Méthodes d'agrégation avec Python

Novembre 2024

Cédric Dangeard
cedric.dangeard@businessdecision.com

Cours 2 : Au programme

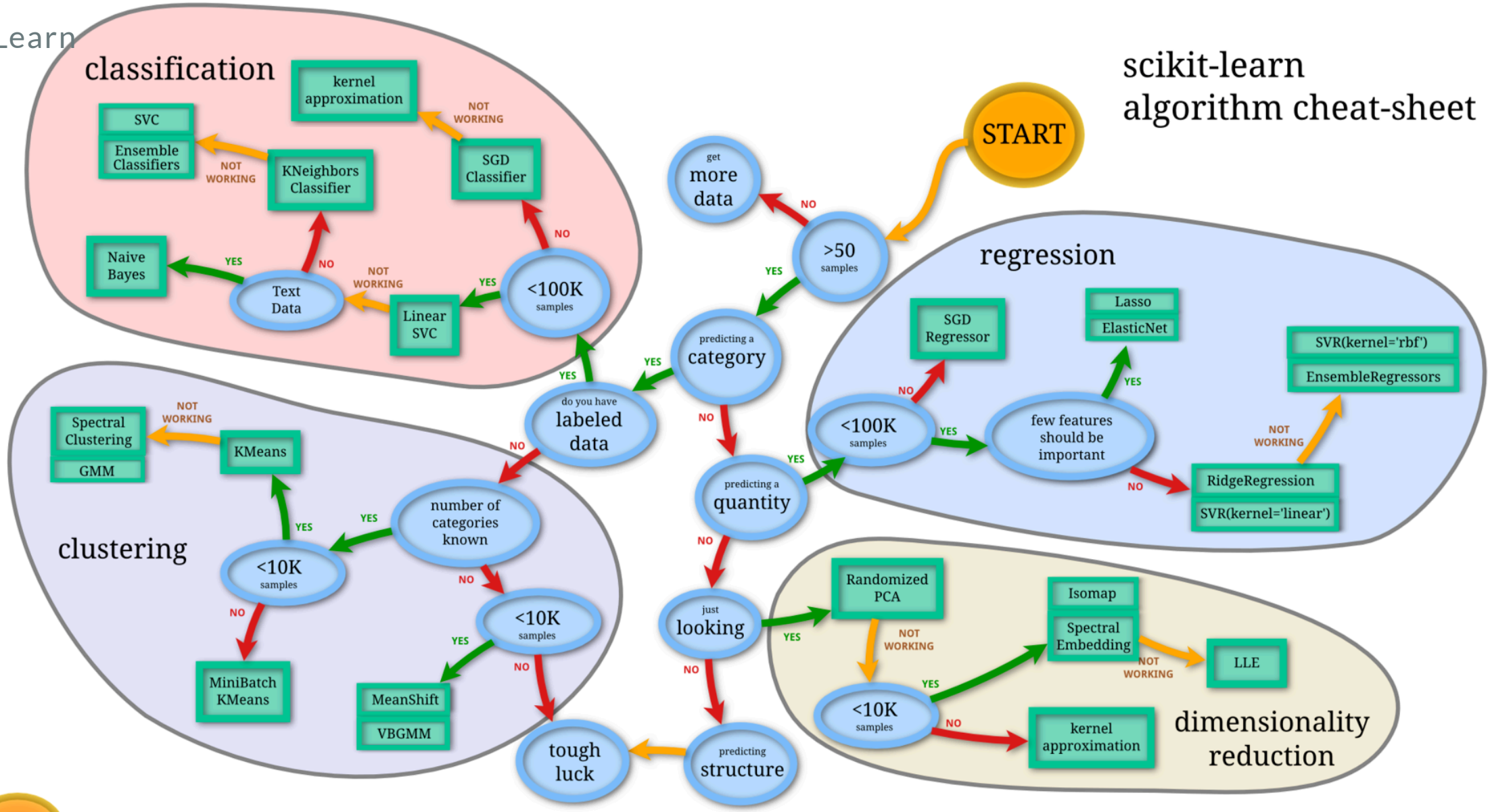
- Présentation des librairies principales
- Objets Sklearn
- Encodage des variables avec Sklearn et Pandas
- **TP : Encoding**
- Rappel des CARTS
- Rappel sur les méthodes d'aggregations
- **TP : Machine Learning**

Les Librairies Fondamentales

- Numpy
  - *alias :* Numerical Python
  - *objet de base :* numpy.array
  - *missions :* puissance et rapidité de calcul sur des vecteurs
  - *implémentation :* C
  - code, site
- Scipy
  - *alias :* Scientific Python
  - *objet de base :* numpy
  - *missions :* Algorithmes plus haut niveau, optimisation, regression, interpolation, équations différentielles, ...
  - *implémentation :* C, Fortran, C++, Cython
  - code, site

