Introduction au Machine Learning
L3 MIASHS - IDS

Guillaume Metzler
guillaume.metzler@univ-lyon2.fr

Université de Lyon, Lyon 2, ERIC EA3083, Lyon, France

Automne 2020

Généralités

Qu'est-ce que le Machine Learning

Le *Machine Learning* ou *Apprentissage Machine* en français est une composante de l'*Intelligence Artificielle*

Les machines peuvent-elles penser ?

Les machine sont-elles capables d'effectuer les tâches, que nous, en tant qu'être humain, réalisons au quotidien ? (A. Turing)

- piloter une voiture
- créer une œuvre d'art
- ...

Tom Mitchell (1998)

On dit qu'un programme informatique est capable d'apprendre d'une expérience E correspondant à l'ensemble d'une tâche T et une d'une mesure de performance P, si sa performance vis-à-vis de la tâche T, mesurée par P, augmente avec l'expérience E.

Une définition

Machine Learning

Le *Machine Learning* est une discipline de l'informatique, sous branche de l'*IA*, qui explore à la construction et l'étude des algorithmes capables d'apprendre et d'effectuer des tâches de prédictions à partir de données.

Exemple : reconnaissance dans les images : Chat *vs* Chien

Chat vs Chien

1) Représentation des données

Images : ce sont des matrices (une matrice dont chaque entrée correspond à la valeur d'un pixel) de nombres où chaque matrice représente une intensité de couleur pour un pixel donné –> superposition de trois matrices pour trois canaux de couleurs

$$R = \begin{bmatrix} 1 \\ 1 \\ 0.9 \\ 0 \\ .. \end{bmatrix}, \qquad G = \begin{bmatrix} 1 \\ 1 \\ 0.1 \\ 0.3 \\ .. \end{bmatrix}, \qquad B = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0.7 \\ .. \end{bmatrix}.$$

Chat vs Chien

2) Construction de descripteurs

Exemples : formes ou motifs - diamètres - couleurs cependant

- le diamètre varie avec la notion de distance
- les formes sont complexes à représenter (point de vue)
- la couleur varie avec l'exposition à la lumière

Voir transformation de caractéristiques visuelles invariantes à l'échelle (SIFT) de *David Lowe (1999)* pour le calcul de descripteur sur des images.

3) Apprentissage de descripteurs

Apprendre de nouveaux descripteurs ou de nouvelles représentations de nos données.

Exemples : réseaux de neurones, auto-encodeurs, apprentissage de métrique, noyaux, ...

Chat vs Chien

Les données

Elles sont au cœur des algorithmes d'apprentissage et peuvent donc être de différentes natures :

- brutes
- transformées - créées
- apprises

Nous verrons cependant un problème majeur lié à ces données et qui est propre à l'apprentissage : **ces données sont issues d'une distribution inconnue ce qui ouvre des perspectives de recherche en Machine Learning !**

Regardons maintenant leur usage d'un point de vue pratique

Chat vs Chien

4) L'apprentissage comme un entraînement

- **Input :** nos images (sous forme vectorielle)
- **Output :** classe de l'image (chien ou chat)
- **Ensemble d'apprentissage :** notre ensemble d'images + label

Erreur d'entraînement = 1 - taux de bonne classification en entraînement

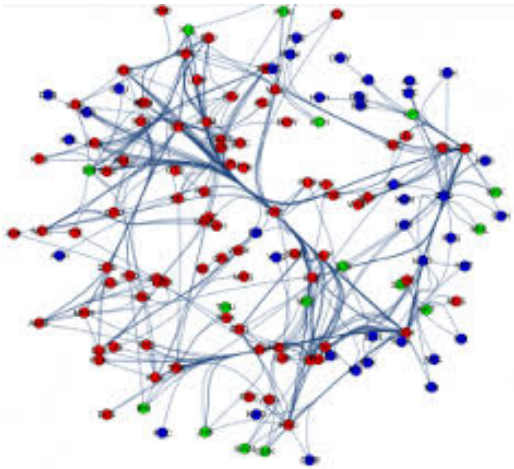
Chat vs Chien

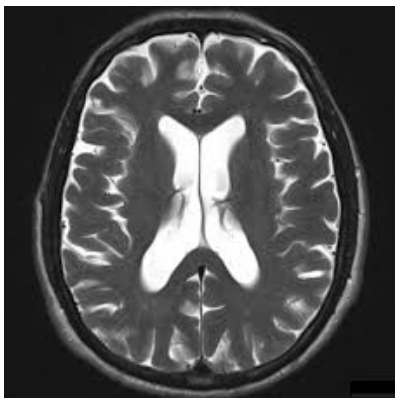
5) Généralisation sur de nouvelles données

On cherche maintenant à savoir si notre modèle est performant sur de nouvelles données ? Est-ce qu'il est capable d'identifier le genre des images non rencontrées pendant sa phase d'entraînement ?

Erreur en test= 1 - taux de bonne classification en test

Des données sous toutes les formes ...





126	10,3	0,3	0,00	16,31	0,0	
166	11,8	1,1	-0,06	10,56	0,4	
105	13,2	1,9	-0,03	11,89	1,8	
15	16,9	0,9	0,00	12,81	1,2	
6	18,7	0,4	0,12	10,92	0	

... pour des applications multiples

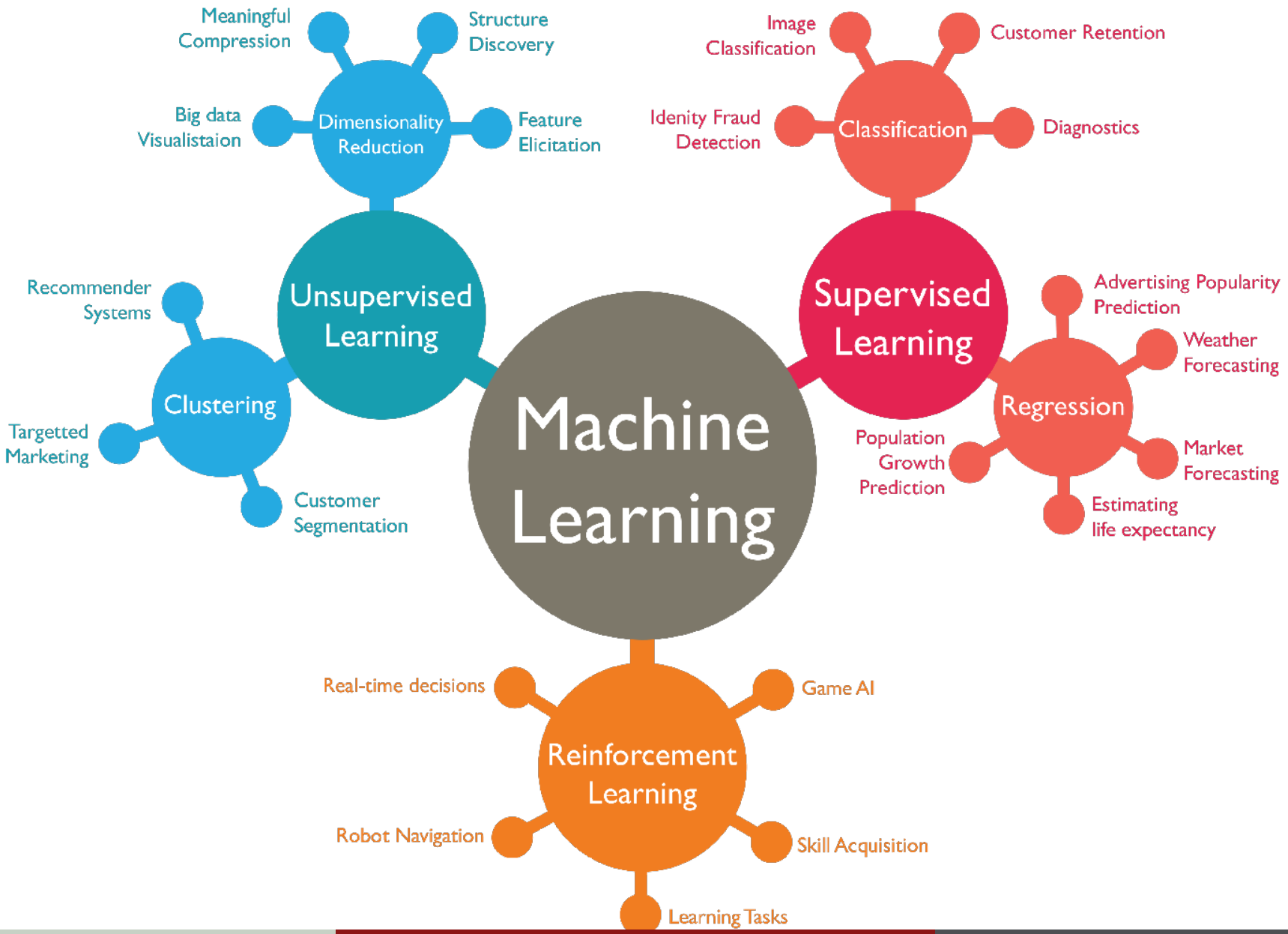
Différents contextes en Machine Learning

Nous pouvons décomposer le *Machine Learning* en trois grandes catégories principales

Catégories en Machine Learning

- **Apprentissage supervisé :** on y inclut une grande partie des algorithmes de classification (SVM - k-NN, ...) mais aussi d'apprentissage de similarité (Metric Learning, Transfert Learning) ou encore des tâches de régression
- **Apprentissage non supervisé :** on y retrouve les algorithmes de clustering (K-means ou HCA) mais aussi de l'apprentissage par transfert non supervisé
- **Apprentissage par renforcement :** ou comment apprendre de ses expériences

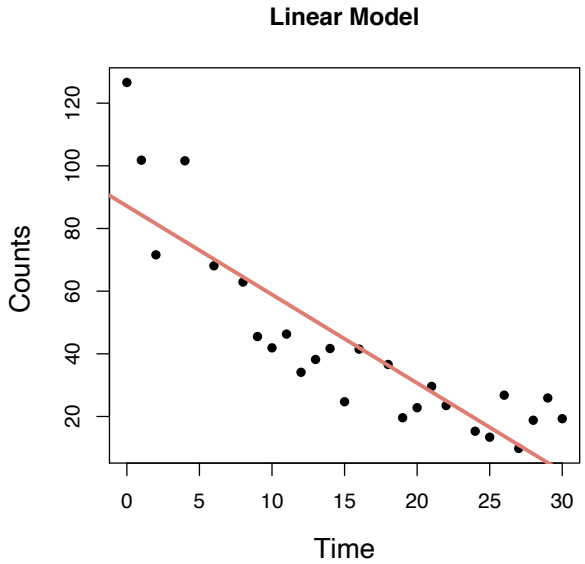
Segmentation Machine Learning

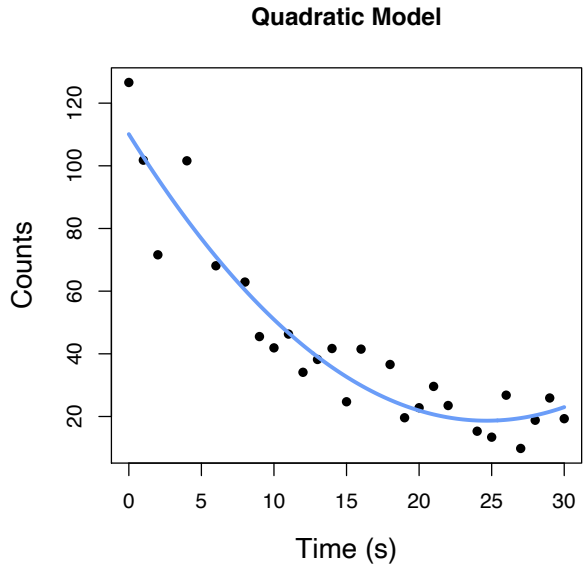


Apprentissage supervisé : Régression

Apprentissage supervisé

Le système a accès un ensemble de données décrites à la fois par des descripteurs mais également par une étiquette fournie par un "*oracle*". L'objectif sera d'apprendre une "règle" à partir des descripteurs et permettant de prédire l'étiquette de la donnée mais aussi celle de nouvelles données.



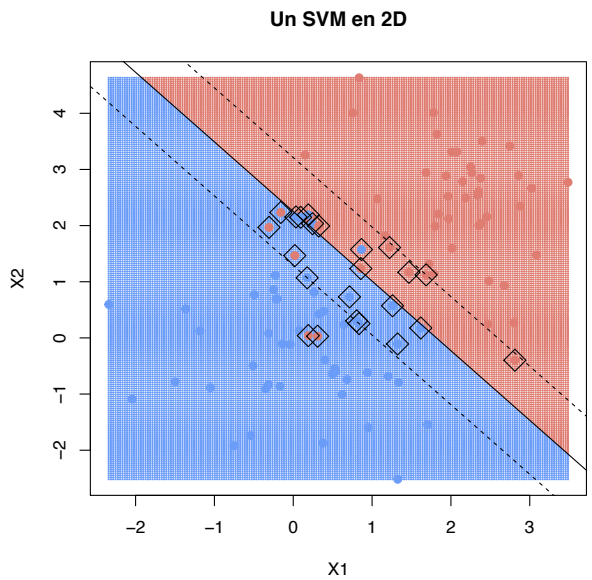


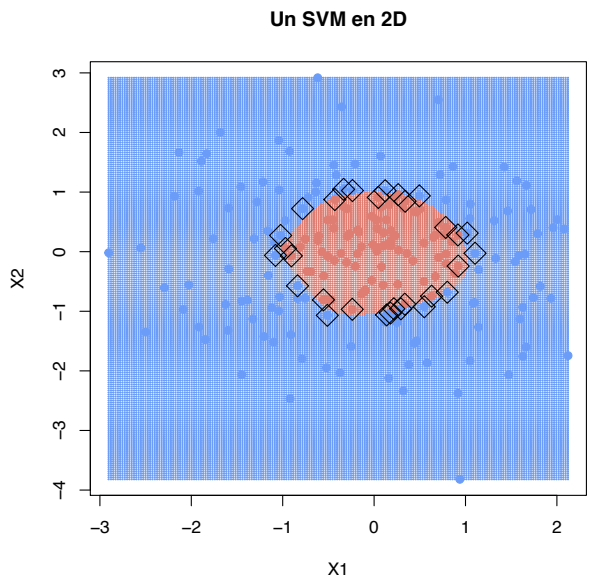
Ouput : variable continue

Apprentissage supervisé : Classification

Apprentissage supervisé

Le système a accès un ensemble de données décrites à la fois par des descripteurs mais également par une étiquette fournie par un "*oracle*". L'objectif sera d'apprendre une "règle" à partir des descripteurs et permettant de prédire l'étiquette de la donnée mais aussi celle de nouvelles données.



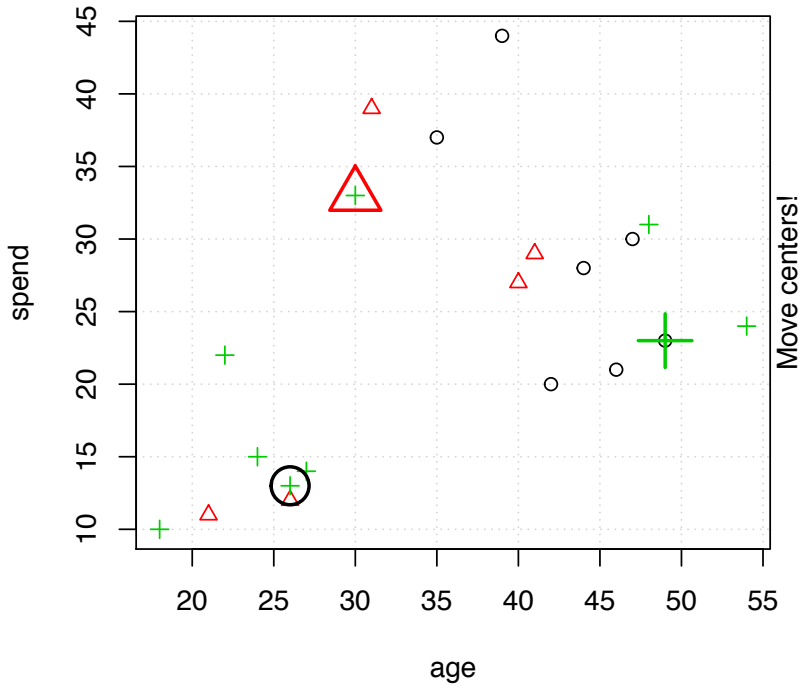


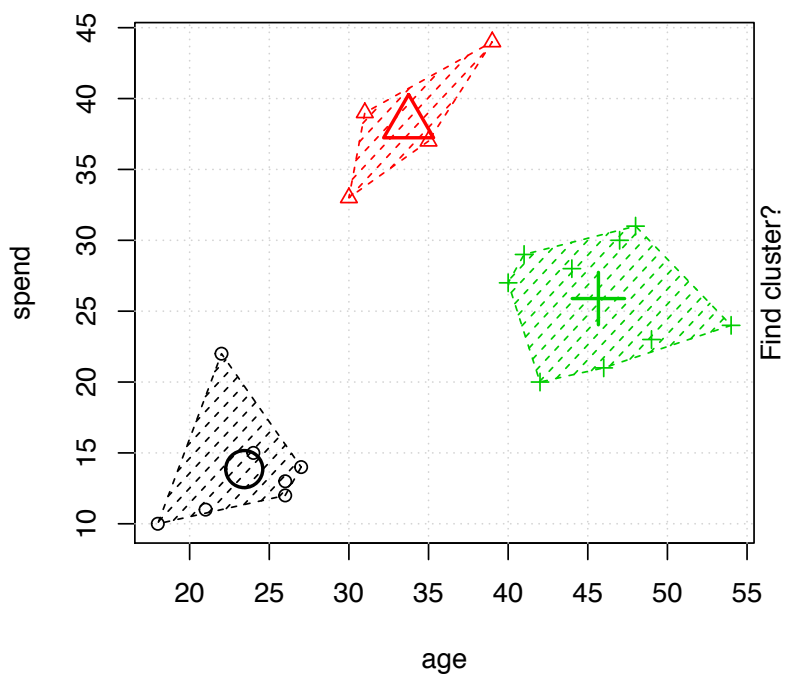
Ouput : variable discrète (étiquette(s))

Apprentissage non supervisé : Clustering

Apprentissage non supervisé

On ne dispose pas d'étiquette lors de la phase d'apprentissage. L'idée principale est de regrouper les exemples présentants des **similarités** ou des **ressemblances** afin de créer des groupes (**clustering**). On peut également utiliser cela pour faire de l'**estimation de densité** ou apprendre une **nouvelle représentation des données**.



