Introduction à l'apprentissage machine
Etude de cas en agriculture de précision

Introduction to Machine learning
Case studies in precision agriculture

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

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

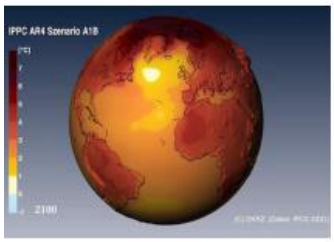
Données = carburant de l'apprentissage automatique [2]

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

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

Applications

Anti-Spam (*Classifieur Bayesien*)

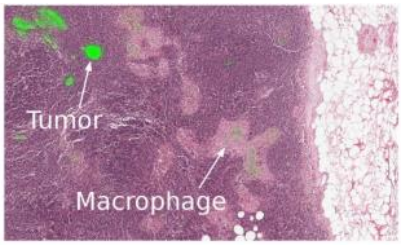
1997 : DeepBlue bat Kasparov
2017: Alpha GO bat Ke Jie
2019: AlphaStar champion de StarCraft

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]

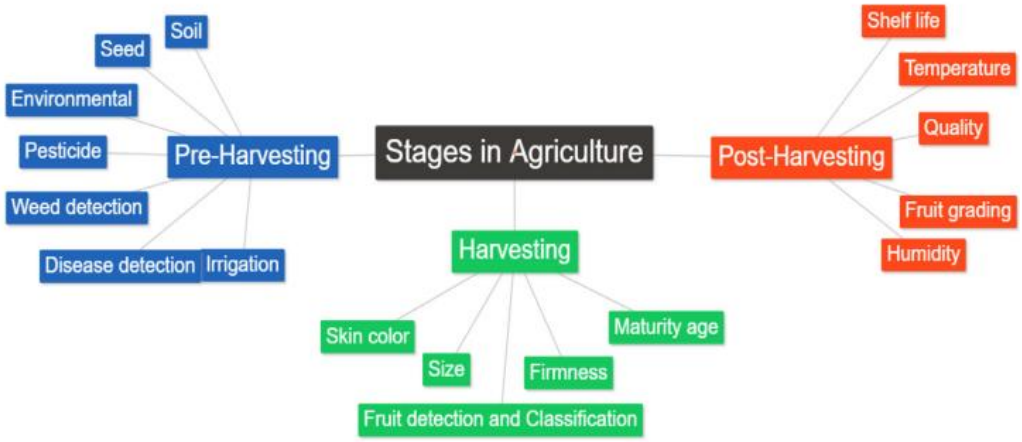
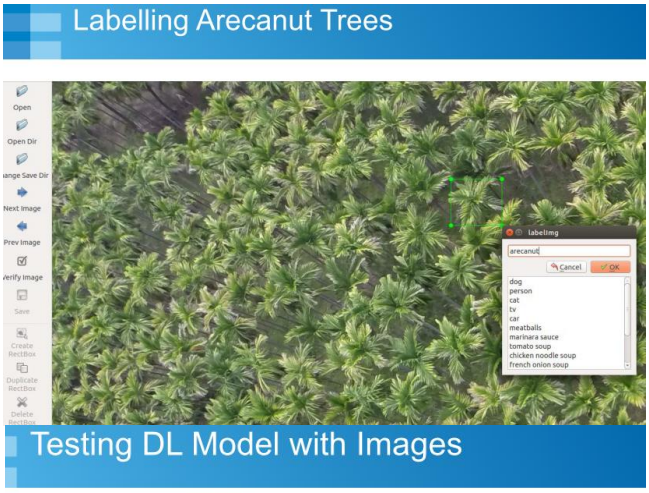
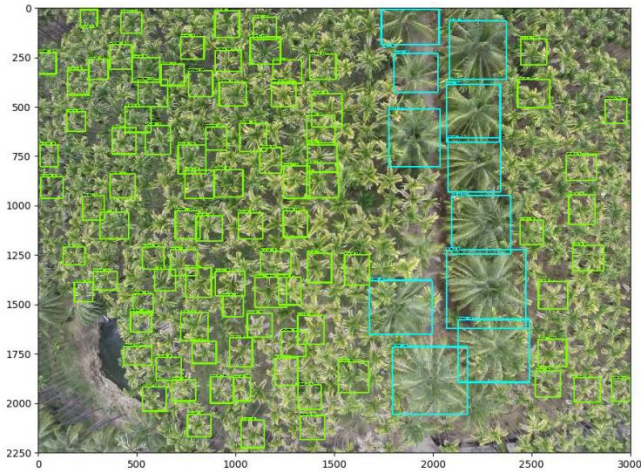


Fig. 2. Important parameters considered in each stage of farming.