Scikit-Learn

- Librairie de machine Learning sur Python
- Libre et Open source
- Orientée Objets
- Documentée
- Communauté active
- code, site


scikit-learn
algorithm cheat-sheet
START
classification
kernel approximation
NOT WORKING
SVC
Ensemble Classifiers
NOT WORKING
KNeighbors Classifier
SGD Classifier
NO
Naive Bayes
YES
NO
NOT WORKING
YES
Text Data
Linear SVC
<100K samples
get more data
NO
>50 samples
YES
regression
SGD Regressor
Lasso
ElasticNet
SVR(kernel='rbf')
EnsembleRegressors
predicting a category
YES
NO
<100K samples
YES
few features should be important
YES
NOT WORKING
RidgeRegression
SVR(kernel='linear')
NO
YES
do you have labeled data
NO
predicting a quantity
YES
NO
clustering
Spectral Clustering
GMM
NOT WORKING
KMeans
YES
number of categories known
YES
<10K samples
NO
NO
MiniBatch KMeans
<10K samples
YES
NO
MeanShift
VBGMM
just looking
YES
Randomized PCA
NOT WORKING
Isomap
Spectral Embedding
NOT WORKING
LLE
YES
<10K samples
NO
kernel approximation
dimensionality reduction
NO
predicting structure
tough luck
Back
scikit learn

source

Objets Sklearn

- Estimateur
  - fit : Entraîner le modèle
  - transform : Transformer les données
  - predict : Prédire la variable cible
  - score : Évaluer les performances du modèle

```python
from sklearn.base import BaseEstimator, TransformerMixin

class MyEstimator(BaseEstimator, TransformerMixin):
    def __init__(self, param : int  = 1):
        self.param = 1

    def fit(self, X, y):
        self.is_fitted_ = True
        return self

    def transform(self, X):
        return X

    def predict(self, X):
        return np.full(shape=X.shape[0], fill_value=self.param)
```

Label Encoding

- Chaque modalité prend une valeur
- Peut être Ordinal
- Pas de nouvelles variables
- Perte d'information
- Que faire des valeurs manquantes ?

```python
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder

df = pd.DataFrame({
    'Name' : ['Sweet Mask', 'Bald Cape', 'Blizzard of Hell',
     'King', 'Glasses', 'Metal Bat', 'Mumen Rider'],
    'Class' : ['A', 'C', 'B', 'S', 'A', 'S', 'C']})

le = LabelEncoder()
oe = OrdinalEncoder(categories=[['S', 'A', 'B', 'C']])

df['label'] = le.fit_transform(df[['Class']])
df['ord'] = oe.fit_transform(df[['Class']])
```

Name	Sweet Mask	Bald Cape	Blizzard of Hell	King	Glasses	Metal Bat	Mumen Rider
Class	A	C	B	S	A	S	C
label	0	2	1	3	0	3	2
ord	1.0	3.0	2.0	0.0	1.0	0.0	3.0

One Hot Encoding

- Une nouvelle variable par modalité
- Chaque variable prend la valeur 0 ou 1
- Peut ajouter beaucoup de variables
- Deux façon de faire :
  - pandas.get_dummies
  - sklearn.OneHotEncoder

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(categories=[['S', 'A', 'B', 'C']],
                    drop = 'first',
                    sparse_output=False)

hot_encoded = ohe.fit_transform(df[['Class']])

data_hot_encoded = pd.DataFrame(hot_encoded, index=df.index)
data_hot_encoded.columns = ohe.get_feature_names_out()