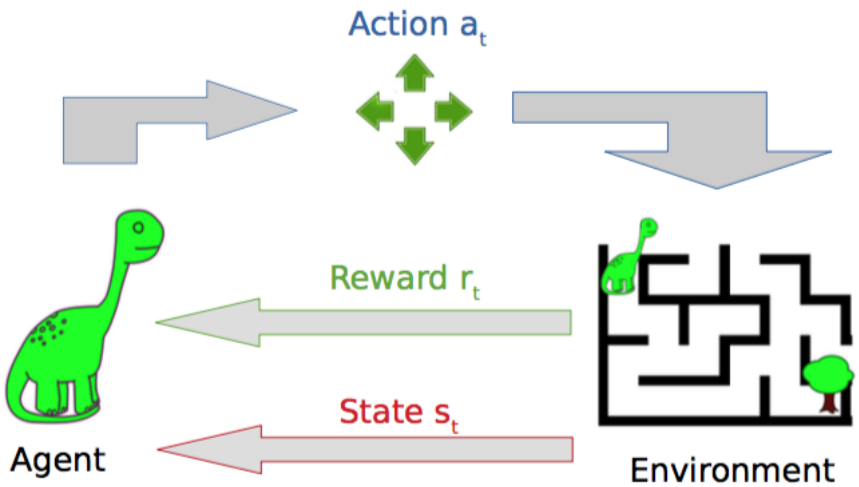


Apprentissage par renforcement

Principe

La modèle doit apprendre à effectuer les bonnes **actions** en fonction du **contexte**, *i.e.* il doit prendre les bonnes **décisions** en fonction des **observations effectuées**. Cela se fait à l'aide d'un système de **récompenses**.

Dans ce type d'apprentissage, on ne dispose de données lui indiquant les actions optimales à entreprendre selon la situation. On le laisse découvrir par lui même ce qu'il doit faire avec une méthode d'**essai-erreur**.



Petite parenthèse sur les données

Une ressource clef

Les exemples précédents montrent bien que les données sont au cœur des algorithmes d'apprentissage *Machine Learning*. Elles peuvent se trouver sous différentes formes et formats, elles peuvent être structurées ou non, parfois contenir des anomalies ou des valeurs manquantes ...

Une convention

En *Machine Learning*, il est d'usage de voir ces données comme un ensemble de m exemples ayant la même nature. La représentation la plus naturelle qui est choisie est sous forme d'un vecteur $\mathbf{x}$ en dimension d : $\mathbf{x} = (x_1, x_2, ..., x_d)$ où $\mathbf{x} \in \mathbb{R}^d$

Représentation des données

La représentation choisie, *i.e.* ce que représente la vecteur, va dépendre de la nature des données.

Les images peuvent se représenter par des vecteurs (après transformation)
données génétiques pour une séquence de gènes (dimension = longueur du gène, vecteur = encodage)
sons comme des suites de signaux (dimension = fréquence, vecteur des amplitudes)
documents textuels comme ensemble de mots (dimension = nombre de mots dans le corpus, vecteurs des occurrences)
métadonnées : auteurs, dates, ...

Les données réelles sont parfois très complexes, avec une grande redondance et présentent une grande variabilité. Il est donc nécessaire d'adopter une bonne représentation de ces dernières.

Echelles des données

Outre l'aspect représentation, il faut aussi s'intéresser aux valeurs prises par chaque descripteur de nos données afin de ne pas accorder une importance fortuite à un sous-ensemble. Cela pourrait se révéler particulièrement nocif pour certains algorithmes basés sur la notion de distance !

Exemple : Soit $\mathbf{x} = (x_1, x_2), \mathbf{x}' = (x'_1, x'_2)$ où $x_1, x'_1 \in [0, 1]$ et $x_2, x'_2 \in [500, 1000]$, alors la distance Euclidienne

$$d(\mathbf{x}, \mathbf{x}') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2}$$

est semblable à la distance entre x_2 et x'_2.

Une solution consistance alors à normaliser les données, ce qui, dans la majorité des cas, permet d'augmenter les performances des algorithmes en accordant le même poids à chaque variable.

Normalisation des données

Il existe plusieurs processus de normalisation des données dont les plus connus/utilisés sont les suivants :

- **mise à l'échelle** ou **normialisation min-max** pour que les valeurs se trouvent dans $[0, 1]$ ou $[-1, 1]$

$$x = \frac{x - \min(x)}{\max(x) - \min(x)} \quad \text{ou} \quad x = 1 - 2 \times \frac{x - \min(x)}{\max(x) - \min(x)},$$

- **standardisation** : faire en sorte que chaque feature suivent une loi normale centrée réduite

$$x = \frac{x - \mu(x)}{\sigma(x)},$$

- **normalisation** : diviser chaque vecteur par sa norme de telle sorte que $\|\mathbf{x}\| = 1$

$$\mathbf{x} = \frac{\mathbf{x}}{\|\mathbf{x}\|}.$$

Une hypothèse concernant les données

Comme évoqué plus tôt, nous n'avons jamais accès à l'ensemble de la distribution lors de l'apprentissage d'un modèle, mais seulement à un **petit échantillon**. Pour s'assurer que le modèle appris sur ces données est capable de **généraliser sur de nouvelles données**, nous devons supposer que ces dernières sont *i.i.d.*

Definition

Un ensemble d'apprentissage S est dit *i.i.d.* si tous les exemples sont issus d'une même distribution de probabilité **inconnue** $\mathcal{D}$ et qu'ils sont mutuellement indépendants.

$\rightarrow$ **une hypothèse fondamentale pour la théorie en Machine Learning**

Données et dimension

Si on dispose de suffisamment d'exemples dans un espace de dimension "limitée", on montre que nos modèles sont capables de généraliser, car les données permettent d'approximer correctement la distribution.

Quid dans le cas contraire ? Malédiction de la dimension !

Cela se traduit par l'apparition de phénomènes qui n'apparaissent pas en dimension raisonnable : comme des volumes qui deviennent anormalement faibles, et qui peuvent avoir des conséquences sur certains algorithmes.

$\rightarrow$Réduire la dimension de l'espace des données !

Qu'est-ce que l'apprentissage ?

Considérons $\mathcal{D} = \mathcal{X} \times \mathcal{Y}$ la distribution inconnue de nos données. L'espace $\mathcal{X}$ est appelé **input space** ou encore **feature space**, en général $\mathcal{X} \subset \mathbb{R}^d$. L'espace $\mathcal{Y}$ est **l'espace des étiquettes** ou encore **l'output space** :

- $\mathcal{Y} = \emptyset$: apprentissage non supervisé
- $\mathcal{Y} = \mathbb{R}$: régression
- $\mathcal{Y} = \{0, 1\}$: classification binaire
- $\mathcal{Y} = \{1, \ldots, C\}$: classification multi-classe
- $\mathcal{Y} = \{0, 1\}^C$: classification multi-label

Objectif : trouver un algorithme $\mathcal{A}$, générant une hypothèse h capable d'effectuer la tâche souhaitée.
Problème : en pratique $\mathcal{D}$ est inconnue

Apprentissage supervisé

Contexte

On dispose uniquement d'un échantillon de taille finie $S = \{\mathbf{x}_i, y_i\}_{i=1}^m \underset{i.i.d.}{\sim} \mathcal{D} = \mathcal{X} \times \mathcal{Y}$ où $\mathcal{X} \subset \mathbb{R}^d$ et $\mathcal{Y} \subset \mathbb{Z}$ (classification) ou $\subset \mathbb{R}$ (régression).
Dans la suite, on supposera que nos données $\mathbf{x}_i \in \mathbb{R}^d$ et on notera

$$X = (\mathbf{x}_1, ..., \mathbf{x}_m) = \begin{pmatrix} \mathbf{x}_{11} & \cdots & \mathbf{x}_{1d} \\ \vdots & \ddots & \vdots \\ \mathbf{x}_{m1} & \cdots & \mathbf{x}_{md} \end{pmatrix}$$

On souhaite donc trouver une **hypothèse** h, telle que $h(\mathbf{x}) = y$ qui soit performante sur notre échantillon S mais aussi sur tout nouvel exemple $(\mathbf{x}, y)$.

Comment apprendre une telle hypothèse ?

Notion de risque

Afin d'apprendre et de trouver la meilleur hypothèse $h^\star$, on a besoin de définir un critère qui permet de quantifier la qualité de l'hypothèse apprise. Ce critère va prendre la forme de **Mesure de performance** (que l'on va alors chercher à maximiser) ou plus traditionnellement de **Risque** (que l'on va chercher à minimiser).
Idéalement, on va chercher à minimiser la risque sur l'**ensemble de la distribution des données**, ce que l'on appelle le **Risque réel**.

Risque réel

Le risque réel $\mathcal{R}(h)$, encore appelé **risque en généralisation** d'une hypothèse h correspond à l'erreur moyenne (donc l'espérance) de l'hypothèse h sur toute la distribution

$$\mathcal{R}(h) = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}} \left[\mathbb{1}_{\{h(\mathbf{x})\neq y\}}\right].$$

Notion de risque

L'objectif de l'apprentissage supervisé est donc d'apprendre une hypothèse h qui va minimiser le risque réel, *i.e.* le risque sur notre distribution toute entière. Malheureusement, ce risque réel $\mathcal{R}(h)$ ne peut pas être calculé étant donné le caractère inconnu de $\mathcal{D}$.

Nous pouvons uniquement l'**estimer** ou le **mesurer** sur notre ensemble d'apprentissage. Cette estimation est appelée **Risque empirique**, noté $\mathcal{R}_S(h)$.

D'un point de vue pratique c'est donc cette quantité que l'on va chercher à minimiser ... enfin ... pas tout à fait comme nous le verrons après ! Regardons déjà sa définition.

Notion de risque

Risque emprique

Soit $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ un ensemble d'entraînement. Le risque empirique $\mathcal{R}_S(h)$, appelé aussi risque d'erreur, d'une hypothèse h correspond à l'erreur moyenne sur notre ensemble d'apprentissage, ou encore l'espérance de cette erreur sur S

$$\mathcal{R}_S(h) = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}} \left[\mathbb{1}_{\{h(\mathbf{x})\neq y\}}\right] = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}_{\{h(\mathbf{x}_i)\neq y\}} = \mathbb{P}\left[h(\mathbf{x}) \neq y\right].$$

Ce risque empirique mesure donc, dans le cas présent la probabilité que l'hypothèse h se trompe dans la prédiction effectuée pour l'exemple $\mathbf{x}$.

Fonctions de loss

Cette notion de risque d'erreur est spécifique à ce qu'on appelle la *0-1 loss* (prédiction correcte 0, erreur de prédiction 1). En revanche, ce ne sont pas les seuls loss qui sont utilisées, d'ailleurs cette dernière n'est que **très rarement utilisée en pratique**.

Loss

Une fonction de loss ℓ est une fonction $\mathcal{H} \times \mathcal{Z} \to \mathbb{R}^+$ qui sert à mesurer le désaccord entre la prédiction effectuée par l'hypothèse h, $h(\mathbf{x})$ et la valeur à prédire y. $\mathcal{H}$ est appelé espace d'hypothèse.

Erreur en classification ou 0-1 loss

$$\ell(h(\mathbf{x}), y) = \begin{cases} 1 & \text{if } h(x) \times y < 0, \\ 0 & \text{sinon.} \end{cases}$$

Surrogates

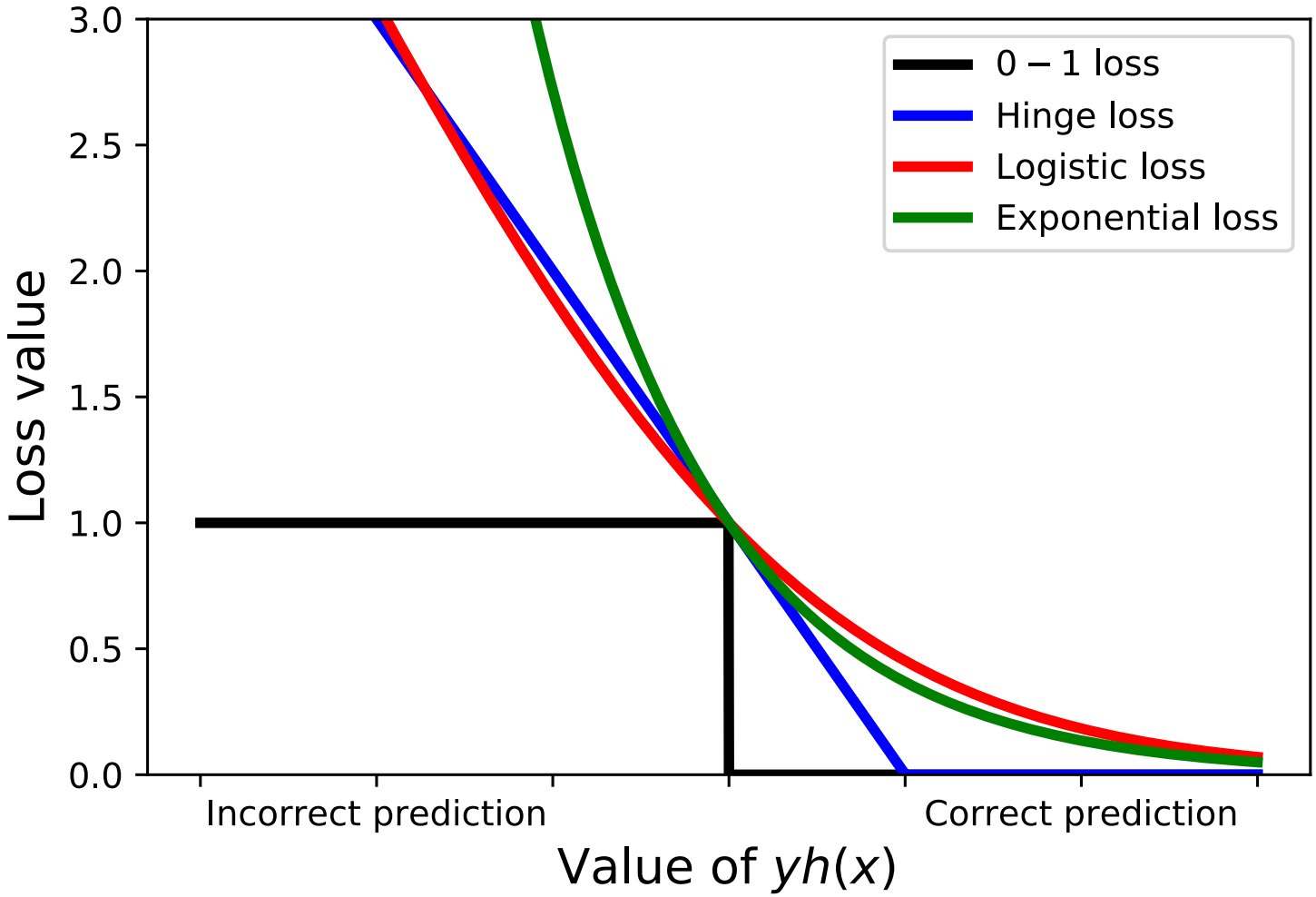
Cette loss présente de nombreux inconvénients : elle n'est pas convexe, elle n'est pas dérivable, donc peu intéressante pour des **algorithmes d'optimisation**. Le problème de minimisation avec une telle loss est connue pour être NP-hard (Ben-David et al., 2003).

$\rightarrow$ **utilisation de surrogate, des fonctions dites de substitutions, de la 0-1 loss qui présentent l'avantage d'être convexe et parfois même différentiable.**

On utilisera des surrogates différentes en fonction de l'algorithme que l'on souhaite utiliser.

Surrogates : quelques exemples

Ces surrogates se présentent comme des bornes supérieures de la 0-1 loss.