[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

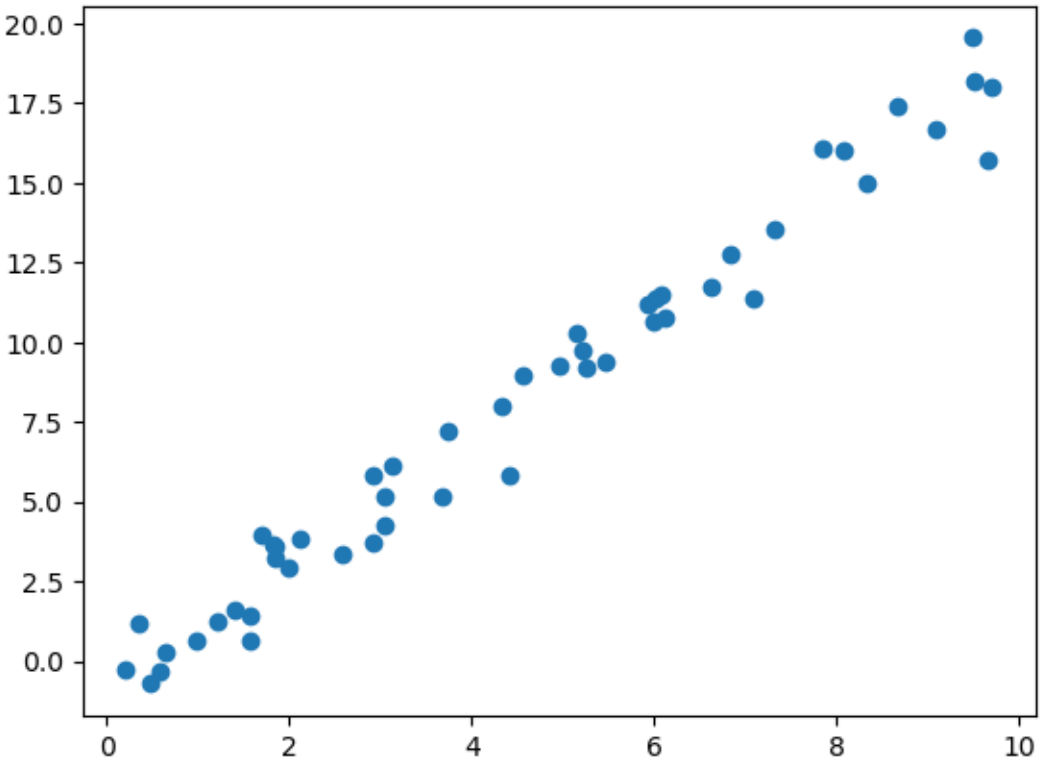
Exemple 1

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

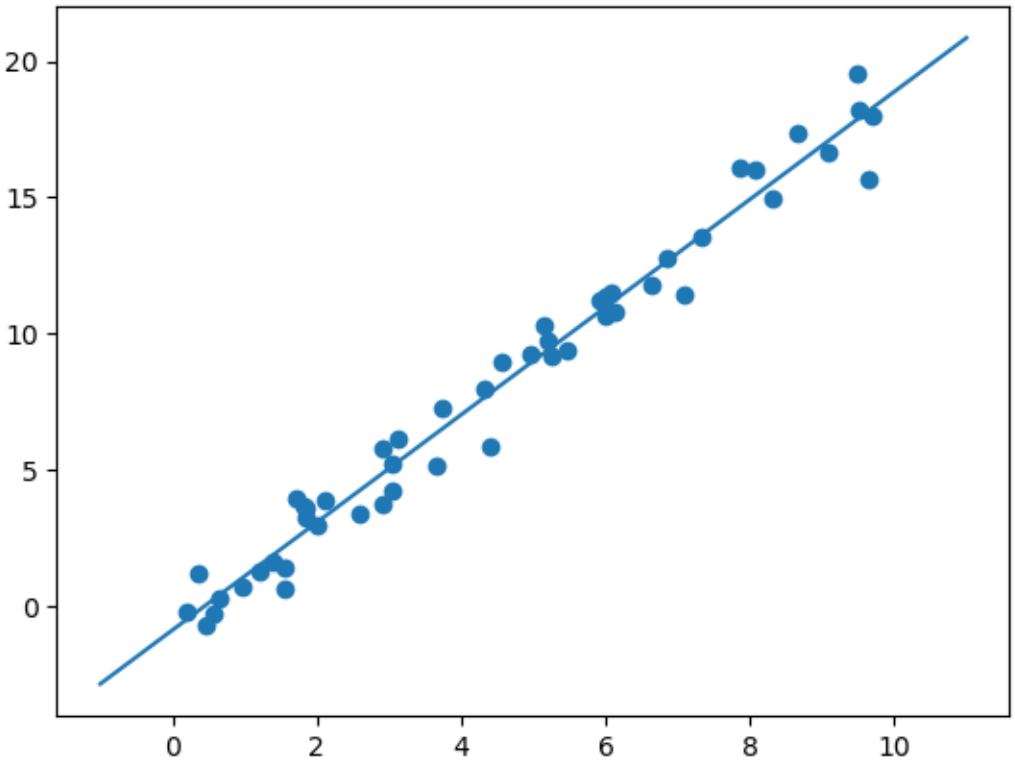
N°	N	R
27	1.9967	2.9215
28	5.1423	10.2882
29	5.9241	11.2099
30	0.4645	_
31	6.0754	11.5123
32	1.7052	3.9485
33	0.6505	0.2652
34	9.4889	19.5424
35	9.6563	15.6929
36	8.0840	15.9898
37	3.0461	5.1793
38	0.9767	0.6544
39	6.8423	12.7764
40	4.4015	5.8155
41	1.2204	1.2211
42	4.9518	9.2607
43	0.3439	1.1657
44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

