

Introduction à l'apprentissage automatique

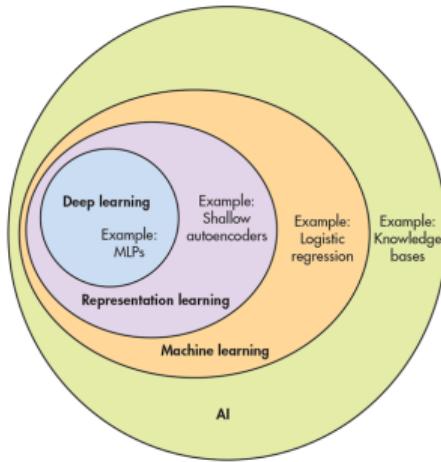
16 mars 2022

1. Intro : Qu'est ce que le *Machine Learning* ?
2. L'apprentissage supervisé
 - 2.1 La régression linaire
 - 2.2 La régression logistique
 - 2.3 Les problèmes de biais et de variances
 - 2.4 Les arbres de décisions
3. L'apprentissage non supervisé
 - 3.1 Les algorithmes de clustering
 - 3.2 Analyse en composantes principales
4. Autres algorithmes
 - 4.1 Recommandation
 - 4.2 Optimisation

- Introduction à l'apprentissage automatique (ou *machine learning* en anglais)
- En pratique : *Python*, *Jupyter*. Packages *numpy* *pandas*, *scikit-learn*
- Peu de pré-requis mathématiques : dérivées partielles et calcul matriciel
- Largement inspiré du cours de **Andrew Ng** sur [Coursera](#).

Allez, on démarre en douceur !

1. Intro : Qu'est ce que le *Machine Learning*



[From [MIT Press book Deep Learning](#)]

A screenshot of a tweet from Mat Veloso (@matveloso). The tweet reads:
Difference between machine learning and AI:
If it is written in Python, it's probably machine learning
If it is written in PowerPoint, it's probably AI
The tweet has 198 replies, 8.3k retweets, and 23k likes.
Timestamp: 02:25 - 23 nov. 2018

- **AI** : Domaine d'étude ⇒ abus de langage (NN, RL)
- **ML** : Algorithmes/outils développés dans le cadre de la recherche sur l'IA

1. Intro : Qu'est ce que le *Machine Learning*

- **Arthur Samuel :**

"The field of study that gives computers the ability to learn without being explicitly programmed."

- **Tom Mitchell :**

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ."

- **L'idée :** Une machine apprend *seule* à réaliser une tache complexe à l'aide de processus itératifs simple.

1. Intro : Qu'est ce que le *Machine Learning*

- Les principaux types d'apprentissage :

Supervisé

- Utilise des données *labeledisées*
- La machine apprend par l'exemple
- *Prédit* le résultat pour de nouveaux événements
- Problèmes de prédictions et de classification
- Regression linéaire et logistique
- Réseaux de Neurones
- Arbres de décisions

Non-supervisé

- Données non *labeledisées*
- La machine apprend par elle même à identifier une structure
- Évaluation des performances compliqué.
- Problèmes de classification, réduction de dimensions
- K-means
- Analyse en Composante Principale

Par renforcement

- Un agent A, effectue une action Ac, l'environnement E lui renvoie une récompense.
- Récompenses à court et long terme
- Utilisé par Deepmind (alphaGo)

À quelles problématiques répond le Machine Learning ?

- **Prédictions** Prédire une valeur continue à partir de caractéristique données
- **Projections** Prédictions spécifique de séries temporelles : $y = f(y(t - 1), y(t - 2), \dots)$
- **Classifications** Prédire la classe (discret) d'un objet en fonction de ses caractéristiques
- **Segmentations** Regrouper des objets par similarité dans l'espace des variables utilisé
- **Compréhensions** Comprendre l'importance de variables d'intérêt dans un contexte donné

2. L'apprentissage supervisé

- Utilise des données *labélisées*
- La machine apprend par l'exemple
- *Prédit* le résultat pour de nouveaux événements
- Problèmes de régression et de classification
- Régression linéaire et logistique
- Réseaux de Neurones
- Arbres de décisions

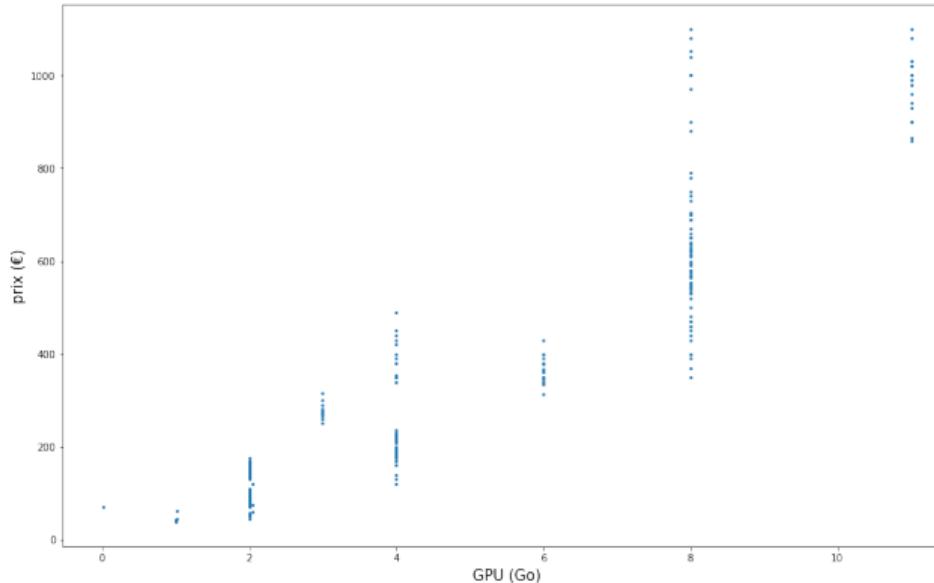
2.1 La régression linéaire

- Déterminer une relation *linéaire* entre *input(s)* (features) et *output* :
⇒ **Apprentissage Supervisé**
- Prédiction d'une valeur **continue** (e.g. non discrète, non catégorielle)
- Applications :
 - Recherche de corrélations
 - En science, modélisation de phénomènes (physiques, biologiques, ...)
 - Dans le domaine médical : les études épidémiologique
 - Dans la finance/économie : prédictions des tendances
 - ...

Sujet Data Science ⇒ Premier algorithme à tester !

2.1 Un exemple : le prix d'une carte graphique

- La propriété principale d'une carte Graphique : valeur de **GPU**
- Données, liste de carte graphiques dont on connaît le couple $\{GPU; \text{prix}\}$:



2.1 Construire un modèle (regression linéaire)

- Soit : x_1 la valeur de GPU de nos m carte graphiques, et y leur prix
- On cherche à déterminer le modèle pour prédire un prix \hat{y} à partir x_1 :

$$\hat{y} = h_{\theta}(x_1)$$

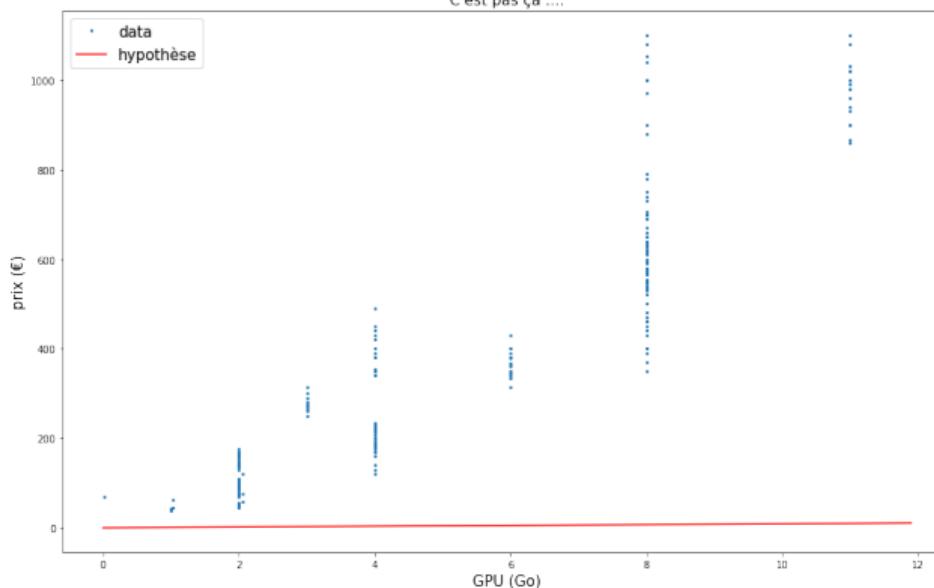
- On définit le paramètre θ_1 qui va lier x_1 à \hat{y} :

$$h_{\theta}(x) = \theta_1 x_1$$

- Rappel math : **fonction linéaire** $f(x) = kx$

2.1 Construire un modèle (regression linéaire)

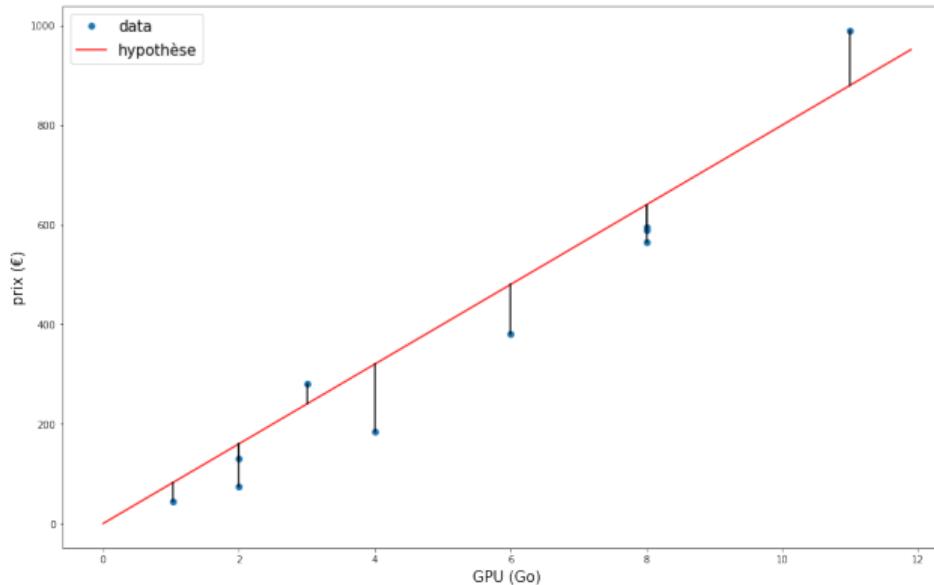
- Initialisons aléatoirement la valeur de θ_1



- C'est pas encore ça ...

