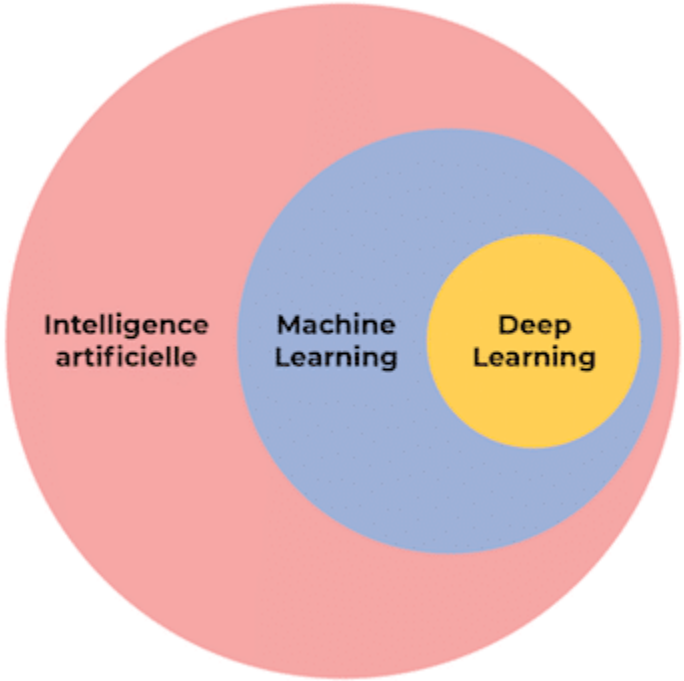
Introduction : le Machine Learning

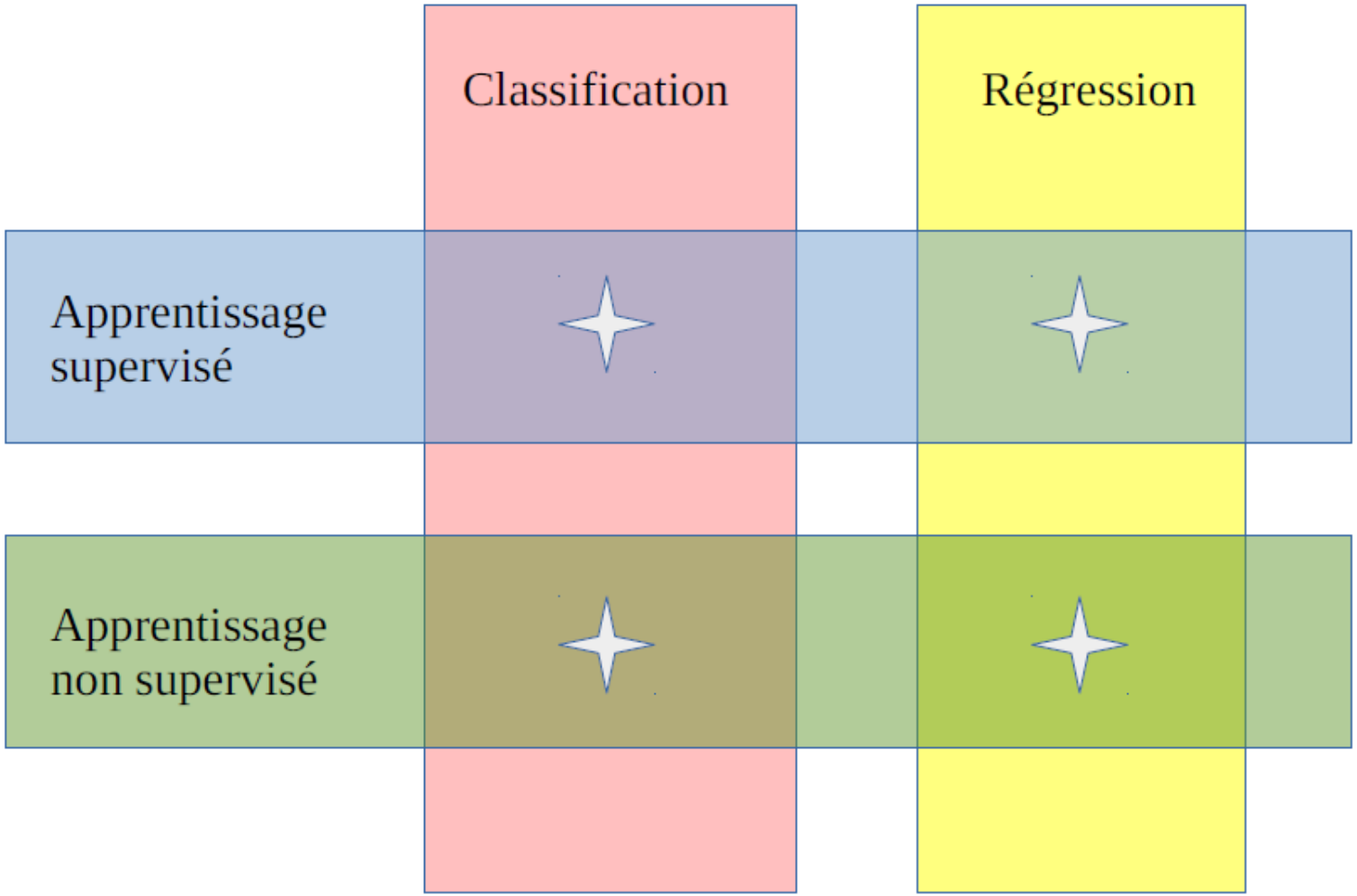
Quelques définitions



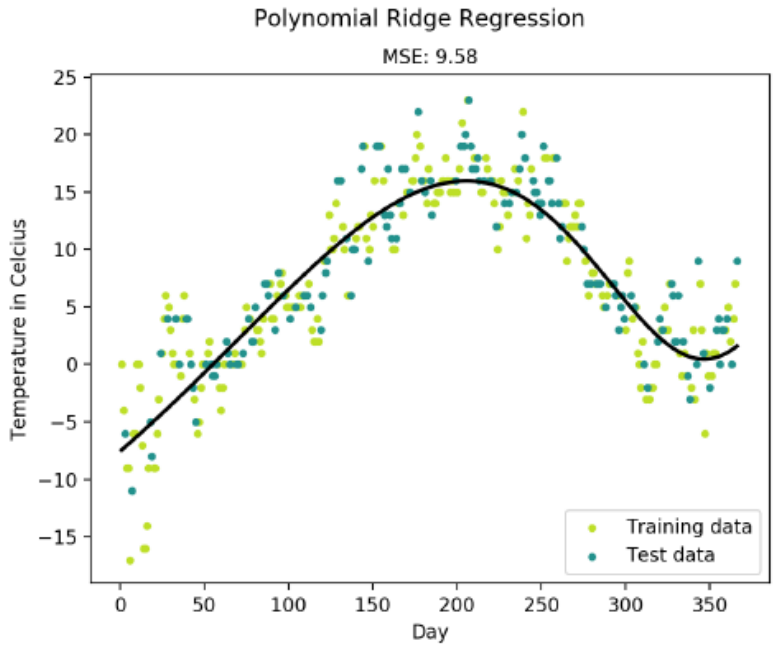
Machine learning : Apprentissage automatique

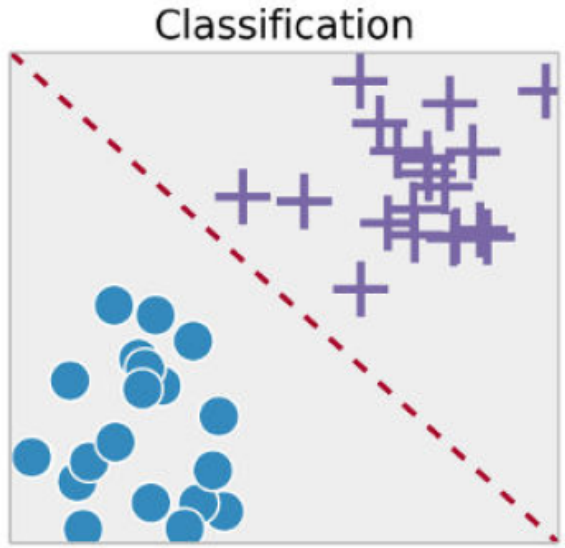
Deep Learning : Réseaux de neurones profonds

Grandes catégories d'algorithmes de machine learning



Classification / Régression





Régression	Classification
Prédire une variable quantitative	Prédire une classe (qualitative, discrète)

Apprentissage supervisé / non supervisé

■ **Apprentissage supervisé : Nécessite un jeu d'entraînement X (prédicteurs), y (prédictants, ie. variables que l'on veut prévoir)**

- Application principale: la classification ou la regression
- Exemples: corriger la température prévue par modèle de PNT, dire si de la neige est présente sur une image webcam

■ **Apprentissage non supervisé : Nécessite un jeu d'entraînement contenant uniquement des X**

- Application principale: le clustering
- Exemple: classer des situations météo en groupes homogènes

Une première méthode de Machine Learning:

La Régression Linéaire

La régression linéaire

■ **Méthode d'apprentissage supervisé :**

- Un jeu d'entraînement X, y
  - X : la superficie des maisons
  - y : le prix de vente

■ **Exemple : prévoir le prix de vente des maisons en fonction de leur superficie**

La régression linéaire: l'intuition

Trouver la droite qui se rapproche le plus du nuage de points

La régression linéaire: l'intuition

- Comment définir la « meilleure » droite ?

Définir une fonction de coût

- La « meilleure » droite est celle qui minimise la fonction de coût.

