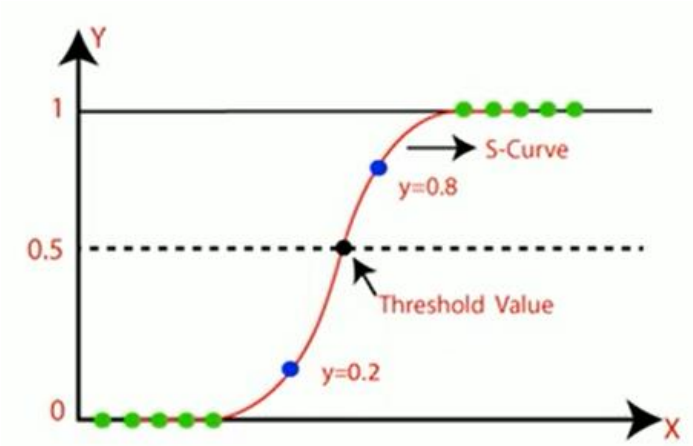
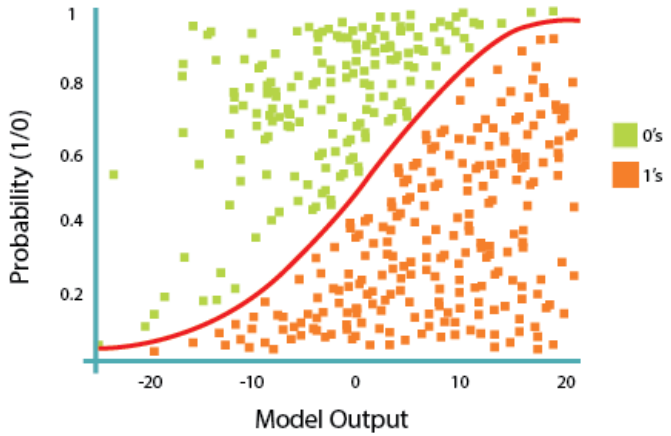
Ch5. Apprentissage Supervisé – Classification

- Régression logistique
  - Initiation et introduction à la classification
- Fonction coût et optimisation
  - Entraîner le Modèle de Régression logistique
- Implémentation et évaluation
  - Mettre en pratique la régression logistique



Les taches d'apprentissage supervisé



Classification Data

X_1	X_2	X_3	X_p	Y
				cat
				dog
				cat
				cat

Categorical "Labels"

APPRENTISSAGE SUPERVISÉ

Une approche fondamentale du Machine Learning permettant à une machine d'apprendre à partir d'exemples **étiquetés** pour faire des prédictions.

> **Principe clé :**
> Les données d'entraînement contiennent des paires (X, y) où X sont les caractéristiques et y est la variable cible.

OBJECTIF :

Apprendre une fonction f(X) qui prédit y avec la plus grande précision possible.

CLASSIFICATION :

Prédire une catégorie discrète (ex: spam/non-spam, chat/chien, Diabète ou non).

L'objectif est de catégoriser de nouvelles observations dans des classes prédéfinies, basée sur des caractéristiques d'entrée.

➔ C'est le domaine d'application de la *régression logistique*.

Les 4 notions fondamentales

Dataset

Ensemble d'exemples étiquetés composé de variables cibles (y) et de caractéristiques (X)

Pour la classification, y est une variable catégorielle (ex: 0 ou 1, ou plusieurs classes).

Modèle

Fonction mathématique avec des paramètres qui associe les caractéristiques aux **probabilités** d'appartenance à une classe.

Pour la classification, le modèle produit des sorties entre 0 et 1.

