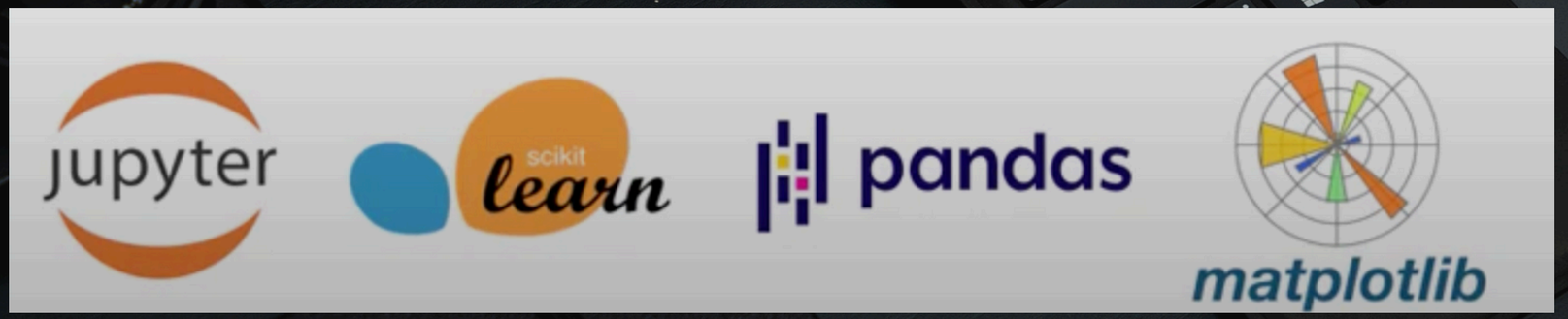
MACHINE LEARNING

SVM

SUPPORT VECTOR MACHINE



BRENDAN SOUFFIANE MOUSTAPHA ROMAIN

HETIC

SVM

FLEXIBILITÉ

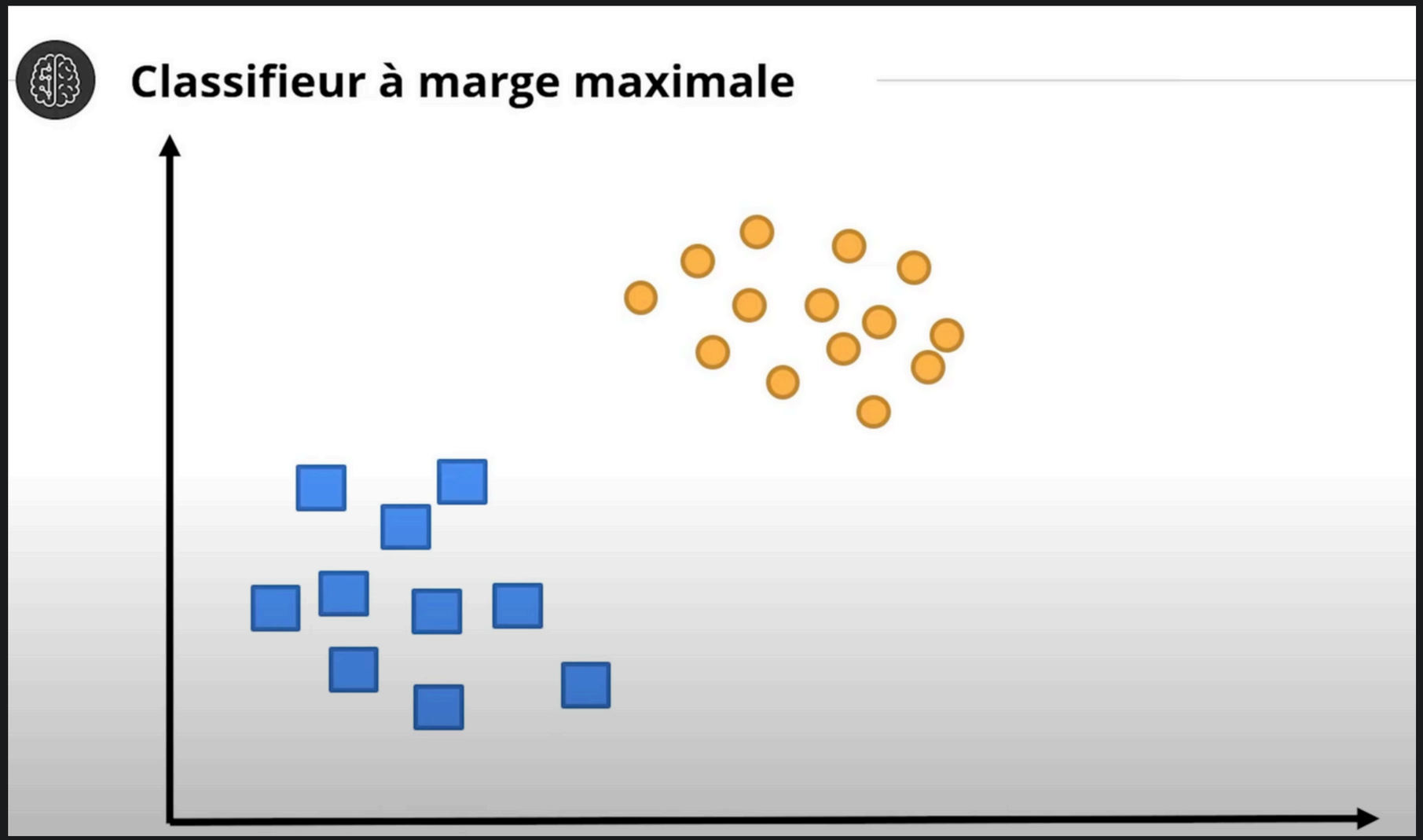
SIMPLICITÉ D’UTILISATION

BIOINFORMATIQUE

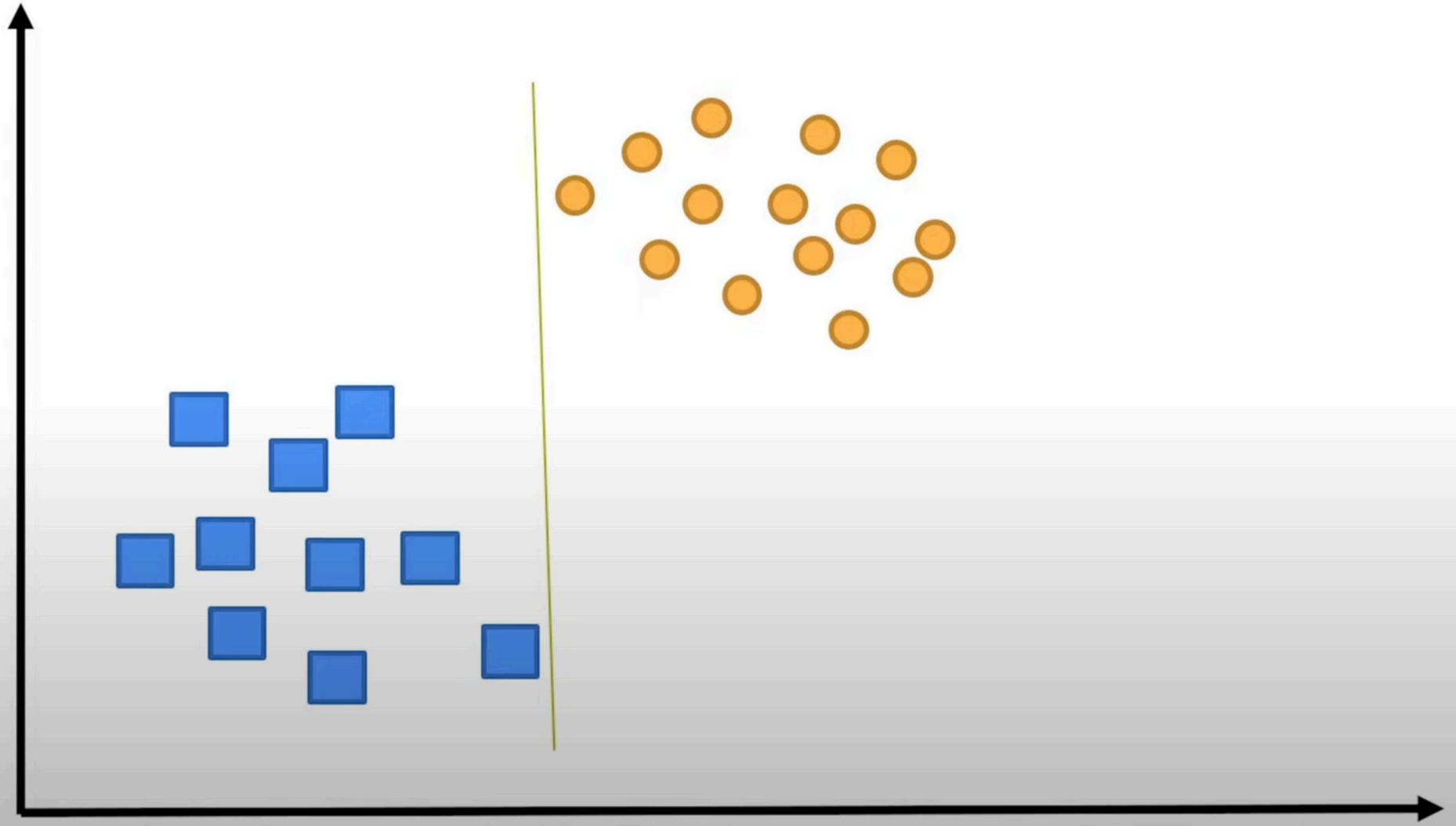
FINANCE

RECHERCHE D’INFORMATION.

