



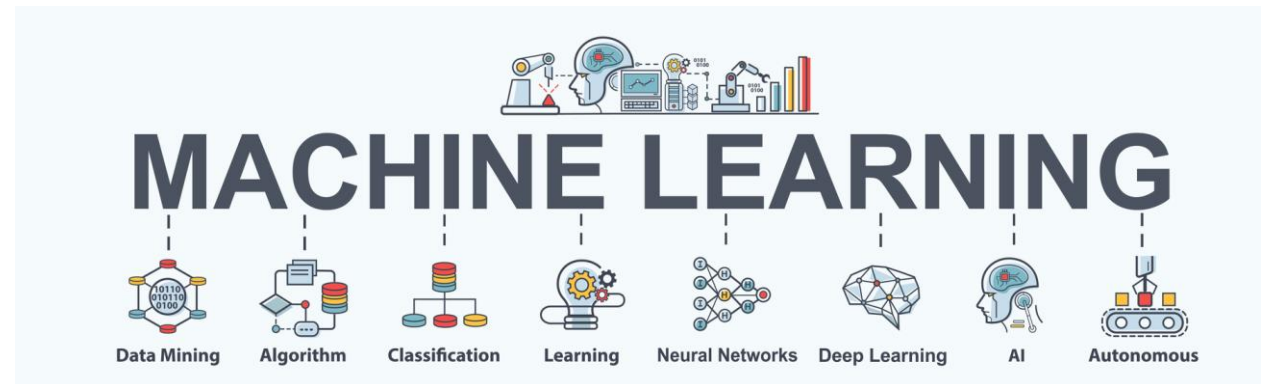
ML pipeline

MME. KHAOULA ELBEDOUI

Sommaire

- 1. Préambule**
- 2. Pipeline basique d'un ML**
- 3. Pipeline avancé d'un ML**
 - 1. Préparation des données**
 - 2. Peaufinage du modèle de ML**

Préambule



Préambule

Outils

ML Language



Data manipulation



Modeling



Visualization

matplotlib




Notebook



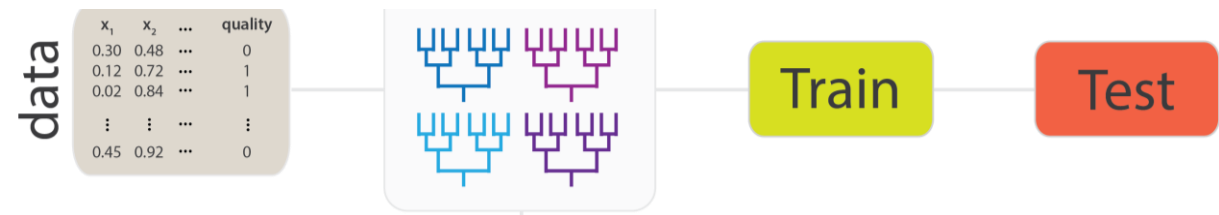
Préambule

Installation si nécessaire



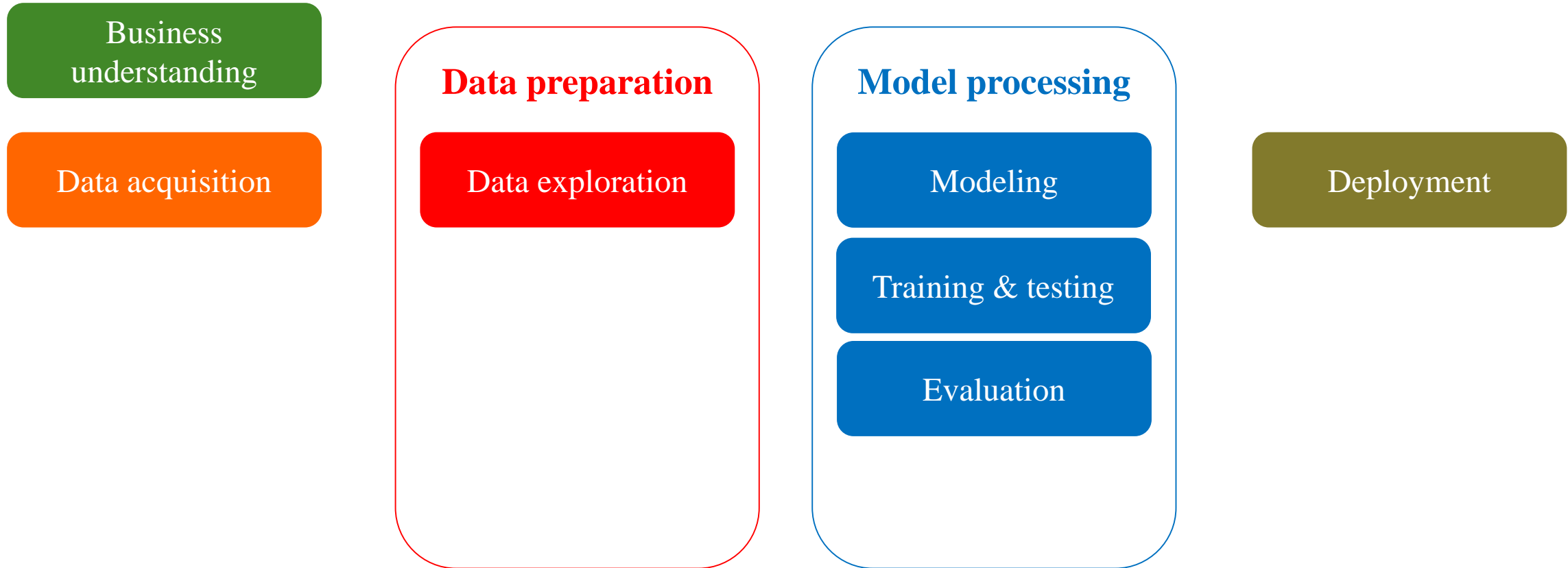
```
# Pour installer pandas s'il n'existe pas sur votre système  
!pip install pandas
```

Pipeline basique d'un ML



Pipeline basique d'un ML

Étapes



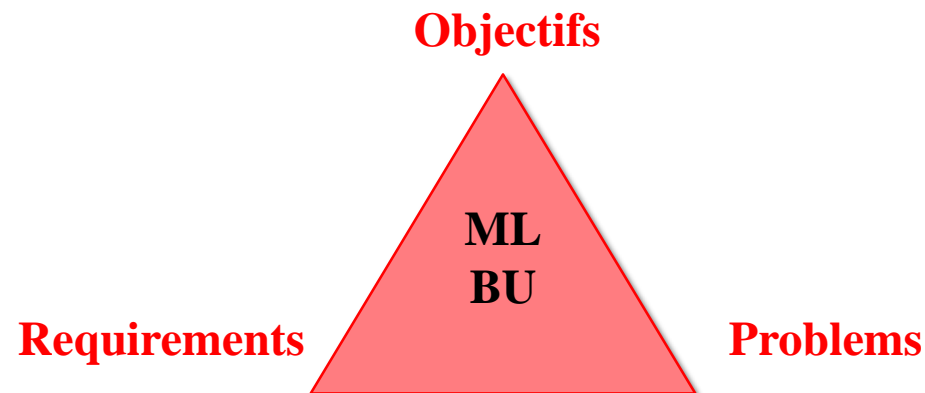
Lien colab :

<https://colab.research.google.com/drive/1cZCL8tFI0aYTzFL9gRGQP1szFNLwwWjp?usp=sharing>

Pipeline basique d'un ML

Etape 0 – Business understanding

- Travailler avec le spécialiste du domaine pour identifier les **objectifs** métiers (détecter, dénombrer, ...)
- Définir les **besoins** métiers (classification binaire, classification non binaire, régression, recommandation, ...)
- Comprendre les **problèmes** métiers (données labélisées ou non, données manquantes ou non, ...)
- Traduire les objectifs, les besoins et les problèmes métiers en des définitions appropriées en ML



Pipeline basique d'un ML

Etape 1 – Data acquisition

Les données peuvent être importées de plusieurs manières et ce à partir :

- d'un fichier local et ceci quelque soit son type
- du drive
- d'un lien externe
- ...

Pipeline basique d'un ML

Etape 1 – Data acquisition

Les données importées peuvent être de plusieurs types :

- un fichier csv
- un fichier Excel
- un fichier json
- ...

On peut faire leurs lectures comme suit :

```
[ ] data = pd.read_csv('data/fichier1.csv')    # Fichier CSV  
    data = pd.read_excel('data/fichier2.xls') # Fichier excel  
    data = pd.read_json('data/fichier3.json') # Fichier json
```

Pipeline basique d'un ML

Etape 1 – Data acquisition

On peut importer le fichier au système colab à travers `files.upload()`

A - Importer un fichier au système de fichiers local de colab

`files.upload` affiche un dictionnaire des fichiers importés. Ce dictionnaire est identifié par le nom du fichier, et les valeurs sont les données qui ont été importées.

```
[ ] from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
    print('Fichier importé : "{name}" avec une taille de {length} octets'.format(
        name=fn, length=len(uploaded[fn])))
```

Sélect. fichiers Aucun fichier choisi

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Pipeline basique d'un ML

Etape 1 – Data acquisition

On peut télécharger le fichier depuis le système colab à travers `files.download()`

B - Télécharger un fichier depuis le système de fichiers local de colab

`files.download` appelle le téléchargement du fichier sur votre ordinateur local via un navigateur.

```
[ ] from google.colab import files

with open('/content/sample_data/README.md', 'w') as f:

    f.write('***** Nouveau texte dans le fichier *****')

files.download('/content/sample_data/README.md')
```

Pipeline basique d'un ML

Etape 1 – Data acquisition

On peut utiliser un fichier sur notre **drive**.

