Détection de fraude bancaire

VOL DE CARTES BLEUS

Dans cette présentation…

I. Présentation des outils

Environnement de développement

Google Colab (GPU offert)

VS Code (notebooks)

Librairies utilisées

Pandas (Dataframes)

Scitkeat learn (machine learning)

Partage des codes

Dépôt github

[yannasyr/TPE2023-FraudDetection: Telecom Physique Strasbourg | 2A ISSD | TPE with Z.Inas on Credit Card Fraud Detection (github.com)]

II. Notre dataset : point de départ

ChAllenge Data By MathA

BNP PARIBAS PERSONAL FINANCE

[Challenge data (ens.fr)]

x_train
variables explicatives pour l'entraînement
y_train
variable(s) cible(s) pour l'entraînement
x_test
variables explicatives pour le test

91471 / 1319

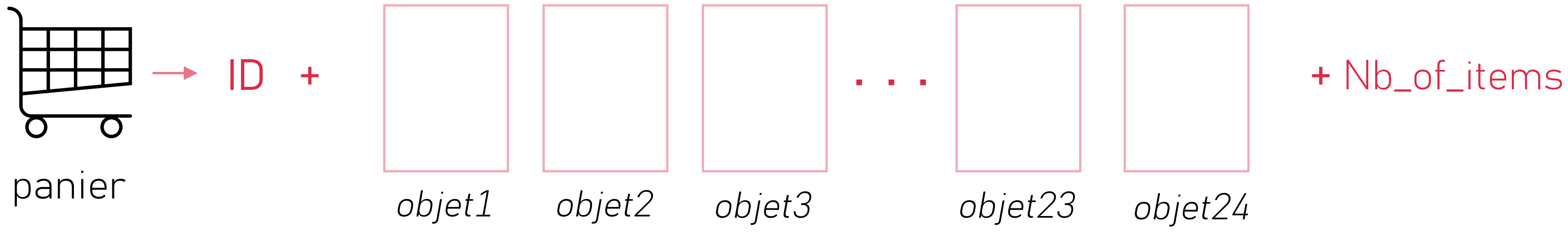
22836 / 362

Métrique imposée : `average_precision_score`

- **Benchmark 1 :** $\text{PR-AUC}_1 = 0,017$
  Le premier benchmark est naïf et considère un modèle qui prédit aléatoirement une probabilité entre 0 et 1.
- **Benchmark 2 :** $\text{PR-AUC}_2 = 0,14$
  Le second benchmark intègre plusieurs étapes de pré-processing et utilise un modèle de Machine Learning optimisé pour prédire le risque de fraude.

II. Notre dataset :

Ligne i :



II. Notre dataset :

X

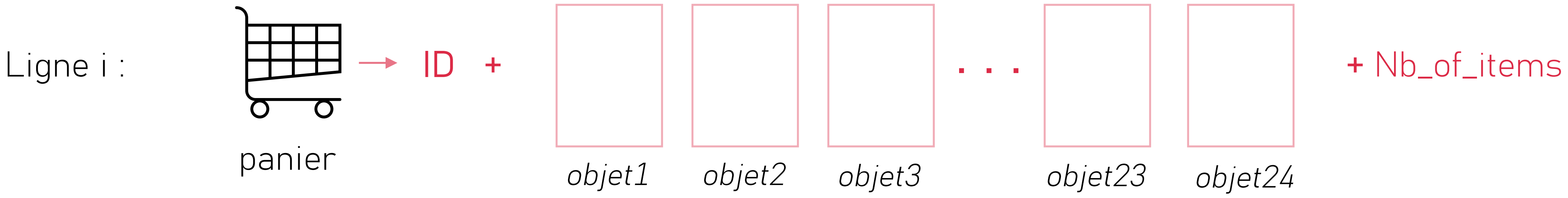
Ligne i :  → ID + objet1 objet2 objet3 . . . objet23 objet24 + Nb_of_items

panier

	ID	item1	cash_price1	make1	model1	goods_code1	Nbr_of_prod_purchas1
0	85517	COMPUTERS	889.0	APPLE	2020 APPLE MACBOOK AIR 13 3 RETINA DISPLAY M1 ...	239246776	1.0

II. Notre dataset :

X



	ID	item1	cash_price1	make1	model1	goods_code1	Nbr_of_prod_purchas1
0	85517	COMPUTERS	889.0	APPLE	2020 APPLE MACBOOK AIR 13 3 RETINA DISPLAY M1 ...	239246776	1.0

Y

	index	ID	fraud_flag
0	0	85517	0
1	1	51113	0

III. Pré-traitement des données.

Nominal Data → One-hot Encoding

`pd.get_dummies()`

	ID	item1	cash_price1	make1	model1	goods_code1	Nbr_of_prod_purchas1
0	85517	COMPUTERS	889.0	APPLE	2020 APPLE MACBOOK AIR 13 3 RETINA DISPLAY M1 ...	239246776	1.0

III. Pré-traitement des données.

sex
male
female
female
male
male
male
male
female
male

pd.get_dummies()

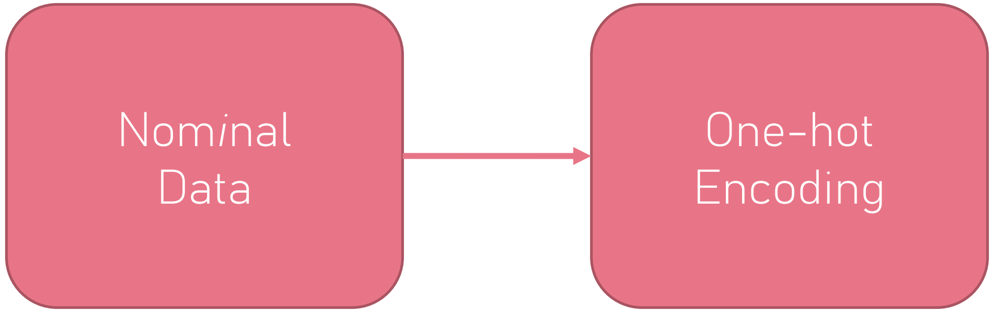
sex_male	sex_female
1	0
0	1
0	1
1	0
1	0
1	0
1	0
0	1
1	0