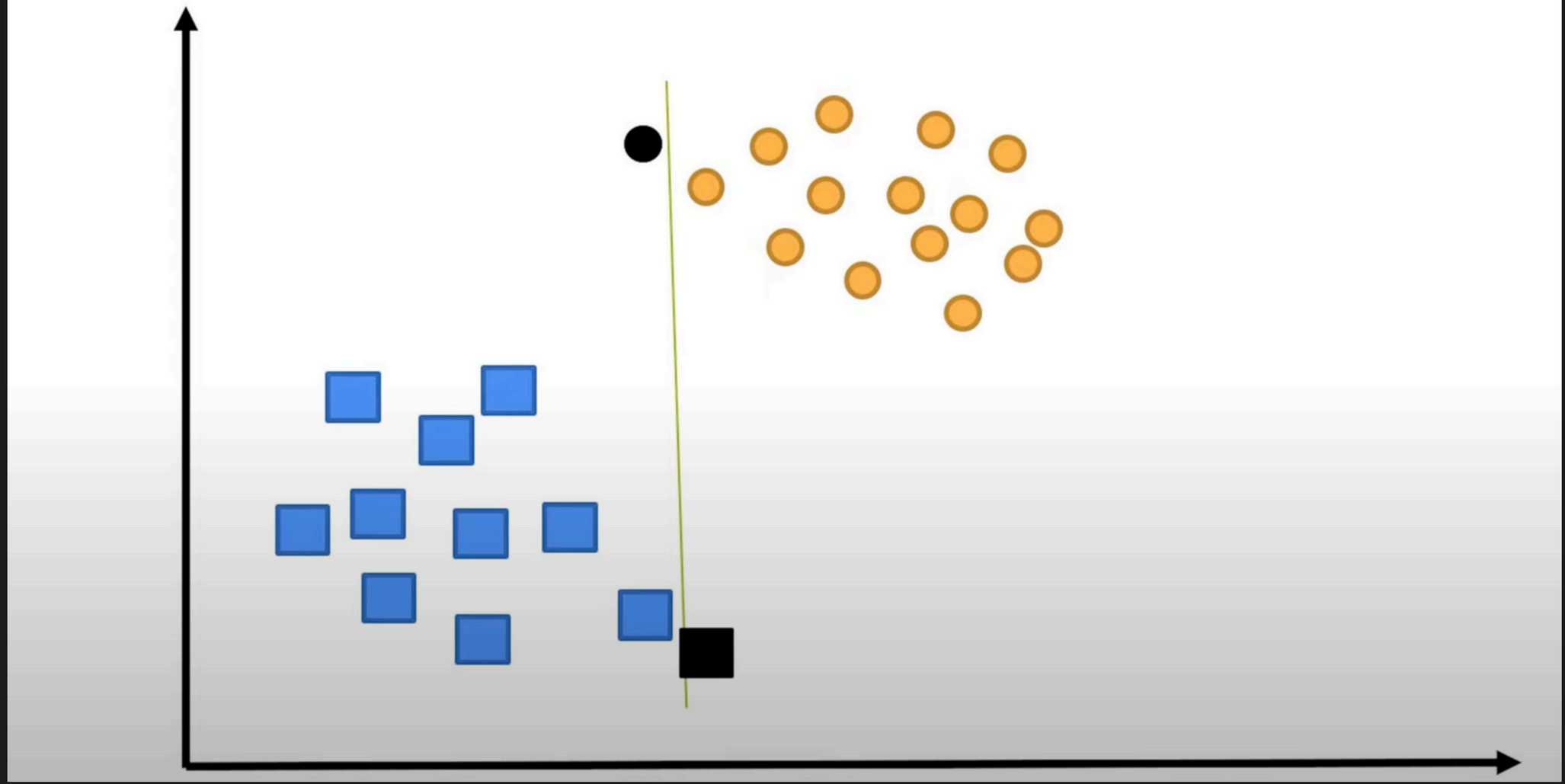
Classifieur à marge maximale



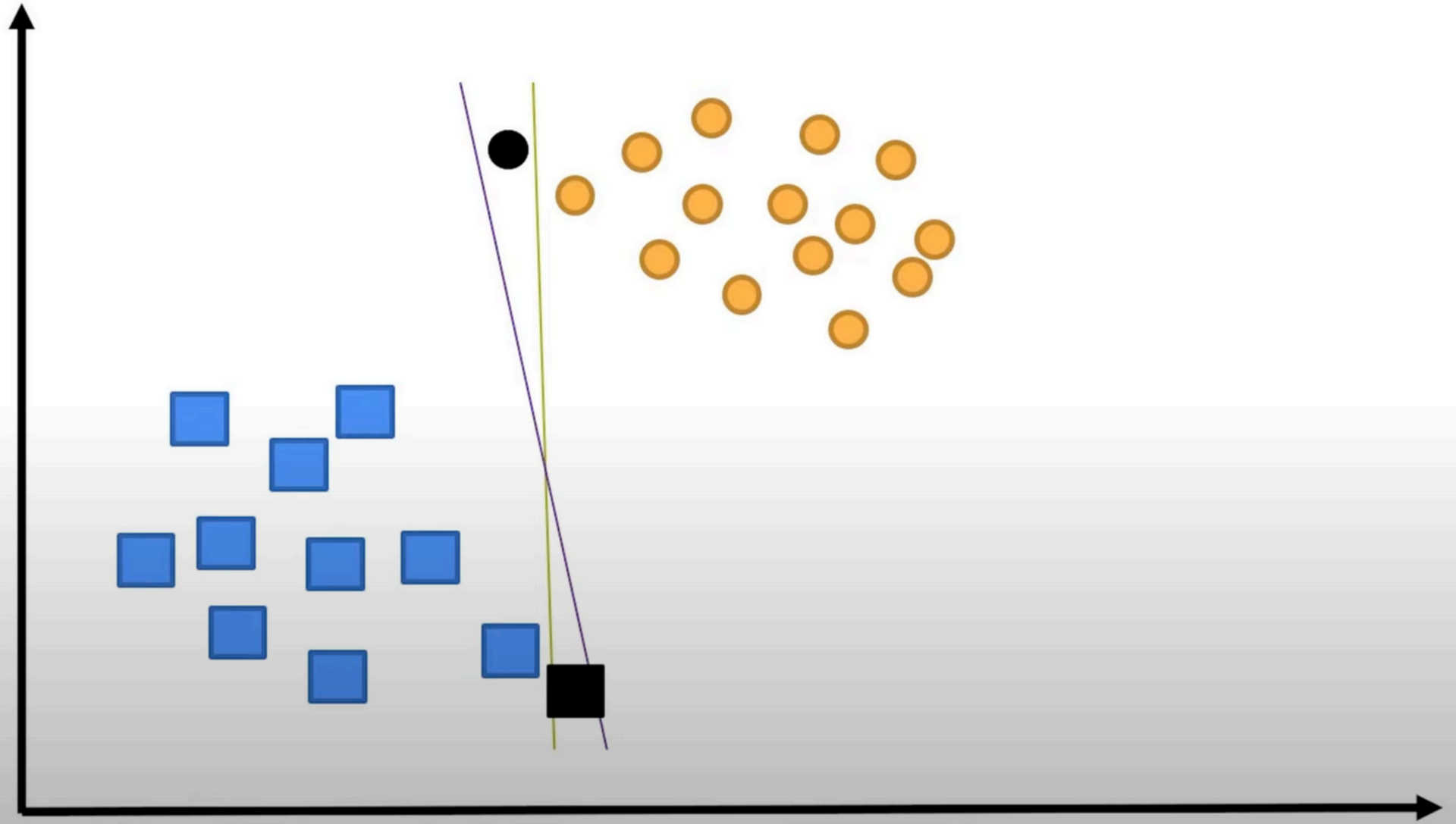
Classifieur à marge maximale



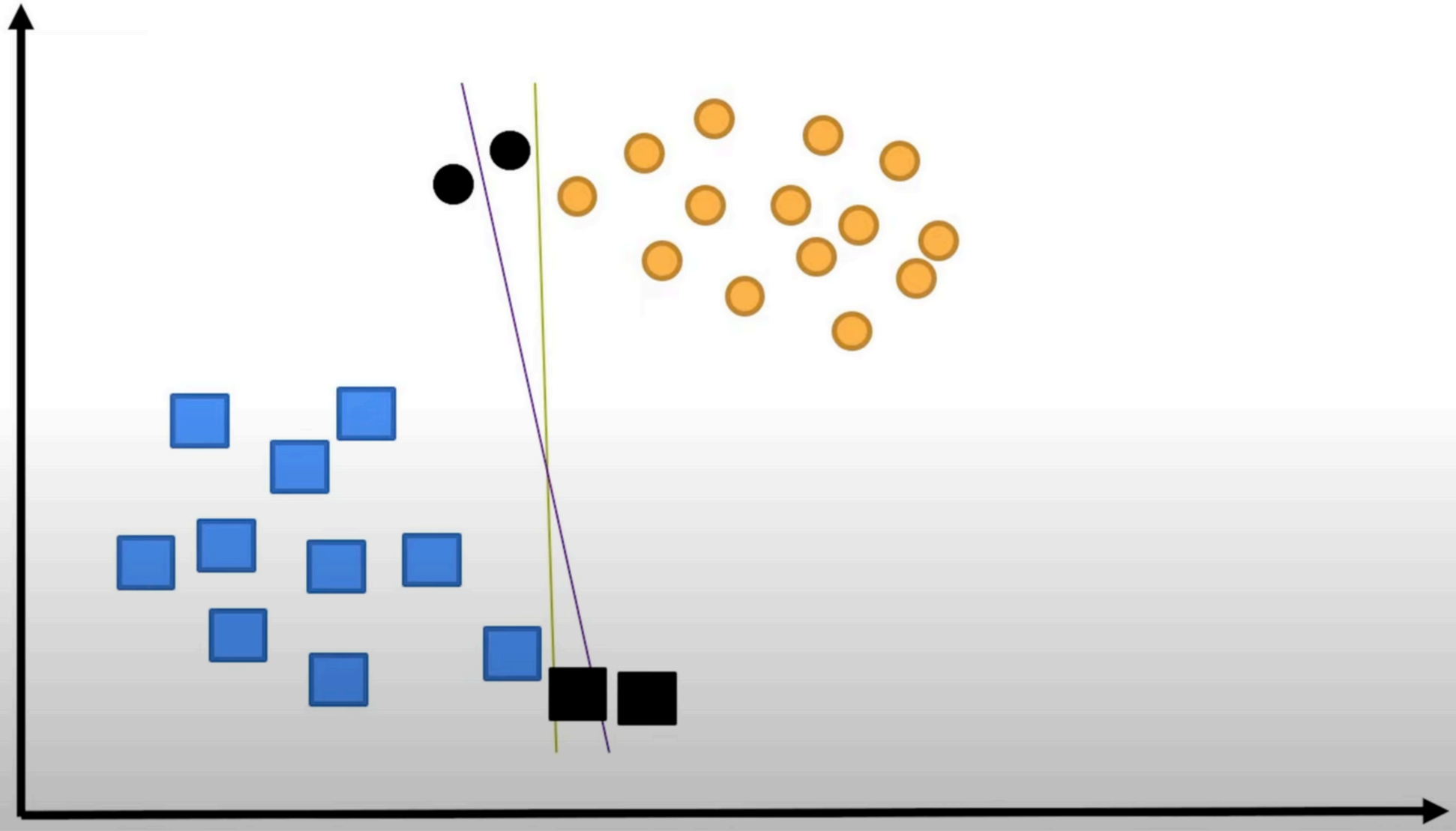
Classifieur à marge maximale



