

Supervised Machine Learning

Master 1 - MIASHS

Guillaume Metzler

Université Lumière Lyon 2
Laboratoire ERIC, UR 3083, Lyon

guillaume.metzler@univ-lyon2.fr

Automne 2022

Avant Propos I

Le cours se décompose de la façon suivante :

- **Trois séances CM de 3h** sur le Machine Learning en général :
Introduction à la problématique général, processus d'apprentissage et présentation des algorithmes supervisés classiques en apprentissage machine et des versions plus avancées.
- **3 séances de TD** pour mettre en pratique les notions tant sur le plan pratique que théorique.

Avant Propos II

Concernant les évaluations

Vous aurez deux évaluations :

- **Un examen** : il portera essentiellement sur le cours avec des questions plus ou moins proches du cours mais aussi une partie pratique.
- **Un projet** : l'idée sera de vous proposer de travailler sur un jeu de données réelles et de vous pratiquer les différents algorithmes, de les analyser et de faire un rapport sur les approches testées.

L'examen comptera pour environ $\rho\%$ de la note finale et $(100 - \rho)\%$ pour le projet.

Déroulement des séances de TD

Vous aurez donc 3 séances de TD qui aborderont l'apprentissage supervisé sur le plan théorique et pratique :

- **TD1** : séance de 2h sur la construction d'une borne en généralisation pour les problèmes convexes.
- **TD2** : séance de deux heures pour la pratique d'algorithmes en Machine Learning Supervisé (Python ou )
- **TD3** : séance de 3h - algorithmes avancés et ... petit examen (définir la forme de l'examen)

Qu'est-ce que le Machine Learning ?

Introduction

Qu'est-ce que le Machine Learning ? I

L'apprentissage Machine est une sous-branche de l'*Intelligence artificielle*. Elle se trouve à la frontière de l'Informatique et des Mathématiques Appliquées (comme les statistiques ou l'optimisation). Cette discipline recoupe également la *Science des Données* car elle nécessite préalablement de récolter/nettoyer et d'analyser des données afin d'en extraire les informations importantes pour l'application souhaitée.

Qu'est-ce que le Machine Learning ? II

Cette définition met en évidence des éléments importants en Machine Learning pour la résolution d'un problème

- **La tâche** : quel est le problème que je souhaite résoudre à l'aide d'un outil informatique ? Il est important de bien identifier et définir le problème.
- **Les données** : quelles sont les données dont je dispose pour m'aider à développer une intelligence capable d'effectuer cette tâche.
- **L'algorithme** : quel type de problème est-ce que je cherche à résoudre ? Mettre le problème sous forme *mathématiques*, développer l'algorithme pour le résoudre et l'implémenter.

Qu'est-ce que le Machine Learning ? III

**Au fait, quelles sont pour vous
les deux grandes phases
en Apprentissage Machine ?**

Qu'est-ce que le Machine Learning ? IV

Apprentissage/Entraînement

Cette première va consister à déterminer les paramètres d'un modèle préalablement défini et qui a pour objectif de répondre à une tache fixée.

Elle utilise un ensemble d'apprentissage, donc un ensemble limité de données.

Les tâches sont variées, comme la classification document, l'identification de chat ou de chien dans les images ou la prévision et l'études des cours d'un produit sur le marché boursier.

Cette phase est qualifiée de préparatoire dans le sens où le modèle est appris avant sa mise en production.

Qu'est-ce que le Machine Learning ? V

Test

On peut aussi retrouver cette phase là sous le nom de phase de *mise en production*.

Le système a été préalablement construit, validé et testé et il peut maintenant être déployé à grande échelle à des fins pratiques !

Lors de cette phase là, le modèle est constamment surveillé afin que son efficacité reste conforme aux observations effectuées lors de son apprentissage. Parfois, des mises à jour sont nécessaires (ré-entraînement du modèle sur des données plus récentes) afin de maintenir son efficacité (on peut parler d'apprentissage *online*).

Données : Chat-Chien I

Regardons de plus près ce que sont nos données



Données : Chat-Chien II

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)

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

Données : Chat-Chien III

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

Données : Chat-Chien IV

Les descripteurs de nos images ne sont pas invariants et peuvent dépendre de plusieurs paramètres ou des conditions de récolte des données.

Exemples : formes ou motifs - diamètres - couleurs

- 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

Ainsi, une même image, prise dans des conditions différentes peut avoir des caractéristiques différentes. (Il existe cependant des algorithmes qui permettent d'identifier des éléments identiques dans images "différentes", comme *SIFT*).

