Arbres de Décision

Octobre 2025

Cédric Dangeard
cedric.dangeard@orange.com

Cours 2 : Au programme

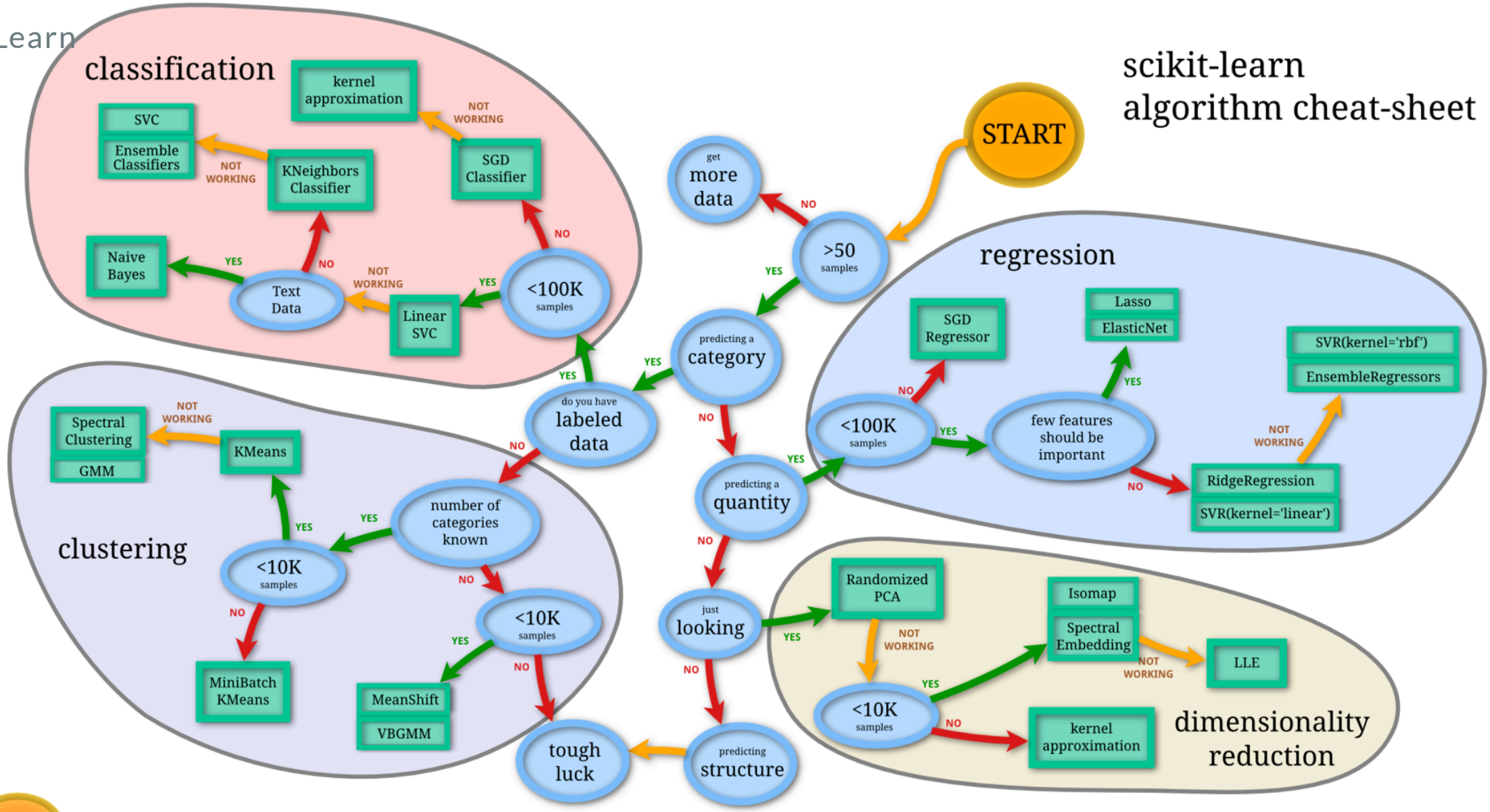
- Les bibilothèques fondamentales
- Nettoyage et Préparation des données
  - Gestion des valeurs manquantes
  - Encodage des variables catégorielles
  - Echantillonnage des données
- TP : Nettoyage des données
- Les arbres de décision
- TP : Implémentation d'un Arbre

Les bibliothèques 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
MiniBatch KMeans
NO
<10K samples
YES
MeanShift
VBGMM
NO
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

source

- 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)
```

Encodage des variables catégorielles

Pourquoi encoder les variables catégorielles ?

- Les algorithmes de machine learning ne peuvent pas travailler directement avec des données non numériques.
- Les variables catégorielles doivent être converties en une représentation numérique pour être utilisées comme entrées dans les modèles.

Différentes méthodes d'encodage existent, chacune ayant ses avantages et inconvénients.

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']])
```

Echantillonnage des données

- Souvent les données sont déséquilibrées
- Plusieurs techniques pour rééquilibrer les classes
  - Sur-échantillonnage (Over-sampling)
  - Sous-échantillonnage (Under-sampling)
  - Synthèse de nouvelles données (SMOTE, ADASYN, ...)
- Imbalanced-learn : Librairie Python pour le rééchantillonnage