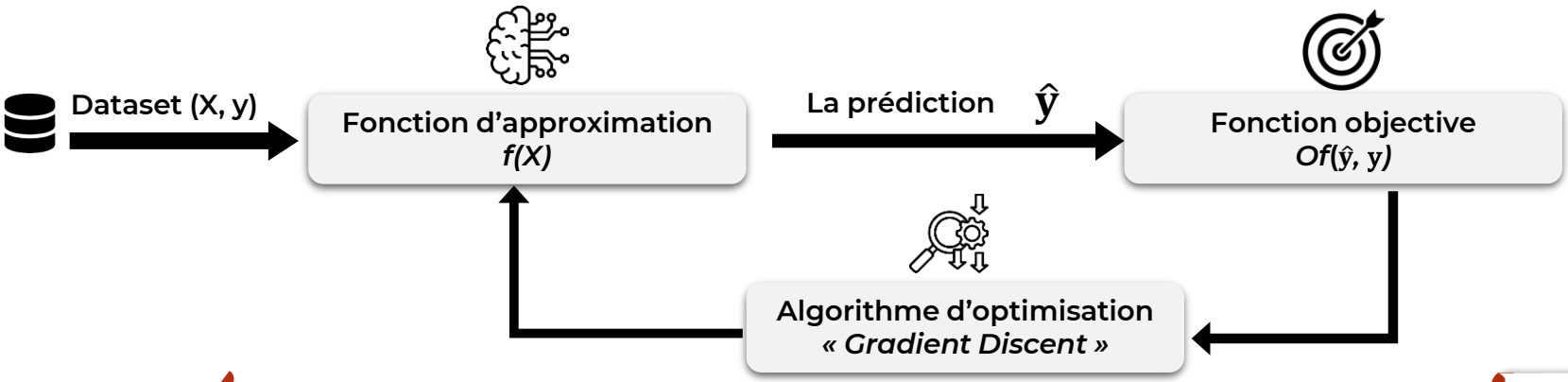
Fonction Coût

Mesure l'écart entre les probabilités prédites par le modèle et les vraies étiquettes de classe.

Pour la classification, on utilise généralement l'entropie croisée (Log Loss).

Algorithme de Minimisation

Stratégie pour trouver les paramètres optimaux qui minimisent la fonction coût



Limitations de la Régression Linéaire

SORTIES ET INTERPRÉTATION

Sorties Non Bornées

La régression linéaire produit des sorties qui peuvent être **n'importe quel nombre reel** , positif ou négatif, sans limite supérieure ou inférieure.

Exemple :

- Pour un problème de classification binaire où $y \in \{0, 1\}$, la régression linéaire peut prédire $y = -0.5$ ou $y = 2.3$, ce qui n'a aucun sens comme probabilité.
- Ces valeurs sortent complètement de l'intervalle **[0, 1]** requis pour une classification binaire.

Cette absence de contrainte sur les sorties rend impossible l'interprétation des résultats comme des probabilités valides.

Absence de Garantie Probabiliste

Les sorties de la régression linéaire **ne peuvent pas être interprétées comme des probabilités** puisqu'elles ne sont pas contraintes à l'intervalle [0, 1].

Problème :

- Une probabilité doit toujours être entre 0 et 1. Si la régression linéaire prédit 1.5 ou -0.2, comment interpréter ces valeurs ? Elles ne représentent rien en termes de probabilité d'appartenance à une classe.

Nous avons besoin d'une **fonction de transformation** qui convertit n'importe quelle entrée réelle en une valeur entre 0 et 1, garantissant une interprétation probabiliste valide.

Ces deux premiers problèmes révèlent l'inadéquation fondamentale de la régression linéaire pour la classification. Mais il y a encore d'autres limitations liées à l'optimisation et à la prise de décision que nous explorerons dans la suite.

Limitations de la Régression Linéaire

OPTIMISATION ET DÉCISION

Fonction Coût Non Convexe

Si on applique l'erreur quadratique moyenne (MSE) avec la régression linéaire à un problème de classification, la fonction coût résultante est **non convexe**

Conséquence :

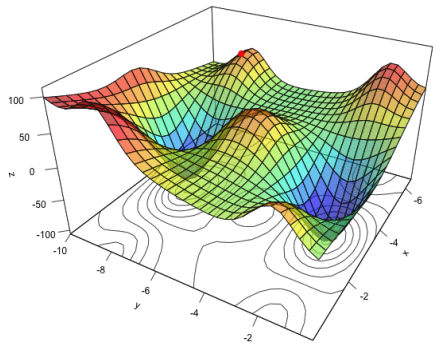
- L'algorithme d'optimisation (descente de gradient) peut se bloquer dans des minimums locaux au lieu de trouver le minimum global, ce qui rend l'apprentissage inefficace.

