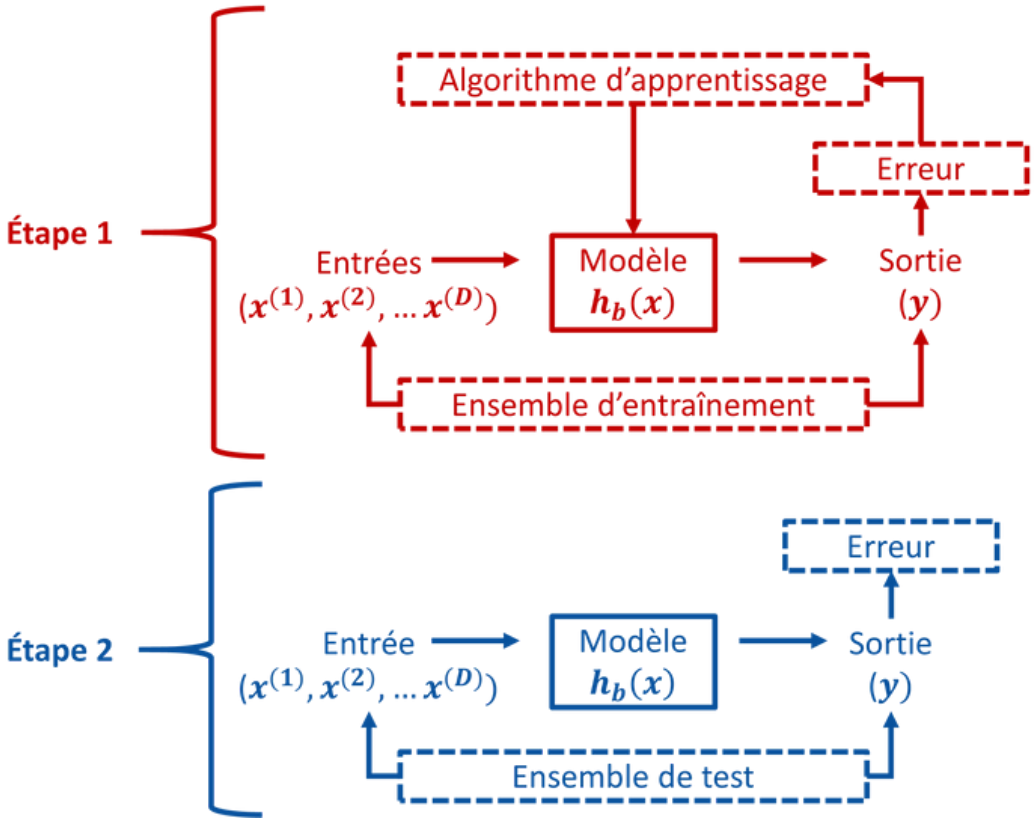
COURS 3

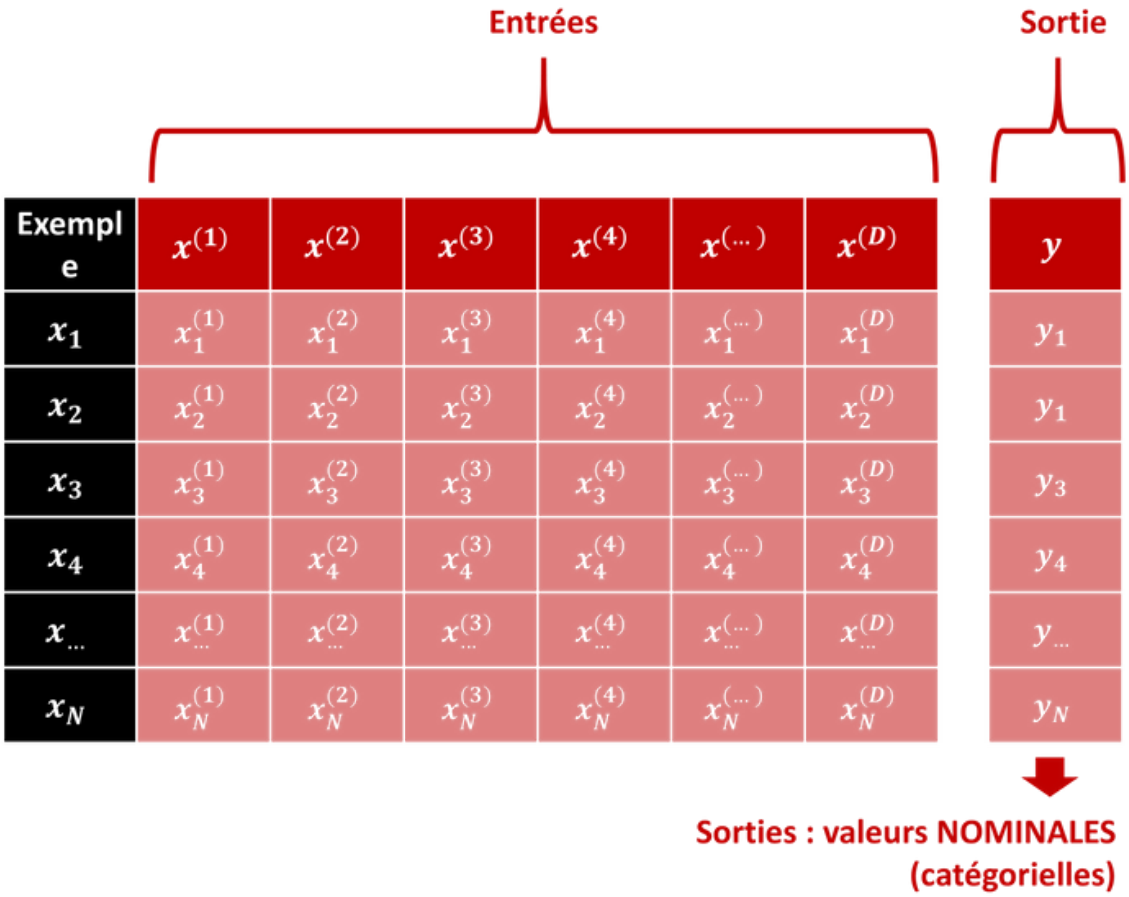
APPRENTISSAGE SUPERVISÉ : CLASSIFICATION

1. ÉVALUATION
2. RÉGRESSION LOGISTIQUE
3. MACHINES À VECTEURS DE SUPPORT

La structure est la même que pour la régression.



Toutefois, la variable « cible » de sortie est nominale.



1. ÉVALUATION

1. ÉVALUATION

1.1. Matrice de confusion

- Au moment d'évaluer le modèle (pour sélectionner les hyperparamètres ou pour évaluer le modèle final), on peut utiliser une autre métrique que la « justesse ».

  - La « précision » (*precision*):

$$\frac{VP}{VP + FP} = \frac{Vrais\ positifs}{Toutes\ les\ valeurs\ \boldsymbol{prédites}\ positives} = \frac{10}{20} = 0.5$$

  - Le « rappel » (*recall*) :

$$\frac{VP}{VP + FN} = \frac{Vrais\ positifs}{Toutes\ les\ valeurs\ \boldsymbol{réelles}\ positives} = \frac{10}{11} = 0.91$$

  - La « justesse » (*accuracy*) :

$$\frac{VP + VN}{VP + VN + FP + FN} = \frac{Toutes\ les\ cas\ prédits\ correctement}{Tous\ les\ cas\ à\ prédire} = \frac{210}{221} = 0.95$$

  - Score F_1 :

$$\frac{2(Précision \times Rappel)}{Précision + Rappel} = \frac{2(0.5 \times 0.91)}{0.5 + 0.91} = 0.65$$

		VALEURS RÉELLES	
		Présence de maladie	**Absence** de maladie
VALEURS PRÉDITES	**Présence** de maladie	**10 (VP)**	10 (FP)
	Absence de maladie	1 (FN)	**200 (VN)**

