

TP : Régression Logistique - Analyse Mathématique Complète

Groupe d'Optimisation INF4127

17 janvier 2026

Table des matières

1	Introduction	3
2	Partie 1 : Génération des Données	3
2.1	Code	3
2.2	Analyse Mathématique	3
2.3	Explication Littéraire	4
3	Partie 2 : Standardisation des Données	4
3.1	Code	4
3.2	Analyse Mathématique	4
3.3	Explication Littéraire	4
4	Partie 3 : Modèle de Régression Logistique	5
4.1	Code	5
4.2	Analyse Mathématique	5
4.2.1	Fonction Sigmoïde	5
4.2.2	Fonction de Coût (Log Loss)	5
4.3	Explication Littéraire	6
5	Partie 4 : Pénalités L1 et L2	6
5.1	Code	6
5.2	Analyse Mathématique	6
5.2.1	Régularisation L2 (Ridge)	6
5.2.2	Régularisation L1 (Lasso)	6
5.2.3	Solution Analytique des Effets	7
5.3	Explication Littéraire	7
5.3.1	L2 - Le Professeur Équilibré	7
5.3.2	L1 - Le Sélecteur Impitoyable	7
5.3.3	Paramètre C - Le Niveau de Confiance	7

6 Partie 5 : Solveurs d'Optimisation	8
6.1 Analyse Mathématique des Solveurs	8
6.1.1 L-BFGS (Limited-memory BFGS)	8
6.1.2 LIBLINEAR (Coordinate Descent)	8
6.1.3 Newton-CG	8
6.1.4 SAGA (Stochastic Average Gradient Accelerated)	8
6.2 Explication Littéraire des Solveurs	9
6.2.1 L-BFGS - Le Randonneur Intelligent	9
6.2.2 LIBLINEAR - Le Réparateur Méticuleux	9
6.2.3 Newton-CG - L'Ingénieur de Précision	9
6.2.4 SAGA - Le Chef Rapide	9
7 Partie 6 : Évaluation des Performances	9
7.1 Code	9
7.2 Analyse Mathématique	10
7.2.1 Accuracy	10
7.2.2 Log Loss	10
7.2.3 Courbe ROC	10
7.3 Explication Littéraire	10
7.3.1 Accuracy - Le Compteur Simple	10
7.3.2 Log Loss - Le Juge Sévère	10
7.3.3 Courbe ROC - Le Compromis	11
8 Partie 7 : Comparaison des Solveurs - Résultats	11
8.1 Analyse Mathématique des Comparaisons	11
8.1.1 Temps de Calcul	11
8.1.2 Impact de max_iter	11
8.2 Recommandations Mathématiques	11
9 Conclusion	11

1 Introduction

Ce document explique mathématiquement et littérairement chaque portion du programme de comparaison des solveurs en régression logistique. L'objectif est de comprendre non seulement *comment* fonctionne le code, mais aussi *pourquoi* il fonctionne ainsi.

2 Partie 1 : Génération des Données

2.1 Code

```
x_synth, y_synth = make_classification(  
    n_samples=1000,  
    n_features=20,  
    n_informative=15,  
    n_redundant=5,  
    n_classes=2,  
    random_state=42,  
    class_sep=0.8  
)
```

2.2 Analyse Mathématique

La fonction génère un problème de classification binaire avec :

- $X \in \mathbb{R}^{1000 \times 20}$: Matrice de données avec $n = 1000$ échantillons et $p = 20$ caractéristiques
- $y \in \{0, 1\}^{1000}$: Vecteur des labels

Le processus de génération suit :

$$y_i = \mathbb{I}\{w^T x_i + \epsilon > 0\}$$

où :

- $w \in \mathbb{R}^{20}$: Vecteur de poids (seuls les 15 premiers sont non-nuls)
- $x_i \sim \mathcal{N}(\mu_{y_i}, \Sigma)$: Vecteur de features
- $\epsilon \sim \mathcal{N}(0, \sigma^2)$: Bruit gaussien

Les caractéristiques informatives suivent :

$$x_{ij} \sim \begin{cases} \mathcal{N}(\mu_1, \sigma^2) & \text{si } y_i = 1 \\ \mathcal{N}(\mu_0, \sigma^2) & \text{si } y_i = 0 \end{cases}$$

avec $\|\mu_1 - \mu_0\| = \text{class_sep}$.

