in the soil.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import model_selection
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.ensemble import BaggingClassifier
```

```
df = pd.read_csv('D:/ enseignement/poly-ml-ia-bigd-ap/Crop_recommendation.csv')
```

8.1.1 About the data

- Nitrogen is so vital because it is a major component of chlorophyll, the compound by which plants use sunlight energy to produce sugars from water and carbon dioxide (i.e., photosynthesis). It is also a major component of amino acids, the building blocks of proteins. Without proteins, plants wither and die.

- Phosphorus is, therefore, important in cell division and development of new tissue. Phosphorus is also associated with complex energy transformations in the plant. Adding phosphorus to soil low in available phosphorus promotes root growth and winter hardiness, stimulates tillering, and often hastens maturity.

- Potassium is a critical nutrient that plants absorb from the soil, and from fertilizer. It increases disease resistance, helps stalks to grow upright and sturdy, improves drought tolerance, and helps plants get through the winter.

- The average soil temperatures for bioactivity range from 50 to 75F. These values are favorable for normal life functions of earth biota that ensure proper organic matter decomposition, increased nitrogen mineralization, uptake of soluble substances, and metabolism.

- The pH range 5.5–6.5 is optimal for plant growth as the availability of nutrients is optimal.

- Besides disease, rainfall can also determine how fast a crop will grow from seed, including when it will be ready for harvesting. A good balance of rain and proper irrigation can lead to faster-growing plants, which can cut down on germination time and the length between seeding and harvest.

```
df.head()
```

```
print("Shape of the dataframe: ",df.shape)
df.isna().sum()
```

```
df.info()
```

```
df.describe()
```

```
df.dtypes
```

8.1.2 Data distribution

```
sns.displot(x=df['N'], bins=20,kde=True,edgecolor="black",color='black',facecolor='#ffb03b')
plt.title("Nitrogen",size=20)
plt.show()
```

```
sns.displot(x=df['P'],bins=20,color='black',edgecolor='black',kde=True,facecolor='#ffb03b')
plt.title("Phosphorus", size=20)
plt.xticks(range(0,150,20))
plt.show()
```

```
sns.displot(x=df['K'],kde=True, bins=20, facecolor='#ffb03b',edgecolor='black', color='black')
plt.title("Potassium",size=20)
plt.show()
```

```
sns.displot(x=df['temperature'],
bins=20,kde=True,edgecolor="black",color='black',facecolor='#ffb03b')
plt.title("Temperature",size=20)
lt.show()
```

```
sns.displot(x=df['humidity'], color='black',facecolor='#ffb03b',kde=True,edgecolor='black')
plt.title("Humidity",size=20)
plt.show()
```

```
sns.displot(x=df['rainfall'], color='black',facecolor='#ffb03b',kde=True,edgecolor='black')
plt.title("Rainfall",size=20)
plt.show()
```

8.1.3 Categorical plot

```
sns.relplot(x='rainfall',y='temperature',data=df,kind='scatter',hue='label',height=5)
plt.show()
```

```
sns.pairplot(data=df,hue='label')
plt.show()
```

8.1.4 Outerlier detection using graphs

```
# Unique values in the label column
crops = df['label'].unique()
print(len(crops))
print(crops)
print(pd.value_counts(df['label']))
```

```
# Filtering each unique label and store it in a list df2 for to plot the box plot
df2=[]
for i in crops:
    df2.append(df[df['label'] == i])
df2[1].head()
```

```
sns.catplot(data=df, x='label', y='temperature', kind='box', height=10, aspect=20/8.27)
# plt.xticks(rotation='vertical')
plt.title("Temperature", size=20)
plt.show()
```

```
sns.catplot(data=df, x='label', y='humidity', kind='box', height=10, aspect=20/8.27)
# plt.xticks(rotation='vertical')
plt.title("Humidity", size=20)
plt.show()
```

```
sns.catplot(data=df, x='label', y='temperature', kind='box', height=10, aspect=20/8.27)
plt.show()
```

```
sns.catplot(data=df, x='label', y='N', kind='box', height=10, aspect=20/8.27)
# plt.xticks(rotation='vertical')
plt.show()
```

```
sns.catplot(data=df, x='label', y='ph', kind='box', height=10, aspect=20/8.27)
# plt.xticks(rotation='vertical')
plt.title("Nitrogen",size=20)
plt.show()
```

```
sns.catplot(data=df, x='label', y='P', kind='box', height=10, aspect=20/8.27)
# plt.xticks(rotation='vertical')
plt.title("Phosphorus",size=20)
plt.show()
```

```
sns.catplot(data=df, x='label', y='K', kind='box', height=10, aspect=20/8.27)
# plt.xticks(rotation='vertical')
plt.title("Potassium",size=20)
plt.show()
```

These graphs shows that thers is no outliers present in this dataset

8.1.5 Lets check through Mathematics (Statistics)

```
def detect_outlier(x):
    q1 = x.quantile(0.25)
    q3 = x.quantile(0.75)
    IQR = q3-q1
    lower_limit = q1 - (1.5*IQR)
    upper_limit = q3 + (1.5*IQR)
    print(f"Lower limit: {lower_limit} Upper limit: {upper_limit}")
    print(f"Minimum value: {x.min()}   MAximum Value: {x.max()}")
```

```
    for i in [x.min(),x.max()]:
        if i == x.min():
            if lower_limit > x.min():
                print("Lower limit failed - Need to remove minimum value")
            elif lower_limit < x.min():
                print("Lower limit passed - No need to remove outlier")
        elif i == x.max():
            if upper_limit > x.max():
                print("Upper limit passed - No need to remove outlier")
            elif upper_limit < x.max():
                print("Upper limit failed - Need to remove maximum value")
detect_outlier(df['K'][df['label']=='grapes'])
```

```
for i in df['label'].unique():
    detect_outlier(df['K'][df['label']==i])
    print('-----------------------------------------------')
```

These graphs shows that thers is no outliers present in this dataset and it is confirmed with the help of Statistics(IQR)

8.1.6 Prediction

8.1.6.1 Spliting the train and test data

```
x = df.drop(['label'], axis=1)
x.head()
```

```
Y = df['label']
encode = preprocessing.LabelEncoder()
y = encode.fit_transform(Y)
print("Label length: ",len(y))
```

```
x_train,x_test,y_train,y_test = model_selection.train_test_split(x,y)
print(len(x_train),len(y_train),len(x_test),len(y_test))
```

8.1.6.2 Best model choosing

8.1.6.2.1 Decision Tree, Support vector mechanism, Random forest.

```
a={'decision tree' : {
      'model' : DecisionTreeClassifier(criterion='gini'),
      'params':{'decisiontreeclassifier__splitter':['best','random']}
   },
   'svm': {
      'model': SVC(gamma='auto',probability=True),
      'params' : {
         'svc__C': [1,10,100,1000],
         'svc__kernel': ['rbf','linear']
      }
   },
   'random_forest': {
      'model': RandomForestClassifier(),
      'params' : {
         'randomforestclassifier__n_estimators': [1,5,10]
```

```
        }
    },
    'k classifier':{
        'model':KNeighborsClassifier(),

'params':{'kneighborsclassifier__n_neighbors':[5,10,20,25],'kneighborsclassifier__weights':['un
iform','distance']}
    }
}

score=[]
details = []
best_param = {}
for mdl,par in a.items():
    pipe = make_pipeline(preprocessing.StandardScaler(),par['model'])
    res = model_selection.GridSearchCV(pipe,par['params'],cv=5)
    res.fit(x_train,y_train)
    score.append({
        'Model name':mdl,
        'Best score':res.best_score_,
        'Best param':res.best_params_
    })
    details.append(pd.DataFrame(res.cv_results_))
    best_param[mdl]=res.best_estimator_
pd.DataFrame(score)

details[0]

details[1]

details[2]

score

pd.DataFrame(score)

for i in best_param.keys():
    print(f'{i} : {best_param[i].score(x_test,y_test)}')
```

8.1.6.3 *Best model - Random forest*

```
predicted = best_param['random_forest'].predict(x_test)
predicted

plt.figure(figsize=(12,8))
sns.heatmap(confusion_matrix(y_test,predicted),annot=True)
plt.xlabel("Original")
plt.ylabel("Predicted")
plt.show()
```

8.1.6.4 *Bagging classifier for more accuracy*

```
pipe1 = make_pipeline(preprocessing.StandardScaler(),RandomForestClassifier(n_estimators =
10))
bag_model = BaggingClassifier(base_estimator=pipe1,n_estimators=100,
```

```
                    oob_score=True,random_state=0,max_samples=0.8)

bag_model.fit(x_train,y_train)

bag_model.score(x_test,y_test)

predict = bag_model.predict(x_test)

plt.figure(figsize=(12,8))
sns.heatmap(confusion_matrix(y_test,predict),annot=True)
plt.show()
```

8.1.7 Conclusion

Value mapping shows that which value is belongs to which crop. It helps in easy reading the predicted value. Eg: If predicted value id 20 then its belongs to Crop rice. So on...

```
dha2 =pd.DataFrame(Y)
code = pd.DataFrame(dha2['label'].unique())

dha = pd.DataFrame(y)
encode = pd.DataFrame(dha[0].unique())
refer = pd.DataFrame()
refer['code']=code
refer['encode']=encode
refer
```

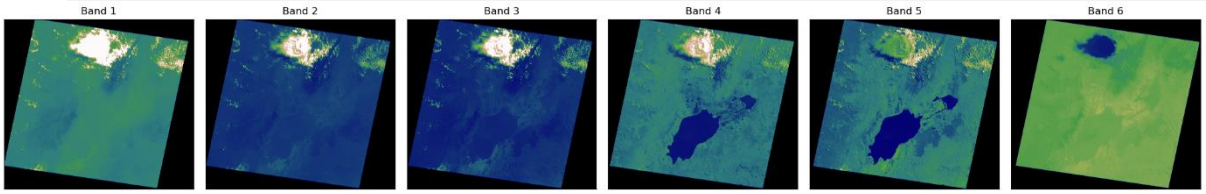
8.2 Images satellites

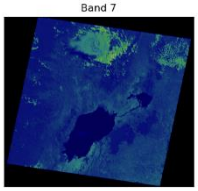
```
#!pip install earthpy
#!pip install spyndex
import earthpy as et
from glob import glob
import earthpy as et
import earthpy.plot as ep
import earthpy.spatial as es
import plotly.express as px
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sn
import pandas as pd
from pathlib import Path
sn.axes_style('whitegrid');
from matplotlib.colors import ListedColormap
from scipy.io import loadmat
import rasterio as rio
from rasterio import plot
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
import statistics
import spyndex
import time
from sklearn.decomposition import PCA
S_sentinel_bands = glob("D:/ylebbah/data/TP-AP-oct-2023/big/poly-liege/*.TIF")
S_sentinel_bands
```