pd.concat([df, data_hot_encoded], axis=1)
```

```
pd.concat([df,
  pd.get_dummies(df[['Class']], drop_first=True)], axis=1)
```

Name	Sweet Mask	Bald Cape	Blizzard of Hell	King	Glasses	Metal Bat	Mumen Rider
Class	A	C	B	S	A	S	C
Class_A	1.0	0.0	0.0	0.0	1.0	0.0	0.0
Class_B	0.0	0.0	1.0	0.0	0.0	0.0	0.0
Class_C	0.0	1.0	0.0	0.0	0.0	0.0	1.0

Target Encoding

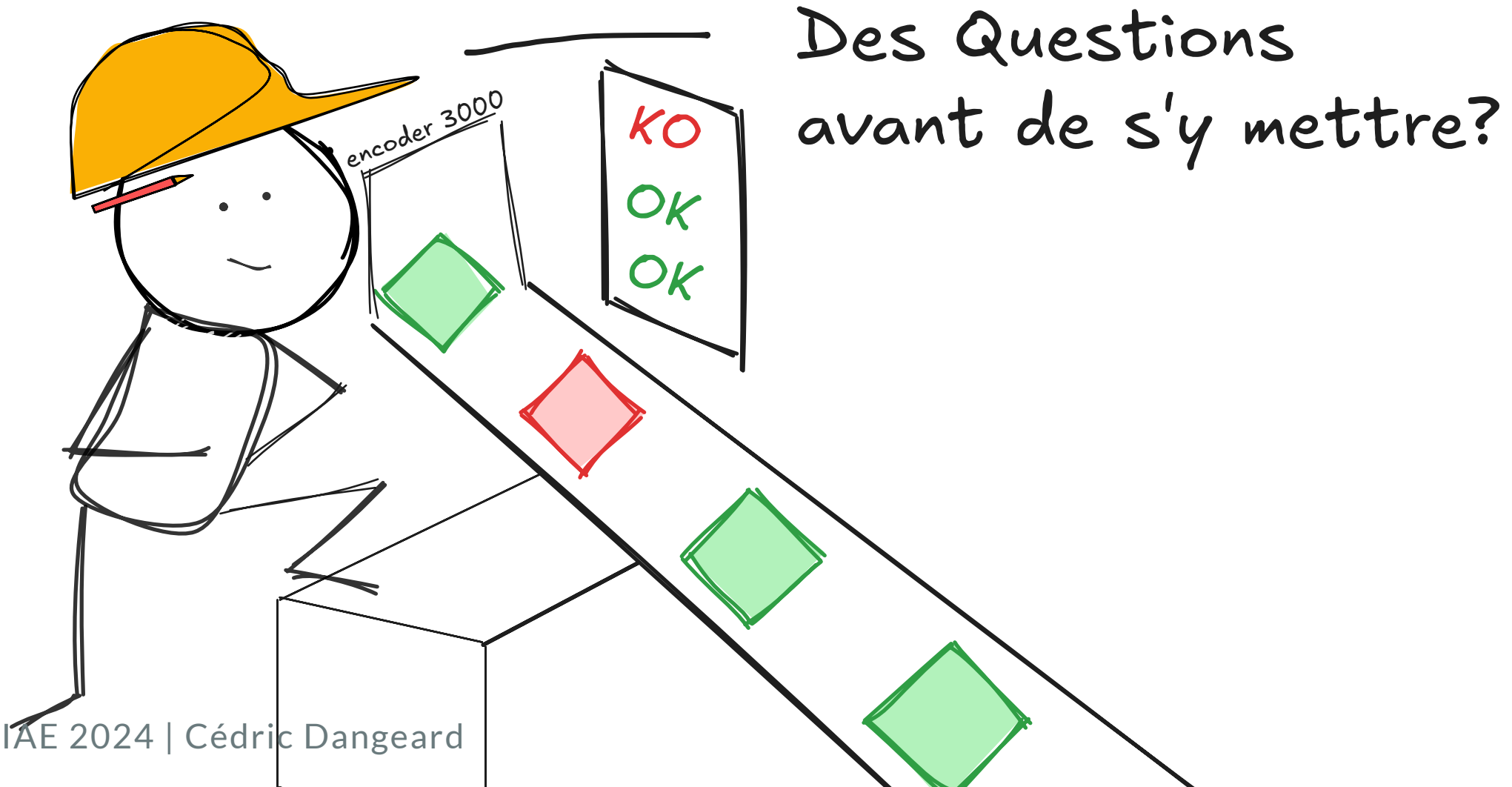
- Remplacer les modalités par la une valeur dérivé de la valeur à prédire (ex : la moyenne)
  - Peut créer du sur-apprentissage
  - A utiliser sur des modalités à hautes cardinalités

```
df = pd.DataFrame({
    'Name' : ['Le Bron','Kawamura', 'Wembanya', 'Durant',
              'Curry', 'Jokić', 'Paul', 'Davis'],
    'Taille' : [206, 173, 224, 211, 188, 211, 183, 208],
    'Club' : ['Lakers', 'Grizzlies', 'Spurs', 'Suns',
              'Warriors', 'Nuggets', 'Spurs', 'Lakers',]
})