Il faut commencer par le montage, ensuite la lecture du fichier à partir du point du montage en précisant le [chemin absolu](#)

A - Google drive

```
[ ] # Il faut monter Google Drive, ensuite ouvrir le fichier en précisant son chemin
    from google.colab import drive
    drive.mount('/google_drive',force_remount=True)
    %cd /google_drive
```

```
▶ # Lecture à partir du point du montage
  import pandas as pd
  data = pd.read_csv('')
```

Pipeline basique d'un ML

Etape 1 – Data acquisition

On peut utiliser un fichier de **google sheet**.

Il faut commencer par installer **gsread**, ensuite s'authentifier et créer l'interface sheet

B - Google sheet

Les exemples ci-dessous utilisent la bibliothèque Open Source [gsread](#) pour interagir avec Google Sheets.

```
[ ] # Il faut commencer par installer le package en utilisant l'outil gsread

!pip install --upgrade gsread

# Ensuite, il faut importer la bibliothèque, s'authentifier et créer l'interface avec Sheet

from google.colab import auth
auth.authenticate_user()

import gsread
from oauth2client.client import GoogleCredentials

gc = gsread.authorize(GoogleCredentials.get_application_default())
```

Pipeline basique d'un ML

Etape 1 – Data acquisition

On peut créer un fichier dans **google sheet** avec `create()` et ensuite l'ouvrir et le remplir par des valeurs

```
▶ # Créer une feuille googlesheet avec des données  
sh = gc.create('exemple_sheet')  
worksheet = gc.open('exemple_sheet').sheet1  
  
# Remplir la feuille googlesheet créée  
cell_list = worksheet.range('A1:C2')  
  
import random  
for cell in cell_list:  
    cell.value = random.randint(1, 10)  
  
worksheet.update_cells(cell_list)
```

Pipeline basique d'un ML

Etape 1 – Data acquisition

On peut télécharger une feuille ([sheet1](#)) d'un fichier **google sheet**

```
# Télécharger des données d'une feuille googlesheet  
  
worksheet = gc.open('exemple_sheet').sheet1  
  
# Récupérer toutes les valeurs  
rows = worksheet.get_all_values()  
print(rows)  
  
pd.DataFrame.from_records(rows)
```


Pipeline basique d'un ML

Etape 1 – Data acquisition

Le fichier de données peut être issue de Sklearn, Kaggle ou Facebook ...

▼ C - Sklearn

```
[ ] from sklearn import datasets  
  
# Importation des données  
iris = datasets.load_iris()
```

D - Kaggle

```
[ ] # il faut commencer par installer kaggle  
  
!pip install -q kaggle  
  
from google.colab import files  
uploaded = files.upload()  
  
!mkdir ~/.kaggle1  
!cp kaggle.json /root/.kaggle1
```

E - Facebook

```
[ ] # From Facebook  
!pip install facebook-scraper  
  
from facebook_scraper import get_posts  
  
for post in get_posts('nintendo', pages=2):  
    print(post)
```

Pipeline basique d'un ML

Etape 2 – Data Exploration

2.1. Affichage des informations

```
[ ] import pandas as pd

data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data')
data
```

```
[ ] # Peaufinage de l'affichage des données

data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',
                  names = ['Longueur_sepale', 'Largeur_sepale', 'Longueur_petal', 'Largeur_petal', 'Classe'])
data
```

```
[ ] # Affichage des premières lignes

data.head()
```

```
[ ] # Affichage des dernières lignes

data.tail()
```

Pipeline basique d'un ML

Etape 2 – Data Exploration

```
[ ] # Forme de la matrice des données : lignes x colonnes ?  
  
data.shape
```

```
[ ] # Affichage d'une description sur les données  
  
data.describe()
```

```
[ ] # Affichage des colonnes  
  
data.columns
```

```
[ ] # Affichage des valeurs de certaines colonnes  
  
data[['Longueur_sepale', 'Largeur_petal']]
```

```
[ ] # Affichage des types des champs  
  
data.dtypes
```

Pipeline basique d'un ML

Etape 2 – Data Exploration

On peut filtrer les données avec des conditions simples ou multiples

2.2. Filtrage des données

```
[ ] # Filtrage avec condition simple  
data[data['Longueur_sepale'] < 5 ]
```

```
[ ] # Filtrage avec conditions multiples : et ( & ), ou ( | )  
data[ (data['Largeur_sepale'] > 4.0) & (data['Largeur_petal'] < 2 ) ]
```

Pipeline basique d'un ML

Etape 2 – Data Exploration

On peut supprimer certaines données (lignes ou colonnes)

2.3. Suppression de certaines informations

```
[ ] # Suppression de certaines colonnes (axis=1) sur place (inplace=True)
    data.drop(['Longueur_sepale', 'Longueur_petal'], axis=1, inplace=True)
```

```
[ ] # Données après suppression de certaines colonnes

data
```

```
[ ] # Suppression de certaines lignes (axis=0) sur place (inplace=True)
    data.drop([0,2], axis=0, inplace=True)
    data.reset_index()
```

Pipeline basique d'un ML

Etape 2 – Data Exploration

On peut visualiser les données via matplotlib

2.4. Visulation graphique

```
[ ] data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',
                        names = ['Longueur_sepale', 'Largeur_sepale', 'Longueur_petal', 'Largeur_petal', 'Classe'])

# Visualisation avec matplotlib
# Visualisation de certains attributs dans le même graphique
import matplotlib.pyplot as plt

data[['Longueur_petal', 'Largeur_petal']].plot()
```

```
[ ] # Visualiation d'un attribut en fonction d'un autre

data.plot(x="Longueur_petal", y="Largeur_petal", style="o")
```

Pipeline basique d'un ML

Etape 2 – Data Exploration

D'autres opérations sont possibles sur les données

```
[ ] # Comptabilisation des valeurs uniques d'une colonne
```

```
data['Longueur_sepale'].nunique()
```

```
[ ] # Affichage du nombre d'échantillons selon les différentes valeurs d'une colonne
```

```
data['Longueur_sepale'].value_counts()
```

```
[ ] # Affichage du nombre d'échantillon pour chaque attribut ayant la même valeur désignée
```

```
data.groupby(by='Longueur_sepale').count()
```

```
[ ] # Visualisation graphique de nombres de chaque valeur d'une colonne
```

```
data['Longueur_sepale'].value_counts().plot()
```

Pipeline basique d'un ML

Etape 2 – Data Exploration

D'autres opérations sont possibles sur les données

```
[ ] # Visualisation graphique en histogramme de nombres de chaque valeur d'une colonne  
data['Longueur_sepale'].value_counts().plot.hist()
```

```
[ ] # Renommage d'une colonne  
data.rename(columns={'Classe':'Label'}, inplace=True)  
data
```

```
[ ] # Tri des lignes selon la valeur d'une colonne  
data.sort_values('Longueur_petal', ascending=True)  
data
```


Pipeline basique d'un ML

Etape 3 – Model Processing

Les données doivent être disponibles pour commencer le ML

3.1. Etape de base

```
[ ] import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',
                  names = ['Longueur_sepale', 'Largeur_sepale', 'Longueur_petal', 'Largeur_petal', 'Classe'])
```

Pipeline basique d'un ML

Etape 3 – Model Processing

Ensuite passer à répartir les données en attributs (features ou les x) et label (classe ou y)

3.2. Répartition des données

```
[ ] # Définition de la partie attribut

features = data[['Longueur_sepale', 'Largeur_sepale', 'Longueur_petal', 'Largeur_petal']]

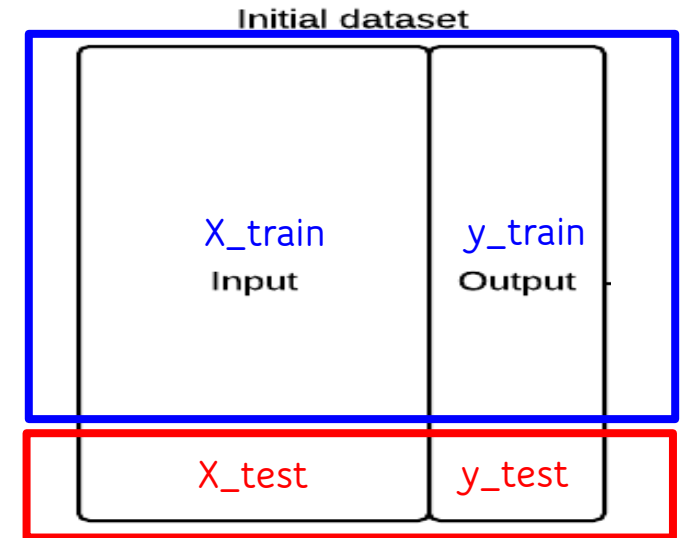
# Définition de l'objectif (target= label qui est dans notre cas Classe )

labels = data.Classe
```

Pipeline basique d'un ML

Etape 3 – Model Processing

Par la suite, les répartir en parties d'apprentissage (X_train et y_train) et de partie de test (X_test et y_test)



```
[ ] from sklearn.model_selection import train_test_split

# Répartition des données en partie d'apprentissage et partie de test et ce au niveau des fatures et des labels

X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=0)

# random_state doit prendre une valeur exacte pour avoir à chaque fois la même répartition (split)
```