La fonction de coût

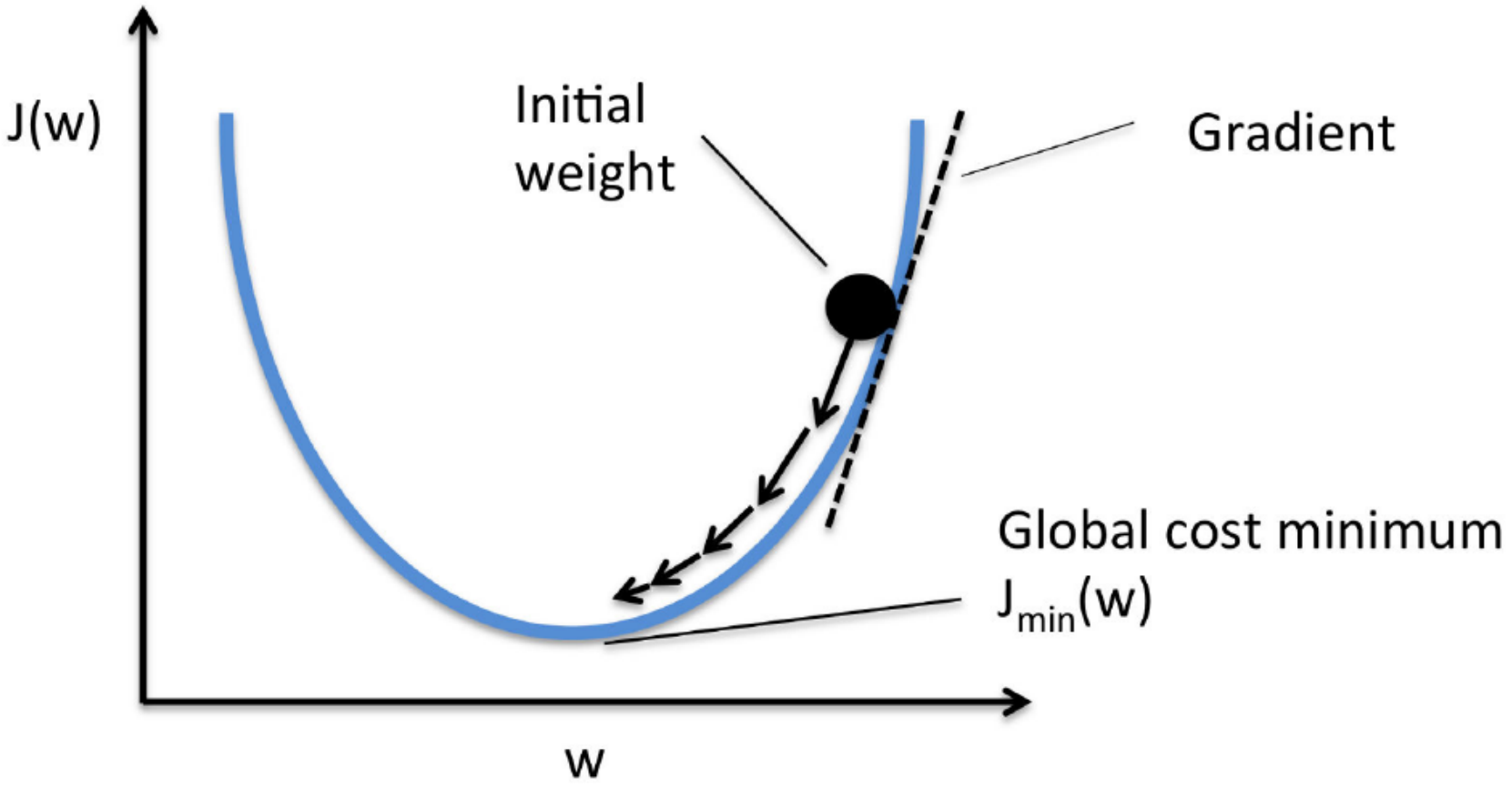
- **Soit x, y un échantillon du jeu d'entraînement (superficie et prix d'une maison)**
- **Soit $h(x)$ notre prédiction : $h(x) = w_0 . x + w_1$**
- **Une fonction de coût possible: écart quadratique moyen entre les prédictions et la vérité terrain**

$$J = \frac{1}{2m} \times \sum_{i=1}^{m} (h(x_i) - y_i)^2$$

m étant le nombre d'échantillons dans le jeu d'entraînement.

La fonction de coût: Trouver le minimum

La fonction de coût: La descente de gradient



La fonction de coût: Le calcul du gradient

■ Application à la régression linéaire

$$J = \frac{1}{2m} \times \sum_{i=1}^{m} (h(x_i) - y_i)^2$$

Avec $h(x) = w_0 . x + w_1$

■ Gradient :