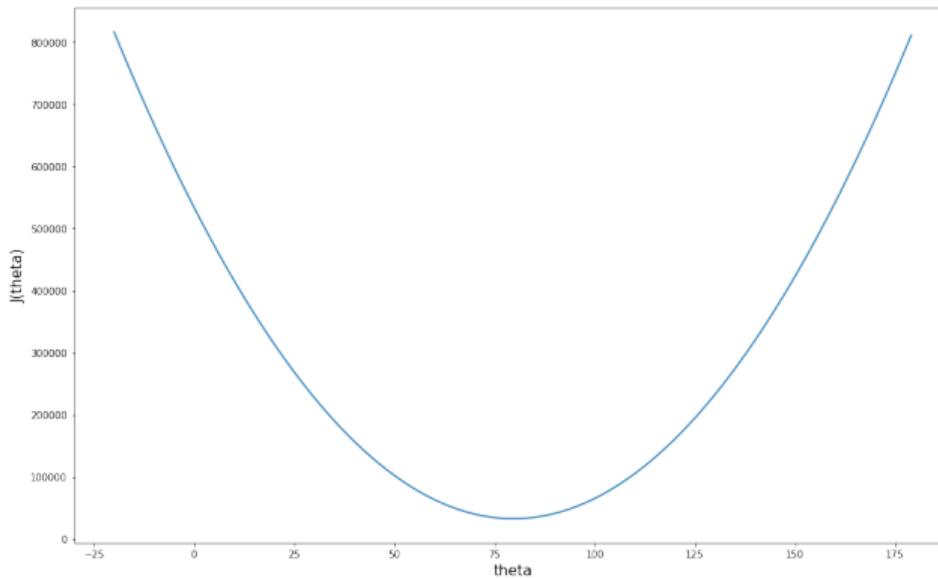
2.1 La fonction de coût

- $J(\theta)$: véracité de notre modèle
- Ex, somme quadratique des erreurs : $J(\theta) = \frac{1}{2m} \sum_{i=0}^m (\hat{y}^{(i)} - y^{(i)})^2$



2.1 La fonction de coût

- On cherche à trouver la valeur de θ_1 qui **minimise** $J(\theta)$
- En Brute ...



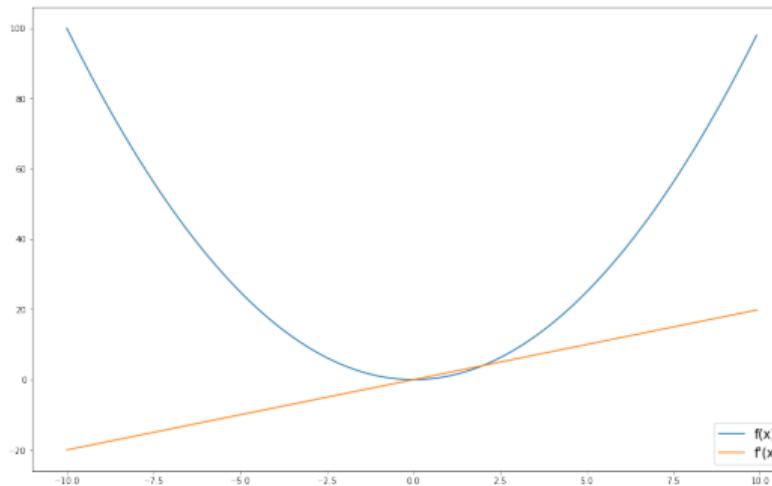
- ... essayons d'optimiser

2.1 La descente de gradient

- Algorithme pour arriver “*rapidement*” au minimum de $J(\theta)$

- On va utiliser la *dérivation* : $\frac{d}{d\theta_1} J(\theta)$:

Si $J(\theta)$ est croissant : $\frac{d}{d\theta_1} J(\theta) > 0$, Si $J(\theta)$ est décroissant : $\frac{d}{d\theta_1} J(\theta) < 0$



2.1 La descente de gradient

- (Encore) un peu de math, la descente de gradient s'écrit :

Descente de gradient

Répéter jusqu'à convergence : {

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta)$$

}

- α : taux d'apprentissage (*learning rate*), **seul** paramètre de l'algorithme.
- On va itérativement modifier la valeur de θ_1 en fonction de la dérivée de $J(\theta)$, jusqu'à minimiser $J(\theta)$ (*convergence*).

2.1 La descente de gradient

- Dérivons donc notre fonction de coût :

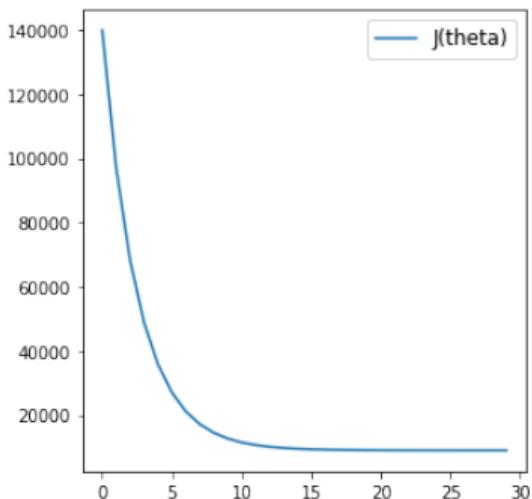
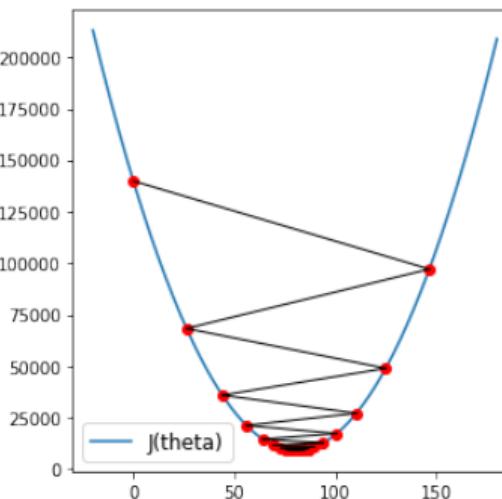
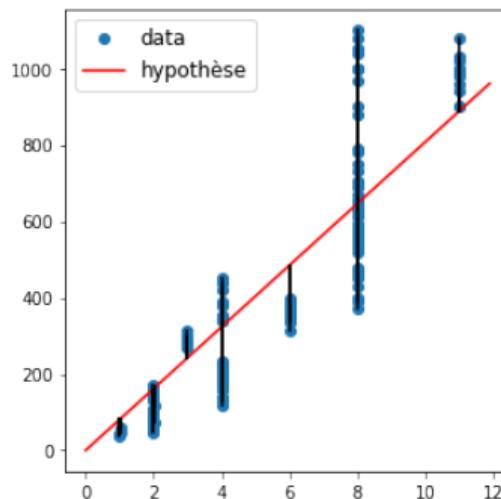
$$J(\theta) = \frac{1}{2m} \sum_{i=0}^m (\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{2m} \sum_{i=0}^m (\theta_1 x_1^{(i)} - y^{(i)})^2$$
$$\frac{d}{d\theta_1} J(\theta) = \frac{1}{m} \sum_{i=0}^m (\hat{y}^{(i)} - y^{(i)}) x_1^{(i)}$$

- Un peu de *hand-tunning* :
 - Le learning rate (α) est fixé à 0.03 (0.045 pour la démo)
 - Précision $\epsilon = 0.0001$ pour arrêter la descente de gradient

2.1 Préparation du dataset

- On sépare les données en trois sous-échantillons :
 - **Train** : (60%) échantillon d'entraînement
 - **Validation** : (20%) échantillon qui nous permet de mesurer les performances de différents algorithmes et pour différentes valeurs d'hyperparamètres
 - **Test** : (20%) échantillon de test qui donne la performance du modèle final
- On peut utiliser deux échantillons train/test (80/20 ou 70/30) dans certains cas.
- Il est important de séparer les données en différents sous-échantillons de test pour être sûr que le modèle de généralise bien
- S'assurer que les sous-échantillons proviennent de la même source et soient représentatifs