```
['D:/ylebbah/data/TP-AP-oct-2023/big/poly-
liege\\LT05_L1TP_173060_19870807_20170211_01_T1_B1.TIF',
 'D:/ylebbah/data/TP-AP-oct-2023/big/poly-
liege\\LT05_L1TP_173060_19870807_20170211_01_T1_B2.TIF',
 'D:/ylebbah/data/TP-AP-oct-2023/big/poly-
liege\\LT05_L1TP_173060_19870807_20170211_01_T1_B3.TIF',
 'D:/ylebbah/data/TP-AP-oct-2023/big/poly-
liege\\LT05_L1TP_173060_19870807_20170211_01_T1_B4.TIF',
 'D:/ylebbah/data/TP-AP-oct-2023/big/poly-
liege\\LT05_L1TP_173060_19870807_20170211_01_T1_B5.TIF',
 'D:/ylebbah/data/TP-AP-oct-2023/big/poly-
liege\\LT05_L1TP_173060_19870807_20170211_01_T1_B6.TIF',
 'D:/ylebbah/data/TP-AP-oct-2023/big/poly-
liege\\LT05_L1TP_173060_19870807_20170211_01_T1_B7.TIF']
```

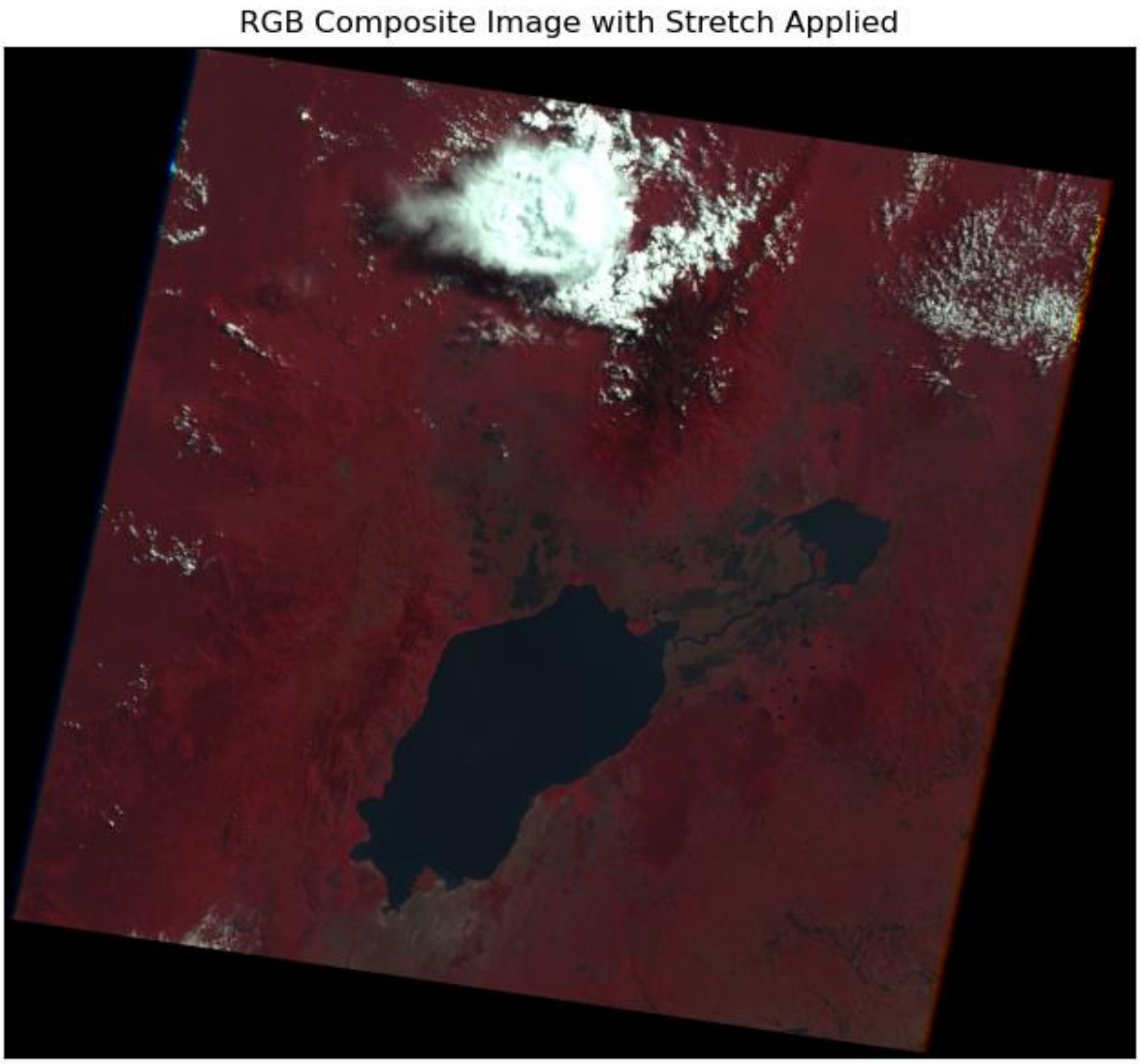
```
l = []
for i in S_sentinel_bands:
  with rio.open(i, 'r') as f:
    l.append(f.read(1))
arr_st = np.stack(l)
print(f'Height: {arr_st.shape[1]}\nWidth: {arr_st.shape[2]}\nBands: {arr_st.shape[0]}')
Height: 6921
Width: 7741
Bands: 7
ep.plot_bands(arr_st, cmap = 'gist_earth', figsize = (20, 12), cols = 6, cbar = False)
plt.show()
```



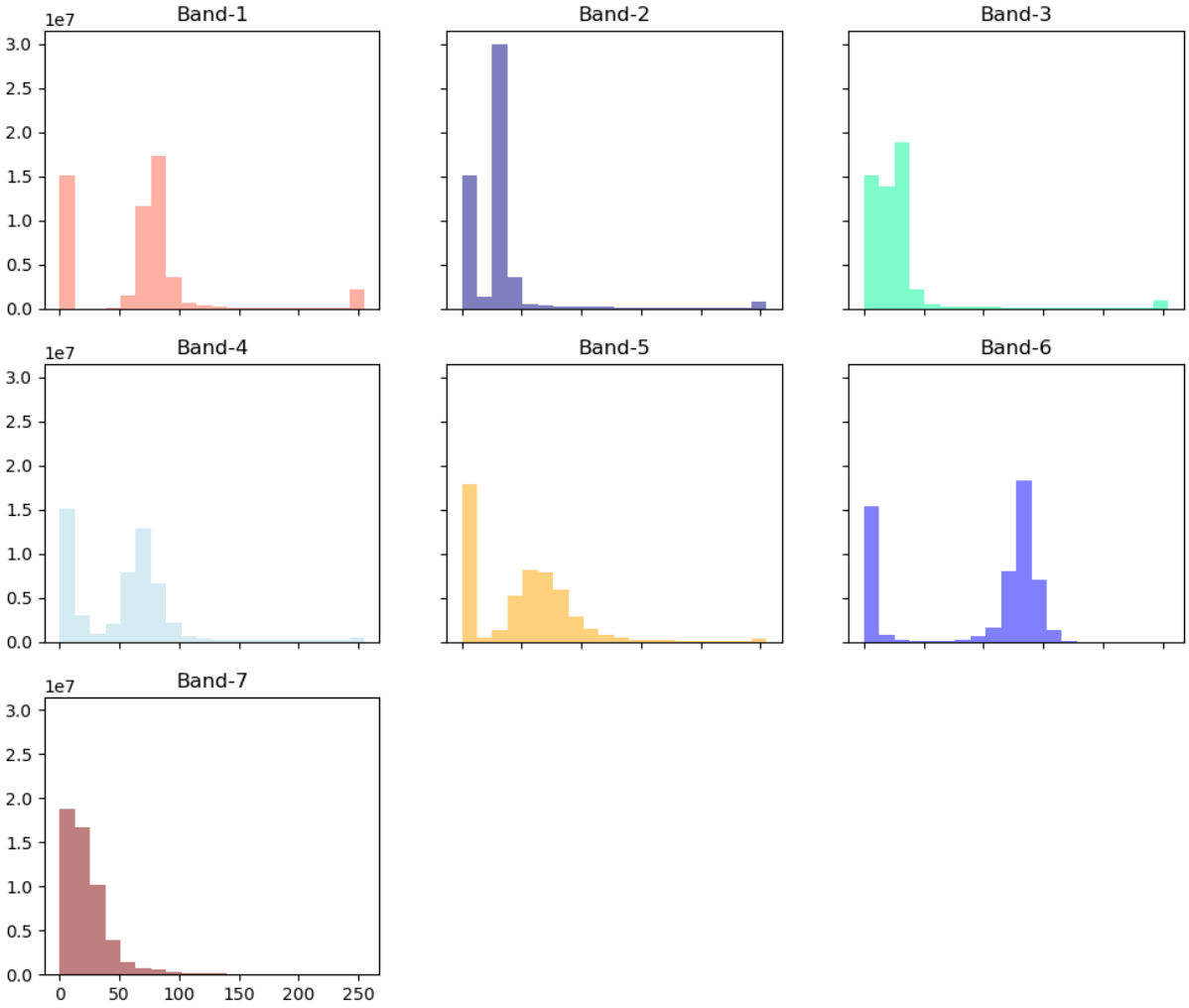



```
ep.plot_rgb(
   arr_st,
   rgb=(3, 2, 1),
```

```
    stretch=True,
    str_clip=0.0,
    figsize=(8, 10),
    title="RGB Composite Image with Stretch Applied",
)
plt.show()
```




```
colors = ['tomato', 'navy', 'MediumSpringGreen', 'lightblue', 'orange', 'blue',
          'maroon']

ep.hist(arr_st,
        colors = colors,
        title=[f'Band-{i}' for i in range(1, 8)],
        cols=3,
        alpha=0.5,
        figsize = (12, 10)
        )

plt.show()
```



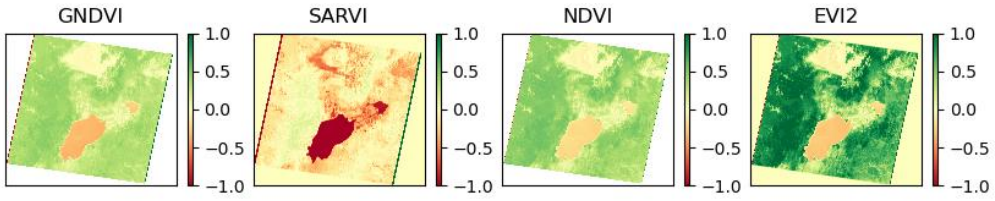

```
## https://github.com/awesome-spectral-indices/awesome-spectral-indices#expressions
## https://github.com/awesome-spectral-indices/spyndex
## All indices of the library
## spyndex.indices

