Calcul de risque à l'aide la régression logistique binaire

Comment la régression logistique peut aider à déterminer la probabilité d'avoir un enfant prématuré ?

LAFIN Emérance
39764

Sources : https://www.iconfinder.com/ et https://www.magicmaman.com/

PLAN

I. Modèle de la régression logistique

1) Fonction LOGIT
2) Maximum de vraisemblance
3) Algorithme de Newton-Raphson

II. Application

1) Enfant prématuré
2) Interprétation des résultats

Source : https://www.iconfinder.com/

I. Modèle de la régression logistique

1) Fonction LOGIT

Variables / Individus	X_1	X_2	X_3	…	X_n	Y
ω_1						0
ω_2						1
ω_3						1
…						
ω_i						0

$\underline{Probabilité\ \pi\ d'un\ individu\ \omega :}$

$$Logit(\pi(\omega)) = Log\left(\frac{\pi(\omega)}{1-\pi(\omega)}\right)$$

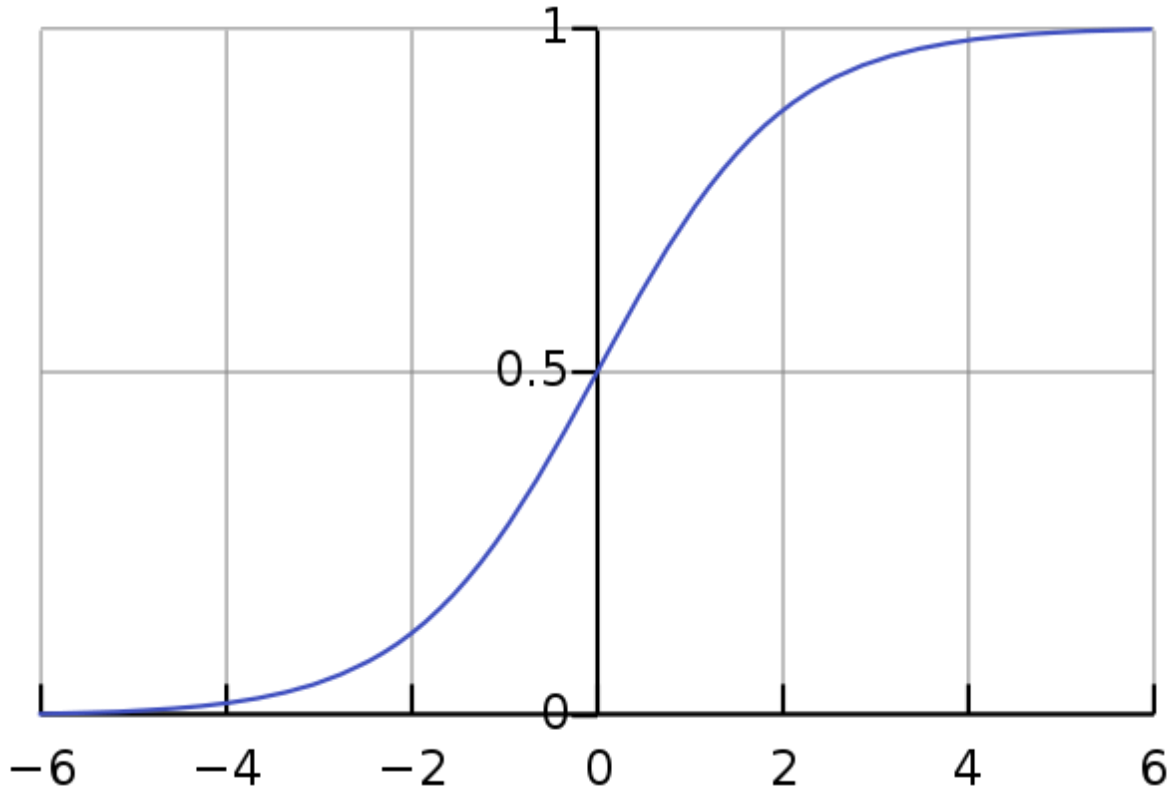
$$Logit(\pi(\omega)) = \beta_0 + \sum_{k=1}^{n} \beta_k * X_k(\omega) = C(\omega)$$

$$\begin{pmatrix} C(\omega_1) \\ \vdots \\ C(\omega_i) \end{pmatrix} = \begin{pmatrix} 1 & X_1(\omega_1) & \dots & X_n(\omega_1) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & X_1(\omega_i) & \dots & X_n(\omega_i) \end{pmatrix} \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_n \end{pmatrix}$$

$$= X(\Omega) * \beta$$

I. Modèle de la régression logistique

1) Fonction LOGIT



$$\pi(\omega) = \frac{1}{1 + e^{-C(\omega)}}$$

$$0 \leq \pi(x) \leq 1$$

$$Règle\ de\ décision : \pi(0) = 0.5$$

- $Y = 0$ si $\pi(\omega) < 0.5$ $(C(\omega) < 0)$
- $Y = 1$ si $\pi(\omega) \geq 0.5$ $(C(\omega) \geq 0)$

Source : https://fr.abcdef.wiki/wiki/Logistic_function

I. Modèle de la régression logistique

2) Maximum de vraisemblance

Soit $y_p(\omega_{\mathrm{k}})$ la variable cible prédite à l'aide de la fonction LOGIT

$$P\left((y(\omega_1),\dots,y(\omega_{\mathrm{i}})) = \left(y_p(\omega_1),\dots,y_p(\omega_{\mathrm{i}})\right)\right) = P\left(\bigcap_{k=1}^{i}(y(\omega_{\mathrm{k}}) = y_p(\omega_{\mathrm{k}}))\right)$$

$$= \prod_{k=1}^{i} P(y(\omega_{\mathrm{k}}) = y_p(\omega_{\mathrm{k}}))$$

Loi de Bernoulli : $P(Y = y(\omega)) = \pi(\omega)^{y(\omega)} * (1 - \pi(\omega))^{(1-y(\omega))}$

Vraisemblance :

$$L = \prod_{\omega \in \Omega} \pi(\omega)^{y_p(\omega)} * (1 - \pi(\omega))^{(1-y_p(\omega))}$$

I. Modèle de la régression logistique

2) Maximum de vraisemblance

Vraisemblance :

$$L = \prod_{\omega \in \Omega} \pi(\omega)^{y_p(\omega)} * (1 - \pi(\omega))^{(1 - y_p(\omega))}$$

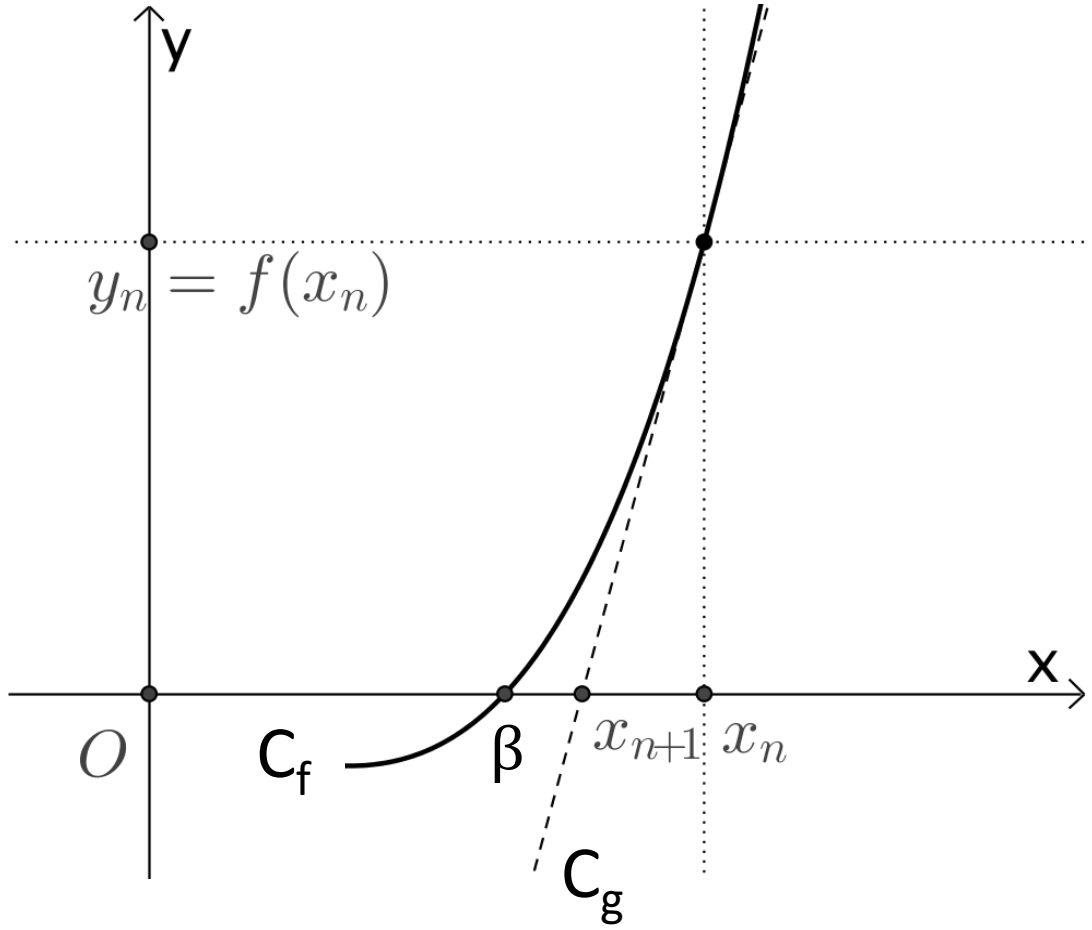
Log-vraisemblance :

$$LL = \sum_{\omega \in \Omega} y_p(\omega) * \ln\big(\pi(\omega)\big) + \Big(1 - y_p(\omega)\Big) * \ln(1 - \pi(\omega))$$

$$\frac{\partial LL(\beta)}{\partial \beta} = 0$$

I. Modèle de la régression logistique

3) Algorithme de Newton-Raphson



- <u>Dimension 1 :</u>

Equation de la tangente en x_n:

$$g(x) = f'(x_n)(x - x_n) + f(x_n)$$

$$g(x_{n+1}) = 0 \Rightarrow x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Source : https://www.mathprepa.fr/

I. Modèle de la régression logistique

3) Algorithme de Newton-Raphson

- Dimension i :