Nominal Data → One-hot Encoding

`pd.get_dummies()`

	ID	item1	cash_price1	make1	model1	goods_code1	Nbr_of_prod_purchas1
0	85517	COMPUTERS	889.0	APPLE	2020 APPLE MACBOOK AIR 13 3 RETINA DISPLAY M1 ...	239246776	1.0

III. Pré-traitement des données.



`pd.get_dummies()`

	ID	item1	cash_price1	make1	model1	goods_code1	Nbr_of_prod_purchas1
0	85517	COMPUTERS	889.0	APPLE	~~2020 APPLE MACBOOK AIR 13 3 RETINA DISPLAY M1 ...~~	~~239246776~~	1.0

III. Pré-traitement des données.

674 marques dans le dataset

Seuillage : + de 20achats, +0,1% fraudes

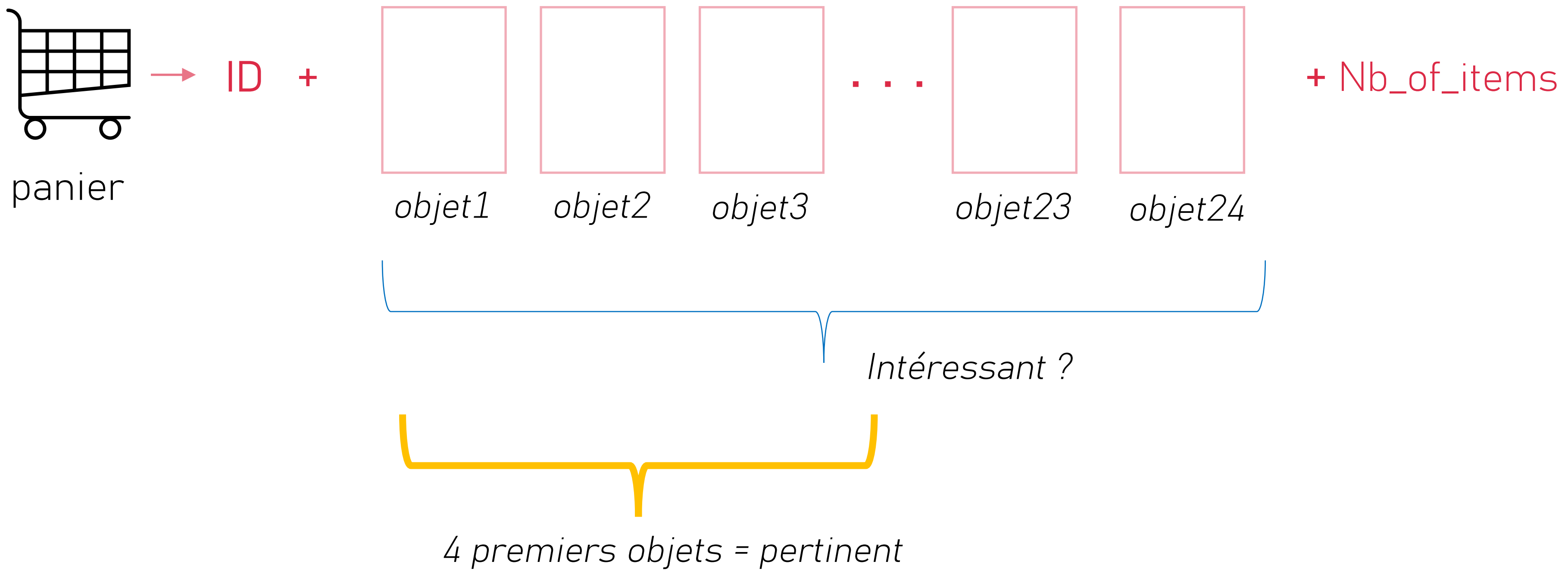
26 marques intéressantes

161 catégories dans le dataset

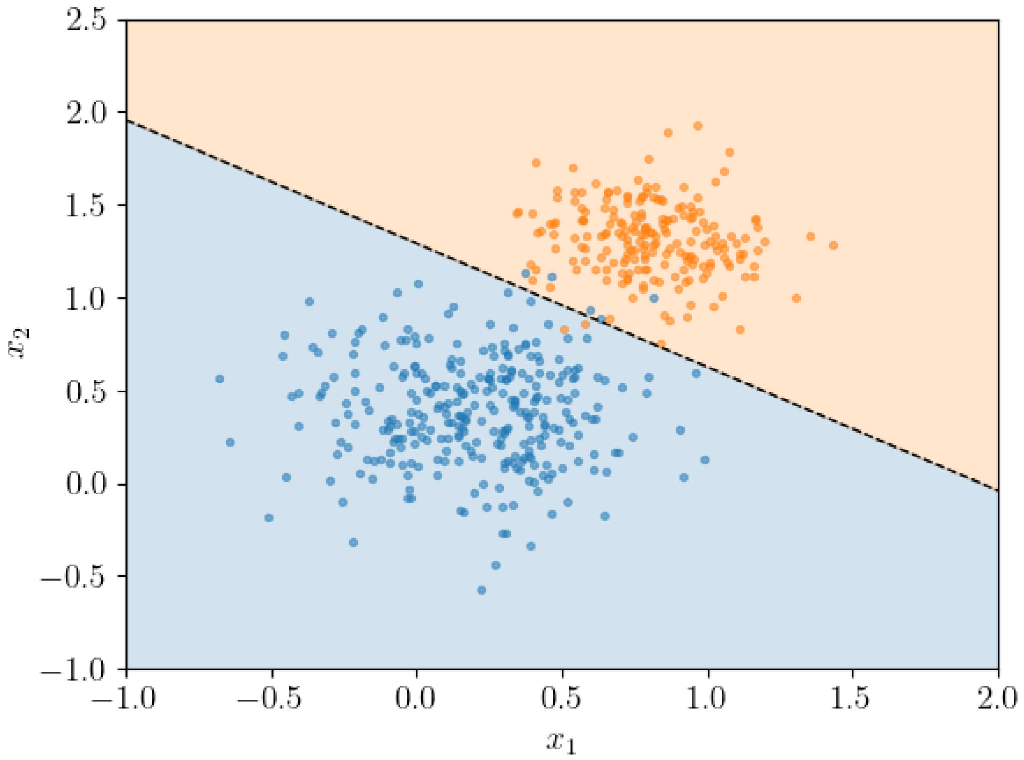
Seuillage : + de 10achats, +1% fraudes

20 catégories intéressantes

III. Pré-traitement des données.



IV. Modèles : Logistic Regression



The Logistic Function

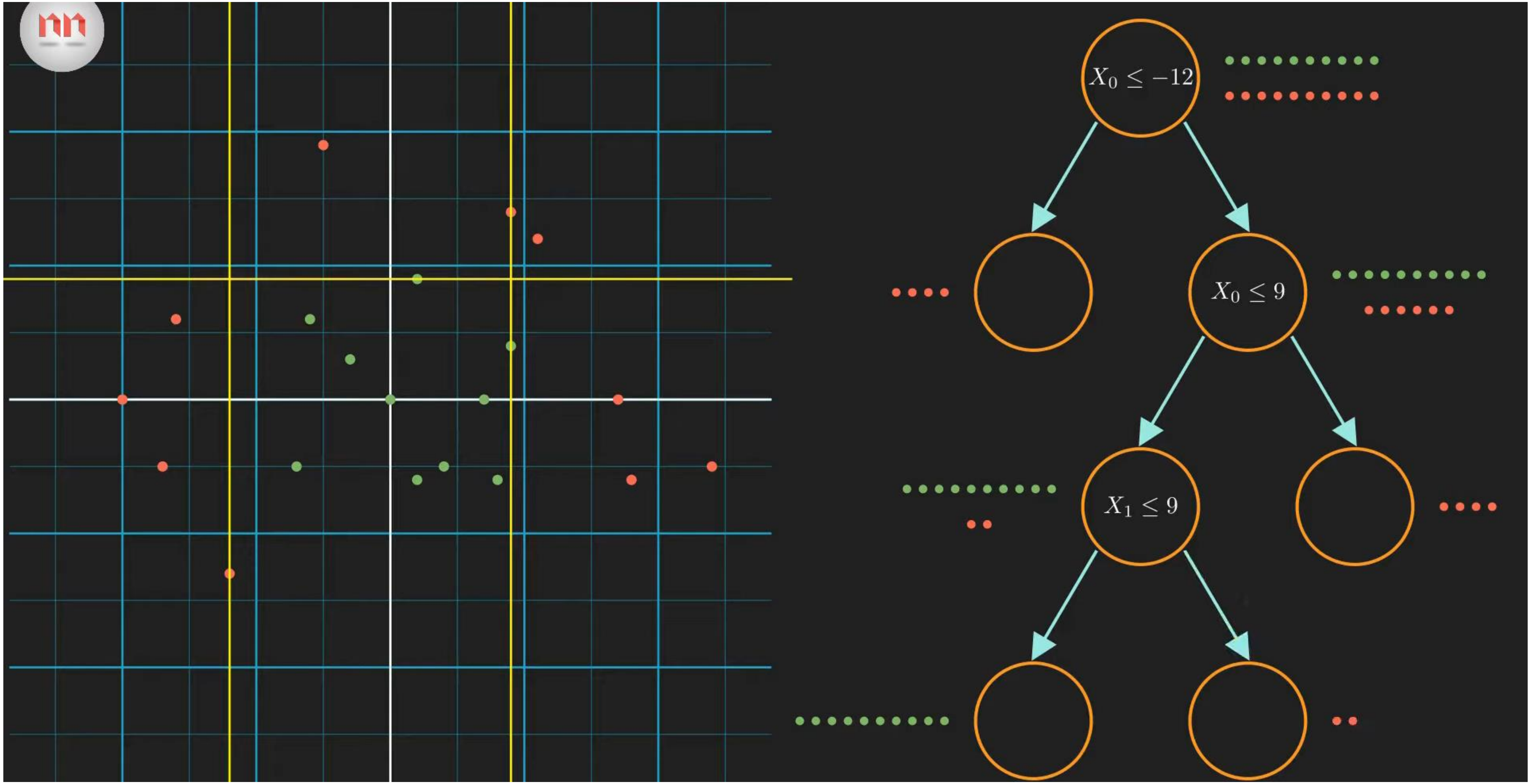
$$y = \frac{1}{1 + e^{-f(x_1, x_2, \ldots, x_n)}} \in (0, 1)$$