Données : Chat-Chien V

On se contente rarement, et c'est d'autant plus vrai dans un contexte industriel, de travailler sur les données brutes. On va souvent chercher à les transformer pour **créer de nouvelles variables** ou transformer **les variables de bases** :

- A l'aide de la connaissance métier et d'experts qui connaissent les informations importantes ou les facteurs discriminants.
- A l'aide de transformation mathématiques ou de modèles en apprentissage : réseaux de neurones, auto-encodeurs, apprentissage de métrique, noyaux, . . .

Données : Chat-Chien VI

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 complexifie l'apprentissage.**

Données : Chat-Chien VII

Regardons maintenant ces données sur le plan pratique, *i.e.* comment les utiliser. Si on adopte une présentation algorithmique, nous avons :

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



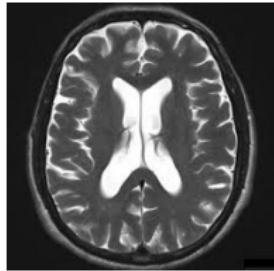
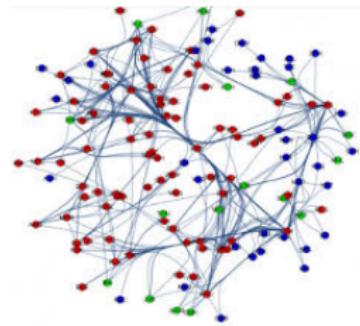
Données : Chat-Chien VIII

Une fois l'entraînement terminé, il est bon de savoir ce qui se passe si notre modèle peut être mis en production. On va donc reproduire les conditions d'une mise en production.

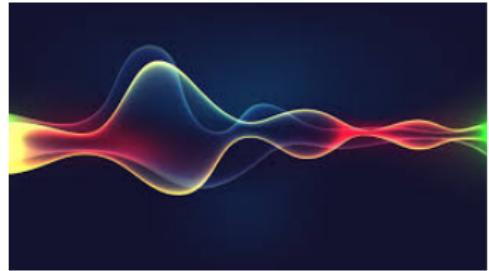
On cherche maintenant à savoir si notre modèle est performant sur de nouvelles données ? Est-ce qu'il est capable d'identifier le genre des images non rencontrées pendant sa phase d'entraînement ?