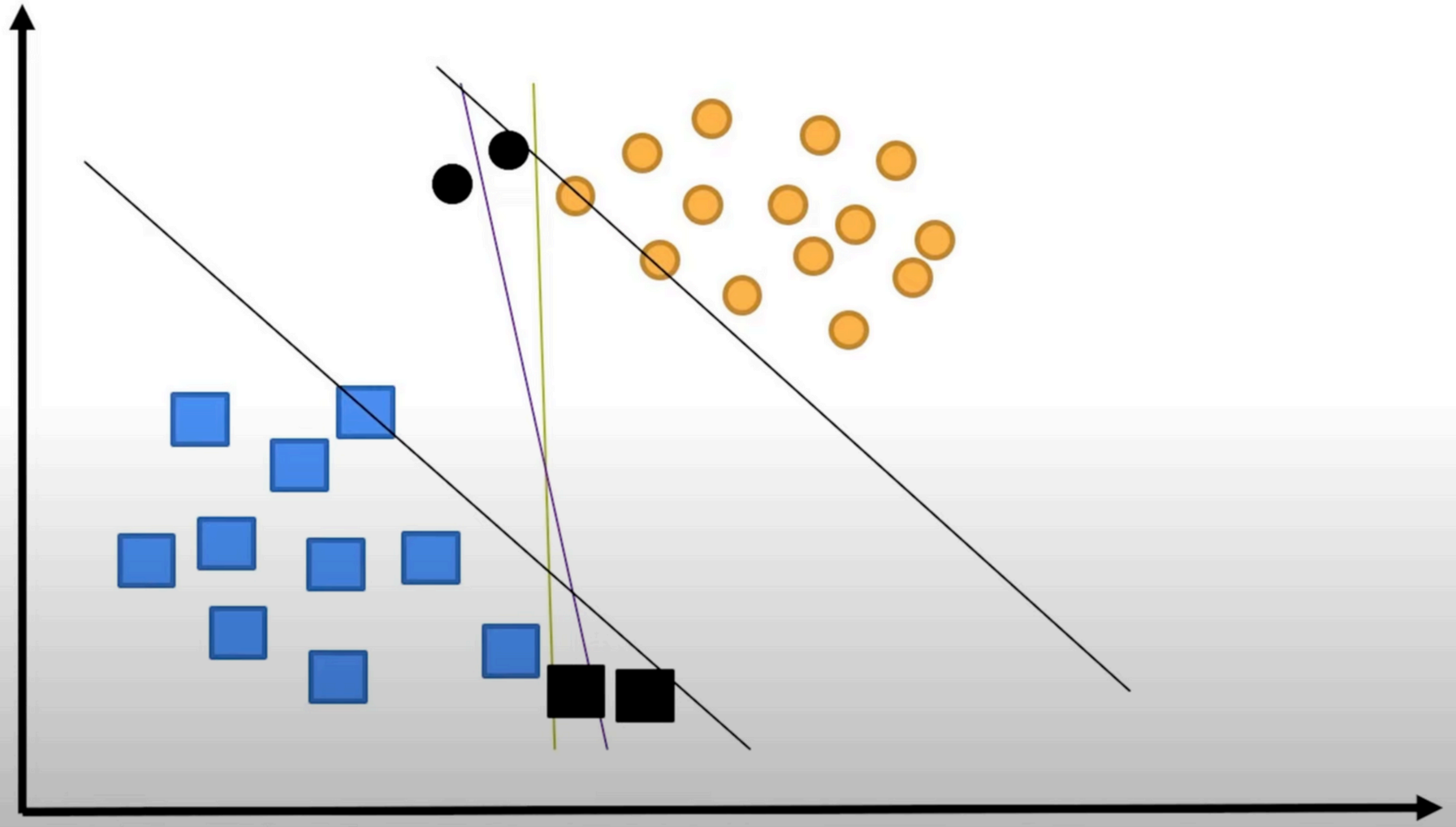
Classifieur à marge maximale



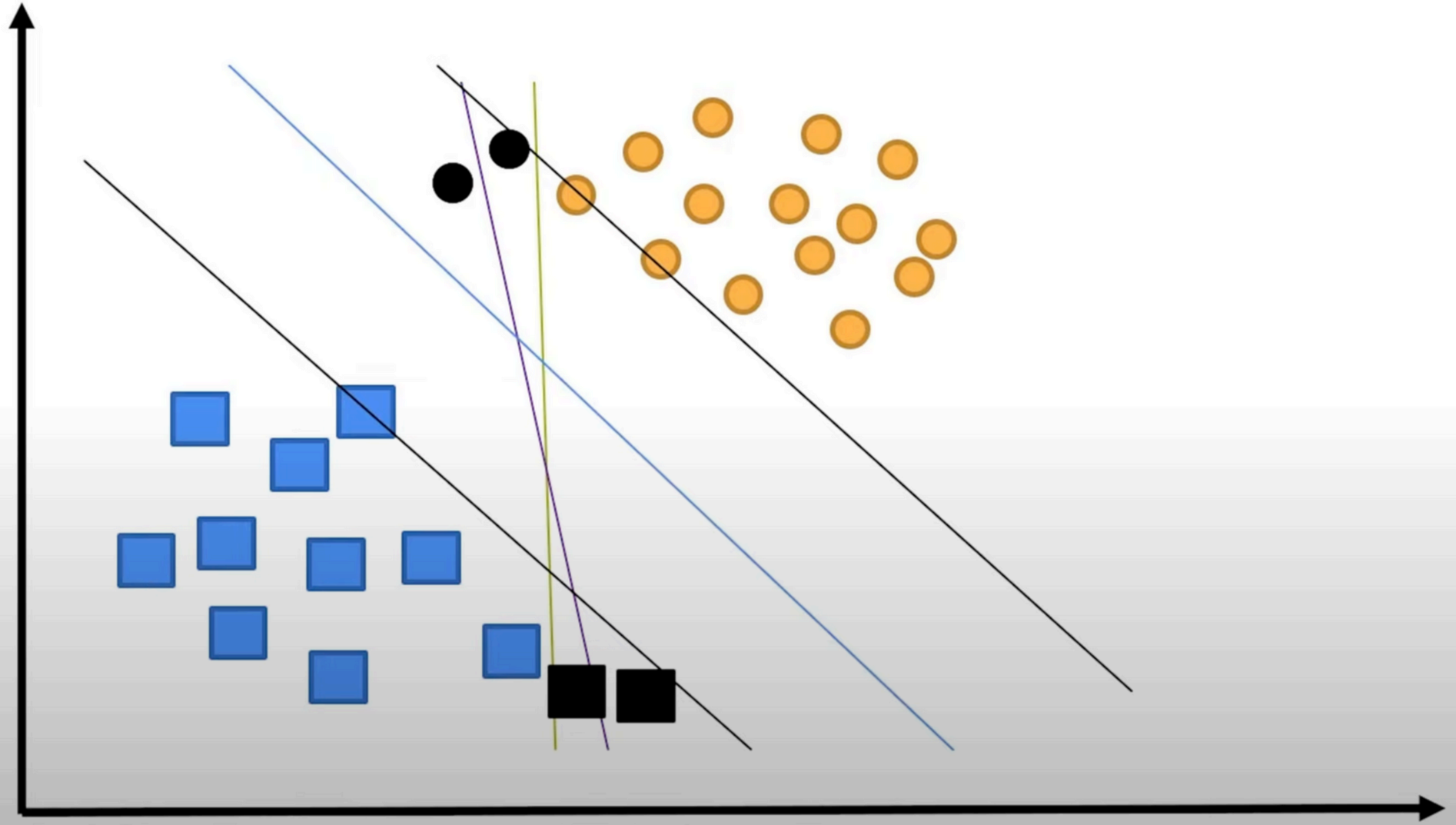
Classifieur à marge maximale



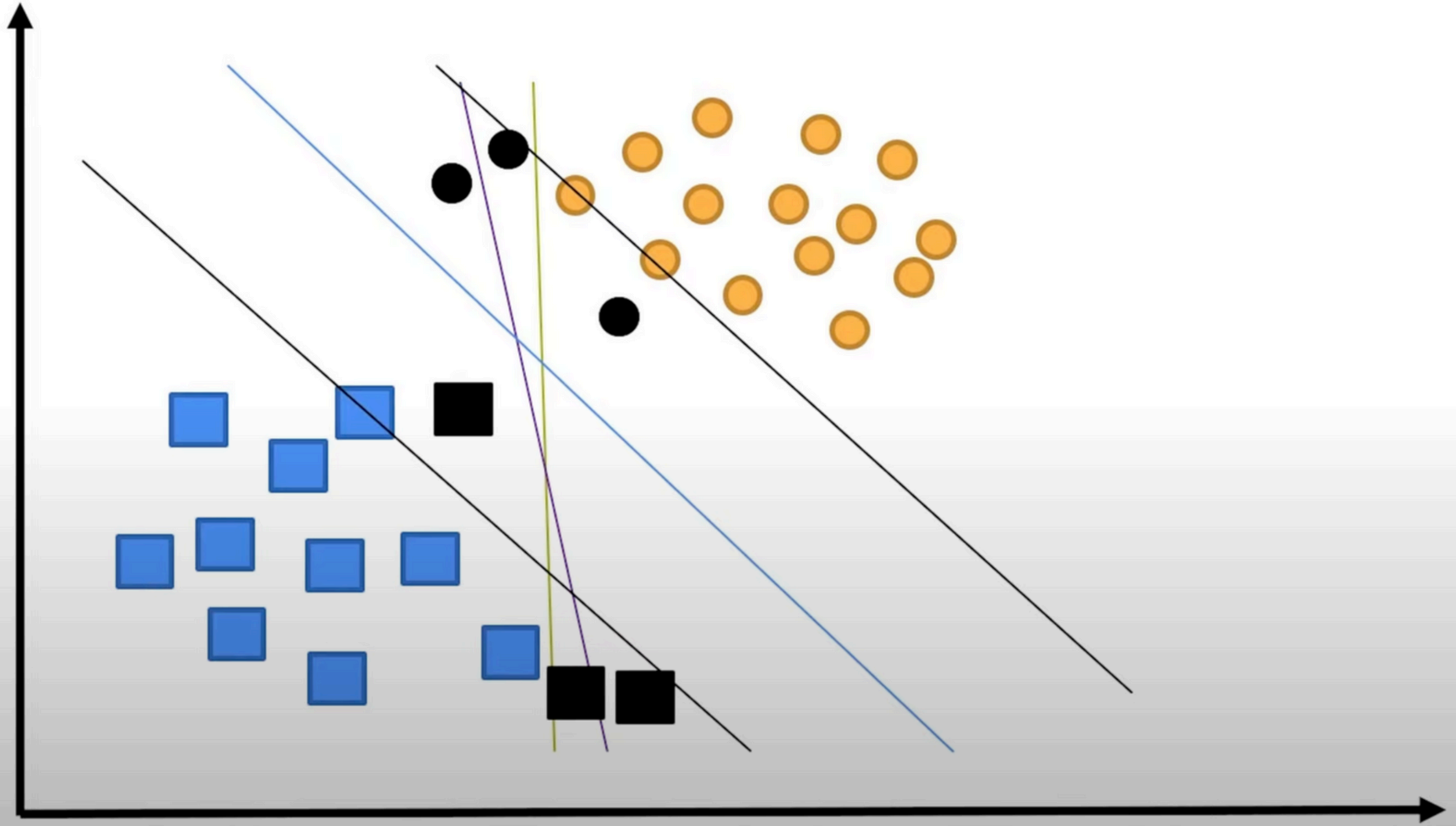
Classifieur à marge maximale