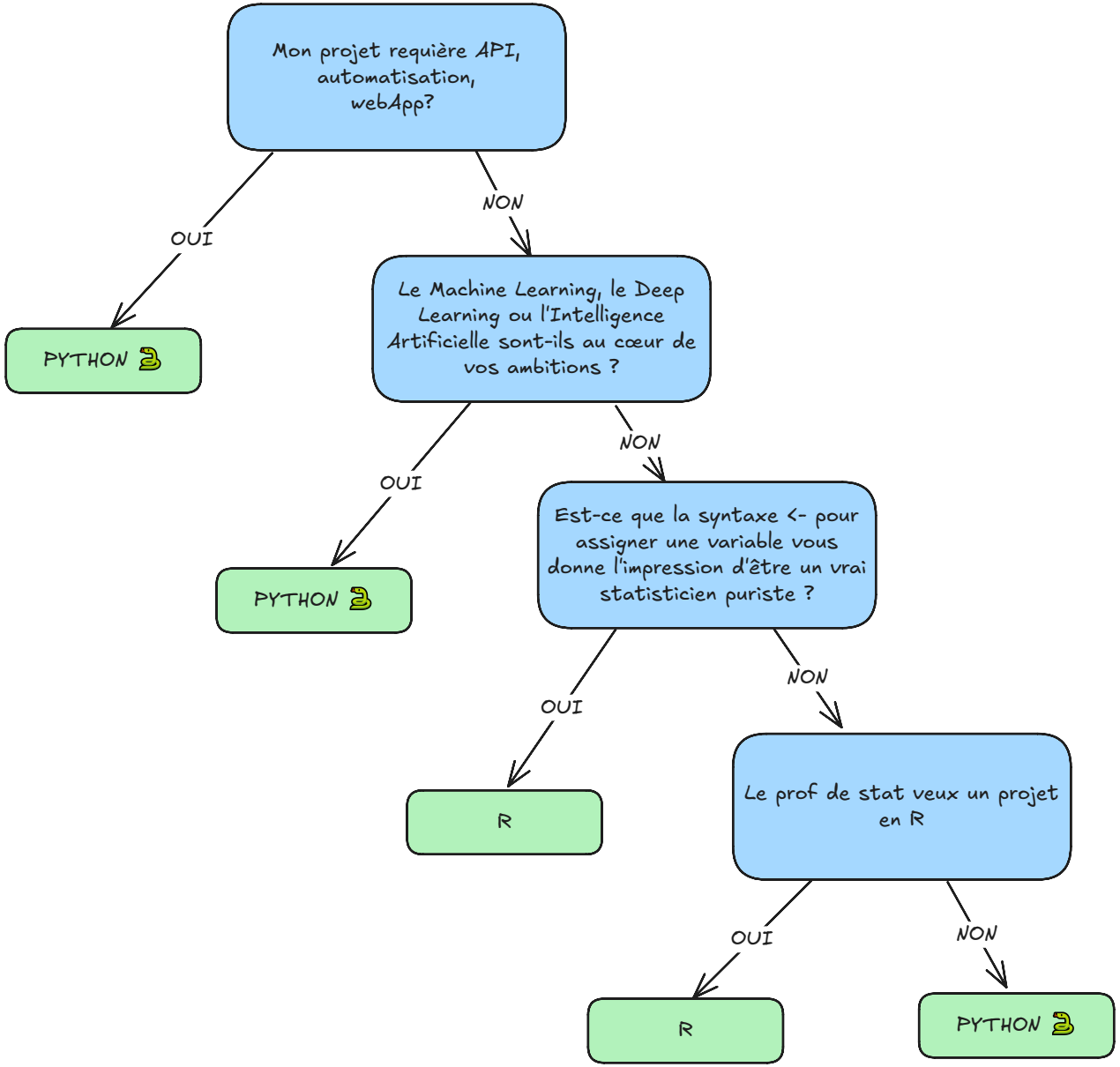
TP : Encodage

Arbres de décisions

Qu'est ce qu'un arbre de décision ?

Vocabulaire sur les arbres

- **Nœud racine** : Le nœud supérieur de l'arbre, qui ne possède pas de parent.
- **Nœud de décision** : Un nœud à partir qui divise les données en fonction d'une caractéristique.
- **Feuille** (ou **nœud terminal**) : Un nœud qui ne se divise pas davantage et représente une prédiction ou une classe.
- **Branche** : Une connexion entre deux nœuds, représentant le flux de décision.


Mon projet requière API, automatisation, webApp?
OUI
NON
PYTHON
Le Machine Learning, le Deep Learning ou l'Intelligence Artificielle sont-ils au cœur de vos ambitions ?
OUI
NON
PYTHON
Est-ce que la syntaxe <- pour assigner une variable vous donne l'impression d'être un vrai statisticien puriste ?
OUI
NON
R
Le prof de stat veux un projet en R
OUI
NON
R
PYTHON