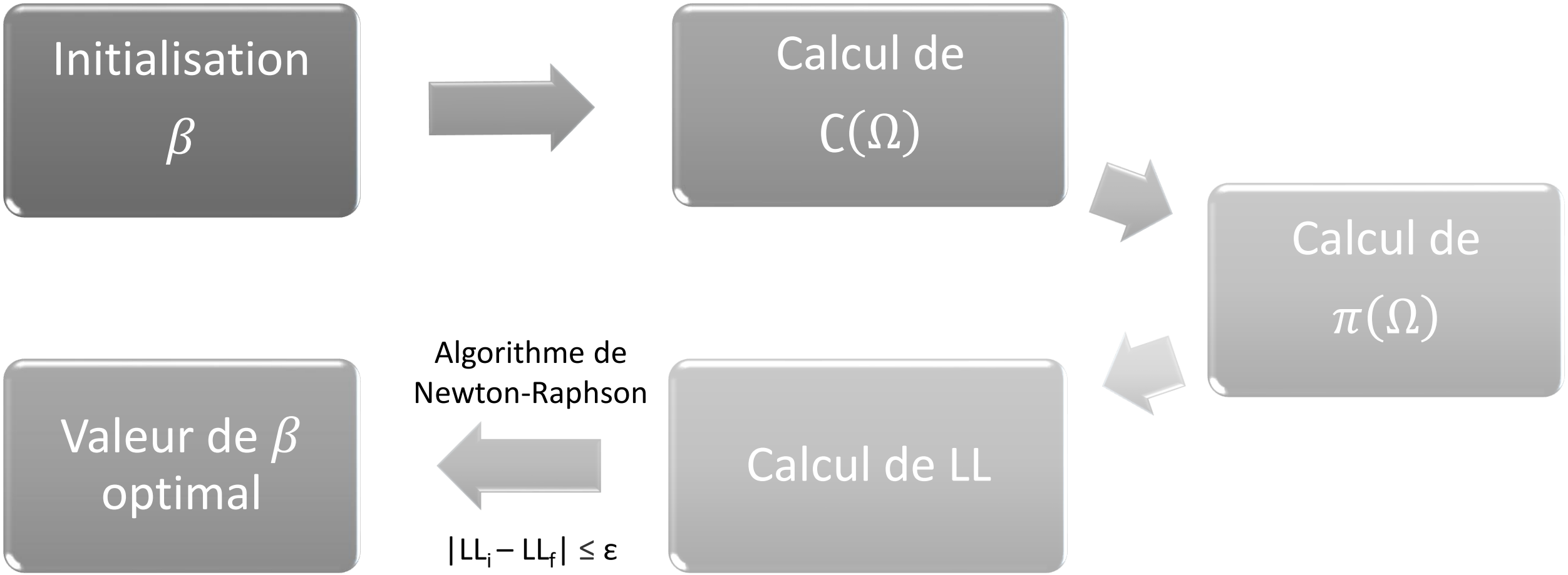
$$\beta_{n+1} = \beta_n - \left(\frac{\Delta LL}{\Delta \beta}\right)^{-1} \frac{\partial LL}{\partial \beta} \text{ avec :}$$

- Gradient : $\frac{\partial LL}{\partial \beta} = X(\Omega)^T \begin{pmatrix} y(\omega_1) - \pi(\omega_1) \\ \vdots \\ y(\omega_i) - \pi(\omega_i) \end{pmatrix}$

- Matrice hessienne : $\frac{\Delta LL}{\Delta \beta} = -X(\Omega)^T \begin{pmatrix} \pi(\omega_1)(1-\pi(\omega_1)) & & 0 \\ & \ddots & \\ 0 & & \pi(\omega_i)(1-\pi(\omega_i)) \end{pmatrix} X(\Omega)$

I. Modèle de la régression logistique

3) Algorithme de Newton-Raphson



II. Application

1) Enfant prématuré

	GEST	DILATE	EFFACE	CONSIS	CONTR	...	PARIT	DIAB	TRANSF	GEMEL	PREMATURE
0	31	3	100	3	1	...	0	2	2	1	1
1	28	8	0	3	1	...	0	2	1	2	1
2	31	3	100	3	2	...	0	2	1	1	1
3	27	2	75	3	2	...	1	2	1	2	1
4	28	6	75	3	2	...	0	2	1	1	1
..	...	...	...	...	...	...	...	...	...	...	...
385	34	3	75	3	1	...	2	2	1	1	1
386	31	0	0	1	1	...	1	2	1	1	1
387	31	2	100	3	1	...	0	2	1	2	1
388	29	0	0	1	1	...	0	2	1	1	1
389	31	1	100	3	1	...	1	2	2	1	1

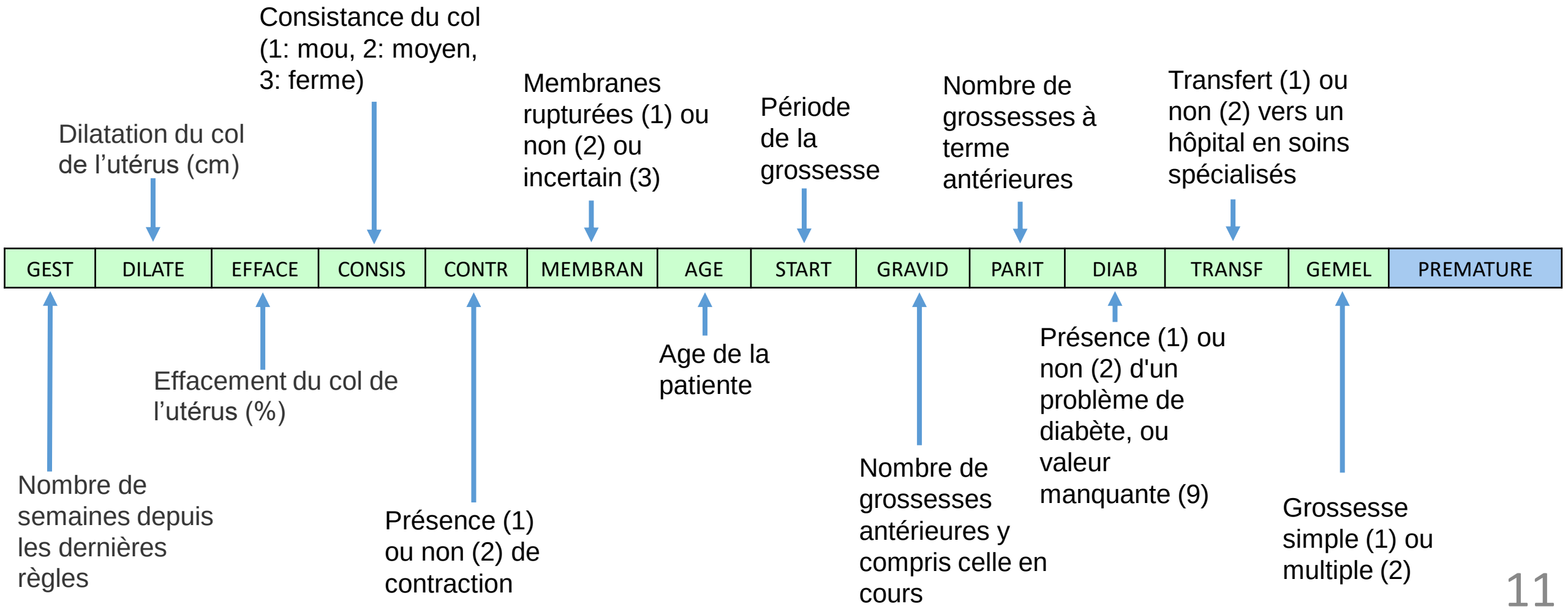
[390 rows x 14 columns]

Pourcentage de prématurés : **68.2051282051282 %**

Source : Ricco Rakotomalala, documentation cours M2 SISE, université de Lyon

II. Application

1) Enfant prématuré



II. Application

1) Enfant prématuré

GEST	DILATE	…	TRANSF	GEMEL	PREMATURE
31	3		2	1	1
28	8		2	1	1
31	3		1	1	1
…					
31	1		2	1	1

390

Test

GEST	DILATE	…	TRANSF	GEMEL	PREMATURE
31	3		2	1	1
27	2		1	2	1
28	2		1	2	1
…					
31	3		1	1	1

78

Entraînement

GEST	DILATE	…	TRANSF	GEMEL	PREMATURE
33	2		1	1	1
28	6		1	1	1
32	4		1	1	1
…					
30	1		1	1	1

312

II. Application

1) Enfant prématuré

Algorithme de Newton-Raphson

$$\beta = \begin{pmatrix} 3.37472998 \\ 0.07\mathbf{7}59746 \\ 0.39590916 \\ 0.01934558 \\ -0.005\mathbf{7}9219 \\ -0.6\mathbf{1}336834 \\ -1.36054406 \\ -0.03226219 \\ -0.81236701 \\ 0.1537388 \\ -0.55345183 \\ 0.14472214 \\ -0.62736333 \\ 1.\mathbf{3}1034411 \end{pmatrix}$$

Modules Python

$$\beta = \begin{pmatrix} 3.39959741 \\ 0.07\mathbf{5}62588 \\ 0.39587814 \\ 0.01932879 \\ -0.005\mathbf{4}7599 \\ -0.6\mathbf{0}55254 \\ -1.35383509 \\ -0.03214462 \\ -0.80416484 \\ 0.1532196 \\ -0.55186015 \\ 0.14476405 \\ -0.62550344 \\ 1.\mathbf{2}9199169 \end{pmatrix}$$

II. Application

2) Interprétation des résultats

Matrice de confusion

	Présence prédit	Absence prédit	Total
Présence (1)	a	b	a + b
Absence (0)	c	d	c + d
Total	a + c	b + d	n = a + b + c + d

Taux d'erreur	(b + c) / n
Taux de succès	(a + d)/n = 1 – tx_err
Sensibilité	a/(a + b)
Spécificité	d/(c + d)
Précision	a/(a + c)

II. Application

2) Interprétation des résultats

Algorithme de Newton-Raphson et modules Python

	Présence prédit	Absence prédit
Présence (1)	51	5
Absence (0)	8	14

Taux d'erreur	0.17
Taux de succès	**0.83**
Sensibilité	0.91
Spécificité	0.64
Précision	**0.86**

II. Application

2) Interprétation résultats

R^2 de McFadden	$1 - \frac{LL}{LL0}$
R^2 de Cox and Snell	$R^2_{CS} = 1 - \left(\frac{e^{LL0}}{e^{LL}}\right)^{\frac{2}{n}}$
R^2 de Nagelkerke	$\frac{R^2_{CS}}{1 - (e^{LL0})^{\frac{2}{n}}}$

$p_+ = n_+/n$

$LL0 = n * \ln(1 - p_+) + n_+ * \ln(p_+/(1 - p_+))$

Algorithme de Newton-Raphson

Log-vraisemblance (LL)	-151.9
R^2 de McFadden	0.2297
R^2 de Cox and Snell	0.2519
R^2 de Nagelkerke	0.3512