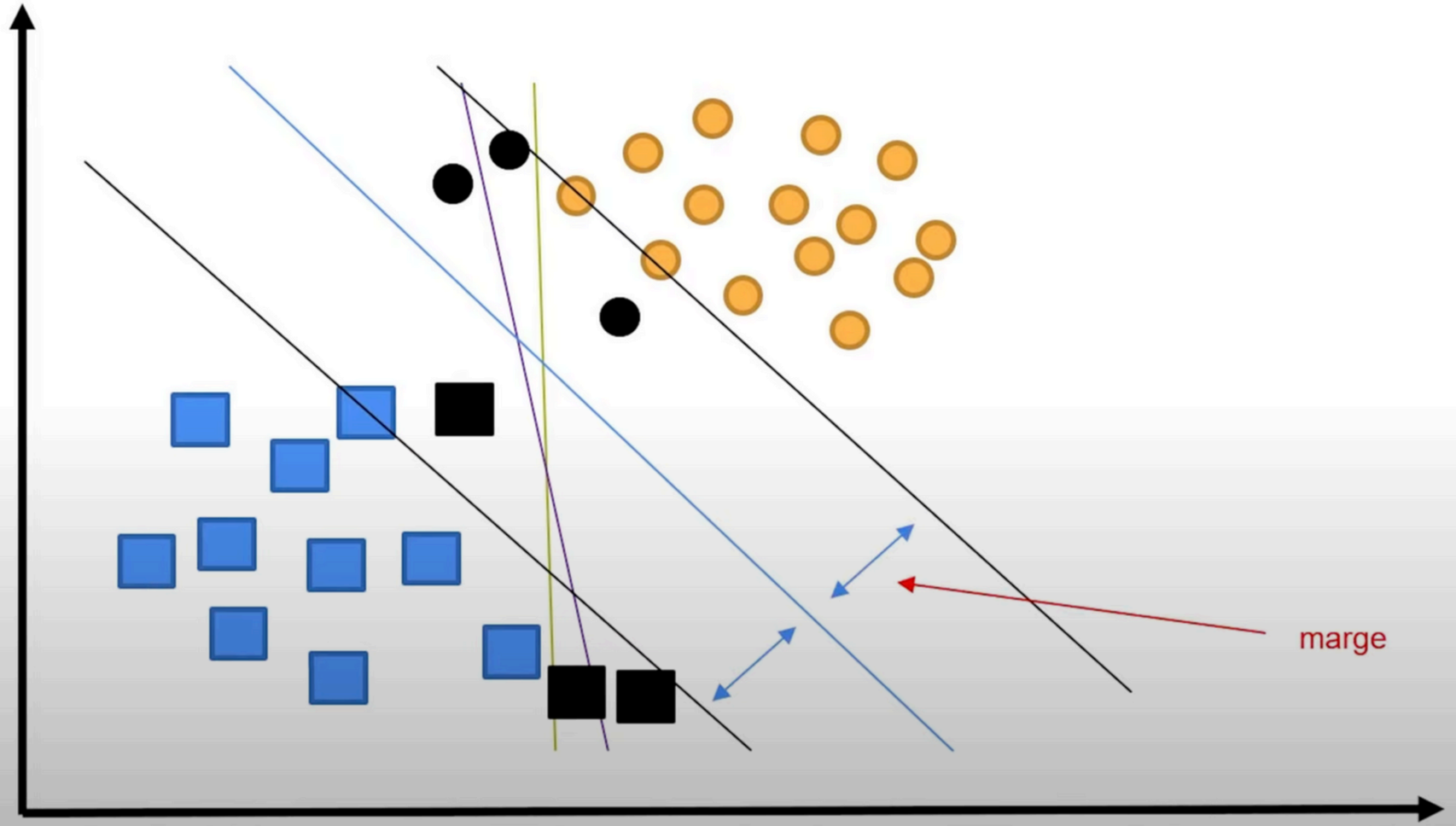
Classifieur à marge maximale



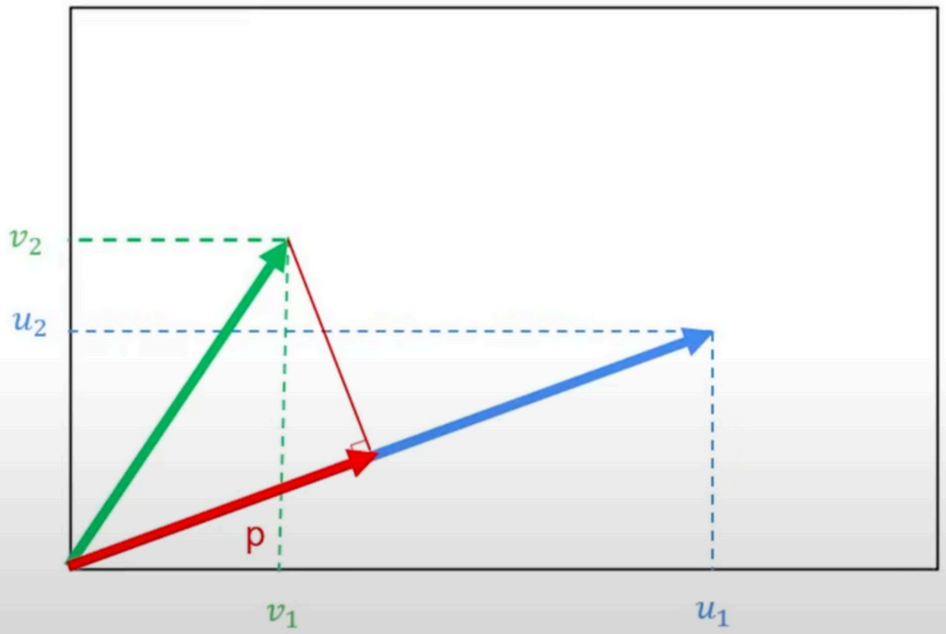
Classifieur à marge maximale



Classifieur à marge maximale



Produit scalaire



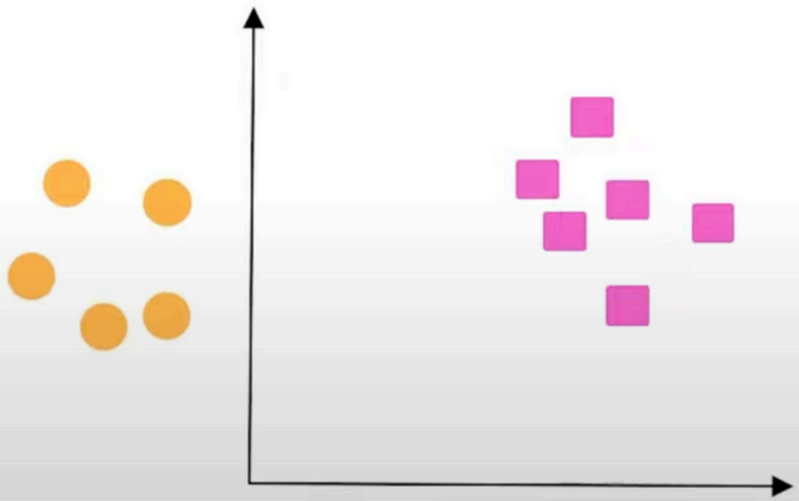
$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \; ; \quad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$\|u\| = longueur\ du\ vecteur\ u$$
$$= \sqrt{{u_1}^2 - {u_2}^2} \in \mathbb{R}$$