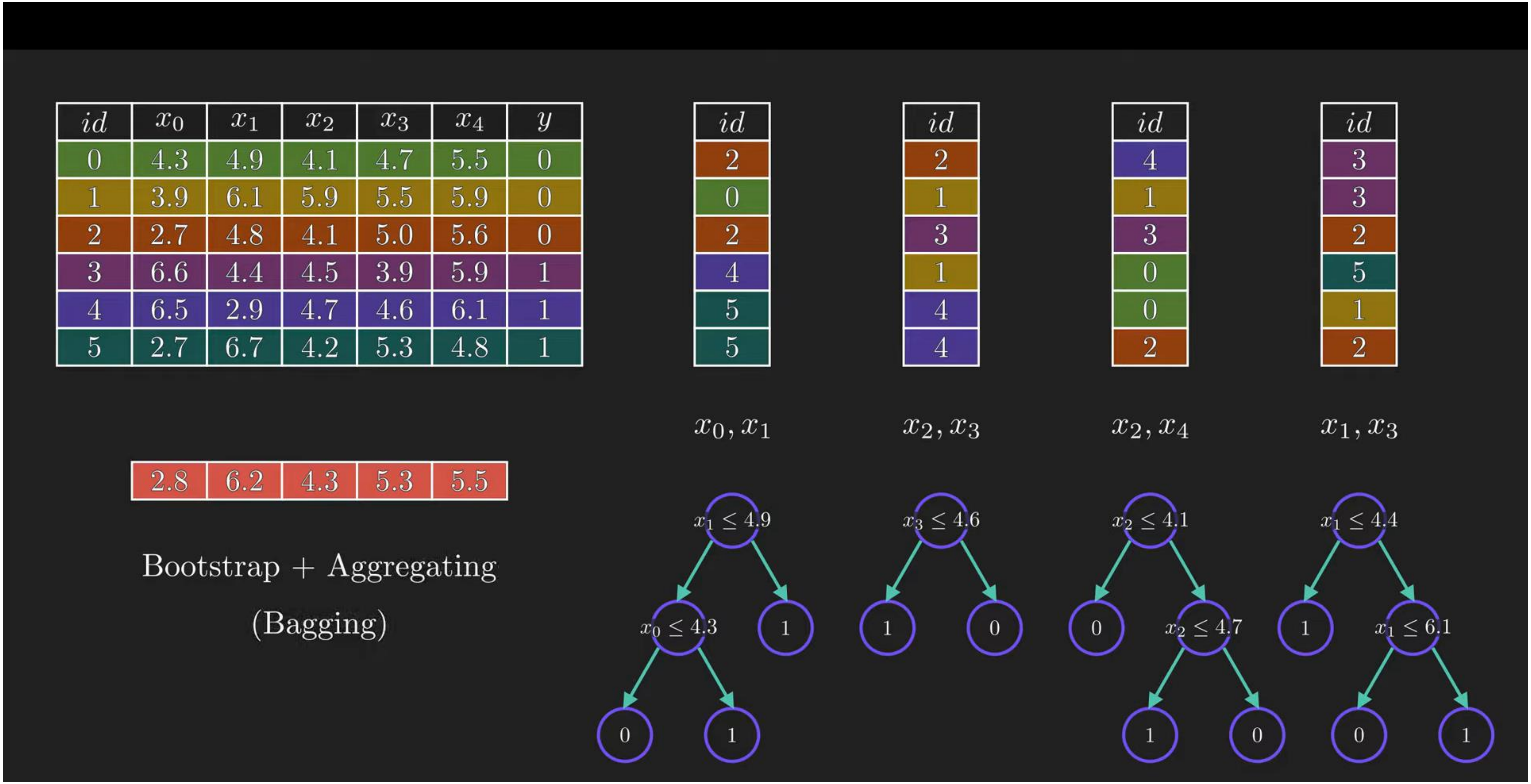
where

$$f(x_1, x_2, \ldots, x_n) = a_0 + a_1 x_1 + \ldots + a_n x_n \in (-\infty, +\infty)$$

- Output Y
- Input X = { x1 , x2 , …, xn}
- Poids/Paramètre : a0, a1, …, an

IV. Modèles : RandomForestClassifier


$X_0 \leq -12$
$X_0 \leq 9$
$X_1 \leq 9$

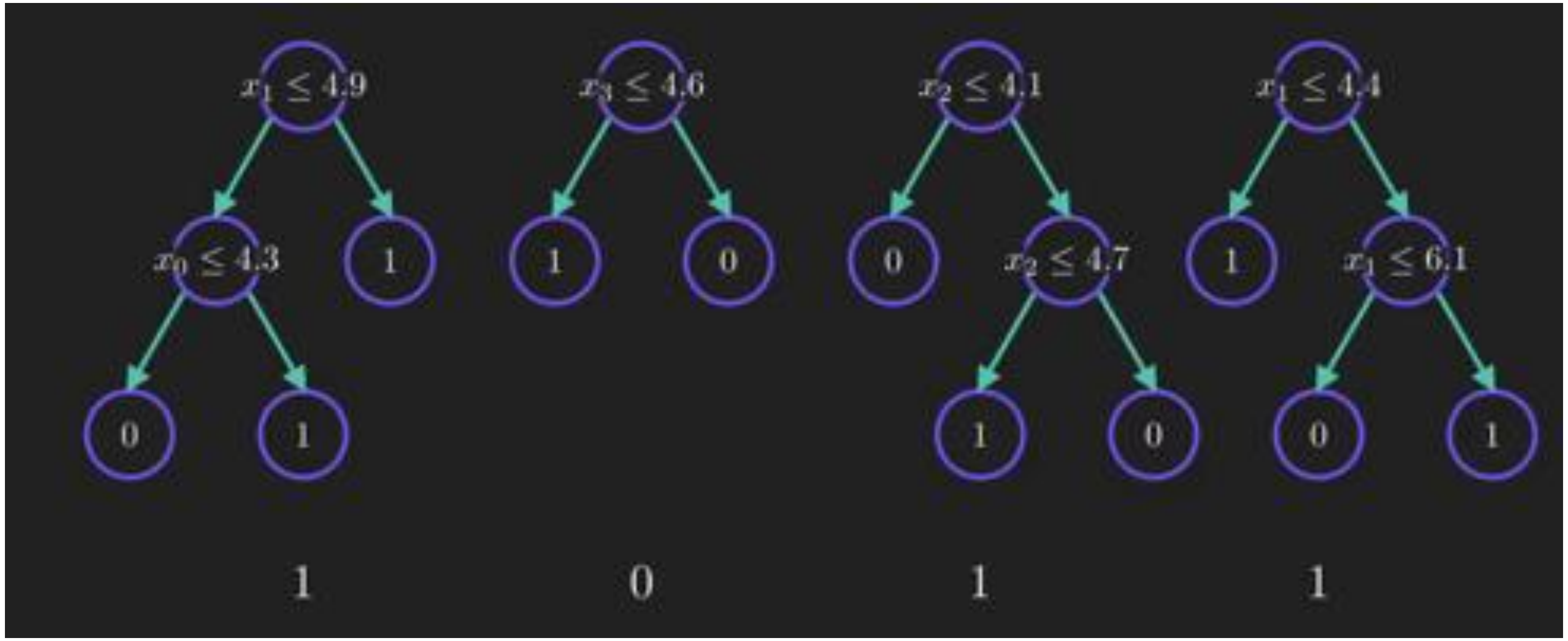


id	x_0	x_1	x_2	x_3	x_4	y
0	4.3	4.9	4.1	4.7	5.5	0
1	3.9	6.1	5.9	5.5	5.9	0
2	2.7	4.8	4.1	5.0	5.6	0
3	6.6	4.4	4.5	3.9	5.9	1
4	6.5	2.9	4.7	4.6	6.1	1
5	2.7	6.7	4.2	5.3	4.8	1