Seuil de Décision Arbitraire

Appliquer un seuil fixe (ex: 0.5) sur une sortie linéaire pour la classification est **arbitraire et non justifié** mathématiquement.

Problème :

- Pourquoi 0.5 et pas 0.3 ou 0.7 ? La régression linéaire ne fournit aucune justification théorique pour ce choix.



La régression linéaire n'est pas adaptée à la classification. Nous avons besoin d'un modèle qui produit des **probabilités bornées entre 0 et 1** avec une **fonction coût convexe**. La **régression logistique** résout tous ces problèmes.

Introduction à la régression logistique

Définition

La régression logistique est un modèle de **classification supervise** qui utilise une **fonction logistique** (ou sigmoïde) pour modéliser la probabilité qu'une observation appartienne à une classe particulière.

⚠ Malgré son nom contenant le mot « régression », la régression logistique est un algorithme de classification, non de régression

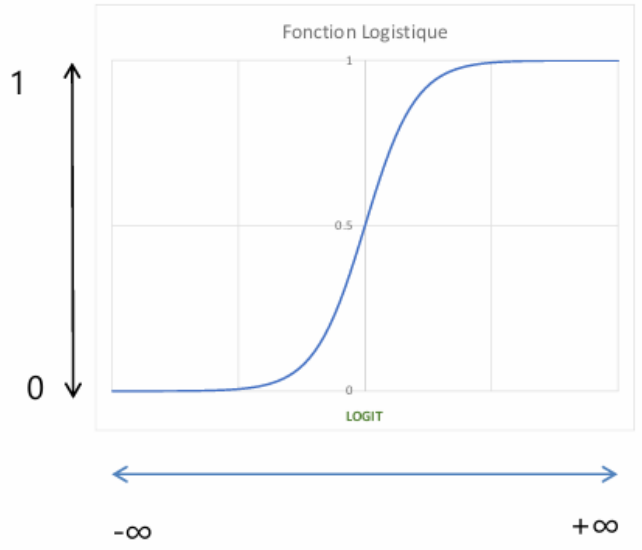
Caractéristiques Clés

- **Sortie Probabiliste :**
  Elle produit une sortie entre 0 et 1, qui peut être interprétée comme une probabilité d'appartenance à la classe positive.
- **Seuil de Décision :**
  Un seuil (généralement 0.5) est appliqué à cette probabilité pour assigner l'observation à une classe (0 ou 1).
- **Interprétabilité :**
  Les paramètres du modèle ont une interprétation probabiliste directe, ce qui rend le modèle très interprétable.
- **Simplicité et Efficacité :**
  Malgré sa simplicité, elle est très efficace pour les problèmes de classification binaire et peut être étendue à la classification multi-classe

La fonction Sigmoïde (Logistique)

Formule Mathématique

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



Rôle et Interprétation

- **Transformation :** La fonction sigmoïde transforme une entrée z (qui peut être n'importe quel nombre réel) en une valeur comprise entre 0 et 1

- **Probabilité :** La sortie de la sigmoïde représente la probabilité que l'observation appartienne à la classe positive (y=1).

- **Monotone :** La fonction est strictement croissante, garantissant une relation cohérente entre l'entrée et la probabilité.

$\sigma(0) = 0.5$ (point d'inflexion)

$z \to +\infty \Longrightarrow \sigma(z) \to 1$

$z \to -\infty \Longrightarrow \sigma(z) \to 0$

Forme : Courbe en 'S' (sigmoïde)

Le Modèle de Régression Logistique

PARTIE 1 : COMBINAISON LINÉAIRE DES CARACTÉRISTIQUES

La régression logistique combine une **combinaison linéaire** des caractéristiques d'entrée (X) avec une **fonction de transformation** pour produire une probabilité de classe compris entre 0 et 1.