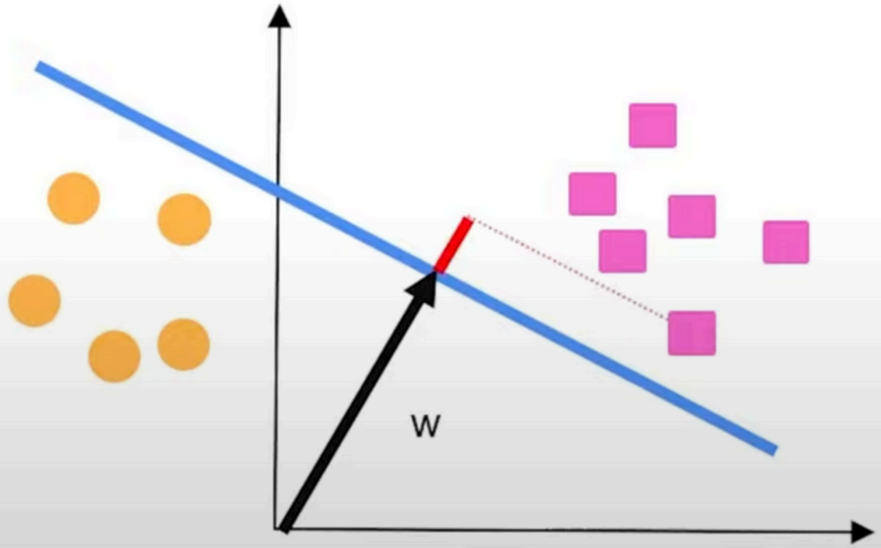
$$P = longueur\ de\ la\ projection\ de\ v\ sur\ u$$