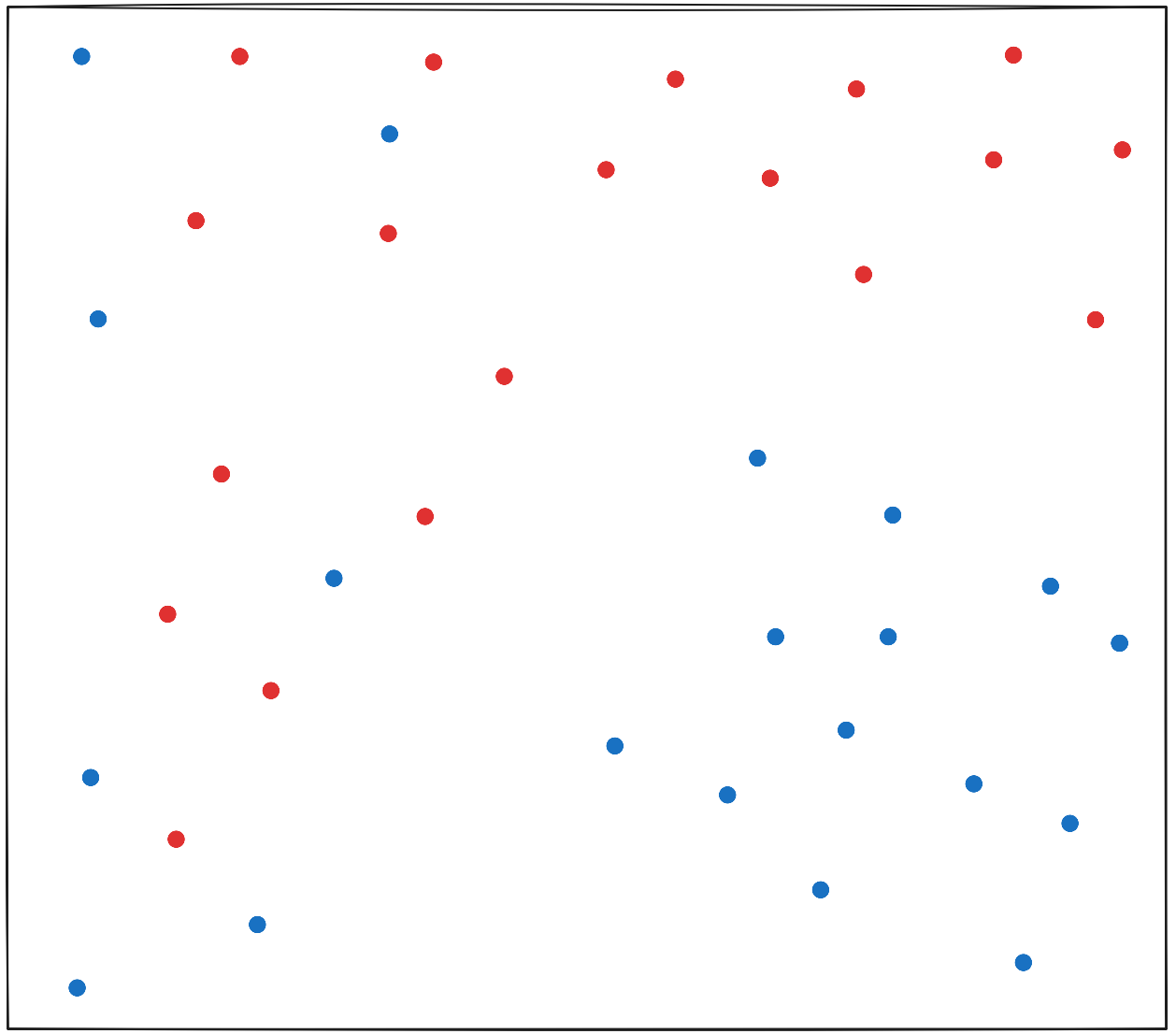
Arbre de décision

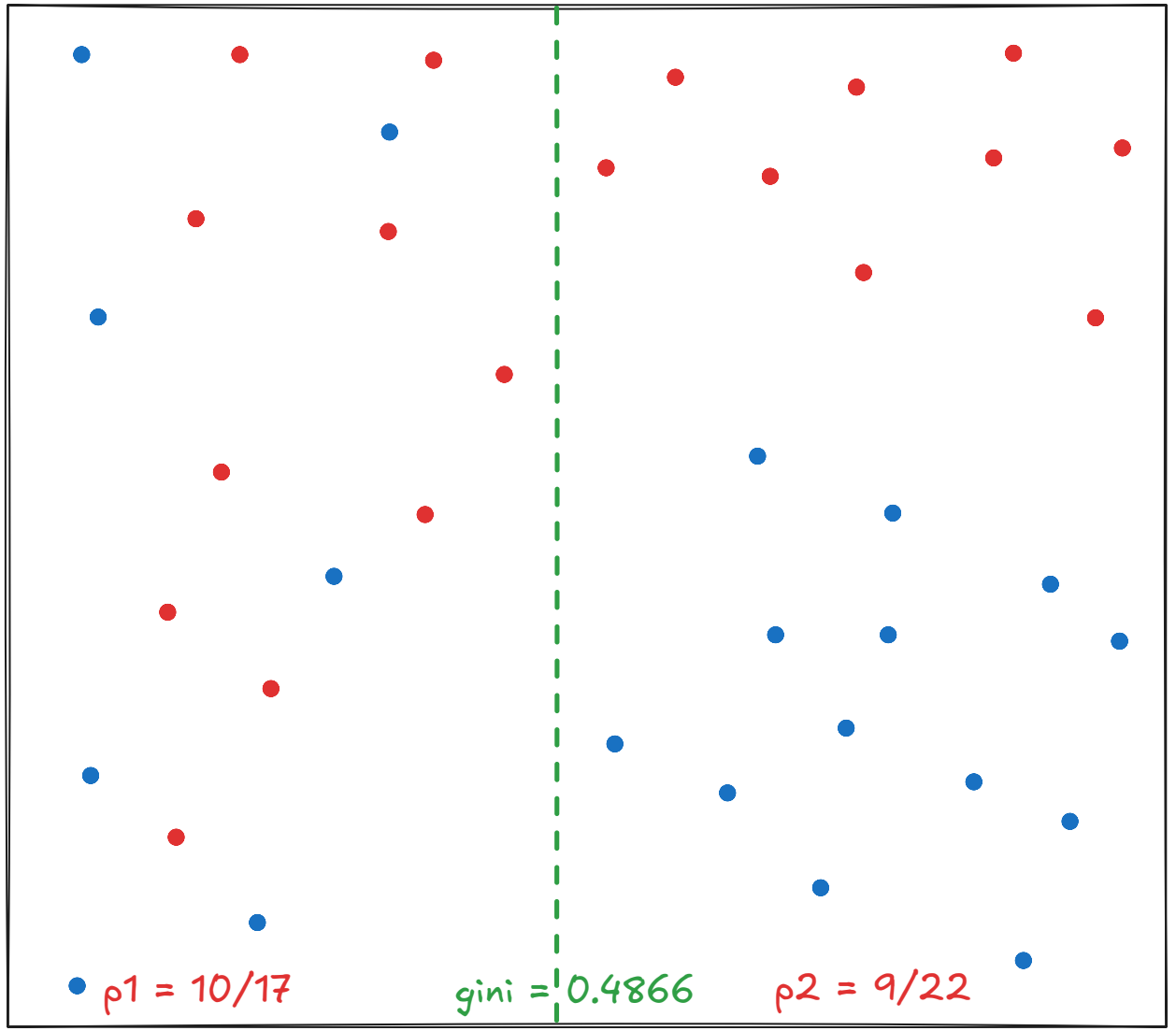
- Modèle de classification et de régression
- Structure arborescente composée de nœuds
- Chaque nœud représente une décision basée sur une caractéristique
- Chaque branche représente le résultat d'une décision
- Chaque feuille représente une prédiction ou une classe