2.1 C'est parti !

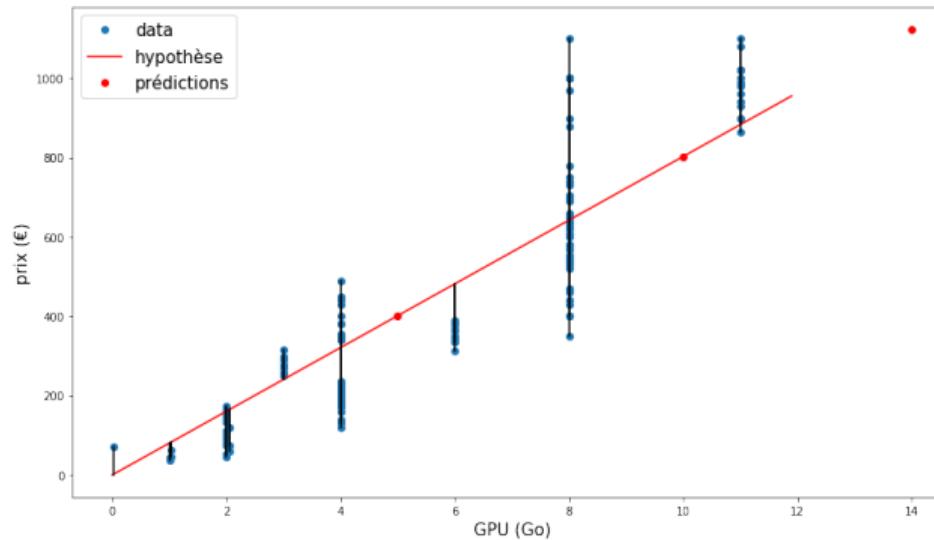


- La descente de gradient c'est achevée au bout de quelques itérations
- On peut voir que $J(\theta)$ a continuellement diminué à chaque itération

$$\theta_1 \approx 80$$
$$err_{train} \approx err_{test}$$

2.1 On peut maintenant faire une prédition

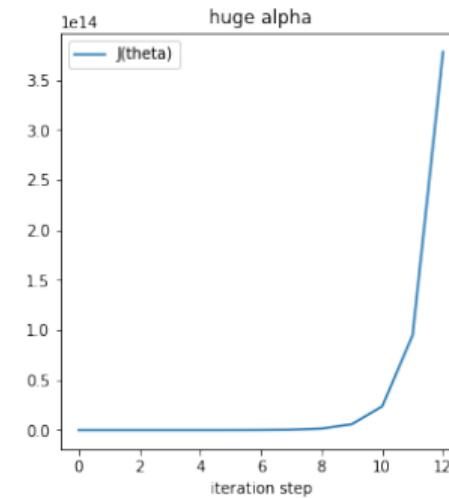
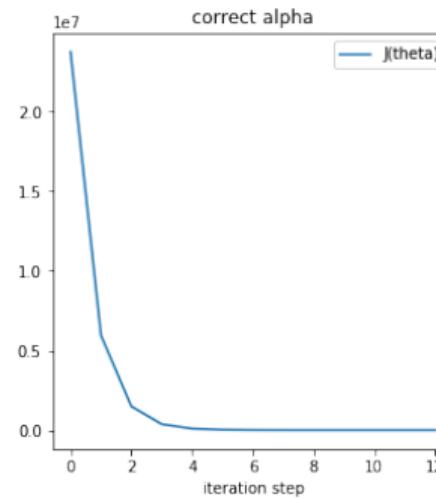
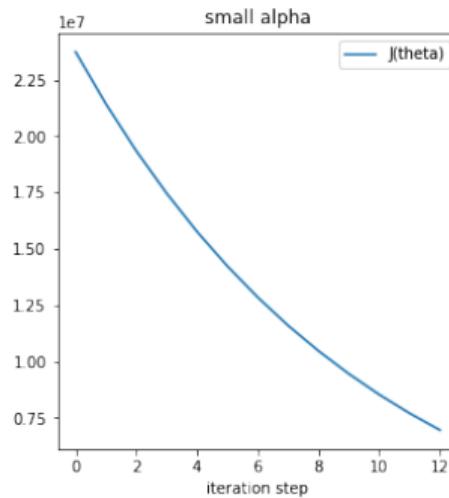
- Quel serait le prix de cartes avec 5, 10 et 14 Go de GPU ?



- On pourra les vendre autour de 400, 800 et 1100 euros !

2.1 Le choix du taux d'apprentissage

- **α Trop grand** : la descente de gradient diverge
- **α Trop petit** : la descente de gradient est très longue



2.1 Un mot sur la régression linéaire multivariée

- Même principe, mais avec plusieurs variables x_i (donc plusieurs paramètres θ_i)
- on peut rajouter un biais θ_0 , cad un terme constant : $\hat{y} = \theta_0 + \theta_1 x_1 + \dots$
- Notre fonction hypothèse s'écrit alors :

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n = \theta_0 + \sum_{i=1}^n \theta_i x_i$$

- **Astuce :** On définit $x_0 = 1$, et on re-écrit la fonction hypothèse :

$$h_{\theta}(x) = \theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n = \sum_{i=0}^n \theta_i x_i$$

- La fonction de coût reste inchangée

2.1 Un mot sur la regression linéaire multivariables

- Dans le cas multivariables, la descente de gradient devient :

Descente de gradient (cas multivariables)

Répéter jusqu'à convergence : {

$$\theta_0 := \theta_0 - \alpha \frac{d}{d\theta_0} J(\theta)$$

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta)$$

$$\theta_2 := \theta_2 - \alpha \frac{d}{d\theta_2} J(\theta)$$

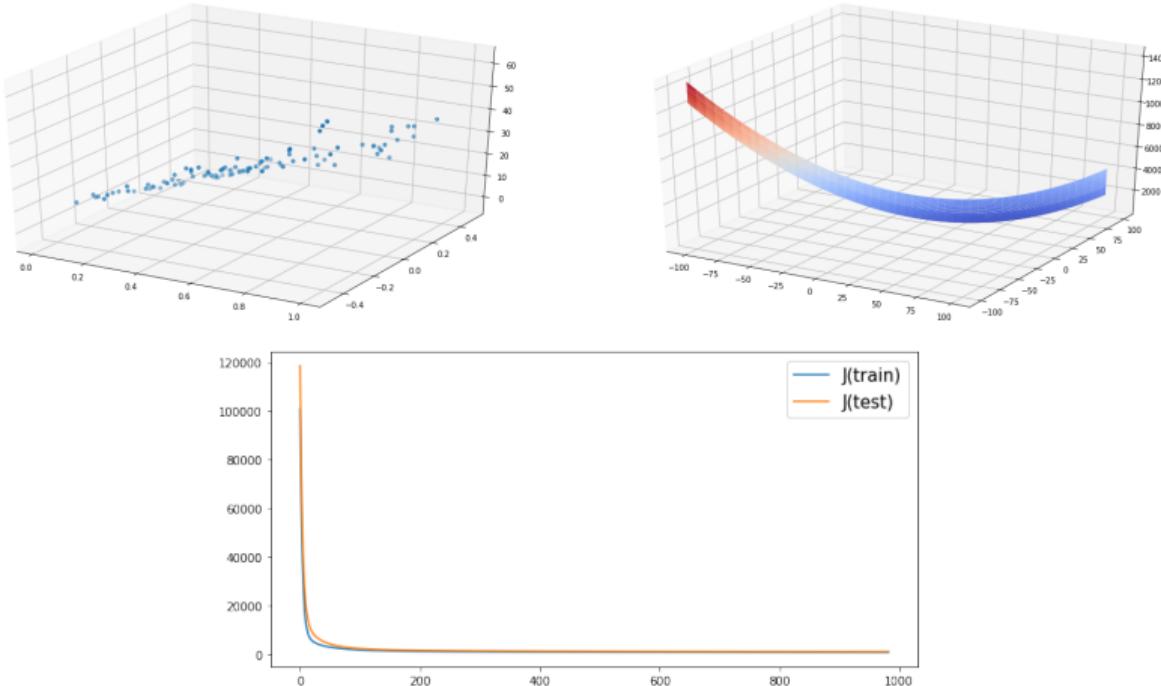
...

}

- Il est très important de simultanément changer les valeurs des paramètres.

2.1 Un mot sur la régression linéaire multivariées

- Pour illustrer : régression linéaire à deux dimensions



2.1 Affinons notre modèle de carte graphiques

- Plus de features : chipset, fréquence, consommation, ...
- Il va falloir explorer et nettoyer les données :
 - Gestion des données manquantes / aberrantes
 - *Features engineering*
 - Normaliser le dataset (pour accélérer la descente de gradient)

2.1 Features scaling

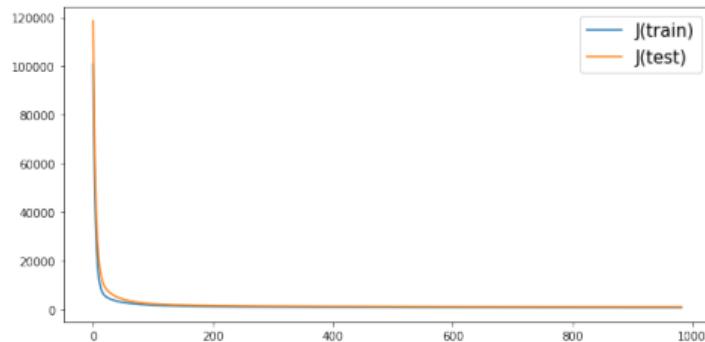
- Mettre les variables à la même échelle (performances)
- Normaliser les variables : $-1 \leq x_i \leq 1$

	Feature Scaling	Mean normalization
Start range	$x_{min} \leq x \leq x_{max}$	$x_{min} \leq x \leq x_{max}$
Transformation	$x := \frac{x - x_{min}}{x_{max} - x_{min}}$	$x := x - x_{mean}$
New range	$0 \leq x \leq 1$	$(x_{min} - x_{mean}) \leq x \leq (x_{max} - x_{mean})$

- En combinant les deux : $x := \frac{x - x_{mean}}{x_{max} - x_{min}} \Rightarrow -1 \leq x \leq 1$
- Remarque : Il est possible de remplacer $x_{max} - x_{min}$ par l'écart type : $\sigma_x = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$

2.1 Régression linéaire multivariées : Résultats

- On utilise la même valeur de $\epsilon = 0.0001$ et $\alpha = 0.03$
- Plus long ! Mais meilleur résultats :
- Modèle simple : $err \approx 100$
- Modèle multivariable : $err \approx 30$



2.1 Pour conclure sur la régression linéaire

- **Regression Linéaire** : \hat{y} est une valeur *continue*
 - Valeur discrète : **Regression Logistique** (*classification*)
- Le résultat \hat{y} dépend **linéairement** des variables x_i si :

$$\hat{y} = \theta_1 x_1 + \cdots + \theta_n x_n = \sum_{i=1}^n \theta_i x_i$$

- **Supervisé** : y connu pour chaque élément du jeu de données d'entraînement
- Facile à implémenter (encore plus avec Scikit-learn ...), rapide :
 - ⇒ bon point de départ sur un sujet

- **Classification** : prédire un nombre limités de valeurs discrètes
 - **Classification binaire** : Deux valeurs possibles : Vrai ou Faux (spam / non spam)
 - **Classification multiclasse** : plusieurs valeurs possibles (camion, voiture, piétons, vélos, ...)
- **Classification binaire** : En utilisant la régression linéaire ?
 - $y \in \{0, 1\} \Rightarrow$ On définit un seuil S pour $h_\theta(x)$:
$$\begin{cases} h_\theta(x) \geq S \rightarrow y = 1 \\ h_\theta(x) < S \rightarrow y = 0 \end{cases}$$
- **Problème** : On voudrait que $0 \leq h_\theta(x) \leq 1$
 \Rightarrow Il faut redéfinir notre fonction hypothèse !