$$u^T v = p\,.\,\|u\|$$
$$= u_1 v_1 + u_2 v_2$$

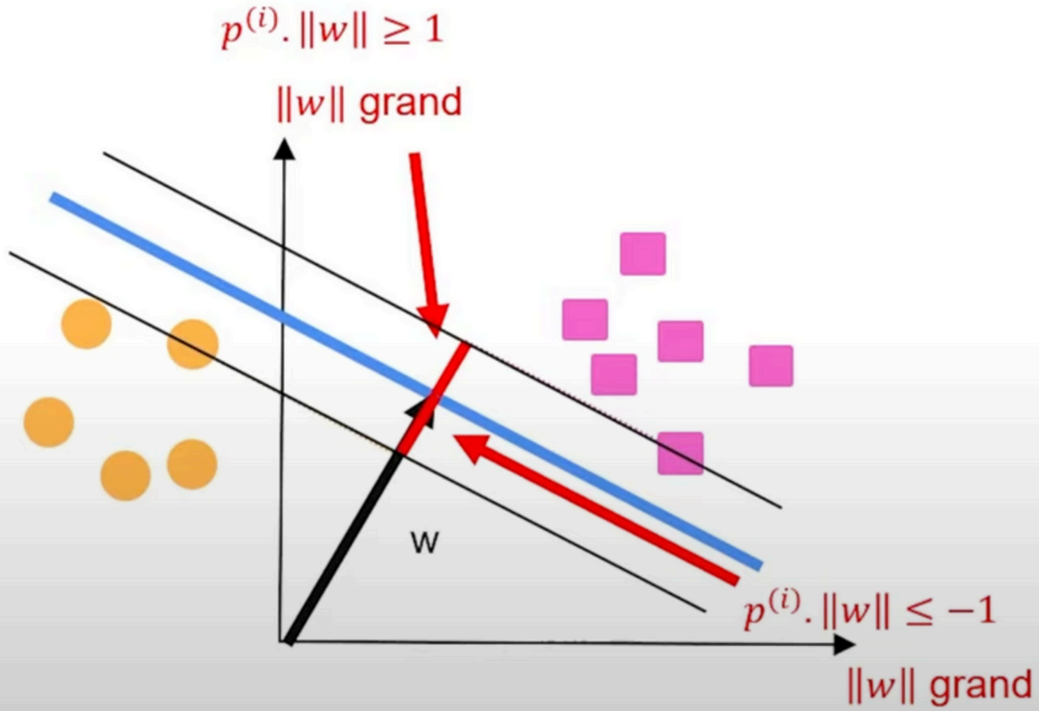
Les vecteurs de support



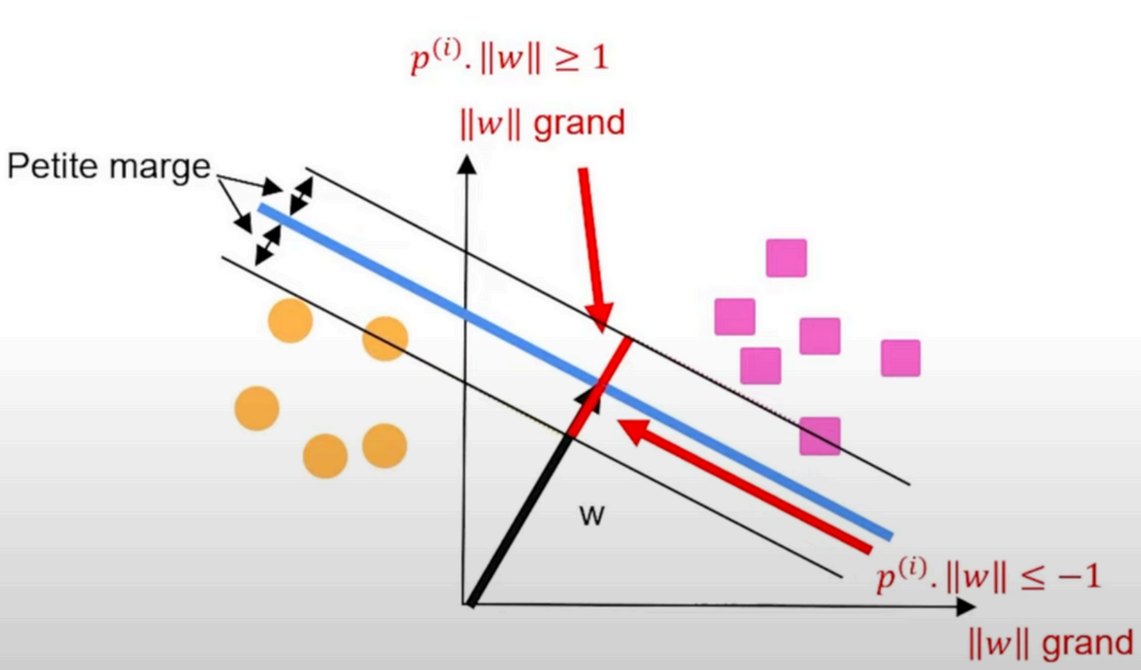
Les vecteurs de support

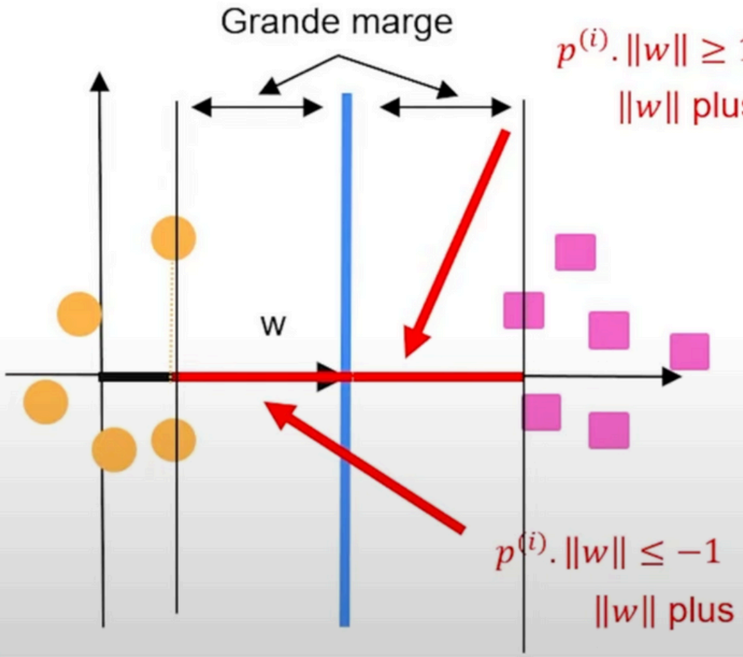


Les vecteurs de support



Les vecteurs de support

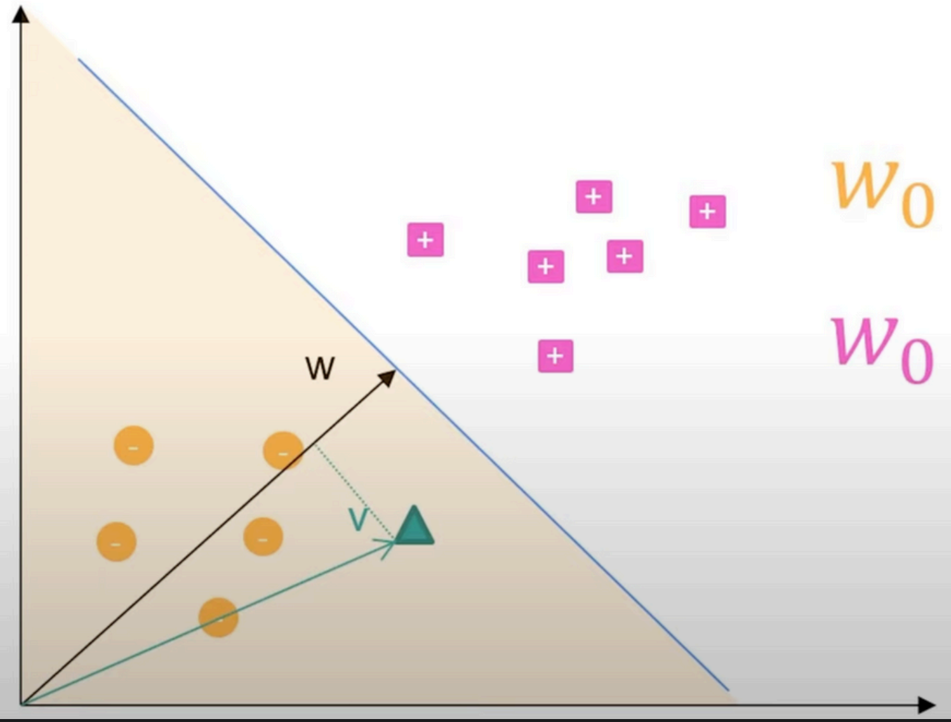




$$\min_w \frac{1}{2}\|w\|^2$$

Frontière de décision

$w_0 + v.w$



$w_0 + v_-.w \leq -1$

$w_0 + v_+.w \geq 1$

Calcul de l'erreur

$$w_0 + v_-.w \leq -1$$

$$y_-(w_0 + v_-.w) \geq 1$$

$$y_-(w_0 + v_-.w) - 1 \geq 0$$

$$y \begin{cases} y_- = -1 \\ y_+ = 1 \end{cases}$$

$$w_0 + v_+.w \geq 1$$

$$y_+(w_0 + v_+.w) \geq 1$$

$$y_+(w_0 + v_+.w) - 1 \geq 0$$

$$y_i(w_0 + v_i.w) - 1 \geq 0$$

$$i = [-1,1]$$

$$min\frac{1}{m}\sum \min(y_i(w_0 + v_i.w) - 1, 0)$$

$$-y_i(w_0 + v_i.w) + 1 \leq 0$$

$$i = [-1,1]$$

$$min\frac{1}{m}\sum max(1 - y_i(w_0 + v_i.w), 0)$$

Fonction de coût

$$\min_w \frac{1}{2}\|w\|^2 \qquad \min_w \frac{1}{m}\sum max(1 - y_i(w_0 + v_i.w), 0)$$

$$\min_w \frac{1}{2}\|w\|^2 + \frac{1}{m}\sum max(1 - y_i(w_0 + v_i.w), 0)$$

Hinge loss $$\min_w J(w) = \frac{1}{2}\|w\|^2 + C\frac{1}{m}\sum max(1 - y_i(w_0 + v_i.w), 0)$$

Paramètre de régularisation

Gradient descent

Soit n le nombre de variables

Répéter ce processus jusqu'à la convergence :

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(W)$$

$$\mathrm{J}(w) = \frac{1}{2}\|w\|^2 + C\frac{1}{m}\sum \mathrm{max}(1 - y_i(w_0 + v_i.w), 0)$$

$$\frac{\partial}{\partial w_j} J(W) = \begin{cases} w, & si\ 1 - y_i(w_0 + v_i.w) \leq 0 \\ w - \sum \frac{c}{m} y_i v_i, & sinon \end{cases}$$

CLASSIFICATION DES BILLETS

VARIANCE
SKEWNESS
CURTOSIS
ENTROPY

AUTHENTIQUE(1)

NON AUTHENTIQUE(1)

SVM : PRATIQUE

bill_authentification

amount	merchant_category	location_code	hour	auth
27.983604115376476	37.37897510478274	33.920035456522925	28.586984000871812	0
66.46786504192688	64.92878474560648	68.8954766340461	69.28512356811173	1
64.53895217102374	70.62495822385505	70.13427864125435	66.87177036643688	1
66.71298015756908	68.74310891683908	65.70573775098944	67.59200400597828	1
31.515039300986793	19.566182233685403	27.564150043233305	32.11195503119391	0
72.81305319369359	76.15667337709698	66.0186393161957	66.34923439006103	1
71.44261921244478	70.14885588609762	74.79541459552856	72.42212216210855	1
33.48723369324674	26.42935896672908	30.20381920322219	37.82664886227327	0
73.24118956524708	70.7041146616118	68.89441659666612	76.64280326173852	1
74.26282219509692	69.33184599045445	71.23330511287666	65.245008131788	1
26.82621064151152	22.366693384326247	29.808143584622687	36.39137018052579	0
67.71026395304946	81.25997605721415	75.39431790132623	80.17540609799106	1

Fichier Traité

```
#Importer les librairies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
```

```
#importer dataset
Data= pd.read_csv('bill_authentification.csv')
x=Data.iloc[:,:-1].values
y=Data.iloc[:,-1].values
```

```
#Apercu des données
#Data.head()
Data.shape
```

```
(1372, 5)
```

```
from os import X_OK
#Diviser les donées en jeu d'entrainement et jeu de test
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.20)
```

```
#Créer le modèle SVM
classifier=SVC(kernel='linear', random_state=0)
classifier.fit(X_train,y_train)
```

```
SVC
SVC(kernel='linear', random_state=0)
```

```
#Prédiction sur le Test set
y_pred=classifier.predict(X_test)
```

```
#Matrice de confusion
cm=confusion_matrix(y_test,y_pred)
```