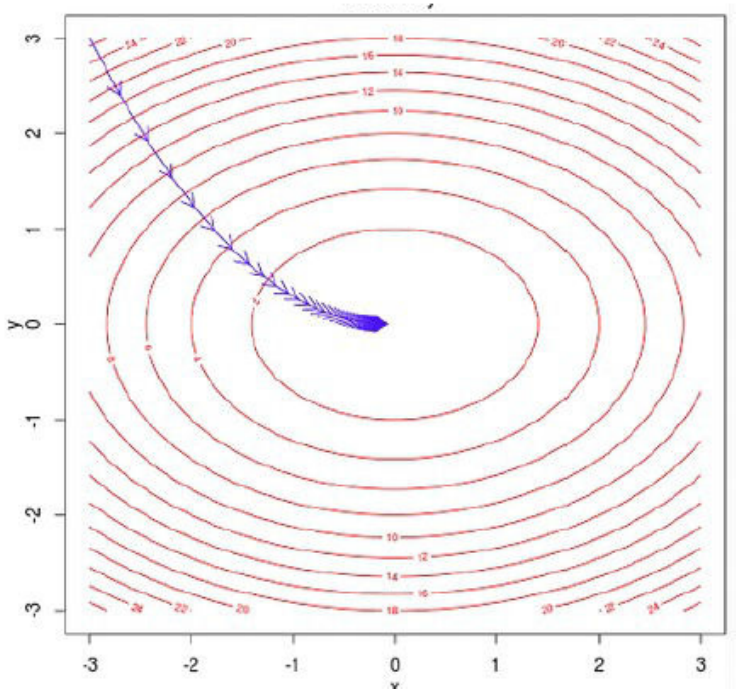
$$\frac{\partial J}{\partial w_0} = \frac{1}{m} \sum_{i=1}^{m} x_i . (h(x_i) - y_i)$$

$$\frac{\partial J}{\partial w_1} = \frac{1}{m} \sum_{i=1}^{m} (h(x_i) - y_i)$$

La fonction de coût: Le calcul du gradient

■ Répéter autant de fois que nécessaire :

$$\left\{ \begin{array}{l} w_0 := w_0 - \alpha . \dfrac{\partial J}{\partial w_0} \\ w_1 := w_1 - \alpha . \dfrac{\partial J}{\partial w_1} \end{array} \right\}$$

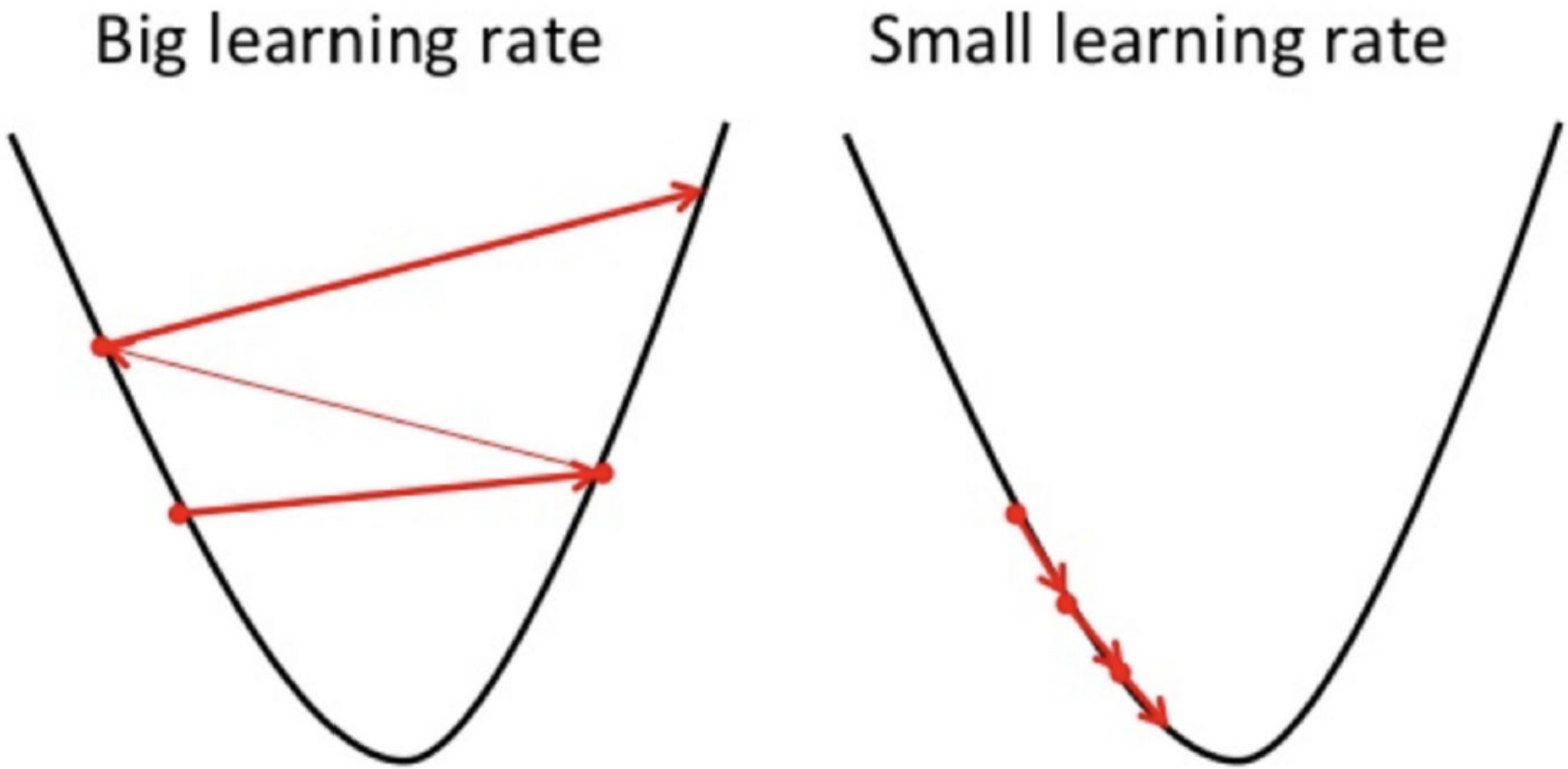


■ α est le coefficient d'apprentissage (learning rate)

La convergence en image

https://www.youtube.com/watch?v=1hGsKphwC-A

Influence du learning rate



Et si le jeu de données est très gros ?