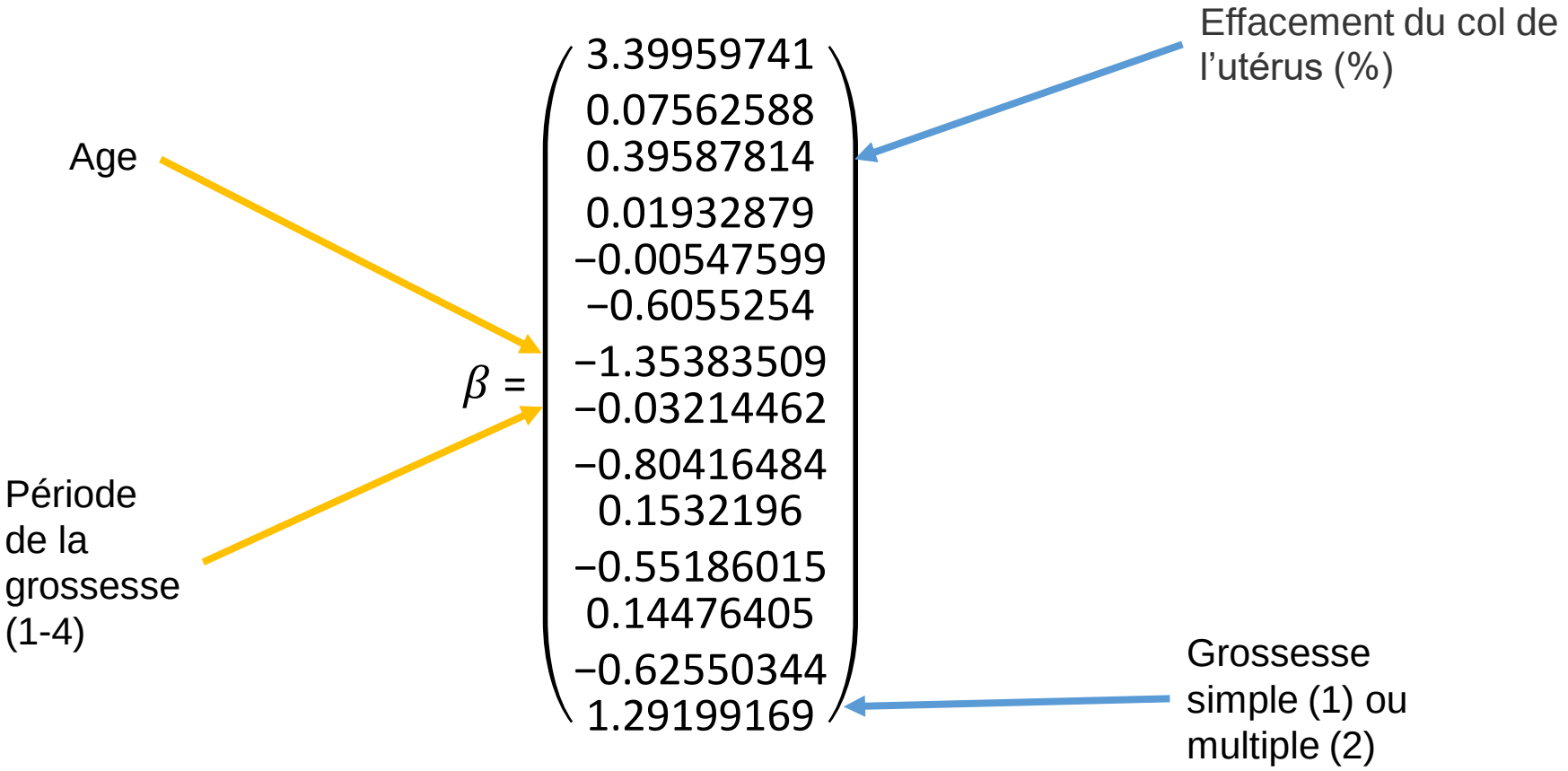
Modules Python

Log-vraisemblance (LL)	-152.3
R^2 de McFadden	0.2275
R^2 de Cox and Snell	0.2499
R^2 de Nagelkerke	0.3482

II. Application

2) Interprétation des résultats

Modules Python



$\beta_k \ll 0$: facteur protecteur
$\beta_k \gg 0$: facteur de risque

II. Application

2) Interprétation des résultats

Type de grossesse = T	PREMATURE = y
1	1
1	1
1	1
1	1

$$\beta = \begin{pmatrix} -0.44991162 \\ 1.00826566 \end{pmatrix} = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}$$

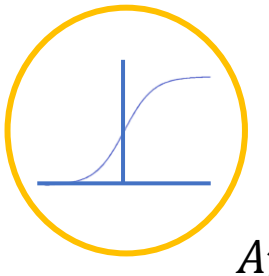
Taux de succès : 79%
Spécificité : 0%

$$P_{(T=1)}(y=1) = \frac{1}{1 + e^{-(\beta_0+\beta_1)}} \approx 0.636$$

$$P_{(T=2)}(y=1) = \frac{1}{1 + e^{-(\beta_0+2*\beta_1)}} \approx 0.827$$

$$P_{(T=2)}(y=1) - P_{(T=1)}(y=1) = 0.191$$

Conclusion



$Application$

$\max(LL)$

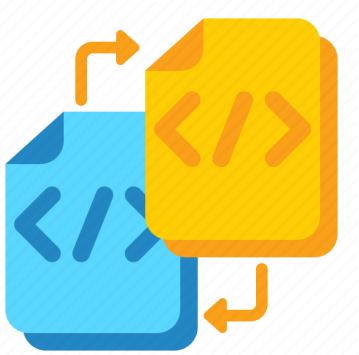
GEST	DILATE	EFFACE	...	GEMEL	*Prématuré*

0 1

Résultats :

- Taux de succès : 83 %
- Pseudos-R^2 > 0
- Facteurs de risque :
  Ex: grossesse multiple

Source : https://www.iconfinder.com/



Annexes

Source : https://www.iconfinder.com/

Programme

```
## Importation des bibliothèques
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix #évaluer modèle
from sklearn.model_selection import train_test_split #fonction diviser base de donnée une partie entrainement l'autre test
import math

## Lecture du fichier PYTHON
data = pd.read_excel('prematures.xls') #importation des données
data.PREMATURE = data.PREMATURE.map({'positif' : 1, 'negatif': 0})


def pourcentage(data):
    """Calcule le pourcentage de la présence de y (dernière colonne) au sein de data"""
    nom_y = data.columns[- 1]
    col_y = data[nom_y]
    count_1 = sum(col_y)
    count_0 = len(col_y) - count_1
    pct_of_y = (count_1/(count_1 + count_0)) * 100
    return pct_of_y


## Régression Logistique PYTHON : fonctions

def reg_logistique(data):
    """Effectue la régression logistique et renvoie le taux de succès du modèle, ses prédictions, ainsi que le modèle, x_train, y_train, y_test"""
    # Préparer les données d'entrainement
    X = data.iloc[:, :-1 ] #tout le tableau en supprimant la dernière colonne
    Y = data.iloc[:, -1 ] # dernière colonne seulement
    # Diviser les données en données d'entraînement et de test
    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) #20% pour le test et 80% pour l'entrainement, données prises au hasard
    # Test model
    model = LogisticRegression(solver='newton-cg')
    model.fit(x_train, y_train) #entrainement des données
    predictions = model.predict(x_test) #résulat des prédictions de y sous forme de np.array
    return accuracy_score(y_test, predictions), predictions, model, x_train, y_train, y_test
```

```python
def coeff(model, x_train):
    """Renvoie sous forme de liste les coefficients du modèle de régression logistique"""
    coeff = [model.intercept_[0]] #np.array
    _, nb_col = x_train.shape
    for i in range(nb_col):
        coeff.append(model.coef_[0][i]) #np.array matrice à une ligne
    return coeff


def log_vraissemblance(model, x_train, y_train):
    """Renvoie la log_vraissemblance d'un modèle à partir des données d'entrainement"""
    #probabilités d'affectation
    proba01 = model.predict_proba(x_train)
    #np.array : première colonne : proba appartenance à (Y = 0), seconde pour (Y = 1)
    somme = 0
    y_train_list = y_train.values.tolist()
    for i in range(len(y_train)):
        somme += y_train_list[i] * math.log(proba01[i][1]) + (1 - y_train_list[i]) * math.log(proba01[i][0])
    return somme


def liste_indice_ligne(tab):
    """Renvoie sous forme de liste les numéros des lignes présentes dans tab"""
    L = []
    for i in range(len(tab)):
        L.append(tab.index[i])
    return L


def lignes_to_data(lignes, data):
    """Renvoie les variables X et la variable cible Y dont les numéros de lignes sont présents dans la liste ligne sous forme de DataFrame"""
    x = data.iloc[:,:-1]
    x = x.iloc[lignes]
    y = data.iloc[:, -1]
    y = y.iloc[lignes]
    return x, y
```

Régression Logistique PYTHON : application

```
tx_succes, pred, model, x_train, y_train, y_test = reg_logistique(data)
print("Le taux de succes de data vaut", tx_succes)

coefficients = coeff(model, x_train)
lignes_train = liste_indice_ligne(x_train)
lignes_test = liste_indice_ligne(y_test)
LL = log_vraissemblance(model, x_train, y_train)
```