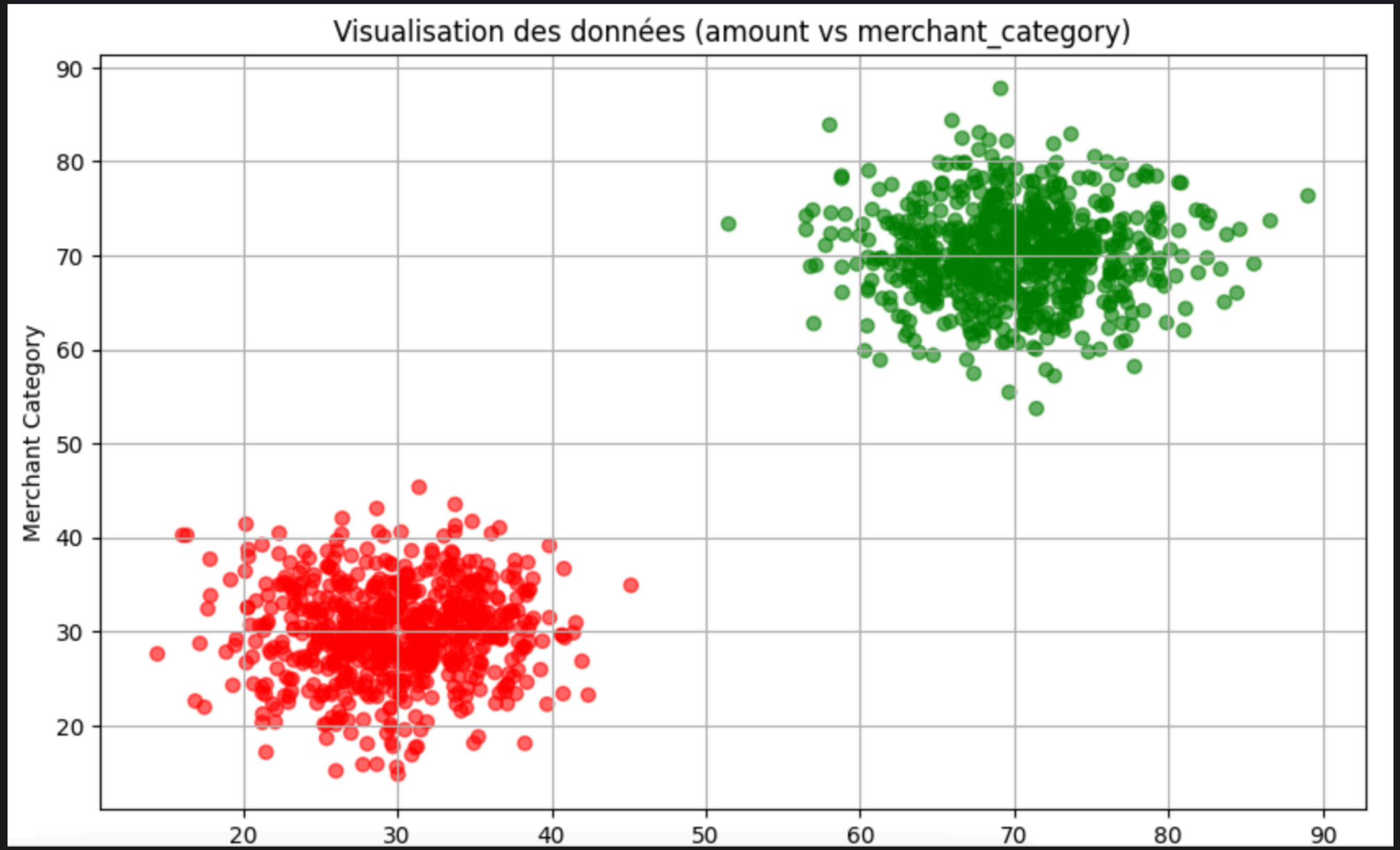
```python
#Rapport de classification
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       146
           1       1.00      1.00      1.00       129

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275
```

```python
# Remplacer par ton nom de fichier si différent
df = pd.read_csv("bill_authentification.csv")
# Afficher un aperçu des données
print(df.head())
```

```
      amount  merchant_category  location_code       hour  auth
0  27.983604          37.378975      33.920035  28.586984     0
1  66.467865          64.928785      68.895477  69.285124     1
2  64.538952          70.624958      70.134279  66.871770     1
3  66.712980          68.743109      65.705738  67.592004     1
4  31.515039          19.566182      27.564150  32.111955     0
```

```python
# 4. Tracer un nuage de points avec deux premières colonnes (ex: amount vs merchant_category)
plt.figure(figsize=(10,6))
colors = ['red' if label == 0 else 'green' for label in df['auth']]  # colorier selon la classe

plt.scatter(df['amount'], df['merchant_category'], c=colors, alpha=0.6)
plt.title("Visualisation des données (amount vs merchant_category)")
plt.xlabel("Amount")
plt.ylabel("Merchant Category")
plt.grid(True)
plt.show()
```


Visualisation des données (amount vs merchant_category)
Merchant Category
90
80
70
60
50
40
30
20
20
30
40
50
60
70
80
90

```python
#Codage Remplacer les fonctions de sklearn
import pandas as pd

#Visualisation de données
import matplotlib.pyplot as plt

# 1. Charger les données
df = pd.read_csv("bill_authentification.csv")

# 2. Convertir en tableaux numpy
#value converti Dataframe en tableau Numpy
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

#value converti Dataframe en tableau Numpy

# 3. Normaliser les étiquettes en -1 et +1 (requis pour SVM)
# important pour la fonction de hinge
y = np.where(y == 0, -1, 1)
```

```python
# 4. Diviser les données manuellement (80% entraînement, 20% test)
class SimpleSVM:
    def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000):
        self.lr = learning_rate         # Taux d'apprentissage (learning rate)
        self.lambda_param = lambda_param  # Paramètre de régularisation
        self.n_iters = n_iters          # Nombre d'itérations d'entraînement
        self.w = None                   # Poids (vecteur)
        self.b = None                   # Biais (scalaire)

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.w = np.zeros(n_features)  # Initialisation des poids à zéro
        self.b = 0                     # Initialisation du biais à zéro

        # Boucle d'optimisation (descente de gradient)
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                # Calcul de la condition SVM (marge ≥ 1)
                condition = y[idx] * (np.dot(x_i, self.w) + self.b) >= 1

                if condition:
                    # Si marge correcte, seulement régularisation des poids
                    self.w -= self.lr * (2 * self.lambda_param * self.w)
                else:
                    # Sinon, mise à jour avec contribution du terme hinge loss
                    self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y[idx]))
                    self.b -= self.lr * y[idx]
```

```
    def predict(self, X):
        approx = np.dot(X, self.w) + self.b  # Calcul du score linéaire
        return np.sign(approx)                # Retourne -1 ou +1 selon le signe (prédiction)


# 6. Entraîner le modèle
model = SimpleSVM()
model.fit(X_train, y_train)

# 7. Prédictions
y_pred = model.predict(X_test)

# 8. Reconvertir -1/+1 en 0/1 pour comparaison
y_test_binary = np.where(y_test == -1, 0, 1)
y_pred_binary = np.where(y_pred == -1, 0, 1)

# 9. Matrice de confusion manuelle
def compute_confusion_matrix(y_true, y_pred):
    TP = np.sum((y_true == 1) & (y_pred == 1)) #(Vrai +) : nombre d'ex correctement prédits + (1)
    TN = np.sum((y_true == 0) & (y_pred == 0)) #(Vrai -) : nombre d'ex correctement prédits - (0).
    FP = np.sum((y_true == 0) & (y_pred == 1)) #(Faux +) : ex - prédit comme + (faux +).
    FN = np.sum((y_true == 1) & (y_pred == 0)) #(Faux -) : ex + prédit comme - (faux -).
    return np.array([[TN, FP], [FN, TP]])
```

```python
# 10. Rapport de classification manuel
def classification_metrics(cm): #entrée cm, une matrice de confusion 2x2
    TN, FP, FN, TP = cm[0,0], cm[0,1], cm[1,0], cm[1,1]
    accuracy = (TP + TN) / (TP + TN + FP + FN) # Exactitude proportion de bonnes prédictions (+ et -) sur l'ensemble des prédictions.
    precision = TP / (TP + FP) if (TP + FP) else 0 #parmi tous les éléments prédits positifs, quelle proportion est réellement positive.
    recall = TP / (TP + FN) if (TP + FN) else 0 #sensibilité : parmi tous les vrais + réels, quelle proportion a été correctement identifiée.
    #Score F1 : moyenne harmonique entre précision et rappel, donne un équilibre entre les deux.
    f1 = 2 * precision * recall / (precision + recall) if (precision + recall) else 0
    return accuracy, precision, recall, f1

#On calcule la matrice de confusion entre les vraies étiquettes et les prédictions.
cm = compute_confusion_matrix(y_test_binary, y_pred_binary)

#On calcule les métriques (accuracy, précision, rappel, F1) à partir de cette matrice.
acc, prec, rec, f1 = classification_metrics(cm)

# 11. Affichage
print("Matrice de confusion :\n", cm)
print(f"Exactitude (Accuracy): {acc:.2f}")
print(f"Précision: {prec:.2f}")
print(f"Rappel (Recall): {rec:.2f}")
print(f"F1-score: {f1:.2f}")
```

MERCI !