2.8	6.2	4.3	5.3	5.5

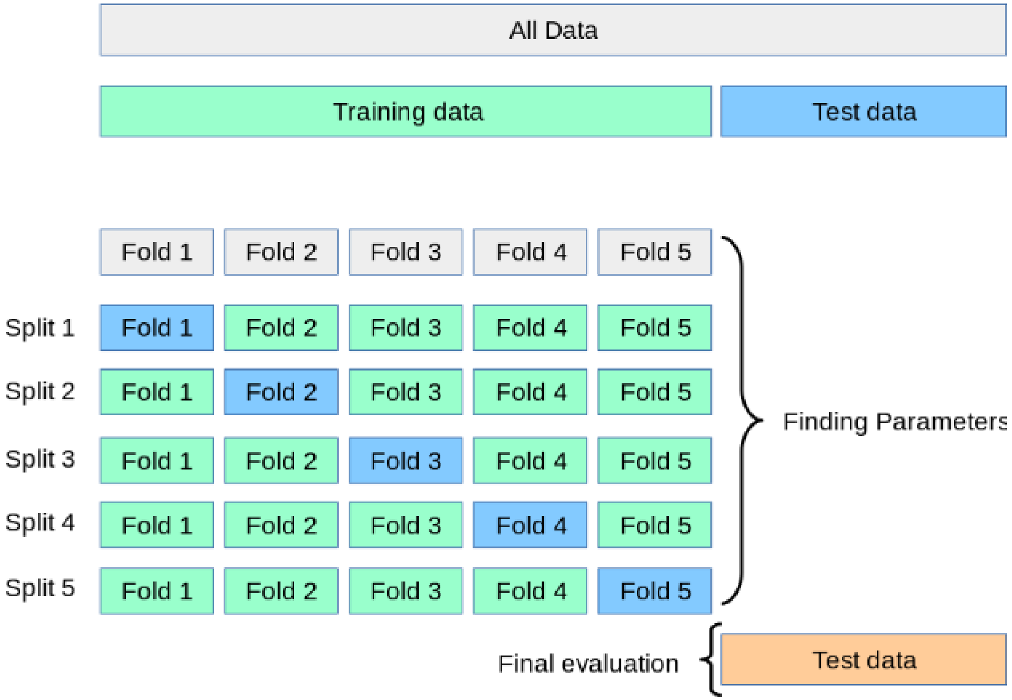
id	id	id	id
2	2	4	3
0	1	1	3
2	3	3	2
4	1	0	5
5	4	0	1
5	4	2	2

IV. Modèles : XGBoost




```
Boosted Ensemble =
First Tree + η * Second Tree

Loss(Boosted Ensemble) < Loss(First Tree)
```

IV. Modèles : cross-validation

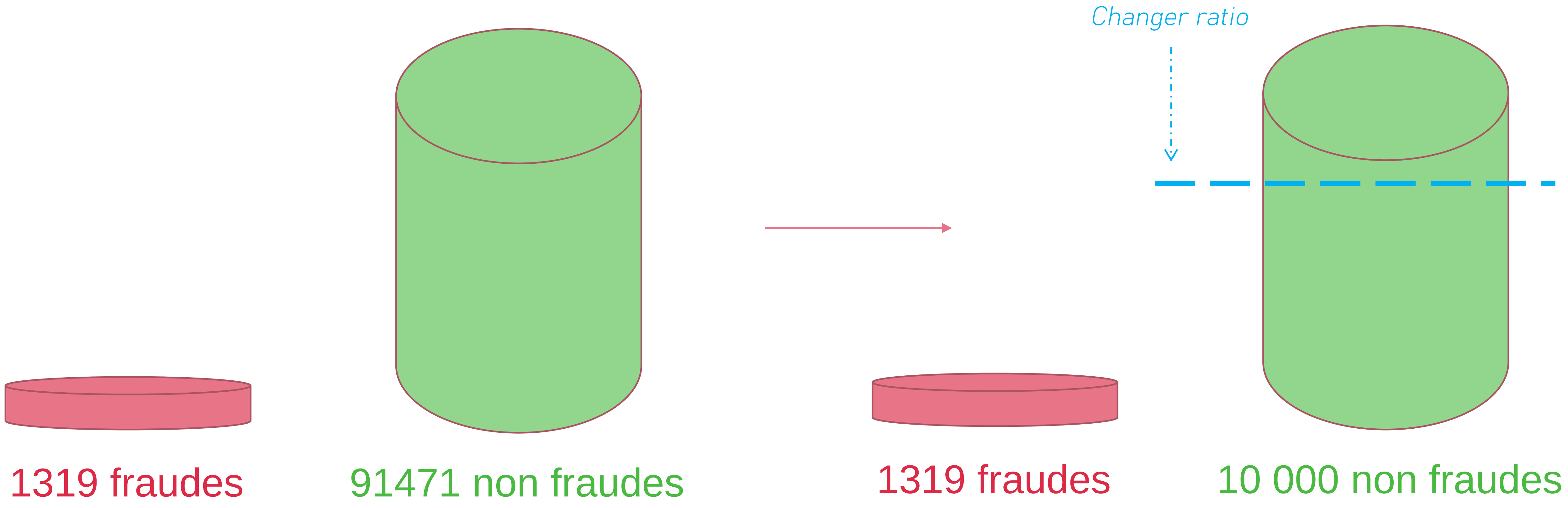



```
StratifiedKFold(n_splits=5, shuffle=True)
```