# All bands
bands = spyndex.bands
print(bands)
constants = spyndex.constants
print(constants)

np.seterr(invalid='ignore') #to ignore divide by zero exeption message

parameters = {"B": arr_st[0].astype(float),
            "G": arr_st[1].astype(float),
            "R": arr_st[2].astype(float),
            "N": arr_st[3].astype(float),
            "S1": arr_st[4].astype(float),
            "T": arr_st[5].astype(float),
            "S2": arr_st[6].astype(float),
            #Constants
            "L" : 0.5,
            "g" : 2.5,
```

```
        "C1": 6,
        "C2": 7.5,
        #"beta":5.5
        # Parameters required for kNDVI
        "kNN" : 1.0,
        "kNR" : spyndex.computeKernel(
        kernel = "RBF",
        params = {
           "a": arr_st[3].astype(float),
           "b": arr_st[2].astype(float),
           "sigma": 0.5 * (arr_st[3].astype(float) + arr_st[2].astype(float))
        }),
       }
```

```
['A', 'B', 'G', 'G1', 'N', 'N2', 'R', 'RE1', 'RE2', 'RE3', 'S1', 'S2', 'T',
'T1', 'T2', 'WV', 'Y']
['C1', 'C2', 'L', 'PAR', 'alpha', 'beta', 'c', 'cexp', 'epsilon', 'fdelta',
'g', 'gamma', 'k', 'lambdaG', 'lambdaN', 'lambdaR', 'nexp', 'omega', 'p',
'sigma', 'sla', 'slb']
```

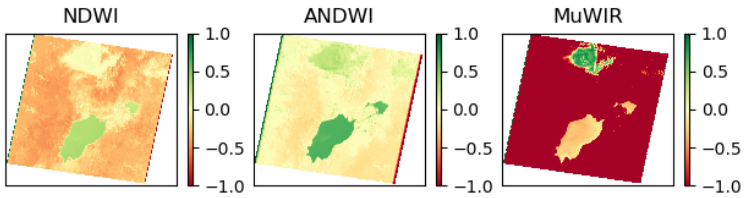
```
IdxList = []
#### 1- Vegetation ####
print('\n---------- Vegetation indices ----------\n')
# print(spyndex.indices["NDVI"])

# GNDVI: Green Normalized Difference Vegetation Index.
# GOSAVI: Green Optimized Soil Adjusted Vegetation Index.
# NDMI: Normalized Difference Moisture Index.
# SARVI: Soil Adjusted and Atmospherically Resistant Vegetation Index.
# SEVI: Shadow-Eliminated Vegetation Index.
# ARI2: Anthocyanin Reflectance Index 2.
# AVI: Advanced Vegetation Index.
# BNDVI: Blue Normalized Difference Vegetation Index.
# NDVI: Normalized Difference Vegetation Index.
# CIRE: Chlorophyll Index Red Edge.
# ATSAVI: Adjusted Transformed Soil-Adjusted Vegetation Index.
# CVI: Chlorophyll Vegetation Index.
# DSI: Drought Stress Index.
# DSWI5: Disease-Water Stress Index 5.
# EVI2: Two-Band Enhanced Vegetation Index.
# GDVI: Generalized Difference Vegetation Index.
# GM2: Gitelson and Merzlyak Index 2.
# NMDI: Normalized Multi-band Drought Index.
# SI: Shadow Index.
VegetationIndices = ["GNDVI","SARVI","NDVI","EVI2"] # "NDMI",
VIdx = spyndex.computeIndex(VegetationIndices, parameters)
print(f'Height: {VIdx.shape[1]}\nWidth: {VIdx.shape[2]}\nSpectral Indices: {VIdx.shape[0]}\n')

ep.plot_bands(VIdx, cmap="RdYlGn", cols=5, vmin=-1, vmax=1, figsize=(10, 14),
title=VegetationIndices)
plt.show()

## Adding computed indices to the global list
for elt in VIdx:
```

```
        IdxList.append(elt)
---------- Vegetation indices ----------------------

Height: 6921
Width: 7741
Spectral Indices: 4
```



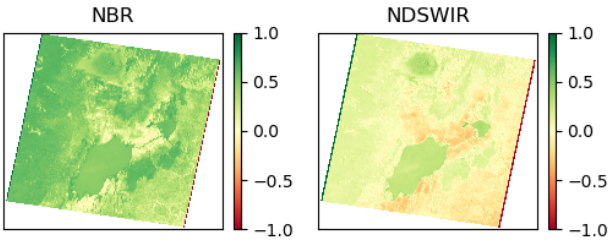

```
#### 2- Water ####
print('\n---------- Water indices ----------\n')

# NDWI: Normalized Difference Water Index.
```

```
    # ANDWI: Augmented Normalized Difference Water Index.
    # LSWI: Land Surface Water Index.
    # MBWI: Multi-Band Water Index.
    # MuWIR: Revised Multi-Spectral Water Index.
    # NDCI: Normalized Difference Chlorophyll Index.
    # NDTI: Normalized Difference Turbidity Index.
    # S2WI: Sentinel-2 Water Index.

    WaterIndices = ["NDWI","ANDWI","MuWIR"] # "LSWI", "NDTI"
    WatIdx = spyndex.computeIndex(WaterIndices, parameters)
    print(f'Height: {WatIdx.shape[1]}\nWidth: {WatIdx.shape[2]}\nSpectral Indices:
    {WatIdx.shape[0]}\n')

    ep.plot_bands(WatIdx, cmap="RdYlGn", cols=5, vmin=-1, vmax=1, figsize=(10, 14),
    title=WaterIndices)
    plt.show()

    ## Adding computed indices to the global list
    for elt in WatIdx:
      IdxList.append(elt)
---------- Water indices ----------------------

Height: 6921
Width: 7741
Spectral Indices: 3
```




```
#### 3- Burn ####
print('\n---------- Burn indices ----------\n')

# BAI: Burned Area Index.
```

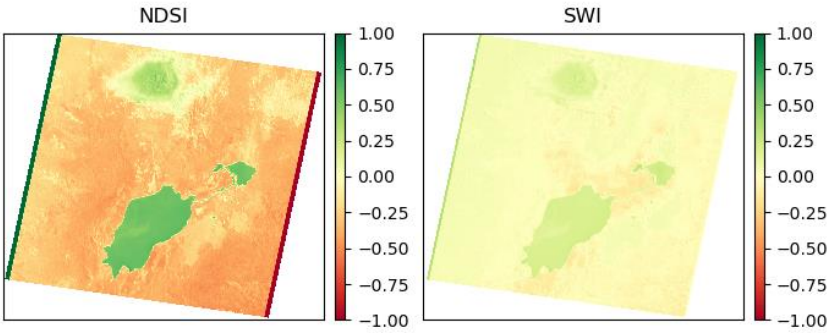
```
    # BAIS2: Burned Area Index for Sentinel 2.
    # NBR: Normalized Burn Ratio.
    # NDSWIR: Normalized Difference SWIR.
    BurnIndices = ["NBR","NDSWIR"] # "BAI","BAIS2",
    BuIdx = spyndex.computeIndex(BurnIndices, parameters)
    print(f'Height: {BuIdx.shape[1]}\nWidth: {BuIdx.shape[2]}\nSpectral Indices:
    {BuIdx.shape[0]}\n')

    ep.plot_bands(BuIdx, cmap="RdYlGn", cols=4, vmin=-1, vmax=1, figsize=(10, 14),
    title=BurnIndices)
    plt.show()

    ## Adding computed indices to the global list
    for elt in BuIdx:
       IdxList.append(elt)
---------- Burn indices ----------------------

Height: 6921
Width: 7741
Spectral Indices: 2
```




```
#### 4- Snow ####
print('\n---------- Snow indices ----------\n')

# NDSI: Normalized Difference Snow Index.
```

```
    # NDSInw: Normalized Difference Snow Index with no Water.
    # SWI: Snow Water Index.
    SnowIndices = ["NDSI","SWI"]
    SnIdx = spyndex.computeIndex(SnowIndices, parameters)
    print(f'Height: {SnIdx.shape[1]}\nWidth: {SnIdx.shape[2]}\nSpectral Indices:
    {SnIdx.shape[0]}\n')

    ep.plot_bands(SnIdx, cmap="RdYlGn", cols=3, vmin=-1, vmax=1, figsize=(10, 14),
    title=SnowIndices)
    plt.show()

    ## Adding computed indices to the global list
    for elt in SnIdx:
      IdxList.append(elt)
```

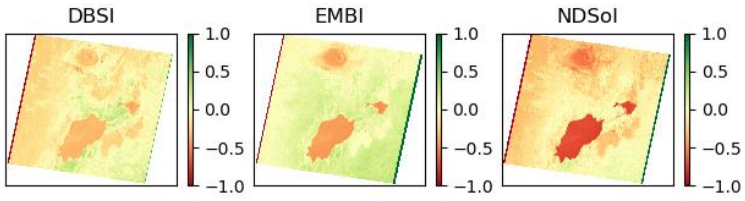
```
---------- Snow indices ----------------------

Height: 6921
Width: 7741
Spectral Indices: 2
```




```
#### 5- Soil ####
print('\n---------- Soil indices ----------\n')

# BI: Bare Soil Index.
```

```
    # DBSI: Dry Bareness Index.
    # EMBI: Enhanced Modified Bare Soil Index.
    # NDSoI: Normalized Difference Soil Index.
    # NSDS: Normalized Shortwave Infrared Difference Soil-Moisture.
    SoilIndices = ["DBSI","EMBI","NDSoI"] # "BI", ,"NSDS"
    SoilIdx = spyndex.computeIndex(SoilIndices, parameters)
    print(f'Height: {SoilIdx.shape[1]}\nWidth: {SoilIdx.shape[2]}\nSpectral Indices:
    {SoilIdx.shape[0]}\n')

    ep.plot_bands(SoilIdx, cmap="RdYlGn", cols=5, vmin=-1, vmax=1, figsize=(10, 14),
    title=SoilIndices)
    plt.show()

    ## Adding computed indices to the global list
    for elt in SoilIdx:
      IdxList.append(elt)
```

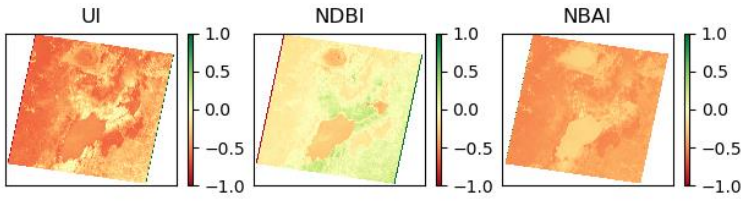
```
---------- Soil indices ----------------------