Pipeline basique d'un ML

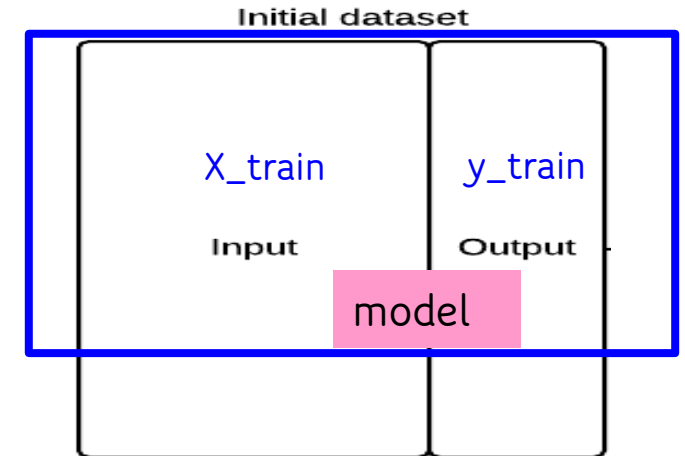
Etape 3 – Model Processing

Commencer par l'apprentissage en précisant le **modèle** de ML

```
[ ] # Importation du modèle
    from sklearn.naive_bayes import GaussianNB

    # Instantiation du modèle
    model = GaussianNB()

    # Apprentissage du modèle
    model.fit(X_train, y_train)
```



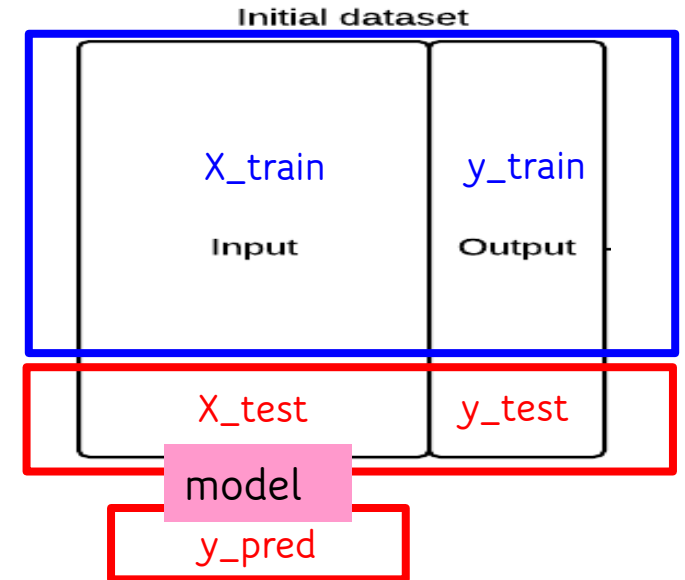
Pipeline basique d'un ML

Etape 3 – Model Processing

Ensuite faire la **prédiction** de la classe de la partie du test X_{test}

```
[ ] # Prédiction des labels à partir des échantillon de test
y_pred = model.predict(X_test)

y_pred
```



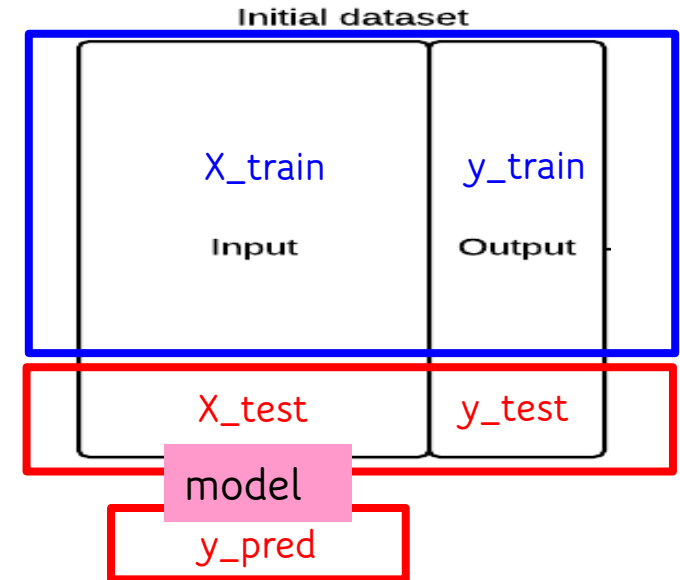
Pipeline basique d'un ML

Etape 3 – Model Processing

Enfin évaluer la **prédiction** réalisée selon les métriques choisies

```
[ ] # Calcul de la métrique accuracy
```

```
from sklearn.metrics import accuracy_score  
accuracy_score(y_test, y_pred)
```



Pipeline basique d'un ML

Etape 4 – Deployment

Le déploiement se fait selon le principe dump-load

```
[ ] # pour sauvegarder le modèle réalisé

from joblib import dump
dump(model, 'model_final.joblib') # format .joblib qui est échangé
```

```
[ ] ### code à insérer dans la partie web, mobile, embarquée, ... qui utilise le langage python

### code
from joblib import load

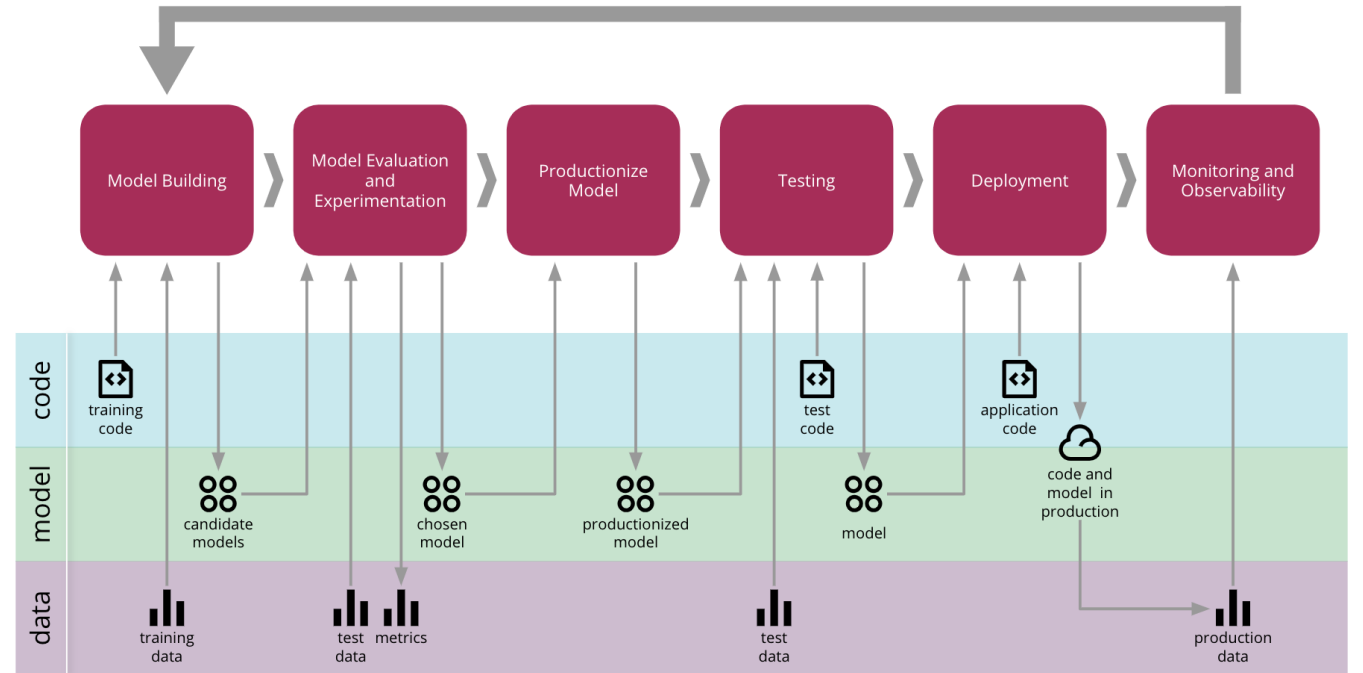
model = load('model_final.joblib')

input_data = get_data() # code à définir aussi

return model.predict(input_data)

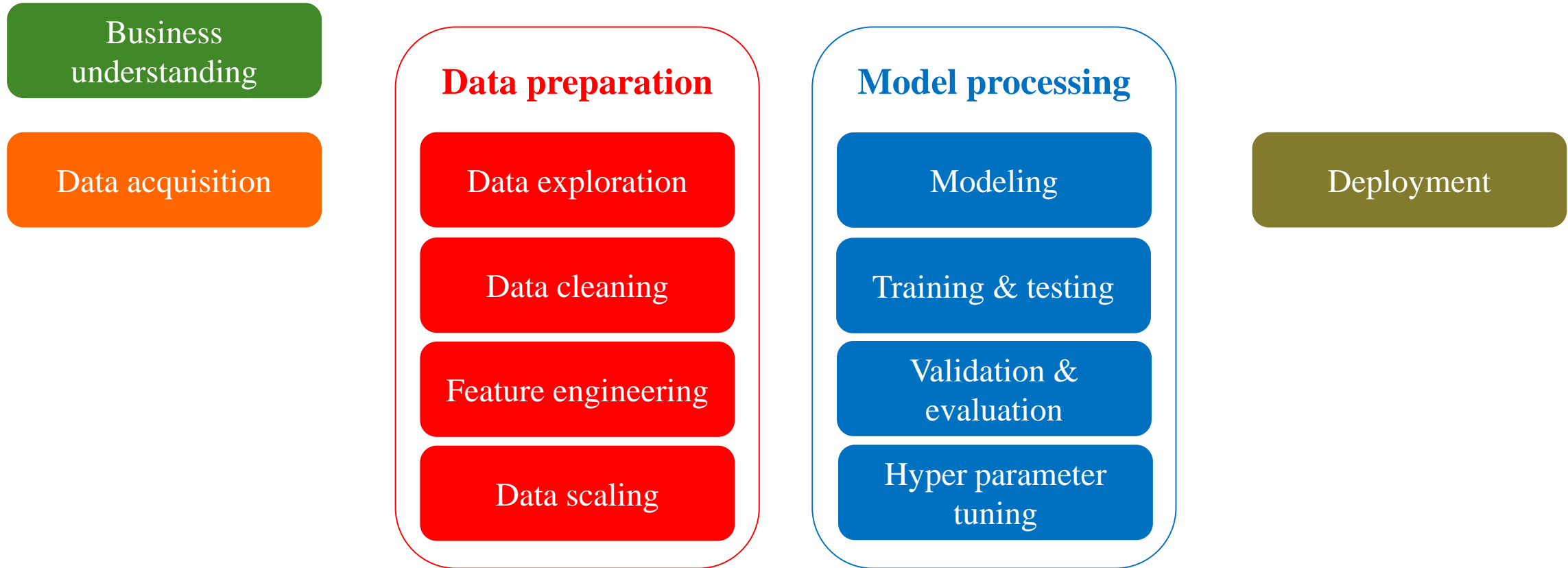
### code
```