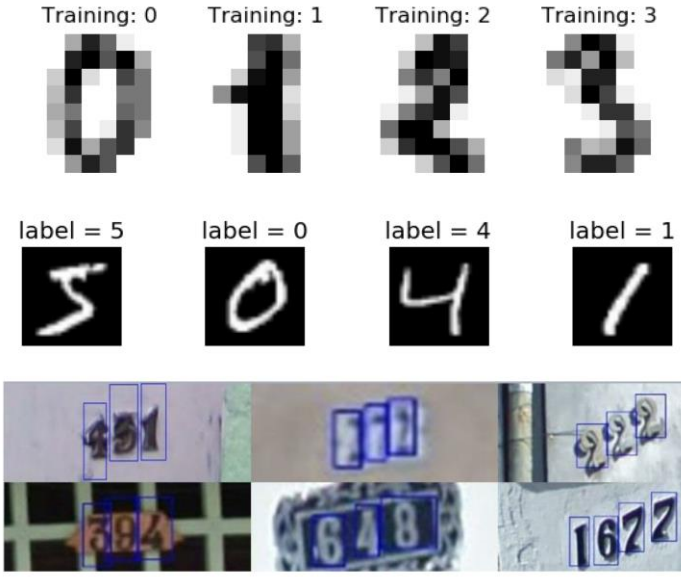


$$\mathbf{R = 1.9776566\ N}$$

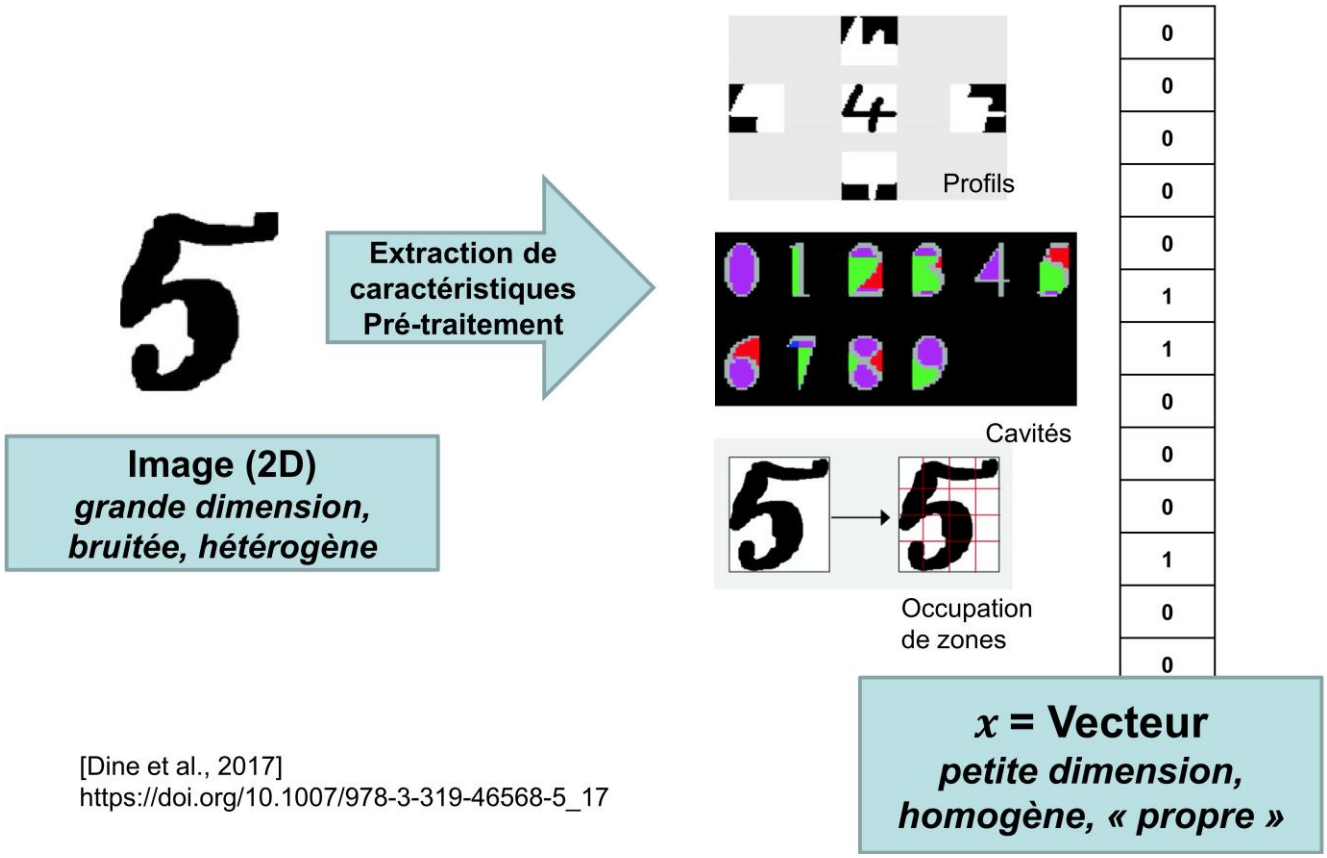
Exemple 2

Etape 1

Etape 2



Etape 3

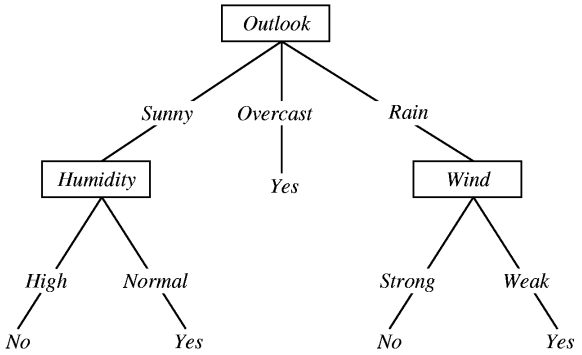


[Dine et al., 2017]
https://doi.org/10.1007/978-3-319-46568-5_17

Etape 4 : Choix du modèle

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

Arbre de décision :



SVM : …

Naive-bayes : …

…

Deep-learning (Réseaux de neurones) : …

2 Pyhton par l'exemple

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")
```

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)
```

Chaines de caractères

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)
```

Exception

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

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)
```

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]}
```

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]
```

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)
```

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
```

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)
```

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]
```

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
```

3 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

```
movies = ["Annie Hall", "Ben-Hur", "Casablanca", "Gandhi", "West Side Story"]
num_oscars = [5, 11, 3, 8, 10]
# bars are by default width 0.8, so we'll add 0.1 to the left coordinates
# so that each bar is centered
xs = [i + 0.1 for i, _ in enumerate(movies)]
# plot bars with left x-coordinates [xs], heights [num_oscars]
plt.bar(xs, num_oscars)
plt.ylabel("# of Academy Awards")
plt.title("My Favorite Movies")
# label x-axis with movie names at bar centers
plt.xticks([i + 0.5 for i, _ in enumerate(movies)], movies)
plt.show()
```

Puis,

```
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")
```

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

4.1 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.

4.2 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);
```

4.3 Classification 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)
```

4.4 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);
```

4.5 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.

5 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.

5.1 Les arbres de décision

Référence : https://philippe-preux.github.io/Documents/notes-de-cours-de-fouille-de-donnees.pdf
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.

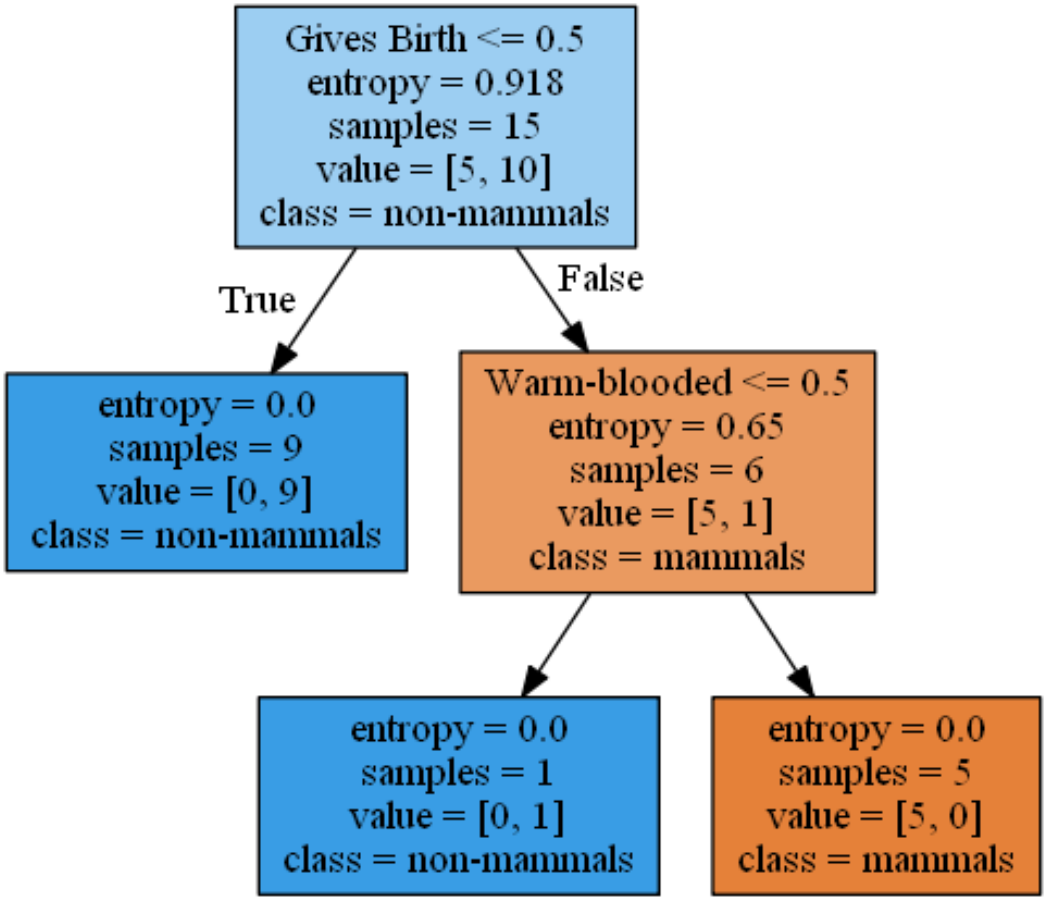
```
import pandas as pd
data = pd.read_csv('D:/ylebbah/tex/enseignement/poly-ml-ia-bigd-ap/journee-PAAI-2024/vertebrate.csv',header='infer')
data
```