IV. Modèles : trouver les hyperparamètres

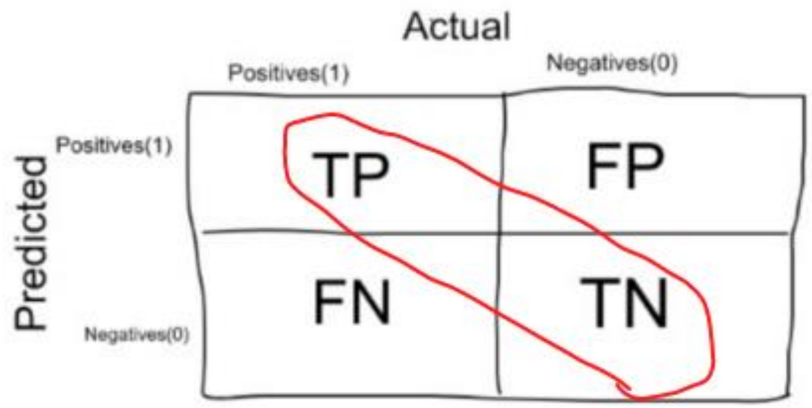
```python
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier

# Voici la grille de paramètres
param_grid = {
    'max_depth': [3, 4, 5, 6, 7, 8, 9, 10, None],
    'n_estimators': range(100, 1000, 50),
    'min_samples_split': range(2, 11, 1),
    'min_samples_leaf': range(1, 11, 1),
    'max_features': ['sqrt', 'log2', None],
    'bootstrap': [True, False]
}

# On choisit un modèle sans préciser ses hyperparamètres
rfc = RandomForestClassifier(random_state=12)

# Partie cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=12)

# Va créer aléatoirement n_iter combinaisons de param_grid
random_search = RandomizedSearchCV(estimator=rfc, param_distributions=param_grid, n_iter=100,
                                   cv=cv, verbose=2, random_state=12, n_jobs=-1)

# Va tester ces hyperparamètres sur X_train et Y_train pour nous sortir la meilleure config
random_search.fit(X_train, Y_train)
```

V. Imbalanced learning : notre dataset



V. Imbalanced learning : notre dataset

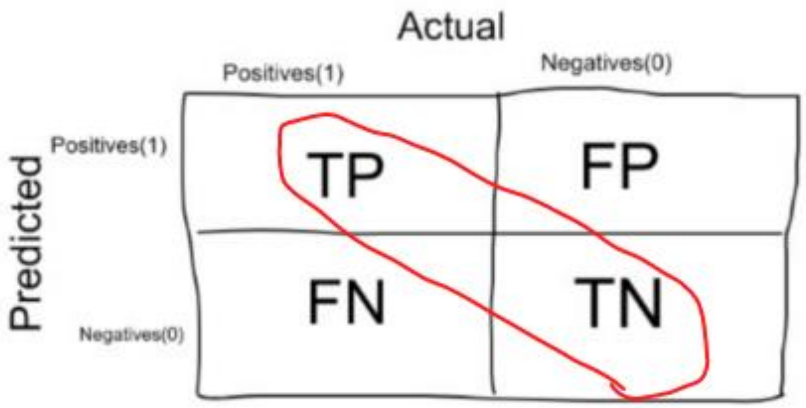
Métriques d'évaluation :



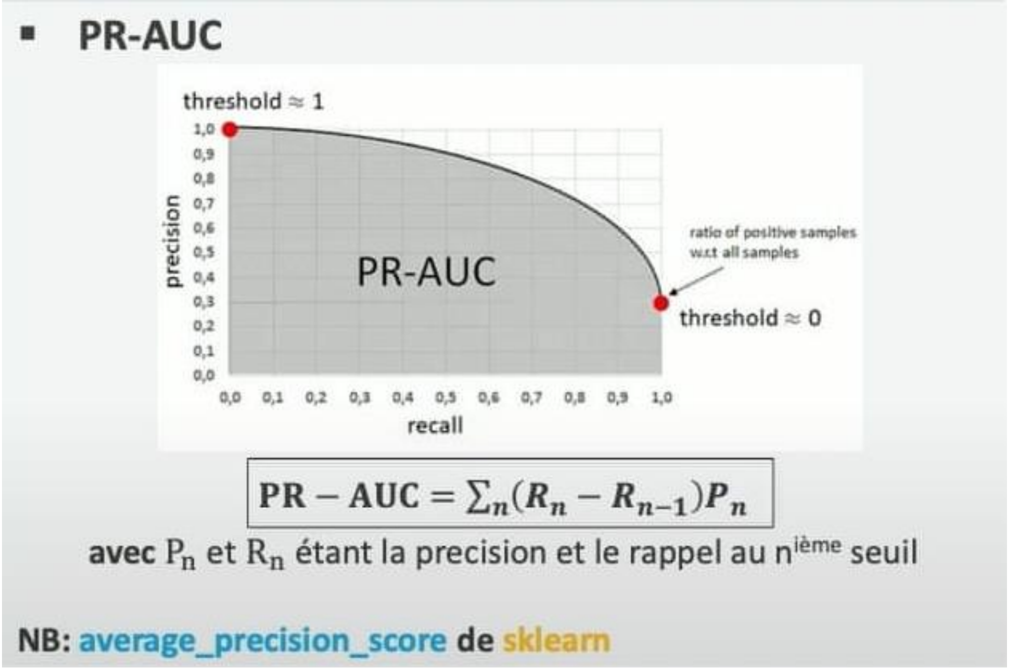
$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

V. Imbalanced learning : notre dataset

Métriques d'évaluation :

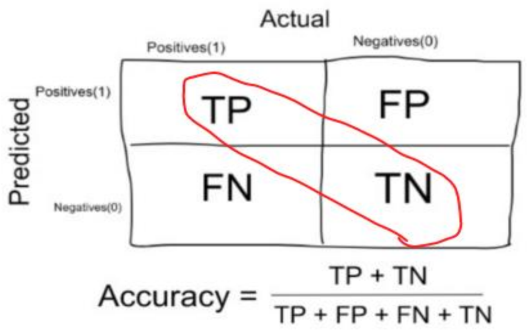


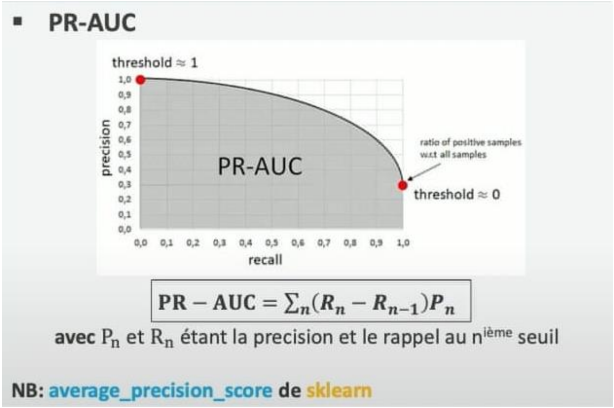
$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