Régression Logistique PYTHON : application avec un taux de succès à 83%

```
coeff_83_pyth = [3.399597413120615, 0.0756258765706579, 0.3958781416459474, 0.019328789719555626, -0.0054759920508549624, -0.6055254403437981,
-1.3538350931564633, -0.03214462151066887, -0.8041648429410235, 0.15321955979787294, -0.5518601466874565, 0.14476404905074136, -0.625503438596877,
1.2919916864720984]

lignes_train_83 = [259, 176, 269, 169, 272, 241, 216, 117, 347, 380, 232, 321, 28, 156, 118, 79, 86, 376, 68, 27, 383, 206, 327, 307, 300, 153, 150,
213, 191, 361, 154, 237, 63, 151, 18, 130, 125, 69, 178, 105, 245, 162, 62, 226, 204, 250, 72, 301, 106, 78, 205, 83, 190, 160, 179, 384, 75, 6, 49,
359, 184, 144, 227, 306, 180, 214, 44, 266, 295, 338, 370, 37, 135, 33, 247, 38, 252, 188, 2, 152, 218, 54, 172, 360, 10, 374, 9, 124, 11, 59, 326,
220, 4, 352, 275, 1, 318, 270, 284, 313, 364, 173, 215, 158, 212, 177, 311, 265, 16, 129, 47, 148, 64, 114, 159, 111, 351, 381, 91, 310, 304, 323,
116, 115, 56, 291, 356, 353, 141, 8, 366, 93, 264, 278, 121, 279,70, 257, 375, 52, 228, 282, 199, 388, 147, 165, 61, 346, 57, 40, 357, 294, 271,109,
19, 132, 15, 208, 230, 131, 161, 183, 92, 34, 45, 175, 166, 100, 142, 211, 95, 322, 238, 328, 127, 344, 325, 332, 314, 20, 285, 349, 382, 187, 289,
243, 12, 171, 316, 29, 51, 222, 157, 81, 80, 368, 262, 225, 133, 139, 249, 281, 377, 287, 181, 341, 25, 202, 251, 274, 386, 60, 146, 149, 293, 288,
299, 84, 197, 17, 385, 292, 112, 26, 331, 97, 339, 334, 53, 89, 362, 42, 209, 102, 3, 85, 297, 71,155, 134, 365, 340, 108, 248, 23, 343, 224, 223,
219, 22, 167, 267, 309, 201, 244, 120, 164, 389, 41, 122, 48, 87, 193, 32, 254, 192, 280, 319, 239, 98, 268, 373, 255, 210, 182, 379, 330, 303, 354,
235, 369, 200, 195, 94, 231, 335, 13, 236, 140, 324, 315, 168, 348, 337, 65, 286, 24, 88, 240, 35, 5, 242, 229, 302, 256,246, 90, 298, 50, 170, 317,
263]

lignes_test_83 = [221, 55, 234, 67, 207, 137, 308, 372, 128, 58, 273, 7, 277, 283, 367, 123, 296, 174, 0, 74, 104, 253, 320, 110, 185, 107, 126, 336,
355, 96, 99, 363, 143, 371, 31, 103, 46, 77, 82, 290, 43, 350, 329, 276, 387, 76, 73, 312, 186, 198, 138, 345, 163, 145, 260, 261, 194, 136, 39, 113,
358, 189, 119, 36, 333, 14, 101, 21, 233, 378, 66, 258, 305, 30, 203, 196, 342, 217]

x_train_83, y_train_83 = lignes_to_data(lignes_train_83, data)
x_test_83, y_test_83 = lignes_to_data(lignes_test_83, data)


LL_83_pyth = -152.3287080826743

def reg_logistique_tx_succes(x_train, y_train, x_test):
    """Effectue la régression logistique sous python avec des valeurs d'entrainement et de test déja choisis"""
    model = LogisticRegression(solver='newton-cg')
    model.fit(x_train, y_train)
    predictions = model.predict(x_test)
    return predictions, model

pred_83, model_83 = reg_logistique_tx_succes(x_train_83, y_train_83, x_test_83)
```

```
## Lecture des données NR
valeurs = data.values.tolist()

def selection(valeurs, ind_l):
    """Supprime à valeurs toutes les lignes qui ne sont pas dans la liste ind_l """
    L = []
    for elt in ind_l:
        L.append(valeurs[elt])
    return L

valeurs_train = selection(valeurs, lignes_train_83)

## Régression Logistique NR : fonctions
def beta(valeurs):
    """Initialise le vecteur beta au départ"""
    b = [1]
    for _ in range(len(valeurs[0]) - 1):
        b.append(0)
    poids = np.array(b)
    return poids.reshape((len(poids), 1))
    #mettre sous forme de matrice afin de faire les calculs

def calcul_mat_X(valeurs):
    """ Matrice X : chaque ligne correspond aux variables d'un individu prédédé par 1 """
    L = [[1] for i in range(len(valeurs))]
    for i in range(len(valeurs)):
        L[i].extend(valeurs[i][:-1])
    return np.array(L)

def calcul_C(poids, mat_X):
    """Calcule C sous forme de matrice selon le vecteur beta"""
    return (mat_X).dot(poids)

def calcul_pi(C):
    """Calcul de pi sous forme de liste"""
    pi = []
    for i in range(len(C)):
        pi.append(1/(1 + math.exp(-C[i][0])))
    return pi

def calcul_LL(pi, valeurs):
    """Calcul de la log-vraisemblance LL"""
    LL = []
    pos_y = len(valeurs[0]) - 1 # correspond à la position de la variable étudié que l'on suppose à la dernière place
    for i in range(len(pi)):
        LL.append(valeurs[i][pos_y] * math.log(pi[i]) + (1 - valeurs[i][pos_y]) * math.log(1 - pi[i]))
    return sum(LL)
```

```python
def calcul_mat_grad_col(valeurs, pi):
    """Calcule la matrice colonne qui intervient dans le calcul du gradient"""
    mat = np.array([[valeurs[i][-1] - pi[i]] for i in range(len(valeurs))])
    return mat


def calcul_mat_hessienne_diag(pi, valeurs):
    """Calcule la matrice diagonale qui intervient dans le calcul de la matrice hessienne"""
    mat = np.diag([pi[i] * (1 - pi[i]) for i in range(len(valeurs))])
    return mat

def LL_mat_hessienne_gradient(valeurs, poids, mat_X):
    """Renvoie la valeur de la déviance, la matrice hésienne et la matrice du gradient"""
    C = calcul_C(poids, mat_X)
    pi = calcul_pi(C)
    LL = calcul_LL(pi, valeurs)
    mat_grad_col = calcul_mat_grad_col(valeurs, pi)
    mat_hessienne_diag = calcul_mat_hessienne_diag(pi, valeurs)
    gradient = np.dot(mat_X.T, mat_grad_col)
    mat_hessienne = (- mat_X.T).dot(mat_hessienne_diag).dot(mat_X)
    return LL, mat_hessienne, gradient


def LOGIT(valeurs, epsilon):
    """Applique la régression logistique en utilisant l'algorithme de Newton-Raphson dans le but de trouver la matrice poids maximisant la log-vraisemblance"""
    #def des paramètres initiaux
    poids = beta(valeurs)
    mat_X = calcul_mat_X(valeurs)
    #Initial
    LL_i, mat_hessienne_i, gradient_i = LL_mat_hessienne_gradient(valeurs, poids, mat_X)
    #Etat Initial + 1
    poids = poids - np.dot(np.linalg.inv(mat_hessienne_i), gradient_i)
    LL_f, mat_hessienne_f, gradient_f = LL_mat_hessienne_gradient(valeurs, poids, mat_X)
    while abs(LL_f - LL_i) > epsilon:
        LL_i = LL_f
        poids = poids - np.dot(np.linalg.inv(mat_hessienne_f), gradient_f)
        LL_f, mat_hessienne_f, gradient_f = LL_mat_hessienne_gradient(valeurs, poids, mat_X)
    return poids


coeff_83_nr = [[ 3.37472998],
       [ 0.07759746],
       [ 0.39590916],
       [ 0.01934558],
       [-0.00579219],
       [-0.61336834],
       [-1.36054406],
       [-0.03226219],
       [-0.81236701],
       [ 0.1537388 ],
       [-0.55345183],
       [ 0.14472214],
       [-0.62736333],
       [ 1.31034411]]
```

```
## Evaluation : Matrice de confusion

def mat_conf_pyth(y_test, pred):
    """Renvoie la matrice de confusion de la régression logistique sous python, sous la forme souhaité"""
    mat_conf_pyth = confusion_matrix(y_test,pred) #ligne = observé, colonne = prédictions
    observe_0, observe_1 = mat_conf_pyth
    vrai_neg, faux_pos = observe_0
    faux_neg, vrai_pos = observe_1
    mat_conf_pyth = np.array([[vrai_pos, faux_neg],[faux_pos, vrai_neg]])
    return mat_conf_pyth



def proba(valeurs, poids):
    """Calcul la probabililté de la présence de y pour les individus dans valeurs selon beta"""
    X = calcul_mat_X(valeurs)
    C = calcul_C(poids, X)
    pi = calcul_pi(C)
    return pi

def mat_conf_nr(valeurs, poids):
    """Construit la matrice de confusion pour la régression logistique implémenté avec l'algorithme de Newton-Raphson"""
    pi = proba(valeurs, poids)
    vrai_pos, faux_pos, vrai_neg, faux_neg = 0, 0, 0, 0
    ind_y = len(valeurs[0]) - 1
    for i in range(len(pi)):
        if valeurs[i][ind_y] == 1:
            if pi[i] >= 0.5:
                vrai_pos += 1
            else:
                faux_neg += 1
        else:
            if pi[i] < 0.5:
                vrai_neg += 1
            else:
                faux_pos += 1
    return np.array([[vrai_pos, faux_neg],[faux_pos, vrai_neg]])
```

```python
def indicateurs(mat_conf):
    """Renvoie le taux d'erreur, le taux de succès, la sensibilité, la spécificité et la précision de notre modèle d'après la matrice de confusion"""
    taux_erreur = (mat_conf[0][1] + mat_conf[1][0])/sum(sum(mat_conf))
    taux_succes = 1 - taux_erreur
    sensibilite = mat_conf[0][0]/(mat_conf[0][0] + mat_conf[0][1])
    specificite = mat_conf[1][1]/(mat_conf[1][1] + mat_conf[1][0])
    precision = mat_conf[0][0]/(mat_conf[0][0] + mat_conf[1][0])
    return taux_erreur, taux_succes, sensibilite, specificite, precision


#nr
valeurs_test = selection(valeurs, lignes_test_83)
mat_conf_83_nr = mat_conf_nr(valeurs_test, coeff_83_nr)

#python
mat_conf_83_pyth = mat_conf_pyth(y_test_83, pred_83)


ind = indicateurs(mat_conf_83_nr)
```