df['encoded_Club'] = df.groupby('Club')['Taille'].transform('mean')
```

Name	Le Bron	Kawamura	Wembanya	Durant	Curry	Jokić	Paul	Davis
Taille	206	173	224	211	188	211	183	208
Club	Lakers	Grizzlies	Spurs	Suns	Warriors	Nuggets	Spurs	Lakers
encoded_Club	207.0	173.0	203.5	211.0	188.0	211.0	203.5	207.0

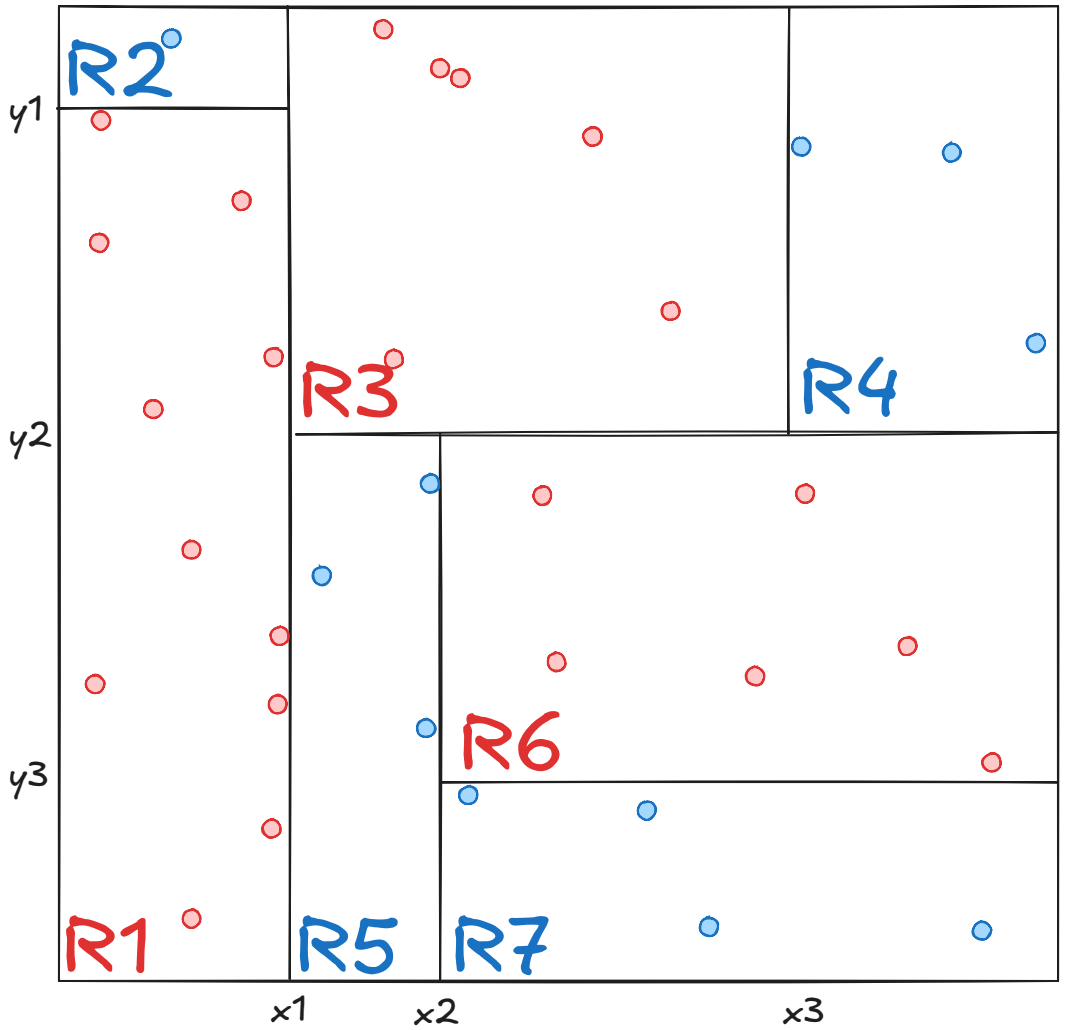
```
from sklearn.preprocessing import TargetEncoder