1. ÉVALUATION

1.1. Matrice de confusion

Et il existe encore quelques autres métriques…

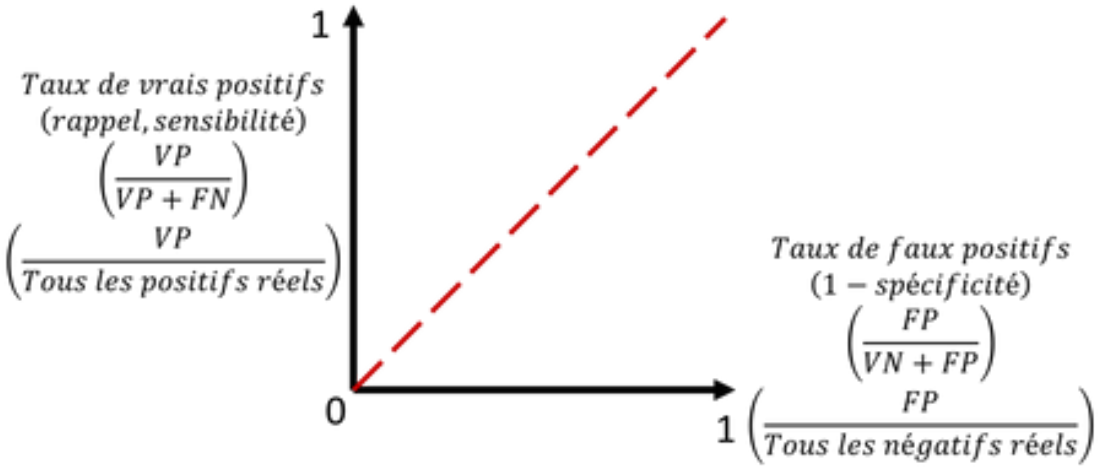
		True condition			
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$		

https://en.wikipedia.org/wiki/Receiver_operating_characteristic, 2019-11-03

1. ÉVALUATION

1.2. Courbe ROC

Au niveau graphique, il existe une représentation très utilisée, nommée ROC (*Receiver Operating Characteristic*)



L'idée générale est la suivante:

- On fait varier le seuil de décision de classification et on observe les performances du modèle.
- La sensibilité et la spécificité sont inversement proportionnelles.
  - Si la sensibilité augmente, la spécificité diminue (et vice-versa).

Quand on **diminue** le seuil de classification, on répond plus souvent positif et ainsi:

- On augmente la sensibilité.
- On diminue la spécificité.

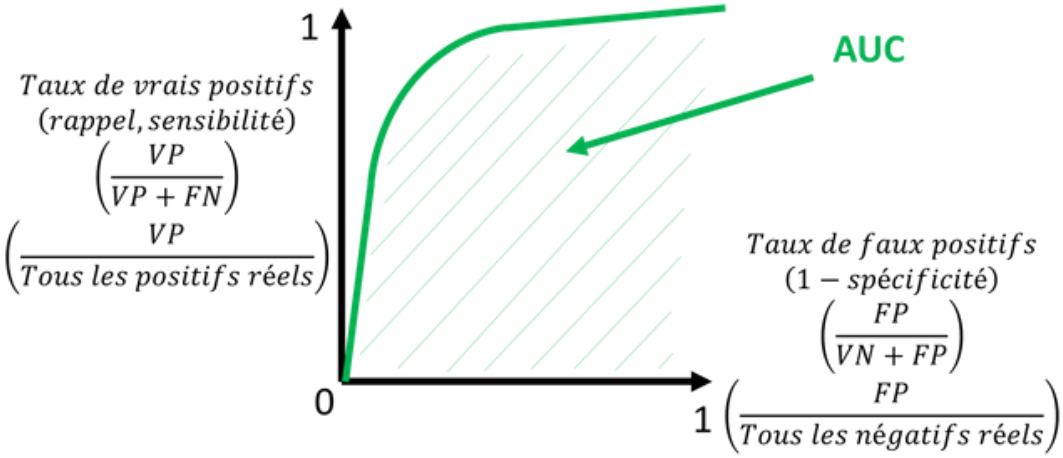
Quand on **augmente** le seuil de classification, on répond moins souvent positif et ainsi:

- On diminue la sensibilité.
- On augmente la spécificité.

1. ÉVALUATION

1.2. Courbe ROC

La métrique généralement utilisée est **l'aire sous la courbe** (AUC : *Area Under the Curve*).



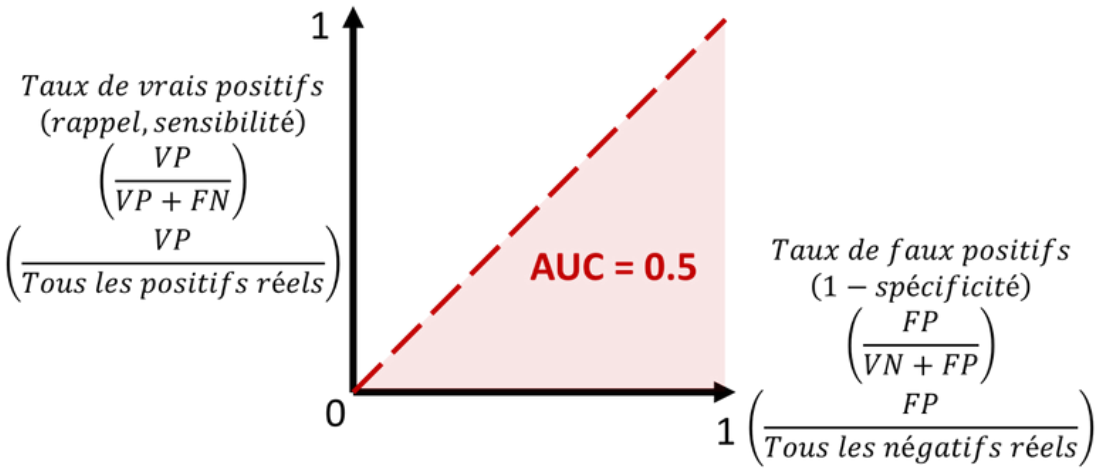
L'aire sous la courbe mesure la capacité du modèle à:

- Conserver une bonne sensibilité lorsque le seuil de décision augmente.
- Conserver une bonne spécificité lorsque le seuil de décision diminue.

1. ÉVALUATION

1.2. Courbe ROC

Cas où le modèle répond au hasard.

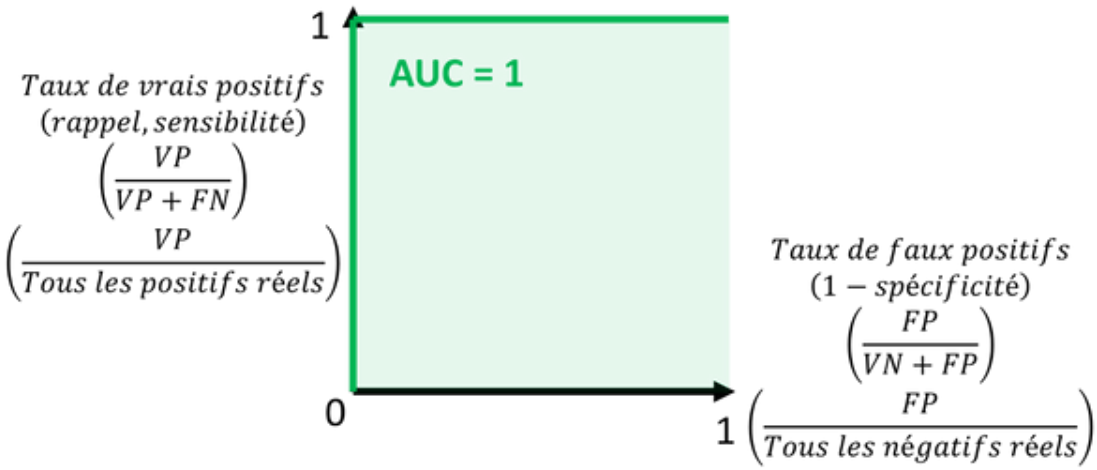


- On n'arrive pas à distinguer les cas réels positifs des négatifs.
  - Plus la sensibilité augmente, plus la spécificité diminue.