Evaluation : Les pseudos-R²

```python
def calcul_LL0(y_train):
    """Renvoie le log-vraissemblance du modèle trivial"""
    nb_pos = sum(y_train)
    p_pos = nb_pos/len(y_train)
    LL0 = len(y_train) * math.log(1 - p_pos) + nb_pos * math.log(p_pos/(1 - p_pos))
    return LL0

def R2_McFadden(LL0, LL):
    """Retourne le R2 de McFadden"""
    return 1 - LL/LL0

def R2_Cox_and_Snell(LL, LL0, y_train):
    """Retourne le R2 de Cox and Snell"""
    L = math.exp(LL)
    L0 = math.exp(LL0)
    R2CS = 1 - (L0/L)**(2/len(y_train))
    return R2CS
```

```
def R2_Nagelkerke(LL0, y_train, R2CS):
    """Retourne le R2 de Nagelkerke"""
    L0 = math.exp(LL0)
    max_R2CS = 1 - (L0)**(2/len(y_train))
    return R2CS/max_R2CS


LL0 = calcul_LL0(y_train_83) #-197.17718619089442

#nr
LL_83_nr = calcul_LL(proba(valeurs_train, coeff_83_nr), valeurs_train)  #-151.88303645925757

R2MF_nr = R2_McFadden(LL0, LL_83_nr) # 0.229712932853124
R2CS_nr = R2_Cox_and_Snell(LL_83_nr, LL0, valeurs_train) #0.2519961198571321
R2N_nr = R2_Nagelkerke(LL0, valeurs_train, R2CS_nr) #0.3512307563050402 5

#python

R2MF_pyth = R2_McFadden(LL0, LL_83_pyth) #0.22745267327631236
R2CS_pyth = R2_Cox_and_Snell(LL_83_pyth, LL0, y_train_83) #0.2498561150736 1127
R2N_pyth = R2_Nagelkerke(LL0, y_train_83, R2CS_pyth) #0.348248029828781 1
```

```
## Régression Logistique selon juste GEMEL (= type de grossesse)


coeff_GEMEL = [-0.4499116214117206, 1.0082656556309353]

lignes_train_GEMEL = [69, 15, 245, 114, 142, 223, 227, 181, 305, 250, 7, 207, 157, 368, 68, 197, 110, 339, 278, 293, 2, 287, 289, 163,
357, 298, 96, 77, 5, 193, 109, 195, 70, 326, 184, 233, 369, 26, 313, 327, 374, 363, 316, 266, 187, 20, 167, 330, 317, 279, 241, 30, 354,
388, 118, 57, 244, 81, 101, 380, 44, 366, 148, 13, 375, 67, 54, 133, 239, 61, 355, 71, 43, 178, 389, 120, 59, 58, 84, 315, 358, 102, 85,
126, 229, 98, 151, 171, 206, 337, 161, 25, 186, 226, 303, 132, 79, 53, 27, 209, 292, 146, 179, 341, 103, 42, 165, 230, 324, 73, 33, 384,
40, 304, 277, 129, 11, 131, 204, 48, 47, 340, 60, 342, 381, 123, 282, 288, 78, 336, 377, 300, 215, 256, 29, 306, 35, 160, 45, 320, 91,
200, 192, 321, 56, 370, 228, 159, 108, 235, 318, 299, 333, 21, 124, 31, 225, 252, 62, 19, 286, 92, 383, 268, 134, 331, 112, 144, 64, 52,
247, 111, 182, 173, 127, 387, 211, 334, 34, 170, 263, 97, 93, 154, 32, 18, 322, 362, 219, 314, 75, 270, 198, 238, 364, 349, 261, 76,
386, 8, 224, 255, 276, 272, 183, 249, 348, 382, 6, 28, 191, 296, 353, 285, 344, 371, 372, 385, 89, 302, 356, 217, 325, 307, 376, 367,
199, 164, 86, 117, 328, 343, 237, 16, 88, 17, 87, 212, 130, 128, 145, 260, 115, 295, 46, 104, 168, 153, 176, 273, 253, 36, 24, 281, 0,
63, 150, 10, 41, 259, 347, 80, 143, 301, 359, 139, 274, 72, 138, 162, 201, 310, 242, 210, 172, 23, 309, 107, 22, 51, 338, 38, 379, 202,
373, 3, 50, 232, 194, 329, 243, 152, 352, 196, 65, 190, 360, 113, 319, 350, 240, 216, 312, 82, 213, 121, 262, 291, 37, 269, 1, 175]


lignes_test_GEMEL = [345, 135, 149, 271, 99, 378, 264, 208, 234, 147, 9, 66, 290, 105, 323, 4, 189, 205, 257, 335, 100, 169, 116, 311,
248, 258, 90, 218, 275, 251, 267, 12, 156, 185, 294, 74, 49, 155, 280, 332, 140, 254, 166, 83, 180, 177, 220, 94, 125, 365, 122, 265,
136, 221, 361, 137, 106, 222, 158, 297, 246, 14, 214, 55, 308, 236, 284, 346, 174, 283, 188, 141, 231, 203, 351, 39, 95, 119]

x_train_GEMEL, y_train_GEMEL = lignes_to_data(lignes_train_GEMEL, data.iloc[:, [-2, -1]])
x_test_GEMEL, y_test_GEMEL = lignes_to_data(lignes_test_GEMEL, data.iloc[:, [-2, -1]])

pred_GEMEL, model_GEMEL = reg_logistique_tx_succes(x_train_GEMEL, y_train_GEMEL, x_test_GEMEL)

#interprétations

mat_conf_GEMEL = mat_conf_pyth(y_test_GEMEL, pred_GEMEL)
#[[62,  0],[16,  0]]

ind_GEMEL = indicateurs(mat_conf_GEMEL)
#taux_erreur, taux_succes, sensibilite, specificite, precision = (0.20512820512820512, 0.7948717948717949, 1.0, 0.0, 0.7948717948717948)
```

```
## Evaluation : Diagramme de fiabilité
coeff_83_pyth_tab = np.array(coeff_83_pyth)
coeff_83_pyth_mat = coeff_83_pyth_tab.reshape((len(coeff_83_pyth_tab), 1))


def creation_liste_travail(valeurs, coeff):
    """Créee une liste où chaque élément est une liste contenant la proba et le numéro de la ligne de l'individu"""
    pi = proba(valeurs, coeff)
    L = []
    for i in range(len(pi)):
        L.append([pi[i], i])
    L.sort()
    return L


#L = creation_liste_travail(valeurs, coeff_83_nr) #coeff_83_pyth_mat
L = creation_liste_travail(valeurs_test, coeff_83_nr) #coeff_83_pyth_mat

def division(L, val):
    """Indique l'indice de la 1ère valeur strictement supérieure à val sachant que les éléments de la liste sont sous la forme [val, ind]"""
    n = len(L)
    a = 0
    b = n - 1
    m = (a + b)//2
    while b - a != 1:
        if L[m][0] > val:
            b = m
        else:
            a = m
        m = (a + b)//2
    return b


ind_25 = division(L,0.25)
ind_5 = division(L,0.5)
ind_75 = division(L,0.75)

division = [L[: ind_25], L[ind_25 : ind_5], L[ind_5 : ind_75], L[ind_75:]]
```

```python
def somme_liste_premier_elt(L):
    """Renvoie la somme de la première colonne dans L"""
    somme = 0
    for i in range(len(L)):
        somme += L[i][0]
    return somme

def moy_score(division):
    """Renvoie la moyenne des probas pour chaque division"""
    L = []
    for i in range(len(division)):
        somme = somme_liste_premier_elt(division[i])
        L.append(somme/len(division[i]))
    return L

def proportion_pos(division, valeurs_tot):
    """Retourne la proportion de postifs dans chaque division"""
    pos_y = len(valeurs_tot[0]) - 1
    prop_pos = []
    for i in range(len(division)):
        l = division[i]
        somme = 0
        for j in range(len(l)):
            ind = l[j][1]
            somme += valeurs_tot[ind][pos_y]
        prop_pos.append(somme/len(l))
    return prop_pos


moyenne_score = moy_score(division)
prop_pos = proportion_pos(division, valeurs)

# TRACER GRAPH

plt.plot(moyenne_score, prop_pos, color = 'b', marker='s', linestyle= ':')
plt.plot([0, 1], [0, 1], color = 'g')
plt.ylabel("Proportion des positifs")
plt.xlabel("Moyenne des probabilités")
plt.title("Diagramme de fiabilité : intervalle 0.25")
plt.show()
```

```
## Evaluation : Séparateur linéaire
def extraire_col_et_var(valeurs, ind):
    """Sépare les valeurs (sous forme de liste) de la colonne ind selon y = 0 et y = 1 en deux listes col_0 et col_1"""
    col_0 = []
    col_1 = []
    ind_var = len(valeurs[0]) - 1
    for i in range(len(valeurs)):
        if valeurs[i][ind_var] == 0:
            col_0.append(valeurs[i][ind])
        else:
            col_1.append(valeurs[i][ind])
    return col_0, col_1

def reg_log_2_var(nb_col_ord, nb_col_abs, data):
    """Renvoie sous forme de tableau les antécédéents et les images de la frontière formé entre ord et abs ainsi que le taux de succès de la
régression logistique avec les variables abs et ord"""
    fichier_liste = data.values.tolist()
    X = data.iloc[:, [nb_col_ord, nb_col_abs]]
    y = data.iloc[:, -1]
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    model = LogisticRegression(solver='newton-cg')
    model.fit(x_train, y_train)
    predictions = model.predict(x_test)
    coeffi = coeff(model, x_train)
    col_0_abs, col_1_abs = extraire_col_et_var(fichier_liste, nb_col_abs)
    antecedent = col_0_abs + col_1_abs
    def f(x):
        return (- coeffi[0] - coeffi[2] * x)/coeffi[1]
    f_vect = np.vectorize(f)
    image = f_vect(np.array(antecedent))
    return np.array(antecedent),image, accuracy_score(y_test, predictions)


def graph(nb_col_x, nb_col_y, nom_x, nom_y, data):
    """Trace le séparateur linéaire avec en absicisse x et en ordonnée y"""
    fichier_liste = data.values.tolist()
    x, y, tx_succ = reg_log_2_var(nb_col_y, nb_col_x, data)
    print(tx_succ)
    x_0, x_1 = extraire_col_et_var(fichier_liste, nb_col_x)
    y_0, y_1 = extraire_col_et_var(fichier_liste, nb_col_y)
    plt.scatter(x_0, y_0, c='b', marker='D', label='Non prématuré')
    plt.scatter(x_1, y_1, c='g', marker='o', label='Prématuré' )
    plt.plot(x, y, color='k', label='Frontière')
    plt.ylabel(nom_y)
    plt.xlabel(nom_x)
    plt.title("Nuage de points et frontière de séparation")
    plt.legend(loc='best')
    plt.show()

#crise cardiaque
data_h = pd.read_csv('heart.csv')
graph(0, 7,  "Age", "fréquence cardiaque maximale atteinte", data_h)
```

```python
#enfant prématuré
def graph_subd(nb_col_x, nb_col_y, nom_x, nom_y, data):
    """Trace le séparateur linéaire avec en absicisse x et en ordonnée y pour var_cible = 1 et 0, pour var_cible = 1, et var_cible = 0"""
    fichier_liste = data.values.tolist()
    x, y, tx_succ = reg_log_2_var(nb_col_y, nb_col_x, data)
    print(tx_succ)
    x_0, x_1 = extraire_col_et_var(fichier_liste, nb_col_x)
    y_0, y_1 = extraire_col_et_var(fichier_liste, nb_col_y)
    #fig1
    plt.subplot(1, 2, 1) #nombre de lignes, le nombre de colonnes, numéro du tracé
    plt.plot(x, y, color='k', label='Frontière')
    plt.scatter(x_0, y_0, c='b', marker='D', label='Non prématuré')
    plt.scatter(x_1, y_1, c='g', marker='o', label='Prématuré' )
    plt.ylabel(nom_y)
    plt.xlabel(nom_x)
    plt.title("Nuage de points et frontière de séparation")
    plt.legend(loc='best')
    #fig2
    plt.subplot(2, 2, 2)
    plt.scatter(x_1, y_1, c='g', marker='o', label='Prématuré' )
    plt.plot(x, y, color='k', label='Frontière')
    plt.title("Nuage de points et frontière de séparation pour les enfants prématurés")
    plt.legend(loc='best')
    plt.ylabel(nom_y)
    plt.xlabel(nom_x)
    #fig3
    plt.subplot(2, 2, 4)
    plt.scatter(x_0, y_0, c='b', marker='D', label='Non prématuré')
    plt.plot(x, y, color='k', label='Frontière')
    plt.title("Nuage de points et frontière de séparation pour les enfants non prématurés")
    plt.legend(loc='best')
    plt.ylabel(nom_y)
    plt.xlabel(nom_x)
    plt.show()


graph_subd(6, 2,  "Age", "Effacement du col de l'utérus (%)", data)
```

```
##Verif de formule LL0 sur un exemple
val = valeurs[:]
for i in range(390):
    liste_vide = [0 for i in range(13)]
    prem = val[i][-1]
    val[i] = liste_vide
    val[i].append(prem)

data_bis = pd.DataFrame( val, columns = ['GEST', 'DILATE', 'EFFACE', 'CONSIS', 'CONTR', 'MEMBRAN', 'AGE','STRAT', 'GRAVID', 'PARIT', 'DIAB', 'TRANSF', 'GEMEL', 'PREMATURE'])

tx_succes_0, pred_0, model_0, x_train_0, y_train_0, y_test_0 = reg_logistique(data_bis)
print(tx_succes) #0.7435897435897436

LL_pyth_0 = log_vraisemblance(model_0, x_train_0, y_train_0) #-198.59242050798096

LL_form = calcul_LL0(y_train_0) #-198.59242050798164
```

Diagramme de fiabilité

Construction d'un diagramme de fiabilité

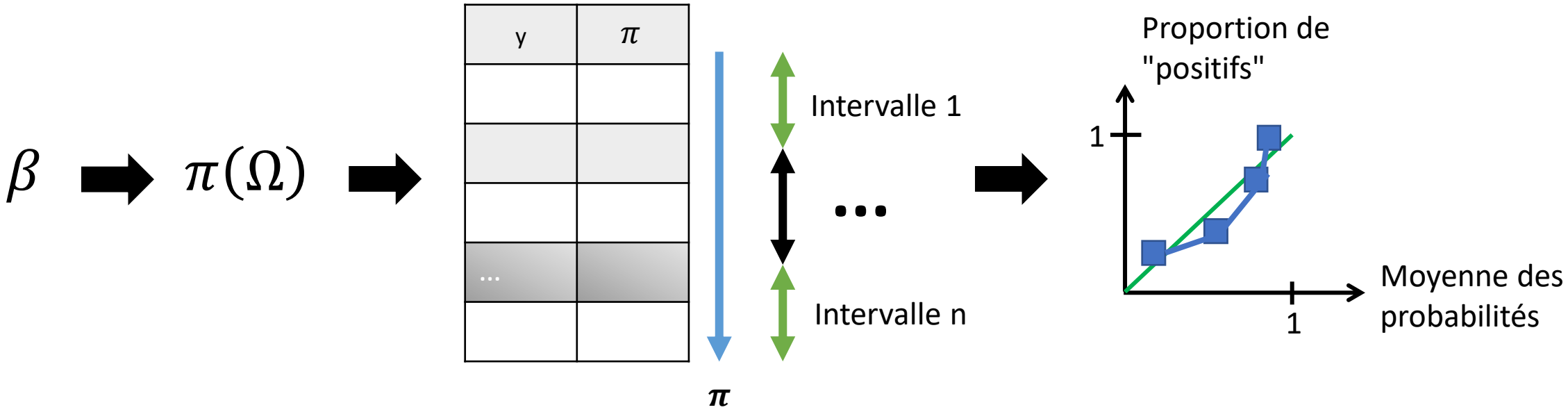
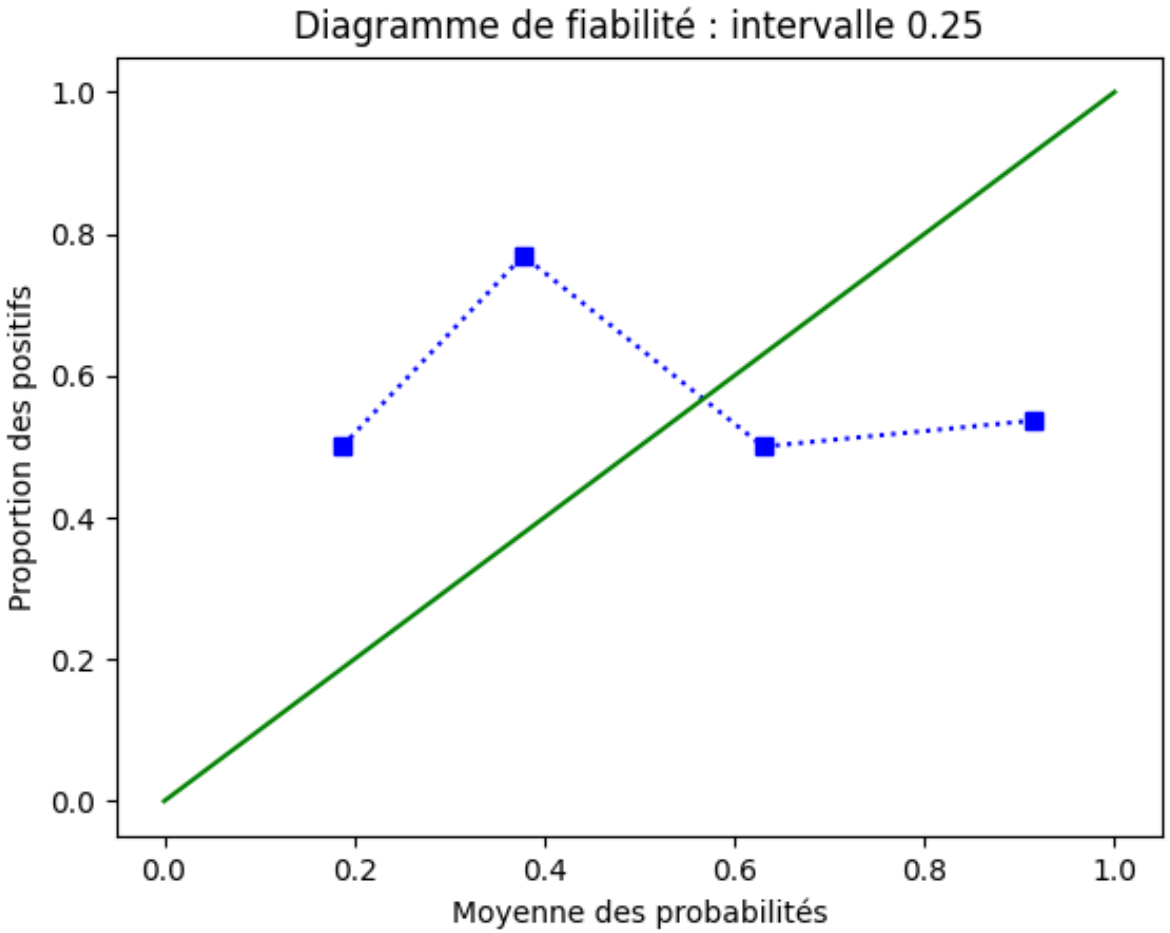