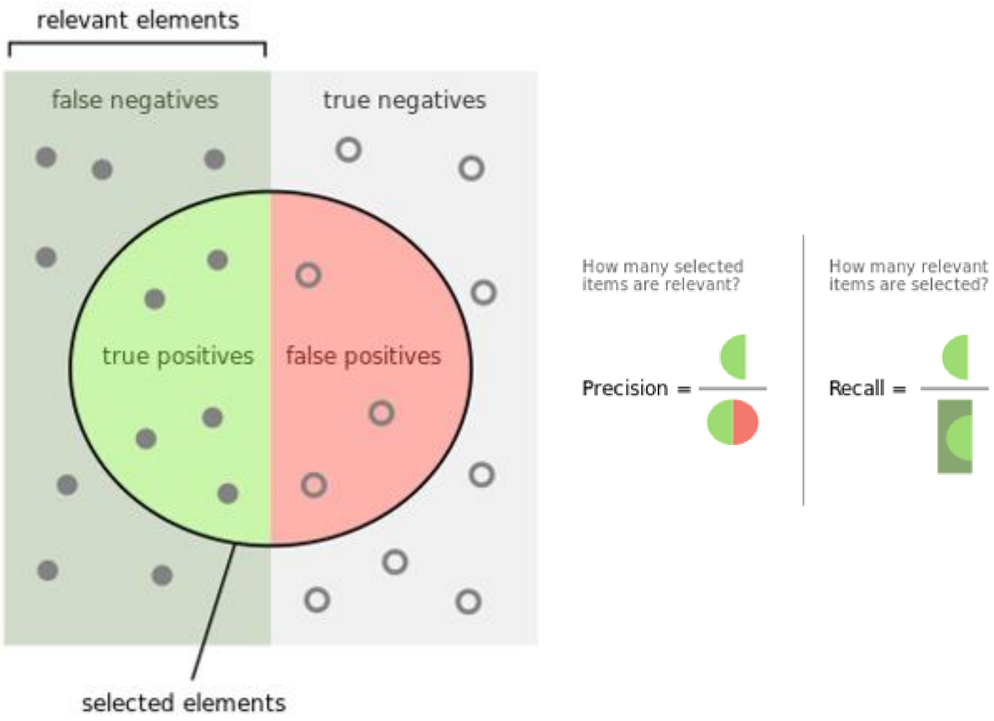


V. Imbalanced learning : notre dataset

Métriques d'évaluation :







V. Imbalanced learning : notre dataset

Résultats :

Sans undersampling

```
[15] # accuracy on test data
     Y_test_prediction = model.predict(X_test)
     test_data_accuracy = accuracy_score(Y_test_prediction, Y_test)
```

```
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9860437547149478
```

```
from sklearn.metrics import average_precision_score

average_precision = average_precision_score(Y_test,Y_test_prediction)
print(average_precision)
```

```
0.013956245285052269
```

V. Imbalanced learning : notre dataset

Résultats :

Undersampling, ratio = 1

```
[29] # accuracy on test data
     Y_test_prediction = model.predict(X_test)
     test_data_accuracy = accuracy_score(Y_test_prediction, Y_test)
```

```
[30] print('Accuracy score on Test Data : ', test_data_accuracy)

     Accuracy score on Test Data :  0.6231060606060606
```

```
from sklearn.metrics import average_precision_score

average_precision = average_precision_score(Y_test,Y_test_prediction)
print(average_precision)

0.6153496157821416
```

V. Imbalanced learning : notre dataset

Résultats :

Undersampling, ratio = 10

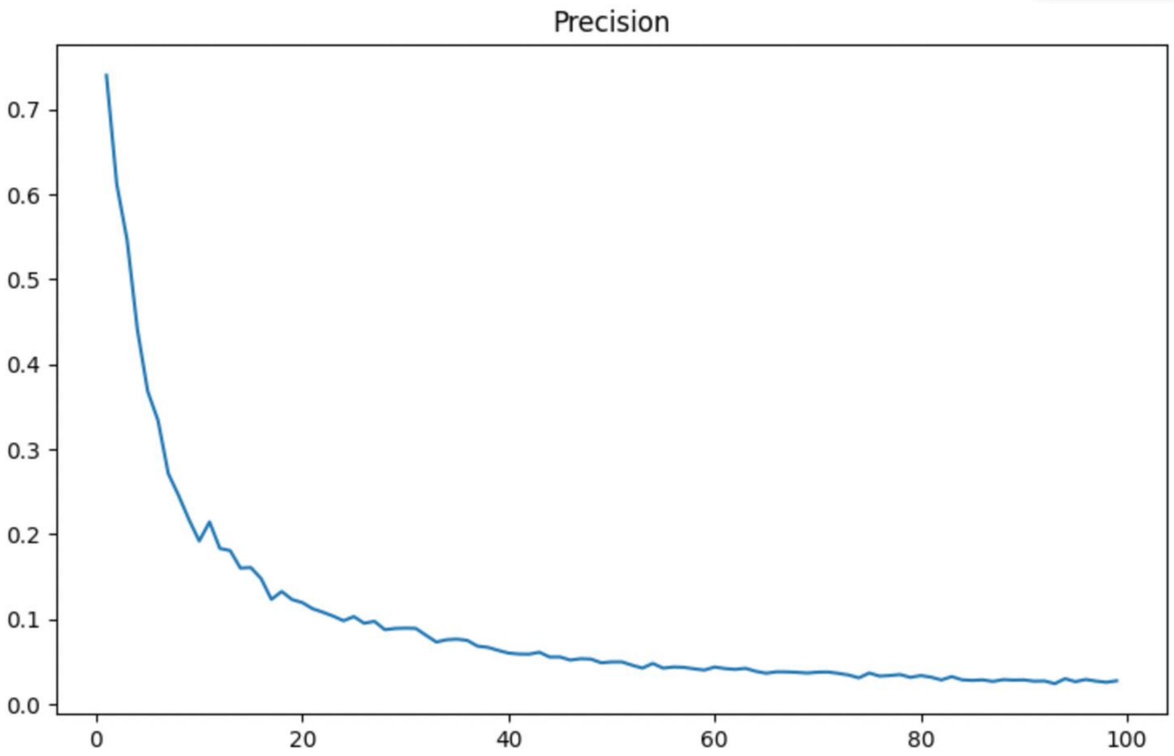
```
[39] # accuracy on test data
     Y_test_prediction = model.predict(X_test)
     test_data_accuracy = accuracy_score(Y_test_prediction, Y_test)
```