te = TargetEncoder(smooth='auto',
                   cv=2)
target_encoded = te.fit_transform(X=df[['category']], y = df[['target']])
```

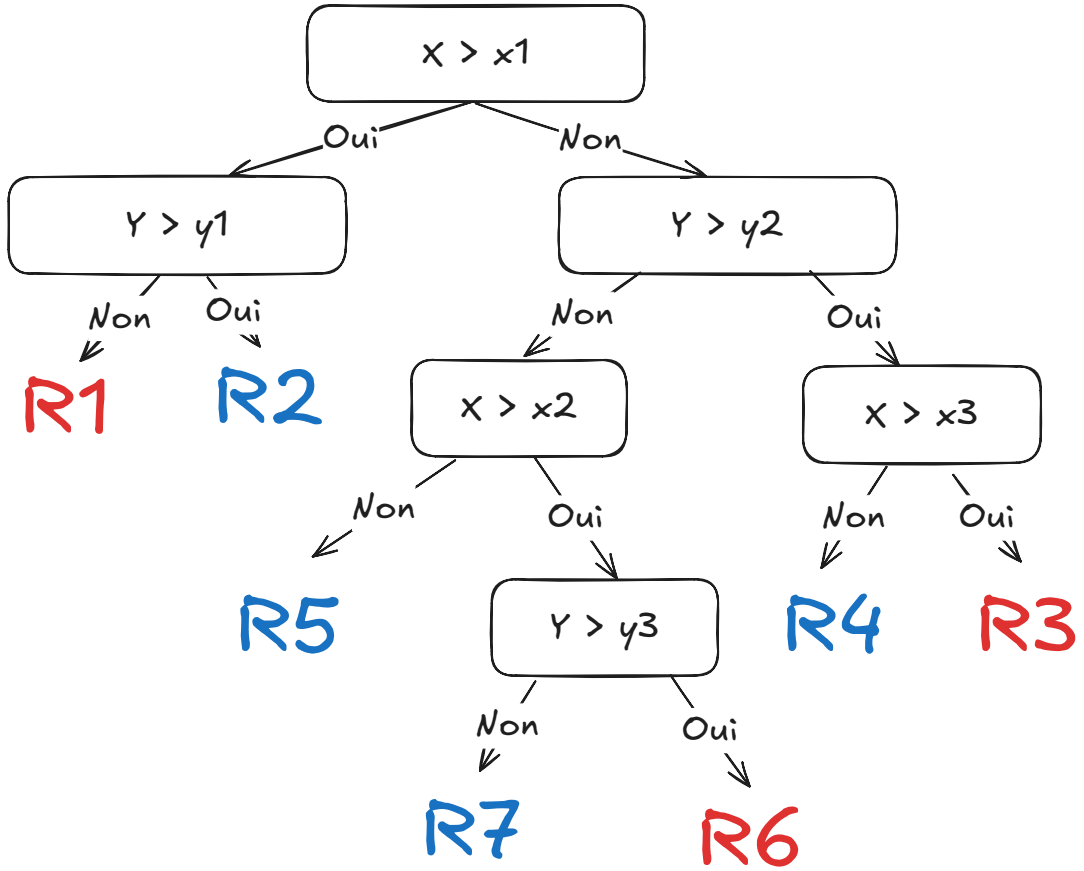
TP : Encoding



Des Questions
avant de s'y mettre?

Arbres de décisions





CART : L'algorithme

- Définir un critère d'homogénéité
  - CART : Indice de diversité de Gini
    $I_G = 1 - \sum_{i=1}^{m} f_i(1 - f_i)$
    où f_i est la proportion de classe i dans le jeu de données.
  - Autres : Entropie de Shanon, Gain d'information, ...
- Calculer ce critère pour un ensemble de segmentations des données.
- Choisir la segmentation qui minimise le critère.

CART : Où s'arreter ?

Un individu par feuille ?

Jusqu'à avoir des feuilles mono-classes ?

Taille des feuilles, nombre de feuilles ?

Profondeur de l'arbre ?

CART : Où s'arreter ?

- Un individu par feuille ?
- Une classe par feuille ?
- Taille des feuilles, nombre de feuilles ?
- Profondeur de l'arbre ?

Le choix de ce critère d'arrêt, va constituer l'un des hyperparamêtres à optimiser de notre modèle.

CART : Prunning

Plus un arbre est profond, plus la variance est élevée, et le biais est faible.

- Élagage *(Prunning)*