	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

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[4] :

```
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

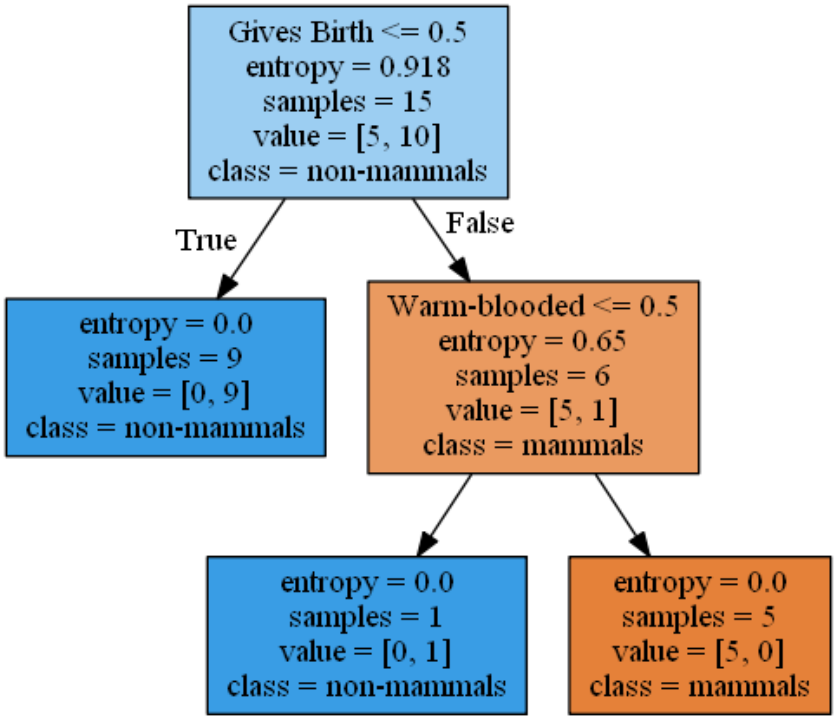
Puis taper (dans Anaconda Prompt) :

```
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)
```

[4] 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

```
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```



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)))
```

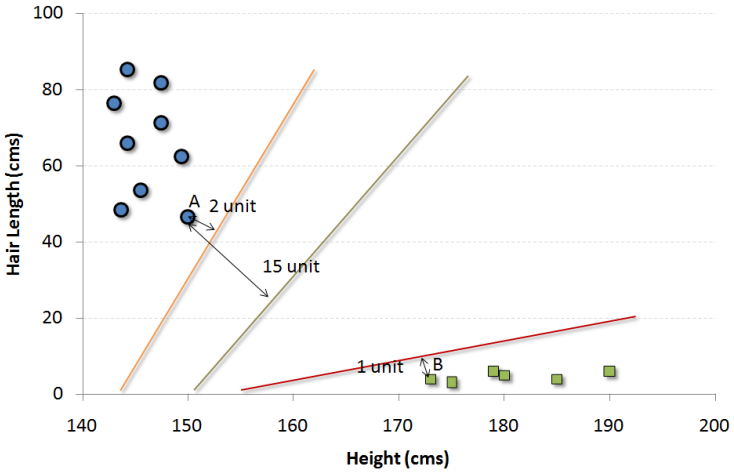
```
Accuracy on test data is 0.75
```

5.2 Les séparateurs à vastes marges non linéaires

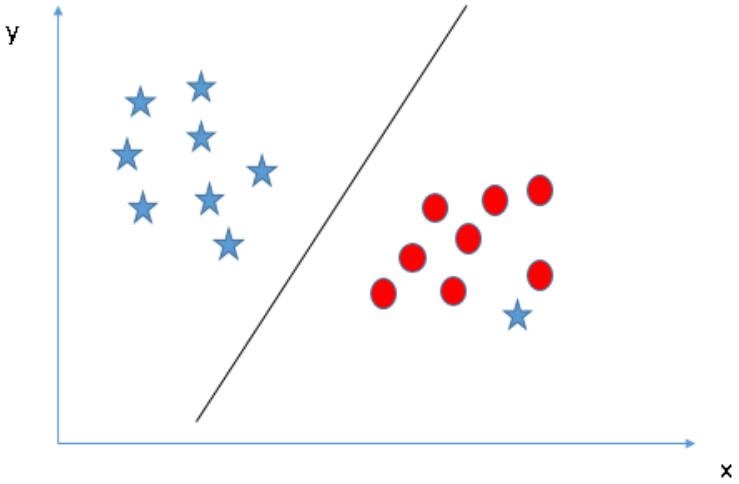
Le code ci-dessous montre un exemple d'utilisation d'un séparateur à vastes marges non linéaire avec un noyau de fonction de base radiale gaussienne pour s'adapter à l'ensemble de données bidimensionnel.

Principe

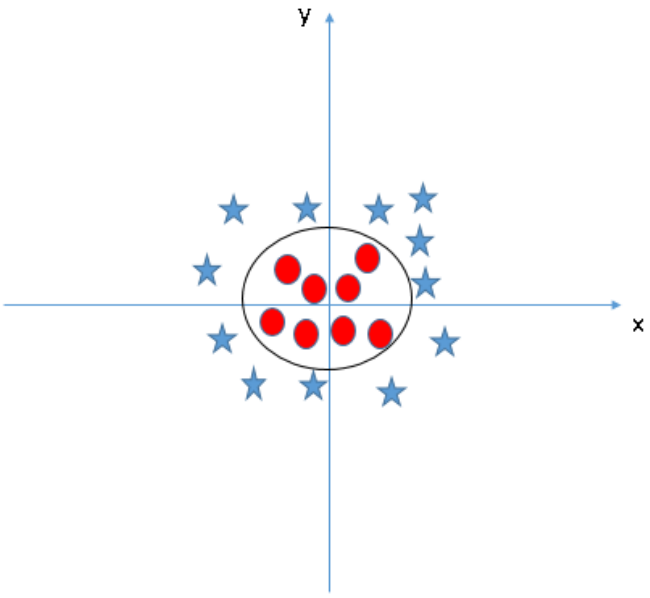
Cas 1 :



Cas 2 :



Cas 3 :

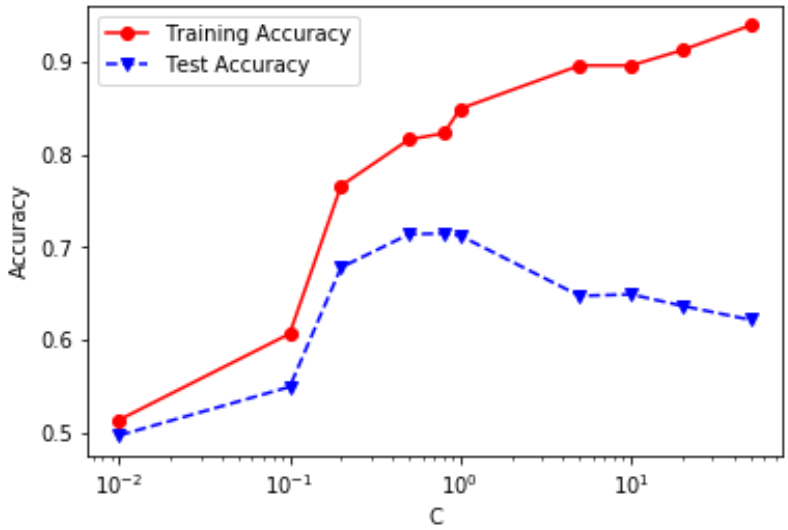


Mise en pratique

Soit C la paramètre de régularisation[5].