Height: 6921
Width: 7741
Spectral Indices: 3
```




```
#### 6- Urban ####
print('\n---------- Urban indices ----------\n')

# UI: Urban Index.
```

```
# BLFEI: Built-Up Land Features Extraction Index.
# NDBI: Normalized Difference Built-Up Index.
# VIBI: Vegetation Index Built-up Index.
# NBAI: Normalized Built-up Area Index.
UrbanIndices = ["UI","NDBI","NBAI"] # "VIBI", "BLFEI",
np.seterr(divide='ignore', invalid='ignore') #to ignore divide by zero exeption message
UrbIdx = spyndex.computeIndex(UrbanIndices, parameters)
print(f'Height: {UrbIdx.shape[1]}\nWidth: {UrbIdx.shape[2]}\nSpectral Indices:
{UrbIdx.shape[0]}\n')

ep.plot_bands(UrbIdx, cmap="RdYlGn", cols=5, vmin=-1, vmax=1, figsize=(10, 14),
title=UrbanIndices)
plt.show()

## Adding computed indices to the global list
for elt in UrbIdx:
    IdxList.append(elt)
```

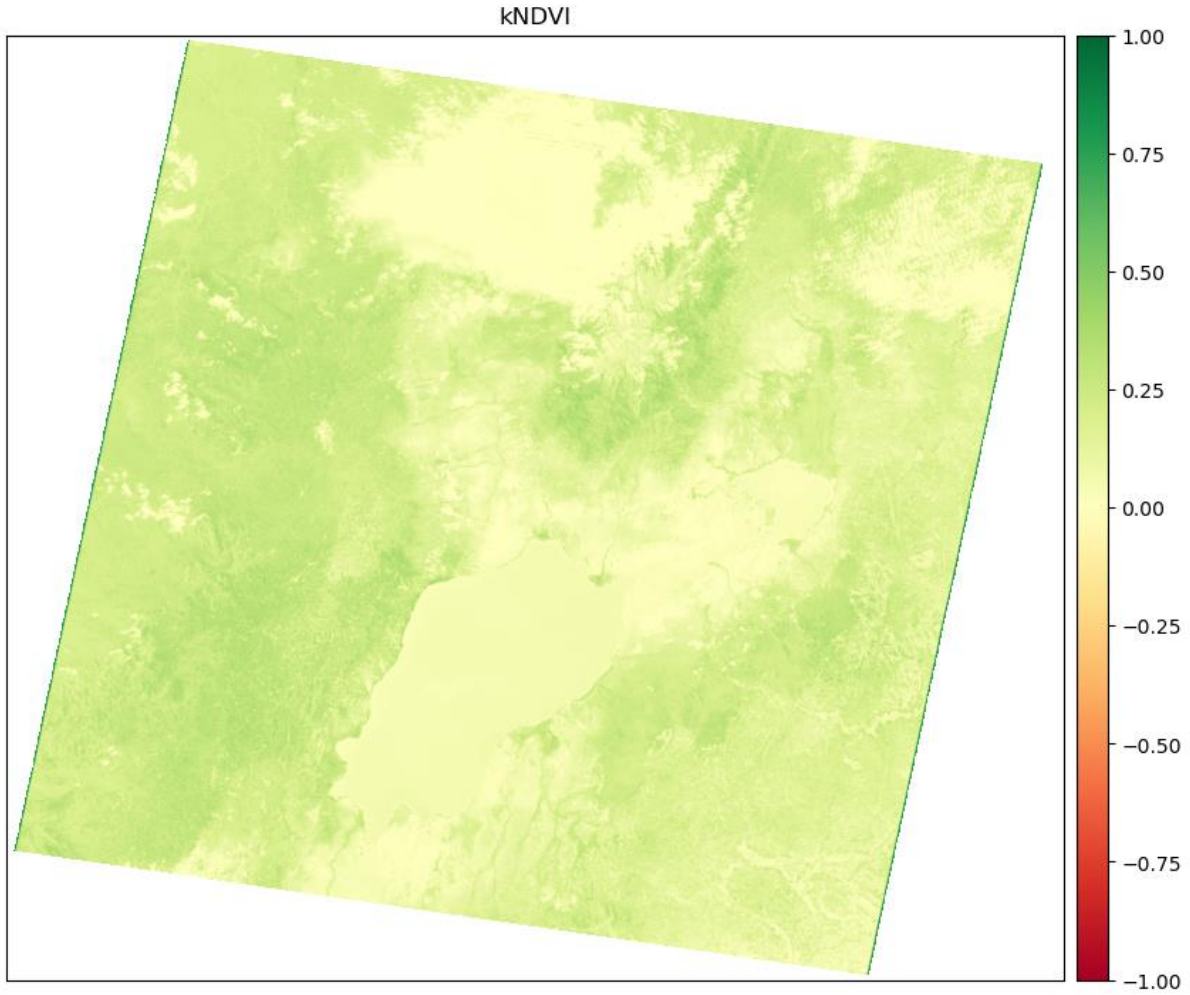
```
---------- Urban indices ----------------------

Height: 6921
Width: 7741
Spectral Indices: 3
```




```
print('\n---------- Kernel indices ----------\n')

# kEVI: Kernel Enhanced Vegetation Index.
# kNDVI: Kernel Normalized Difference Vegetation Index.
```

```
# kVARI: Kernel Visible Atmospherically Resistant Index.
np.seterr(divide='ignore', invalid='ignore') #to ignore divide by zero exeption message

KernelIndices = ["kNDVI"]
KerIdx = spyndex.computeIndex(KernelIndices, parameters)
KerIdx = KerIdx.reshape((1, KerIdx.shape[0], KerIdx.shape[1]))
#print(f'Height: {KerIdx.shape[1]}\nWidth: {KerIdx.shape[2]}\nSpectral Indices:
{KerIdx.shape[0]}\n')
print(KerIdx.shape)

ep.plot_bands(KerIdx, cmap="RdYlGn", cols=1, vmin=-1, vmax=1, figsize=(10, 14),
title=KernelIndices)
plt.show()
```

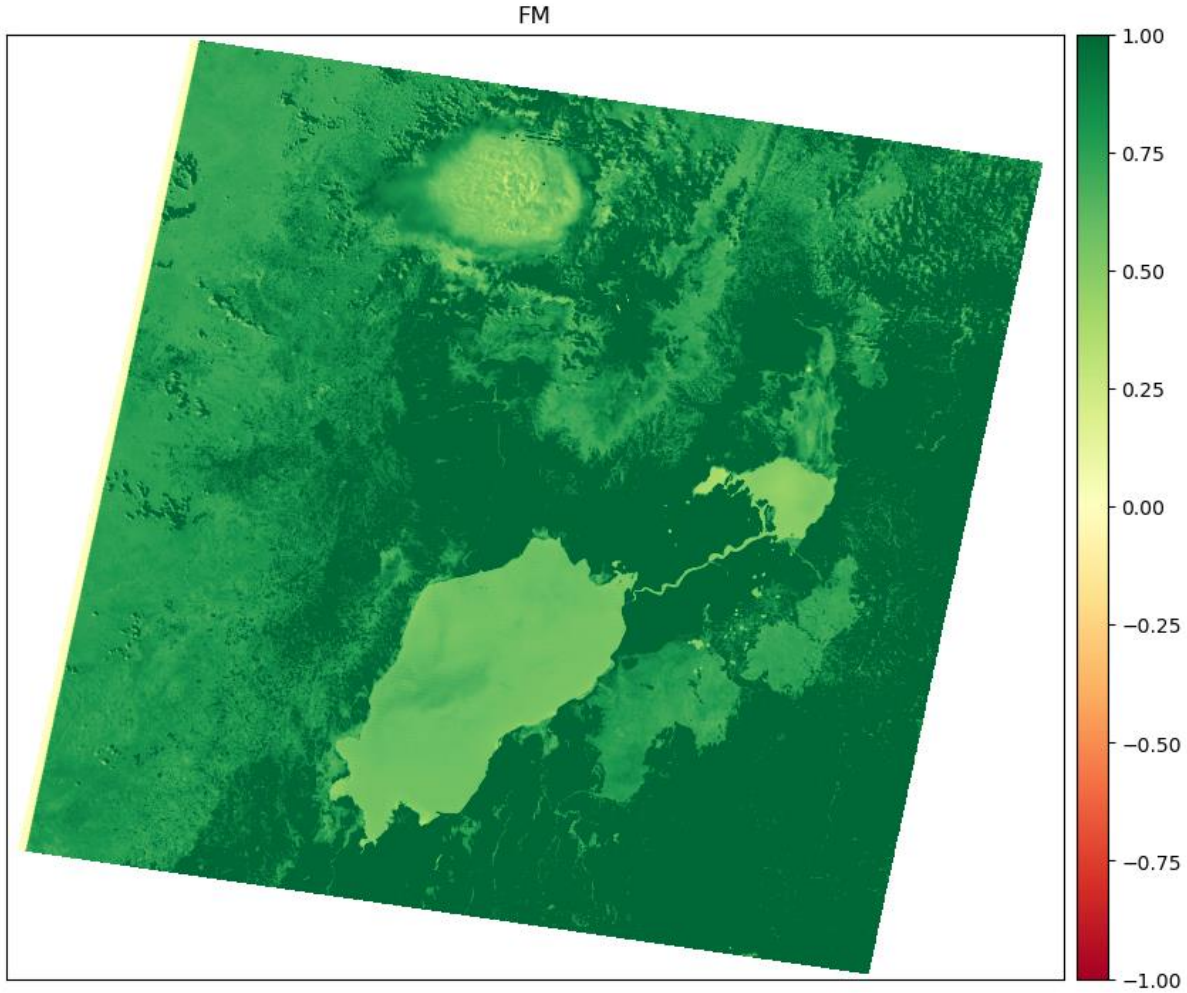
```
---------- Kernel indices ----------------------

(1, 6921, 7741)
```




```
## Adding computed indices to the global list
for elt in KerIdx:
   IdxList.append(elt)