  - Pré-élagage :
    - Instaurer des règles d'arrêt pendant l'apprentissage
  - Post-élagage :
    - Partir d'un arbre profond et réduire la variance en supprimant des feuilles.

CART sur SkLearn : DecisionTreeClassifier

- Paramètres :
  - **criterion**
  - **max_depth**
  - **min_samples_split**
  - **min_samples_leaf**
  - **max_features**

CART sur SkLearn : DecisionTreeClassifier

- Paramètres :
  - **criterion** *[Gini, entropy, logloss]* : Criteres d'homogénité
  - **max_depth** *int* : Profondeur de l'arbre
  - **min_samples_split** *int* : Individus minimum dans la feuille pour procéder à un split.
  - **min_samples_leaf** *int* : Nombre d'individus minimum dans les feuilles filles pour accepter le split
  - **max_features** *[int, float, 'sqrt', 'log2', None]* : Nombre maximum de variables à tester pour créer un noeud

Arbres de décisions : Limites

- Facilement interprétable
- Variance élevé
- Sur-apprentissage pour les arbres trop profonds
- Peu performant en général

Solutions :

- Comment améliorer les performances des arbres ?
- Comment réduire la variance ?

Bootstrap Aggregating

Soit X_i un ensemble de variables indépendantes de même loi dont la variance est $\sigma^2 < \infty$. La variance de la moyenne est :

$$Var(\frac{\sum_{i=1}^{n} X_i}{n}) = \frac{1}{n^2}\sum_{i=1}^{n} Var(X_i) = \frac{1}{n^2} n\sigma^2 = \frac{\sigma^2}{n}$$

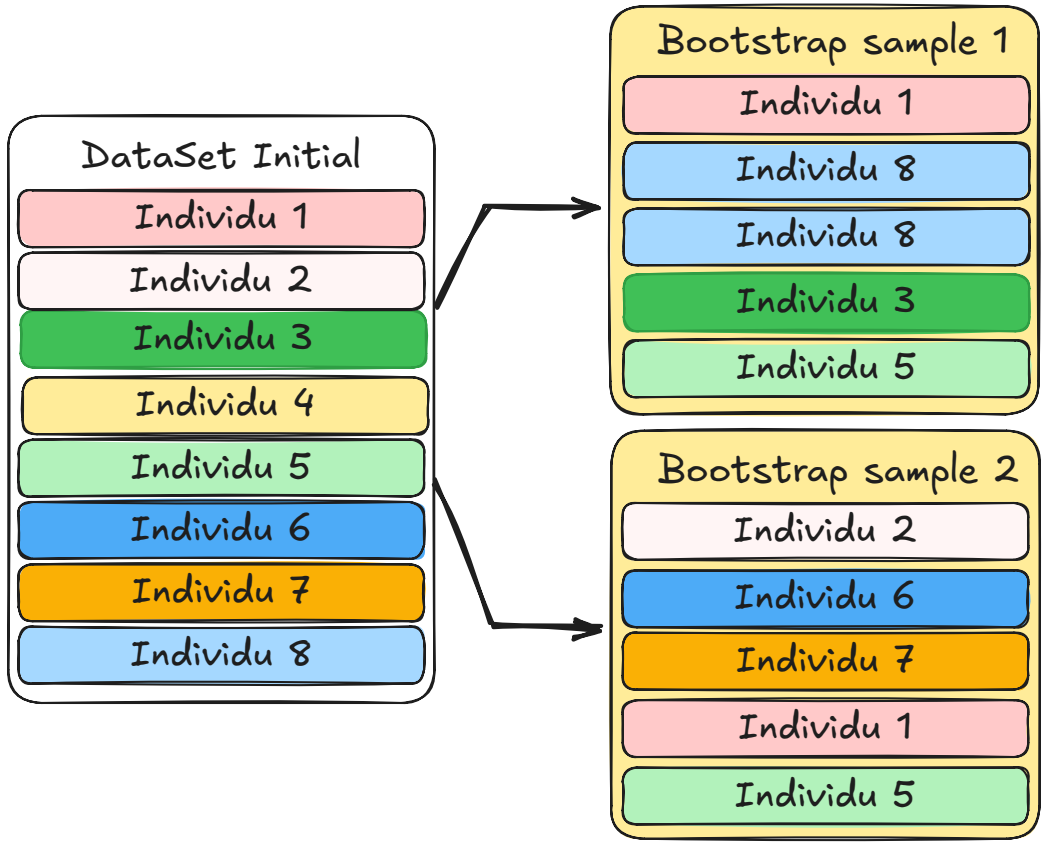
Dans le cas de variable correlés d'un facteur ρ

$$Var(\frac{\sum_{i=1}^{n} X_i}{n}) = \frac{1-\rho}{n}\sigma^2 + \rho\sigma^2$$

Bootstrap Sampling

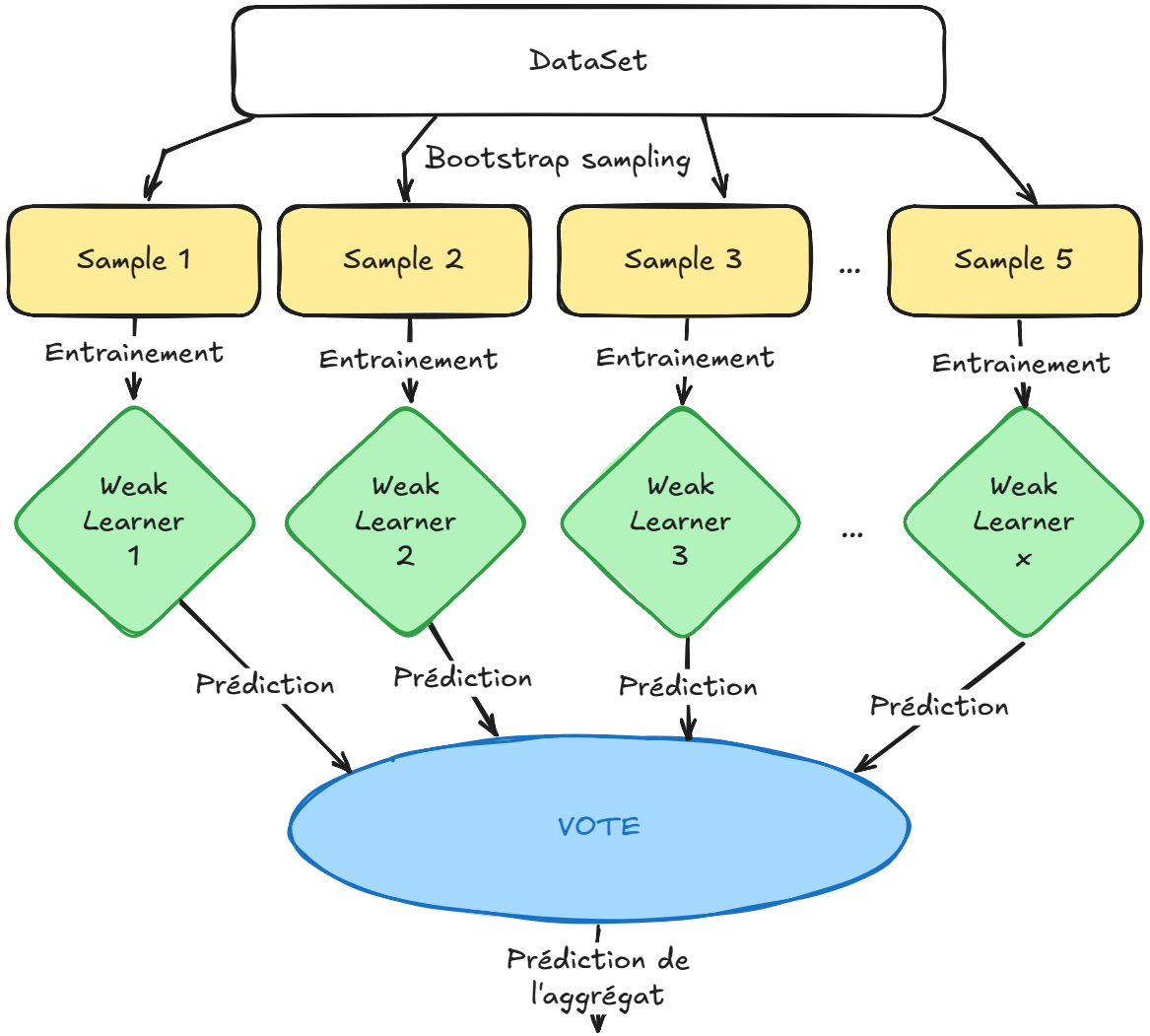
Problème :

- Il nous faut donc idéalement plusieurs jeux de données
  Solutions :
- Utiliser notre jeu de données pour produire plusieurs jeux de données : Bootstrap Sampling.



Aggregating

- Le bagging consiste donc à prendre la moyenne d'un ensemble de modèles
- On appelle en général ces modèles *Weak Learner*
- On peut utiliser tout types de modèles, idéalement des modèles faible biais /forte variance.



Bagging sur Sklearn : BaggingClassifier

- **max_features** / **max_samples** : *(float/int)*
  Nombre d'individus et de variables tirés avec remise
- **bootstrap_features** / **bootstrap** :*(bool)*:
  Tirage avec remise ou non des variables/données.
- **oob_score** *(bool)* : Si **bootstrap** erreur out of bag ou