Diagramme de fiabilité
Sur les valeurs tests

Algorithme de Newton-Raphson



Modules Python

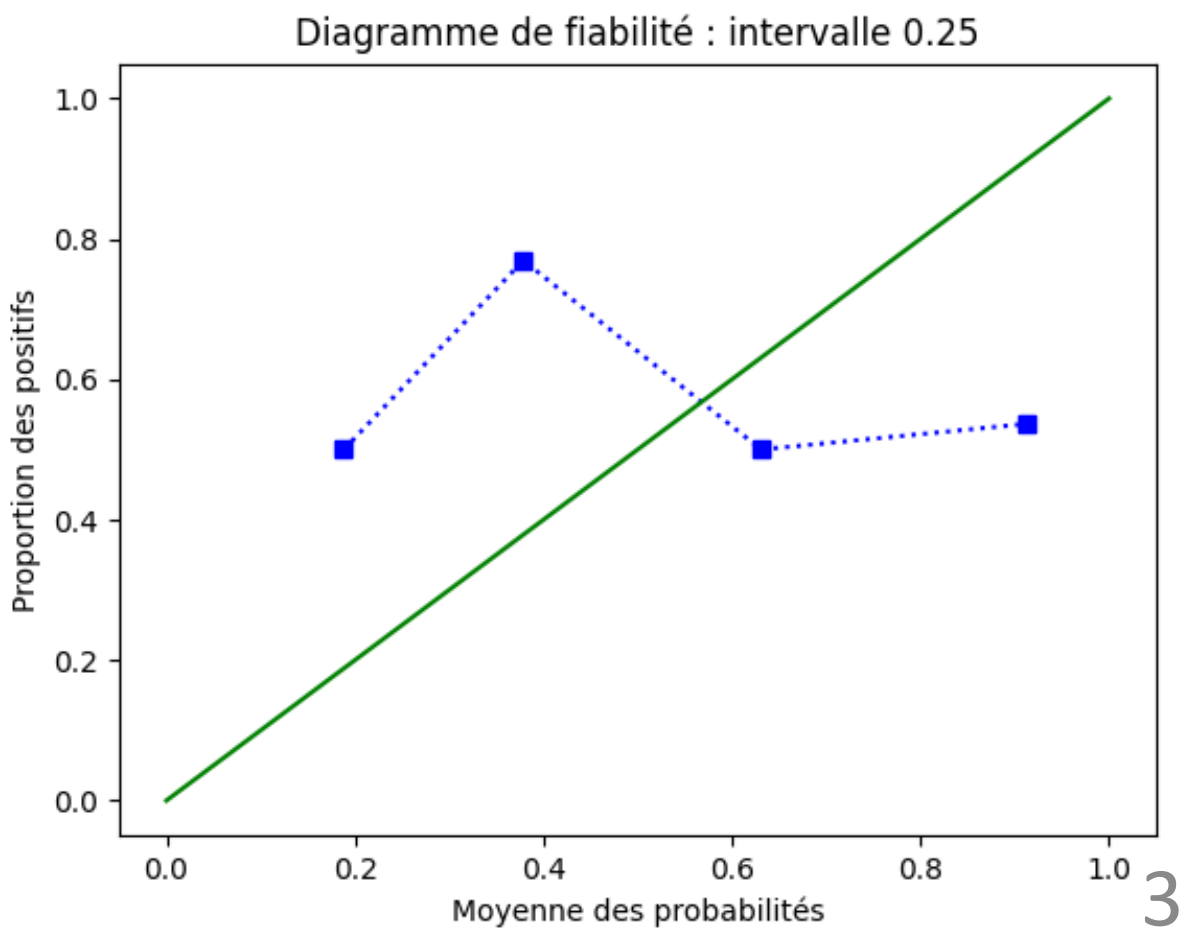
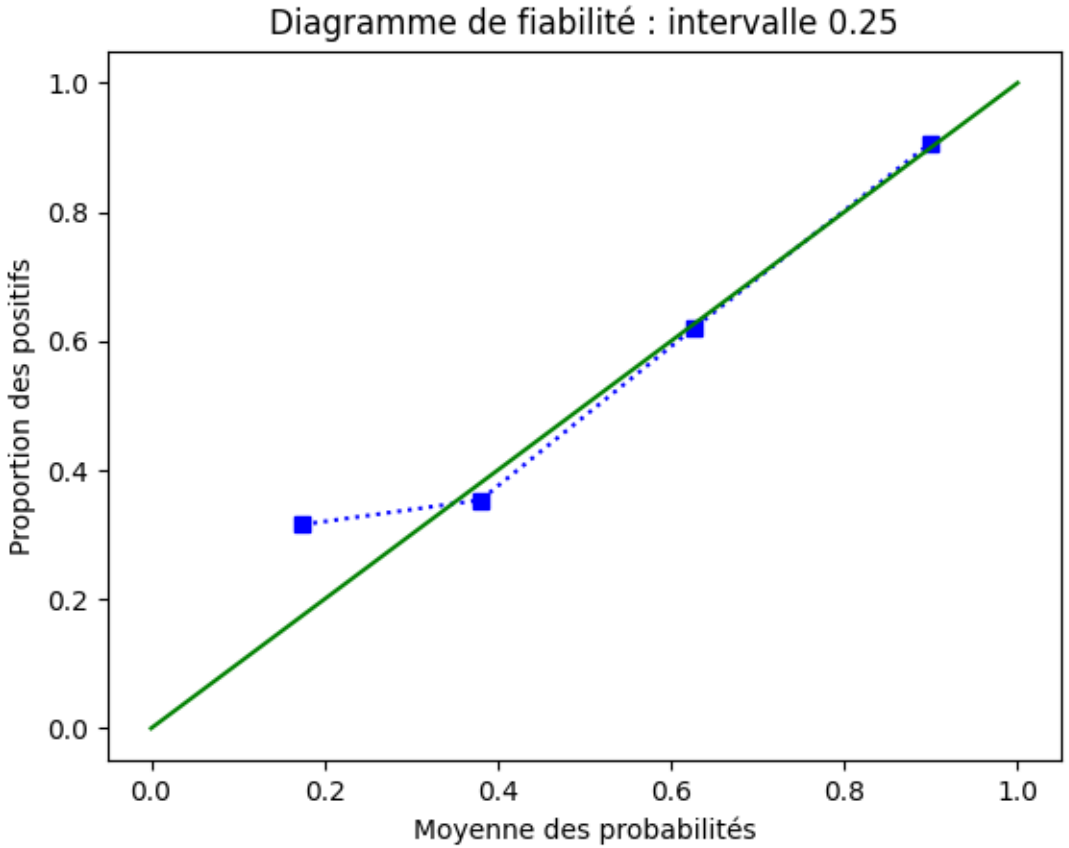