## see if that works or use the formula without the labrary
#### Geology Indices ####
print('\n---------- Geology indices ----------------------\n')
```

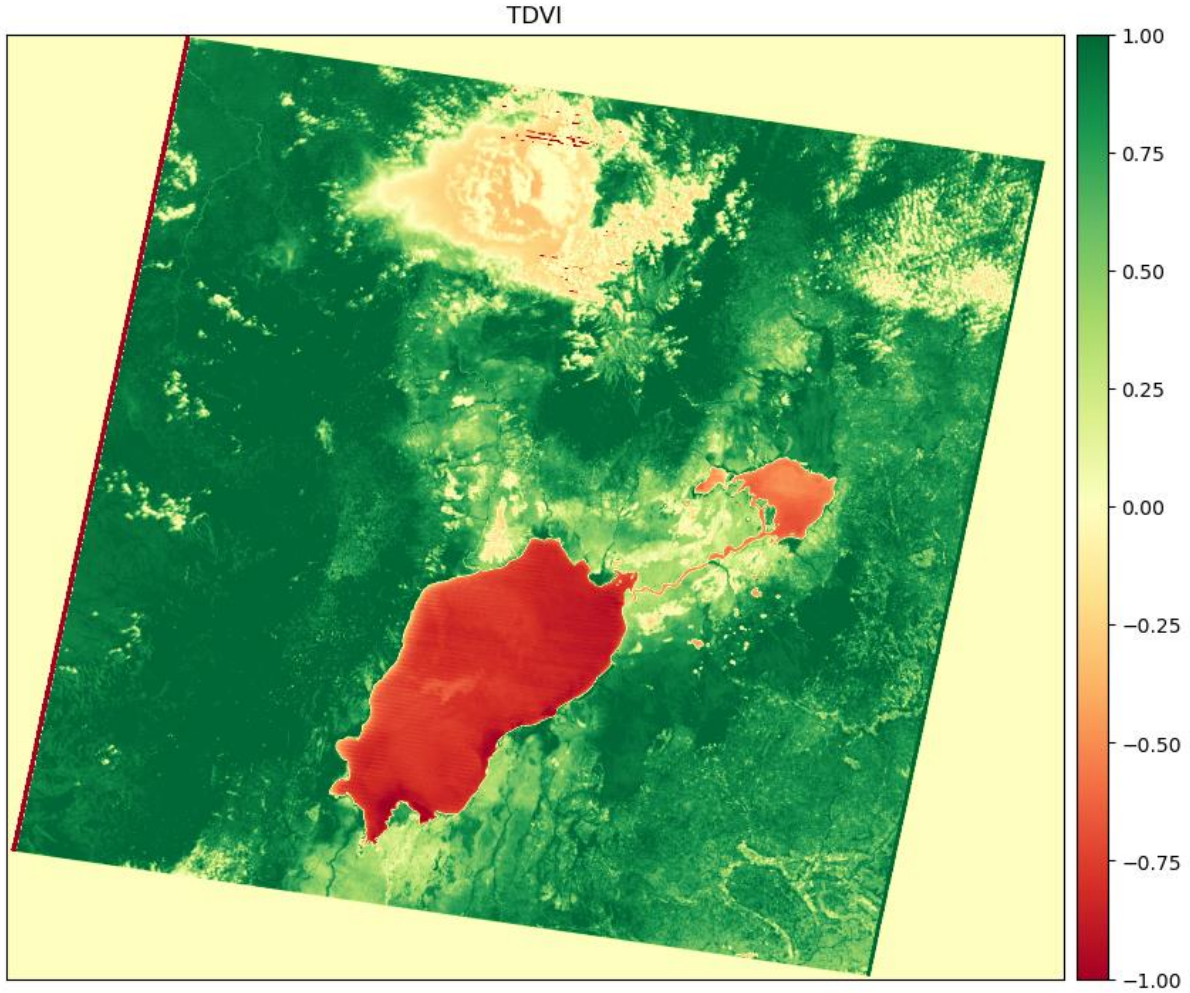
```
# Clay Minerals (CM) : Clay Minerals Ratio =  SWIR1÷SWIR2
# Ferrous Minerals (FM) :  Ferrous Minerals Ratio  =SWIR1÷NIR

GeologyIndices = ["FM"] # "CM",
# GeoIdx = np.zeros((2, arr_st.shape[1], arr_st.shape[2]))
GeoIdx = np.zeros((1, arr_st.shape[1], arr_st.shape[2]))


# GeoIdx[0] = arr_st[9].astype(float) / arr_st[10].astype(float)
# GeoIdx[1] = arr_st[9].astype(float) / arr_st[7].astype(float)
GeoIdx = arr_st[4].astype(float) / arr_st[3].astype(float)

GeoIdx = GeoIdx.reshape((1, GeoIdx.shape[0], GeoIdx.shape[1]))

print(GeoIdx.shape)

ep.plot_bands(GeoIdx, cmap="RdYlGn", cols=1, vmin=-1, vmax=1, figsize=(10, 14),
title=GeologyIndices)
plt.show()

## Adding computed indices to the global list
for elt in GeoIdx:
   IdxList.append(elt)


#### Indices pregnant en compte les temperatures des surfaces: ####
print('\n---------- Temperature indices ----------\n')

# Temperature Index (TI)
# TDVI

np.seterr(divide='ignore', invalid='ignore') #to ignore divide by zero exeption message

TemperatureIndices = ["TDVI"]
TemIdx = spyndex.computeIndex(TemperatureIndices, parameters)
TemIdx = TemIdx.reshape((1, TemIdx.shape[0], TemIdx.shape[1]))

print(f'Height: {TemIdx.shape[1]}\nWidth: {TemIdx.shape[2]}\nSpectral Indices:
{TemIdx.shape[0]}\n')

ep.plot_bands(TemIdx, cmap="RdYlGn", cols=1, vmin=-1, vmax=1, figsize=(10, 14),
title=TemperatureIndices)
plt.show()
## Adding computed indices to the global list
for elt in TemIdx:
   IdxList.append(elt)
```

```
---------- Geology indices ----------------------

(1, 6921, 7741)
```



---------- Temperature indices ----------

Height: 6921
Width: 7741
Spectral Indices: 1




```
print('\n---------- PUTTING ALL THE INDICES TOGETHER (STACKING) ----------\n')
stackedIndx = np.stack(IdxList)
print(f'Height: {stackedIndx.shape[1]}\nWidth: {stackedIndx.shape[2]}\nSpectral Indices: {stackedIndx.shape[0]}\n')
allTitles = VegetationIndices + WaterIndices + BurnIndices + SnowIndices + SoilIndices + UrbanIndices + KernelIndices + GeologyIndices + TemperatureIndices
# allTitles = VegetationIndices
print(len(allTitles))
ep.plot_bands(stackedIndx, cmap="RdYlGn", cols=5, vmin=-1, vmax=1, figsize=(10, 13), title=allTitles)
plt.show()
```

```
---------- PUTTING ALL THE INDICES TOGETHER (STACKING) ----------
-----------

---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_25664\686072775.py in <module>
      1 print('\n---------- PUTTING ALL THE INDICES TOGETHER (STACKING) ---
----------\n')
----> 2 stackedIndx = np.stack(IdxList)
      3 print(f'Height: {stackedIndx.shape[1]}\nWidth:
{stackedIndx.shape[2]}\nSpectral Indices: {stackedIndx.shape[0]}\n')
      4 allTitles = VegetationIndices + WaterIndices + BurnIndices +
SnowIndices + SoilIndices + UrbanIndices + KernelIndices + GeologyIndices +
TemperatureIndices
      5 # allTitles = VegetationIndices
```

```
~\anaconda3\lib\site-packages\numpy\core\overrides.py in stack(*args,
**kwargs)

~\anaconda3\lib\site-packages\numpy\core\shape_base.py in stack(arrays,
axis, out, dtype, casting)
    469     sl = (slice(None),) * axis + (_nx.newaxis,)
    470     expanded_arrays = [arr[sl] for arr in arrays]
--> 471     return _nx.concatenate(expanded_arrays, axis=axis, out=out,
    472                            dtype=dtype, casting=casting)
    473

~\anaconda3\lib\site-packages\numpy\core\overrides.py in concatenate(*args,
**kwargs)

MemoryError: Unable to allocate 7.98 GiB for an array with shape (20, 6921,
7741) and data type float64
spyndex.indices
SpectralIndices(['AFRI1600', 'AFRI2100', 'ANDWI', 'ARI', 'ARI2', 'ARVI',
'ATSAVI', 'AVI', 'AWEInsh', 'AWEIsh', 'BAI', 'BAIM', 'BAIS2', 'BCC', 'BI',
'BITM', 'BIXS', 'BLFEI', 'BNDVI', 'BRBA', 'BWDRVI', 'BaI', 'CCI', 'CIG',
'CIRE', 'CSI', 'CSIT', 'CVI', 'DBI', 'DBSI', 'DPDD', 'DSI', 'DSWI1',
'DSWI2', 'DSWI3', 'DSWI4', 'DSWI5', 'DVI', 'DVIplus', 'DpRVIHH', 'DpRVIVV',
'EBBI', 'EBI', 'EMBI', 'EVI', 'EVI2', 'ExG', 'ExGR', 'ExR', 'FCVI', 'GARI',
'GBNDVI', 'GCC', 'GDVI', 'GEMI', 'GLI', 'GM1', 'GM2', 'GNDVI', 'GOSAVI',
'GRNDVI', 'GRVI', 'GSAVI', 'GVMI', 'IAVI', 'IBI', 'IKAW', 'IPVI', 'IRECI',
'LSWI', 'MBI', 'MBWI', 'MCARI', 'MCARI1', 'MCARI2', 'MCARI705',
'MCARIOSAVI', 'MCARIOSAVI705', 'MGRVI', 'MIRBI', 'MLSWI26', 'MLSWI27',
'MNDVI', 'MNDWI', 'MNLI', 'MRBVI', 'MSAVI', 'MSI', 'MSR', 'MSR705', 'MTCI',
'MTVI1', 'MTVI2', 'MuWIR', 'NBAI', 'NBLI', 'NBLIOLI', 'NBR', 'NBR2',
'NBRSWIR', 'NBRT1', 'NBRT2', 'NBRT3', 'NBRplus', 'NBSIMS', 'NBUI', 'ND705',
'NDBI', 'NDBaI', 'NDCI', 'NDDI', 'NDGI', 'NDGlaI', 'NDII', 'NDISIb',
'NDISIg', 'NDISImndwi', 'NDISIndwi', 'NDISIr', 'NDMI', 'NDPI', 'NDPolI',
'NDPonI', 'NDREI', 'NDSI', 'NDSII', 'NDSIWV', 'NDSInw', 'NDSWIR', 'NDSaII',
'NDSoI', 'NDTI', 'NDVI', 'NDVI705', 'NDVIMNDWI', 'NDVIT', 'NDWI', 'NDWIns',
'NDYI', 'NGRDI', 'NHFD', 'NIRv', 'NIRvH2', 'NIRvP', 'NLI', 'NMDI', 'NRFIg',
'NRFIr', 'NSDS', 'NSDSI1', 'NSDSI2', 'NSDSI3', 'NSTv1', 'NSTv2', 'NWI',
'NormG', 'NormNIR', 'NormR', 'OCVI', 'OSAVI', 'PISI', 'PSRI', 'QpRVI',
'RCC', 'RDVI', 'REDSI', 'RENDVI', 'RFDI', 'RGBVI', 'RGRI', 'RI', 'RI4XS',
'RVI', 'S2REP', 'S2WI', 'S3', 'SARVI', 'SAVI', 'SAVI2', 'SAVIT', 'SEVI',
'SI', 'SIPI', 'SLAVI', 'SR', 'SR2', 'SR3', 'SR555', 'SR705', 'SWI', 'SWM',
'SeLI', 'TCARI', 'TCARIOSAVI', 'TCARIOSAVI705', 'TCI', 'TDVI', 'TGI',
'TRRVI', 'TSAVI', 'TTVI', 'TVI', 'TWI', 'TriVI', 'UI', 'VARI', 'VARI700',
'VDDPI', 'VHVVD', 'VHVVP', 'VHVVR', 'VI6T', 'VI700', 'VIBI', 'VIG',
'VVVHD', 'VVVHR', 'VVVHS', 'VgNIRBI', 'VrNIRBI', 'WDRVI', 'WDVI', 'WI1',
'WI2', 'WI2015', 'WRI', 'kEVI', 'kIPVI', 'kNDVI', 'kRVI', 'kVARI',
'mND705', 'mSR705'])
# Extracting Pixels
def extract_pixels(X, cols):