1. COMBINAISON LINÉAIRE

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

↳ Sous forme vectorielle :

$$z = X\theta$$

- z : La combinaison linéaire (peut être n'importe quel nombre réel)
- θ_0 : Terme de biais (intercept), l'ordonnée à l'origine
- $\theta_1, \theta_2, \ldots, \theta_n$: Poids (coefficients) associés à chaque caractéristique
- $x_1, x_2, \ldots, x_n$: Caractéristiques d'entrée (features) du modèle

Cette combinaison linéaire z est la première étape du modèle. Elle capture la relation linéaire entre les caractéristiques d'entrée et la variable cible

Les valeurs de θ_0, θ_1, θ_2, ..., θ_n sont initialement inconnues et doivent être apprises à partir des données d'entraînement. C'est l'objectif de l'algorithme d'optimisation.

➔ **Prochaine étape** Cette combinaison linéaire z sera ensuite transformée par la fonction sigmoïde pour produire une probabilité entre 0 et 1.

Le Modèle de Régression Logistique

FORMULATION COMPLÈTE

2. Application de la Fonction Sigmoïde

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

▼ La fonction sigmoïde transforme la combinaison linéaire z (définie dans la 1er partie) en une valeur comprise entre 0 et 1, représentant une probabilité.

3. Modèle Complet de Régression Logistique

$$P(y = 1 \mid X) = h_\theta(X) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)}}$$

▼ Le modèle complet combine la combinaison linéaire des caractéristiques avec la fonction sigmoïde pour produire une probabilité d'appartenance à la classe positive.

Probabilité de la Classe Positive

$$P(y = 1 \mid X) = h_\theta(X)$$

Probabilité de la Classe Négative

$$P(y = 0 \mid X) = 1 - h_\theta(X)$$

OBJECTIF D'APPRENTISSAGE

Apprendre les paramètres θ qui **maximisent la probabilité** de prédire correctement les classes. Cela revient à **minimiser la fonction coût** d'entropie croisée

Fonction Coût et Optimisation

ENTRAÎNER LE MODÈLE DE RÉGRESSION LOGISTIQUE

Nous avons maintenant compris les fondements théoriques de la régression logistique et comment elle résout les limitations de la régression linéaire pour la classification. Cependant, pour que le modèle apprenne réellement à partir des données, nous avons besoin de deux éléments essentiels :

① Une Fonction Coût Appropriée

Une fonction qui mesure précisément l'erreur de classification.

➔ Pour la régression logistique, nous utilisons l'entropie croisée (Log Loss), qui pénalise les mauvaises prédictions de probabilités et garantit une optimisation convexe.

② Un Algorithme d'Optimisation

Un algorithme qui ajuste les paramètres du modèle pour minimiser la fonction coût.

➔ La descente de gradient est l'algorithme standard qui itérativement améliore les paramètres jusqu'à la convergence.

Comment l'entropie croisée et la descente de gradient travaillent ensemble pour entraîner un modèle de classification ?

➔ Nous explorerons dans les slides suivants

Fonction Coût et Optimisation

Le Défi de la Fonction Coût en Régression Logistique

Pourquoi l'Erreur Quadratique Moyenne (MSE) Ne Convient Pas?

✗ Problème Principal

Si on applique l'erreur quadratique moyenne (MSE) avec la régression logistique, la fonction coût résultante est **non convexe**. Cela signifie qu'elle a plusieurs minimums locaux.

⚠ Conséquences

L'algorithme d'optimisation (descente de gradient) peut se bloquer dans un minimum local au lieu de trouver le minimum global. Cela rend l'apprentissage inefficace et imprévisible.

La Nécessité d'une Fonction Coût Convexe

✓ Propriété Requise

Nous avons besoin d'une fonction coût **convexe** qui garantit que la descente de gradient converge vers le minimum global, quel que soit le point de départ.