Les caractéristiques redondantes sont des combinaisons linéaires :

$$x_{ij} = \sum_{k=1}^{15} \alpha_k x_{ik} + \eta, \quad j = 16, \dots, 20$$

2.3 Explication Littéraire

Imaginez que vous êtes un conseiller d'orientation qui doit prédire si 1000 étudiants réussiront leur année. Vous disposez de 20 informations par étudiant :

- 15 informations sont **pertinentes** : notes, assiduité, etc.
- 5 informations sont **redondantes** : par exemple, "moyenne générale" et "moyenne trimestrielle" donnent la même information
- `class_sep=0.8` : les étudiants qui réussissent et ceux qui échouent sont assez différents (écart de 0.8 écart-type)

Vous créez ce jeu de données artificiel pour pouvoir **tester** différents modèles dans des conditions contrôlées.

3 Partie 2 : Standardisation des Données

3.1 Code

```
scaler = StandardScaler()  
X_train_scaled = scaler.fit_transform(X_train)
```

3.2 Analyse Mathématique

La standardisation transforme chaque feature pour qu'elle ait moyenne nulle et variance unitaire :

$$x'_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

où :

$$\mu_j = \frac{1}{n} \sum_{i=1}^n x_{ij}, \quad \sigma_j^2 = \frac{1}{n} \sum_{i=1}^n (x_{ij} - \mu_j)^2$$

Cette transformation est cruciale pour :

1. Éviter que les features avec de grandes échelles ne dominent l'optimisation
2. Assurer la convergence rapide des algorithmes de descente de gradient
3. Rendre la régularisation équitable entre toutes les features

3.3 Explication Littéraire

Supposons que parmi vos 20 caractéristiques d'étudiants, vous ayez :

- Le salaire des parents (en euros, de 0 à 100000)
- L'âge (en années, de 18 à 25)
- Le nombre d'heures d'étude par semaine (de 0 à 40)

Si vous utilisez ces données brutes, le salaire (valeurs autour de 30000) influencera le modèle **1000 fois plus** que l'âge (valeurs autour de 21). C'est comme comparer des kilomètres et des millimètres sans conversion !

La standardisation ramène tout à une échelle commune : chaque caractéristique est mesurée en "**nombre d'écart-types par rapport à la moyenne**". Ainsi :

- Un étudiant dont les parents gagnent 60000€ (si la moyenne est 30000 et écart-type 15000) aura une valeur de 2.0
 - Un étudiant de 24 ans (si la moyenne est 21 et écart-type 2) aura une valeur de 1.5
- Maintenant, toutes les caractéristiques ont le même "poids" potentiel dans le modèle.

4 Partie 3 : Modèle de Régression Logistique

4.1 Code

```
model = LogisticRegression(
    solver='lbfgs',
    penalty='l2',
    C=1.0,
    max_iter=1000
)
```

4.2 Analyse Mathématique

Le modèle cherche les paramètres $w \in \mathbb{R}^{20}$ (poids) et $b \in \mathbb{R}$ (biais) qui minimisent la fonction de coût :

4.2.1 Fonction Sigmoïde

$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z}$$

Cette fonction transforme un score linéaire $z = w^T x + b$ en une probabilité $P(y = 1|x) \in [0, 1]$.

Propriétés importantes :

$$\begin{aligned}\sigma(0) &= 0.5 \\ \sigma'(z) &= \sigma(z)(1 - \sigma(z)) \\ \sigma(-z) &= 1 - \sigma(z)\end{aligned}$$

4.2.2 Fonction de Coût (Log Loss)

Pour un échantillon (x_i, y_i) :

$$\ell_i(w, b) = -[y_i \log(\sigma(w^T x_i + b)) + (1 - y_i) \log(1 - \sigma(w^T x_i + b))]$$

Pour n échantillons avec régularisation L2 :

$$J(w, b) = \frac{1}{n} \sum_{i=1}^n \ell_i(w, b) + \frac{\lambda}{2} \|w\|^2$$

où $\lambda = \frac{1}{C}$ et $\|w\|^2 = \sum_{j=1}^{20} w_j^2$.