    """
    A function to extract pixels from a Hyperspectral image Saves a CSV
file containing the pixels values

    Parameters
    ----------
    X : 3D Array
        Data
    y : 2D Array
        Ground Truth / Classes
```

```
    Returns
    -------
    A 2-Dimensional DataFrame containing 21025 pixels along with their
assigned class


    """
    data = X.reshape(-1, X.shape[2])
    df = pd.DataFrame(data = data)
    #df = pd.concat([df, pd.DataFrame(data = y.ravel())], axis=1)
    # df.columns= [f'Band{i}' for i in range(1, 1+X.shape[2])]+['Class']
    df.columns= cols
    # df.to_csv('Pixels_Dataset.csv', index=False)
    return df
cols = VegetationIndices + WaterIndices + BurnIndices + SnowIndices +
SoilIndices + UrbanIndices + KernelIndices + GeologyIndices +
TemperatureIndices
# cols = VegetationIndices
print(cols)
x = np.moveaxis(stackedIndx, 0, -1)
x.shape
['GNDVI', 'SARVI', 'NDVI', 'EVI2', 'NDWI', 'ANDWI', 'MuWIR', 'NBR',
'NDSWIR', 'NDSI', 'SWI', 'DBSI', 'EMBI', 'NDSoI', 'UI', 'NDBI', 'NBAI',
'kNDVI', 'FM', 'TDVI']
(6921, 7741, 20)
#Extracting Pixels:
x = np.nan_to_num(x) # eliminating NaN values
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_25664\16660857.py in <module>
      1 #Extracting Pixels:
----> 2 x = np.nan_to_num(x) # eliminating NaN values

~\anaconda3\lib\site-packages\numpy\core\overrides.py in nan_to_num(*args,
**kwargs)

~\anaconda3\lib\site-packages\numpy\lib\type_check.py in nan_to_num(x,
copy, nan, posinf, neginf)
    496     array([222222.+111111.j, 111111.     +0.j, 111111.+222222.j])
    497     """
--> 498     x = _nx.array(x, subok=True, copy=copy)
    499     xtype = x.dtype.type
    500

MemoryError: Unable to allocate 7.98 GiB for an array with shape (6921,
7741, 20) and data type float64
# Convert the input data into a tabular data (lines = pixels, columns =
spectral indices)
pixels = extract_pixels(x, cols)
x.shape
(6921, 7741, 20)
pixels
```

	GNDVI	SARVI	NDVI	EVI2	NDWI	ANDWI	MuWIR	NBR	NDSWIR	NDSI	SWI	DBSI	EMBI	NDSoI	UI	NDBI	NBAI	kNDVI	FM	TDVI
0	NaN	0.0	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
1	NaN	0.0	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
2	NaN	0.0	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
3	NaN	0.0	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
4	NaN	0.0	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
53575456	NaN	0.0	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
53575457	NaN	0.0	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
53575458	NaN	0.0	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
53575459	NaN	0.0	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
53575460	NaN	0.0	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0

53575461 rows × 20 columns

8.3 Plant Disease Resnet50

https://www.kaggle.com/code/aryanml007/plant-disease-resnet50/notebook

```
#!pip install tensorflow
#!pip install opencv-python

#IMPORTING LIBRARIES
import numpy as np
```

```
import pandas as pd
import os
import matplotlib.pyplot as plt
path='D:/ylebbah/data/plant-kaggle/PlantVillage-Dataset/data_distribution_for_SVM/train'
plt.figure(figsize=(70,70))
count=0
plant_names=[]
total_images=0
for i in os.listdir(path):
 count+=1
 plant_names.append(i)
 plt.subplot(7,7,count)

 images_path=os.listdir(path+"/"+i)
 print("Number of images of "+i+":",len(images_path),"||",end=" ")
 total_images+=len(images_path)

 image_show=plt.imread(path+"/"+i+"/"+images_path[0])

 plt.imshow(image_show)
 plt.xlabel(i)

 plt.xticks([])
 plt.yticks([])


print("Total number of images we have",total_images)
```

```
Number of images of 0: 111 || Number of images of 1: 114 || Number of
images of 10: 212 || Number of images of 11: 187 || Number of images of 12:
233 || Number of images of 13: 159 || Number of images of 14: 67 || Number
of images of 15: 847 || Number of images of 16: 386 || Number of images of
17: 74 || Number of images of 18: 158 || Number of images of 19: 241 ||
Number of images of 2: 39 || Number of images of 20: 153 || Number of
images of 21: 168 || Number of images of 22: 25 || Number of images of 23:
44 || Number of images of 24: 885 || Number of images of 25: 270 || Number
of images of 26: 187 || Number of images of 27: 84 || Number of images of
28: 297 || Number of images of 29: 147 || Number of images of 3: 288 ||
Number of images of 30: 291 || Number of images of 31: 146 || Number of
images of 32: 275 || Number of images of 33: 300 || Number of images of 34:
211 || Number of images of 35: 897 || Number of images of 36: 65 || Number
of images of 37: 224 || Number of images of 4: 239 || Number of images of
5: 181 || Number of images of 6: 132 || Number of images of 7: 83 || Number
of images of 8: 199 || Number of images of 9: 132 || Total number of images
we have 8751
```

```
import tensorflow
from tensorflow import keras
from keras.models import Sequential,load_model,Model
from keras.layers import
Conv2D,MaxPool2D,AveragePooling2D,Dense,Flatten,ZeroPadding2D,BatchNormalization,Acti
vation,Add,Input,Dropout,GlobalAveragePooling2D
from keras.optimizers import SGD
from keras.initializers import glorot_uniform
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint,EarlyStopping,ReduceLROnPlateau
```

WARNING:tensorflow:From D:\programs\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
base_model_tf=ResNet50(include_top=False,weights='imagenet',input_shape=(224,224,3),clas
ses=38)
```

```
WARNING:tensorflow:From D:\programs\anaconda3\Lib\site-
packages\keras\src\backend.py:1398: The name
```

```
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From D:\programs\anaconda3\Lib\site-
packages\keras\src\layers\normalization\batch_normalization.py:979: The
name tf.nn.fused_batch_norm is deprecated. Please use
tf.compat.v1.nn.fused_batch_norm instead.

Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 63s 1us/step
#Model building
base_model_tf.trainable=False
```

```
pt=Input(shape=(224,224,3))
func=tensorflow.cast(pt,tensorflow.float32)
x=preprocess_input(func) #This function used to zero-center each color channel wrt Imagenet
dataset
model_resnet=base_model_tf(x,training=False)
model_resnet=GlobalAveragePooling2D()(model_resnet)
model_resnet=Dense(128,activation='relu')(model_resnet)
model_resnet=Dense(64,activation='relu')(model_resnet)
model_resnet=Dense(38,activation='softmax')(model_resnet)


model_main=Model(inputs=pt,outputs=model_resnet)
model_main.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.cast (TFOpLambda)        (None, 224, 224, 3)       0

 tf.__operators__.getitem (  (None, 224, 224, 3)       0
 SlicingOpLambda)

 tf.nn.bias_add (TFOpLambda  (None, 224, 224, 3)       0
 )

 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 global_average_pooling2d (  (None, 2048)              0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 128)               262272

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 38)                2470

=================================================================
Total params: 23860710 (91.02 MB)
Trainable params: 272998 (1.04 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

```
#Image augmentation
train_datagen=
ImageDataGenerator(shear_range=0.2,zoom_range=0.2,horizontal_flip=False,vertical_flip=Fals
e
                   ,fill_mode='nearest',width_shift_range=0.2,height_shift_range=0.2)

val_datagen=ImageDataGenerator()

path_train='D:/ylebbah/data/plant-kaggle/PlantVillage-
Dataset/data_distribution_for_SVM/train'

path_valid='D:/ylebbah/data/plant-kaggle/PlantVillage-
Dataset/data_distribution_for_SVM/test'

train=
train_datagen.flow_from_directory(directory=path_train,batch_size=32,target_size=(224,224),
                          color_mode='rgb',class_mode='categorical',seed=42)

valid=val_datagen.flow_from_directory(directory=path_valid,batch_size=32,target_size=(224,
224),color_mode='rgb',class_mode='categorical')
```

```
Found 8751 images belonging to 38 classes.
Found 10547 images belonging to 38 classes.
```

```
es=EarlyStopping(monitor='val_accuracy',verbose=1,patience=7,mode='auto')
mc=ModelCheckpoint(filepath='D:/ylebbah/data/plant-kaggle/PlantVillage-
Dataset/data_distribution_for_SVM/content',monitor='val_accuracy',verbose=1,save_best_on
ly=True)
lr=ReduceLROnPlateau(monitor='val_accuracy',verbose=1,patience=5,min_lr=0.001)
```