✓ Interprétation Probabiliste

La régression logistique modélise des probabilités, ce qui nécessite une fonction coût qui pénalise les prédictions incorrectes de probabilités de manière appropriée.

Qu'est-ce qu'une Fonction Convexe ?

Une fonction est convexe si toute ligne droite reliant deux points de la fonction se situe entièrement au-dessus ou sur la courbe. Pour une fonction convexe, tout minimum local est également un minimum global, ce qui garantit que la descente de gradient trouvera la meilleure solution.

Fonction Coût et Optimisation

INTRODUCTION À L'ENTROPIE CROISÉE (LOG LOSS)

L'entropie croisée, ou **Log Loss**, est la fonction de coût standard pour les problèmes de classification en régression logistique.

➔ Elle mesure la performance d'un modèle de classification dont la sortie est une valeur de probabilité entre 0 et 1.

INTUITION

Comment Fonctionne l'Entropie Croisée ?

Elle pénalise fortement les prédictions qui sont loin de la vérité.

Par exemple, prédire une probabilité de 0.1 quand la vraie classe est 1 est très pénalisé.

À l'inverse, les prédictions correctes de probabilités sont récompensées.

AVANTAGES PAR RAPPORT À MSE

✓ **Convexité :** Garantit une optimisation efficace vers le minimum global

✓ **Interprétation Probabiliste :** Directement liée à la théorie des probabilités

✓ **Pénalité Appropriée :** Pénalise davantage les erreurs de confiance (ex: prédire 0.9 quand c'est 0)

✓ **Convergence Rapide :** L'algorithme d'optimisation converge plus rapidement qu'avec MSE

Fonction Coût et Optimisation

FORMULE DE L'ENTROPIE CROISÉE (LOG LOSS)

$$J(\theta) = \frac{1}{m}\sum\left[-y^{(i)}\log(h_\theta(x^{(i)}) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

- **m :** Nombre d'exemples d'entraînement
- $h_\theta(x)$**:** Probabilité prédite par le modèle
- **y :** Vraie étiquette (0 ou 1)

↳ Si $y^{(i)} = 1$

$$J^{(i)} = -\log(h_\theta(x^{(i)}))$$

↳ Si $y^{(i)} = 0$

$$J^{(i)} = -\log(1-h_\theta(x^{(i)}))$$

Cette formule est compacte : Si $y = 1$, seul le premier terme reste — Si $y = 0$, seul le second terme reste

$$J(\theta) = \frac{1}{m}\sum\left[-y^{(i)}\log(h_\theta(x^{(i)}) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

```
def compute_cost(X, y, theta):
    m = len(y)
    y_pred = model(X, theta)
    cost = - (1/m) * np.sum(y * np.log(y_pred) + (1 - y) * np.log(1 - y_pred))
    return cost
```

Propriété Clé

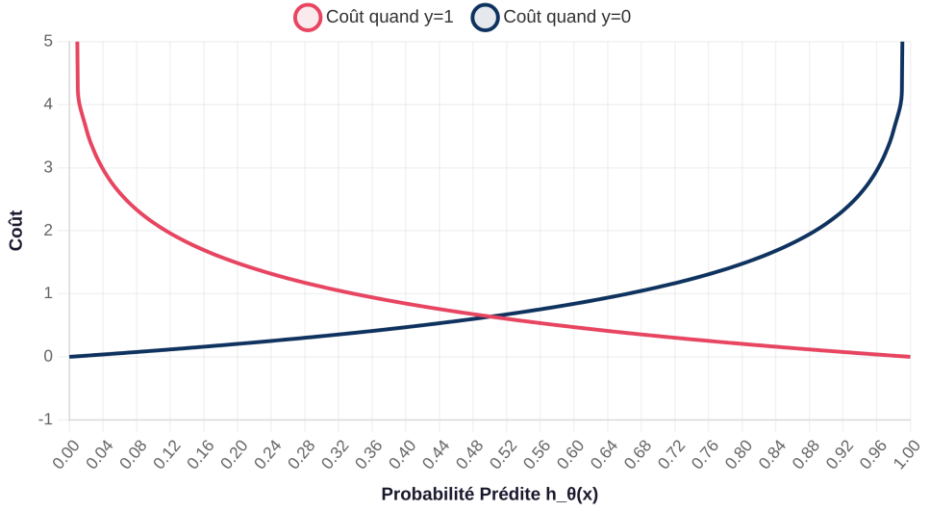
La fonction **log** pénalise exponentiellement les mauvaises prédictions, ce qui rend l'optimisation très efficace.