Surrogates : quelques exemples

Surrogates à la 0-1 loss

- **la hinge loss :** on la rencontre plus particulièrement lors de l'apprentissage de modèles comme les *SVM* (classification ou régression)

$$\ell(h(\mathbf{x}), y) = \max(0, 1 - yh(\mathbf{x})),$$

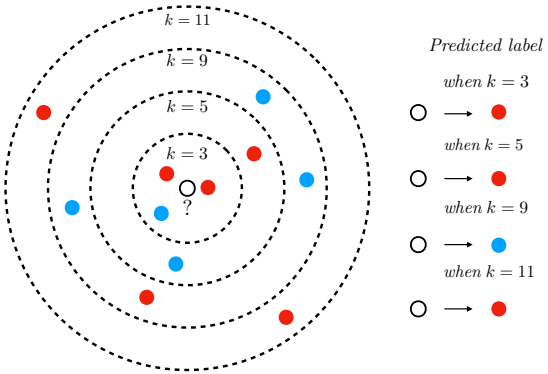
- **la loss logistique :** que l'on rencontre lors de l'utilisation d'un modèle de *régression logistique* (classification)

$$\ell(h(\mathbf{x}), y) = \frac{1}{\ln(2)} \ln\left(1 + \exp(-yh(\mathbf{x}))\right),$$

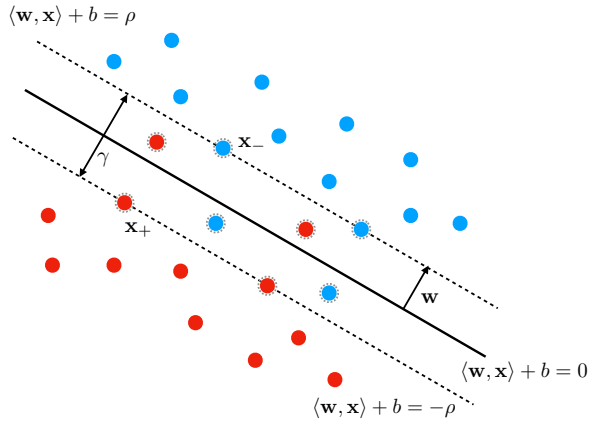
- la **loss exponentielle :** on l'a rencontre surtout dans des contextes de *boosting*

$$\ell(h(\mathbf{x}), y) = \exp(-yh(\mathbf{x})).$$

Des algorithmes



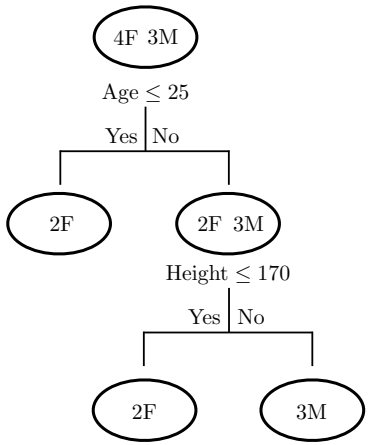
Plus proches - voisins (k-NN)



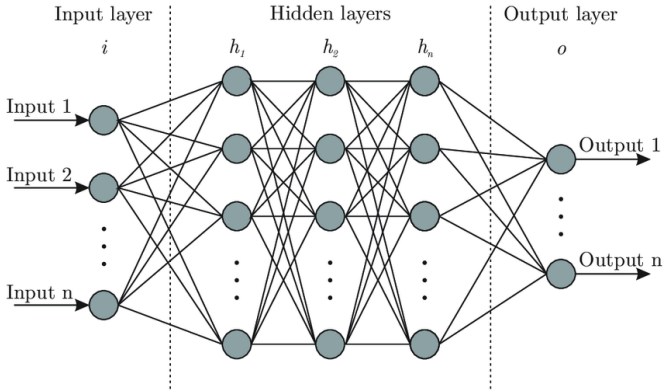
Séparateur à Vaste Marge (SVM)

Toy dataset

Age	Height	Sex
20	175	F
32	180	M
40	175	M
28	172	M
22	165	F
40	169	F
70	170	F



Arbres de décisions (DT)



Réseaux de neurones (NN)

Qu'est-ce qu'un bon modèle ?

Maintenant que l'on dispose de données S, d'une loss ℓ et d'un algorithme, comment savoir si notre modèle est performant ?
Pour cela on va chercher à minimiser le risque empirique $\mathcal{R}_S(h)$

$$\mathcal{R}_S(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h(\mathbf{x}_i), y).$$

et tenter d'estimer le risque réel $\mathcal{R}(h)$ ou risque en généralisation

$$\mathcal{R}(h) = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}} \left[\ell(h(\mathbf{x}), y)\right].$$

Mais on ne connait pas la distribution ... donc va utiliser **un processus de validation** !

Cross validation

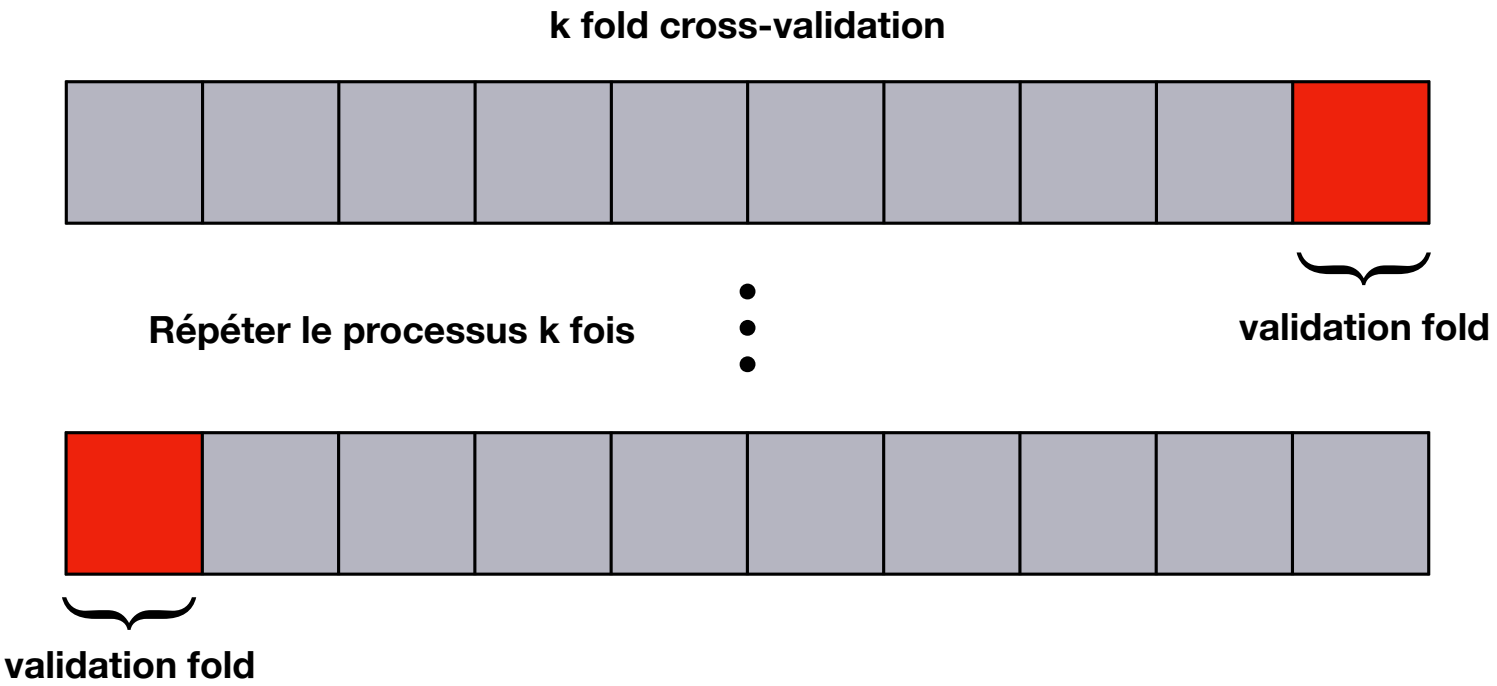
Il s'agit de conserver une partie des données de l'ensemble d'entraînement qui ne sera pas utiliser pour apprendre les paramètres de notre modèle, on les utilsera pour faire une première évaluation (entraînement - validation) mais on peut faire encore mieux.

Cross validation

Il s'agit d'un moyen algorithmique permettant d'estimer les performances en généralisation d'un algorithme donné sur des données inconnues.
Le principe est simple, on sépare nos données en k groupes et on utilise $k-1$ groupes pour apprendre notre modèle et le dernier groupe pour l'évaluer.

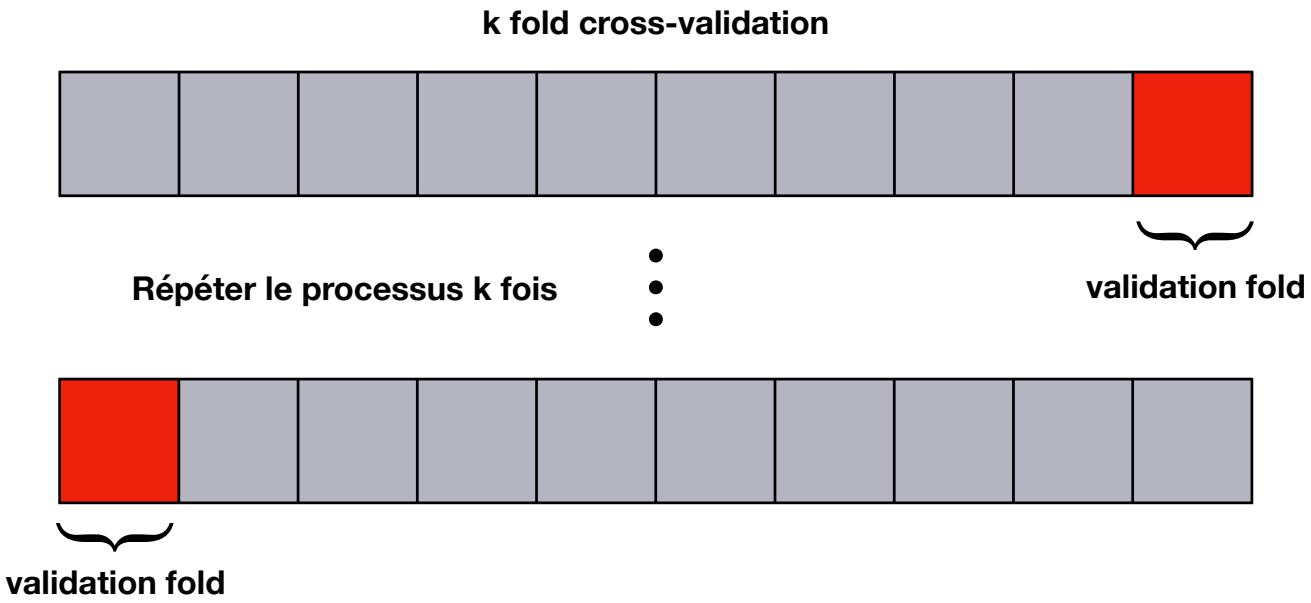
k-fold Cross validation

On effectue une série de k expériences en utilisant une validation classique, mais on change l'apprentissage et la validation à chaque run.



Au final, notre algorithme sera donc évalué sur l'ensemble des données disponibles, ce qui permet une estimation plus fiable.

k-fold Erreur

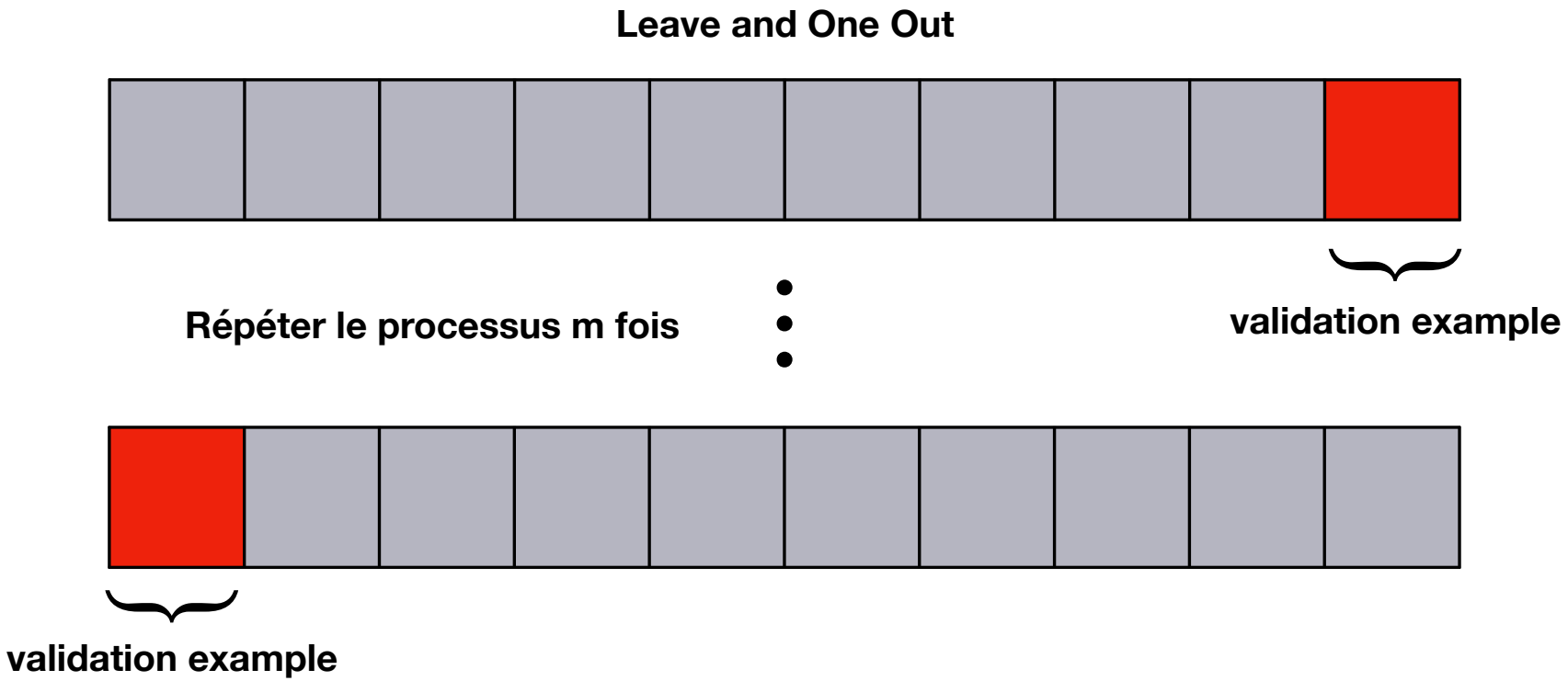


Erreur moyenne en CV

L'évaluation de $\frac{1}{k}\sum_{j=1}^{k}\mathcal{R}_{S_j}(h)$, appelée erreur moyenne en cross-validation permet de quantifier à quel point notre algorithme est capable de généraliser sur des nouvelles données issues de la même distribution. On verra que ce n'est pas son seul usage ...

Leave and One Out

Le principe reste le même mais on utilise cette fois-ci un seul exemple pour valider à chaque run, on doit effectuer un plus grand nombre d'apprentissage (m au lieu de k).



On retient le modèle ayant les meilleurs résultats en moyenne sur les m-folds qui servent à la validation

En ensuite ... ?

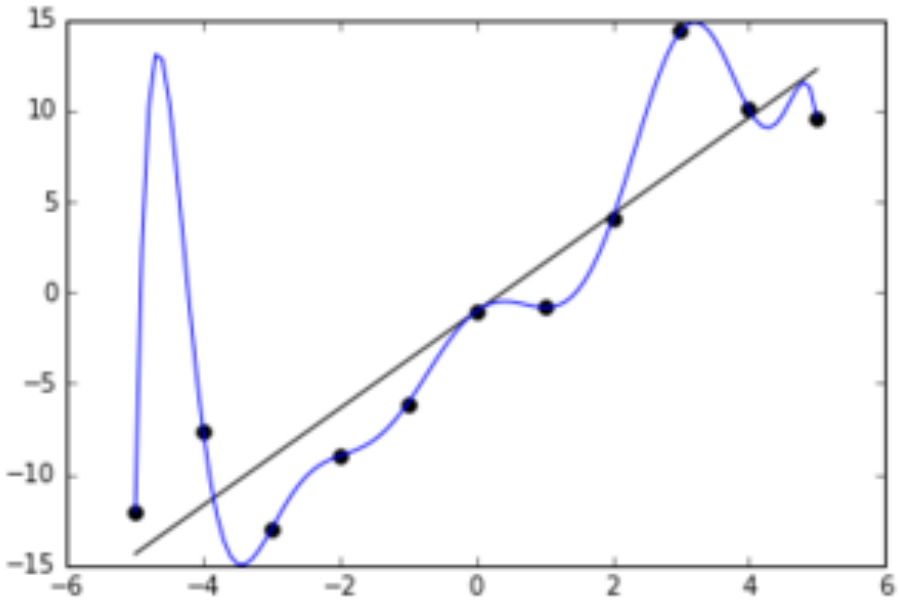
On dispose de données S, d'une loss ℓ et d'un algorithme et d'un processus permettant de sélectionner le meilleur modèle ... donc a priori on a tout ce qu'il faut pour se lancer dans la pratique (modulo la présentation des modèles).

En fait non ... avant il nous reste une dernière chose à regarder/étudier ! Qu'est-ce qui se passe si l'erreur observée sur l'ensemble d'apprentissage est sensiblement différente de l'erreur observée en validation ?

- si l'erreur en validation est **plus faible** que l'erreur sur l'ensemble d'entraînement,
- ou inversement, si l'erreur en validation **est plus** grande que l'erreur sur l'ensemble d'entraînement.

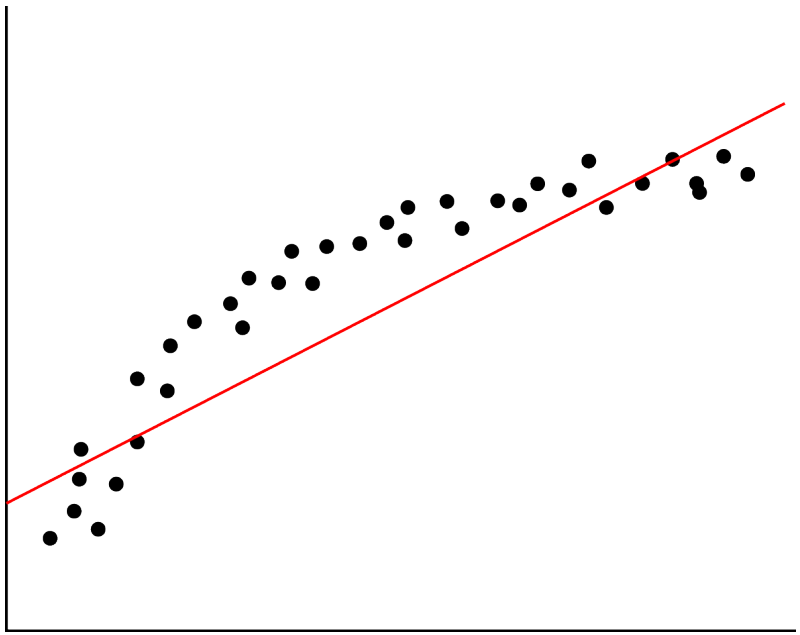
Overfitting ou Sur-Apprentissage

En statistiques, l'overfitting survient quand le modèle se focalise tellement sur les données qu'il cherche également à apprendre le bruit présent. Ce phénomène apparaît lorsque **le modèle appris est trop complexe** ou lorsque **l'ensemble d'entraînement est trop petit** par rapport au nombre de paramètres de votre modèle (degrés de liberté).