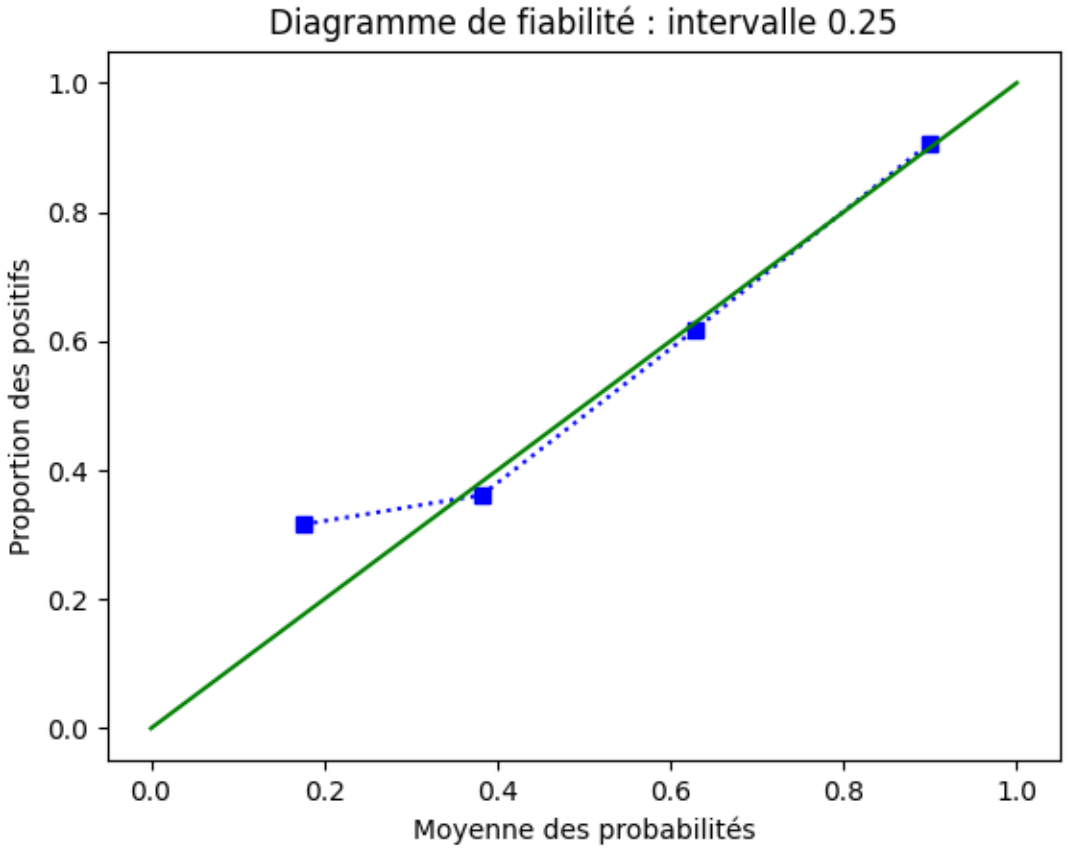
Diagramme de fiabilité

Sur l'ensemble des valeurs

Algorithme de Newton-Raphson



Modules Python



Séparateur linéaire

Y	X	y
…		

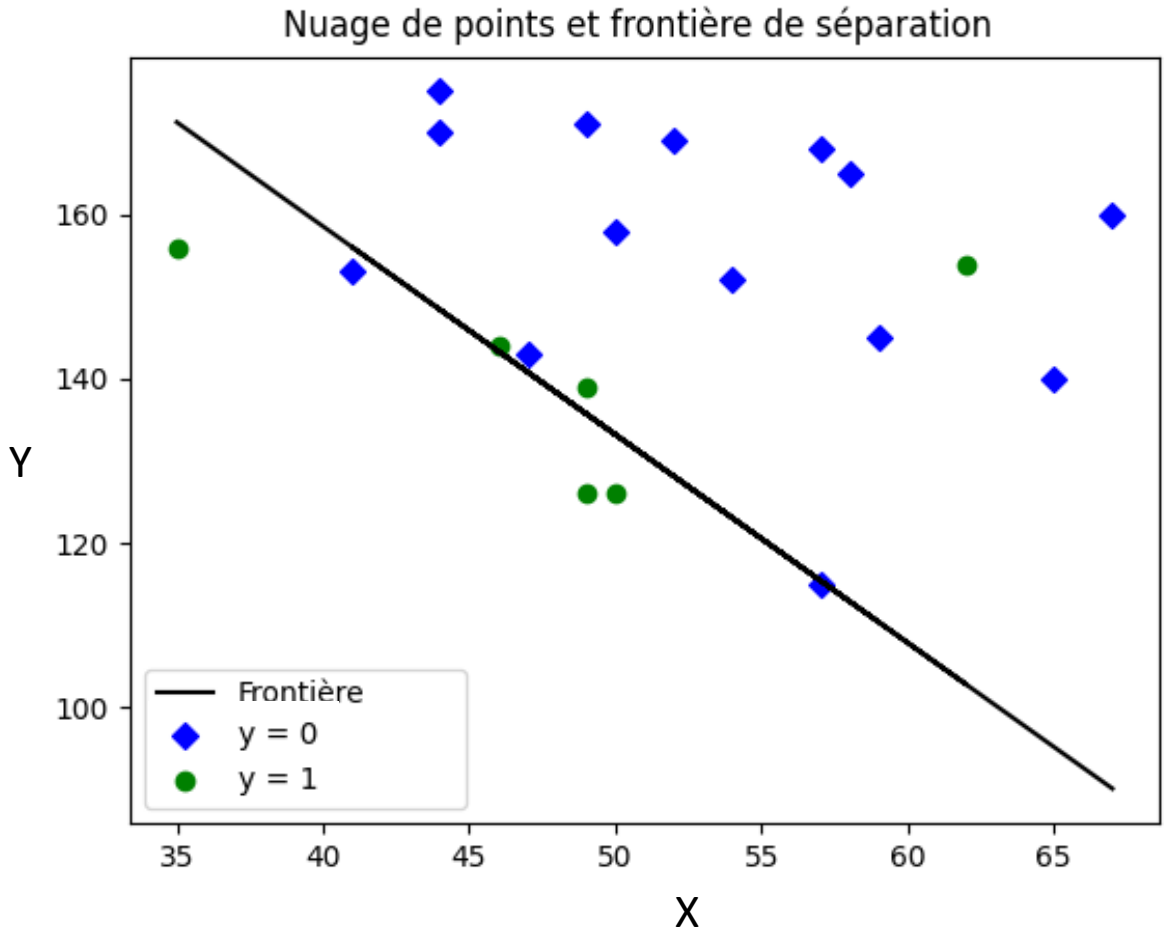
$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix}$$

Si C > 0, alors y_p = +, sinon y_p = −

=> C = 0 définit la frontière séparant les positifs des négatifs

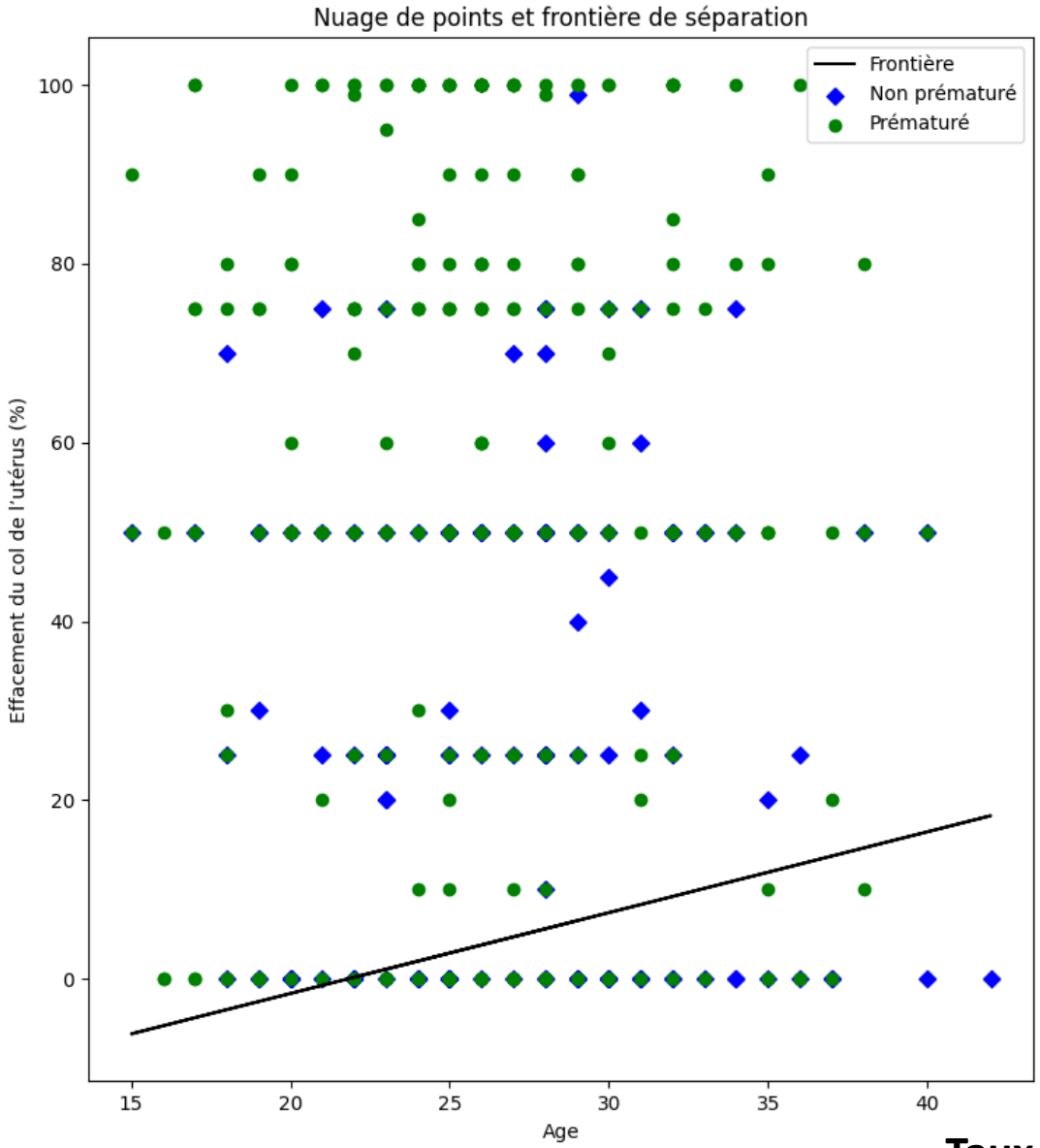
$$C = \beta_0 + \beta_1 * Y + \beta_2 * X$$

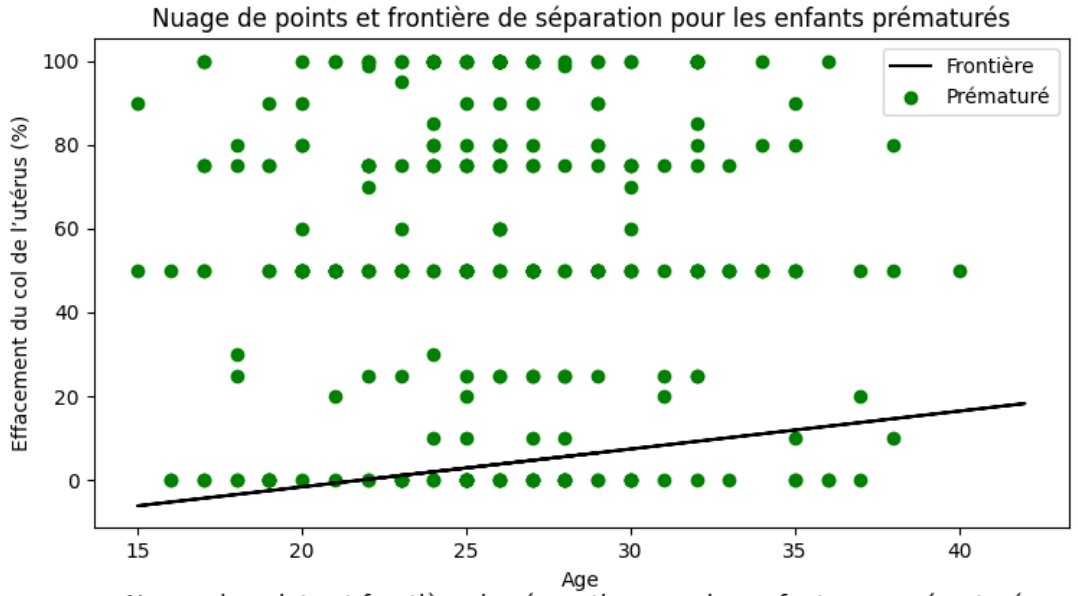
$$Y = \frac{-\beta_0 - \beta_2 * X}{\beta_1}$$