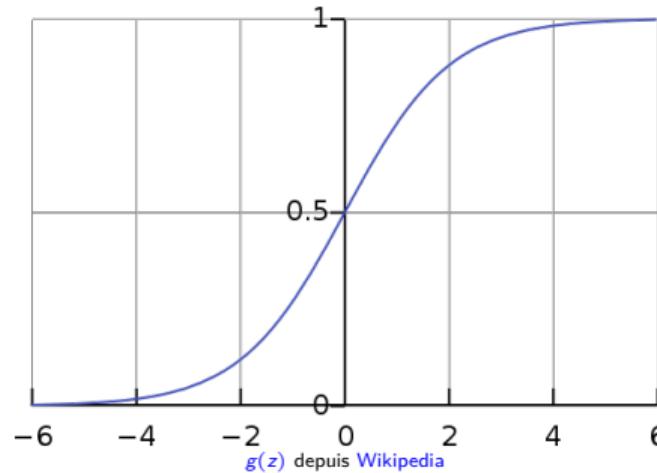
2.2 La régression logistique

- Modèle de la régression logistique :

- On utilise la **Fonction Sigmoïde** $g(z) = \frac{1}{1+e^{-z}}$

$$\begin{cases} z = \sum_{i=0}^n \theta_i x_i \\ h_\theta(x) = g(z) \end{cases}$$

$$0 \leq h_\theta(x) \leq 1$$



- Classification** : On prédit une **probabilité**

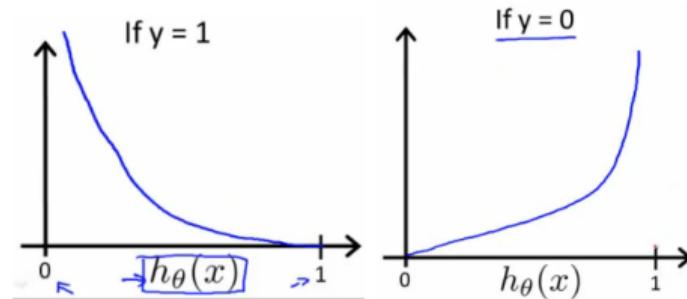
2.2 La régression logistique

- On change également la fonction de coût :

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m Cost(h_\theta(x^{(i)}), y^{(i)})$$

- Avec :

$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



2.2 La régression logistique

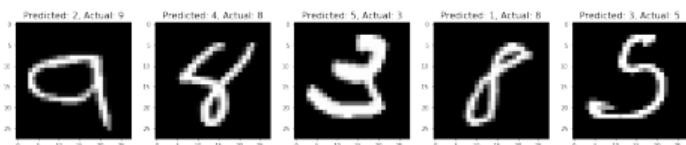
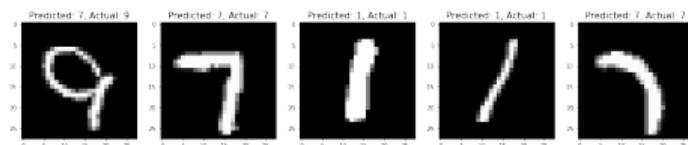
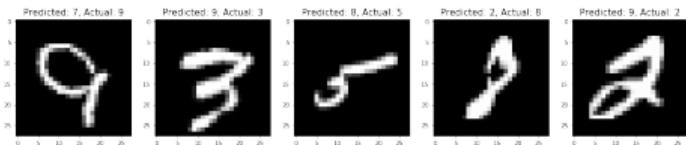
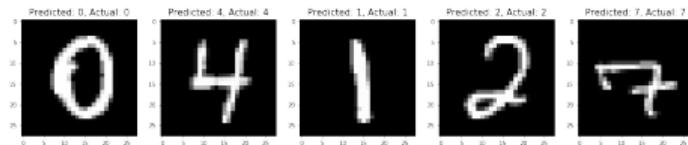
- Et la **Classification multiclasse**? $y \in \{0, 1, 2, \dots, n\}$
- La **descente de gradient** ne permet pas de la résoudre
- On peut *tricher* en utilisant la méthode '*One-vs-all*' :
 - On remplace le problème multiclasse par $n + 1$ problèmes binaire
 - Probabilité que y soit dans une classe ou qu'il soit dans une des autres :

$$\begin{cases} h_{\theta}^{(0)} = P(y = 0|x; 0) \\ h_{\theta}^{(1)} = P(y = 1|x; 0) \\ \dots \\ h_{\theta}^{(n)} = P(y = n|x; 0) \\ \text{prediction} = \max_i(h_{\theta}^{(i)}(x)) \end{cases}$$

- Algorithme d'optimisation avancés : *Conjugate gradient*, *(L-)BFGS*, ...

2.2 La régression logistique

- Application : vision par ordinateur, reconnaissance d'images.
- Objets "*simples*" : chiffres manuscrits
- Régression logistique multiclasse sur MNIST :
 - Durée d'apprentissage $\sim 20\text{sec}$ / Précision $\approx 91\%$



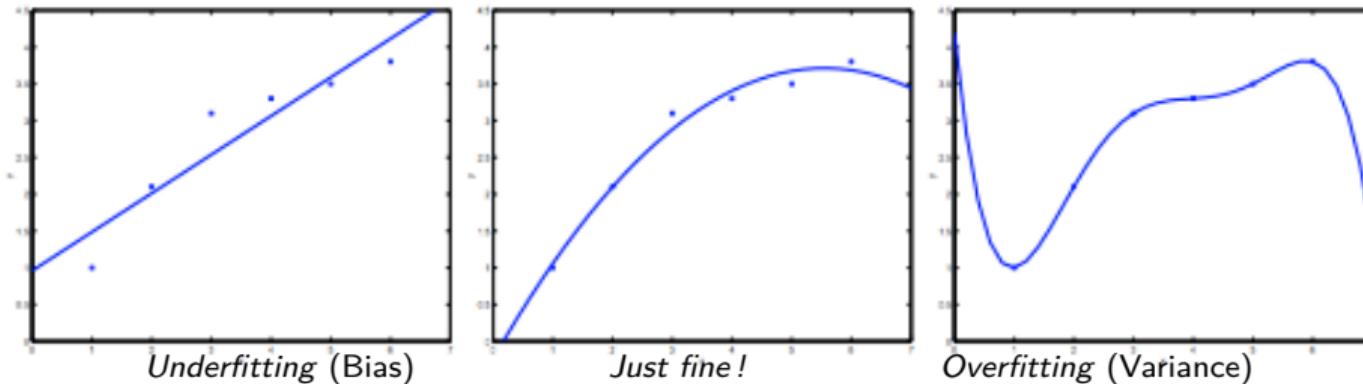
2.3 Mesurer la performance des algorithmes de classifications

- Selon les problèmes, il existe différentes métriques permettant de mesurer la performance d'un algorithme

- Accuracy = $\frac{N_{\text{correct_pred}}}{N_{\text{items}}}$
- Precision (PPV) = $\frac{N_{\text{True_pos}}}{N_{\text{pred_pos}}}$
- Recall (Sensitivity) = $\frac{N_{\text{True_pos}}}{N_{\text{real_pos}}}$
- F1-Score = $\frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$

Predicted condition			Sources: [3][4][5][6][7][8][9][10] view · talk · edit	
Total population = P + N	Positive (PP)	Negative (PN)	Informedness, bookmaker informedness (BM) = TPR + TNR - 1	Prevalence threshold (PT) = $\frac{\sqrt{\text{TPR} \times \text{FPR}} - \text{FPR}}{\text{TPR} - \text{FPR}}$
Actual condition	Positive (P)	False negative (FN), type II error, miss, underestimation	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power = $\frac{\text{TP}}{\text{P}} = 1 - \text{FNR}$	False negative rate (FNR), miss rate = $\frac{\text{FN}}{\text{P}} = 1 - \text{TPR}$
Negative (N)	False positive (FP), type I error, false alarm, overestimation	True negative (TN), correct rejection	False positive rate (FPR), probability of false alarm, fall-out = $\frac{\text{FP}}{\text{N}} = 1 - \text{TNR}$	True negative rate (TNR), specificity (SPC), selectivity = $\frac{\text{TN}}{\text{N}} = 1 - \text{FPR}$
Prevalence $= \frac{P}{P+N}$	Positive predictive value (PPV), precision $= \frac{\text{TP}}{\text{P}} = 1 - \text{FDR}$	False omission rate (FOR) $= \frac{\text{FN}}{\text{P}} = 1 - \text{NPV}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Negative likelihood ratio (LR-) $= \frac{\text{FNR}}{\text{TNR}}$
Accuracy (ACC) = $\frac{\text{TP} + \text{TN}}{P + N}$	False discovery rate (FDR) $= \frac{\text{FP}}{\text{P}} = 1 - \text{PPV}$	Negative predictive value (NPV) = $\frac{\text{TN}}{\text{N}}$ $= 1 - \text{FOR}$	Markedness (MK), deltaP (Δp) $= \text{PPV} + \text{NPV} - 1$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR-}}$
Balanced accuracy (BA) $= \frac{\text{TPR} + \text{TNR}}{2}$	F1 score $= \frac{2\text{PPV} \times \text{TPR}}{\text{PPV} + \text{TPR}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$	Fowlkes-Mallows index (FM) $= \sqrt{\text{PPV} \times \text{TPR}}$	Matthews correlation coefficient (MCC) $= \sqrt{\text{TPR} \times \text{TNR} \times \text{PPV} \times \text{NPV}} - \sqrt{\text{FNR} \times \text{FPR} \times \text{FOR} \times \text{FDR}}$	Threat score (TS), critical success index (CSI), Jaccard index = $\frac{\text{TP}}{\text{TP} + \text{FN} + \text{FP}}$