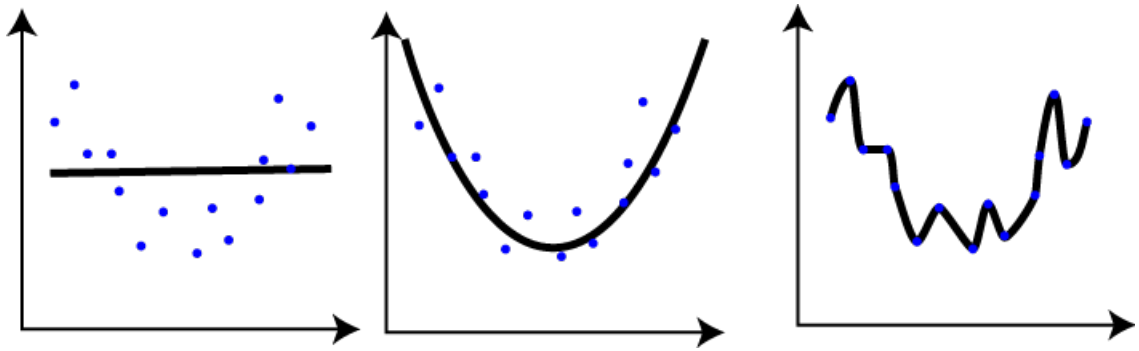


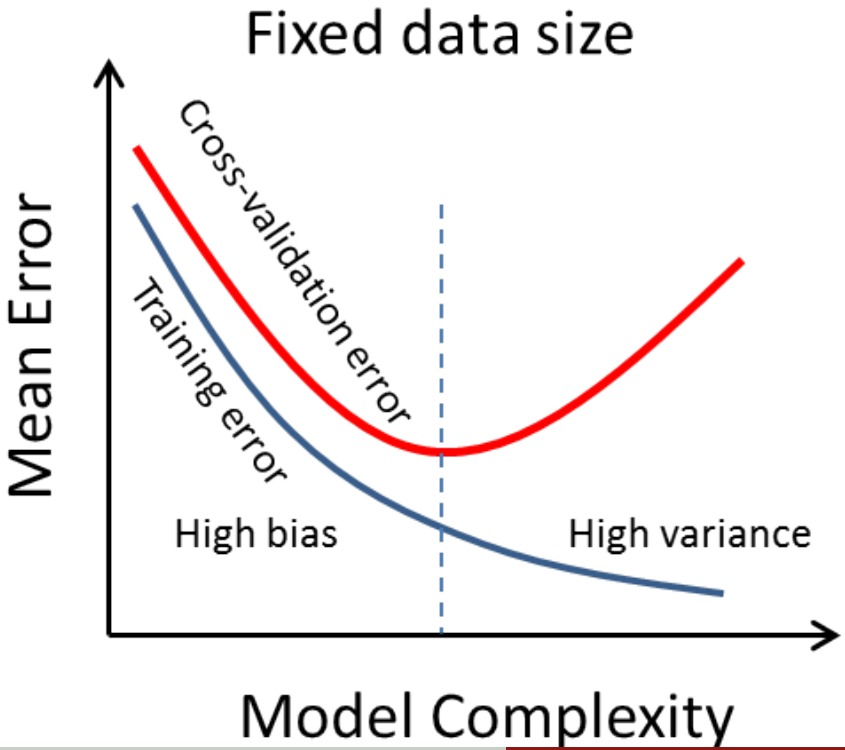
Underfitting ou Sous-Apprentissage

L'underfitting apparaît lorsque le modèle utilisé n'est pas capable d'appréhender la tendance présente dans les données. Cela survient surtout lorsque le modèle utilisé est beaucoup trop simple, comme l'utilisation d'un modèle linéaire pour approximer une courbe quadratique



Underfitting-Overfitting





Comment savoir s'il y overfitting ou underfitting ?

si l'erreur d'entraînement << erreur en CV, **overfitting**

si l'erreur d'entraînement >> erreur en CV ou simplement élevée, **underfitting**

Que faire ?

Pour traiter l'underfitting on peut tout simplement changer la classe de modèle que l'on apprend. En revanche pour l'overfitting, il va falloir trouver un moyen de *simplifier le modèle appris*.

Le rasoir d'Occam

Il est également appelé principe de simplicité ou de parcimonie, il consiste à trouver l'explication (le modèle) le plus simple permettant d'expliquer les données :

no sunt multiplicanda entia praeter necessitatem" (William of Ockham)

Que faire ?

Pour traiter l'underfitting on peut tout simplement changer la classe de modèle que l'on apprend. En revanche pour l'overfitting, il va falloir trouver un moyen de *simplifier le modèle appris*.

Le rasoir d'Occam

Il est également appelé principe de simplicité ou de parcimonie, il consiste à trouver l'explication (le modèle) le plus simple permettant d'expliquer les données :

no sunt multiplicanda entia praeter necessitatem" (William of Ockham)

Cela passe par l'ajout d'un terme de Régularisation dans notre problème d'optimisation.

Régularisation

L'introduction d'un terme de régularisation dans notre problème d'optimisation

$$h^\star = \arg\min_{h \in \mathcal{H}} \mathcal{R}_S(h).$$

Définition

Une régularisation est un terme, en statistiques et plus particulièrement en Machine Learning qui fait référence à l'**ajout d'un terme additionnel dans un problème d'optimisation** permettant d'éviter le sur-apprentissage. Il prend la forme d'une fonction qui va **pénaliser les modèles trop complexes** : il va donc chercher à lisser les modèles pour les rendre plus lisses ou encore restreindre les valeurs que peuvent prendre les paramètres d'un modèle.

Risque empirique régularisé

Le nouveau problème d'optimisation s'écrit alors :

$$h^\star = \arg\min_{h \in \mathcal{H}} \mathcal{R}_S(h) + \lambda \|h\|,$$

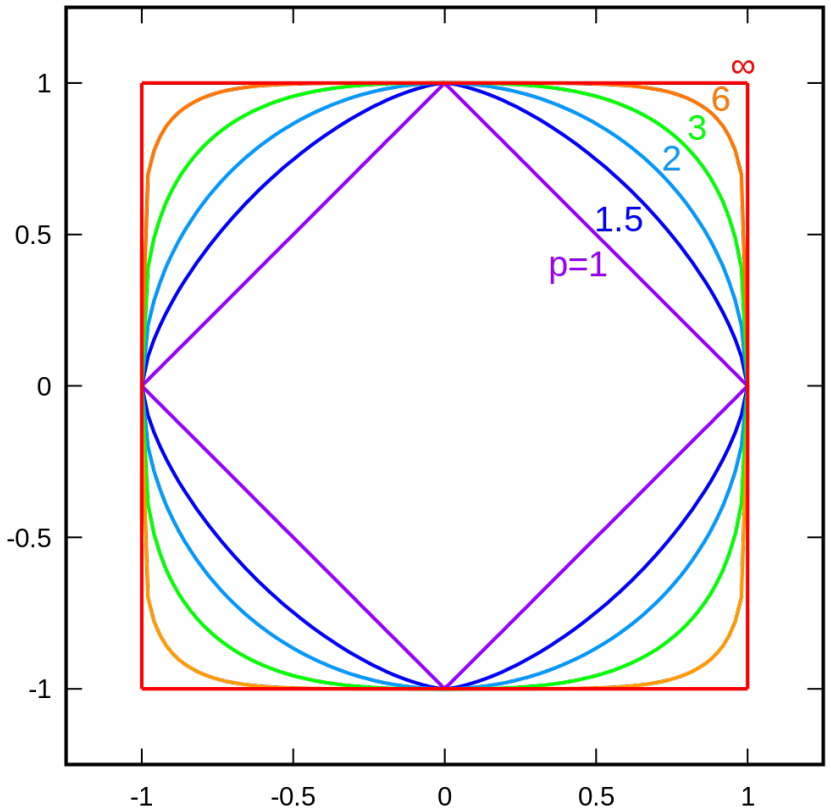
où

- λ est un paramètre de régularisation (on l'appelle aussi hyper-paramètre pour les distinguer des paramètres du modèle h),
- $\|\cdot\|$ est une fonction norme.

On va donc retourner l'hypothèse qui est capable d'atteindre le meilleur compromis entre performance et complexité.

Normes usuelles

$$\|h\|_p = \left(\sum_{i=1}^{d} h_i^p \right)^{\frac{1}{p}}$$

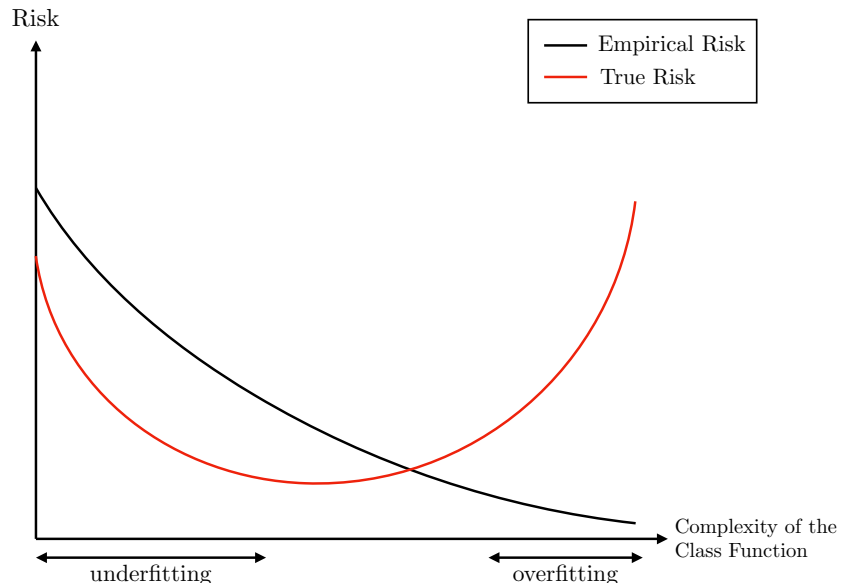


Rôle et choix du paramètre λ

Retournons à notre problème d'optimisation

$$h^\star = \arg\min_h \sum_{i=1}^{m} \ell(h(\mathbf{x}), y) + \lambda \|h\|,$$

où λ : paramètre de régularisation, contrôle la complexité de notre hypothèse, indirectement, on dit aussi qu'il contrôle la richesse de notre espace d'hypothèses $\mathcal{H}$.



Choisir λ qui se trouve à l'intersection des deux courbes ! Mais comment ?

Choix de λ et ... retour à la validation croisée

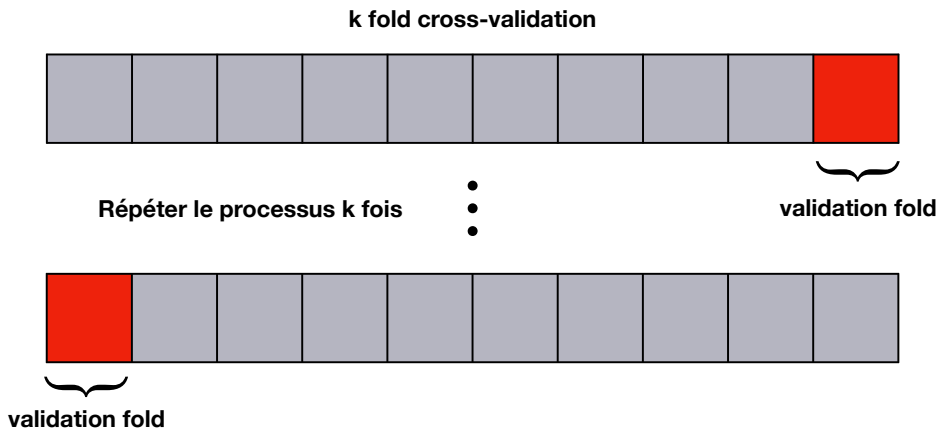
Notre ensemble d'apprentissage peut aussi servir à vérifier les capacités qu'à notre modèle **à généraliser pour le choix d'un hyper-paramètre en particulier**

- **Validation standard :** on sépare notre échantillon d'apprentissage en deux ensembles *train/valid* (2/3 - 1/3)
- **k-fold cross validation :** partition de l'échantillon en k ensembles. $k-1$ servent à apprendre et 1 sert à la validation
- **Leave and One Out :** on apprend sur tous les exemples sauf 1 qui sert à valider le modèle

k-fold cross validation comme processus de tunage

Le processus se fait de la façon suivante :

- on se fixe une grille de valeurs pour l'hyper-paramètre à optimiser : $\lambda \in (\lambda_1, \lambda_2, ..., \lambda_p)$
- pour chaque valeur de λ_i on calcule notre erreur moyenne de cross validation à l'aide d'une k-CV : $\frac{1}{k} \sum_{j=1}^{k} \mathcal{R}_{S_j}(h, \lambda_i)$



- on conserve la valeur de λ_i pour laquelle l'erreur moyenne en CV est la plus faible.

Quelques remarques

- Pour éviter le problème de sur-apprentissage on utilise à la fois un (ou plusieurs) terme de régularisation dans notre problème d'optimisation
- On combine cela à de la k-fold cross validation (on prendra communément $k = 10$) afin de s'assurer que le modèle généralise bien
- **Attention :** ces approches sont empiriques ! Pour bien faire, il faudrait étudier les garanties en généralisation

$$|\mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}} \ell(h(\mathbf{x}), y) - \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim S} \ell(h(\mathbf{x}), y)| \leq \mathcal{O}(\sqrt{1/m}, \lambda^{-1})$$

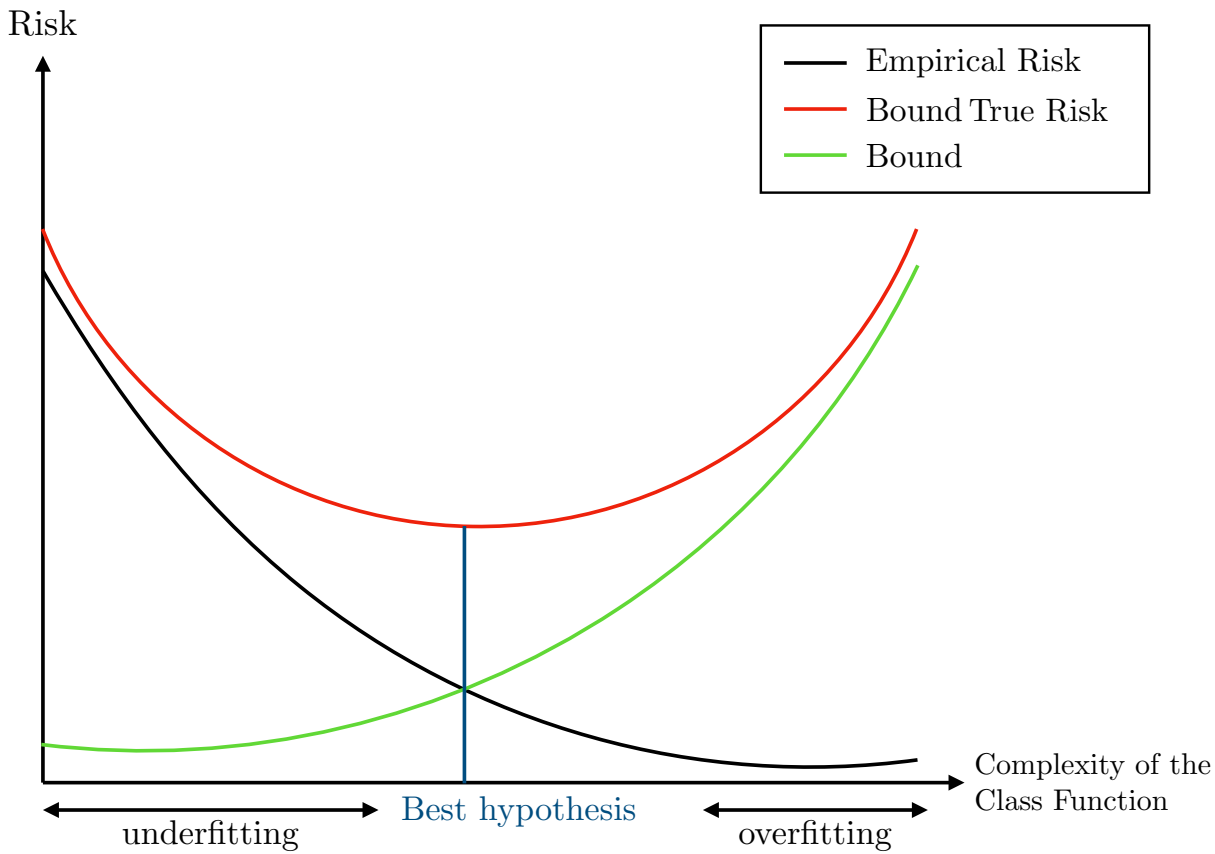
- Il faut parfois faire la distinction entre le problème d'optimisation et le critère de performance que vous souhaitez optimiser.