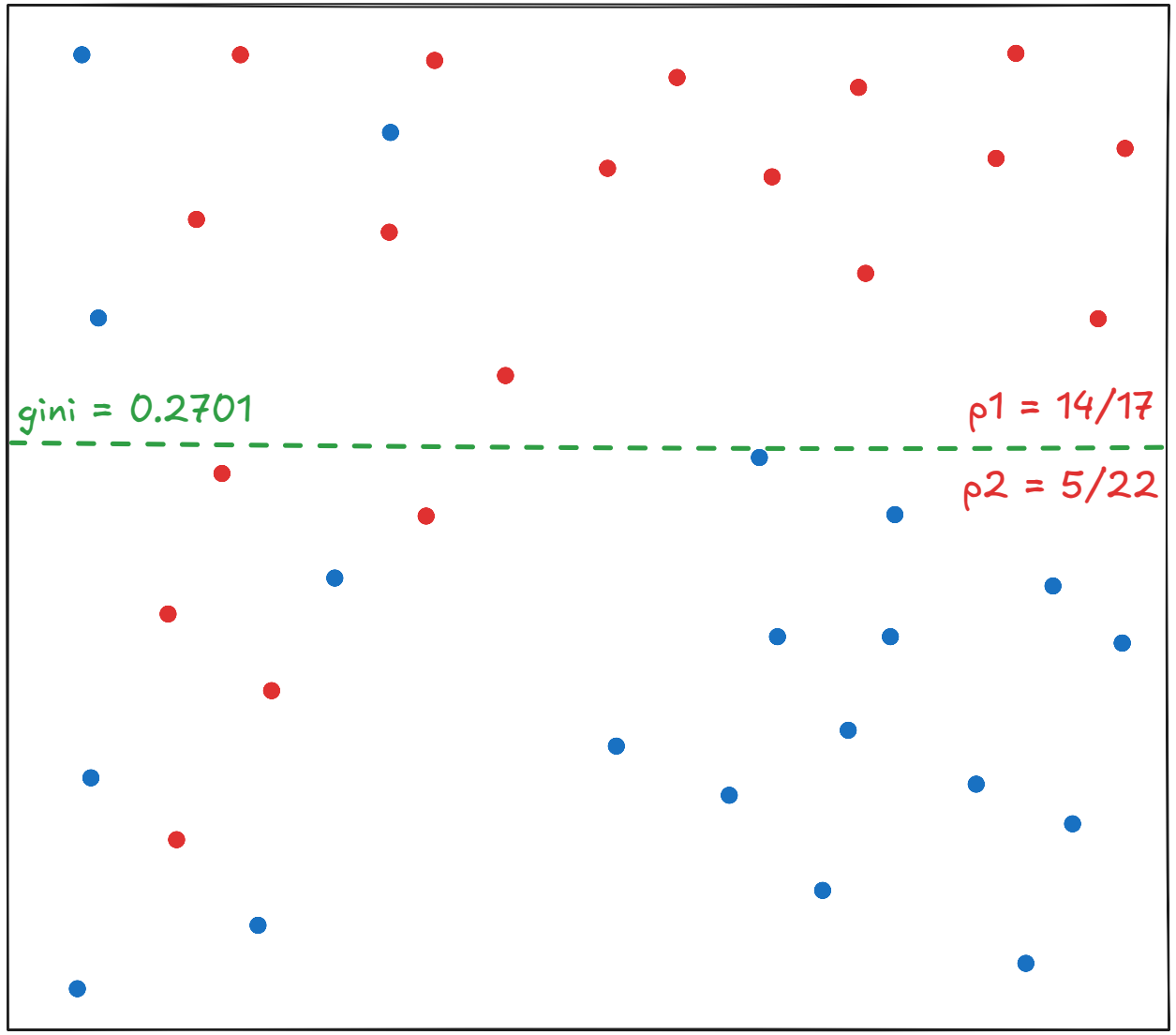
Construction d'un arbre de décision

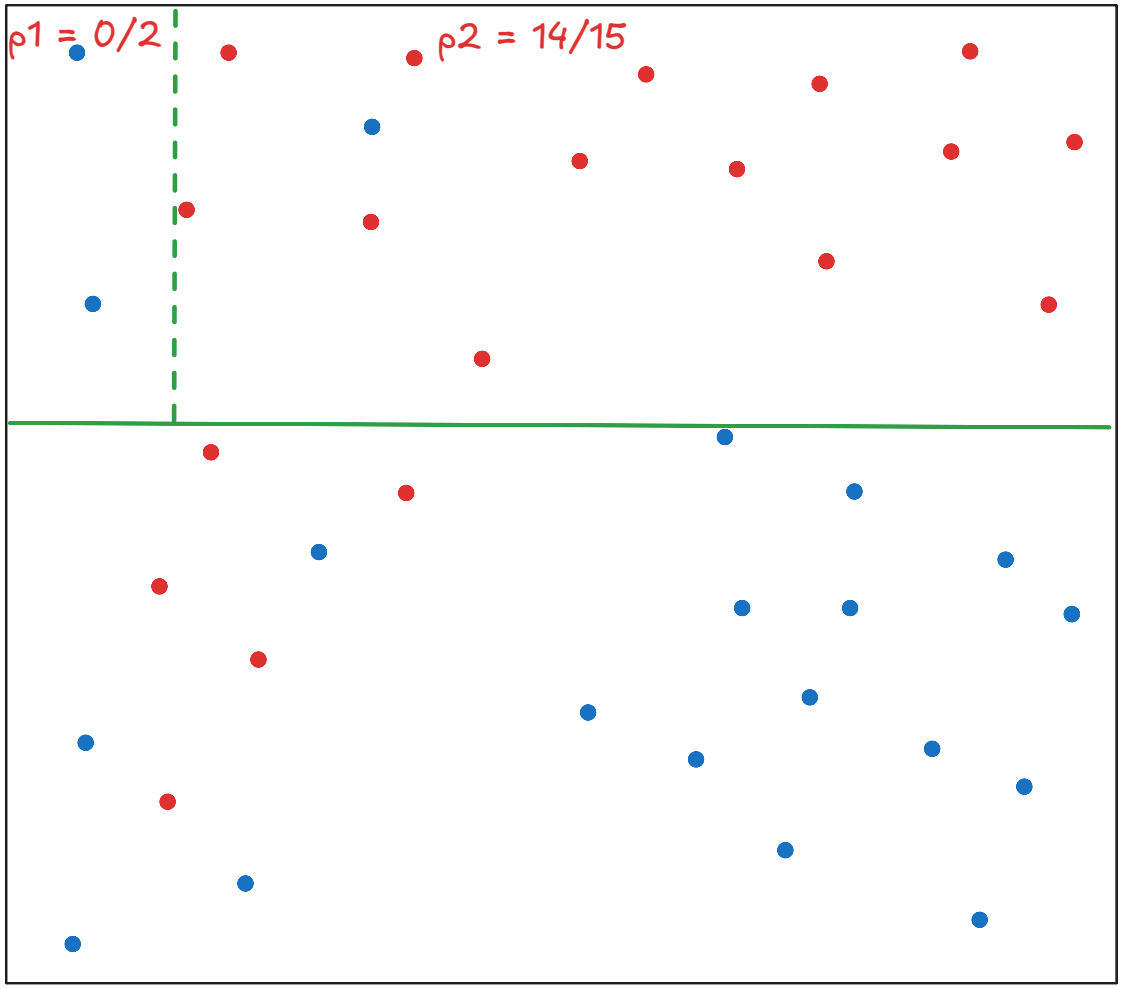
La construction se fait par *partitionnement récursif* des données.