1. ÉVALUATION

1.2. Courbe ROC

Meilleure ROC imaginable :

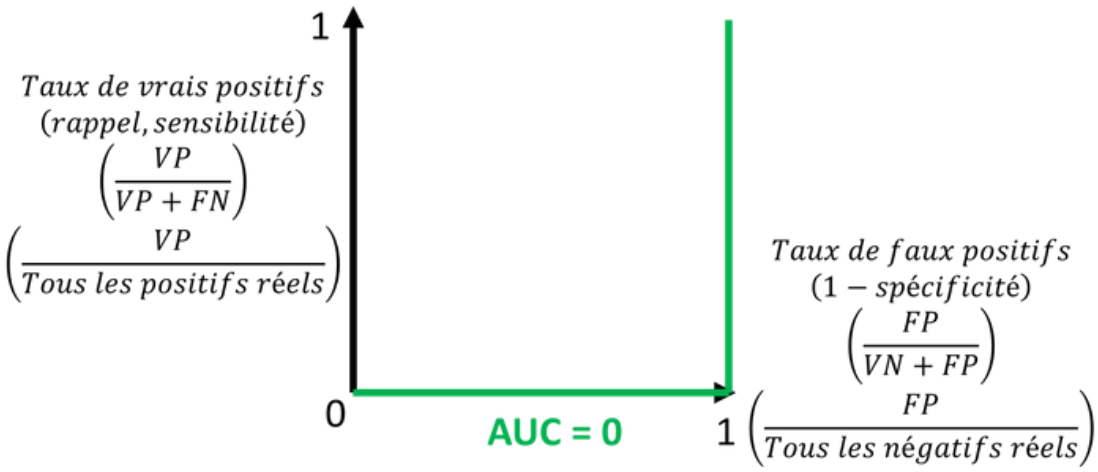


- On ne ratte jamais les cas réellement positifs.
  - On est très sensibles aux vrais cas réels.
  - Plus la sensibilité est grande, plus la valeur est près de 1 sur l'axe des ordonnées.
- On ne prend jamais un cas négatif pour un cas positif.
  - On est très spécifique dans nos choix.
  - Plus la spécificité est grande, plus la valeur est près de 0 sur l'axe des abscisses.

1. ÉVALUATION

1.2. Courbe ROC

Cas où l'on inverse les cas réels positifs et négatifs.



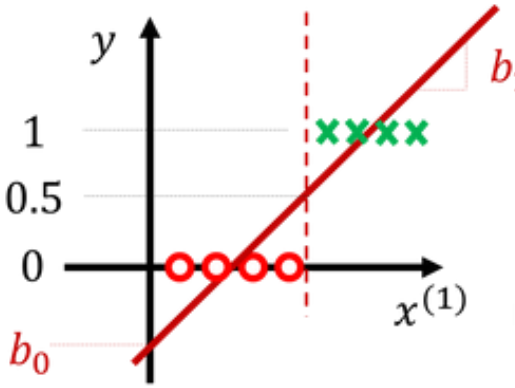
- On ne détecte aucun cas réel positif
  - La sensibilité demeure à 0.
- On prend tous les cas négatifs pour des cas positifs.
  - La spécificité demeure à 0.

2. RÉGRESSION LOGISTIQUE

2. RÉGRESSION LOGISTIQUE

Problème de la régression linéaire en classification

$$h_b(x) = b_0 + b_1 x^{(1)}$$



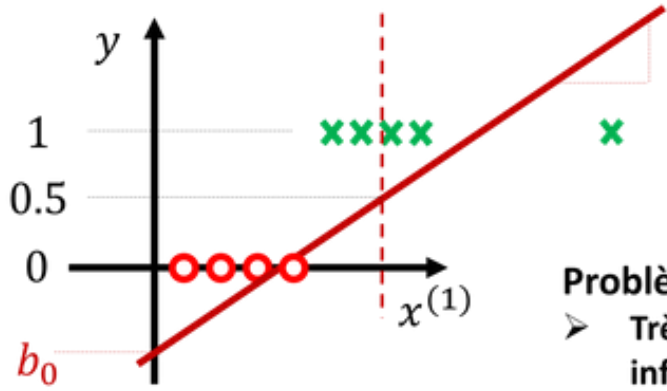
Si $h_b(x) \geq 0.5 \Rightarrow y = 1$

Si $h_b(x) < 0.5 \Rightarrow y = 0$

Problème 1 :

- **Peut avoir des valeurs > 1**
- **Peut avoir des valeurs < 0**

$$h_b(x) = b_0 + b_1 x^{(1)}$$



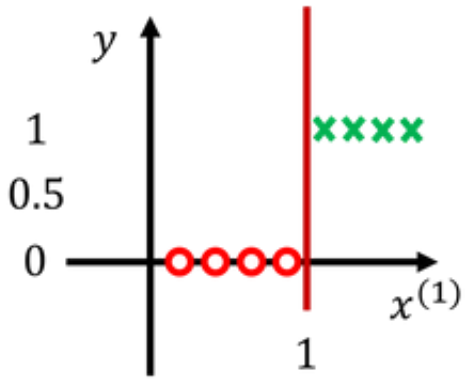
Problème 2 :

- **Très sensible aux données influentes.**

2. RÉGRESSION LOGISTIQUE

Solution : régression logistique

- Partie 1 : la droite devient une frontière entre les catégories.

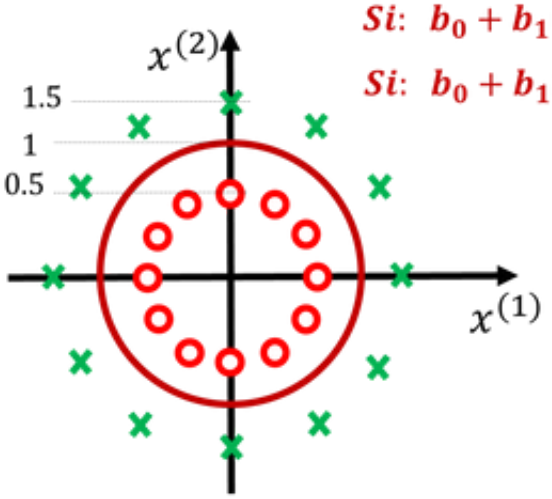


$$Si:\ b_0 + b_1 x^{(1)} \geq 0 \Rightarrow y = 1$$
$$Si:\ b_0 + b_1 x^{(1)} < 0 \Rightarrow y = 0$$

$$Avec:\ b_0 = -1\ et\ b_1 = 1$$

$$-1 + x^{(1)} \geq 0$$
$$x^{(1)} \geq 1 \Rightarrow y = 1$$

$$-1 + x^{(1)} < 0$$
$$x^{(1)} < 1 \Rightarrow y = 0$$



$$Si:\ b_0 + b_1\left(x^{(1)}\right)^2 + b_2\left(x^{(2)}\right)^2 \geq 0 \Rightarrow y = 1$$
$$Si:\ b_0 + b_1\left(x^{(1)}\right)^2 + b_2\left(x^{(2)}\right)^2 < 0 \Rightarrow y = 0$$

$$Avec:\ b_0 = -1, b_1 = 1\ et\ b_2 = 1$$

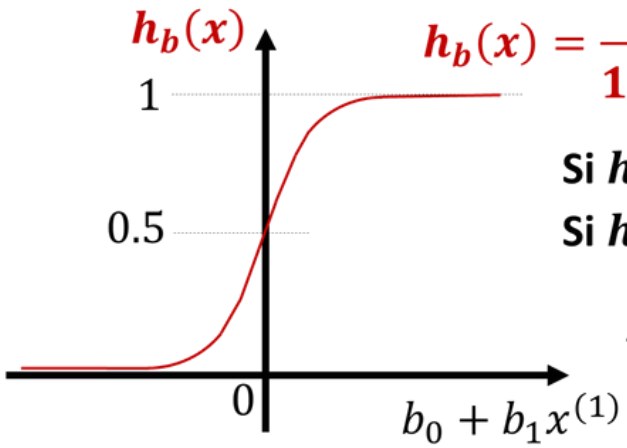
$$-1 + \left(x^{(1)}\right)^2 + \left(x^{(2)}\right)^2 \geq 0$$
$$\left(x^{(1)}\right)^2 + \left(x^{(2)}\right)^2 \geq 1 \Rightarrow y = 1$$

$$-1 + \left(x^{(1)}\right)^2 + \left(x^{(2)}\right)^2 < 0$$
$$\left(x^{(1)}\right)^2 + \left(x^{(2)}\right)^2 < 1 \Rightarrow y = 0$$

2. RÉGRESSION LOGISTIQUE

Solution : régression logistique

- Partie 2 : le modèle donne en sortie non pas la valeur de la classe, mais plutôt la probabilité d'appartenir à la classe 1.



$$h_b(x) = \frac{1}{1 + e^{-(b_0 + b_1 x^{(1)})}}$$

Si $h_b(x) \geq 0.5 \Rightarrow y = 1$

Si $h_b(x) < 0.5 \Rightarrow y = 0$

Avantage:

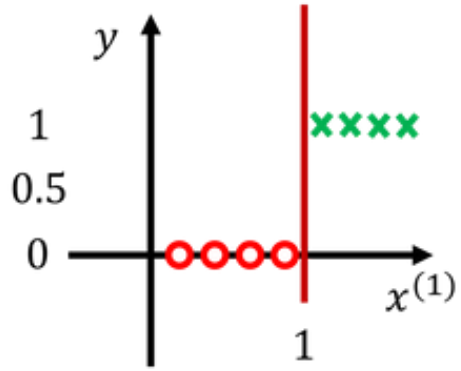
- **La fonction est limitée à des valeurs entre 0 et 1.**

On appelle cette fonction : « sigmoïde » (ou logistique)

2. RÉGRESSION LOGISTIQUE

Solution : régression logistique

- Partie 2 : le modèle donne en sortie non pas la valeur de la classe, mais plutôt la probabilité d'appartenir à la classe 1.