Bornes en généralisation

Les bornes en généralisation reposent sur :

- des inégalités de concentration, comme l'inégalité de Azuma-Hoeffding (nous verrons cela au cours d'un TD)
- des propriétés de la loss : convexité, caractère lispchitz ou α-elliptique, ...
- la complexité de notre espace d'hypothèses $\mathcal{H}$ qui peut se traduire par le choix de l'hyper-paramètre dans notre terme de régularisation ou des mesures de complexité : **VC-dim** ou **Complexité de Rademacher**
- une mesure de divergence entre une distribution *a priori* et *a posteriori* de nos hypothèses $\rightarrow$ **KL-divergence**
- ou encore la notion de **robustesse algorithmique** (Xu and Mannor, 2010)

Bornes en généralisation



Pour aller plus loin (Valiant, 1984; Bousquet and Elisseeff, 2002; Bousquet et al., 2004; Mohri et al., 2012)

Quelques algorithmes

Sommaire

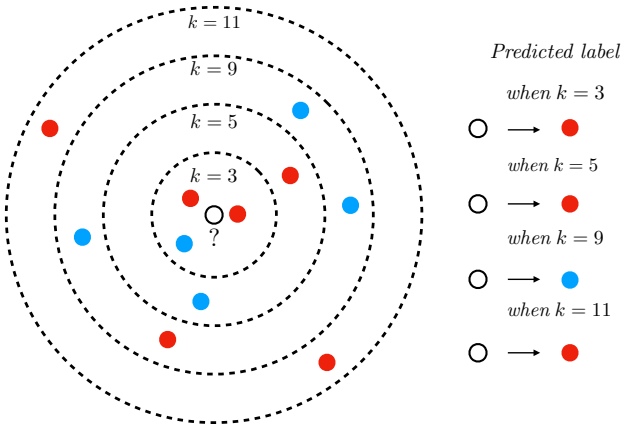
On va rapidement présenter quelques algorithmes assez simples :

- l'algorithme du plus proche voisin
- la régression linéaire (simple et multiple
- la régression logistique
- les séparateurs à vaste marge
- l'algorithme k-means

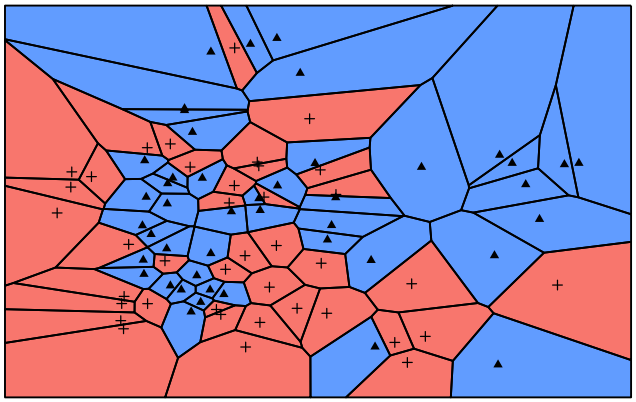
Algorithme du plus proche voisin (k-NN)

$k-$NN

Il s'agit d'un algorithme non paramétrique utilisé pour effectuer des tâches de classification (mais aussi de régression) (Cover and Hart, 1967).



Exemples $k-$NN



Régions de Voronoï, $k = 1$

Il s'agit, en outre, d'un algorithme de classification basé sur la règle de Bayes, i.e. une donnée est prédite comme appartenant à la classe $c^\star$ si

$$c^\star = \arg\max_{c \in \mathcal{Y}} p_k(y = c \mid \mathbf{x}).$$

$k-$NN

Règle de classification

La classe assignée à un nouvel exemple $\mathbf{x}_i$ va dépendre de son positionnement dans l'espace des données par rapport aux données de l'ensemble d'entraînement. Plus précisément, on lui assigne la même étiquette que l'étiquette majoritaire dans son $k-$voisinage

$$c^\star = \arg\max_{c \in \mathcal{Y}} \frac{k_c}{k},$$

ou k_c désigne le nombre de voisins appartenant à la classe c.

- **Apprentissage :** nécessite de garder en mémoire tous les exemples d'entraînement (mémoire en $\mathcal{O}(m)$)
- **Prédiction :** nécessite de calculer la distance à tous les exemples d'apprentissages (complexité $\mathcal{O}(md)$) puis d'ordonner les plus proches voisins

$k-$NN

Cet algorithme repose donc sur la notion de **distance**. On utilise couramment la distance euclidienne $\|\mathbf{x} - \mathbf{x}'\|_2 = \sqrt{\sum_{j=1}^{d}(x_j - x'_j)^2}$, mais nous pourrions utiliser n'importe quelle norme ℓ^p.
Mieux encore ! On pourrait définir sa propre distance $\rightarrow$ **Metric Learning**

Remarque

Bien que très simple d'utilisation, cette méthode présente rapidement quelques limites :

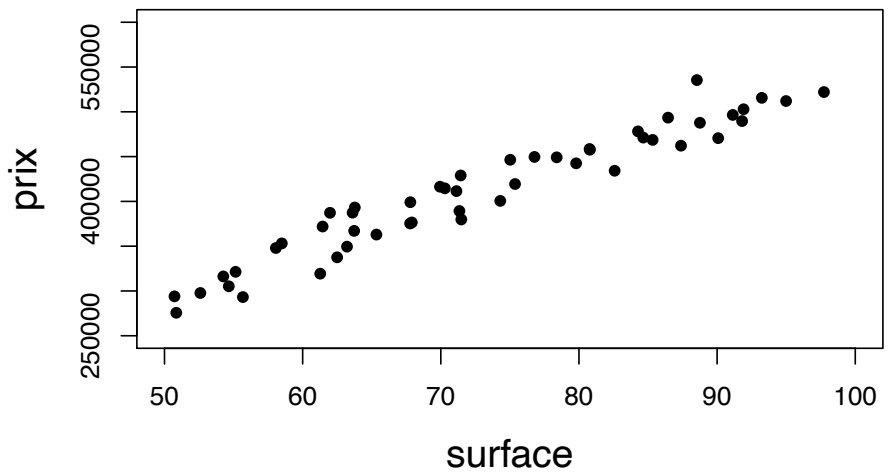
- en grande dimension, tous les exemples seront rapidement éloignés (l'espace est très rapidement vide)
- un temps de calcul qui augmente linéairement avec le nombre d'exemples

Régression linéaire (LM)

Contexte

On considère ensuite un ensemble de données dont les descripteurs sont entièrement numériques, *i.e.* on dispose de données de $(x_i)_{i=1}^m \in \mathbb{R}^d$ et **la variable à prédire y_i est un nombre réel**, *i.e.* $y_i \in \mathbb{R}$.

Exemple : lorsque l'on cherche à prédire la prix d'une maison en fonction de sa surface dans une ville donnée



On peut utiliser un ou plusieurs prédicteur(s)/variable(s) pour essayer d'apprendre ce modèle (proche des statistiques).

Régression Linéaire Simple

Définition modèle gaussien

Le modèle linéaire **gaussien** simple est le cas particulier où l'on cherche à prédire une variable Y en fonction de la seule variable X, pour cela, on suppose que notre modèle s'écrit sous la forme :

$$Y = X\beta + \varepsilon, \quad \text{où} \quad \varepsilon \sim \mathcal{N}(0, \sigma^2),$$

où $X = (1, \mathbf{x})$ pour prendre en compte la constante dans le modèle, ce que l'on peut réécrire

$$y = \beta_0 + \beta_1 x + \varepsilon.$$

On retrouve donc l'équation d'une droite dans le plan.
ε est un bruit gaussien et représente les perturbations du modèle ou l'erreur présente dans les données.

Objectif : trouver un modèle qui minimise l'erreur entre la valeur prédite $\hat{y}$ et la valeur prédite.

Régression linéaire simple

Nous avons vu que nous avons besoin pour cela d'une fonction objective (ou loss) que l'on cherche à optimiser. Mais pour le moment nous n'avons vu que des loss pour des algorithmes de classification. Mais on peut aussi rencontrer des loss dans un contexte de régression, par exemple, pour un **modèle linéaire**, on va chercher à minimiser l'**erreur quadratique (moyenne)**

Erreur quadratique moyenne

$$\ell(y, \hat{y}) = (y - \hat{y})^2 = (y - (\beta_0 + \beta_1 x))^2$$

et donc le risque empirique sur un échantillon S à minimiser serait de la forme

$$\mathcal{R}_S(h) = \frac{1}{m} \sum_{i=1}^{m} (y_i - (\beta_0 + \beta_1 x_i))^2.$$

Estimation des paramètres du modèle

L'estimation des paramètres du modèle se fait par la **méthodes des moindres carrés**.

Méthode des moindres carrés

Il s'agit de trouver les valeurs de β_0 et β_1 qui vont minimiser l'erreur quadratique moyenne :

$$\mathcal{R}_S(h) = \frac{1}{m}\sum_{i=1}^{m}(y_i - (\beta_0 + \beta_1 x_i))^2.$$

Comme il s'agit d'une fonction convexe en les paramètres β_0 et β_1 (comme fonction quadratique), le minimum est alors atteint pour les valeurs de β_0 et β_1 telles que

$$\frac{\partial \mathcal{R}_S(h)}{\partial \beta_0} = 0 \quad \text{et} \quad \frac{\partial \mathcal{R}_S(h)}{\partial \beta_1} = 0.$$

Estimation des paramètres du modèle

On peut montrer que le coefficients β_1 s'exprime uniquement en fonction de la **variance** de X et de la **covariance** entre X et Y et que β_0 s'expriment en fonction des moyennes de X et Y et de β_1.

Exercice

Montrer, en résolvant la solution au problème d'optimisation suivant

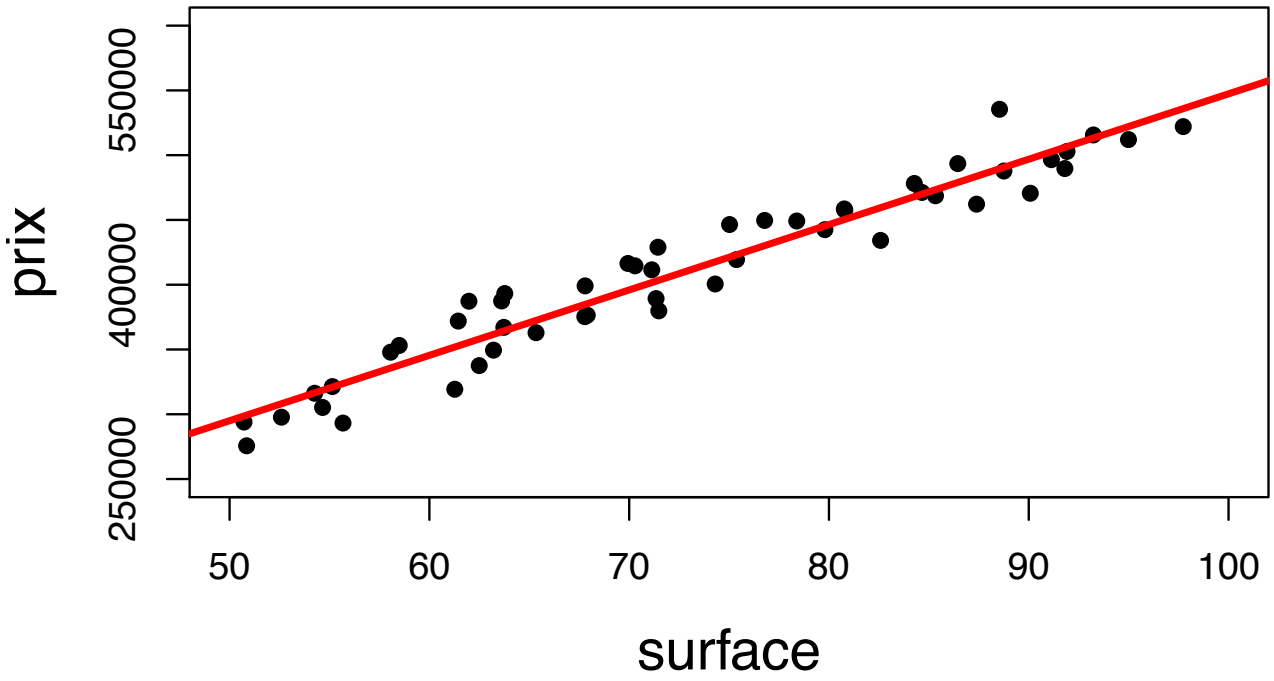
$$\arg\min_{\beta_0,\beta_1} \frac{1}{m}\sum_{i=1}^{m}(y_i - (\beta_0 + \beta_1 x_i))^2$$

est

$$\beta_1 = \frac{Cov(X,Y)}{Var(X)} \quad \text{et} \quad \beta_0 = \mathbb{E}[Y] - \beta_1 \mathbb{E}[X]$$

Résultats

Les valeurs de β_0 (ordonnée à l'origine) et β_1 (coefficient directeur) permettent alors de déterminer l'équation de la droite qui approxime le mieux nos données.
Nous sommes donc capable d'estimer le prix d'un bien en fonction de la surface de ce bien.



Régression linéaire multiple

Le problème est exactement le même que précédemment, à la seule différence que nous utilisons non plus un mais plusieurs descripteurs (variables) X pour essayer d'expliquer les observations Y.
Notre modèle s'écrit toujours sous la forme :

$$Y = X\beta + \varepsilon \quad \text{où} \quad \varepsilon \sim \mathcal{N}(0, \sigma^2)$$

où

- $Y = (y_1, ..., y_m)$ est un vecteur de $\mathbb{R}^m$ est **la variable à expliquer**
- $X = (1, \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_d)$ est une matrice de $\mathbb{R}^{m \times (d+1)}$est **la variable explicative** et $\mathbf{x}_j \in \mathbb{R}^m$ et correspond à un descripteur en particulier
- $\beta = (\beta_0, \beta_1, ..., \beta_d)$ est un vecteur de $\mathbb{R}^{d+1}$ qui représente les paramètres du modèle.

Notons h le modèle associé pour la suite.

Régression linéaire multiple

La résolution peut également se faire par la méthode des moindres carrés.

Estimation par Méthode des moindres carrés

On cherche à résoudre le problème d'optimisation suivant :

$$\underset{\beta \in \mathbb{R}^{d+1}}{\arg\min} \|Y - X\beta\|_2^2 = \sum_{i=1}^{m} (y_i - h(\mathbf{x}_i))^2.$$

On montre alors qu'une estimation de β est donnée par la relation

$$\beta = (X^T X)^{-1} X^T Y$$

à la condition que la matrice $X^T X$ soit bien inversible, *i.e.* que les variables ne soient pas corrélées.

Régression et décomposition biais-variance

On rappelle que l'on note $h(\mathbf{x})$ la valeur prédite par notre modèle. On va maintenant regarder comment se **décompose l'erreur dans un modèle de prédiction**, en s'intéressant à la quantité $\mathbb{E}[(y - \hat{h}(\mathbf{x}))]$, où $\hat{h}$ est donc le modèle appris à partir des données.

Décomposition de l'erreur

Supposons que l'on souhaite étudier le modèle de régression de la forme

$$y = h(\mathbf{x}) + \varepsilon$$

et notons $\hat{h}$ le modèle estimé avec nos données. Sous les hypothèses d'un modèle de régression, on peut alors décomposer l'erreur comme suit :

$$\begin{aligned}\mathbb{E}[(y - \hat{h}(\mathbf{x}))^2] &= \left(\mathbb{E}\left[\hat{h}(\mathbf{x})\right] - h(\mathbf{x})\right)^2 + \mathbb{E}\left[\left(\mathbb{E}\left[\hat{h}(\mathbf{x})\right] - \hat{h}(\mathbf{x})\right)^2\right] \\ &\quad + \mathbb{E}\left[(y - h(\mathbf{x}))^2\right]\end{aligned}$$

Régression et décomposition biais-variance

Chaque terme dispose ainsi de sa propre signification :

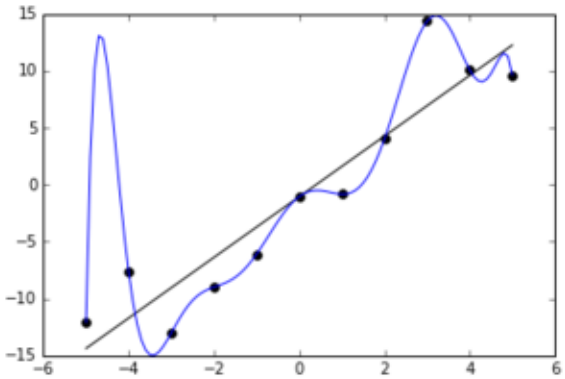
- le premier terme est le carré du biais du modèle $\hat{h}$: donc la différence entre l'estimation et la vraie valeur de h
- le deuxième terme est un terme de variance associé à $\hat{h}$: il quantifie combien l'estimateur varie autour de sa moyenne lorsque les données changent, on quantifie sa sensibilité au changement dans les données
- le troisième terme est ce que l'on appelle l'erreur de Bayes, elle est **indépendante de l'estimateur** et **dépend uniquement de la distribution des données**

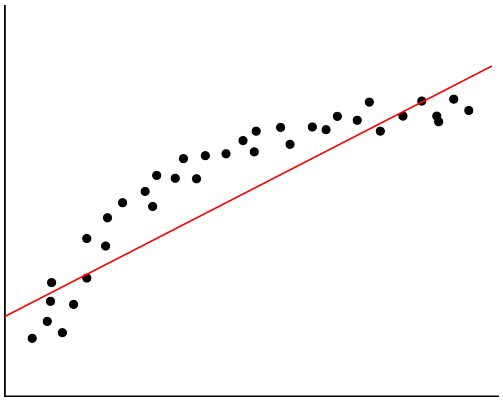
La quantité qui intéresse le plus en Machine Learning est ce que l'on appelle **l'excès de risque** :

$$\mathbb{E}[(y - \hat{h}(\mathbf{x}))^2] - \mathbb{E}\left[(y - h(\mathbf{x}))^2\right].$$

C'est véritablement cette quantité que l'on cherche à minimiser

Décomposition de l'erreur et sur ou sous-apprentissage





On cherche en fait à trouver un bon compromis entre deux types d'erreurs, un compromis biais-variance :

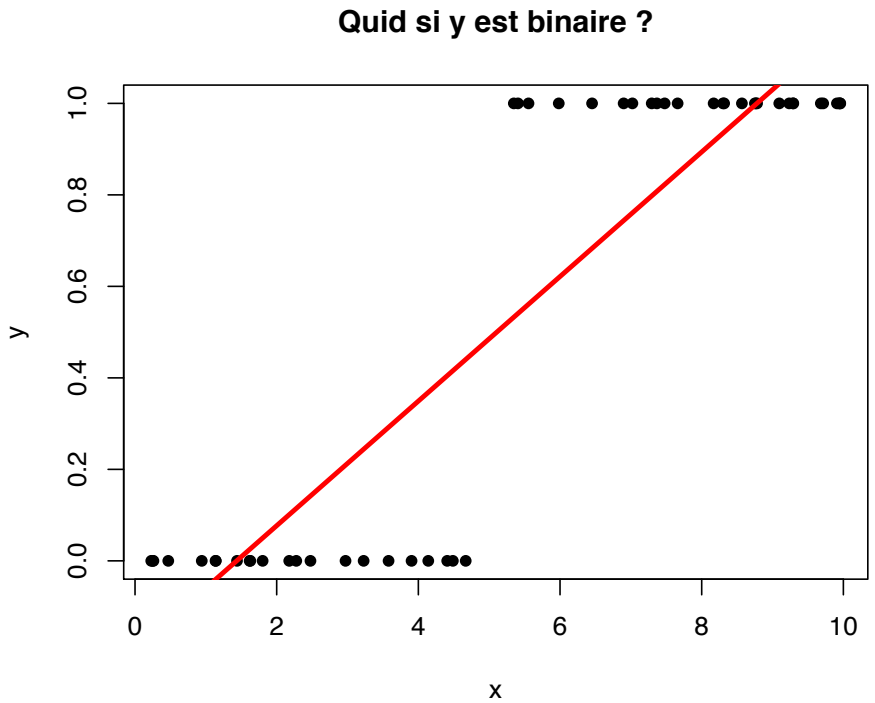
- l'ensemble des fonctions trop simples qui n'arrivent pas à expliquer suffisamment les données :
  elles ont donc un biais trop grand : **underfitting**
- l'ensemble des fonctions trop riches qui expliquent les données avec une grande précision mais qui sont trop spécifiques aux observations effectuées :
  elles présentent donc une grande variance : **overfitting**

Quelques remarques d'un point de vue Statistiques

- Le but était uniquement d'illustrer un modèle de régression, mais nous aurions pu aller plus loin dans l'étude
- Nous aurions pu étudier les propriétés statistiques de ce modèle et vérifier sa validité
- Etudier la qualité de l'estimateur des moindres carrés et faire le lien avec l'estimateur de maximum de vraisemblance
- Vérifier, comme pour une ANOVA, que les résidus sont bien normalement distribués, que l'hypothèse d'homoscédasticité est vérifiée pour les résidus ...
- Voir comment cela est lié à l'étude de la corrélation entre deux ou plusieurs variables quantitatives et regarder les tests statistiques permettant de vérifier la qualité d'un modèle
- Regarder différents critères (AIC ou encore BIC) permettant de juger de la qualité d'un modèle en fonction du nombre de descripteur (intéressant pour faire de la sélection de modèle) ...