```
model_main.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

```
#Training
model_main.fit(train,validation_data=valid,epochs=30,steps_per_epoch=200,verbose=1,callba
cks=[mc,es,lr])
```

```
Epoch 1/30
200/200 [==============================] - ETA: 0s - loss: 0.3709 -
accuracy: 0.8875
Epoch 1: val_accuracy improved from -inf to 0.89533, saving model to
D:/ylebbah/data/plant-kaggle/PlantVillage-
Dataset/data_distribution_for_SVM\content
INFO:tensorflow:Assets written to: D:/ylebbah/data/plant-
kaggle/PlantVillage-Dataset/data_distribution_for_SVM\content\assets
INFO:tensorflow:Assets written to: D:/ylebbah/data/plant-
kaggle/PlantVillage-Dataset/data_distribution_for_SVM\content\assets
200/200 [==============================] - 1066s 5s/step - loss: 0.3709 -
accuracy: 0.8875 - val_loss: 0.3403 - val_accuracy: 0.8953 - lr: 0.0010
Epoch 2/30
200/200 [==============================] - ETA: 0s - loss: 0.2675 -
accuracy: 0.9168
Epoch 2: val_accuracy improved from 0.89533 to 0.90832, saving model to
D:/ylebbah/data/plant-kaggle/PlantVillage-
Dataset/data_distribution_for_SVM\content
INFO:tensorflow:Assets written to: D:/ylebbah/data/plant-
kaggle/PlantVillage-Dataset/data_distribution_for_SVM\content\assets
```

```
INFO:tensorflow:Assets written to: D:/ylebbah/data/plant-
kaggle/PlantVillage-Dataset/data_distribution_for_SVM\content\assets
200/200 [==============================] - 960s 5s/step - loss: 0.2675 -
accuracy: 0.9168 - val_loss: 0.2967 - val_accuracy: 0.9083 - lr: 0.0010
Epoch 3/30
200/200 [==============================] - ETA: 0s - loss: 0.1967 -
accuracy: 0.9336
Epoch 3: val_accuracy did not improve from 0.90832
200/200 [==============================] - 884s 4s/step - loss: 0.1967 -
accuracy: 0.9336 - val_loss: 0.3488 - val_accuracy: 0.8903 - lr: 0.0010
Epoch 4/30
200/200 [==============================] - ETA: 0s - loss: 0.1601 -
accuracy: 0.9505
Epoch 4: val_accuracy improved from 0.90832 to 0.92519, saving model to
D:/ylebbah/data/plant-kaggle/PlantVillage-
Dataset/data_distribution_for_SVM\content
INFO:tensorflow:Assets written to: D:/ylebbah/data/plant-
kaggle/PlantVillage-Dataset/data_distribution_for_SVM\content\assets
INFO:tensorflow:Assets written to: D:/ylebbah/data/plant-
kaggle/PlantVillage-Dataset/data_distribution_for_SVM\content\assets
200/200 [==============================] - 868s 4s/step - loss: 0.1601 -
accuracy: 0.9505 - val_loss: 0.2374 - val_accuracy: 0.9252 - lr: 0.0010
Epoch 5/30
200/200 [==============================] - ETA: 0s - loss: 0.1464 -
accuracy: 0.9528
Epoch 5: val_accuracy did not improve from 0.92519
200/200 [==============================] - 852s 4s/step - loss: 0.1464 -
accuracy: 0.9528 - val_loss: 0.2683 - val_accuracy: 0.9159 - lr: 0.0010
Epoch 6/30
200/200 [==============================] - ETA: 0s - loss: 0.1247 -
accuracy: 0.9559
Epoch 6: val_accuracy did not improve from 0.92519
200/200 [==============================] - 853s 4s/step - loss: 0.1247 -
accuracy: 0.9559 - val_loss: 0.2475 - val_accuracy: 0.9234 - lr: 0.0010
Epoch 7/30
200/200 [==============================] - ETA: 0s - loss: 0.1233 -
accuracy: 0.9593
Epoch 7: val_accuracy improved from 0.92519 to 0.92955, saving model to
D:/ylebbah/data/plant-kaggle/PlantVillage-
Dataset/data_distribution_for_SVM\content
INFO:tensorflow:Assets written to: D:/ylebbah/data/plant-
kaggle/PlantVillage-Dataset/data_distribution_for_SVM\content\assets
INFO:tensorflow:Assets written to: D:/ylebbah/data/plant-
kaggle/PlantVillage-Dataset/data_distribution_for_SVM\content\assets
200/200 [==============================] - 866s 4s/step - loss: 0.1233 -
accuracy: 0.9593 - val_loss: 0.2238 - val_accuracy: 0.9296 - lr: 0.0010
Epoch 8/30
200/200 [==============================] - ETA: 0s - loss: 0.1018 -
accuracy: 0.9658
Epoch 8: val_accuracy improved from 0.92955 to 0.93610, saving model to
D:/ylebbah/data/plant-kaggle/PlantVillage-
Dataset/data_distribution_for_SVM\content
INFO:tensorflow:Assets written to: D:/ylebbah/data/plant-
kaggle/PlantVillage-Dataset/data_distribution_for_SVM\content\assets
INFO:tensorflow:Assets written to: D:/ylebbah/data/plant-
kaggle/PlantVillage-Dataset/data_distribution_for_SVM\content\assets
200/200 [==============================] - 863s 4s/step - loss: 0.1018 -
accuracy: 0.9658 - val_loss: 0.2157 - val_accuracy: 0.9361 - lr: 0.0010
Epoch 9/30
200/200 [==============================] - ETA: 0s - loss: 0.1159 -
accuracy: 0.9604
```

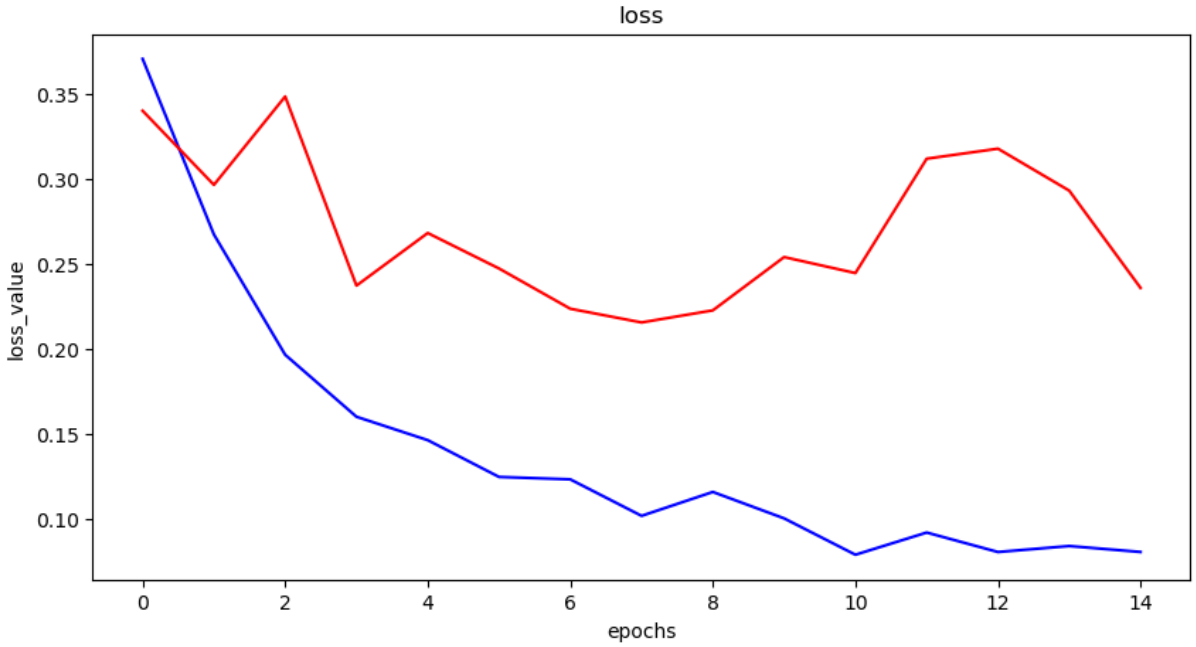
```
Epoch 9: val_accuracy did not improve from 0.93610
200/200 [==============================] - 853s 4s/step - loss: 0.1159 -
accuracy: 0.9604 - val_loss: 0.2228 - val_accuracy: 0.9340 - lr: 0.0010
Epoch 10/30
200/200 [==============================] - ETA: 0s - loss: 0.1003 -
accuracy: 0.9657
Epoch 10: val_accuracy did not improve from 0.93610
200/200 [==============================] - 850s 4s/step - loss: 0.1003 -
accuracy: 0.9657 - val_loss: 0.2542 - val_accuracy: 0.9241 - lr: 0.0010
Epoch 11/30
200/200 [==============================] - ETA: 0s - loss: 0.0789 -
accuracy: 0.9737
Epoch 11: val_accuracy did not improve from 0.93610
200/200 [==============================] - 849s 4s/step - loss: 0.0789 -
accuracy: 0.9737 - val_loss: 0.2448 - val_accuracy: 0.9295 - lr: 0.0010
Epoch 12/30
200/200 [==============================] - ETA: 0s - loss: 0.0920 -
accuracy: 0.9687
Epoch 12: val_accuracy did not improve from 0.93610
200/200 [==============================] - 848s 4s/step - loss: 0.0920 -
accuracy: 0.9687 - val_loss: 0.3121 - val_accuracy: 0.9152 - lr: 0.0010
Epoch 13/30
200/200 [==============================] - ETA: 0s - loss: 0.0805 -
accuracy: 0.9732
Epoch 13: val_accuracy did not improve from 0.93610
200/200 [==============================] - 851s 4s/step - loss: 0.0805 -
accuracy: 0.9732 - val_loss: 0.3180 - val_accuracy: 0.9079 - lr: 0.0010
Epoch 14/30
200/200 [==============================] - ETA: 0s - loss: 0.0840 -
accuracy: 0.9727
Epoch 14: val_accuracy did not improve from 0.93610
200/200 [==============================] - 850s 4s/step - loss: 0.0840 -
accuracy: 0.9727 - val_loss: 0.2933 - val_accuracy: 0.9149 - lr: 0.0010
Epoch 15/30
200/200 [==============================] - ETA: 0s - loss: 0.0805 -
accuracy: 0.9734
Epoch 15: val_accuracy did not improve from 0.93610
200/200 [==============================] - 850s 4s/step - loss: 0.0805 -
accuracy: 0.9734 - val_loss: 0.2360 - val_accuracy: 0.9311 - lr: 0.0010
Epoch 15: early stopping
<keras.src.callbacks.History at 0x22de01b14d0>
```

```
model_main.save("D:/ylebbah/data/plant-kaggle/PlantVillage-
Dataset/data_distribution_for_SVM/RESNET50_PLANT_DISEASE.h5")
```