4.3 Explication Littéraire

Imaginez que vous êtes un professeur qui doit tracer une ligne sur le tableau pour séparer :

- À gauche : les étudiants qui risquent d'échouer

- À droite : les étudiants qui devraient réussir

Cette "ligne" (en réalité un hyperplan en 20 dimensions) est définie par :

$$w_1 \times \text{salaire} + w_2 \times \text{âge} + \cdots + w_{20} \times \text{dernière_caractéristique} + b = 0$$

La sigmoïde vous dit : "si un étudiant est très à droite de la ligne, il a 99% de chances de réussir ; s'il est très à gauche, 1% ; s'il est sur la ligne, 50%" .

Le modèle ajuste w et b pour que :

1. Les étudiants qui ont vraiment réussi soient à droite avec forte probabilité
2. Les étudiants qui ont échoué soient à gauche avec forte probabilité
3. La ligne ne soit pas "trop tordue" (régularisation L2)

5 Partie 4 : Pénalités L1 et L2

5.1 Code

```
penalty='l2' # ou 'l1'
```

5.2 Analyse Mathématique

5.2.1 Régularisation L2 (Ridge)

$$J_{L2}(w, b) = \frac{1}{n} \sum_{i=1}^n \ell_i(w, b) + \frac{\lambda}{2} \|w\|^2$$

où $\|w\|^2 = \sum_{j=1}^p w_j^2$ (norme euclidienne au carré).

L'effet sur le gradient :

$$\frac{\partial J_{L2}}{\partial w_j} = \frac{\partial \text{LogLoss}}{\partial w_j} + \lambda w_j$$

5.2.2 Régularisation L1 (Lasso)

$$J_{L1}(w, b) = \frac{1}{n} \sum_{i=1}^n \ell_i(w, b) + \lambda \|w\|_1$$

où $\|w\|_1 = \sum_{j=1}^p |w_j|$ (norme L1).

L'effet sur le gradient (dans le sens sous-différentiel) :

$$\frac{\partial J_{L1}}{\partial w_j} \in \frac{\partial \text{LogLoss}}{\partial w_j} + \lambda \cdot \text{sign}(w_j)$$

5.2.3 Solution Analytique des Effets

Pour un problème quadratique simple $\min_w \frac{1}{2}(w - \hat{w})^2 + \lambda|w|$:

$$w^* = \begin{cases} \hat{w} - \lambda & \text{si } \hat{w} > \lambda \\ \hat{w} + \lambda & \text{si } \hat{w} < -\lambda \\ 0 & \text{si } |\hat{w}| \leq \lambda \end{cases}$$

C'est l'effet de seuillage de L1.

5.3 Explication Littéraire

5.3.1 L2 - Le Professeur Équilibré

Imaginez des ressorts attachés à chaque coefficient w_j :

- Chaque ressort tire w_j vers 0 avec une force λw_j
- Plus w_j est grand, plus le ressort tire fort
- Résultat : tous les coefficients sont réduits proportionnellement, mais rarement mis exactement à 0

Analogie : C'est comme dire à un étudiant : "Sois raisonnable dans toutes les matières" plutôt que "Excelle en maths mais néglige le français".

5.3.2 L1 - Le Sélecteur Impitoyable

Imaginez plutôt un seuil avec un "coût d'entrée" :

- Pour qu'une caractéristique ait un coefficient non-nul, elle doit prouver qu'elle apporte au moins λ d'utilité
- Sinon, son coefficient est mis exactement à 0
- Résultat : sélection automatique de caractéristiques

Analogie : C'est comme faire ses valises avec un poids limite strict. Vous gardez seulement les vêtements les plus utiles et jetez les autres.