Quid dans le cas présent ?

Imaginons que l'on dispose d'un jeu de données pour lequel la variable Y ne peut prendre que les valeurs $\{0, 1\}$ ou encore $\{-1, 1\}$?



A priori, un modèle linéaire tel que défini précédemment ne semble pas du tout approprié , il va falloir introduire un autre type de modèle.

Régression Logistique (LR)

A propos

Le modèle de Régression Logistique, également appelé *modèle logit* est un modèle qui date du 20ème siècle (Cox, 1958). Ce sont des modèles spécifiquement dédiés au cas où la variable à prédire prend **une valeur discrète**.
Pour simplifier la présentation, on va se placer dans le cas où nous n'avons que deux classes à prédire, *i.e.* y ne prend que deux valeurs -1 et 1.

Idée

Ce type de modèle peut également être vu comme un modèle linéaire mais ... qui ne cherche pas directement à prédire la valeur de y.
C'est un modèle qui va chercher à prédire la probabilité qu'un exemple appartienne a la classe positive, *i.e.* que $y = 1$.
On va donc chercher un modèle qui va approximer la probabilité suivante

$$\eta = \mathbb{P}[Y = 1 \mid X]$$

Modèle

Objectif : apprendre un modèle qui permet d'approximer la probabilité qu'une donnée soit prédite positive $(y = 1)$. Pour faire cela, on va chercher un modèle linéaire qui va approximer l'*odd ratio* entre la classe positive et la classe négative.

Régression Logistique

Apprendre un modèle linéaire de la forme $\beta^T \mathbf{x}$ tel que

$$\ln\left(\frac{\mathbb{P}[y = 1 \mid \mathbf{x}]}{\mathbb{P}[y = 0 \mid \mathbf{x}]}\right) = h(\mathbf{x}) = \beta^T \mathbf{x},$$

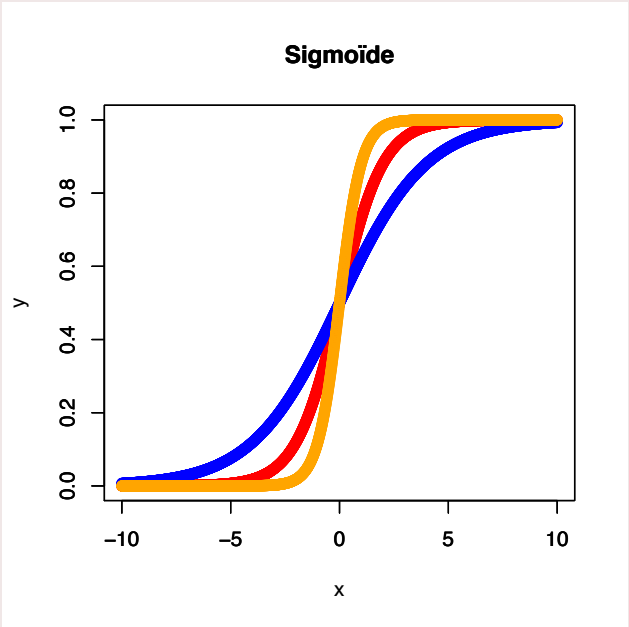
où β sont donc les paramètres du modèle à apprendre. Une fois le modèle appris, on détermine la probabilité qu'une donnée appartienne à la classe positive en calculant :

$$\mathbb{P}[y = 1 \mid \mathbf{x}] = \frac{1}{1 + \exp(-h(\mathbf{x}))}.$$

Fonction Logistique

La fonction logistique est une **sigmoïde** désignent un ensemble de fonction prenant ses valeurs dans $[0, 1]$ ayant une forme particulière. On l'étudie également dans problèmes de dynamique (équations différentielles) et notamment pour les mouvements *chaotiques* ... enfin bref (voir la notion de chaos pour les système dynamique).

Sigmoïde



Apprentissage

Il est plus complexe que pour l'apprentissage d'un modèle linéaire multiple mais peut se faire de la même façon, c'est-à-dire par **maximum de vraisemblance**.

$$
\begin{aligned}
\mathcal{L}(\mathbf{w}, S) &= \prod_{i=1}^{m} \mathbb{P}[Y = y_i \mid X = x_i], \\
&= \prod_{i=1, y_i=1}^{m} \mathbb{P}[Y = y_i \mid X = x_i] \times \prod_{i=1, y_i=-1}^{m} \mathbb{P}[Y = y_i \mid X = x_i], \\
&= \prod_{i=1}^{m} \left(\frac{1}{1 + \exp(-h(\mathbf{x}_i))} \right)^{\frac{1}{2}(1+y_i)} \times \left(\frac{1}{1 + \exp(h(\mathbf{x}_i))} \right)^{\frac{1}{2}(1-y_i)}, \\
&= \prod_{i=1}^{m} \frac{1}{1 + \exp(-y_i h(\mathbf{x}_i))}.
\end{aligned}
$$

Apprentissage

Comme il est d'usage en Statistiques ou encore en Machine Learning, on va plutôt chercher à minimiser l'**opposé de la log-vraisemblance** ℓ. On va donc chercher à résoudre le problème d'optimisation

$$\min_{\beta \in \mathbb{R}^{d+1}} \frac{1}{m} \sum_{i=1}^{m} \ln\left(1 + \exp\left(-y_i(\beta^T \mathbf{x}_i)\right)\right).$$

Bien que ce problème soit convexe (vous pouvez essayer de le vérifier en calculant la matrice hessienne et en vérifiant qu'elle est bien semi-definie positive), il n'existe pas de solution analytique.
Nous sommes obligés de passer par des algorithmes *d'optimisation* pour trouver la solution : **algorithme de descente de gradient comme la méthode Newton**, mais vous verrez cela en temps voulu en cours d'optimisation ;)

Prédiction

Règle de prédiction

Une fois que l'on dispose de notre modèle, nous sommes donc capable d'évaluer la probabilité qu'une nouvelle donnée $\mathbf{x}$ appartienne à la classe positive.
Pour prédire son étiquette, on utilise couramment la valeur 0.5 comme valeur critique ou comme valeur seuil (*threshold* en anglais) de telle sorte que :

$$y = \begin{cases} 1 & \text{si } \mathbb{P}[Y \mid X = \mathbf{x}] > 0.5, \\ 0 & \text{sinon.} \end{cases}$$

Mais c'est bien évidemment loin d'être la seule règle et il est parfois intéressant de bouger ce seuil dans certains contextes de classification

Les Séparateurs à Vaste Marge (SVM)

A propos des SVM

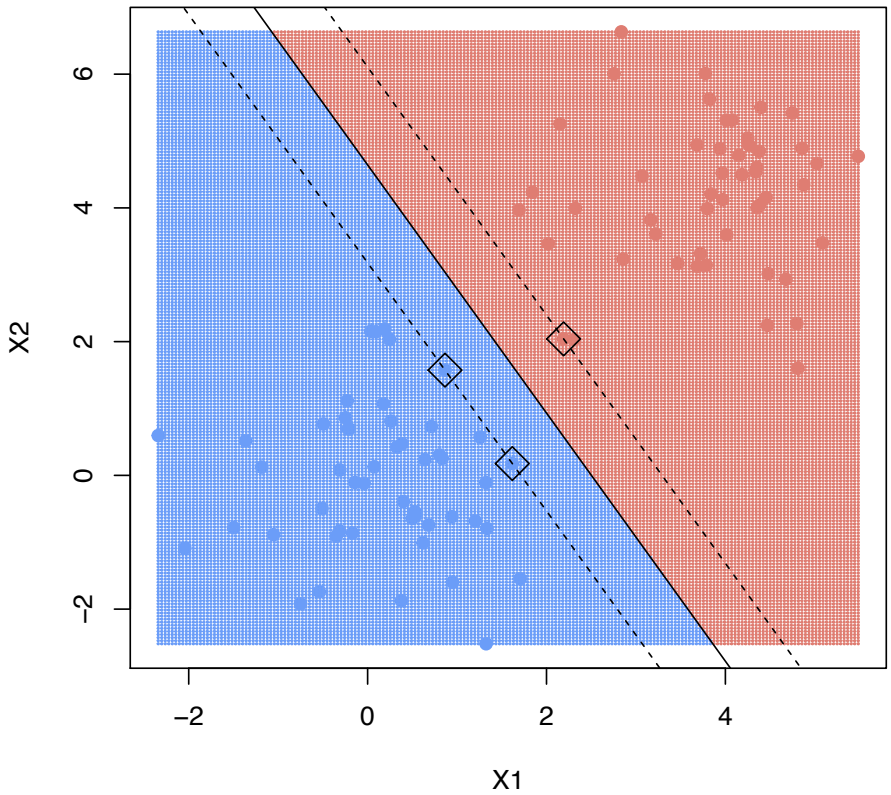
Certainement l'un des algorithmes les plus répandues en apprentissage machine (Vapnik and Cortes, 1995) pour effectuer des tâches de classification binaires.

Idée : apprendre une hypothèse h dont le but est de prédire l'étiquette d'une donnée. Elle se présente sous la forme d'un hyperplan (affine) qui va séparer l'espace en deux : $\{-1, +1\}$:

$$h(\mathbf{x}) = sign[\langle \mathbf{w}, \mathbf{x} \rangle + b] = \begin{cases} -1 & \text{if} \quad \langle \mathbf{w}, \mathbf{x} \rangle + b < 0, \\ +1 & \text{if} \quad \langle \mathbf{w}, \mathbf{x} \rangle + b > 0. \end{cases}$$

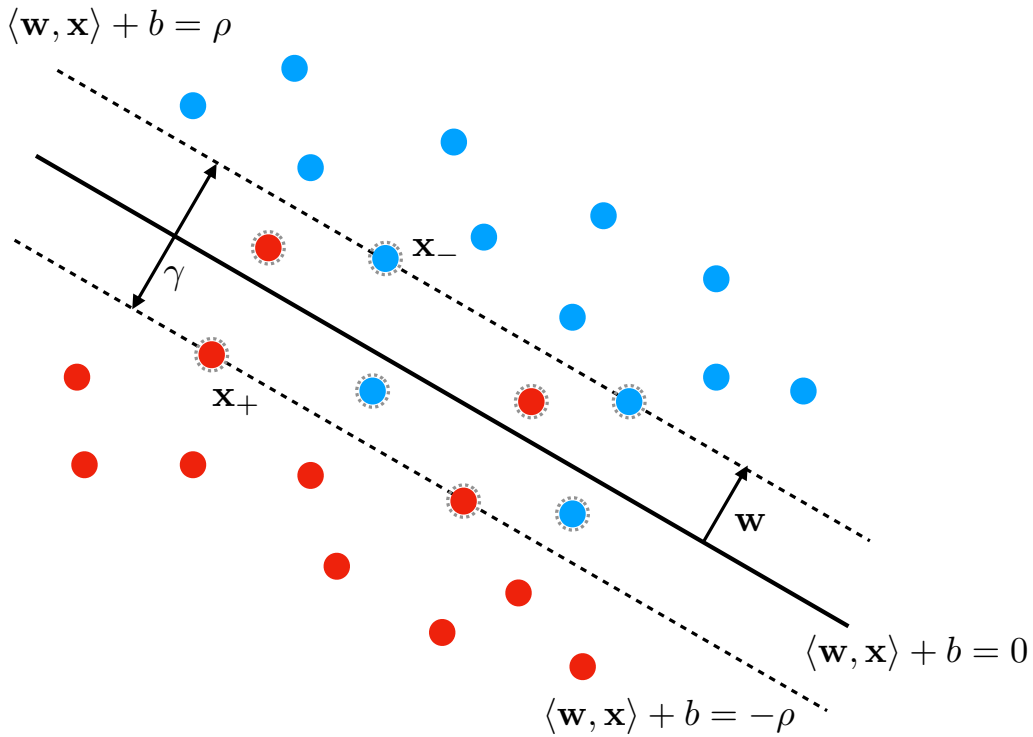
Problème : plusieurs plans peuvent êtres solutions et séparer parfaitement nos données.

A propos des SVM



→ une solution consiste à prendre le SVM qui présente la plus grande marge (Boser et al., 1992), **c'est celui qui généralisera le mieux !**

Marge et définition d'un SVM



La mage γ est la distance entre les deux hyperplans d'équations

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = \pm \rho.$$

On va maintenant voir comment la marge γ est liée à notre hyperplan défini par $\mathbf{w}$.

Marge et définition d'un SVM

On utilisera la décomposition suivante $\mathbf{x} = \mathbf{x}^{\mathbf{w}} + \mathbf{x}^{\mathbf{w}^\perp}$ et on prend deux points $\mathbf{x}_+$ et $\mathbf{x}_-$ se trouvant sur la marge

$$\begin{aligned}
h(\mathbf{x}_+) - h(\mathbf{x}_-) &= 2, \\
\langle \mathbf{w}, \mathbf{x}_+ \rangle + b - (\langle \mathbf{w}, \mathbf{x}_- \rangle + b) &= 2, \\
\langle \mathbf{w}, \mathbf{x}_+^{\mathbf{w}} \rangle + \underbrace{\langle \mathbf{w}, \mathbf{x}_+^{\mathbf{w}^\perp} \rangle}_{=0} + b - (\langle \mathbf{w}, \mathbf{x}_-^{\mathbf{w}} \rangle + \underbrace{\langle \mathbf{w}, \mathbf{x}_-^{\mathbf{w}^\perp} \rangle}_{=0} + b) &= 2, \\
\langle \mathbf{w}, \mathbf{x}_+^{\mathbf{w}} \rangle - \langle \mathbf{w}, \mathbf{x}_-^{\mathbf{w}} \rangle &= 2, \\
2\langle \mathbf{w}, \mathbf{x}_+^{\mathbf{w}} \rangle &= 2.
\end{aligned}$$

En prenant la norme de chaque côté :

$$\gamma = 2\|\mathbf{x}_+^{\mathbf{w}}\|_2 = \frac{2}{\|\mathbf{w}\|_2}.$$

Marge et définition d'un SVM

Maximiser la marge $\Leftrightarrow$ minimiser $\|\mathbf{w}\|$

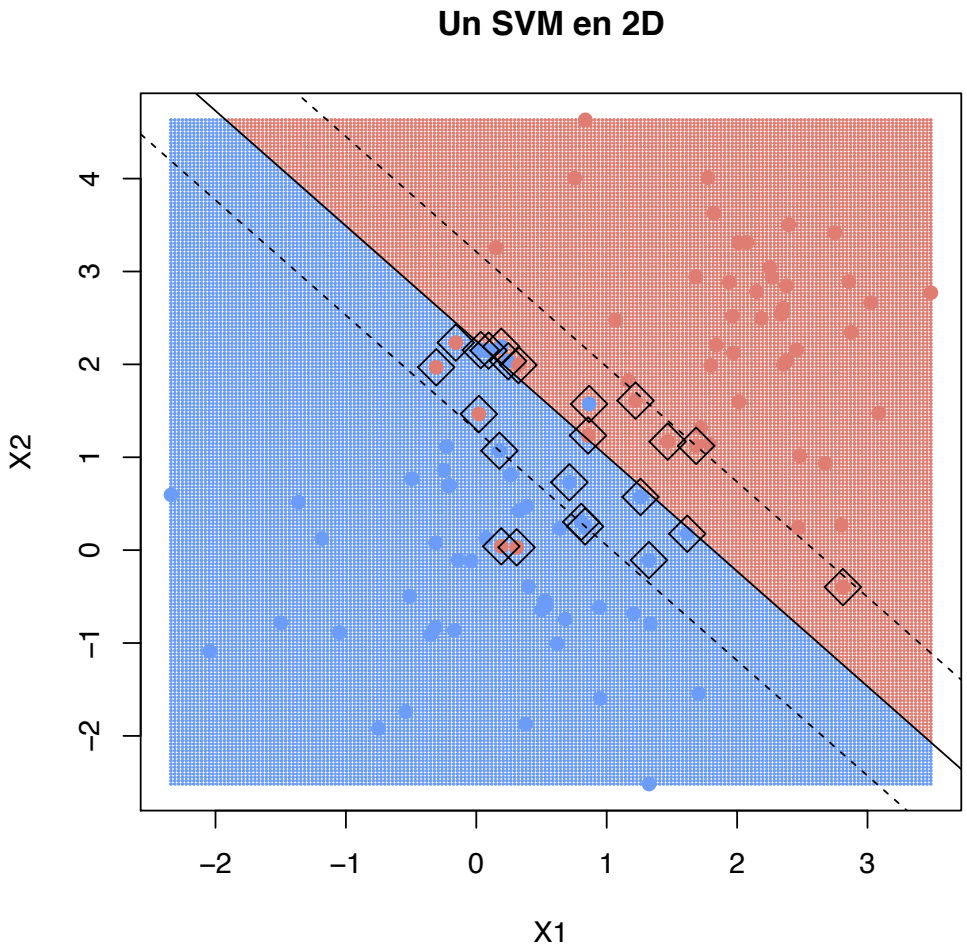
On cherche donc l'hyperplan séparateur qui vérifie le problème d'optimisation suivant :

$$\begin{aligned} \min_{(\mathbf{w},b)\in\mathbb{R}^{d+1}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ s.t. \quad & y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) \geq 1, \quad \text{for all } i = 1, ..., m. \end{aligned}$$

$\rightarrow$ Formulation connue sous le nom de **Hard Margin SVM**

Problème : les problèmes ne sont que très rarement linéairement séparables ...