```
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')
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.

[5] 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.

5.3 Classifieurs linéaires : régression logistique

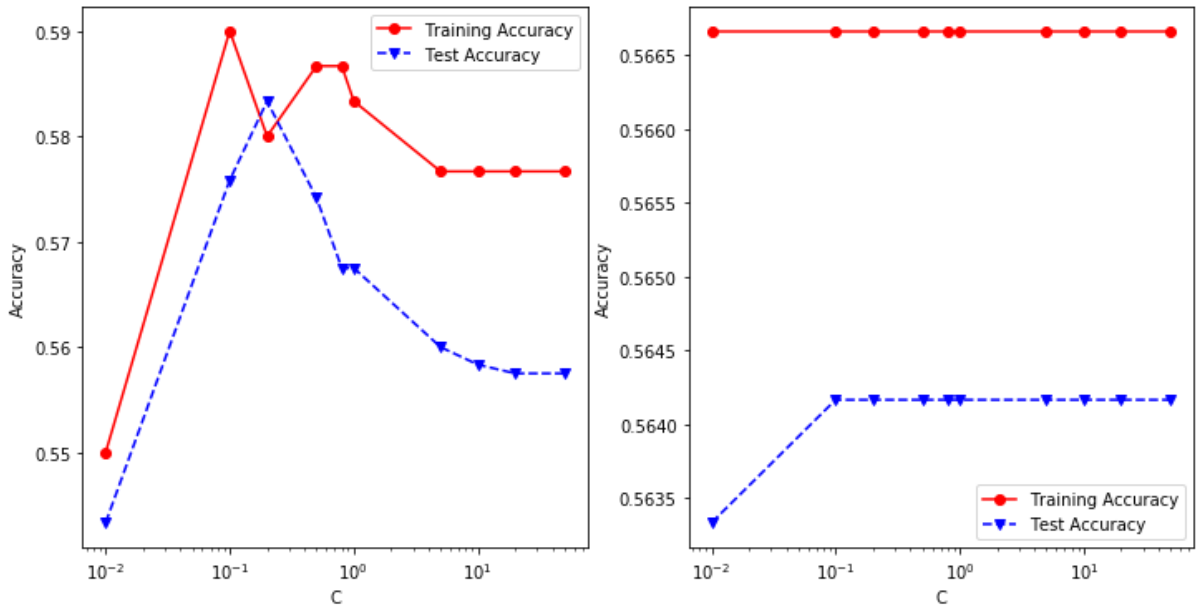
Les classifieurs linéaires tels que la régression logistique et le séparateur à vaste marge (SVM) construisent un hyperplan de séparation linéaire pour distinguer les instances de différentes classes.

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".

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')
```

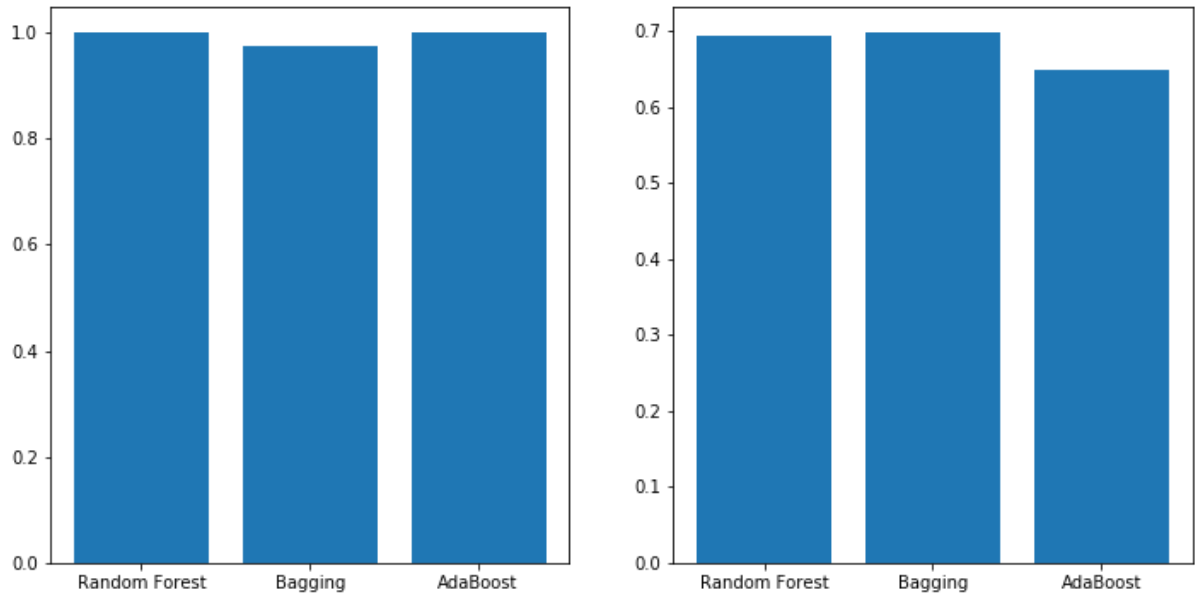

Training Accuracy
Test Accuracy
Accuracy
C
0.59
0.58
0.57
0.56
0.55
0.5665
0.5660
0.5655
0.5650
0.5645
0.5640
0.5635
10^{-2}
10^{-1}
10^{0}
10^{1}

5.4 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=numBaseClassif
iers)
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=numBaseClas
sifiers)
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.

6 Apprentissage non-supervisé : Clustering des K-moyennes (K-means)

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.

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.

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