■ Rappel calcul des gradients pour la régression linéaire :

$$\frac{\partial J}{\partial w_0} = \frac{1}{m} \sum_{i=1}^{m} x_i . (h(x_i) - y_i)$$

$$\frac{\partial J}{\partial w_1} = \frac{1}{m} \sum_{i=1}^{m} (h(x_i) - y_i)$$

■ Si m est très grand et X de dimension élevée, alors calculer la somme devient très long, voire interminable…

■ Exemple : 1 million d'images en 1024x1024

[IL FAUT CENTER!] PROBLEME…

La solution : la descente de gradient stochastique

■ **A chaque étape de descente de gradient, au lieu de prendre l'ensemble des échantillons d'un coup comme ceci :**

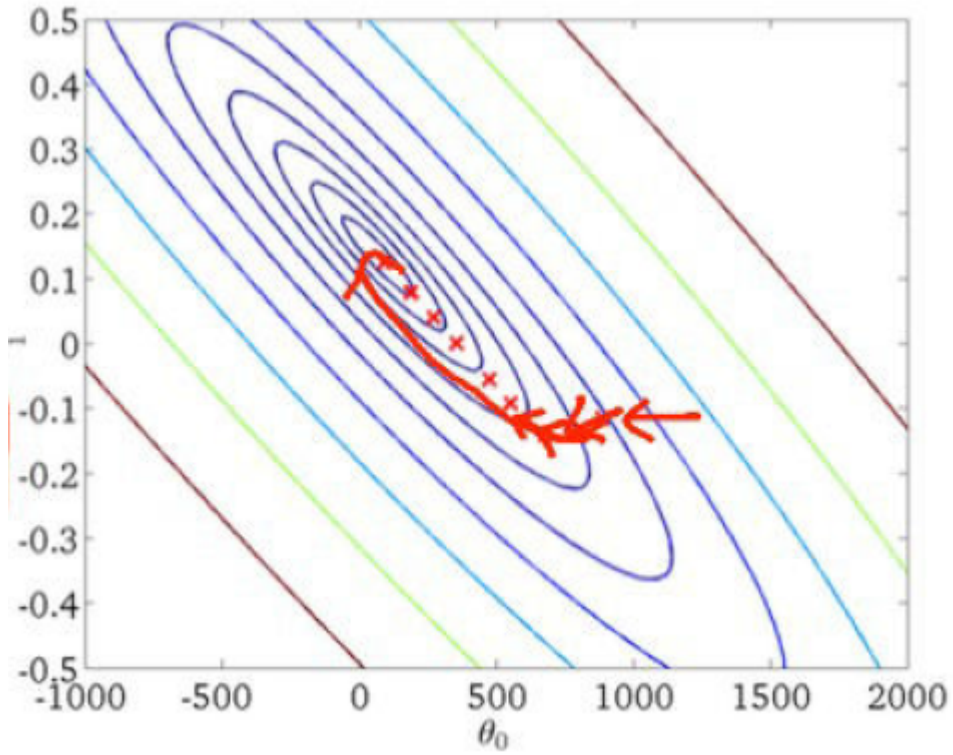
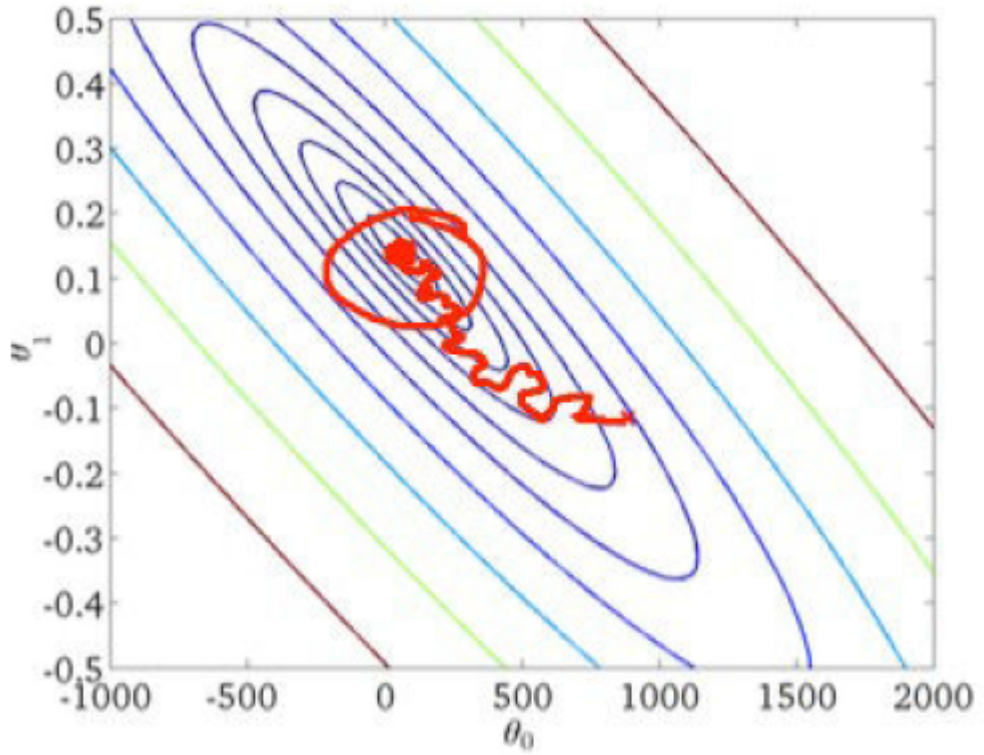
$$\begin{cases} w_0 := w_0 - \alpha . \frac{1}{m} \sum_{i=1}^{m} x_i . (h(x_i) - y_i) \\ w_1 := w_1 - \alpha . \frac{1}{m} \sum_{i=1}^{m} (h(x_i) - y_i) \end{cases}$$