Limite du Hard Margin SVM



... il faut donc savoir relâcher les contraintes sur notre hypothèse et l'autoriser à faire quelques erreurs.

Soft Margin SVM

On commence par relâcher nos contraintes en autorisant des données d'une certaine étiquette à se trouver du mauvais côté de l'hyperplan
$\rightarrow$ **introduction de variables slacks** ξ_i que l'on va inclure dans notre problème d'optimisation :

$$\begin{array}{ll} \min\limits_{\xi \in \mathbb{R}^m,\ (\mathbf{w},b)\in\mathbb{R}^{d+1}} & \frac{1}{2}\|w\|_2^2 + \frac{C}{m}\sum_{i=1}^m \xi_i \\ s.t. & y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) \geq 1 - \xi_i, \quad \forall\ i = 1, ..., m, \\ & \xi_i \geq 0, \qquad\qquad\qquad\quad \forall\ i = 1, ..., m. \end{array}$$

Une autre écriture :

$$\min_{(\mathbf{w},b)\in\mathbb{R}^{d+1}} \frac{1}{2}\|w\|_2^2 + \frac{C}{m}\sum_{i=1}^m [1 - y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle) + b]_+.$$

C : paramètre de régularisation entre **erreurs** et **complexité du modèle**.

Soft Margin SVM

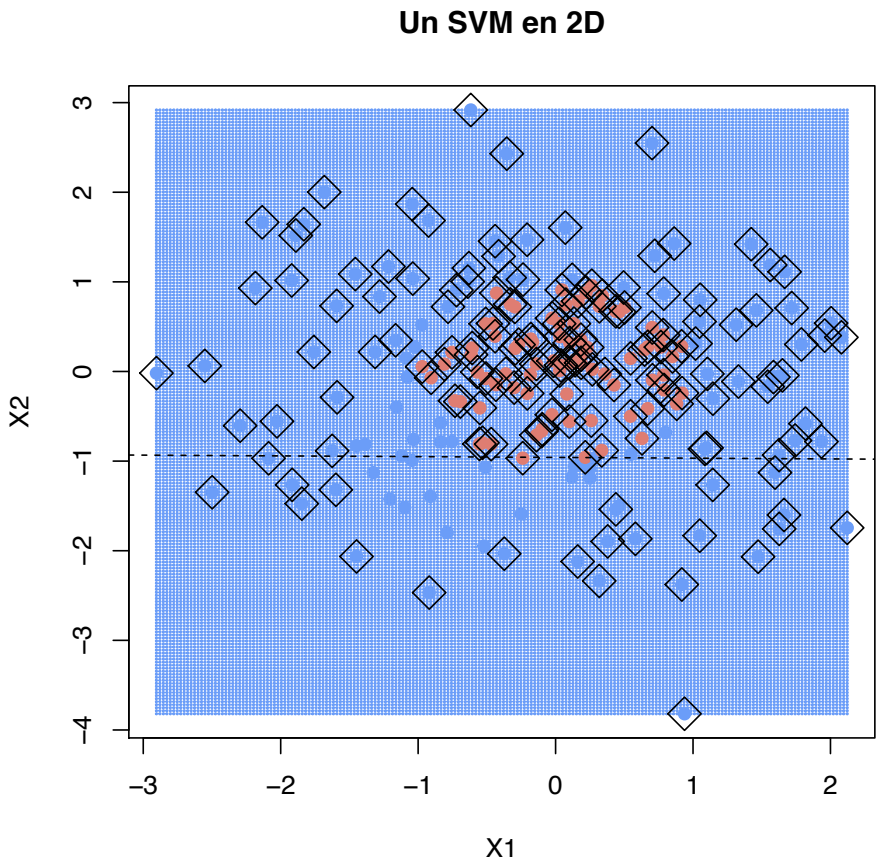
Quelques remarques

- Seuls les vecteurs du support participent à la construction de la solution (conséquence des conditions de KKT + lemme de Farkas)
- Problème difficile à optimiser car beaucoup de contraintes.
- Une difficulté croissante lorsque la dimension du jeu de données est élevée avec une complexité en $\mathcal{O}(d^3)$ (Chapelle, 2007).
- On passe le plus souvent, lorsque la taille du jeu de données reste raisonnable, par la formulation duale qui se présente sous la forme :

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j + \sum_{i=1}^{m}\alpha_i, \\ s.t. \quad & 0 \leq \alpha_i \leq \frac{C}{m} \qquad \forall\, i = 1, ..., m, \\ & \sum_{i=1}^{m} y_i\alpha_i = 0. \end{aligned}$$

Nécessité des modèles non linéaires

Certaines tâches ne peuvent se résoudre avec de simples modèles linéaires !



→ **Astuce du noyaux**

Méthodes à noyaux

Qu'est-ce qu'un noyaux ?

Une application à noyau $\mathbf{K}$ est une application de $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ qui est (i) symétrique et (ii) semi-définie positive

(i) $\forall\, (\mathbf{x}, \mathbf{x}') \in \mathbb{R}^d \times \mathbb{R}^d$, nous avons: $K(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}', \mathbf{x})$,

(ii) $\forall\, (\mathbf{x}_i, \mathbf{x}_j) \in \mathbb{R}^d \times \mathbb{R}^d$ et $\forall\, \mathbf{c} \in \mathbb{R}^d$, nous avons :
$c^T \mathbf{K} c = \sum_{i=1}^m \sum_{j=1}^m c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0.$

Le cas classique $\mathbf{K} = \mathbf{I}_d$: on retrouve ce que l'on appelle le *noyau linéaire* (*cf* cas précédent).

L'idée est donc d'introduire ce type d'application dans notre précédent problème d'optimisation

Méthodes à noyaux

Formulation d'un problème d'optimisation avec une méthode à noyau

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j) + \sum_{i=1}^{m}\alpha_i, \\ s.t. \quad & 0 \leq \alpha_i \leq \frac{C}{m}, \qquad \text{for all } i = 1, ..., m, \\ & \sum_{i=1}^{m}\alpha_i = 0, \end{aligned}$$

où $\mathbf{K} = K(\mathbf{x}_i, \mathbf{x}_j)$ est donc une matrice à noyau, *i.e.* notre application noyau évaluée en les différents couples de points de notre jeu de données.

En quoi cela va nous permettre de résoudre notre problème précédent ? Utiliser un noyau revient à projeter nos données dans un espace de dimension potentiellement infini ! (Mercer, 1909).

Méthodes à noyaux

Théorème de Mercer

Soit $\mathcal{X}$ un sous espace compact de $\mathbb{R}^d$ et soit K une application SDP (symétrique définie positive), *i.e.* un noyau. Alors il existe une base orthonormale de fonctions $(\Phi_j)_{n\in\mathbb{N}}$, et une suite de nombres $(\lambda_j)_{j\in\mathbb{N}}$ où $\lambda_j \geq 0$ pour tout j, tels que :

$$K(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{\infty} \lambda_j \Phi_j(\mathbf{x}) \Phi_j(\mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle,$$

où $\Phi(\mathbf{x}) = (\sqrt{\lambda_1}\Phi_1(\mathbf{x}), ..., \sqrt{\lambda_j}\Phi_j(\mathbf{x}), ...)$ est la représentation de la donnée $\mathbf{x}$ dans un nouvel espace.

Faire le lien avec le développement en série entière des fonctions (dimension infinie dans le cas gaussien).

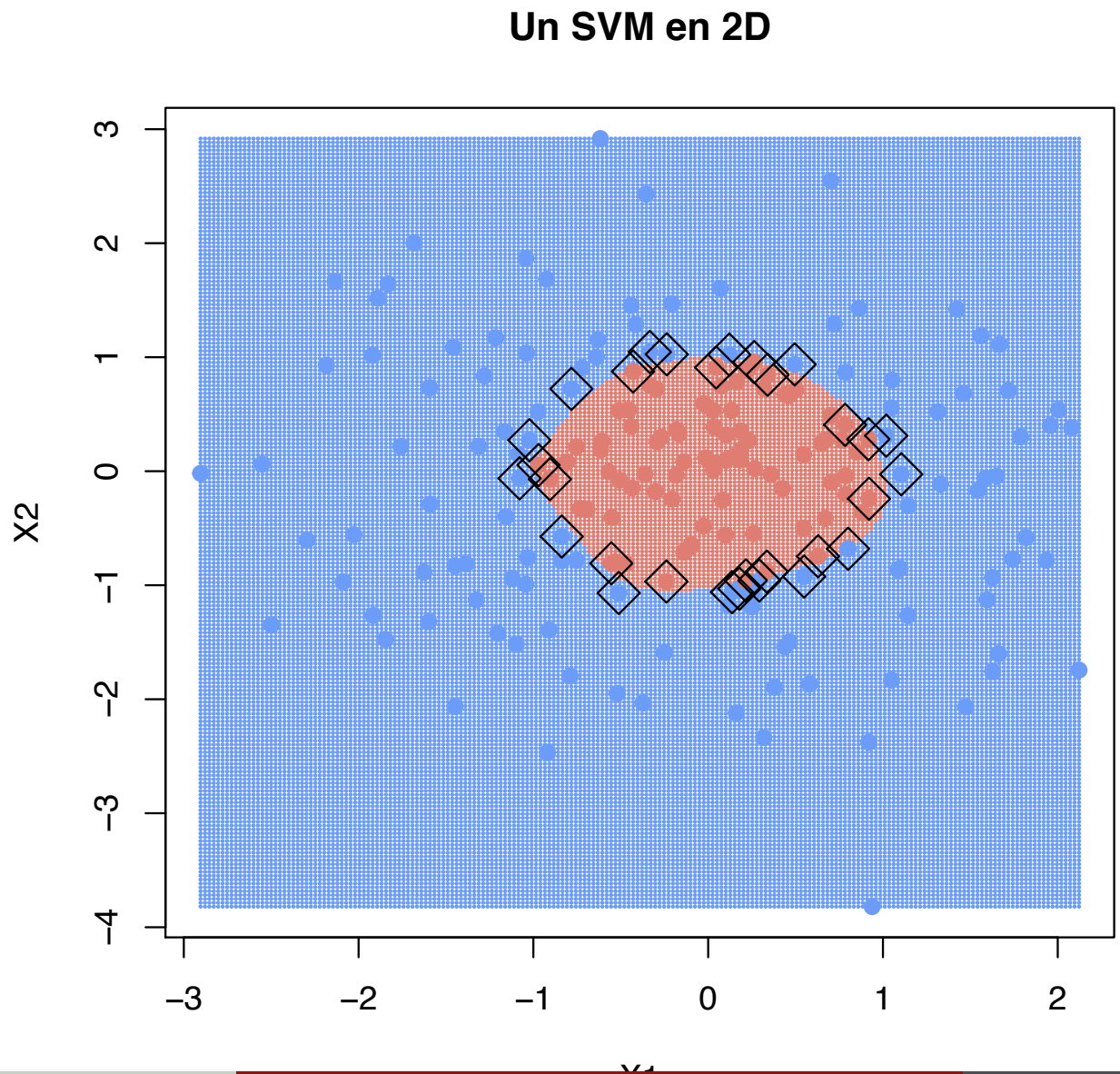
Méthodes à noyaux

Quelques exemples de noyaux

- **Noyau linéaire** : $K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$, il s'agit du produit scalaire standard
- **Noyau gaussien** : $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\sigma^2}\right)$, ou σ est un hyper-paramètre à tuner. Il contrôle l'importance que va avoir la similarité entre deux points. Plus ce paramètre est grand, moins on accorde d'importance à la similarité entre deux exemples et plus le rôle de chaque exemple sera uniforme dans l'apprentissage du modèle (moins de risque d'overfitter).
- **Noyau polynomial** : $K(\mathbf{x}, \mathbf{x}') = (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^p$, p est le degré du polynôme.

Vous pouvez trouver une liste plus complète de noyaux utilisés en ML chez Genton (2002).

A l'aide d'un noyau gaussien



Clustering (k-means)

Clustering

Objectif du clustering : déterminer un partitionnement de nos données, *i.e.* assigner à chaque individu $\mathbf{x}_i$ un groupe d'appartenance y_i. Ces groupes pourront ainsi être interprétés mais aussi représentés par un seul individu (moyenne de chaque groupe)

On pourra, avec ce type d'algorithme, apprendre une représentation simplifiée de nos données (pratique si on souhaite réduire la quantité de données à analyser !)

Cela permet également de mettre en exergue différents profils présents dans nos données, ce qui peut se révéler très important en marketing pour orienter les publicités.

Clustering

Fonctionnement : il repose sur une seule et unique chose fondamentale : la notion de **distance**.
D'ailleurs ... pouvez-vous me rappeler comment est définie une distance ?

On peut utiliser n'importe quelle distance pour cet algorithme. Bien évidemment, la distance utilisée va conditionner le clustering :

$$d_p(\mathbf{x}, \mathbf{x}')^p = \sum_{j=1}^{p} \left| x_j - x'_j \right|^p .$$

On prendra classiquement la distance euclidienne, *i.e.* $p = 2$.

Clustering : k-means

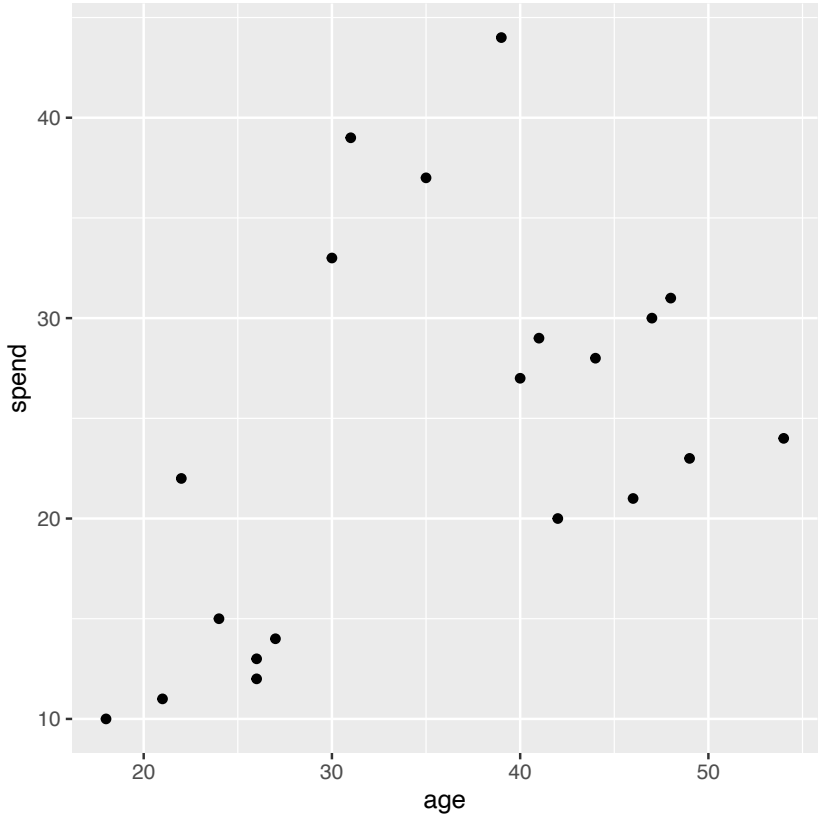
Il s'agit donc d'un algorithme de partitionnement des données qui fonctionnent de la façon suivante :