5.3.3 Paramètre C - Le Niveau de Confiance

$$C = \frac{1}{\lambda}$$

- C grand (λ petit) : "Je fais confiance à mes données, le modèle peut être complexe"
- C petit (λ grand) : "Je me méfie du bruit, gardons le modèle simple"

Échelle typique :

$$C \in [10^{-3}, 10^3]$$

6 Partie 5 : Solveurs d'Optimisation

6.1 Analyse Mathématique des Solveurs

6.1.1 L-BFGS (Limited-memory BFGS)

Algorithme quasi-Newton qui approxime la matrice Hessienne $H_k \approx \nabla^2 J(w_k)$.
Mise à jour BFGS :

$$H_{k+1} = H_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{H_k s_k s_k^T H_k}{s_k^T H_k s_k}$$

où $s_k = w_{k+1} - w_k$, $y_k = \nabla J(w_{k+1}) - \nabla J(w_k)$.

Version mémoire limitée : ne stocke que les m derniers (s_k, y_k) .

6.1.2 LIBLINEAR (Coordinate Descent)

Optimise une coordonnée à la fois. Pour la régression logistique avec L2 :

$$w_j^{(k+1)} = \frac{\sum_{i=1}^n x_{ij}(y_i - p_i^{(k)})}{\sum_{i=1}^n x_{ij}^2 + \lambda}$$

où $p_i^{(k)} = \sigma(w^{(k)T} x_i + b)$.

6.1.3 Newton-CG

Résout $H_k d_k = -\nabla J(w_k)$ avec gradient conjugué :

1. $r_0 = -\nabla J(w_k)$, $p_0 = r_0$
2. Pour $i = 0, 1, \dots$:

$$\alpha_i = \frac{r_i^T r_i}{p_i^T H_k p_i}, \quad w_{k,i+1} = w_{k,i} + \alpha_i p_i, \quad r_{i+1} = r_i - \alpha_i H_k p_i$$

ération jusqu'à convergence

6.1.4 SAGA (Stochastic Average Gradient Accelerated)

Version stochastique avec variance réduite :

$$w_{k+1} = w_k - \alpha \left(\nabla f_{i_k}(w_k) - g_{i_k}^{(k)} + \frac{1}{n} \sum_{j=1}^n g_j^{(k)} \right)$$

où $g_j^{(k)}$ est le dernier gradient calculé pour l'échantillon j .

6.2 Explication Littéraire des Solveurs

6.2.1 L-BFGS - Le Randonneur Intelligent

Imaginez gravir une montagne dans le brouillard :

- Vous sentez la pente sous vos pieds (gradient)
- Vous mémorisez : "la dernière fois, après 10 pas vers le nord, la pente a changé de X"
- Vous devinez la courbure de la montagne sans la voir
- Vous faites des pas adaptés à cette courbure devinée

Avantage : Intelligent mais peu gourmand en mémoire.

6.2.2 LIBLINEAR - Le Réparateur Méticuleux

Imaginez régler une vieille radio avec 20 boutons :

- Vous fixez 19 boutons, ajustez le 1er jusqu'à l'optimum
- Vous fixez 18 boutons + le nouveau 1er, ajustez le 2nd
- Vous répétez pour tous les boutons
- Vous recommencez jusqu'à satisfaction

Avantage : Simple, garanti de converger pour L1/L2.

6.2.3 Newton-CG - L'Ingénieur de Précision

Imaginez construire un pont :

- Vous calculez exactement les forces (Hessienne)
- Vous résolvez précisément les équations d'équilibre
- Vous ajustez chaque poutre de la quantité exacte nécessaire
- Très précis, mais coûteux en calculs

6.2.4 SAGA - Le Chef Rapide

Imaginez goûter une soupe pour l'assaisonner :

- Au lieu de goûter toute la soupe (gradient complet)
- Vous goûtez une cuillère (un échantillon)
- Vous vous souvenez : "la dernière fois, cette cuillère était trop salée"
- Vous ajustez en conséquence, en moyennant mentalement

Avantage : Rapide pour les très grands datasets.

7 Partie 6 : Évaluation des Performances

7.1 Code

```
accuracy = accuracy_score(y_test, y_pred)
log_loss_val = log_loss(y_test, y_pred_proba)
```

7.2 Analyse Mathématique

7.2.1 Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{1}{n} \sum_{i=1}^n \mathbb{I}\{\hat{y}_i = y_i\}$$

7.2.2 Log Loss

$$\text{LogLoss} = -\frac{1}{n} \sum_{i=1}^n [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

Propriété : si $y_i = 1$ et $p_i = 0.99$, contribution $= -\log(0.99) \approx 0.01$; si $p_i = 0.01$, contribution $= -\log(0.01) \approx 4.61$.