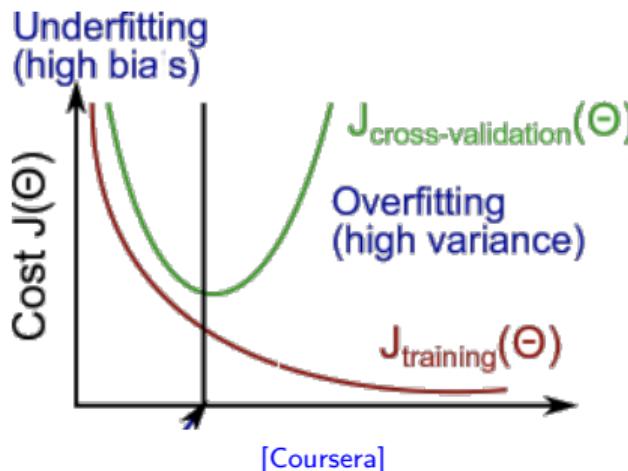
2.3 Les problèmes de biais et de variance



- **Underfitting** : modèle trop simple (pas assez de variables)
- **Overfitting** : modèle trop complexe, deux manière de résoudre :
 - Réduire le nombres de variables ou changer les paramètres du modèle
 - Utiliser des méthodes de **régularisation**

2.3 Les bonnes pratiques : biais et variance

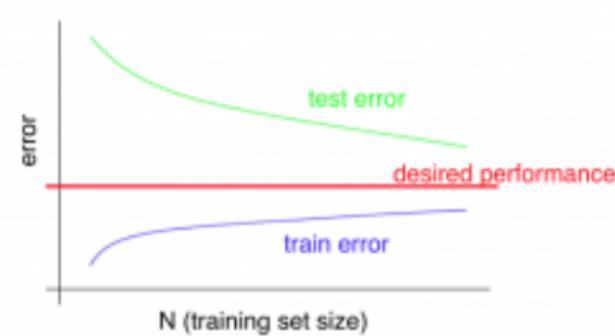
- Si $J_{train}(W, b) \gg J_{valid}(W, b)$: problème de variance : **Overfitting**
- Si $J_{train}(W, b) \approx J_{valid}(W, b) \gg 0$: problème de biais : **Underfitting**
- Comment diagnostiquer ? deux valeurs à regarder :
 - Erreur de l'échantillon d'entraînement : t_{err}
 - Erreur de l'échantillon de validation : v_{err}
- On estime le cas idéal (Bayes ou opérateur humain) : $\approx 0\%$



t_{err}	v_{err}	problème
1%	11%	haute variance
15%	16%	haut biais
15%	30%	haut biais & haute variance
0.5%	1%	bas biais & basse variance

2.3 Les bonnes pratiques : courbes d'apprentissage

- Entrainer un modèle en augmentant le nombre d'exemples et monitorer l'évolution de l'erreur :



Biais et variance [Coursera]

2.3 Les bonnes pratiques : Que faire ?

- **Haut Biais ?**

(Performance entraînement)

→ Oui →

Changer les hyperparamètres

Plus de données et étude qualité

Algorithme plus complexe



Non



- **Haute Variance ?**

(Performance validation)

→ Oui →

Changer les hyperparamètres

Plus de données et étude qualité

Régularisation



Non



OK !



Recommencer ! →



Recommencer ! →

2.3 La régularisation : RIDGE (L2)

- Contraindre les paramètres θ_j afin d'éviter le *sur-apprentissage (overfitting)*
- Particulièrement efficace si les paramètres du problème présentent une covariance importante
- On ré-écrit la fonction de coût en ajoutant un **terme de régularisation**

$$J(\theta) = \frac{1}{2m} \left(\sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right)$$

- λ : Paramètre de régularisation
- Si λ est trop grand : risque d'*underfitting*
- Si $\lambda = 0$: pas de régularisation.
- **Remarque** : On ne régularise pas le terme constant θ_0

2.3 La régularisation : LASSO (L1)

- Utile dans le cas où l'on souhaite éviter que des coefficients prennent une valeur nulle
- Réduit le nombre de paramètre dominants dans la solution
- On ré-écrit la fonction de coût en ajoutant un terme de régularisation

$$J(\theta) = \frac{1}{2m} \left(\sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n |\theta_j| \right)$$

- λ : Paramètre de régularisation
- Si $\lambda = 0$: pas de régularisation.
- **Remarque :** On ne régularise pas le terme constant θ_0

2.3 La régularisation : ELASTIC NET

- Combinaison des régularisation **RIDGE** et **LASSO**
- Permet de combiner les avantages des deux types de régularisation
- On ré-écrit la fonction de coût en ajoutant un **terme de régularisation LASSO** et un **terme de régularisation RIDGE**

$$J(\theta) = \frac{1}{2m} \left(\sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \rho \sum_{j=1}^n |\theta_j| + \frac{\lambda(1-\rho)}{2} \sum_{j=1}^n \theta_j^2 \right)$$

- λ : Paramètre de régularisation
- ρ : *l1_ratio*, permet de contrôler la combinaison des régularisations L1 et L2
 - Si $\rho = 1$: *Régularisation LASSO*
 - Si $\rho = 0$: *Régularisation RIDGE*

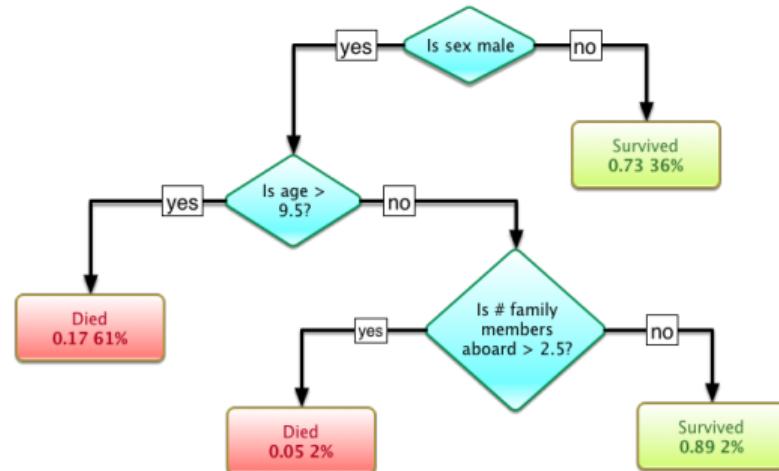
2.4 Autres algorithmes de régression

- Ici, nous nous sommes restreints à la minimisation du carrés des erreurs (Least squares method) à l'aide d'une descente de gradient
- Il existe d'autres outils permettant de traiter ces sujets, selon les caractéristiques du problèmes (corrélations, covariance, hétéroscédasticité, ...)
 - Ordinary least squares : Estimation des paramètres par calcul matriciel : $\theta = (X^T X)^{-1} X^T y$
 - Régression Bayésienne : Les termes de régularisation sont estimés en mêmes temps que les paramètres
 - Maximisation de vraisemblance (Likelihood)
 - L-BFGS : Algorithme d'optimisation appartenant aux méthodes quasi-Newton

2.4 Les arbres de décisions

- Effectue une prédition (regression ou classification) par une succession de décision simples : *if-else-then* sur les différentes variables
- Arbre = suite de décisions (*noeuds*) ammenant à une prédition (*feuille*)

- La complexité de la structure de données qu'il est possible de représenter avec une arbre de décision est proportionnelle à la profondeur de l'arbre



Un arbre pour déterminer la probabilité de survie des passagers du Titanic (depuis [Wikipedia](#))

2.4 Les arbres de décisions : Apprentissage

- La *méilleure* séparation possible est déterminée :
 - Toutes les variables/sélection possibles
 - Celle qui minimise une fonction de coût
- L'échantillon d'entraînement est séparé suivant cette décision, créant ainsi deux feuilles (sous-échantillon)
- On recommence l'opération tant que l'on n'a pas atteint la profondeur voulue ou bien que la pureté de chaque feuille atteint un niveau satisfaisant (à déterminer)

Remarque : Il est possible d'utiliser la même variable pour différents noeuds.