Pipeline avancé d'un ML



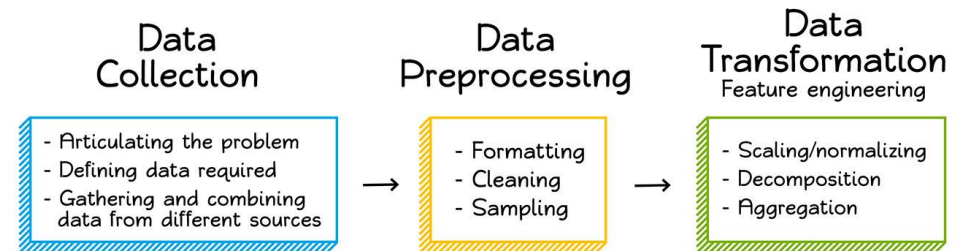
Pipeline avancé d'un ML

Étapes



Préparation des données

Data Preparation Process



Préparation des données

Tâches

Data preparation

Data exploration

Feature engineering

Data scaling

Data cleaning

Lien colab : https://colab.research.google.com/drive/10NVvOciCzE3tYJzdT7pEsu2_WjTy76p?usp=sharing

Préparation des données

1) Réduction des attributs

Matrice de corrélation

Pour analyser l'importance des attributs nous pouvons nous servir de la **matrice de corrélation**

Il faut alors la visualiser via seaborn et ce via `data.corr()`

Corrélation entre les données

```
[ ] # Affichage de la matrice de corrélation entre Tous les attributs (label inclus)

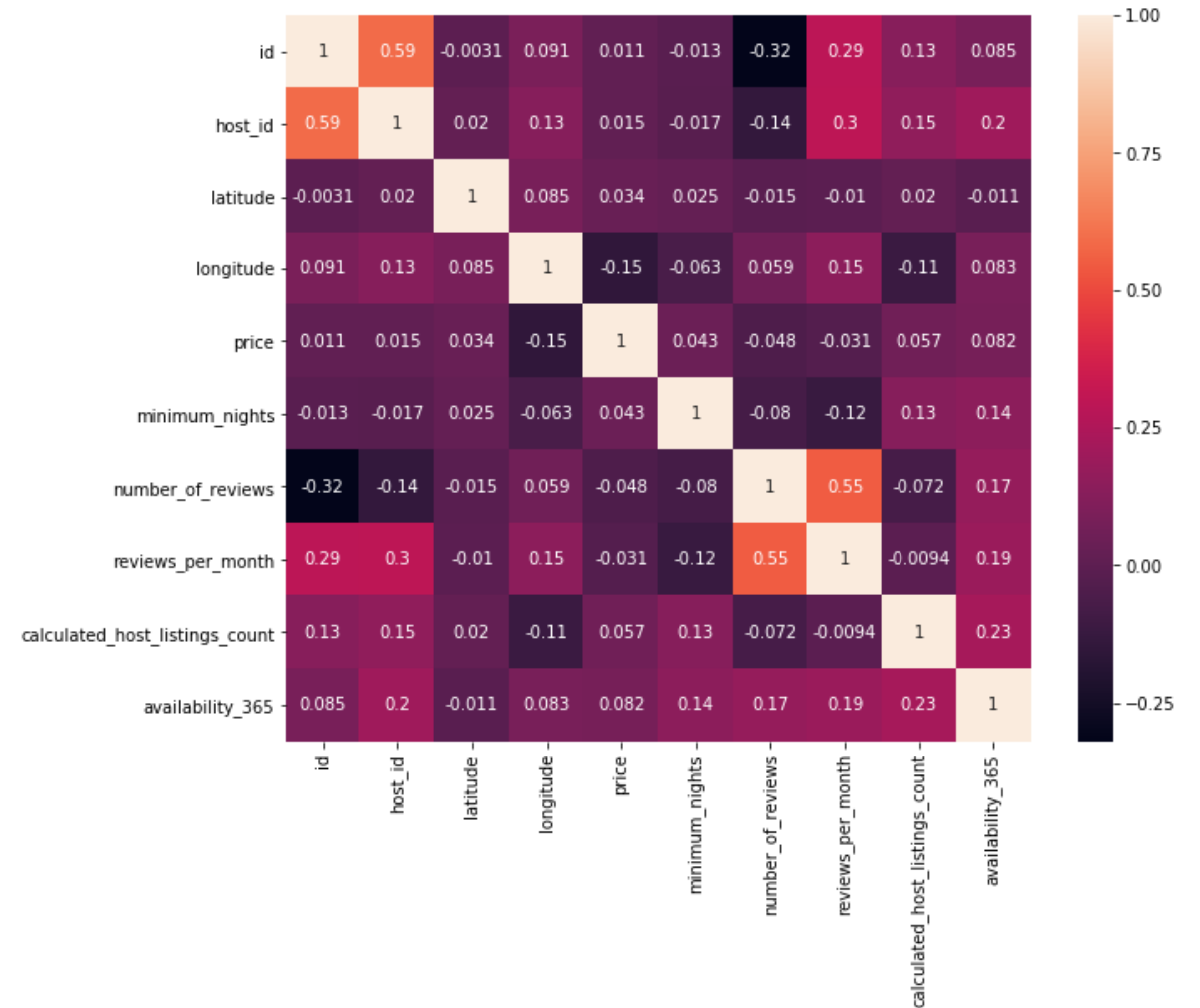
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(20,20))
sns.heatmap(data.corr())
```

Préparation des données

1) Réduction des attributs

Matrice de corrélation

Ensuite analyser les liens entre les données selon les couleurs sur l'échelle et ne retenir que ceux qui ont une valeur de $|\text{corrélation}| > 0.7$



Préparation des données

1) Réduction des attributs

Importance des attributs

On peut avoir l'importance des attributs avec `features_importances_` (sur les colonnes) qui marche seulement pour l'arbre de décision et prendre les attributs qui ont les plus grandes valeurs

Importance des attributs

```
[ ] # Pour voir l'importance des attributs en ce qui concerne le label
    # ça marche seulement pour le modèle de "DecisionTreeClassifier"

col_importance = zip(DT.feature_importances_, features.columns)
sorted_col_imp = sorted(list(col_importance), reverse=True)
sorted_col_imp

[(0.5702426323519294, 'MajorSubsystemVersion'),
 (0.15342437982883572, 'Subsystem'),
 (0.12642127694787397, 'MajorLinkerVersion'),
 (0.05813826971622963, 'TimeStamp'),
 (0.011946037966796734, 'Checksum'),
 (0.0115947014929234, 'DllCharacteristics'),
 (0.007422714689262313, 'MinorOperatingSystemVersion'),
 (0.007371180807489712, 'ImageDirectoryEntrySecurity'),
 (0.006059082638705252, 'MajorOperatingSystemVersion'),
 (0.005668510871616618, 'DirectoryEntryExport'),
 (0.005626516262819393, 'ImageBase'),
 (0.0032480897761018273, 'SectionMinVirtualSize')]
```

Préparation des données

1) Réduction des attributs

Importance des attributs

Sinon on peut calculer les poids des attribut avec

eli5

Poids des attributs

```
[ ] # Valable pour tous les classifieurs pour savoir l'importance des attributs

!pip install eli5

import eli5

eli5.show_weights(DT)

/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning
```

Weight	Feature
0.5702	x39
0.1534	x44
0.1264	x25
0.0581	x19
0.0119	x42
0.0116	x45
0.0074	x36
0.0074	x76
0.0061	x35
0.0057	x71

Préparation des données

2) Mise à l'échelle des données (data scaling)

Si les valeurs des données ne sont pas sur les mêmes grandeurs (certaines trop grandes et d'autres trop petites) alors il faut faire la mise à l'échelle.