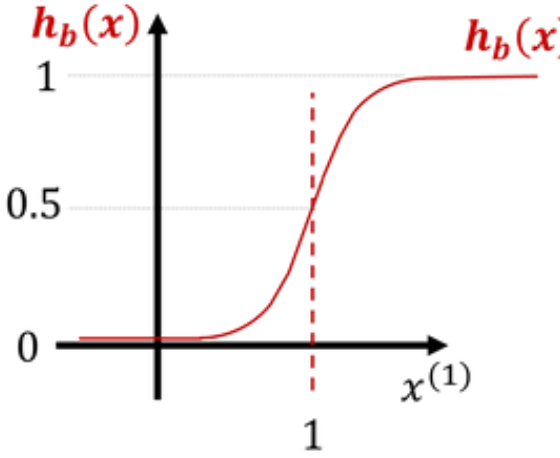


$$Si:\ b_0 + b_1 x^{(1)} \geq 0 \Rightarrow y = 1$$
$$Si:\ b_0 + b_1 x^{(1)} < 0 \Rightarrow y = 0$$

$$Avec:\ b_0 = -1\ et\ b_1 = 1$$

$$-1 + x^{(1)} \geq 0$$
$$x^{(1)} \geq 1 \Rightarrow y = 1$$

$$-1 + x^{(1)} < 0$$
$$x^{(1)} < 1 \Rightarrow y = 0$$



$$h_b(x) = \frac{1}{1 + e^{-(b_0 + b_1 x^{(1)})}}$$

Si $h_b(x) \geq 0.5 \Rightarrow y = 1$

Si $h_b(x) < 0.5 \Rightarrow y = 0$

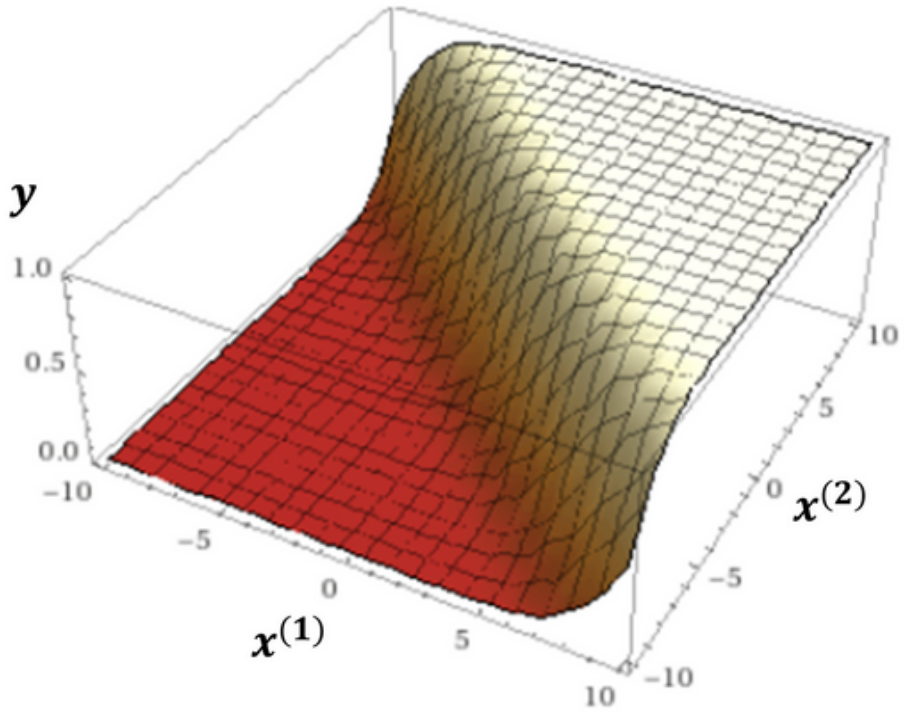
Avantage:

- **La fonction est limitée à des valeurs entre 0 et 1.**

2. RÉGRESSION LOGISTIQUE

Solution : régression logistique (avec 2 caractéristiques)

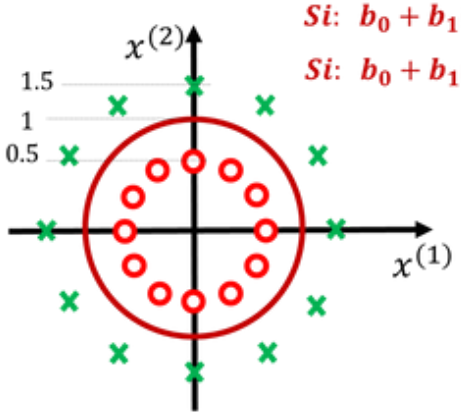
Exemple avec 2 caractéristiques



2. RÉGRESSION LOGISTIQUE

Solution : régression logistique

- Partie 2 : le modèle donne en sortie non pas la valeur de la classe, mais plutôt la probabilité d'appartenir à la classe 1.



$$Si:\ b_0 + b_1\left(x^{(1)}\right)^2 + b_2\left(x^{(2)}\right)^2 \geq 0 \Rightarrow y = 1$$

$$Si:\ b_0 + b_1\left(x^{(1)}\right)^2 + b_2\left(x^{(2)}\right)^2 < 0 \Rightarrow y = 0$$

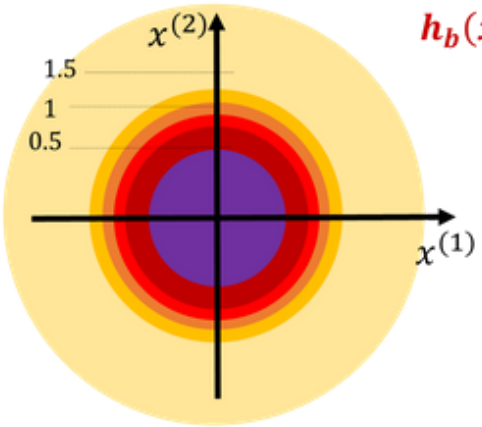
$$Avec:\ b_0 = -1\,, b_1 = 1\ et\ \ b_2 = 1$$

$$-1 + \left(x^{(1)}\right)^2 + \left(x^{(2)}\right)^2 \geq 0$$

$$\left(x^{(1)}\right)^2 + \left(x^{(2)}\right)^2 \geq 1 \Rightarrow y = 1$$

$$-1 + \left(x^{(1)}\right)^2 + \left(x^{(2)}\right)^2 < 0$$

$$\left(x^{(1)}\right)^2 + \left(x^{(2)}\right)^2 < 1 \Rightarrow y = 0$$



$$h_b(x) = \frac{1}{1 + e^{-(b_0 + b_1 x^{(1)})}}$$

Si $h_b(x) \geq 0.5 \Rightarrow y = 1$

Si $h_b(x) < 0.5 \Rightarrow y = 0$

Avantage:

- **La fonction est limitée à des valeurs entre 0 et 1.**

$h_b(x) \approx 0$  $h_b(x) \approx 1$

2. RÉGRESSION LOGISTIQUE

Comme dans le cas de la régression linéaire, la fonction de perte inclut deux termes:

- Une fonction de coût
- Une fonction de régularisation

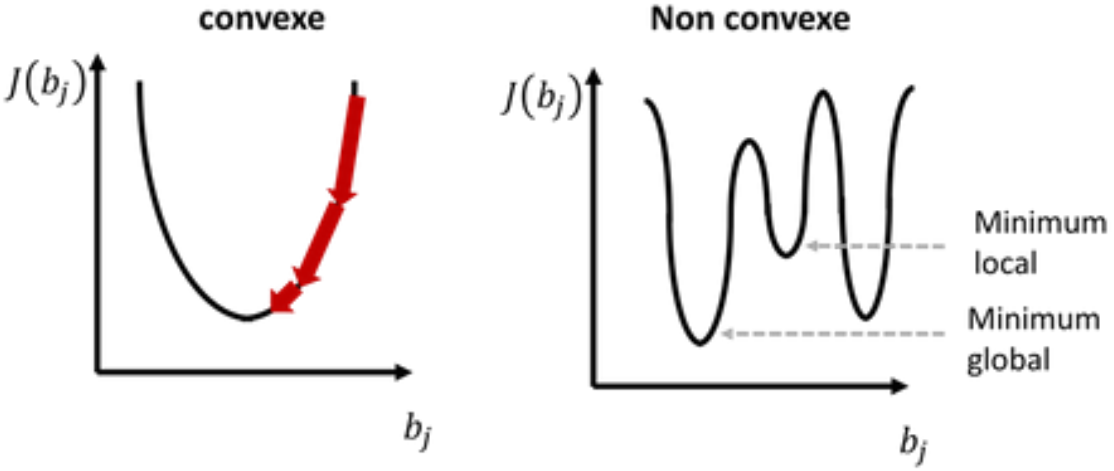
Toutefois, pour ce qui est de la fonction de coût, on ne peut pas prendre directement

$$\sum_{i=1}^{N}\left(y_i - h_{\boldsymbol{b}}(\boldsymbol{x})\right)^2$$

Le problème est que pour une fonction sigmoïde telle que

$$h_{\boldsymbol{b}}(\boldsymbol{x}) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{e}^{-\left(b_0 + b_1 x^{(1)}\right)}}$$

- Il n'existe pas de solution analytique (on doit donc progresser de manière itérative).
  - Or, l'espace de la fonction de coût n'est pas « convexe ».
  - On peut alors être coincé dans un « minimum local » qui est supérieur au « minimum global ».



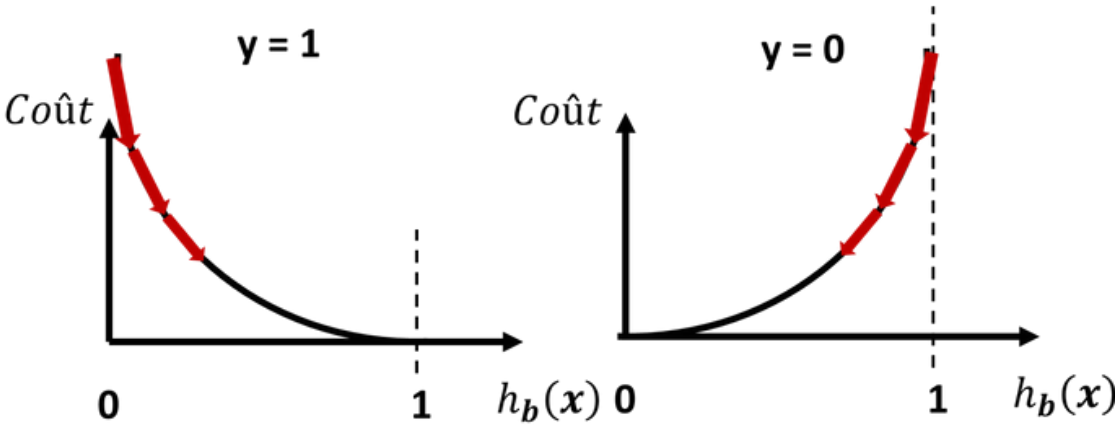
2. RÉGRESSION LOGISTIQUE

Pour obtenir une fonction convexe, on utilise la fonction de coût suivante:

$$Coût = \begin{cases} -log\left(h_{\boldsymbol{b}}(\boldsymbol{x})\right) & si \quad y = 1 \\ -log\left(1 - h_{\boldsymbol{b}}(\boldsymbol{x})\right) & si \quad y = 0 \end{cases}$$

Qu'on peut réécrire ainsi:

$$Coût = -\left[y \cdot log\left(h_{\boldsymbol{b}}(\boldsymbol{x})\right)\right] - \left[(1 - y) \cdot log\left(1 - h_{\boldsymbol{b}}(\boldsymbol{x})\right)\right]$$



2. RÉGRESSION LOGISTIQUE

La fonction de perte correspond alors à :

Qu'on peut réécrire ainsi:

Note: $\alpha = \frac{1}{C}$

L1 :

$$J(\boldsymbol{b}) = \frac{1}{n}\sum_{i=1}^{n}\left(-\left[y \cdot log\left(h_{\boldsymbol{b}}(\boldsymbol{x})\right)\right] - \left[(1-y) \cdot log\left(1-h_{\boldsymbol{b}}(\boldsymbol{x})\right)\right]\right) + \frac{1}{Cp}\sum_{j=1}^{p}|b_j|$$

L2 :

$$J(\boldsymbol{b}) = \frac{1}{n}\sum_{i=1}^{n}\left(-\left[y \cdot log\left(h_{\boldsymbol{b}}(\boldsymbol{x})\right)\right] - \left[(1-y) \cdot log\left(1-h_{\boldsymbol{b}}(\boldsymbol{x})\right)\right]\right) + \frac{1}{2Cp}\sum_{j=1}^{p} b_j^2$$

L1 + L2 (ElasticNet) :

$$J(\boldsymbol{b}) = \frac{1}{n}\sum_{i=1}^{n}\left(-\left[y \cdot log\left(h_{\boldsymbol{b}}(\boldsymbol{x})\right)\right] - \left[(1-y) \cdot log\left(1-h_{\boldsymbol{b}}(\boldsymbol{x})\right)\right]\right) + \alpha\sum_{j=1}^{p}\left(b_j\right)^2 + (1-\alpha)\sum_{j=1}^{p}|b_j|$$

ElasticNet version scikit-learn :

$$J(\boldsymbol{b}) = \frac{1}{n}\sum_{i=1}^{n}\left(-\left[y \cdot log\left(h_{\boldsymbol{b}}(\boldsymbol{x})\right)\right] - \left[(1-y) \cdot log\left(1-h_{\boldsymbol{b}}(\boldsymbol{x})\right)\right]\right) + \frac{\alpha}{2}(1-l1\_ratio)\sum_{j=1}^{p}\left(b_j\right)^2 + \alpha \cdot l1\_ratio\sum_{j=1}^{p}|b_j|$$

2. RÉGRESSION LOGISTIQUE

Quelques notes :

- La validation croisée est applicable comme dans le cas de la régression linéaire.
  - Néanmoins, on doit utiliser une validation croisée « stratifiée » pour s'assurer que chaque « pli » contienne approximativement le même ratio $\frac{\sum(y=1)}{\sum(y=0)}$ que dans tout l'ensemble d'entraînement.

- Si une valeur de *y* est beaucoup plus fréquente qu'une autre (ex. $n_0 = 200$, $n_1 = 10$), il est possible d'utiliser l'une ou l'autre des techniques suivantes:
  - Suréchantillonner le plus petit groupe.
  - Sous-échantillonner le plus grand groupe.
  - Créer des données synthétiques supplémentaires pour le plus petit groupe.
  - ❖ Dans tous les cas, n'inclure **que** les données utilisées pour l'entraînement!