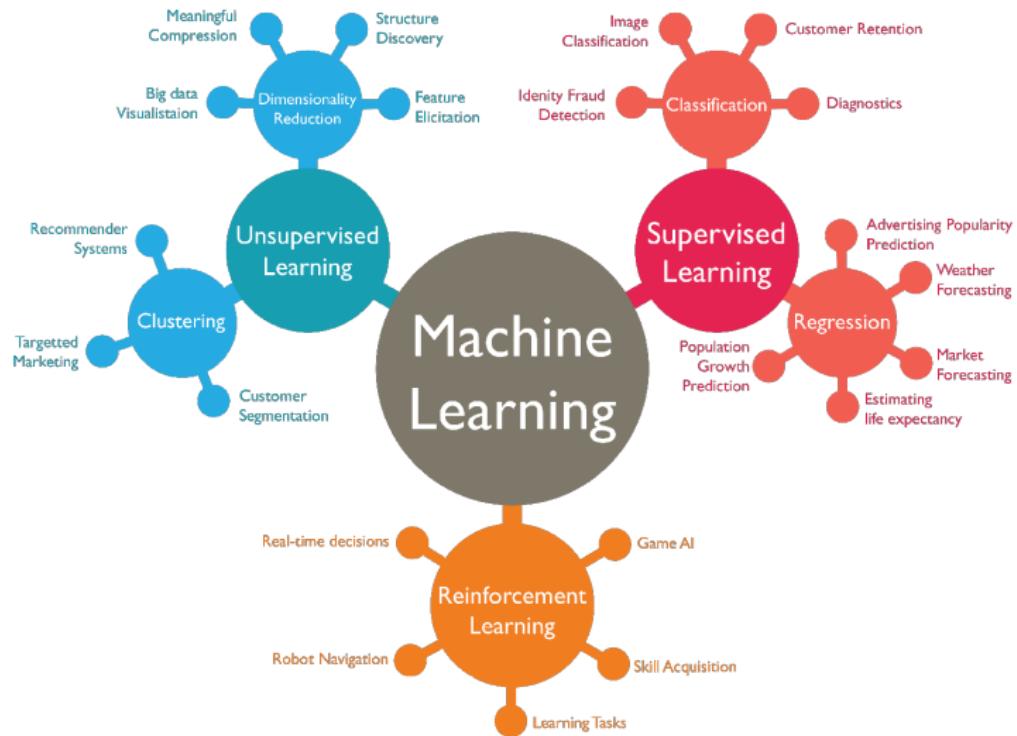
Des données sous toutes leurs formes



144,6	200	25,3	0,7	0,00	0,00	0,00
455,6	200	24,8	1,8	0,00	11,95	1,8
396,7	200	24,5	3,2	0,00	10,15	1,8
2,5	2,9	24,5	0,4	0,00	11,89	0,5
9	10,8	21,8	0,1	0,15	15,78	0,6
126	10,3	0,3	0,00	16,31	0,0	0,00
166	11,8	1,1	-0,06	10,56	0,4	0,00
105	13,2	1,9	-0,03	11,89	1,8	0,00
15	16,9	0,9	0,00	12,81	1,2	0,00
6	18,7	0,4	0,12	10,92	0,0	0,00
10,1	1,7	0,04	0,00	11,91	0,0	0,00



Différents contextes en Machine Learning I



Différents contextes en Machine Learning II

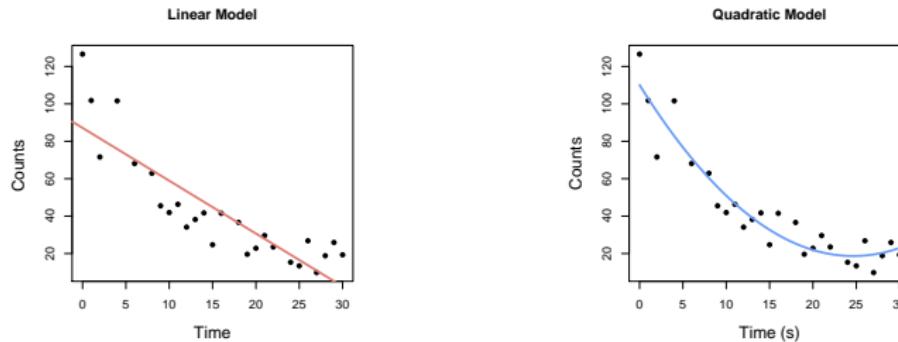
Nous pouvons décomposer le *Machine Learning* en trois grandes catégories principales

- **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

Différents contextes en Machine Learning III

Apprentissage supervisé : Régression

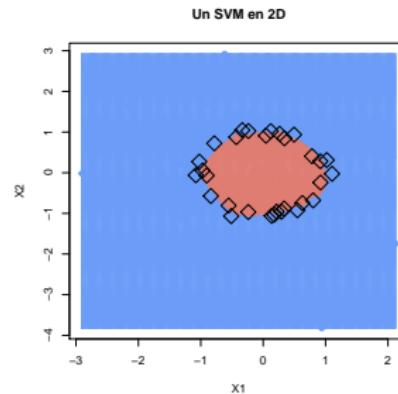
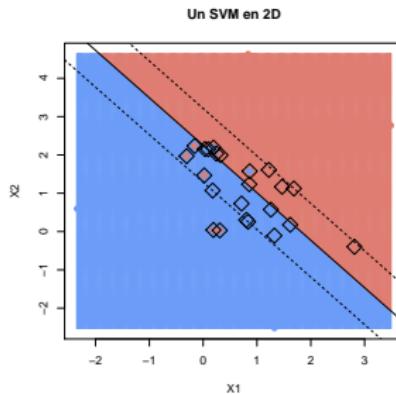
Le système a accès un ensemble de données décrites à la fois par des descripteurs mais également par une étiquette fournie par un "*oracle*". L'objectif sera d'apprendre une "règle" à partir des descripteurs et permettant de prédire la valeur de la donnée mais aussi celle des nouvelles données.