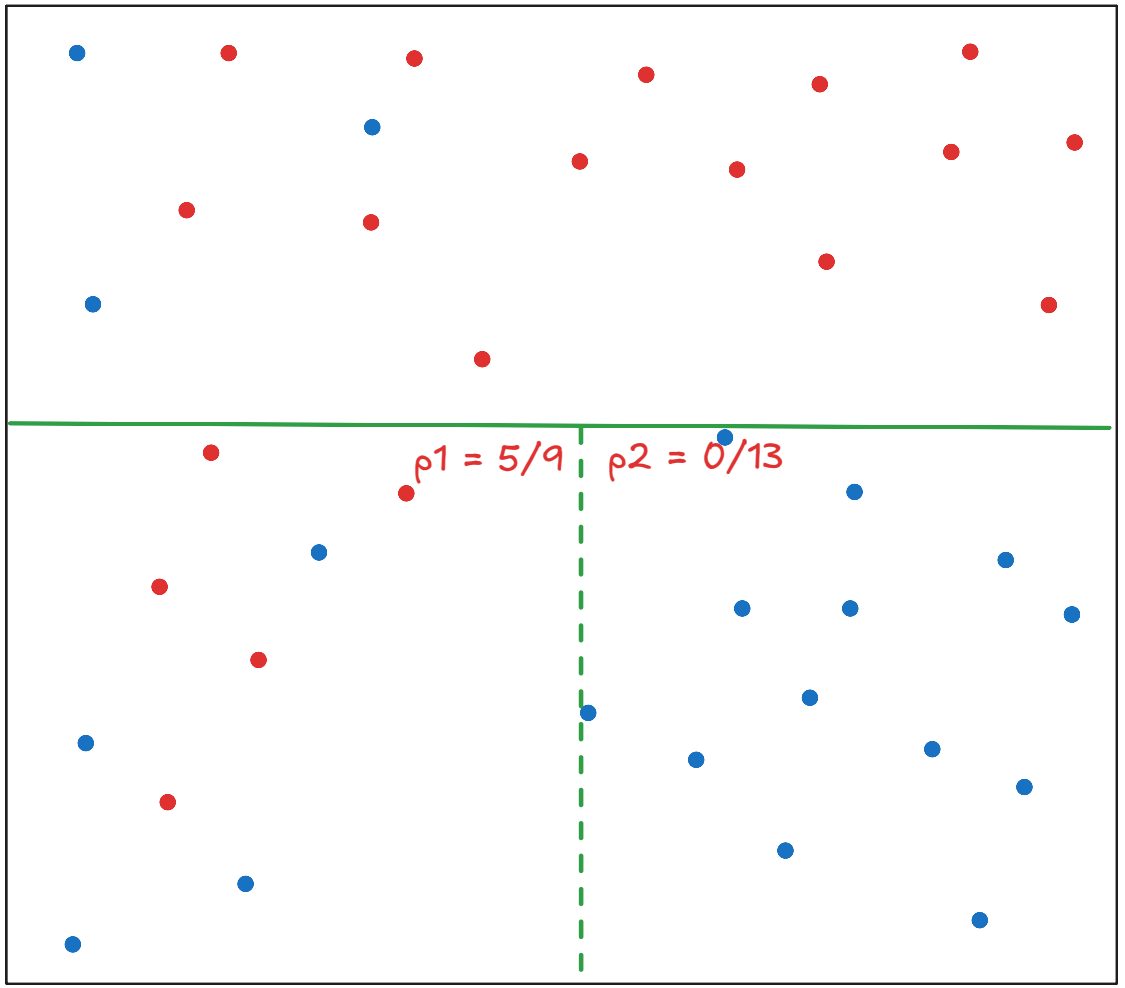
1. Choisir la meilleure division pour diviser les données.
2. Diviser les données en sous-ensembles basés sur cette caractéristique.
3. Répéter le processus pour chaque sous-ensemble jusqu'à ce qu'un critère d'arrêt soit atteint