- On commence par fixer le nombre K de cluster que l'on souhaite créer, *i.e.* fixer le nombre de groupes dans lesquels sont répartis les individus.
- On tire aléatoirement un vecteur moyenne μ_k pour chaque groupe.
- Tant que les groupes ne sont pas stables :
  1) assigner chaque exemple $\mathbf{x}_i$ à son centre de groupe le plus proche μ_k
  2) mettre à jour μ_k :

$$\mu_k = \frac{1}{m_k} \sum_{i \in I_k} \mathbf{x}_i$$

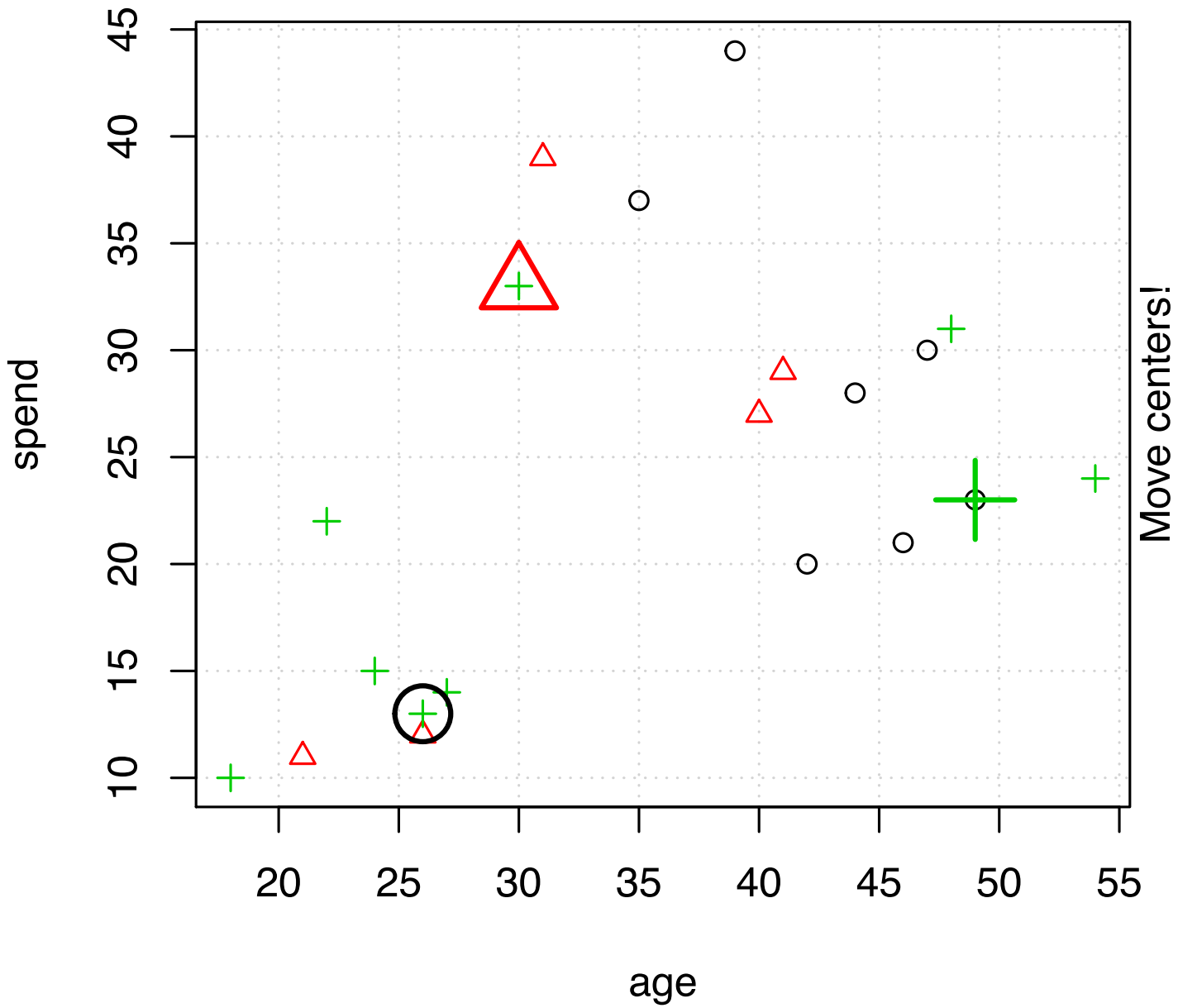
Clustering : k-means

Regardons, ce que cela donne sur un exemple pour mettre en avant la construction.

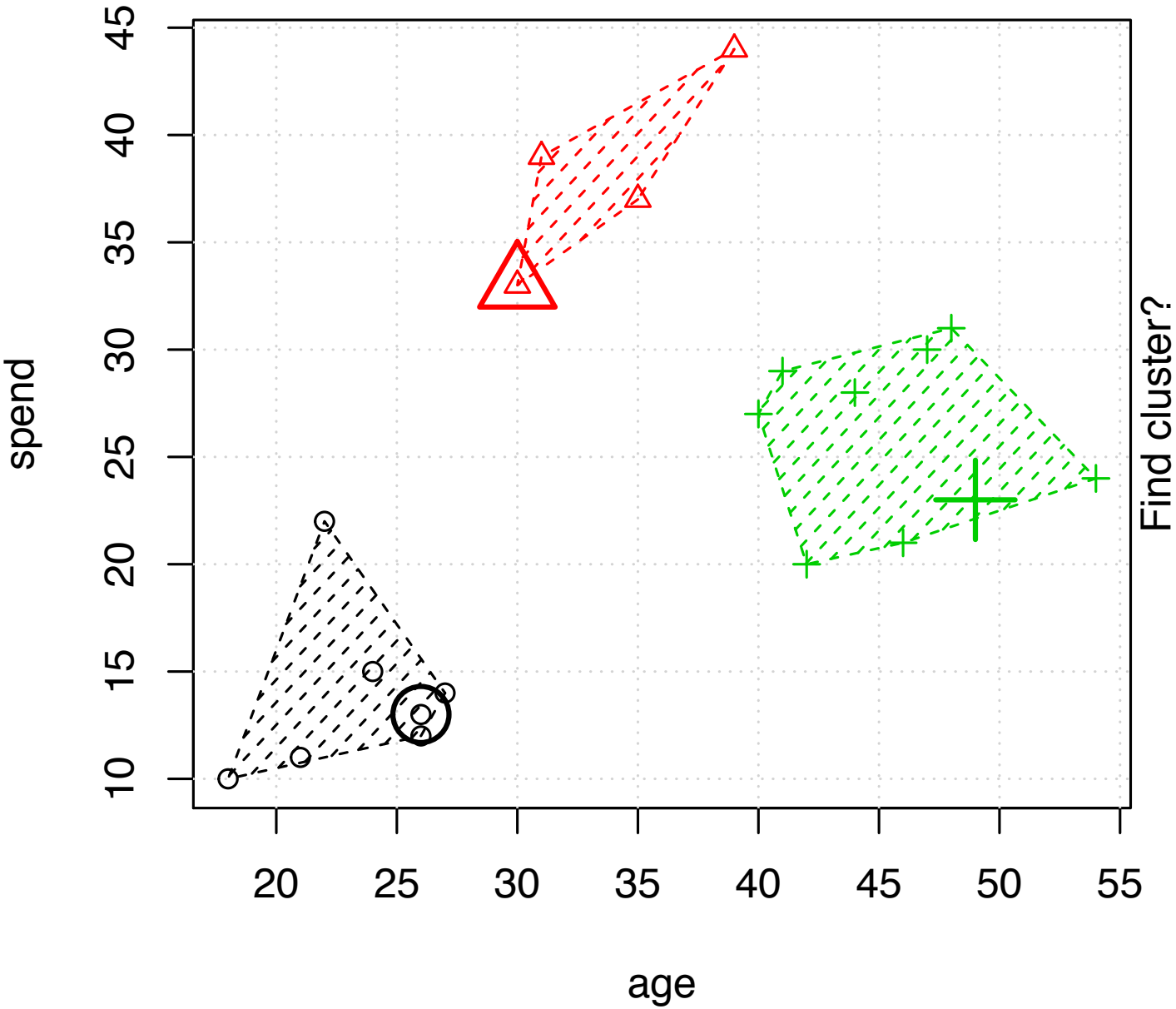


Pourriez-vous identifier rapidement les clusters présents dans ce jeu de données ?

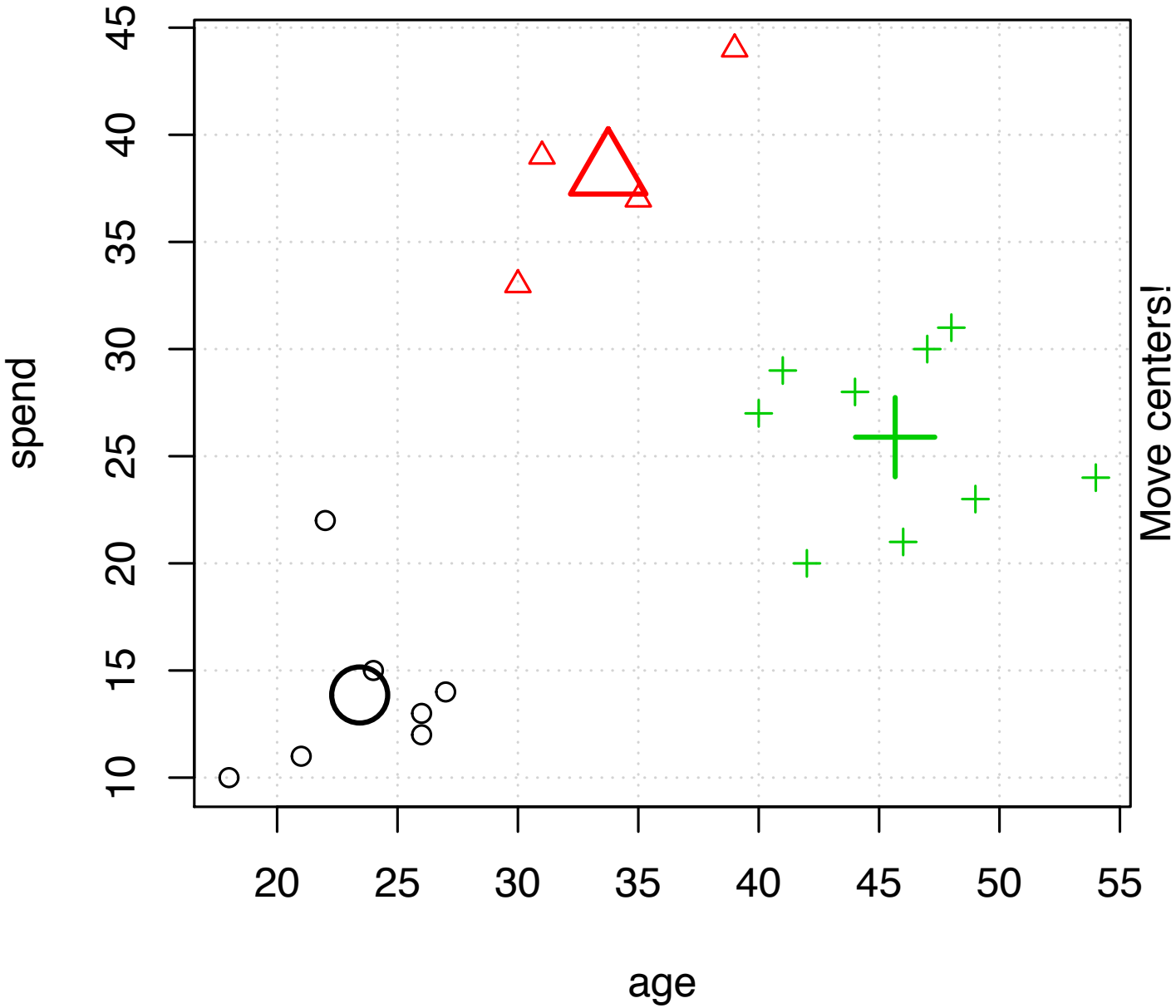
Clustering : k-means



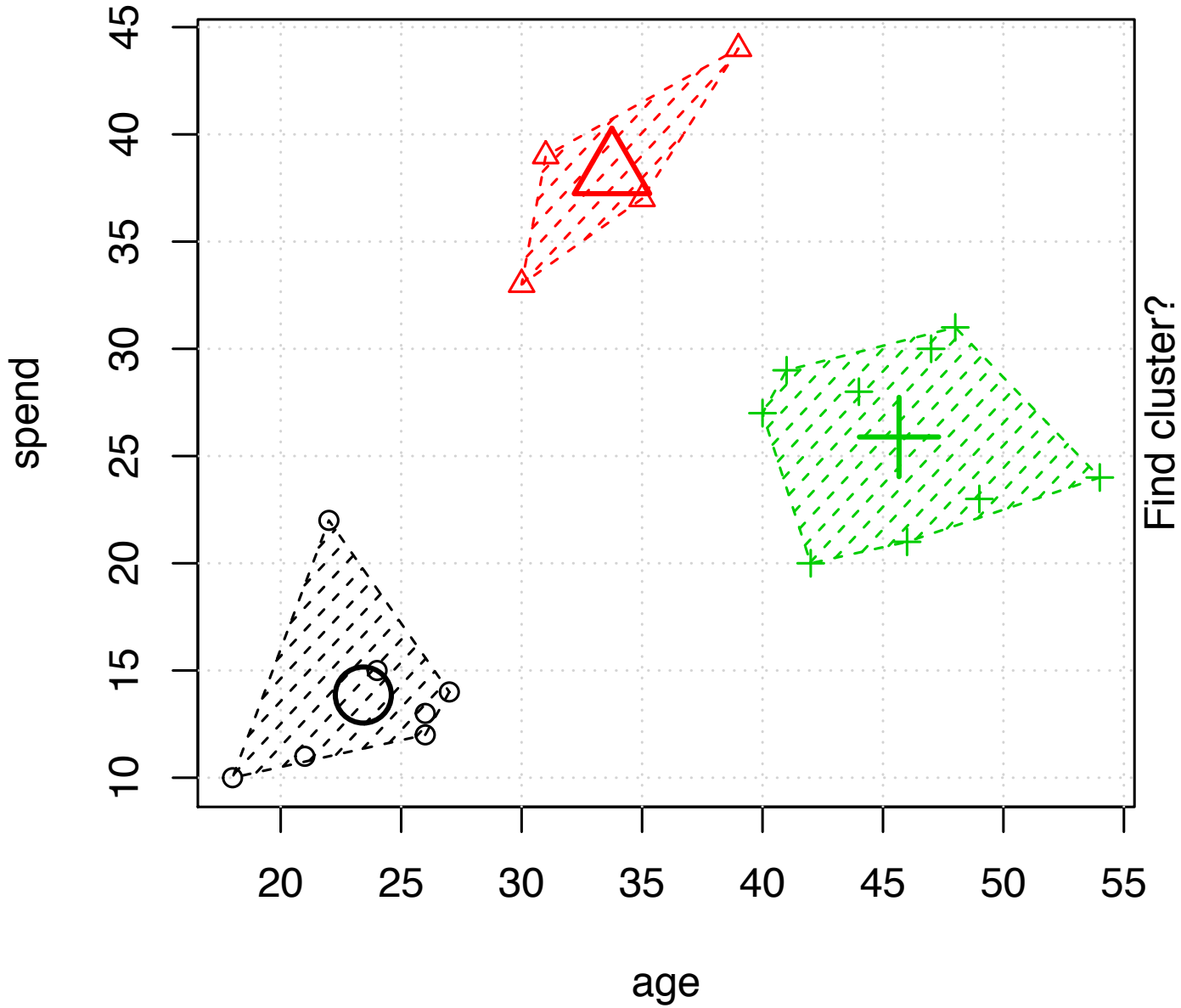
Clustering : k-means



Clustering : k-means



Clustering : k-means



Clustering : k-means

Objectif : l'algorithme k-means cherche à résoudre le problème d'optimisation suivant :

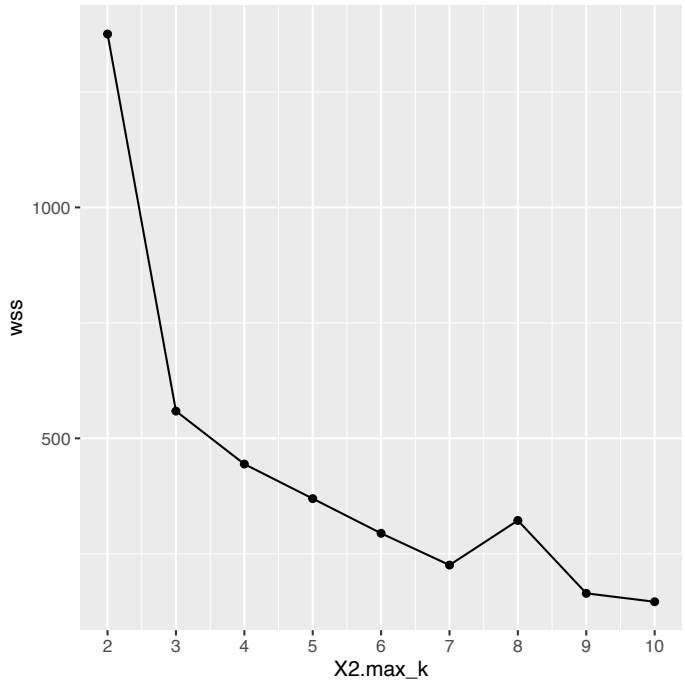
$$\arg\min_{\mathcal{P}} \sum_{k=1}^{K} \sum_{i \in \mathcal{P}_k} \|\mathbf{x}_i - \mu_k\|^2,$$

i.e. il cherche à minimiser l'**inertie (ou la variance) intra-classe**, c'est à dire la variance au sein des différents groupes.

Convergence : l'algorithme converge mais uniquement vers un optimum local, il n'y a pas de garantis que la solution soit optimale !

Clustering : k-means

Choix du nombre de classes : il est évident que plus le valeur de k est grande, plus la valeur de notre problème d'optimisation sera faible. Alors en pratique, pour choisir la valeur de k, on va rechercher un coude dans la graphe suivant :



References I

Ben-David, S., Eiron, N., and Long, P. M. (2003). On the difficulty of approximately maximizing agreements. *Journal of Computer and System Sciences*, 66(3):496–514.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152. ACM.

Bousquet, O., Boucheron, S., and Lugosi, G. (2004). *Introduction to Statistical Learning Theory*, pages 169–207. Springer Berlin Heidelberg.

Bousquet, O. and Elisseeff, A. (2002). Stability and generalization. *Journal of Machine Learning Research*, 2:499–526.

Chapelle, O. (2007). Training a support vector machine in the primal. *Neural Computation*, 19:1155–1178.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

Cox, D. R. (1958). The regression analysis of binary sequences (with discussion). *Journal of the Royal Statistical Society*, 20:215–242.

Genton, M. G. (2002). Classes of kernels for machine learning: A statistics perspective. *Journal of Machine Learning Research*, 2:299–312.

References II

Mercer, J. (1909). Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 209(441-458):415–446.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of Machine Learning*. The MIT Press.

Valiant, L. G. (1984). A theory of the learnable. *Communication of the ACM*, 27(11):1134–1142.

Vapnik, V. and Cortes, C. (1995). Support-vector networks. *Machine Learning*, 20:273–297.

Xu, H. and Mannor, S. (2010). Robustness and generalization. In *Conference on Learning Theory (COLT-10)*, pages 503–515.