2.4 Les arbres de décisions

- Ces algorithmes sont faciles à interpréter (et à visualiser)
- Tout les types de données (catégorielle, numérique, ...) peuvent être mélangés

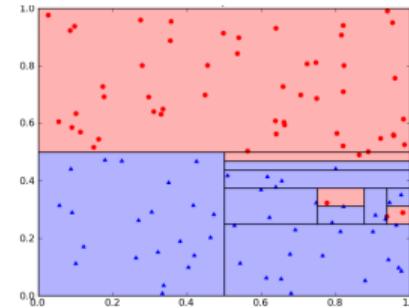
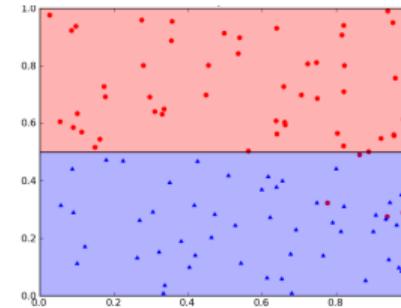
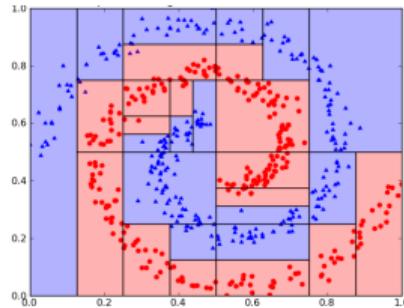


Illustration depuis University of Chicago

- Risque d'*overfitting* (sur-apprentissage) du modèle qui se généralise mal
- Algorithmes très sensibles au jeu de données d'entraînement

2.4 Les arbres de décisions : méthodes d'ensemble

- Pour pallier les faiblesses des arbres de décision, on les utilise comme *base learners* dans des méthodes qui en regroupent plusieurs
- Il existe deux familles de méthodes :
 - **averaging** : plusieurs estimateurs (*learners*) indépendants dont on moyenne les prédictions (*Random Forests*, ...)
 - **boosting** : plusieurs estimateurs combinés (*AdaBoost*, *Gradient Boosting*, ...)

2.4 Exemple d'algorithme d'*averaging* : Random Forests

- *Bagging* : Chaque estimateur (arbre) de l'ensemble est construit à partir d'un sous-échantillon aléatoire b (avec replacement) de l'échantillon d'entraînement B (*Bootstrap aggregating*)
- *Features bagging* : La meilleure séparation possible sur un sous-échantillon aléatoire des variables p (souvent : \sqrt{p})
- Soit f_b , l'arbre entraîné sur le sous-échantillon b ($b \in [1, B]$), on calcule la prédiction globale en moyennant les prédictions de tout les arbres dans le cas d'une prédiction :

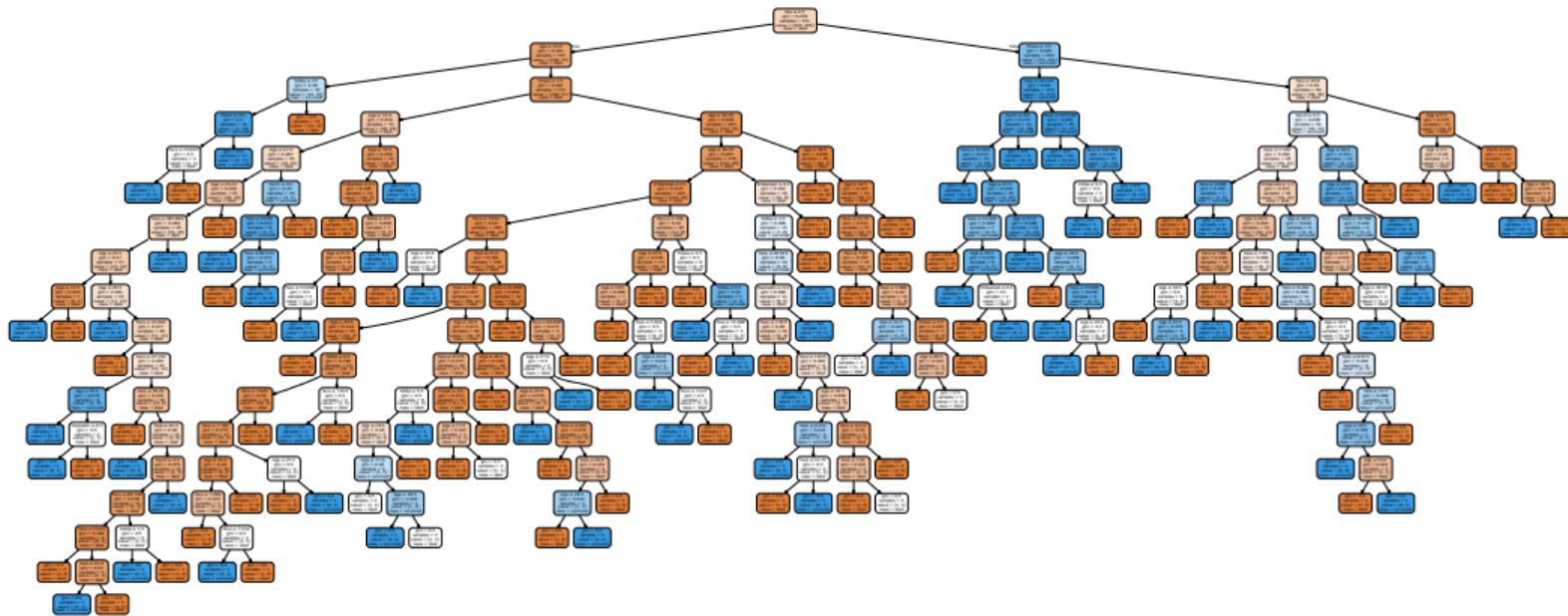
$$\hat{f} = \frac{1}{B} \sum_{b=1}^B f_b(x)$$

- Dans le cas d'une classification, on utilise le résultat du vote majoritaire des *learners*
- Cette méthode obtient de meilleure performance car elle réduit la *variance* du modèle sans accroître le *biais*

2.4 Exemple d'algorithme de *boosting* : AdaBoost

- L'apprentissage d'un même *learner* est répété en modifiant le jeu de données à chaque itération
- On utilise des poids : w_i pour $i \in [1, N]$. Initialement : $w_i = 1/N$
- À chaque itération, le poids des exemples correctement prédits diminue tandis que celui des exemples dont les prédictions sont fausse augmente
- L'algorithme devient plus sensibles aux exemples difficiles à prédire

2.4 Arbres de décision : *Titanic*



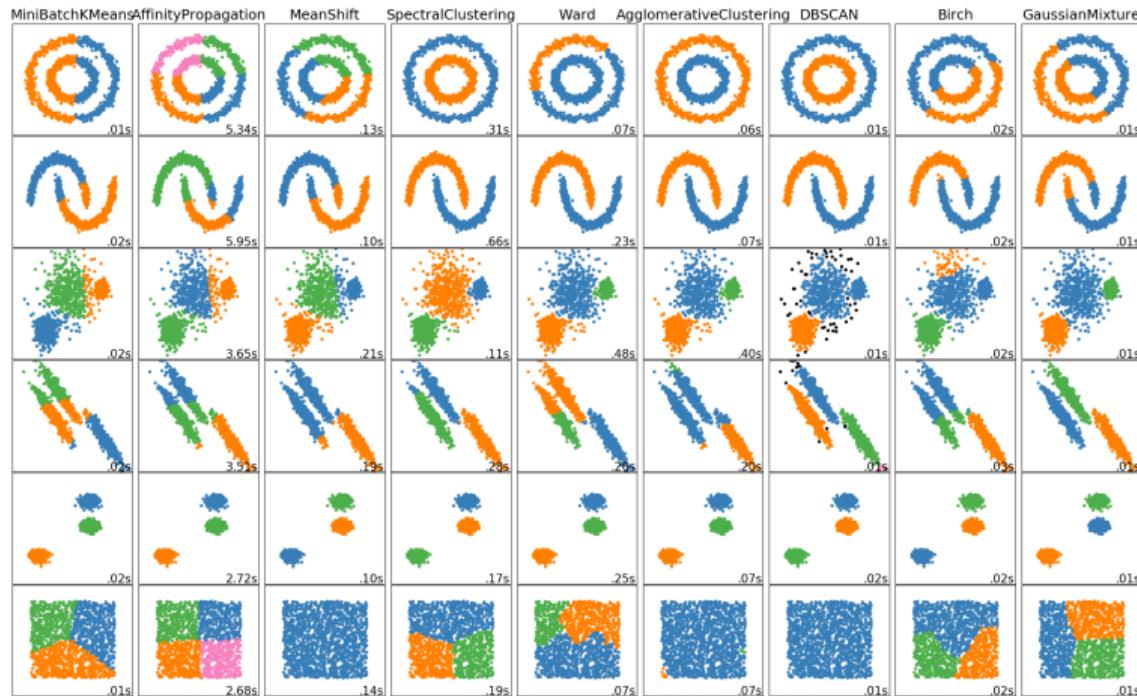
	Decision Tree	Random Forest	AdaBoost	Gradient Boosting
Accuracy (%)	77.6	81.1	83.2	86.0

- Données non *labélisées*
- La machine apprend par elle même à identifier une structure
- Évaluation des performances compliqué.
- Problèmes de classification, réduction de dimensions
- *K-means, Mean Shift, Gaussian Mixture Model*
- Analyse en Composante Principale

3.1 Les algorithmes de clustering

- *Clustering* : Se rapproche d'un problème de classification (sans labels)
- Ces algorithmes cherchent à rassembler les exemples en cluster
- À la différence des arbres de décision, le choix du cluster ne s'effectue pas par une suite de décisions simple, mais en déterminant la plus petite distance possible dans l'espace des variables
- Utilisés pour la segmentation d'utilisateurs/marchés dans le commerce en ligne mais aussi en génétique
- Il faut (dans la majorité des cas) définir au préalable le nombre de clusters à construire (hyperparamètre du modèle)

3.1 Les algorithmes de clustering



3.1 K-Means : description et limites

- Sépare les données en K clusters C_k d'égale variance (dispersion)
- L'algorithme modifie la position des *centroïdes* μ_k afin de minimiser l'écart moyen de l'ensemble des points d'un cluster à son *centroïde*
- Critère de minimisation, **Inertie** :

$$I = \sum_{k=0}^K \sum_{x \in C_k} \|x - \mu_k\|^2$$

(Distance Euclidienne)