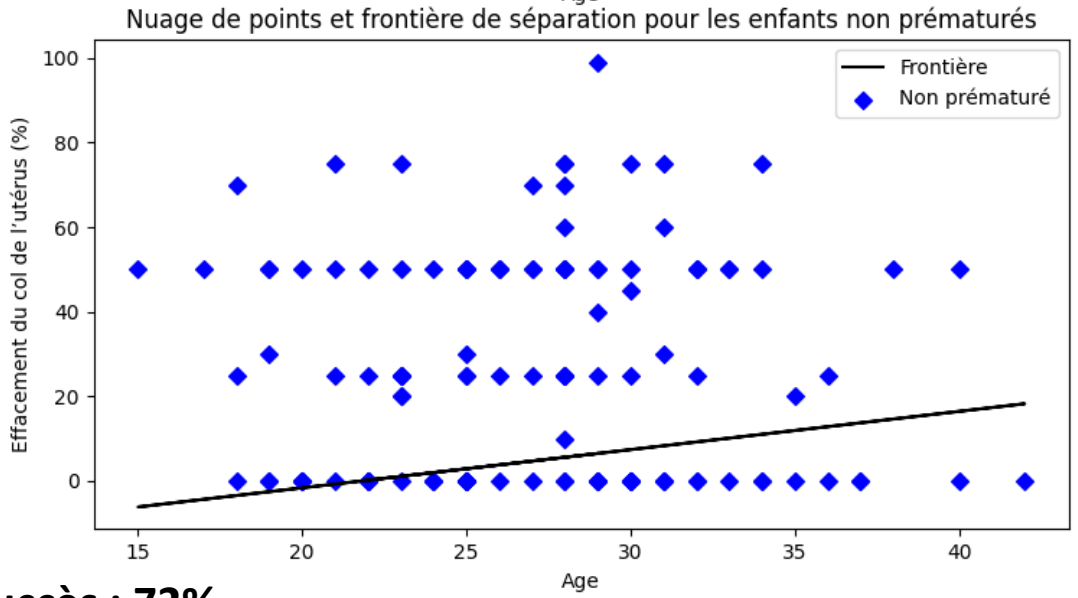


Séparateur linéaire

Enfant prématuré



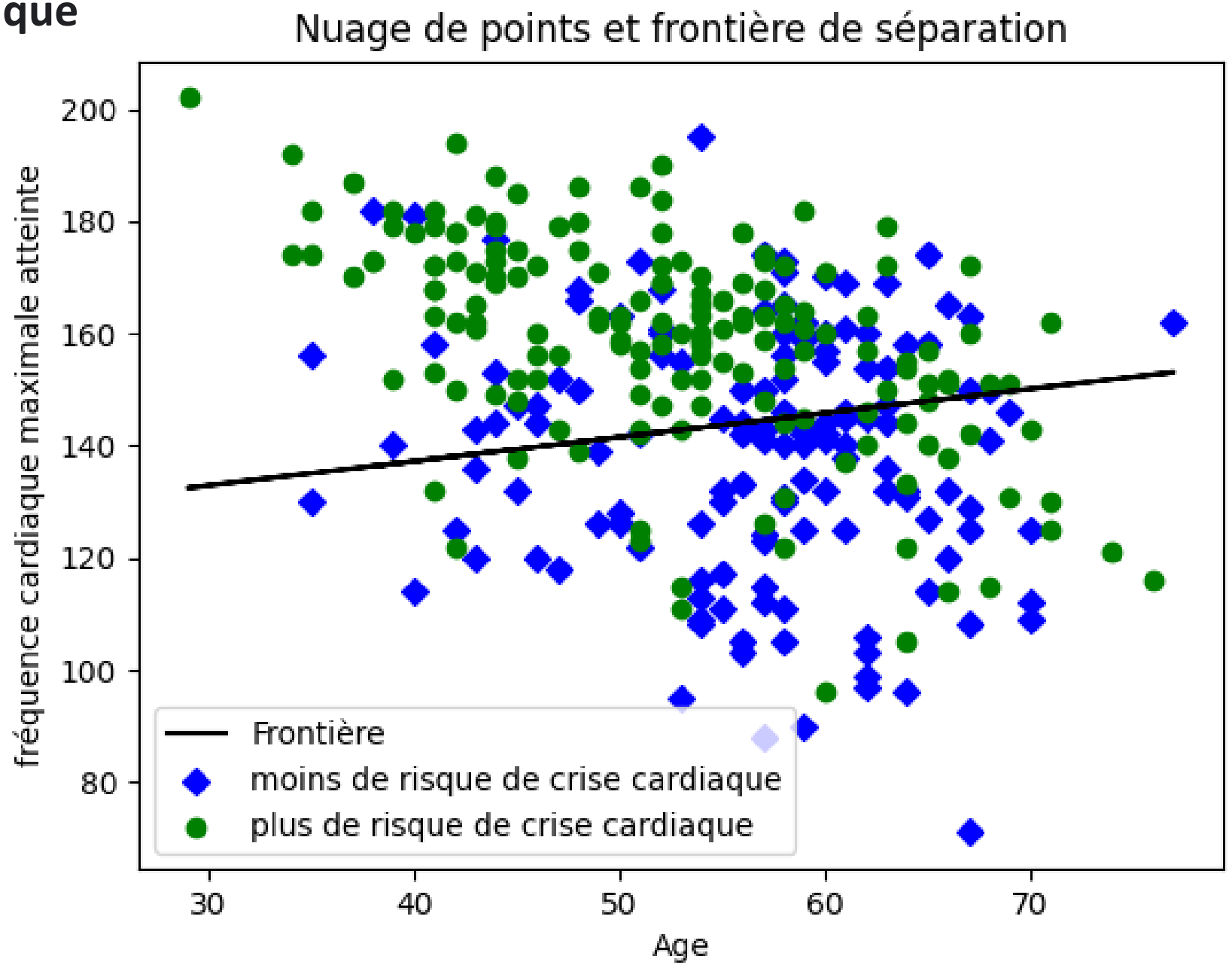




Taux de succès : 72%

Séparateur linéaire

Crises cardiaque



Taux de succès : 77%

Calcul

$$\pi(\omega) = \frac{1}{1 + e^{-(\beta_0 + \sum \beta_k x_k(\omega))}} \qquad 1 - \pi(\omega) = \frac{1 + e^{-(\beta_0 + \sum \beta_k x_k(\omega))} - 1}{1 + e^{-(\beta_0 + \sum \beta_k x_k(\omega))}}$$

$$LL = \sum_j y(\omega_j) \ln(\pi(\omega_j)) + (1 - y(\omega_j)) \ln(1 - \pi(\omega_j))$$

$$\frac{\partial \pi(\omega)}{\partial \beta_i} = \frac{\partial}{\partial \beta_i}\left(\frac{1}{1 + e^{-x_i(\omega)\beta_i} e^{-\sum_{k \neq i} \beta_k x_k(\omega)}}\right)$$

$$= + \frac{e^{-\sum_{k \neq i} \beta_k x_k(\omega)} \; e^{-x_i(\omega)\beta_i} \; x_i(\omega)}{\left(1 + e^{-(\beta_0 + \sum \beta_k x_k(\omega))}\right)^2}$$

$$\frac{\partial LL}{\partial \beta_i} = \sum_j y(\omega_j) \frac{x_i(\omega) e^{-(\beta_0 + \sum \beta_k x_k(\omega_j))}}{1 + e^{-(\beta_0 + \sum \beta_k x_k(\omega_j))}} - (1 - y(\omega_j)) \times \frac{x_i(\omega_j)}{1 + e^{-(\beta_0 + \sum \beta_k x_k(\omega_j))}}$$

$$\frac{\partial LL}{\partial \beta_i} = \sum_j \frac{y(\omega_j) x_i(\omega_j)}{1 + e^{-(\beta_0 + \sum \beta_k x_k(\omega_j))}} \left(e^{-(\beta_0 + \sum \beta_k x_k(\omega_j))} + 1\right)$$

$$- x_i(\omega_j) \pi(\omega_j)$$

$$= \sum_j x_i(\omega_j) \left(y(\omega_j) - \pi(\omega_j)\right)$$

$= 1$ pour $i = 0$

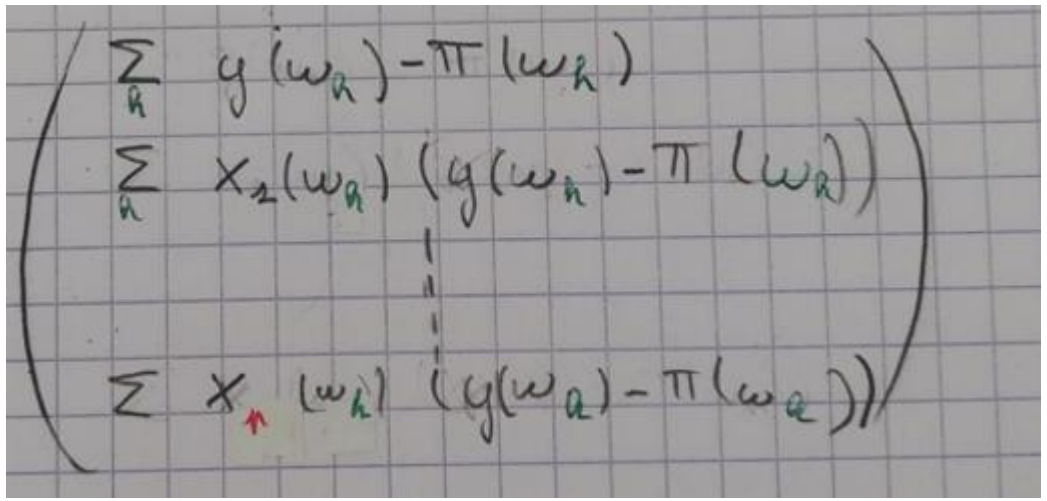
Matrice gradient

$$X = \begin{pmatrix} 1 & x_1(\omega_1) & \cdots & x_m(\omega_1) \\ \vdots & \vdots & & \vdots \\ 1 & x_1(\omega_2) & \cdots & x_m(\omega_2) \end{pmatrix}$$

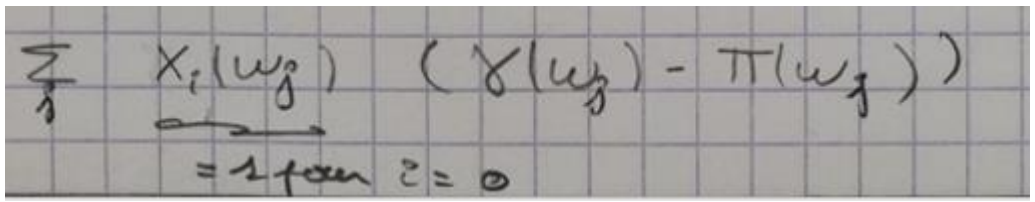
$$X^T = \begin{pmatrix} 1 & \cdots & 1 \\ x_1(\omega_1) & \cdots & x_1(\omega_2) \\ \vdots & & \vdots \\ x_m(\omega_1) & \cdots & x_m(\omega_2) \end{pmatrix}$$

$$X^T \begin{pmatrix} \gamma(\omega_1) - \pi(\omega_1) \\ \vdots \\ \gamma(\omega_2) - \pi(\omega_2) \end{pmatrix} = \begin{pmatrix} \sum_n y(\omega_n) - \pi(\omega_n) \\ \sum_n x_1(\omega_n)\left(y(\omega_n) - \pi(\omega_n)\right) \\ \vdots \\ \sum x_m(\omega_n)\left(y(\omega_n) - \pi(\omega_n)\right) \end{pmatrix}$$

Matrice gradient



=



Matrice gradient

$\boldsymbol{LL0 = ?}$

Objectif : ***Calculer LL0 et donc trouver π***

$$ln\left(\frac{\pi(\omega)}{1-\pi(\omega)}\right) = \beta_0$$

$$Or : \frac{\pi}{1-\pi} = \frac{P(Y=1 \mid X)}{P(Y=0 \mid X)} = \frac{P(Y=1)}{P(Y=0)} = \frac{n_+}{n_-}$$

$$ln\left(\frac{n_+}{n_-}\right) = \beta_0$$

$$\Rightarrow \pi = \frac{1}{1+e^{-\beta_0}} = \frac{1}{1+\frac{n_-}{n_+}} = \frac{n_+}{n} = p_+$$

$\boldsymbol{LL0} = ?$

$$LL0 = \sum_{\omega \in \Omega} y_p(\omega) * \ln\big(\pi(\omega)\big) + \Big(1 - y_p(\omega)\Big) * \ln(1 - \pi(\omega))$$

$$LL0 = \sum_{\omega \in \Omega} y(\omega) * \ln(p_+) + \big(1 - y(\omega)\big) * \ln(1 - p_+)$$

$$LL0 = n_+ * \ln(p_+) + n_- * \ln(1 - p_+)$$

$$LL0 = n * \ln(1 - p_+) + n_+ * \ln\left(\frac{p_+}{1 - p_+}\right)$$