7.2.3 Courbe ROC

Pour chaque seuil $t \in [0, 1]$:

$$\text{TPR}(t) = \frac{|\{i : p_i \geq t \text{ et } y_i = 1\}|}{|\{i : y_i = 1\}|}$$

$$\text{FPR}(t) = \frac{|\{i : p_i \geq t \text{ et } y_i = 0\}|}{|\{i : y_i = 0\}|}$$

$$\text{AUC} = \int_0^1 \text{TPR}(\text{FPR}^{-1}(t)) dt$$

7.3 Explication Littéraire

7.3.1 Accuracy - Le Compteur Simple

”Sur 100 étudiants, combien ai-je bien classé?”

- Avantage : simple à comprendre
- Inconvénient : ne tient pas compte des déséquilibres de classe
- Exemple : si 95% réussissent, prédire ”tous réussissent” donne 95% d’accuracy

7.3.2 Log Loss - Le Juge Sévère

”Quand vous dites ’90% de chance’, l’étudiant doit vraiment réussir!”

- Pénalise sévèrement les certitudes erronées
- Récompense les probabilités bien calibrées
- Meilleure pour comparer des modèles probabilistes

Analogie :

- Accuracy : Nombre de paniers marqués
- Log Loss : Qualité de chaque tir (un tir à 90% qui rate est très pénalisé)

7.3.3 Courbe ROC - Le Compromis

”Si j’accepte plus de faux positives (étiqueter ‘réussite’ à tort), combien de vrais positifs supplémentaires puis-je capturer ?”

- AUC = 1.0 : classificateur parfait
- AUC = 0.5 : aussi bon qu’une pièce de monnaie
- AUC = 0.8 : bon classificateur

8 Partie 7 : Comparaison des Solveurs - Résultats

8.1 Analyse Mathématique des Comparaisons

8.1.1 Temps de Calcul

Complexité théorique pour n échantillons, p caractéristiques, m classes :

- L-BFGS : $O(np \cdot \text{iter})$ + mémoire $O(mp)$
- LIBLINEAR : $O(np \cdot \text{iter})$ pour L2
- Newton-CG : $O(np^2)$ par itération
- SAGA : $O(p \cdot \text{iter})$, constant par itération

8.1.2 Impact de max_iter

Analyse de convergence :

$$\|w_k - w^*\| \leq \begin{cases} \rho^k \|w_0 - w^*\| & \text{L-BFGS (linéaire)} \\ \rho^{2^k} \|w_0 - w^*\| & \text{Newton (quadratique)} \\ \frac{C}{k} & \text{SGD (sous-linéaire)} \end{cases}$$

8.2 Recommandations Mathématiques

Solveur	Quand l'utiliser	Complexité
L-BFGS	Dataset moyen, multiclass, L2	$O(np \cdot \text{iter})$
LIBLINEAR	Petit dataset, besoin de L1	$O(np \cdot \text{iter})$
Newton-CG	p petit, précision requise	$O(np^2 \cdot \text{iter})$
SAGA	n très grand, L1/L2	$O(p \cdot \text{iter})$
SAG	n très grand, seulement L2	$O(p \cdot \text{iter})$

TABLE 1 – Guide de sélection des solveurs

9 Conclusion

La régression logistique est plus qu’un simple classificateur : c’est un cadre mathématique élégant qui combine :

1. Une modélisation probabiliste via la sigmoïde

2. Une optimisation convexe garantie
3. Des techniques de régularisation théoriquement fondées
4. Des algorithmes d'optimisation adaptés à différentes situations

Le choix du solveur n'est pas anodin : il affecte :

- La vitesse de convergence
- La mémoire utilisée
- La capacité à gérer de grandes dimensions
- Le support des différentes régularisations

Ce TP montre l'importance de comprendre la théorie sous-jacente pour faire des choix éclairés en pratique. La beauté de la régression logistique réside dans ce mariage entre élégance mathématique et utilité pratique.