On peut appliquer :

la normalisation

Ou

la standardisation

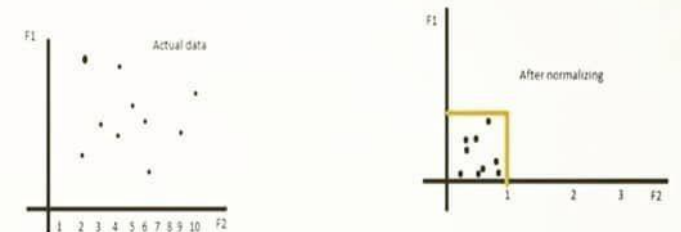
Ou

d'autres techniques

Feature Scaling

❑ Normalization

$$X_{changed} = \frac{X - X_{min}}{X_{max} - X_{min}}$$



❑ Standardization

$$X_{changed} = \frac{X - \mu}{\sigma}$$



Préparation des données

2) Mise à l'échelle des données (data scaling)

Normalisation

Avec le `MinMaxScaler`

Normalisation des données

```
[ ] # Choix du MinMaxScaler pour la standardisation des valeurs des données

from sklearn.preprocessing import MinMaxScaler
# Il y a aussi StandardScaler pour la standardisation des valeurs des données

sc = MinMaxScaler()

features = sc.fit_transform(features)

features
```

Préparation des données

2) Mise à l'échelle des données (data scaling)

Standardisation

Avec le `StandardScaler`

```
from sklearn.preprocessing import StandardScaler  
  
sc = StandardScaler()  
  
features = sc.fit_transform(features)
```

Préparation des données

3) Encodage des données (data encoding)

C'est transformer les valeurs catégoriques en valeurs numériques.
Il y a principalement deux techniques pour le faire :

Label Encoding : transforme le nom de la catégorie d'une classe en une valeur numérique et ce selon le nombre total des classes



Color
Red
Red
Yellow
Green
Yellow

Color
0
0
1
2
1

Préparation des données

3) Encodage des données (data encoding)

C'est transformer les valeurs catégoriques en valeurs numériques.
Il y a principalement deux techniques pour le faire :

Label Encoding : transforme le nom de la catégorie d'une classe en une valeur numérique et ce selon le nombre total des classes

```
# Option 1 : LabelEncoder

from sklearn.preprocessing import LabelEncoder

LE = LabelEncoder()

for column in data.columns:
    data[column] = LE.fit_transform(data[column])

data
```

Préparation des données

3) Encodage des données (data encoding)

C'est transformer les valeurs catégoriques en valeurs numériques.

Il y a principalement deux techniques pour le faire :

One hot encoding : transforme le nom de la catégorie d'une classe en un ensemble des classes dont la valeur est binaire (vrai ou non)

Color		Red	Yellow	Green
Red		1	0	0
Red		1	0	0
Yellow		0	1	0
Green		0	0	1
Yellow		0	0	1

Préparation des données

3) Encodage des données (data encoding)

C'est transformer les valeurs catégoriques en valeurs numériques.
Il y a principalement deux techniques pour le faire :

One hot encoding : transforme le nom de la catégorie d'une classe en un ensemble des classes dont la valeur est binaire (vrai ou non)

```
# Option 2 : One hot encoding

one_hotted = pd.get_dummies(data, columns=['label'])
one_hotted
```

Préparation des données

4) Données manquantes (messy data)

Il faut détecter s'il y a des valeurs manquantes avant de lancer le ML

Et ce à l'aide de `isna()`

On peut les comptabiliser en total avec `sum()`

Test des valeurs manquantes

```
[ ] data.isna().sum()
```

Suburb	0
Address	0
Rooms	0
Type	0
Price	0
Method	0
SellerG	0
Date	0
Distance	0
Postcode	0
Bedroom2	0
Bathroom	0
Car	62
Landsize	0
BuildingArea	6450
YearBuilt	5375
CouncilArea	1369
Latitude	0
Longitude	0
Regionname	0
Propertycount	0
dtype:	int64

Préparation des données

4) Données manquantes (messy data)

Nous pouvons choisir lune des solution suivantes

- Suppression des lignes/colonnes à valeurs manquantes
- Remplacement des valeurs manquantes

Préparation des données

4) Données manquantes (messy data)

Nous pouvons choisir l'une des solutions suivantes

- Suppression des lignes/colonnes à valeurs manquantes avec `dropna()`

```
# "dropna" supprime ligne ou colonne s'il y a des valeurs NaN
# Si Axis=0, on va parcourir ligne par ligne,
# Si Axis=1, on va parcourir colonne par colonne
# Si how="all", on va supprimer ligne/colonne où toutes les valeurs sont NaN
# Si how="any", on va supprimer ligne/colonne même s'il y a une seule valeur NaN

data.dropna(how='any', axis=0, inplace=True)
```

Préparation des données

4) Données manquantes (messy data)

Nous pouvons choisir lune des solution suivantes

- Remplacement des valeurs manquantes avec `fillna()` qui précise pour chaque champs manquant la valeur à mettre

```
[ ] # "fillna" remplace les valeurs NaN par une valeur au choix

data.fillna({
    'age': 40,
    'ch2': 10,
    'ch3': 20,
    'ch4': 'Hi'
})
```

Préparation des données

4) Données manquantes (messy data)

Nous pouvons choisir lune des solution suivantes

- Remplacement des valeurs manquantes avec `SimpleImputer()` qui va remplacer le champs manquant selon une stratégie à préciser (mean, median, constant, most_frequent)

```
from sklearn.impute import SimpleImputer
import numpy as np

imp = SimpleImputer(missing_values=np.nan, strategy="mean")
data.chp2 = imp.fit_transform(data[['ch1', 'chp3']])
```

Peaufinage du modèle de ML



Peaufinage du modèle de ML

Étapes

Model processing

Modeling

Training & testing

Validation &
evaluation

Hyper parameter
tuning

Lien colab :

<https://colab.research.google.com/drive/1onaJrU2J0qkYV6GAK3bu0yayG6zav73a?usp=sharing>

Peaufinage du modèle de ML

1) Comparaison de plusieurs modèles de ML

Nous pouvons comparer plusieurs modèles de ML
et choisir dans la partie déploiement le meilleur

```
[>] from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

RF = RandomForestClassifier()
KNN = KNeighborsClassifier()
DT = DecisionTreeClassifier()
SVM = SVC()
LR = LogisticRegression(solver="liblinear")

models = [RF, KNN, DT, SVM, LR]

for model in models:
    model.fit(X_train, y_train)
    print(f"***** {model} *****")
    print("-----")
    print('Score Train : ', "{:.2%}".format(model.score(X_train, y_train)))
    print('Score Test : ', "{:.2%}".format(model.score(X_test, y_test)))
    print("-----")
```

Peaufinage du modèle de ML

2) Validation croisée

La **validation croisée** (« cross-**validation** ») est une méthode d'estimation de fiabilité d'un modèle fondé sur une technique d'échantillonnage

	Dataset sans l'ensemble de test			
Mesure 1	Validation	Entrainement	Entrainement	Entrainement
Mesure 2	Entrainement	Validation	Entrainement	Entrainement
Mesure 3	Entrainement	Entrainement	Validation	Entrainement
Mesure 4	Entrainement	Entrainement	Entrainement	Validation

Qui consiste à décompose l'ensemble de données en **k-folds** (k parties) . Ensuite et à chaque itération prendre (k-1) échantillons pour l'apprentissage et le reste pour le test. De telle sorte que les parties des tests sont totalement différents. Enfin, chaque itération va donner une **mesure selon la métrique** choisie et entre ces différentes mesures nous pouvons faire la moyenne.

Peaufinage du modèle de ML

2) Validation croisée

La **validation croisée** (« cross-validation ») est une méthode d'estimation de fiabilité d'un modèle fondé sur une technique d'échantillonnage

```
from sklearn import model_selection
# Evaluation en validation croisée : 10 cross-validations
succes = model_selection.cross_val_score(KNN, features, label, cv=10, scoring='accuracy')
print(succes)
print("{:.2%}".format(succes.mean()))
```


Peaufinage du modèle de ML

3) Grille de recherche (Grid Search)

De nombreux algorithmes de ML reposent sur des paramètres qui ne sont pas toujours évidents à déterminer pour obtenir les meilleures performances sur un jeu de données à traiter. On peut donc utiliser la grille de recherche et énumérer les valeurs des paramètres

```
[ ] # Pour énumérer les valeurs des hyperparamètres à évaluer à chaque fois

from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

svm = SVC() #C , kernel et gamma

params_grid = {
    "C": [1, 10],
    "kernel": ['linear', 'rbf', 'poly'],
    "gamma": [0.01, 0.1]
}

grid = GridSearchCV(svm, params_grid , scoring="accuracy", cv=5)
grid.fit(X_train, y_train)
```

Peaufinage du modèle de ML

4) Assemblage (Ensembling)

Il y a trois techniques :

- Voting
- Bagging
- Boosting

Peaufinage du modèle de ML

4) Méthodes ensemblistes (Ensembling)

Voting

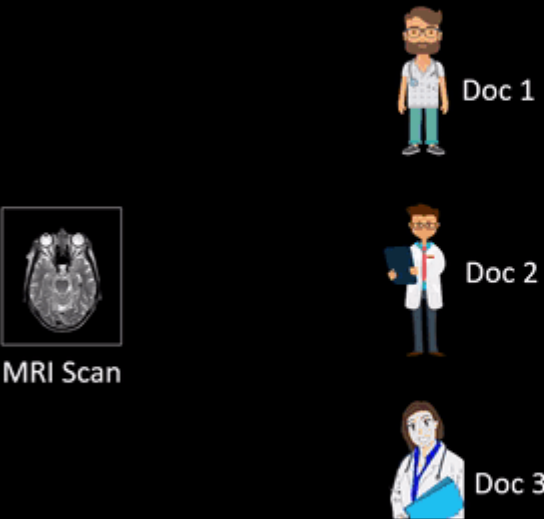
Il consiste à **sous-échantillonner** le training set et de faire générer à l'algorithme voulu **un modèle pour chaque sous-échantillon**.