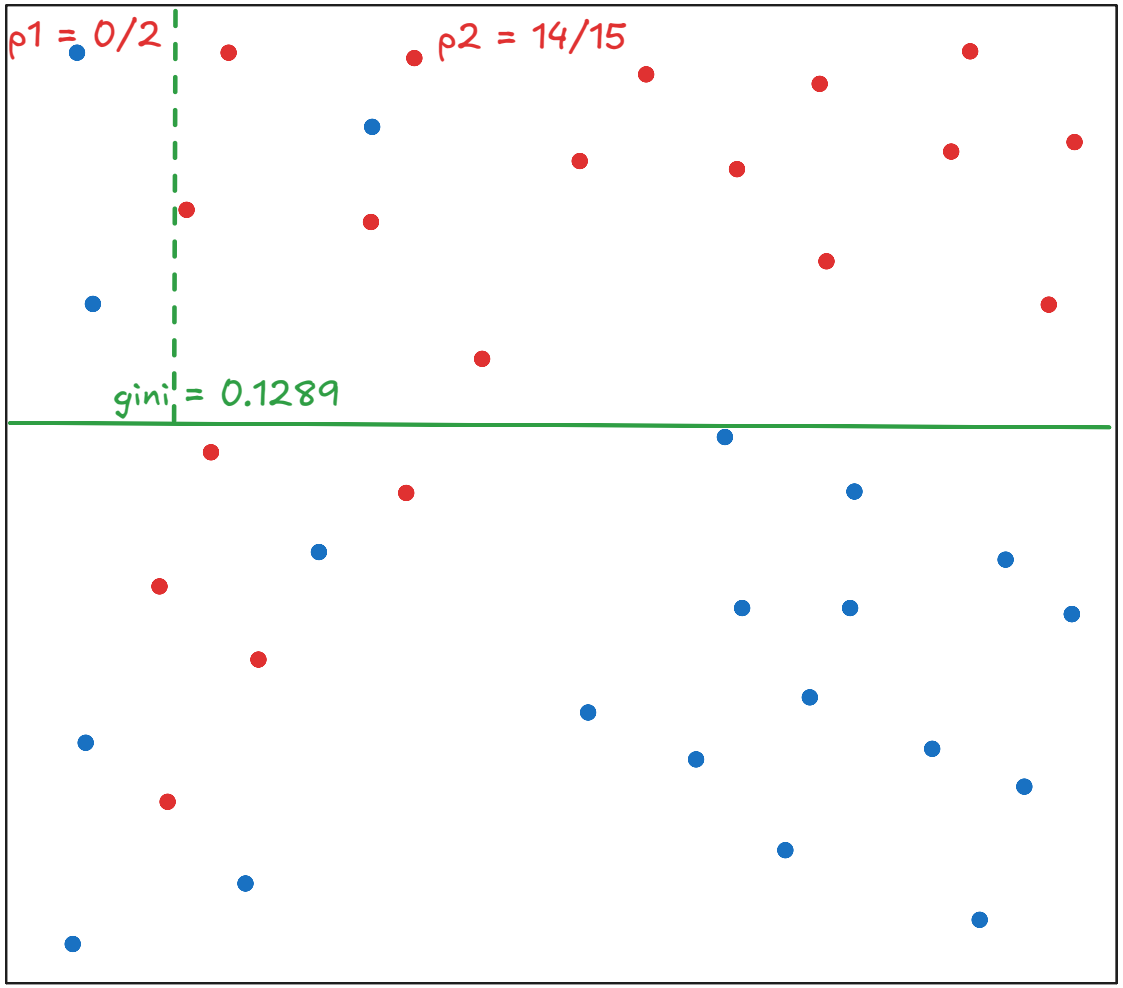


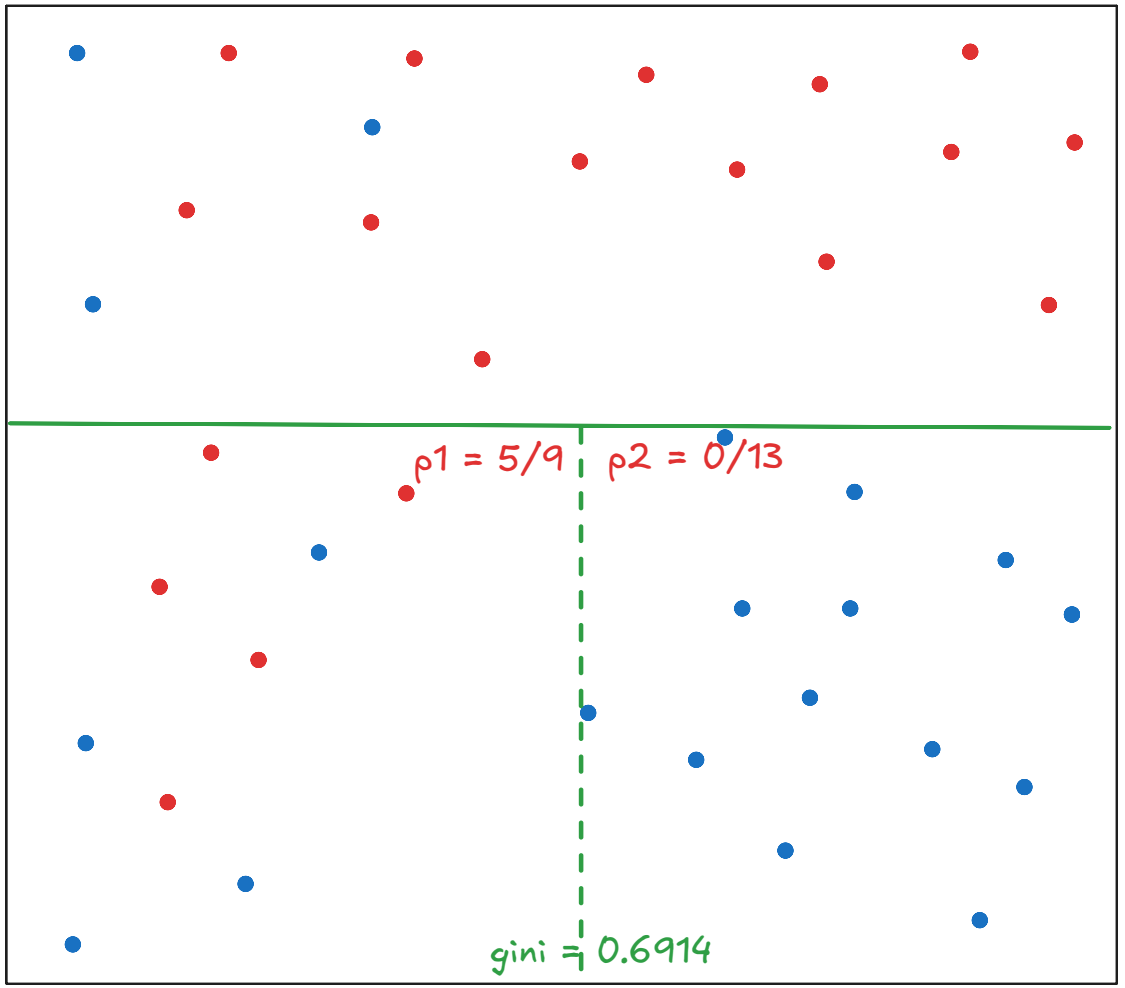

p1 = 10/17
gini = 0.4866
p2 = 9/22

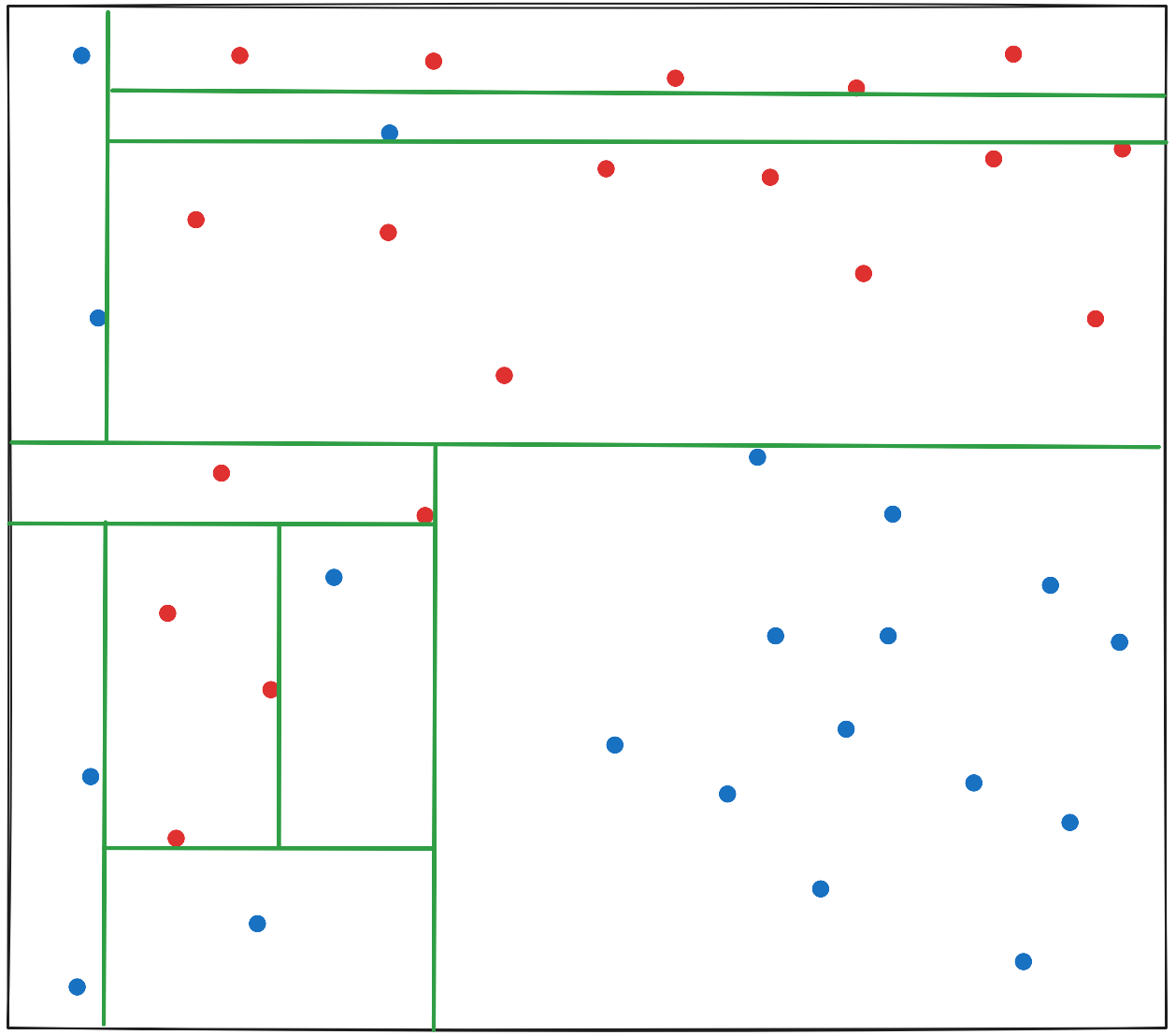

gini = 0.2701
p1 = 14/17
p2 = 5/22


p1 = 0/2
p2 = 14/15

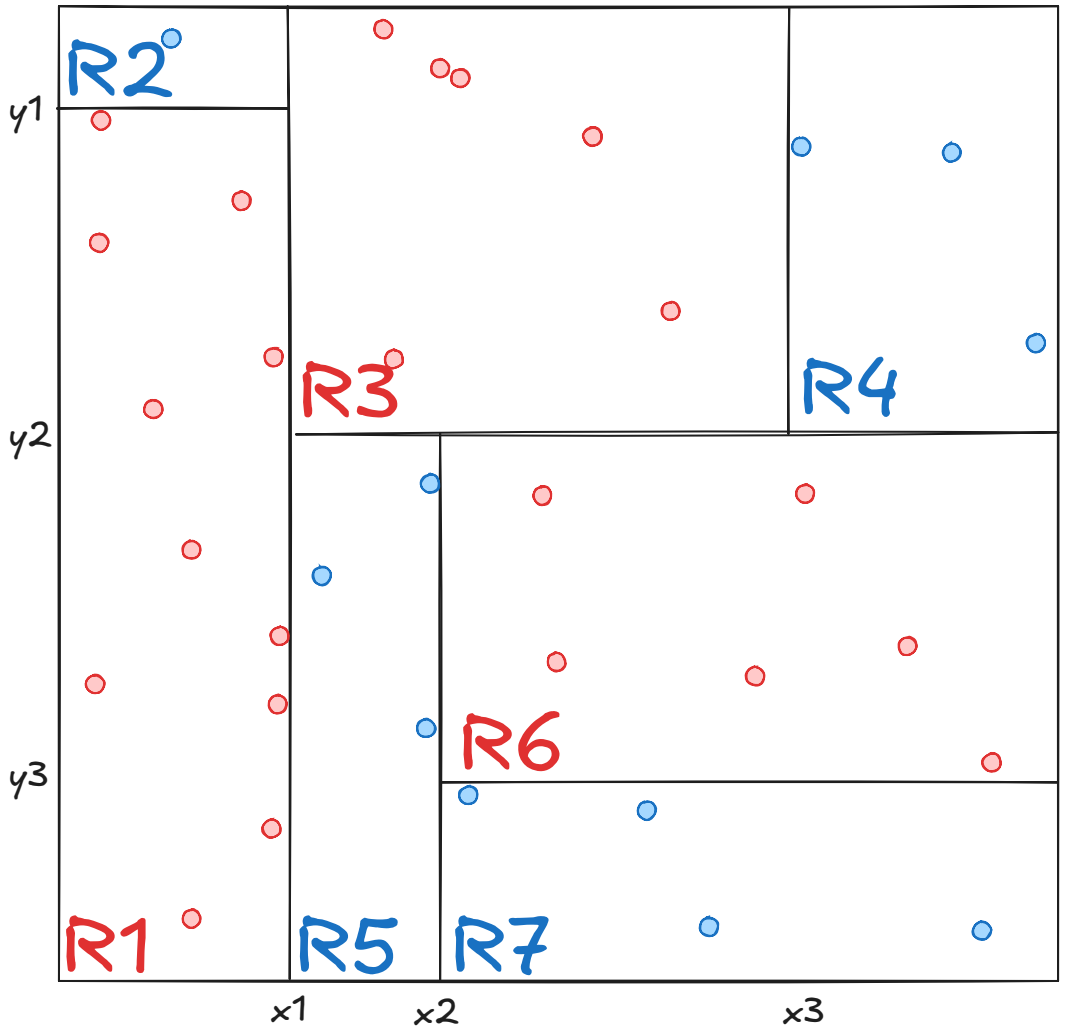

p1 = 5/9
p2 = 0/13

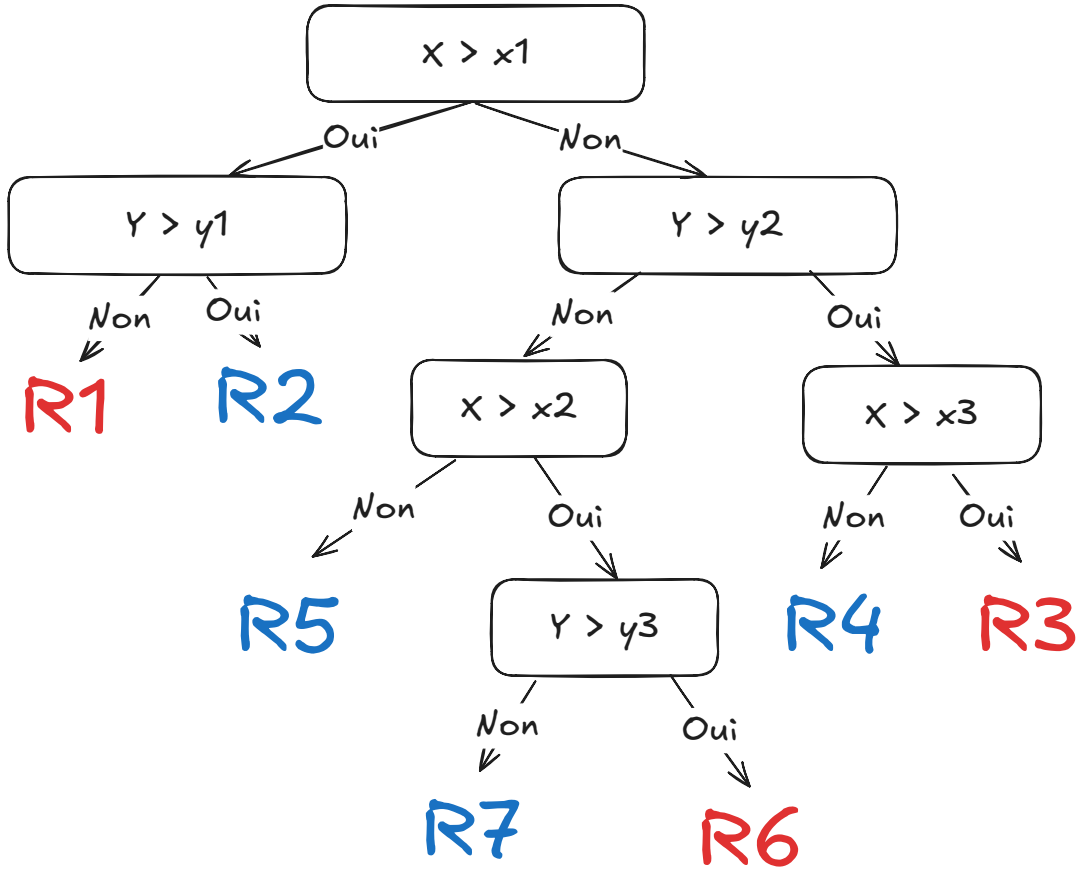

p1 = 0/2
p2 = 14/15
gini = 0.1289


p1 = 5/9
p2 = 0/13
gini = 0.6914



Arbres de décisions





Comment choisir la meilleure division ?

- Quelle variable choisir ?
- Quel seuil de division ?

CART : L'algorithme pour un nœud

- Définir un critère d'homogénéité
  - CART : Indice de gini : $I_G = 1 - \sum_{i=1}^{m} f_i^2$
    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 ? Que cherche t-on à éviter?

- 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.

Arbres de décisions : Conclusion

- 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 ?

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

TP : Arbres de décision