- **Limitations** :
 - Le nombre de clusters se définit “à la main”
 - Le résultat est très dépendant de l'initialisation
 - Le **K-Means** ne fonctionne pas avec les variables catégorielles (non ordonnées)

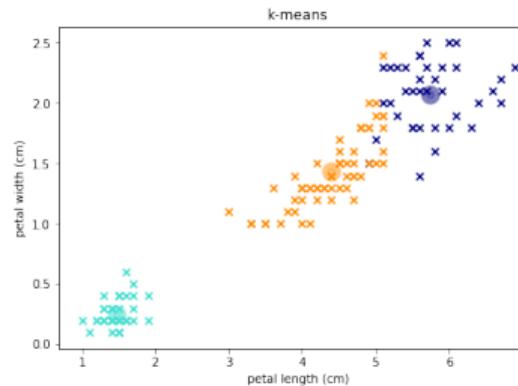
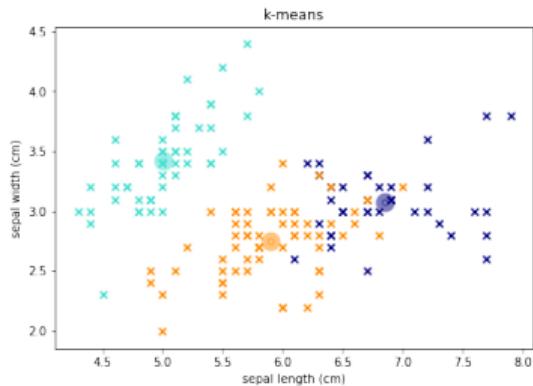
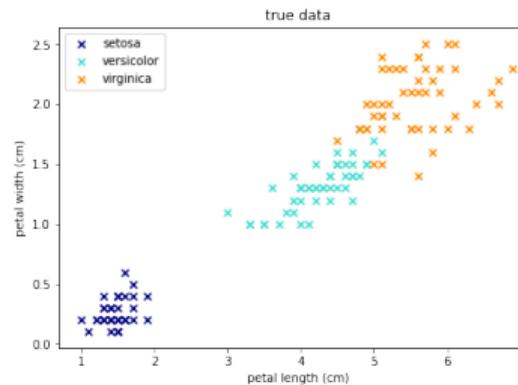
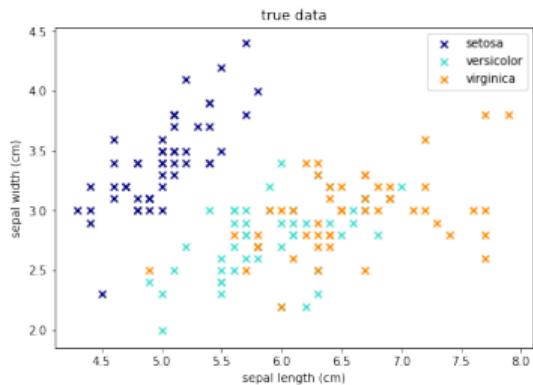
3.1 K-Prototypes

- Implémentation python : <https://github.com/nicodv/kmodes>
- Permet de mélanger variables **numériques (r)** et **catégorielles (c)**
- *centroïdes* : $\mu_k \rightarrow prototypes : Q_k$
- On redéfinit la fonction de coût :

$$I = \sum_{k=0}^K \sum_{x \in C_k} \left(\sum_{j=1}^{m_r} (x_j^r - q_{jk}^r)^2 + \gamma_k \sum_{j=1}^{m_c} \delta(x_j^c, q_{jk}^c) \right) \text{ with } \delta(p, q) = \begin{cases} 0 & \text{if } p = q \\ 1 & \text{if } p \neq q \end{cases}$$

- **Algorithme :**
 1. Initialise K prototypes (aléatoirement)
 2. Détermine le plus proche prototype de chaque entrée, et crée les clusters
 3. Met à jour les prototypes de chaque cluster en minimisant I

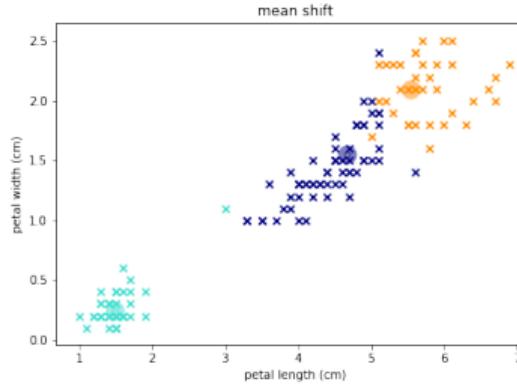
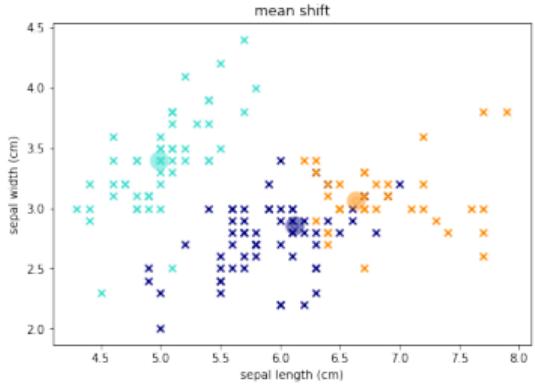
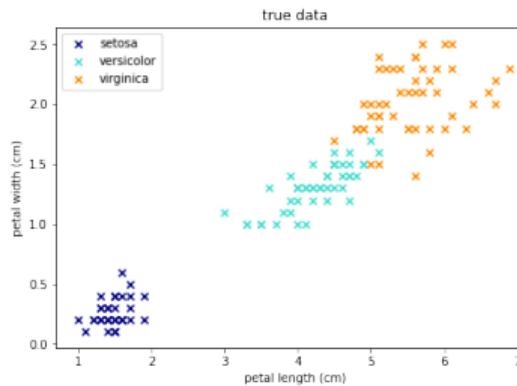
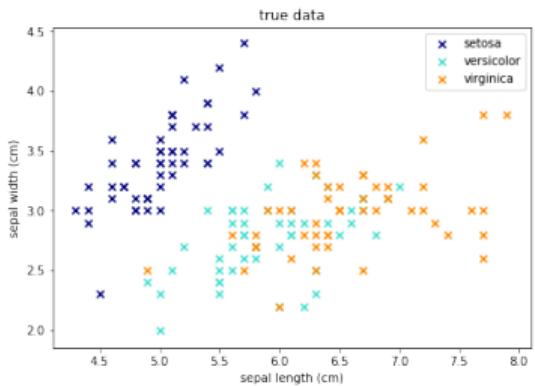
3.1 Clustering : Iris dataset, K-means result



3.1 Mean Shift

- Cherche les zones de fortes densité en modifiant itérativement la position de *centroïdes*
- Des *centroïdes* de rayons R sont aléatoirement initialisés
- Ils sont déplacés vers la région de plus haute densité (nombres de points dans le rayon R)
- On continue jusqu'à maximiser la densité de chaque *centroïde*
- Plusieurs *centroïdes* dans une zone : celui avec la plus haute densité est conservé
- l'ensemble du dataset est labelisé (plus petite distance)
- Le *Mean Shift* détermine le nombre optimal de clusters pour la valeur du rayon choisie (R)

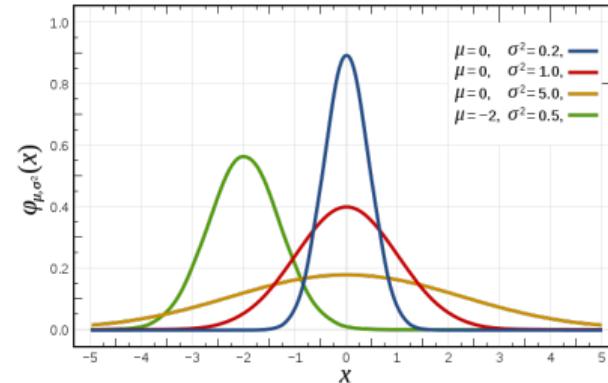
3.1 Clustering : Iris dataset, Mean Shift result



3.1 Gaussian mixture models

- **Hypothèse :** la structure des données est compatible avec un mélange de gaussiennes