On obtient ainsi un ensemble de modèles dont il convient de **moyenner** (lorsqu'il s'agit d'une régression) ou de **faire voter** (pour une classification) les différentes prédictions.

Les différents modèles sont lancés en parallèle.

Ensemble Learning Intuition

Real Life Example



The diagram illustrates the concept of ensemble learning using a real-life example. On the left, there is an image of an MRI scan labeled "MRI Scan". To the right of the MRI scan, three doctors are shown, each representing a model in the ensemble. They are labeled "Doc 1", "Doc 2", and "Doc 3". The doctors are depicted as cartoon characters: Doc 1 is a man with a beard and glasses, Doc 2 is a man with glasses holding a clipboard, and Doc 3 is a woman with a stethoscope. The background is dark blue. In the top right corner, there is a logo for "MLK MAKING AI SIMPLE". In the bottom left corner, there is a copyright notice: "© machinelearningknowledge.ai".

© machinelearningknowledge.ai

Peaufinage du modèle de ML

4) Méthodes ensemblistes (Ensembling)

Voting

```
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import VotingClassifier

SVM = SVC()
KNN = KNeighborsClassifier()
DT = DecisionTreeClassifier()

models = [('svm',SVM),('knn',KNN),('tree',DT)]
voting_result = VotingClassifier(estimators=models)
```

Ensemble Learning Intuition

Real Life Example

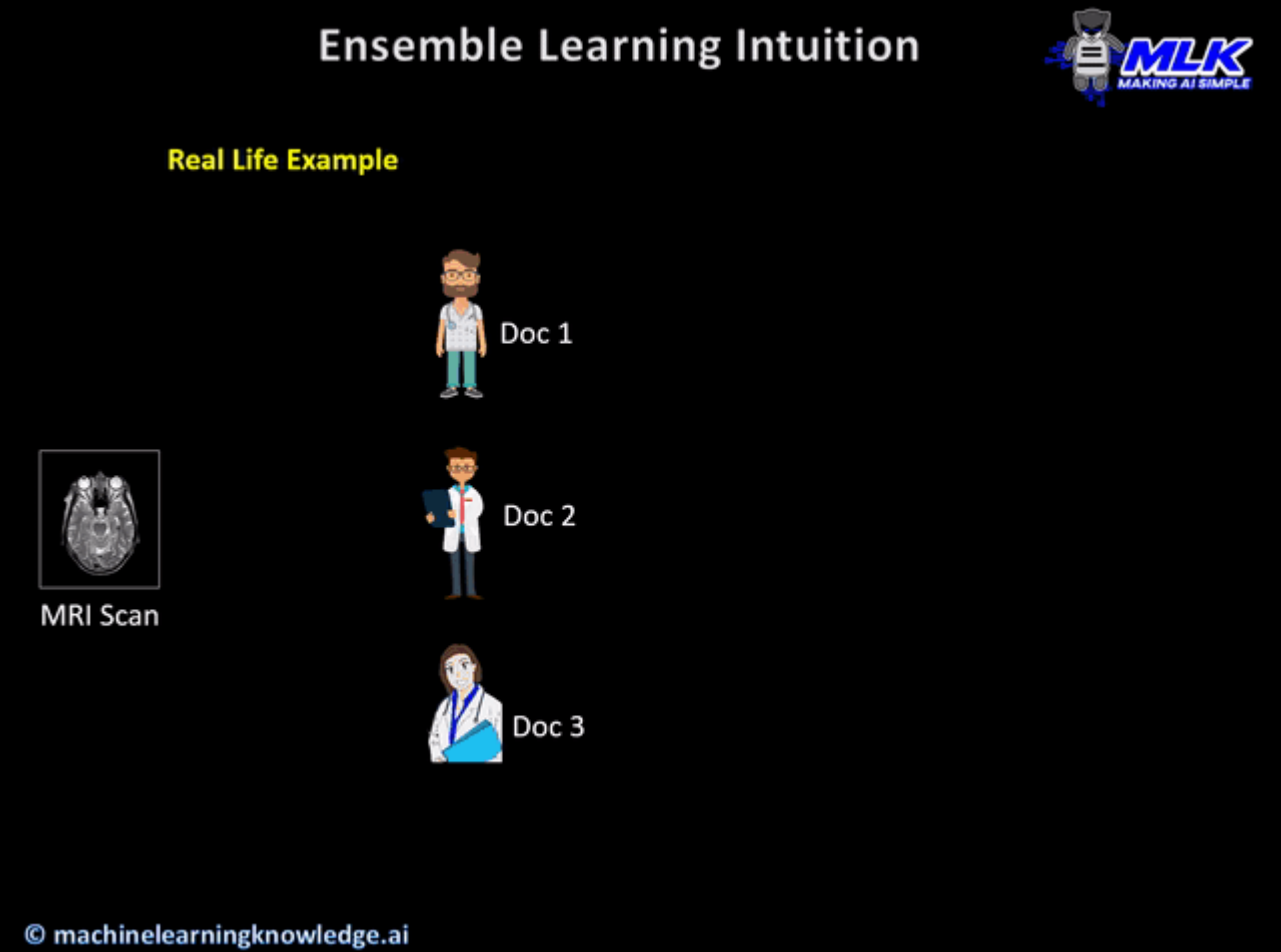
MRI Scan

Doc 1

Doc 2

Doc 3

© machinelearningknowledge.ai



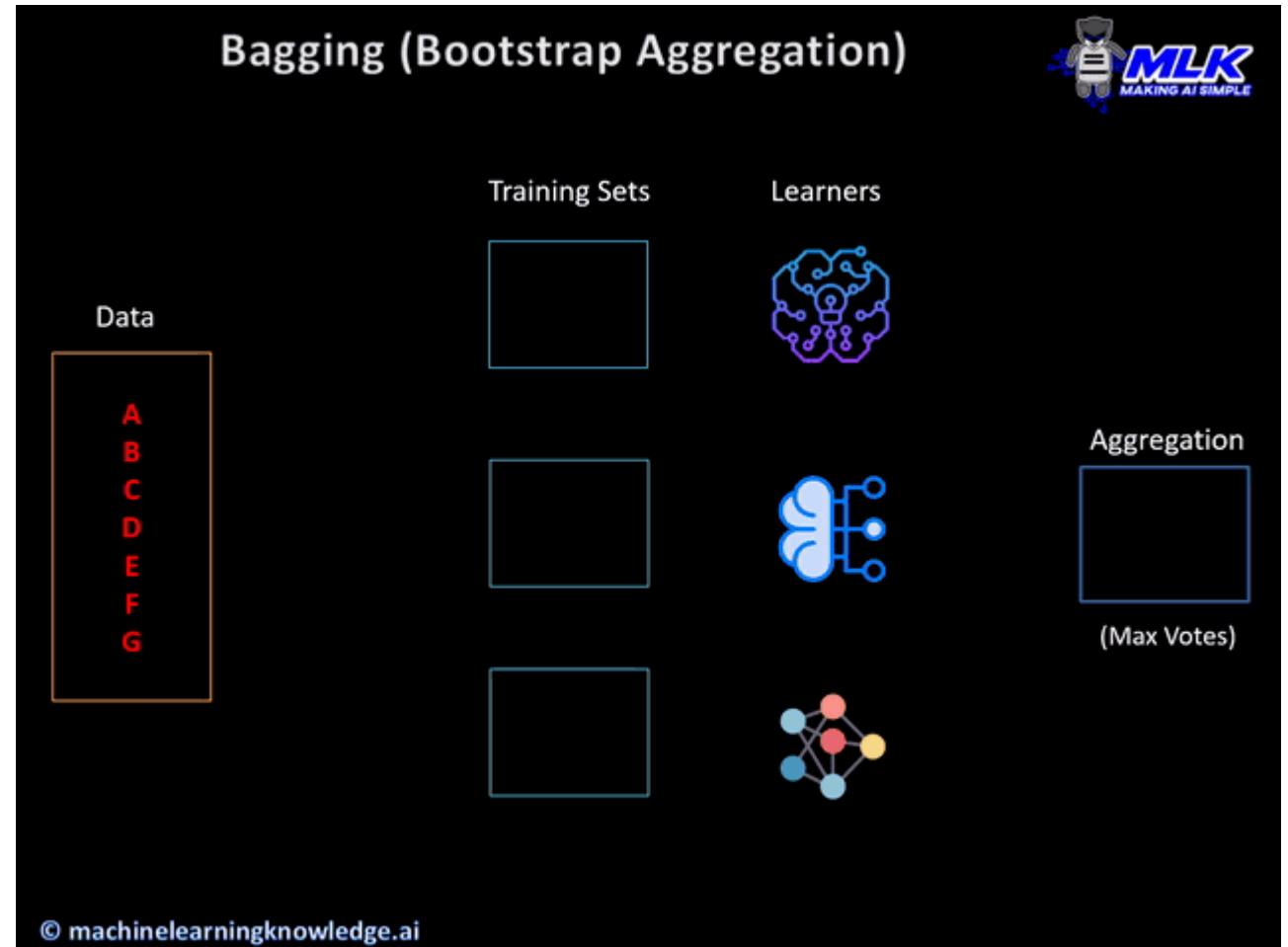
Peaufinage du modèle de ML

4) Méthodes ensemblistes (Ensembling)

Baggaging

On applique le même classifieur mais avec des paramètres et des données différents

Ainsi plusieurs classifieurs au total qu'il va falloir « bagger ». L'avantage principal de ces procédures est que la génération de ces modèles peut être naturellement **parallélisée**.