Minimiser $J(\theta)$ pour trouver les paramètres θ qui réduisent l'erreur de classification

➔ C'est là qu'intervient la **descente de gradient**

Fonction Coût et Optimisation

VISUALISATION DE LA FONCTION DE COÛT



Comportement de y

pour y=1

Quand la vraie classe est 1, le coût est **minimal** (≈ 0) **quand** $h_\theta(x) \approx 1$ et **augmente exponentiellement quand** $h_\theta(x) \approx 0$

pour y=0

Quand la vraie classe est 0, le coût est **minimal** (≈ 0) **quand** $h_\theta(x) \approx 0$ et **augmente exponentiellement quand** $h_\theta(x) \approx 1$

Avantage pour l'Optimisation

Les pentes abruptes près des mauvaises prédictions créent de **gradients forts**, ce qui permet à la descente de gradient de corriger rapidement les erreurs.

Fonction Coût et Optimisation

DESCENTE DE GRADIENT – Rappel

Comme pour la régression linéaire, nous utilisons la descente de gradient pour minimiser la fonction de coût.

La descente de gradient est un algorithme d'optimisation itératif qui ajuste les paramètres du modèle pour **minimiser la fonction coût** . C'est l'algorithme fondamental pour entraîner les modèles de machine learning.

Intuition : Descendre une Montagne

Imaginez que vous êtes au sommet d'une montagne dans le brouillard. Vous ne pouvez pas voir la vallée, mais vous pouvez sentir la pente sous vos pieds. À chaque pas, vous descendez dans la direction de la pente la plus raide. Après plusieurs pas, vous atteindrez le fond de la vallée (le minimum).

Les 4 étapes de la descente de gradient

1. **Initialiser :** Commencer avec des paramètres θ aléatoires
2. **Calculer le Gradient :** Calculer la dérivée de la fonction coût par rapport à θ
3. **Mettre à Jour :** Ajuster θ dans la direction opposée au gradient
4. **Répéter :** Continuer jusqu'à convergence (quand les changements deviennent très petits)

Fonction Coût et Optimisation

DESCENTE DE GRADIENT – Mathématiques

↳ Le dérivé de la fonction coût :

$$\frac{\delta J(\theta)}{\delta \theta_j} = \frac{1}{m}\sum(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Propriété Remarquable

Pour la régression logistique avec entropie croisée, cette formule est identique à celle de la régression linéaire ! Seule la fonction $h_\theta(x)$ change.

↳ Forme vectorielle :

$$\nabla_\theta J(\theta) = \frac{1}{m}X^T(h_\theta(X) - y)$$

↳ La mise à jour des paramètres :

$$\theta_j := \theta_j - \alpha\frac{\delta J(\theta)}{\delta \theta_j}$$

Répéter jusqu'à convergence

- $h_\theta(x^{(i)}) - y^{(i)}$: Erreur de prédiction
- $x_j^{(i)}$: valeur de la caractéristique j
- α : Taux d'apprentissage (controle la taille des pas)

↳ $\nabla_\theta J(\theta) = \frac{1}{m}X^T(h_\theta(X) - y)$

```
def calculer_derive(X, Y, theta):
    m = len(Y)
    predictions = model(X, theta)
    erreur = predictions - Y.values.reshape(-1, 1)
    derive = (1/m) * (X.T.dot(erreur))
    return derive
```

↳ $\theta := \theta - \alpha\frac{\delta J(\theta)}{\delta \theta}$

```
def gradient_descent(X, Y, theta, learning_rate=0.01, n_iterations=1000):
    for i in range(n_iterations):
        derive = calculer_derive(X, Y, theta)
        theta = theta - learning_rate * derive
    return theta
```

Fonction Coût et Optimisation

DESCENTE DE GRADIENT – Mathématiques

↳ Taux d'apprentissage (learning rate) α:

Contrôle la taille des pas que l'algorithme prend à chaque itération. C'est un hyperparamètre crucial qui affecte directement la vitesse et la qualité de la convergence.