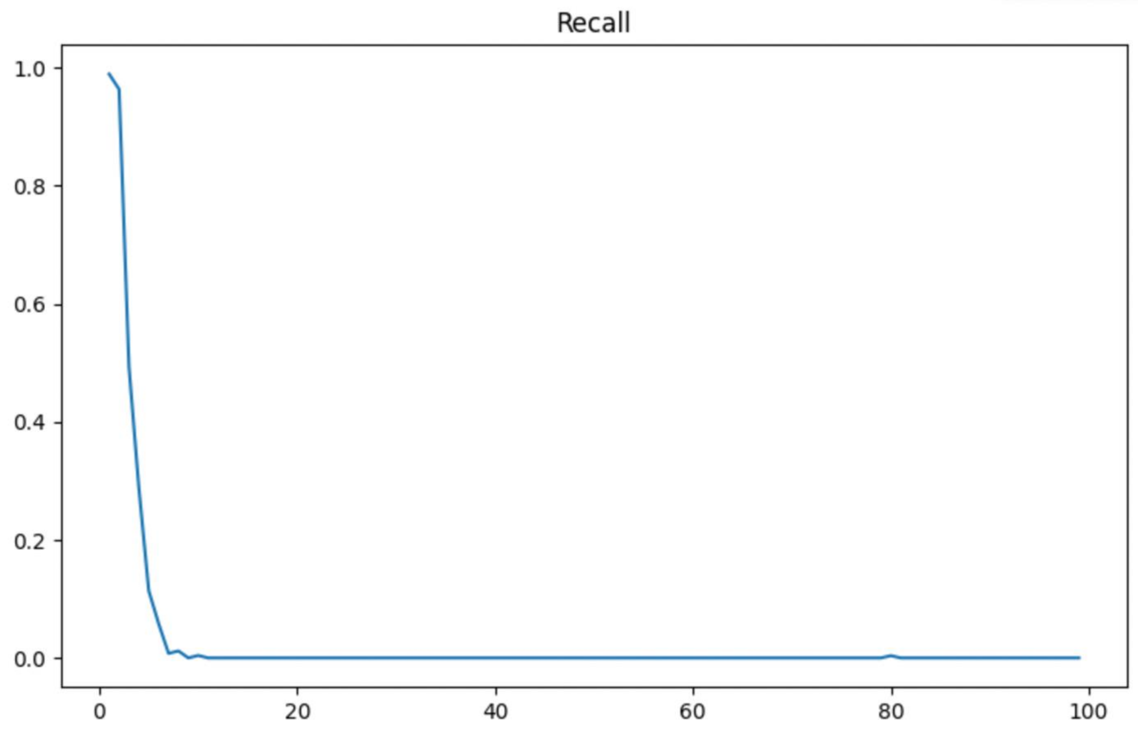
```
[40] print('Accuracy score on Test Data : ', test_data_accuracy)

     Accuracy score on Test Data :  0.8993797381116472
```

```
from sklearn.metrics import average_precision_score

average_precision = average_precision_score(Y_test,Y_test_prediction)
print(average_precision)

0.10062026188835287
```

V. Imbalanced learning : notre dataset

Résultats :





V. Imbalanced learning : Kaggle dataset

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.0907
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.2076
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.7530

	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
1412	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	0.244964	0
9083	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	-0.342475	0
4980	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	1.160686	0
8038	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	0.140534	0
3542	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	-0.073403	0

V. Imbalanced learning : Kaggle dataset

Résultats :

Undersampling, ratio = 1

```
[26] # accuracy on test data
     Y_test_prediction = model.predict(X_test)
     test_data_accuracy = accuracy_score(Y_test_prediction, Y_test)
```

```
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9990519995786665
```

Métrique d'évaluation de BNP

```
from sklearn.metrics import average_precision_score

average_precision = average_precision_score(Y_test,Y_test_prediction)
print(average_precision)
```

```
0.49451688374192126
```

V. Imbalanced learning : Kaggle dataset

Résultats :

Undersampling, ratio = 10

```
[50] # accuracy on test data
     Y_test_prediction = model.predict(X_test)
     test_data_accuracy = accuracy_score(Y_test_prediction, Y_test)
```

```
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9787626962142197
```

Métrique d'évaluation de BNP

```
from sklearn.metrics import average_precision_score

average_precision = average_precision_score(Y_test,Y_test_prediction)
print(average_precision)
```

```
0.7849278215441301
```

V. Imbalanced learning : Kaggle dataset

Résultats :

Undersampling, ratio = 100

```
[63] # accuracy on test data
     Y_test_prediction = model.predict(X_test)
     test_data_accuracy = accuracy_score(Y_test_prediction, Y_test)
```

```
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9957742227588289
```

Métrique d'évaluation de BNP

```
from sklearn.metrics import average_precision_score

average_precision = average_precision_score(Y_test,Y_test_prediction)
print(average_precision)
```

```
0.6161590805957154
```

VI. Conclusion et ouverture

Imbalanced learning → Cost-sensitive techniques

- **Benchmark 1 :** $\text{PR-AUC}_1 = 0,017$
  Le premier benchmark est naïf et considère un modèle qui prédit aléatoirement une probabilité entre 0 et 1.
- **Benchmark 2 :** $\text{PR-AUC}_2 = 0,14$
  Le second benchmark intègre plusieurs étapes de pré-processing et utilise un modèle de Machine Learning optimisé pour prédire le risque de fraude.

```
print(average_precision_score(Y_test, y_pred2))

0.15726139817331086
```

Bibliographie

- Logistic Regression — Detailed Overview | by Saishruthi Swaminathan | Towards Data Science
- https://scikit-learn.org/
- La star des algorithmes de ML : XGBoost - datacorner par Benoit Cayla
- https://fraud-detection-handbook.github.io/fraud-detection-handbook/
- (2) Random Forest Algorithm Clearly Explained! - YouTube
- Credit Card Fraud Detection | Kaggle
- Challenge data (ens.fr)