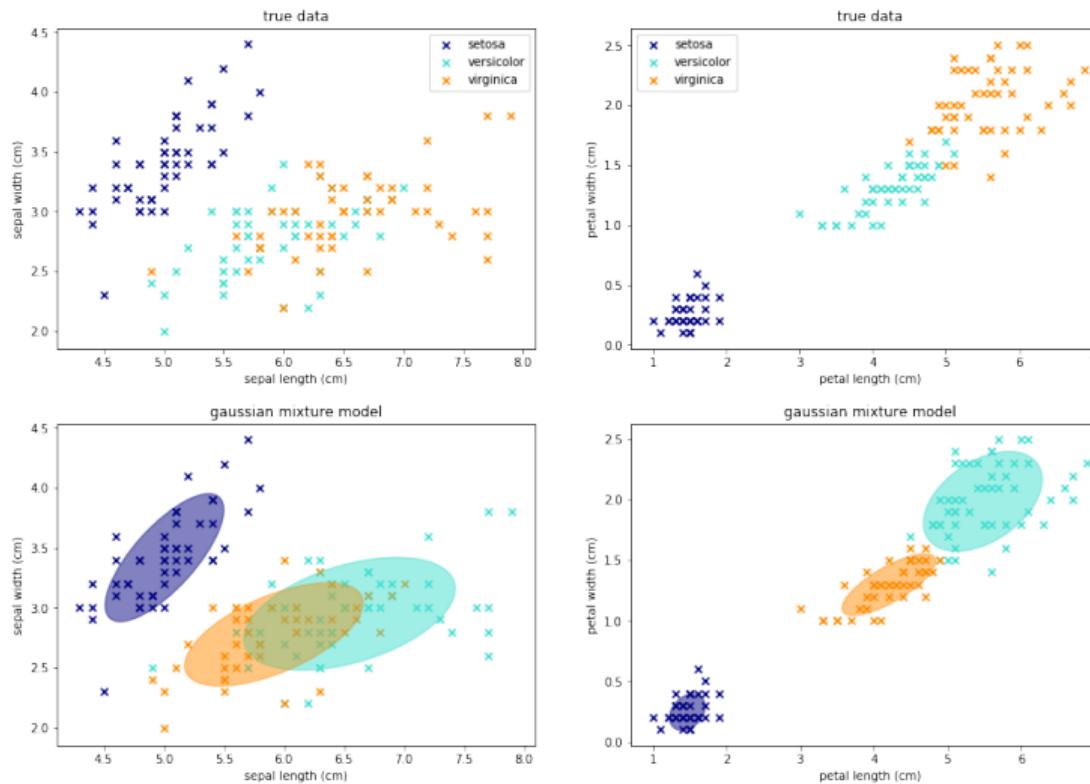
- On ajuste les paramètres de N gaussiennes jusqu'à trouver ceux qui *collent* le mieux aux données



depuis [Wikipedia](#)

- Algorithme d'ajustement : **espérance-maximisation**, on cherche à maximiser la fonction de *log-likelihood* (fonction de vraisemblance)
- *Intuitivement* : on cherche les paramètres qui rendent la distribution de données observée la plus probable

3.1 Clustering : Iris dataset, GMM result

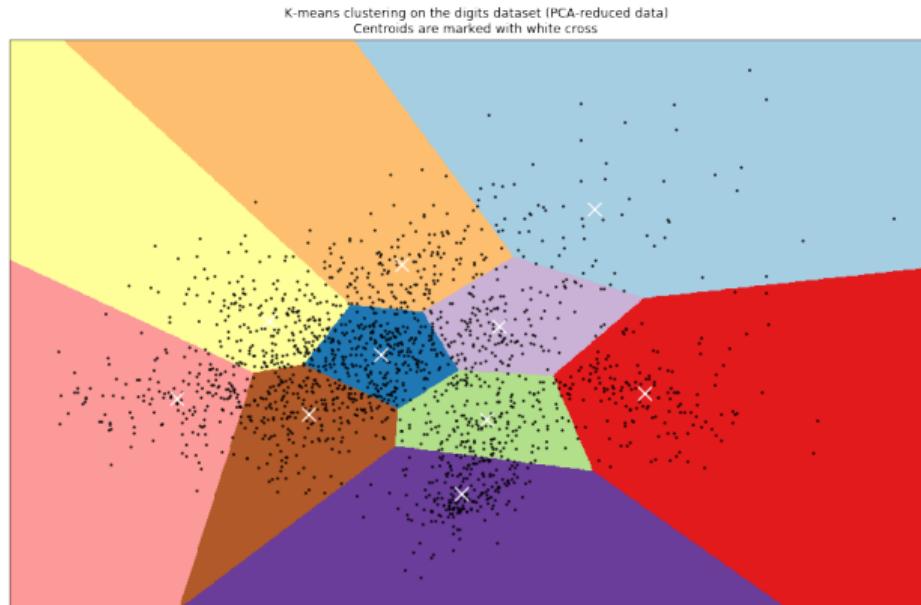


3.2 Analyse en composantes principales

- **PCA** (*Composants principal analysis*) : Algorithme de réduction de dimensions
- Transforme un problème à n variables en un problème à n' variables (avec $n' < n$)
- Explore les corrélations entre les variables pour les *regrouper* (pondération) : **Pertes d'informations**
- Très utile pour :
 - Pre-stage d'un algorithme de clustering (peut améliorer les performances)
 - Pour faire de la visualisation en 2D de données à plus haute dimension

3.2 Analyse en composantes principales

- **Exemple d'utilisation :** digit dataset :
 - 64 variables \Rightarrow 10 pour initialiser le K-means
 - 64 \Rightarrow 2 pour la visualisation



4.1 Recommandation, collaborative filtering

- Algorithme classique de recommandation de produits
- Utilise les notes donné à chaque produit par les utilisateurs
- Vainqueur du challenge Netflix (et toujours utilisé)

User / Item	Batman	Star Wars	Titanic
Bill	3	3	
Jane	1	2	4
Tom	2.38	5	

- Degré de similarité entre utilisateurs (ou produits)
 - Similarité Cosinus ou Pearson
- Estimation de l'intérêt d'un utilisateur pour un produit en se basant sur les utilisateurs similaires :
$$\text{cosSim}(\text{Bill}, \text{Tom}) = 0.71 / \text{cosSim}(\text{Jane}, \text{Tom}) = 0.32$$

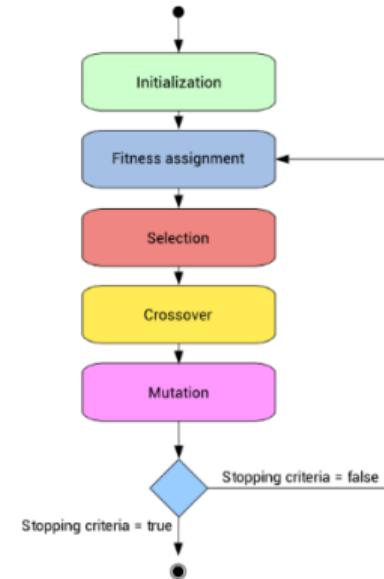
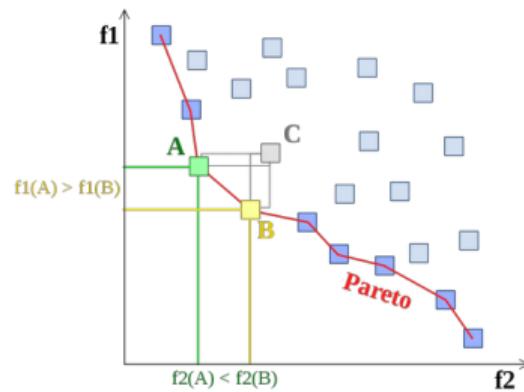
- Filtrage actif : l'utilisateur donne des notes/son avis
- Filtrage passif : Il faut déduire les goûts de l'utilisateur via son comportement
- Limites : temps de calcul, nouveaux arrivants, comportements singuliers

4.1 Recommandation, collaborative filtering, formules

- **Similarité Pearson :** $sim(x, y) = \frac{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r}_x)(r_{y,i} - \bar{r}_y)}{\sqrt{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r}_x)^2} \sqrt{\sum_{i \in I_{xy}} (r_{y,i} - \bar{r}_y)^2}}$
- **Similarité Cosinus :** $simCos(x, y) = cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \times \|\vec{y}\|} = \frac{\sum_{i \in I_{xy}} r_{x,i} r_{y,i}}{\sqrt{\sum_{i \in I_x} r_{x,i}^2} \sqrt{\sum_{i \in I_y} r_{y,i}^2}}$
- **Estimation :** $r_{u,i} = \frac{\sum_{u' \in U} sim(u, u') r_{u',i}}{\sum_{u' \in U} |sim(u, u')|}$

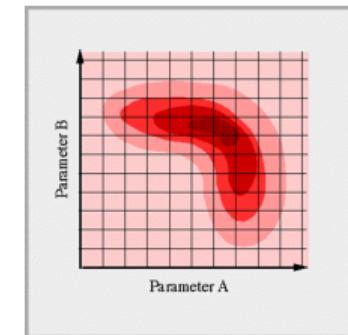
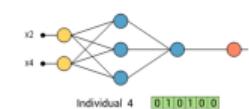
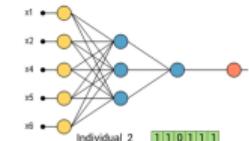
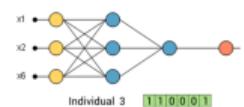
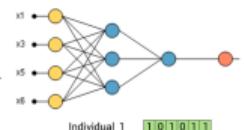
4.2 Optimisation, algorithme génétique

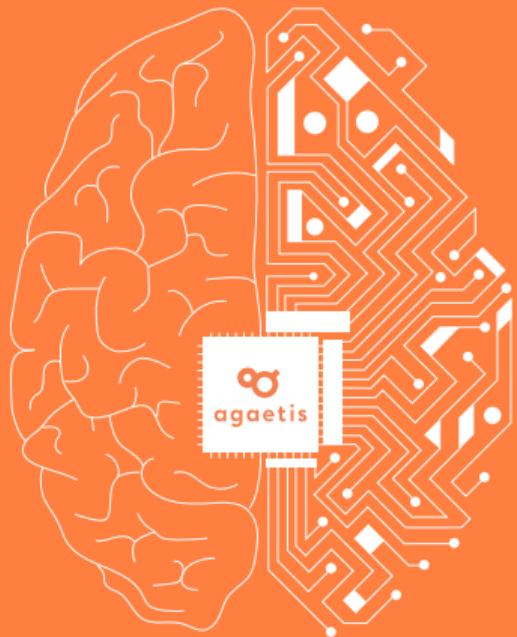
- Algorithme d'optimisation par sélection/mutation successives
 - Réponds aux problèmes à forte combinatoire sans solution unique
 - Détermine le jeu des solutions optimales (front de Pareto)
-
- **population de solutions :**
 - Initialisation → Selection → Reproduction → Mutation



4.2 Optimisation, algorithme génétique

- Applications :
 - Voyageur de commerce
 - Sélection de features
 - Hyper-paramètrage algorithme ML
 - ...





human
after
all