❌ **Taux Trop Petit ($\alpha \approx 0.0001$)**

L'algorithme prend des pas très petits.

- **Convergence très lente :** Nécessite beaucoup d'iterations
- **Coûteux en calcul :** Temps d'entraînement très long

❌ **Taux Trop Grand ($\alpha \approx 1.0$)**

L'algorithme prend des pas très grands.

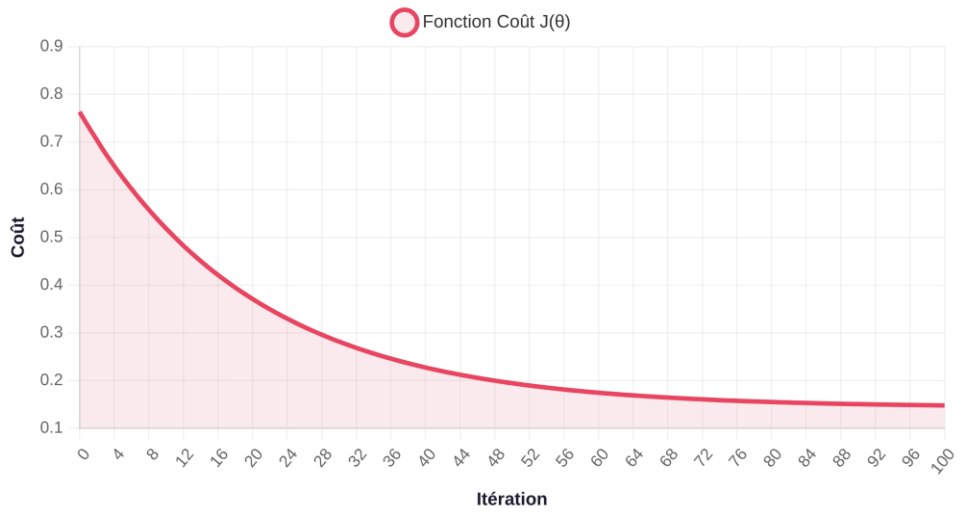
- **Divergence possible :** Peut sauter par-dessus le minimum
- **Instabilité :** Le coût peut augmenter au lieu de diminuer

✓ **Taux Optimal ($\alpha \approx 0.01$ à 0.1)**

Un bon taux d'apprentissage permet une **convergence rapide et stable** . Il faut souvent l'ajuster expérimentalement en observant comment le coût évolue pendant l'entraînement. Des techniques comme la **validation croisée** peuvent aider à trouver la valeur optimale.

Fonction Coût et Optimisation

↳ Fonction coût au cours d'entrainement :



1 Descente Rapide

Au début, le coût diminue **très rapidement** Le gradient est grand, donc les paramètres changent beaucoup à chaque itération.

2 Ralentissement

À mesure que nous nous rapprochons du minimum, la descente **ralentit progressivement** Le gradient diminue.

3 Convergence

Finalement le coût se stabilise, les changements deviennent **négligeables** L'algorithme a convergé

Cette courbe est en forme de **décroissance exponentielle** est typique de la descente de gradient avec une fonction coût convexe. Elle indique que l'optimisation fonctionne correctement

Implémentation et évaluation

Maintenant que nous comprenons comment entraîner un modèle de régression logistique, nous devons apprendre à l'**implémenter en code**, à **évaluer sa performance** et à **interpréter les résultats** dans le contexte des cas d'études différents

Vue d'Ensemble de l'Implémentation

① **Préparation des Données**

Charger, nettoyer et normaliser les données. Diviser en ensembles d'entraînement et de test

② **Entraînement du Modèle**

Initialiser les paramètres et exécuter la descente de gradient pour minimiser la fonction coût

③ **Prédiction**

Utiliser le modèle entraîné pour prédire les probabilités et les classes des nouvelles données.