FAISONS UN EXEMPLE

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

import pandas as pd
import numpy as np

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

# Importer les fonctions de prétraitement
from sklearn.preprocessing import StandardScaler

# Importer une fonction qui nous permette de construire aléatoireme
nt les ensembles "Entraînement" et "Test"
from sklearn.model_selection import train_test_split

# Importer le modèle de régression logistique de sklearn
from sklearn.linear_model import LogisticRegression

# Importer la fonction de validation croisée
from sklearn.model_selection import StratifiedKFold, GridSearchCV

# Importer la fonction permettant d'afficher le rapport de classifi
cation
from sklearn.metrics import roc_auc_score

np.random.seed(42)

# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------

# Importons un ensemble de données
data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()

# Colonnes correspondant à des caractéristiques
features_cols = ['PSQ_SS', 'PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITT
```

```
T', 'STAIYTT', 'AGE', 'SEXE', 'SES']

# Données importées (X: caractéristiques, y: cible)
X = data.loc[:, features_cols]
y = data['WHODASTTB']

# Séparation en données d'entraînement et en données de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.2)

# Standatdisation des entrées
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

print('Data ready')

# ----------------------------------------------------------------
------------
# ÉTAPE 4 : définir et entraîner le modèle
# ----------------------------------------------------------------
------------

# Définir le modèle
model = LogisticRegression()

# Définir les hyperparamètres
hyperparams = {'C':[.0001, .001, .01, .1, 1, 10, 100, 1000, 10000]}

# Définir les plus de la validation croisée
cv_folds = StratifiedKFold(n_splits=5, random_state=42)

# Définir le type de score utilisé pour sélectionner les hyperparam
ètres dans la validation croisée
scoring='roc_auc'

# Réaliser la validation croisée avec grille de recherche pour les
 hyperparamètres.
cv_valid = GridSearchCV(estimator=model, param_grid=hyperparams, cv
=cv_folds, scoring=scoring, iid=False)
cv_valid.fit(X_train, y_train)
best_params = cv_valid.best_params_
best_score = cv_valid.best_score_
model = cv_valid.best_estimator_
print('\nMeilleurs hyperparamètres: \n', best_params)
print('\nScore = \n', best_score)

# Entraîner le modèle final avec toutes les données d'entraînement
model.fit(X_train,y_train)
```

```
Data ready

Meilleurs hyperparamètres:
 {'C': 0.1}

Score =
 0.7248212299380264
```

Out[1]:

```
LogisticRegression(C=0.1, class_weight=None, dual=Fals
e, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None,
max_iter=100,
                   multi_class='warn', n_jobs=None, pen
alty='l2',
                   random_state=None, solver='warn', to
l=0.0001, verbose=0,
                   warm_start=False)
```

3. MACHINES À VECTEURS DE SUPPORT

3. MACHINES À VECTEURS DE SUPPORT

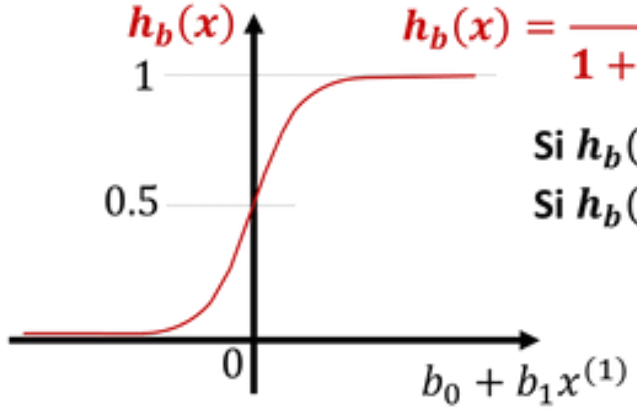
Algorithme de classification :

- alternative à la régression logistique

Pour l'expliquer, partons de la régression logistique.

On avait alors le modèle suivant, qui correspond à une fonction sigmoïde et peut être considéré comme une probabilité d'appartenir à une classe :

$$h_b(x) = \frac{1}{1 + e^{-(b_0 + b_1 x^{(1)})}}$$

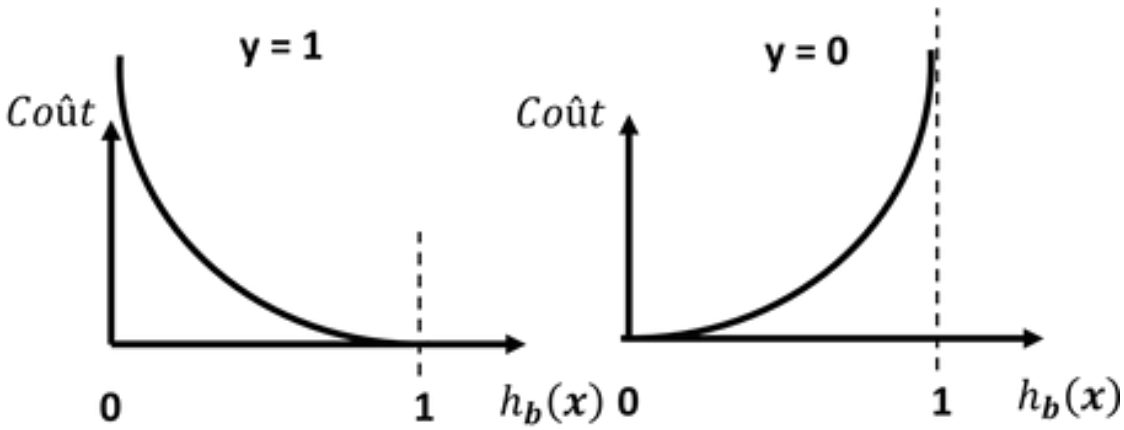


Si $h_b(x) \geq 0.5 \Rightarrow y = 1$

Si $h_b(x) < 0.5 \Rightarrow y = 0$

Nous avions alors la fonction de coût suivante :

$$Coût = -[y \cdot log(h_b(x))] - [(1 - y) \cdot log(1 - h_b(x))]$$



3. MACHINES À VECTEURS DE SUPPORT

Or, si on peut intuitivement interpréter la régression logistique comme une prédiction en termes de probabilités…

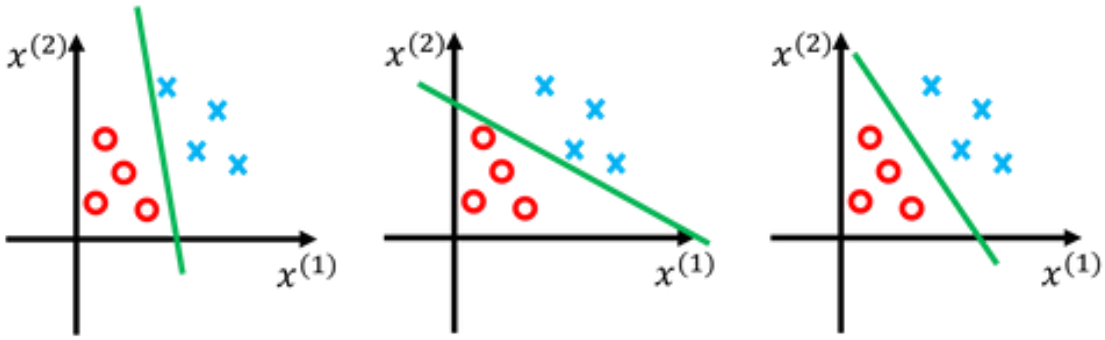
- …les machines à vecteurs de support ont une interprétation plus intuitive en termes géométriques.

On revient avec une forme de modèle linéaire plus classique, soit, une frontière linéaire, avec une équation similaire à :

$$b_0 + b_1 x^{(1)} + b_2 x^{(2)} + \cdots \geq 0$$

Toutefois, on apporte une modification. On ne veut pas simplement trouver un hyperplan qui permette de classifier les données, on veut trouver « **la meilleure** » frontière linéaire possible.

Illustrons:



Par « **la meilleure** » frontière linéaire, on entend…

- …« la frontière qui est la plus éloignée des données les plus proches ».

3. MACHINES À VECTEURS DE SUPPORT

Pour obtenir une telle frontière, on doit modifier légèrement l'équation générale suivante :

$$b_0 + b_1 x^{(1)} + b_2 x^{(2)} + \cdots \geq 0$$

Cette équation dit simplement que les données supérieures à « 0 » sont classées dans une certaine catégorie et que les données inférieures à « 0 » sont classées dans une autre catégorie.

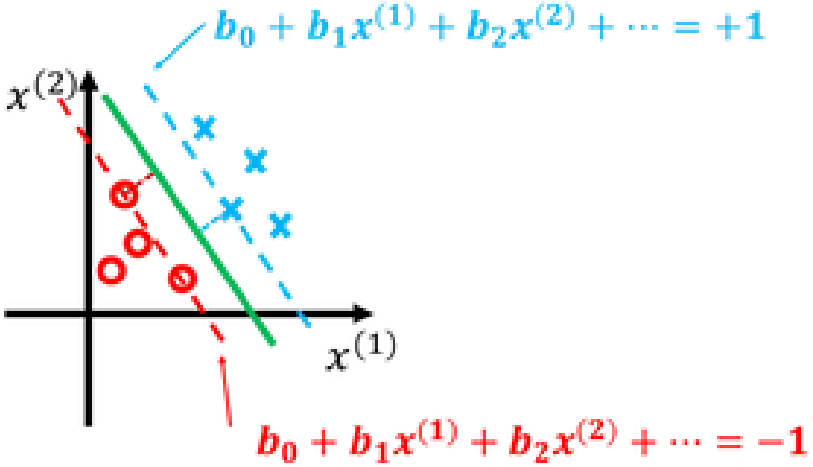
Or, on veut plutôt avoir une équation qui permette d'éloigner le plus possible les données les plus proches.

En d'autres termes, on veut plutôt quelque chose qui reflèterait :

$$Si\ y = +1 : \quad b_0 + b_1 x^{(1)} + b_2 x^{(2)} + \cdots \geq +1$$
$$Si\ y = -1 : \quad b_0 + b_1 x^{(1)} + b_2 x^{(2)} + \cdots < -1$$

Ceci revient à construire non seulement une frontière de séparation, mais également un « coussin de sûreté », appelé ici une « marge », de chaque côté de la frontière de décision.

Illustrons:



3. MACHINES À VECTEURS DE SUPPORT

Les points qui apparaissent directement sur les droites en pointillés sont appelés « **vecteurs de support** ».

La droite qui correspond à la frontière au centre correspond à la médiane des deux droites pointillées;

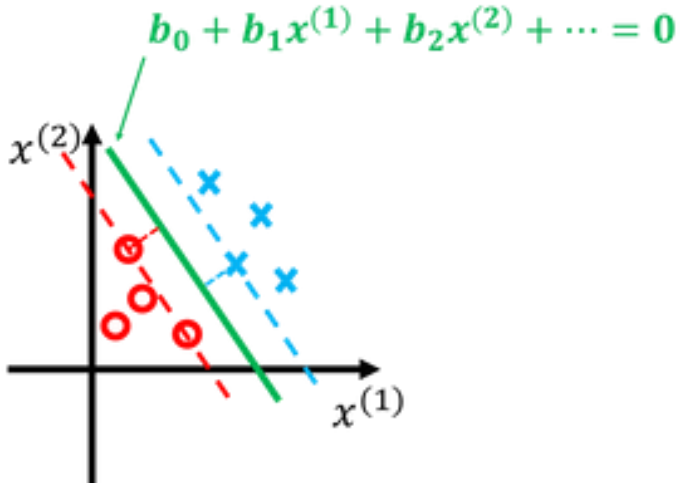
- Cette droite a pour équation:

$$b_0 + b_1 x^{(1)} + b_2 x^{(2)} + \cdots = 0$$

La distance entre la frontière centrale et le point positif le plus proche est appelée « d^+ ».

La distance entre la frontière centrale et le point négatif le plus proche est appelée « d^- ».

La largeur totale correspondant à « $d^+ + d^-$ » est appelée la « **marge** » (notée « d »).



3. MACHINES À VECTEURS DE SUPPORT

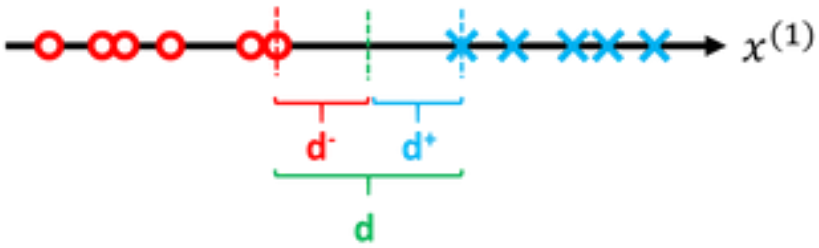
L'objectif est donc ici de trouver la frontière centrale qui **maximise la marge**.

Or, la marge ainsi que la position de la frontière, ne reposent que sur les vecteurs de supports.

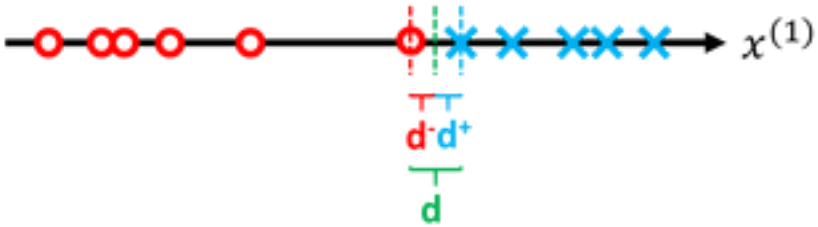
Si on ne tient compte **que de ce critère**, alors l'algorithme s'appelle en fait **classificateur à marge maximal** (*maximal margin classifier*).

- Ce type de classificateur utilise ce que l'on appelle une marge « dure ».

Simplifions l'illustration ainsi :



Qu'advient-il alors dans le cas où on aurait :



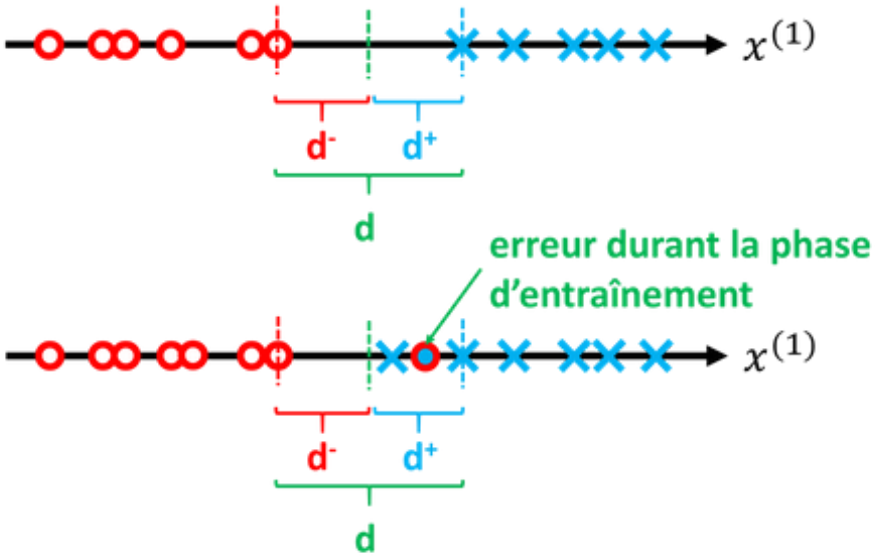
Or, on minimise alors… le biais…
sans considération pour… la variance!

- On ne minimise alors pas l'erreur de généralisation

3. MACHINES À VECTEURS DE SUPPORT

Pour trouver un compromis entre le biais et la variance et ainsi minimiser l'erreur de généralisation, on utilisera plutôt des marges dites « **souples** ».

Illustrons :



Afin de déterminer la souplesse de la marge (i.e. combien d'erreurs de classification on est prêt à accepter durant la phase d'entraînement, on utilise un…. :

- Hyperparamètre de **régularisation**.

3. MACHINES À VECTEURS DE SUPPORT

Fonction de perte

$$J(\boldsymbol{b}) = \boldsymbol{C}\sum_{i=1}^{n}\left([y_i \cdot coût_1(h_{\boldsymbol{b}}(\boldsymbol{x}_i))] - [(1-y_i)coût_0(1-h_{\boldsymbol{b}}(\boldsymbol{x}_i))]\right) + \frac{1}{2}\sum_{j=1}^{p} b_j^2$$

$\boldsymbol{C}$ est l'hyperparamètre qui gère la « **souplesse** » de la marge.

- Plus $\boldsymbol{C}$ est grand, plus la marge sera « **dure** ».
- Plus $\boldsymbol{C}$ est grand, plus un seul exemple peut contribuer de manière importante à la fonction de perte et ainsi devenir un vecteur de support.

- Si $\boldsymbol{C}$ est petit, on pourra diminuer l'importance de chaque erreur individuelle et ainsi permettre une marge plus « **souple** ».

Jouons avec l'hyperparamètre $\boldsymbol{C}$ dans le cadre d'un exemple concret.

In [2]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

import pandas as pd
import numpy as np

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

# Importer les fonctions de prétraitement
from sklearn.preprocessing import StandardScaler

# Importer une fonction qui nous permette de construire aléatoireme
nt les ensembles "Entraînement" et "Test"
from sklearn.model_selection import train_test_split

# Importer le modèle de régression logistique de sklearn
from sklearn import svm

# Importer la fonction de validation croisée
from sklearn.model_selection import StratifiedKFold, GridSearchCV

# Importer la fonction permettant d'afficher le rapport de classifi
cation
from sklearn.metrics import roc_auc_score

np.random.seed(42)

# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------

# Importons un ensemble de données
data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()

# Colonnes correspondant à des caractéristiques
features_cols = ['PSQ_SS', 'PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITT
```

```
T', 'STAIYTT', 'AGE', 'SEXE', 'SES']

# Données importées (X: caractéristiques, y: cible)
X = data.loc[:, features_cols]
y = data['WHODASTTB']

# Séparation en données d'entraînement et en données de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.2)

# Standatdisation des entrées
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

print('Data ready')

# ----------------------------------------------------------------
------------
# ÉTAPE 4 : définir et entraîner le modèle
# ----------------------------------------------------------------
------------

# Définir le modèle
model = svm.SVC(kernel='linear')

# Définir les hyperparamètres
hyperparams = {'C':[.0001, .001, .01, .1, 1, 10, 100, 1000]}

# Définir les plus de la validation croisée
cv_folds = StratifiedKFold(n_splits=5, random_state=42)

# Définir le type de score utilisé pour sélectionner les hyperparam
ètres dans la validation croisée
scoring='roc_auc'

# Réaliser la validation croisée avec grille de recherche pour les
 hyperparamètres.
cv_valid = GridSearchCV(estimator=model, param_grid=hyperparams, cv
=cv_folds, scoring=scoring, iid=False)
cv_valid.fit(X_train, y_train)
best_params = cv_valid.best_params_
best_score = cv_valid.best_score_
model = cv_valid.best_estimator_
print('\nMeilleurs hyperparamètres: \n', best_params)
print('\nScore = \n', best_score)

# Entraîner le modèle final avec toutes les données d'entraînement
model.fit(X_train,y_train)
```

```
Data ready

Meilleurs hyperparamètres:
 {'C': 0.0001}

Score =
 0.721579532814238
```