```
import numpy as np
import pandas as pd
import plotly.express as px
import seaborn as sns
import plotly.graph_objects as go
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans

sse = []
for i in range(1,9):
   kmeans = KMeans(n_clusters=i , max_iter=300)
   kmeans.fit(X)
   sse.append(kmeans.inertia_)

fig = px.line(y=sse,template="seaborn",title='Eblow Method')
fig.update_layout(width=800, height=600,
title_font_color="#BF40BF",
xaxis=dict(color="#BF40BF",title="Clusters"),
yaxis=dict(color="#BF40BF",title="SSE"))
```

Regardons le cluster affecté à chaque élément du dataset :

```
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
clusters = kmeans.fit_predict(X)
```

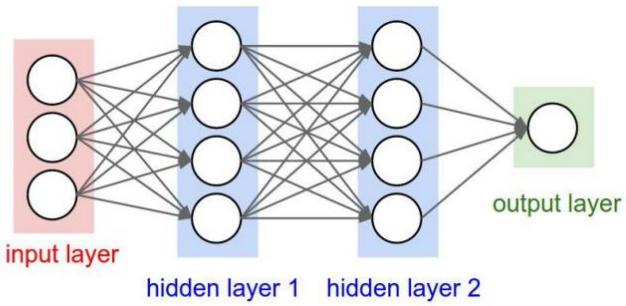
L'algorithme de clustering k-means affecte les trois premiers utilisateurs à un cluster et les trois derniers utilisateurs au second cluster. Les résultats sont conformes à nos attentes. Nous pouvons également afficher le **centroïde** pour chacun des deux clusters.

```
centroids = kmeans.cluster_centers_
pd.DataFrame(centroids,columns=X.columns)
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.901613	2.748387	4.393548	1.433871
1	5.006000	3.428000	1.462000	0.246000
2	6.850000	3.073684	5.742105	2.071053

A Simple Neural Network

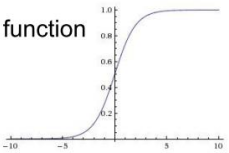
Use recent three days' average temperature to predict tomorrow's average temperature.

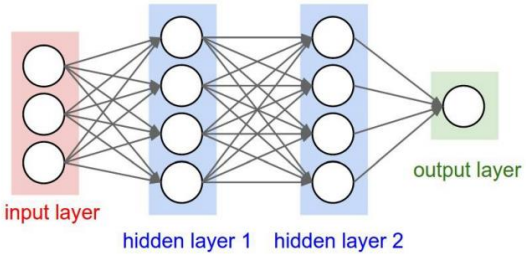


//

A Simple Neural Network

Sigmoid function



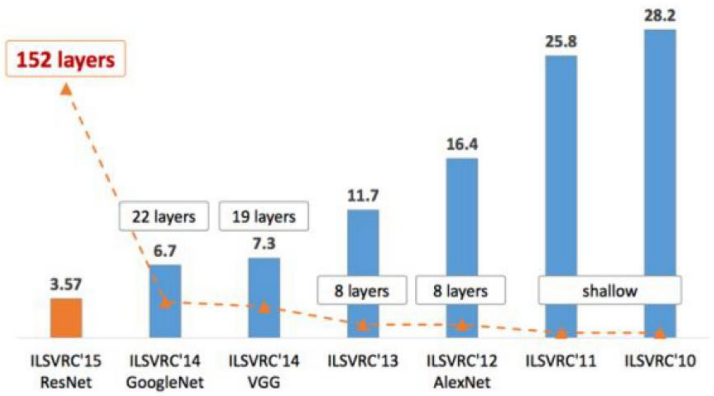


W1, b1, W2, b2, W3, b3 are network parameters that need to be learned.