Peaufinage du modèle de ML

4) Méthodes ensemblistes (Ensembling)

Baggaging

Bagging

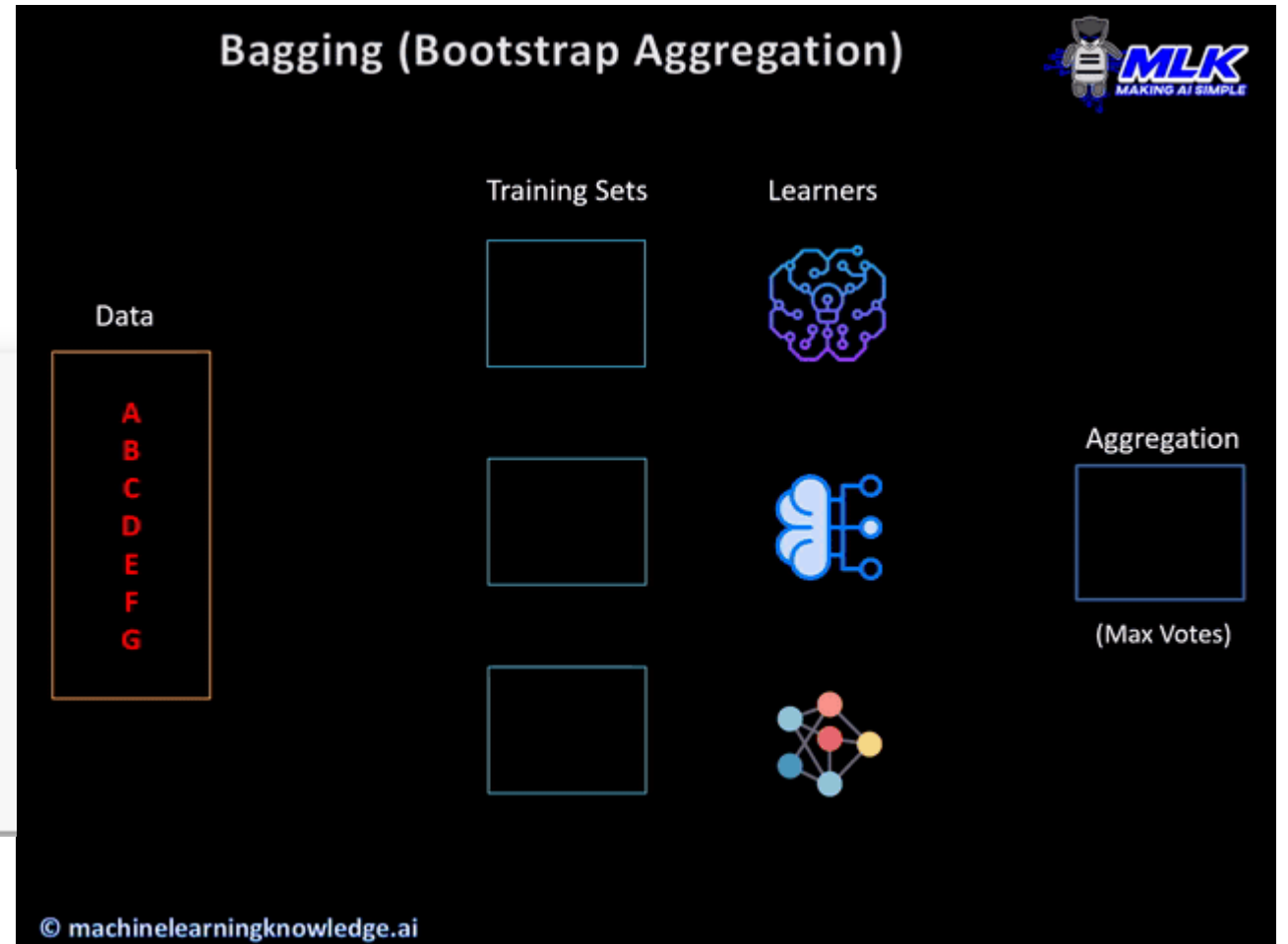
```
from sklearn.ensemble import RandomForestClassifier
```

```
RF = RandomForestClassifier()
```

```
RF.fit(X_train, y_train)
```

```
print('Score Train : ', "{:.2%}".format(RF.score(X_train, y_train)))
```

```
print('Score Train : ', "{:.2%}".format(RF.score(X_test, y_test)))
```

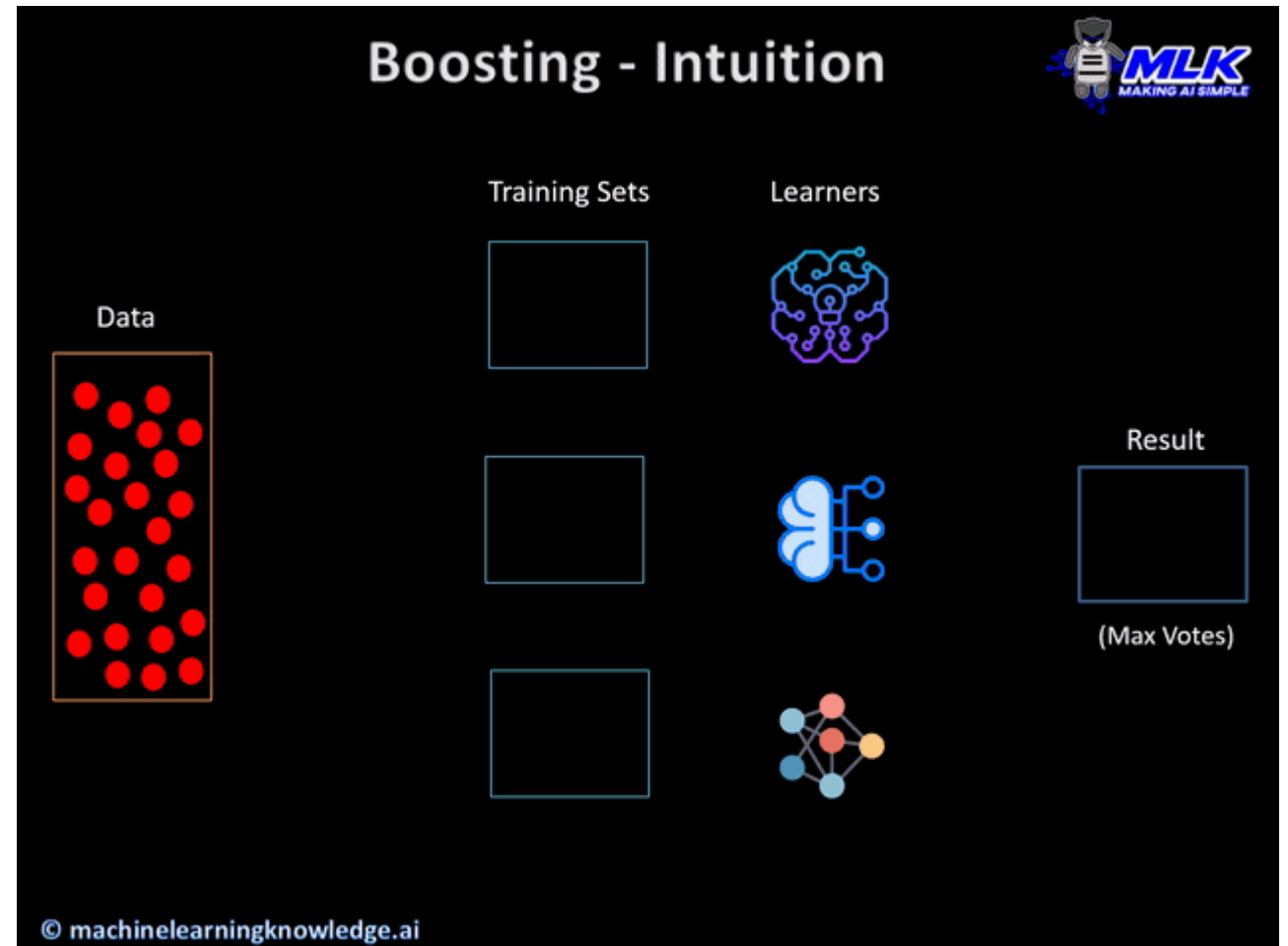


Peaufinage du modèle de ML

4) Méthodes ensemblistes (Ensembling)

Boosting

Le principe du boosting est quelque peu différent du bagging. **Les différents classifieurs sont pondérés** de manière à ce qu'à chaque prédiction, **les classifieurs ayant prédit correctement auront un poids plus fort** que ceux dont la prédiction est incorrecte.