Out[2]:

```
SVC(C=0.0001, cache_size=200, class_weight=None, coef0=
0.0,
    decision_function_shape='ovr', degree=3, gamma='aut
o_deprecated',
    kernel='linear', max_iter=-1, probability=False, ra
ndom_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [3]:

```
# On teste l'algorithme final en prédisant de nouvelles données.
y_pred = model.predict(X_test)

# On évalue les prédictions de l'algorithme final.
auc = roc_auc_score(y_test, y_pred)

# On affiche le résultat
print('\nTest AUC = ', auc)
```

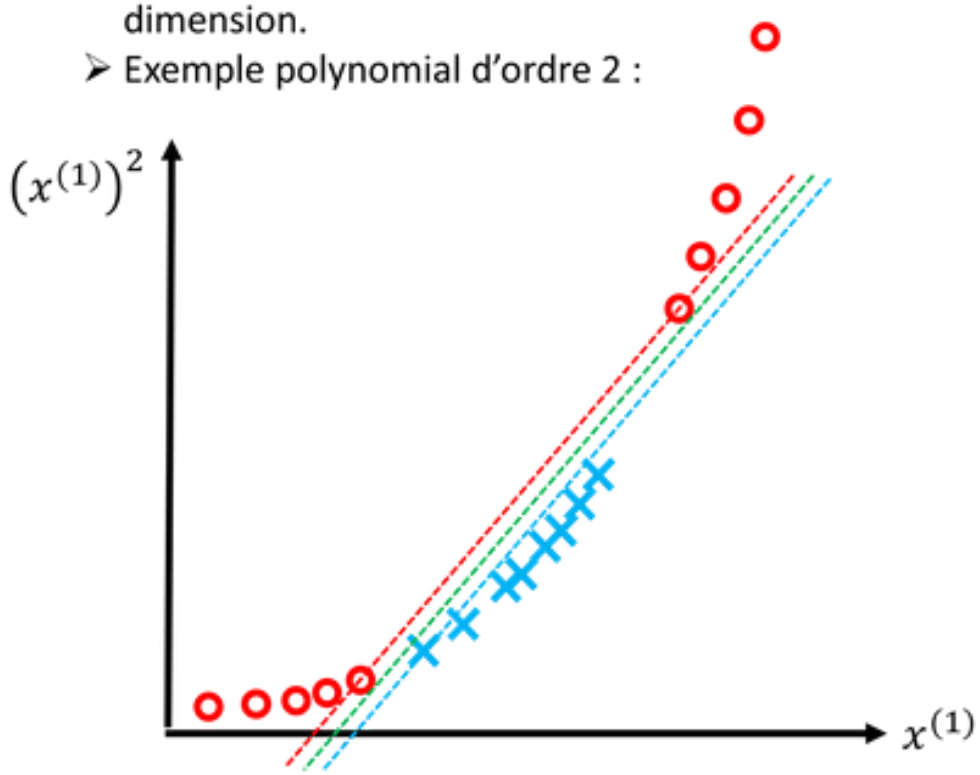
```
Test AUC =  0.5
```

3. MACHINES À VECTEURS DE SUPPORT

Maintenant, que peut-on faire dans le cas suivant ?

On peut utiliser ce que l'on appelle une fonction « noyau » (*kernel function*).

- On projette nos données dans un espace en plus haute dimension.
- Exemple polynomial d'ordre 2 :



La méthode est appelée « **astuce du noyau** », car en fait, on **n'a pas** à calculer la transformation pour calculer les vecteurs de support et donc la frontière centrale.

3. MACHINES À VECTEURS DE SUPPORT

Les fonctions noyau les plus utilisées sont :

- Le noyau polynomial
- Le noyau Gaussien (aussi appelé « noyau à base radiale » ; *radial basis function*)

- La version « sans transformation » correspond au « noyau linéaire ».

Essayons l'astuce du noyau avec un exemple concret.

In [4]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

import pandas as pd
import numpy as np

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

# Importer les fonctions de prétraitement
from sklearn.preprocessing import StandardScaler

# Importer une fonction qui nous permette de construire aléatoireme
nt les ensembles "Entraînement" et "Test"
from sklearn.model_selection import train_test_split

# Importer le modèle de régression logistique de sklearn
from sklearn import svm

# Importer la fonction de validation croisée
from sklearn.model_selection import StratifiedKFold, GridSearchCV

# Importer la fonction permettant d'afficher le rapport de classifi
cation
from sklearn.metrics import roc_auc_score

np.random.seed(42)

# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------

# Importons un ensemble de données
data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()

# Colonnes correspondant à des caractéristiques
features_cols = ['PSQ_SS', 'PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITT
```

```
T', 'STAIYTT', 'AGE', 'SEXE', 'SES']

# Données importées (X: caractéristiques, y: cible)
X = data.loc[:, features_cols]
y = data['WHODASTTB']

# Séparation en données d'entraînement et en données de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.2)

# Standatdisation des entrées
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

print('Data ready')

# ----------------------------------------------------------------
------------
# ÉTAPE 4 : définir et entraîner le modèle
# ----------------------------------------------------------------
------------

# Définir le modèle
model = svm.SVC(kernel='rbf')

# Définir les hyperparamètres
hyperparams = {'C':[.0001, .001, .01, .1, 1, 10, 100, 1000], 'gamm
a':[.0001, .001, .01, .1, 1, 10, 100, 1000]}

# Définir les plus de la validation croisée
cv_folds = StratifiedKFold(n_splits=5, random_state=42)

# Définir le type de score utilisé pour sélectionner les hyperparam
ètres dans la validation croisée
scoring='roc_auc'

# Réaliser la validation croisée avec grille de recherche pour les
 hyperparamètres.
cv_valid = GridSearchCV(estimator=model, param_grid=hyperparams, cv
=cv_folds, scoring=scoring, iid=False)
cv_valid.fit(X_train, y_train)
best_params = cv_valid.best_params_
best_score = cv_valid.best_score_
model = cv_valid.best_estimator_
print('\nMeilleurs hyperparamètres: \n', best_params)
print('\nScore = \n', best_score)

# Entraîner le modèle final avec toutes les données d'entraînement
model.fit(X_train,y_train)
```

```
Data ready

Meilleurs hyperparamètres:
 {'C': 10, 'gamma': 0.01}

Score =
 0.7313840775464803
```

Out[4]:

```
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.0
1, kernel='rbf',
    max_iter=-1, probability=False, random_state=None,
shrinking=True,
    tol=0.001, verbose=False)
```

In [5]:

```
# On teste l'algorithme final en prédisant de nouvelles données.
y_pred = model.predict(X_test)

# On évalue les prédictions de l'algorithme final.
auc = roc_auc_score(y_test, y_pred)

# On affiche le résultat
print('\nTest AUC = ', auc)
```

```
Test AUC =  0.6406025824964132
```