```python
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier

#Definition du modèle
bagging = BaggingClassifier(
    base_estimator=KNeighborsClassifier(),
    n_estimators=10,
    max_samples=0.5,
    max_features=0.5,
    bootstrap=True,
    bootstrap_features=False,
    oob_score=False,
    n_jobs=-1,
)

# Entrainement du modèle
bagging.fit(X, y)
```

Random Forest

- Bagging avec arbres profonds comme modèles de base.
- Un algorithme de bagging, mais avec selection de variables.
- Très facile à paralèliser

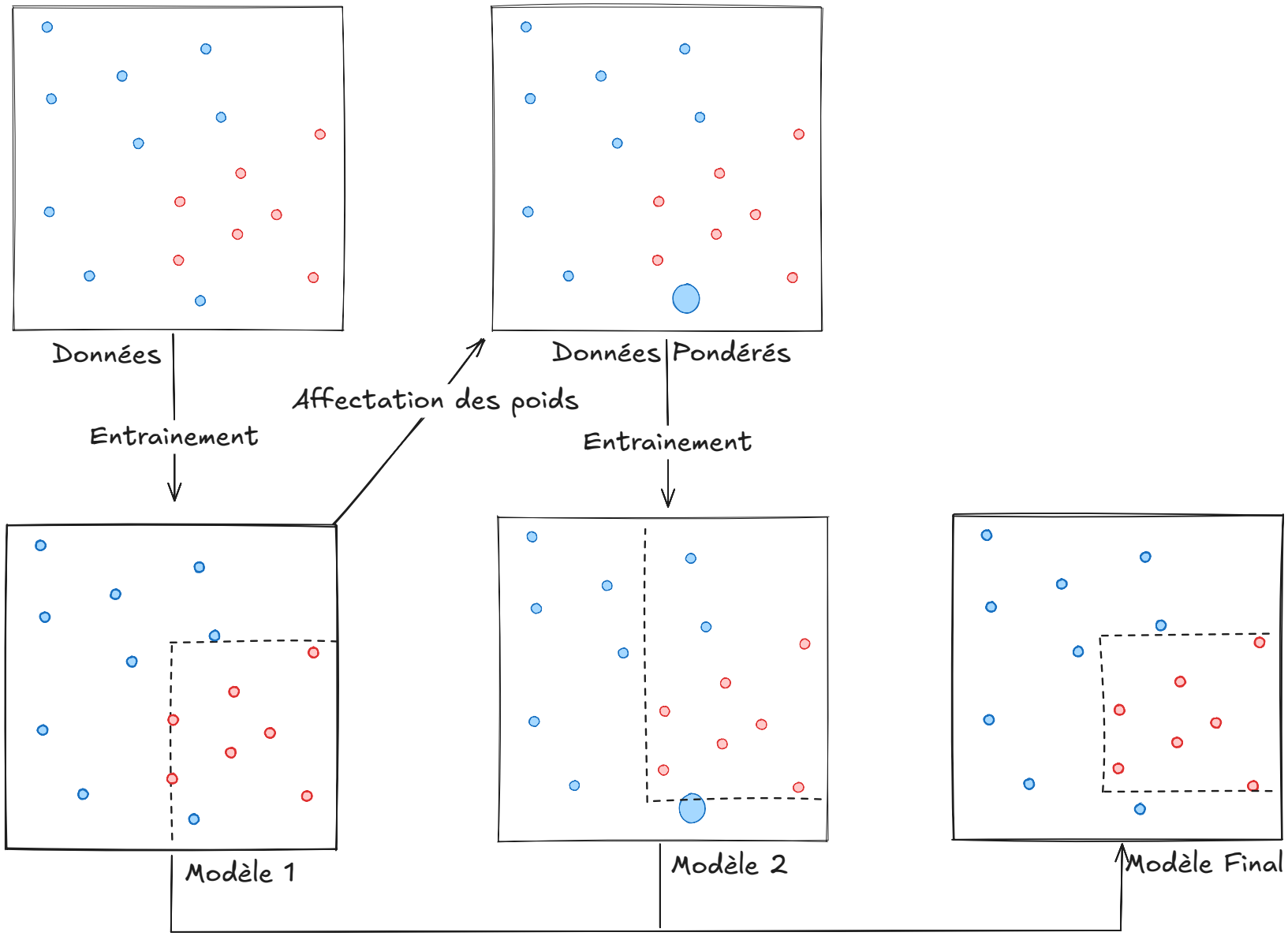
```
from sklearn.ensemble
import RandomForestClassifier

# Définition du modèle
rf = RandomForestClassifier(
    n_estimators=10,
    criterion='gini'
)

# Entrainement du modèle
rf.fit(X, y)
```

Boosting

A l'instar du Bagging, concept du Boosting est d'améliorer la performance de "weak Learners".

Cependant, cette fois on procède de façon itérative, chaque modèle viendra renforcer les faiblesses du modèles précédent.

Il existe plusieurs algorithmes pour les problèmes de régression et de classification. Les plus utilisés sur python sont : adaboost, gradient boosting, XGBoost


Données
Entrainement
Modèle 1
Affectation des poids
Données Pondérés
Entrainement
Modèle 2
Modèle Final

L'algorithme d'Adaptive Boosting

- On initialise de façon **uniforme** les poids de chaque individu i : w_i^0
- Pour chaque itération s :
  - On choisis le modèle qui minimise l'erreur en fonction de w_i^s.
  - On utilise le modèle h^s pour calculer le taux d'erreur E^s
  - On calcule : $\alpha^s = \frac{1}{2}\ln\left(\frac{1-E^s}{E^s}\right)$
  - On met à jour les poids: $w_i^{s+1} = w_i^s . \exp^{-\alpha^s y_i h^s(x_i)}$
- Condition d'arret : $E^s < seuil$ ou $s >$ max_iter
- Le modèle final $H(x) = signe(\sum_{i=1}^{n} \alpha^i h^i(x))$

Adaboost sur Sklearn

base_estimator : *(Estimator)*:
defaut : **DecisionTreeClassifier**

learning_rate : *(float)*:
Augmente la valeur de α^s
Trade off avec n_estimators

n_estimators : *(int)*:
Nombre d'iterations maximale

```python
from sklearn.ensemble
import AdaBoostClassifier

#Definition du modèle
ada = AdaBoostClassifier(
    base_estimator=None,
    learning_rate=1.0,
    n_estimators=10
)
# Entrainement du modèle
ada.fit(X, y)
```

Gradient Boosting

L'idée du boosting est de prédire itérativement les résidus du modèle précedent.

On va utiliser une fonction de perte type (expodentielle, log-loss).

Et chercher à prédire de façon additive les résidus laissés par les modèles précédents.

Sur python : GradientBoostingClassifier, GradientBoostingRegressor, XGBoost

TP : Aggrégations