④ **Évaluation**

Calculer les métriques de performance pour évaluer la qualité des prédictions
(dans notre cas nous allons se concentrer dans un premier temps sur l'accuracy)

Point Important

Ces étapes sont itérative. Si les performances ne sont pas satisfaisantes, on peut revenir à l'étape 1 pour améliorer les données, ou à l'étape 2 pour ajuster les hyperparamètres (taux d'apprentissage, nombre d'itérations, etc.).

Régression Logistique

EN UTILISANT LA BIBLIOTHÈQUE SCKIT-LEARN

Importation	`from sklearn.linear_model import LogisticRegression`
Création du modèle	`model = LogisticRegression()`
Entrainement	`model.fit(X, Y)`
Faire des prédictions	`predictions = model.predict(X)`

Implémentation et évaluation

Cas d'Étude 1 : Diagnostic des maladies cardiaques

🎯 Problème à Résoudre

En utilisant les caractéristiques médicales et cliniques des patients, peut-on prédire la présence d'une maladie cardiaque ?

Classe 0 : Absence de maladie cardiaque

Patients sans pathologie cardiaque identifiée

Classe 1 : Présence de maladie cardiaque

Patients avec diagnostic confirmé de maladie cardiaque.

📊 Caractéristiques (Features)

- X_1 **age :** Âge du patient - Mesuré en années (ex: 40, 49, 37, 48, 54)
- X_2 **Sexe :** Sexe du patient - Variable binaire (0 = femme, 1 = homme)
- X_3 **chest_pain_type :** Type de douleur thoracique - Numérique (1-4) types de douleur angineuse (ex: 1, 2, 3, 4)
- X_4 **resting_bp_s :** Pression artérielle au repos - Mesurée en mm Hg (ex: 140, 160, 130, 138, 150)
- X_5 **cholesterol :** Taux de cholestérol - Mesuré en mg/dl (ex: 289, 180, 283, 214, 195)
- X_6 **fasting_blood_sugar :** Glycémie à jeun - Binaire (1 si > 120 mg/dl, 0 sinon)
- X_7 **resting_ecg :** Résultats de l'électrocardiogramme au repos - Numérique (0, 1, 2)
- X_8 **max_heart_rate :** Fréquence cardiaque maximale atteinte - Battements par minute (ex: 172, 156, 98, 108, 122)
- X_9 **exercise_angina :** Angine induite par l'exercice - Binaire (0 = non, 1 = oui)
- X_{10} **oldpeak :** Dépression du segment ST - Mesurée lors de l'exercice par rapport au repos (ex: 0.0, 0.6, 1.6, 2.0)
- X_{11} **st_slope :** Pente du segment ST lors de l'exercice - Numérique (0, 1, 2, 3)

age	sex	chest_pain_type	resting_bp_s	cholesterol	fasting_blood_sugar	resting_ecg	max_heart_rate	exercise_angina	oldpeak	st_slope	target
40.0	1.0	2.0	140.0	289.0	0.0	0.0	172.0	0.0	0.0	1.0	0.0
49.0	0.0	3.0	160.0	180.0	0.0	0.0	156.0	0.0	1.0	2.0	1.0
37.0	1.0	2.0	130.0	283.0	0.0	1.0	98.0	0.0	0.0	1.0	0.0
48.0	0.0	4.0	138.0	214.0	0.0	0.0	108.0	1.0	1.5	2.0	1.0
54.0	1.0	3.0	150.0	195.0	0.0	0.0	122.0	0.0	0.0	1.0	0.0

كليـــــة العلـــــوم
السمـلاليـــة - مراكـــش
FACULTÉ DES SCIENCES
SEMLALIA – MARRAKECH

Démonstration Pratique