■ **On itère sur les échantillons un par un :**

pour i allant de 1 à m, répéter :

$$\begin{cases} w_0 := w_0 - \alpha . \frac{1}{m} x_i . (h(x_i) - y_i) \\ w_1 := w_1 - \alpha . \frac{1}{m} (h(x_i) - y_i) \end{cases}$$

Illustration

Full batch gradient descent (FBGD)	Stochastic gradient descent (SGD)
	

Démo en images

https://www.youtube.com/watch?v=HvLJUsEc6dw

Taille de batch (batch size)

■ **Full batch GD : calcul du gradient sur l'ensemble du jeu de données**

- Inconvénient : demande le chargement de toutes les données en RAM ce qui est impossible pour les grands jeux de données.

■ **Stochastic GD : on estime le gradient échantillon par échantillon**

- Inconvénient : lent et convergence plus chaotique.

■ **Compromis : mini-batch gradient descent**

- On estime le gradient sur k échantillons à la fois (par exemple 32 échantillons).

En pratique, on essaie d'avoir le plus grand batch que la carte graphique peut acceuillir.

La notion d'epoch

■ **Dans la SGD, on estime le « gradient » échantillon par échantillon, ou par mini-batchs de quelques échantillons**

- Une passe complète sur le jeu de données s'appelle :

UNE EPOCH

■ **Le nombre d'époch est donc le nombre de passes effectuées sur le jeu d'entraînement lors de l'apprentissage.**

Des questions sur la descente de gradient ?

Hyper-paramètres – comment « régler » un modèle de Machine Learning

■ Que peut-on modifier dans un modèle de Machine Learning ?

— Le type et la complexité du modèle

- La régression linéaire est un modèle simple, mais on peut le complexifier : polynôme de degré n, random forest, réseaux de neurones...

— Certains paramètres spécifiques du modèle

- Pour un réseau de neurones : nb de couches, nb de neurones par couche...

— Mais aussi : le learning rate, la taille des batchs, le nombre d'époch ...

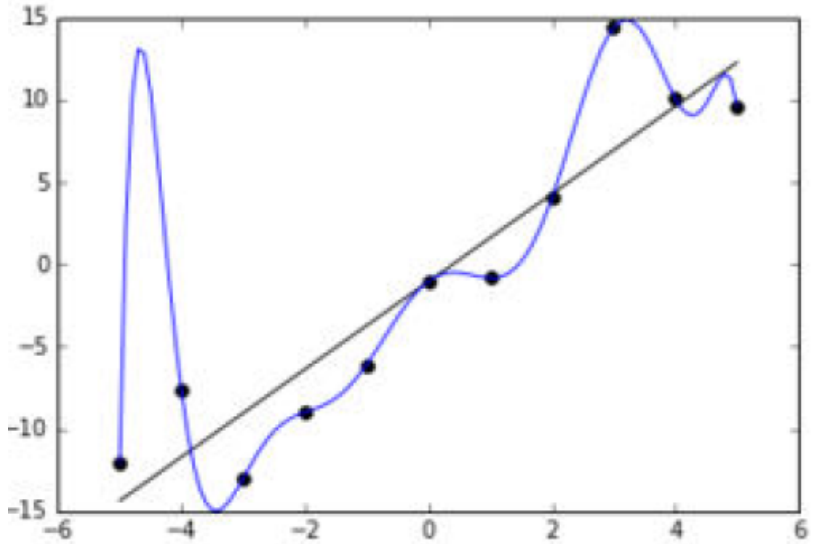
Comment choisir ces hyper-paramètres ?