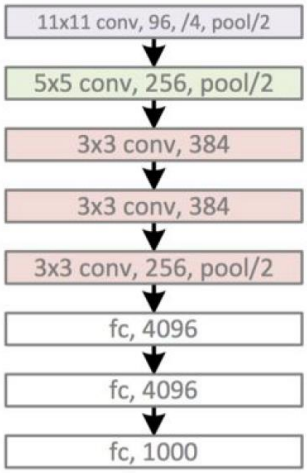
```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate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

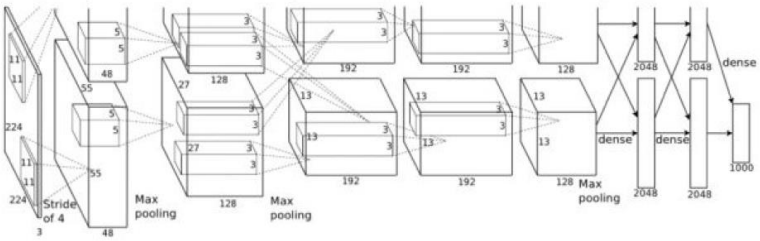
ResNet (2016 by Kaiming He et al.)

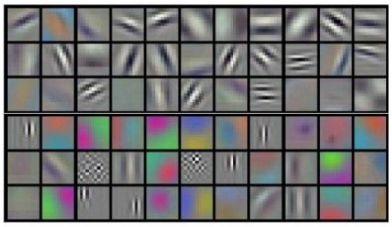
The winner in ILSVRC 2015



AlexNet (2012 by Krizhevsky et al.)



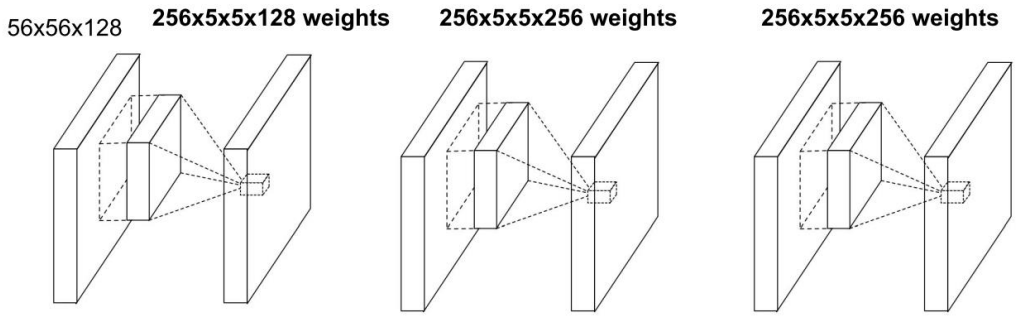




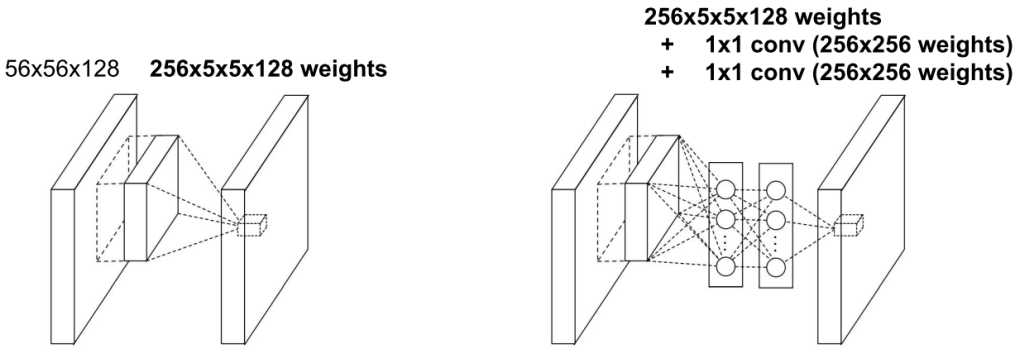
The first work that popularized Convolutional Networks in Computer Vision

What's different?

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



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



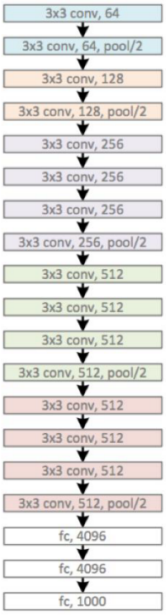
(a) Linear convolution layer

(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



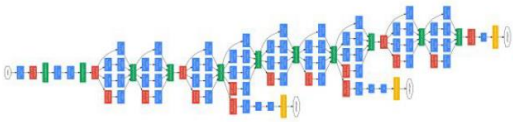
Karen Simonyan, Andrew Zisserman: **Very Deep** Convolutional Networks for Large-Scale Image Recognition.

- 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

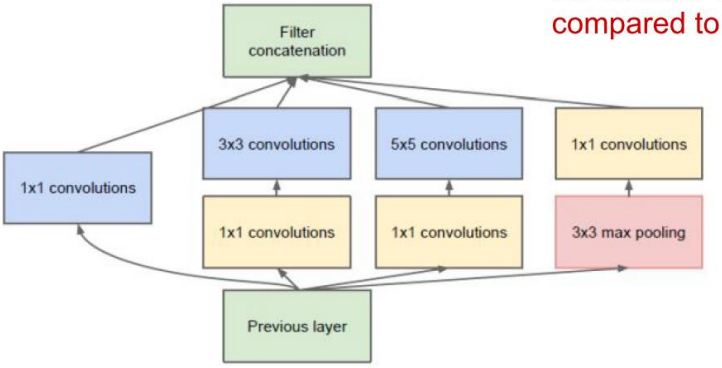
GoogleNet (2015 by Szegedy et al.)

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).

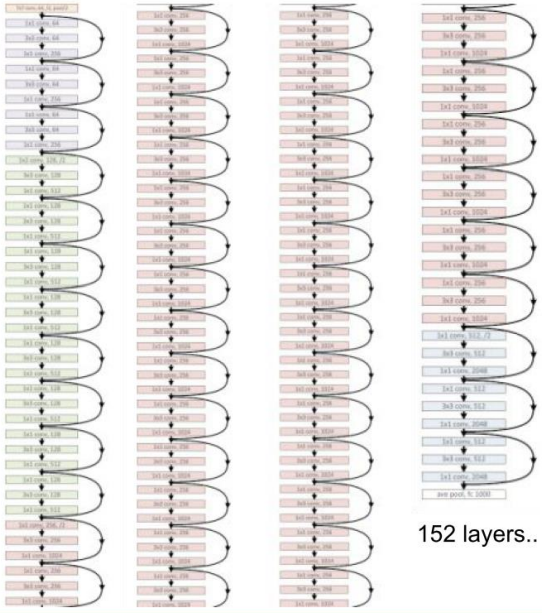
-- edited from CS231N



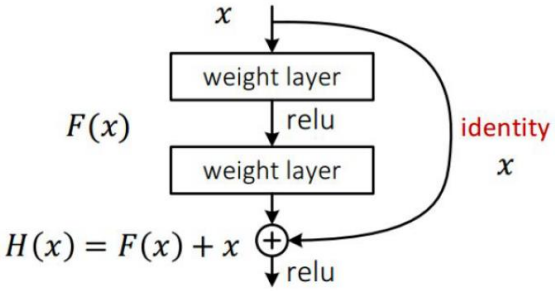
An Inception Module: a new building block..

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.)

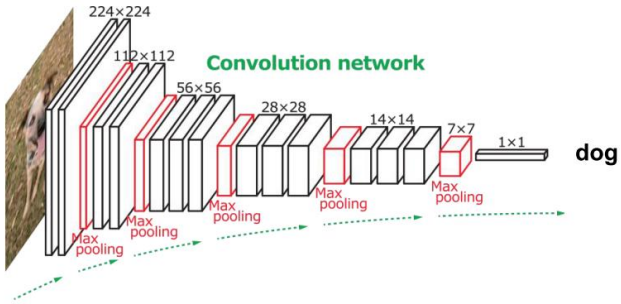


152 layers..

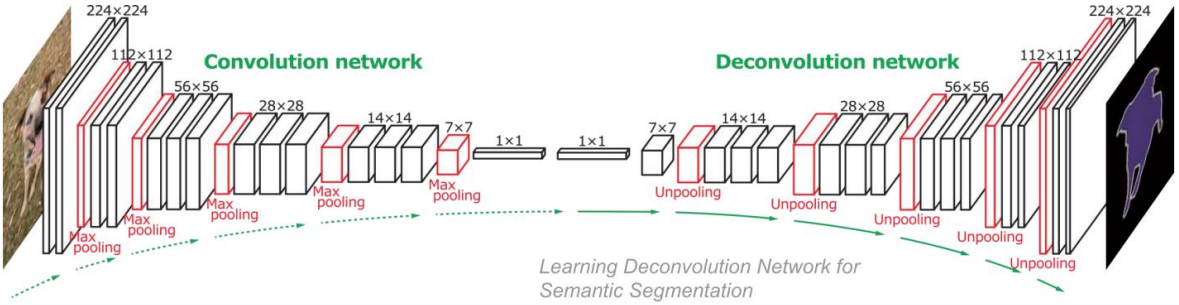


- Deeper network hard to train: Use skip connections for residual learning.
- Heavy use of batch normalization.
- No fully connected layers.

Classification:



Segmentation:



Learning Deconvolution Network for Semantic Segmentation

8 Cas d'étude

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')
```