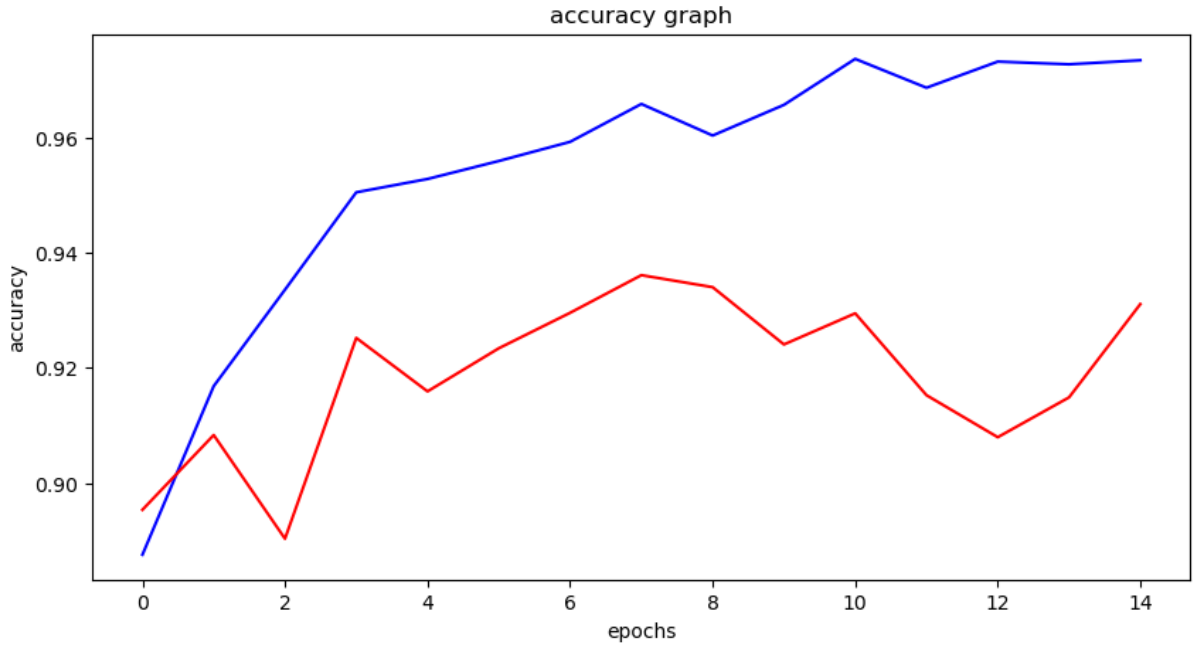
8.3.1 Exploiting the generated model

```
import matplotlib.pyplot as plt
%matplotlib inline
import cv2
from PIL import Image
```

```
plt.figure(figsize=(10,5))
plt.plot(model_main.history.history['loss'],color='b',label='Training loss')
plt.plot(model_main.history.history['val_loss'],color='r',label='Validation loss')
plt.xlabel("epochs")
plt.ylabel("loss_value")
plt.title("loss")
Text(0.5, 1.0, 'loss')
```




```
plt.figure(figsize=(10,5))
plt.plot(model_main.history.history['accuracy'],color='b',label='Training accuracy')
plt.plot(model_main.history.history['val_accuracy'],color='r',label='Validation accsuracy')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.title("accuracy graph")
Text(0.5, 1.0, 'accuracy graph')
```



8.4 Object detection / Yolo V8

https://yolov8.com/

widget-demo-video
-cta-transcode.mp4

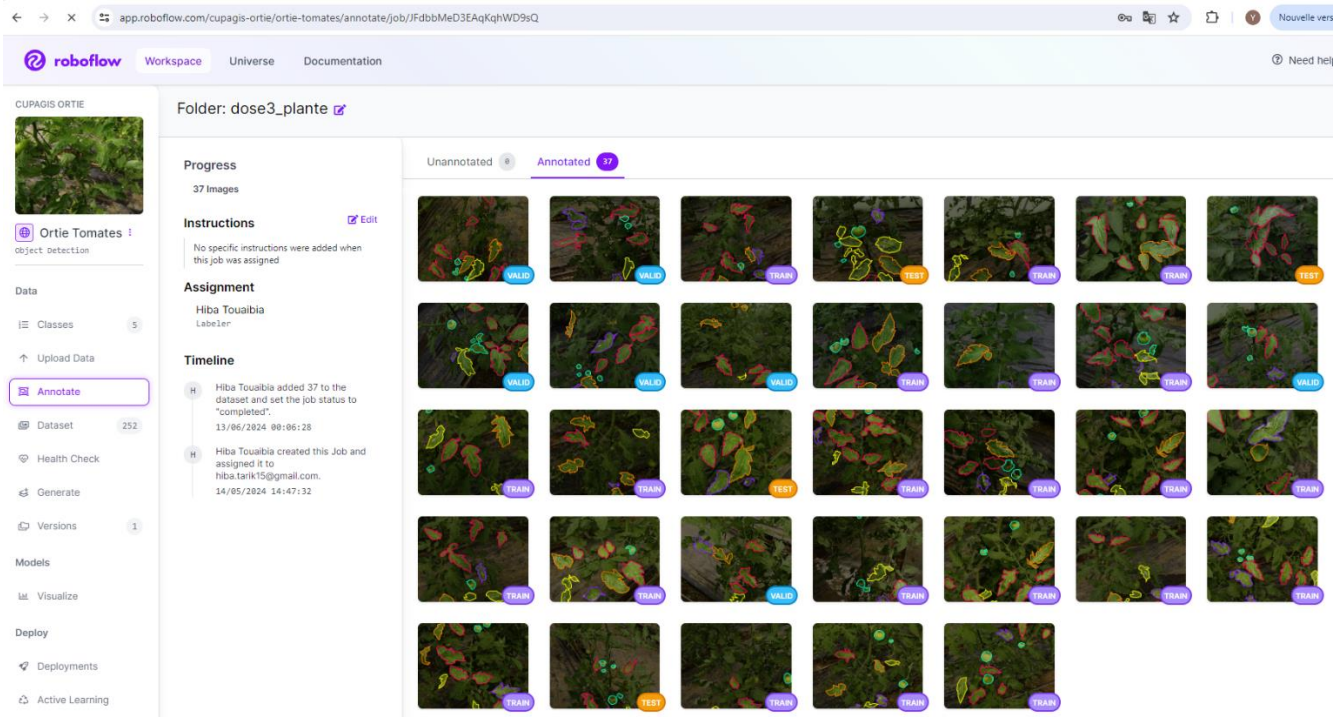
Dataset de plantes

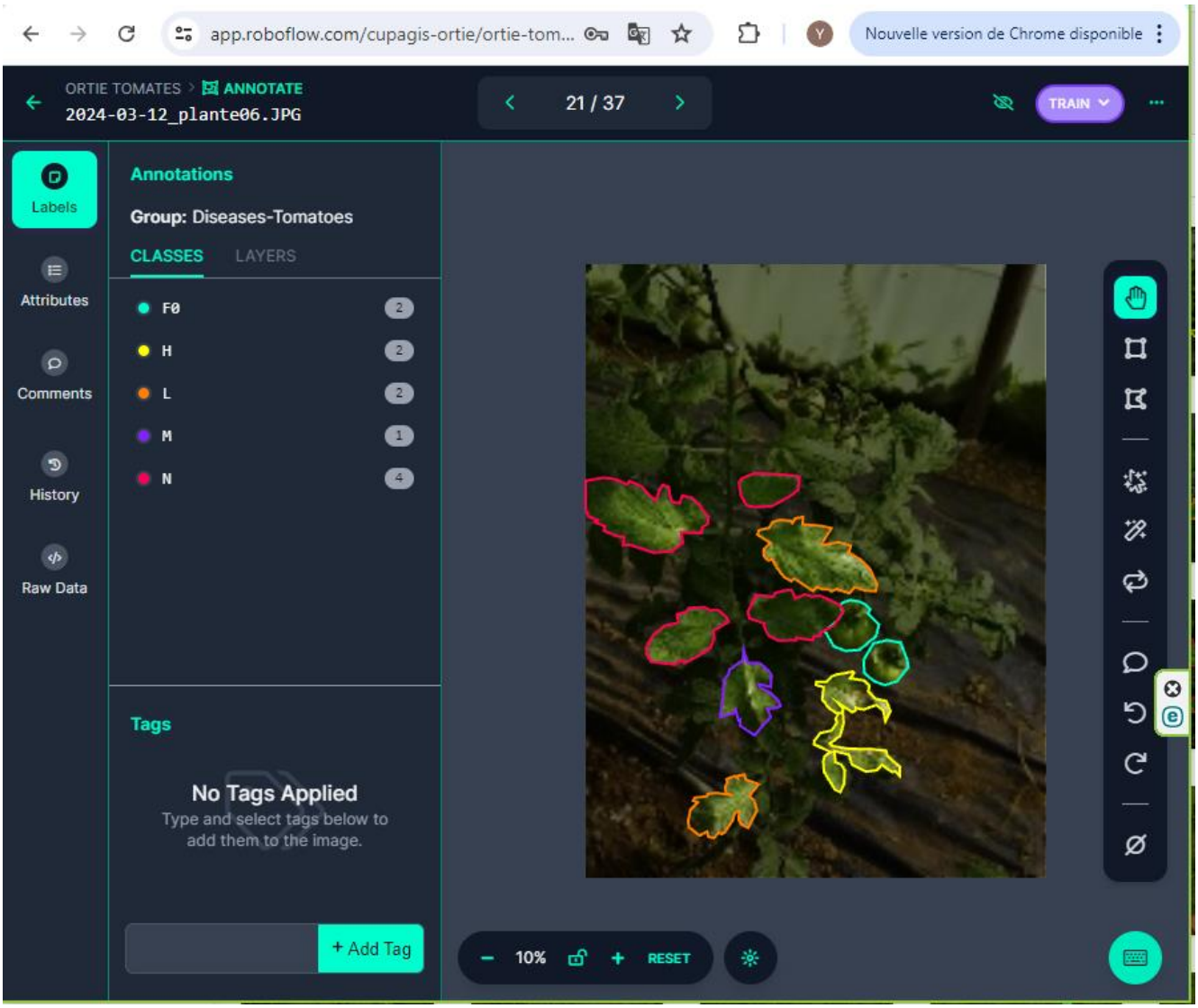





Etiquetage et annotation avec ‘’roboflow’’

https://app.roboflow.com/




app.roboflow.com/cupagis-ortie/ortie-tom...
Nouvelle version de Chrome disponible
ORTIE TOMATES
ANNOTATE
2024-03-12_plante06.JPG
21 / 37
TRAIN
Labels
Attributes
Comments
History
Raw Data
Annotations
Group: Diseases-Tomatoes
CLASSES
LAYERS
F0 2
H 2
L 2
M 1
N 4
Tags
No Tags Applied
Type and select tags below to add them to the image.
+ Add Tag
10%
RESET

Apprentissage avec Yolo V8

```
from ultralytics import YOLO
if __name__ == '__main__':
    # Load the Yolo model
    model = YOLO("yolov8s.pt")

    # Enable Cuda for GPU training
    model = model.to('cuda')

    # Train the model
    results = model.train(data="D:/ylebbah/learning-ap/Ortie
Tomates.v1i.yolov8/data.yaml", epochs=300,
batch=50,  patience=100, save=True)

     # Validate the model
    metrics = model.val()
```