Evaluer le modèle

Première idée: choisir les hyper-paramètres qui fonctionnent le mieux sur le jeu d'entraînement

Jeu de données d'entraînement

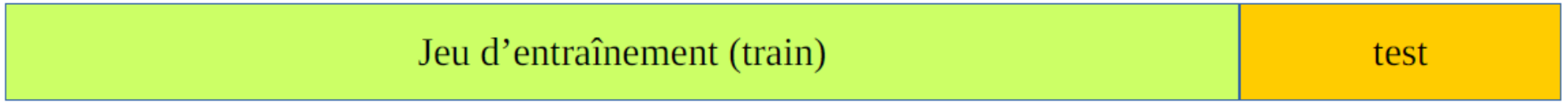
Pas bon. Le modèle risque de ne pas être capable de généraliser.



Evaluer le modèle

Deuxième idée:

- Choisir les hyper-paramètres qui fonctionnent le mieux sur un jeu de test



Pas bon. Aucune garantie que l'algorithme fonctionnera bien sur de nouvelles données.

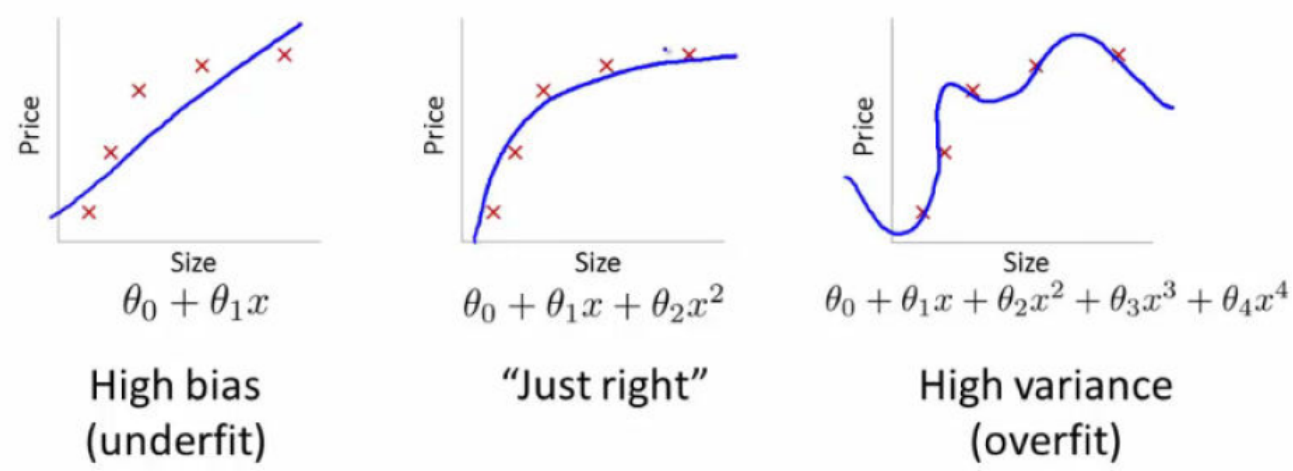
Evaluer le modèle

Troisième idée:

- Entraîner sur le jeu d'entraînement, choisir les hyper-paramètres qui fonctionnent le mieux sur un jeu de validation, puis une fois le modèle réglé, l'évaluer sur un jeu de test.