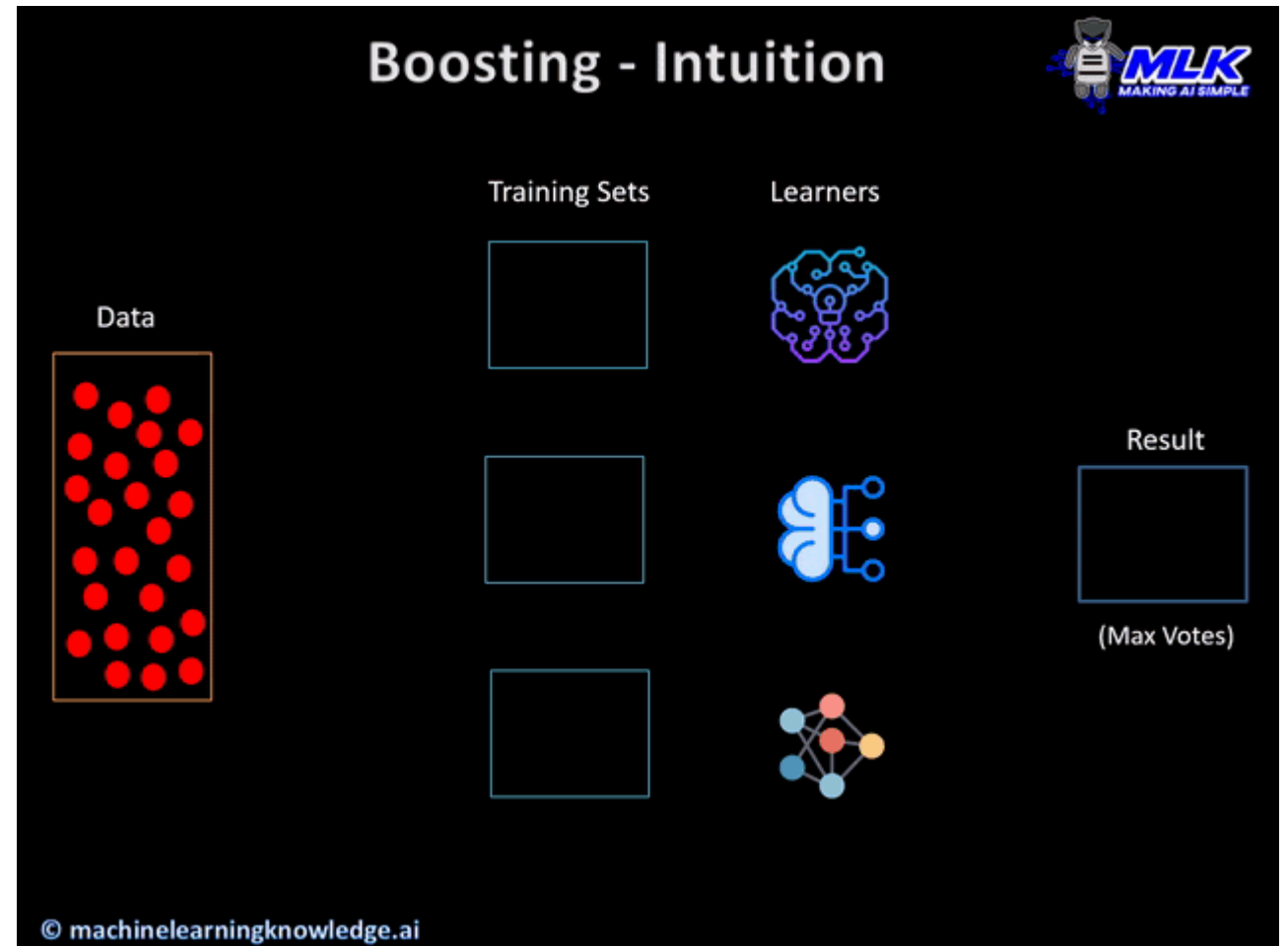


Peaufinage du modèle de ML

4) Méthodes ensemblistes (Ensembling)

Boosting

Adaboost : est un algorithme de boosting qui s'appuie sur ce principe, avec un paramètre de mise à jour adaptatif permettant de donner plus d'importance aux valeurs difficiles à prédire, donc en boostant les classifieurs qui réussissent quand d'autres ont échoué. Des variantes permettent de l'étendre à la classification multiclass. Adaboost s'appuie sur des classifieurs existants et cherche à leur affecter les bons poids vis à vis de leurs performances.

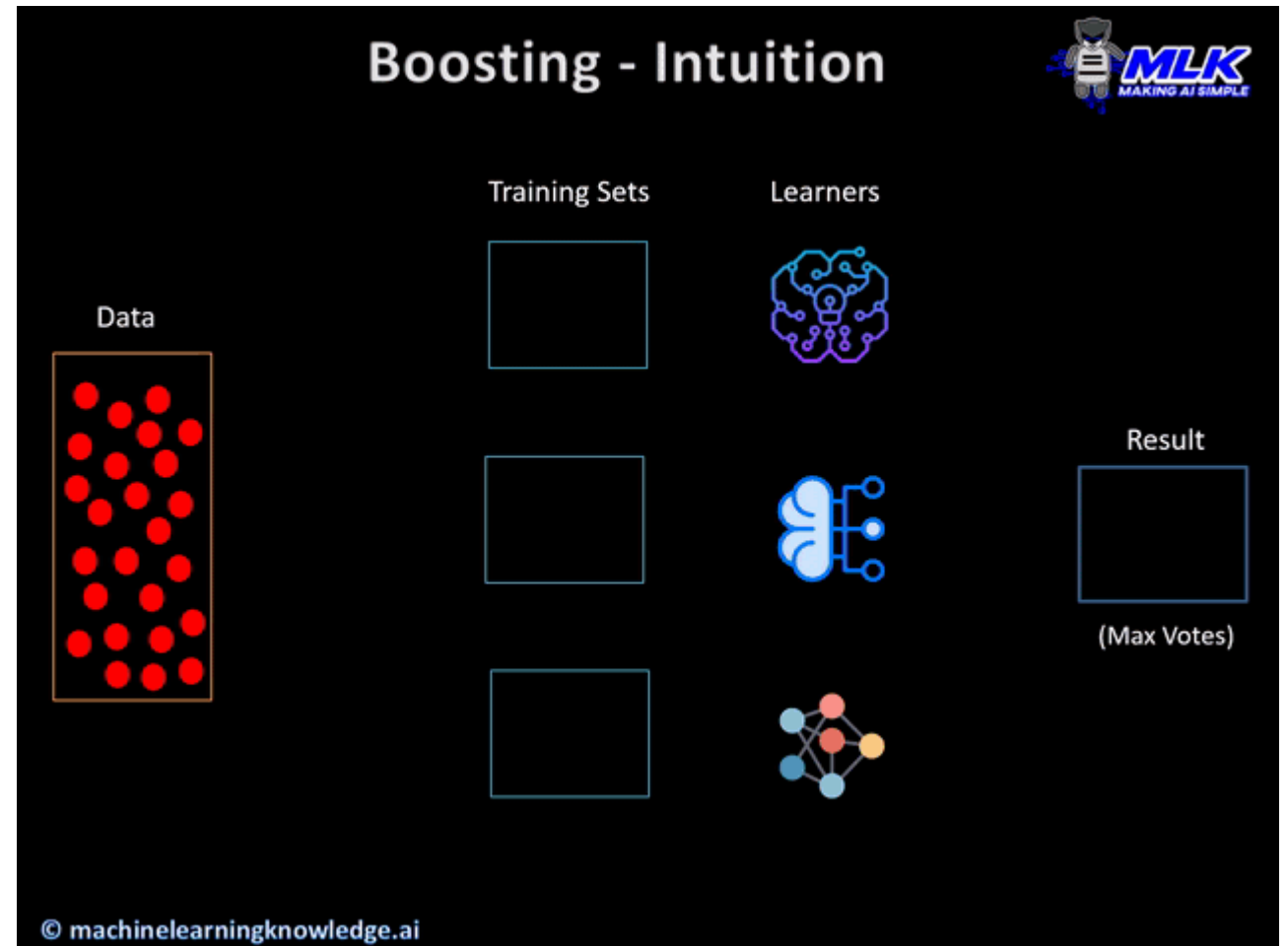


Peaufinage du modèle de ML

4) Méthodes ensemblistes (Ensembling)

Boosting

Gradient boost : est une technique de boosting qui est majoritairement employée avec des arbres de décision. L'idée principale est là encore d'agréger plusieurs classifieurs ensembles mais en les créant itérativement. Ces "mini-classifieurs" sont généralement **des fonctions simples et paramétrées**, le plus souvent des arbres de décision dont chaque paramètre est le critère de split des branches. Le super-classifieur final est **une pondération (par un vecteur w)** de ces mini-classifieurs.



Peaufinage du modèle de ML

4) Méthodes ensemblistes (Ensembling)

Boosting

Boosting

```
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier

ada_boost = AdaBoostClassifier()
GB         = GradientBoostingClassifier()

ada_boost.fit(X_train, y_train)
GB.fit(X_train, y_train)

print('***** AdaBoost *****')
print('Score Train : ', "{:.2%}".format(ada_boost.score(X_train, y_train)))
print('Score Train : ', "{:.2%}".format(ada_boost.score(X_test, y_test)))
print('***** Gradient Boost *****')
print('Score Train : ', "{:.2%}".format(GB.score(X_train, y_train)))
print('Score Train : ', "{:.2%}".format(GB.score(X_test, y_test)))
```