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
```

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()
```

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()
```

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

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)

Prediction

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))
```

Best model choosing

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':['uniform','dis
tance']}
   }
}

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)}')
```

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()
```

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()
```

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 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,Activation,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),classes=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=False
                   ,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.
```

Training

```
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_only=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,callbacks=[mc,e
s,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
```

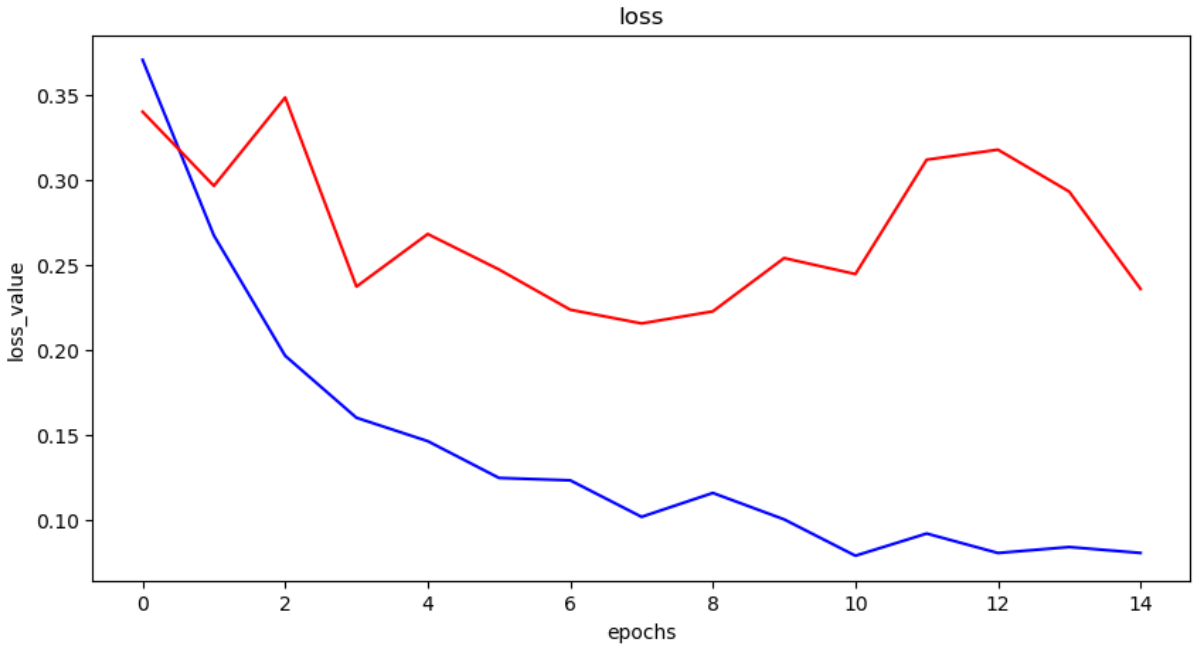
```
Epoch 14/30
200/200 [==============================] - ETA: 0s - loss: 0.0840 - accuracy:
0.9727
Epoch 14: val_accuracy did not improve from 0.93610
200/200 [==============================] - 850s 4s/step - loss: 0.0840 - accuracy:
0.9727 - val_loss: 0.2933 - val_accuracy: 0.9149 - lr: 0.0010
Epoch 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")
```

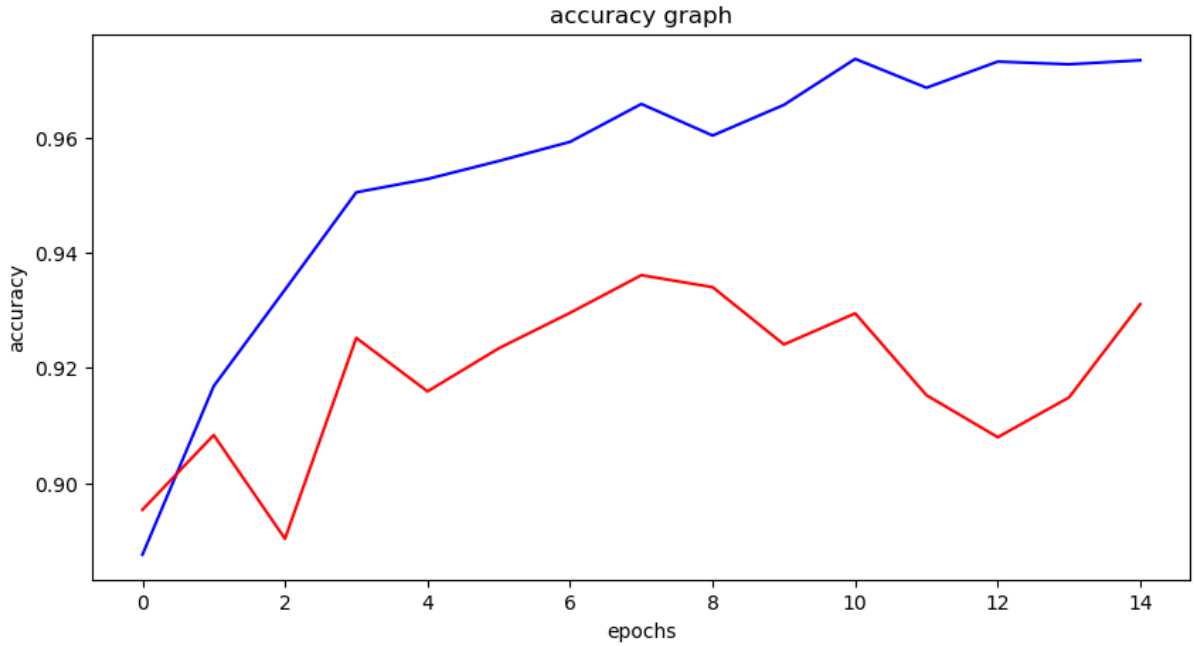
Exploiting the generated model

```
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')
```


accuracy graph
accuracy
epochs
0.96
0.94
0.92
0.90
0
2
4
6
8
10
12
14