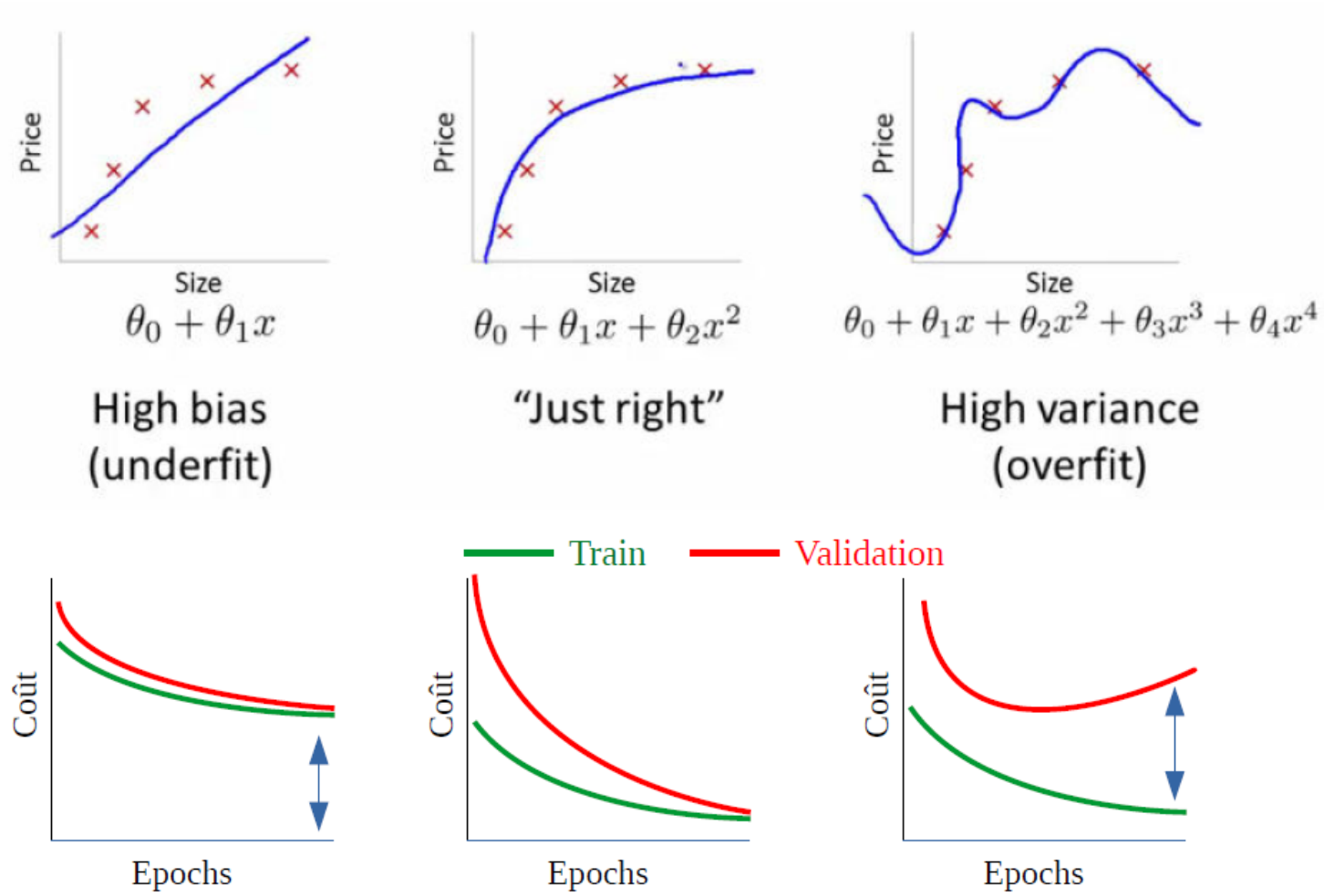
C'est mieux !

Sous-apprentissage - Sur-apprentissage



Sous-apprentissage	Bon modèle	Sur-apprentissage
Trop simple pour expliquer la variance		Colle trop au bruit du jeu de données

Sous-apprentissage - Sur-apprentissage



Combattre l’underfitting ("sous-apprentissage")

— Complexifier le modèle

- Exemple: modèle quadratique au lieu d’un modèle linéaire pour prédire le prix des maisons

— Ajouter des prédicteurs

- Exemple: il existe d’autres paramètres que la superficie qui influent sur le prix des maisons. Par exemple la localisation, le nombre de chambres, la distance au centre-ville...

Combattre l’overfitting ("sur-apprentissage") 1/2

— Ajouter des données d’entraînement

- Préférer situations météorologiques diverses (sur plusieurs années) plutot que sur courte période (situations très corréllées)

— Simplifier le modèle

- Eviter que le modèle dispose de suffisement de poids pour « apprendre par coeur » le jeu d’entraînement.

Combattre l’overfitting ("sur-apprentissage") 2/2

— Limiter la capacité d’apprentissage du modèle

- De nombreuses méthodes de régularisation permettent d'éviter le surapprentissage: *lasso, augmentation de données, early-stopping, dropout*.

— Entraîner le modèle moins longtemps ()

- Réduire le nombre d'épochs pour un réseau de neurones puisque le processus d'apprentissage est itératif

— Utiliser des ensembles

- Comme en météo, entraîner plusieurs modèles et combiner leurs prédictions.

Des questions?

Python pour le Machine Learning

Pourquoi Python ?

Language le plus utilisé dans le domaine

- Richesse des libraires et frameworks
- Communauté active et ressources abondantes

En pratique

- Simplicité et lisibilité du code
- Intégration facile avec d'autres outils et langages
- Performance et efficacité pour le traitement des données et la modélisation

Grandes étapes d'un projet de machine learning

1. Lecture des données et statistiques descritptives
2. Prétraitement des données
3. Modélisation
4. Evaluation
5. Présentation, mise en forme des résultats, export

Pandas

La librairie pour la lecture et le prétraitement des données

- Pandas est la librairie pour les données de type tableaux numériques (sorte d'excell plus avancé) et de séries temporelles
- Utile la notion de dataframe (similaire à R pour les connaisseurs)
- Multiples fonctions pour manipuler les données, le calcul de statistiques, la visualisation ...

Egalement utile en fin de projet: mise en forme/interprétation finale des résultats

La libriaire Scikit Learn

La librairie pour la modélisation

- Algorithmes, découpage des données, ...

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: Gradient boosting, nearest neighbors, random forest, logistic regression, and more...

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, stock prices.
Algorithms: Gradient boosting, nearest neighbors, random forest, ridge, and more...

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, grouping experiment outcomes.
Algorithms: k-Means, HDBSCAN, hierarchical clustering, and more...

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, increased efficiency.
Algorithms: PCA, feature selection, non-negative matrix factorization, and more...

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning.
Algorithms: Grid search, cross validation, metrics, and more...

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: Preprocessing, feature extraction, and more...