Différents contextes en Machine Learning IV

Apprentissage supervisé : **Classification**

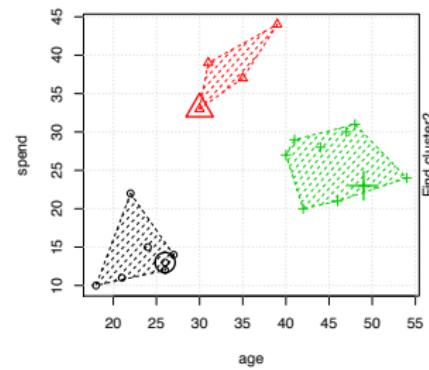
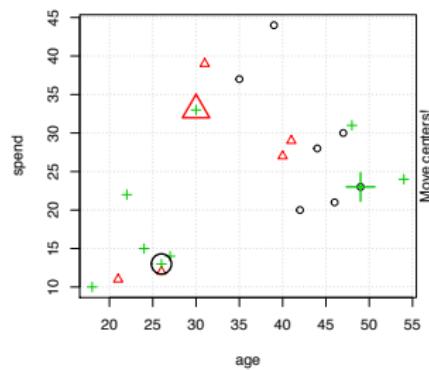
Le système a accès un ensemble de données décrites à la fois par des descripteurs mais également par une étiquette fournie par un "oracle". L'objectif sera d'apprendre une "règle" à partir des descripteurs et permettant de prédire l'étiquette de la donnée mais aussi celle de nouvelles données..



Différents contextes en Machine Learning V

Apprentissage non-supervisé : **Clustering**

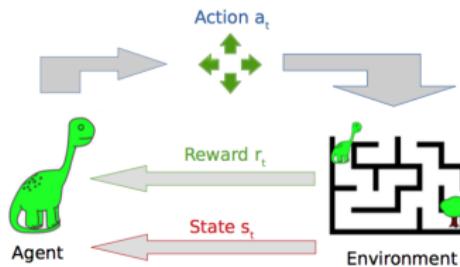
On ne dispose pas d'étiquette lors de la phase d'apprentissage. L'idée principale est de regrouper les exemples présentant des **similarités** ou des **ressemblances** afin de créer des groupes (**clustering**). On peut également utiliser cela pour faire de l'**estimation de densité** ou apprendre une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



Différents contextes en Machine Learning VI

Apprentissage par renforcement

Le 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**.



Données et représentation I

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 ...

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 :
 $\mathbf{x} = (x_1, x_2, \dots, x_d)$ où $\mathbf{x} \in \mathbb{R}^d$

Données et représentation II

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

Les images peuvent se représenter par des vecteurs (après transformation, CNN et "vectorisation")

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.

Normalisation des données I

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 consiste 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 II

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

- **mise à l'échelle** ou **normalisation 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 des données III

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

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

Une hypothèse importante

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.*.

Définition 1.1: Distribution des données

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.

→ 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.

→ Réduire la dimension de l'espace des données !

Apprentissage

Apprentissage I

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
- $\mathcal{Y} = \{1, \dots, C\}$: 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 II

On dispose uniquement d'un échantillon de taille finie

$S = \{\mathbf{x}_i, y_i\}_{i=1}^m \stackrel{i.i.d.}{\sim} \mathcal{D} = \mathcal{X} \times \mathcal{Y}$ 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

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

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 ?

Apprentissage III

Reprendons notre exemple de classification d'image de chats et de chiens.



On cherche à minimiser les erreurs de l'algorithme, ou plus généralement on minimise ce que l'on appelle un **risque**.

Apprentissage IV

Afin d'apprendre et de trouver la meilleure hypothèse h^* , 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**.

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

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

Apprentissage V

En apprentissage supervisé, l'objectif 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.

Apprentissage VI

En apprentissage non-supervisé cette notion de risque se retrouve également mais elle ne fait plus référence à un terme d'erreur.

Elle est simplement employée pour faire référence à la quantité que l'on souhaite minimiser dans un problème minimisation, e.g. une somme de variances dans le cas du *clustering* ou encore une distance entre distributions lorsque l'on fait de l'*Adapatation de Domaine non supervisé*.

Retournons à la notion de risque en Apprentissage Supervisé

Apprentissage VII

Définition 2.1: Risque Empirique

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

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

Apprentissage VIII

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

Mais minimiser les erreurs (de façon binaire) est un problème difficile pour les algorithmes (NP-hard). On cherche donc rarement à minimiser tel quel le taux d'erreur. En pratique, on va plutôt chercher à minimiser un risque basé sur un substitut du taux d'erreur via l'utilisation de **fonctions de coûts/pertes** (**loss functions** en anglais).

Fonctions de loss I

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**.

Définition 2.2: Loss function

Une fonction de loss ℓ est une fonction $\mathcal{H} \times \mathcal{Z} \rightarrow \mathbb{R}^+$ 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.

Fonctions de loss II

Erreur en classification ou 0-1 loss

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

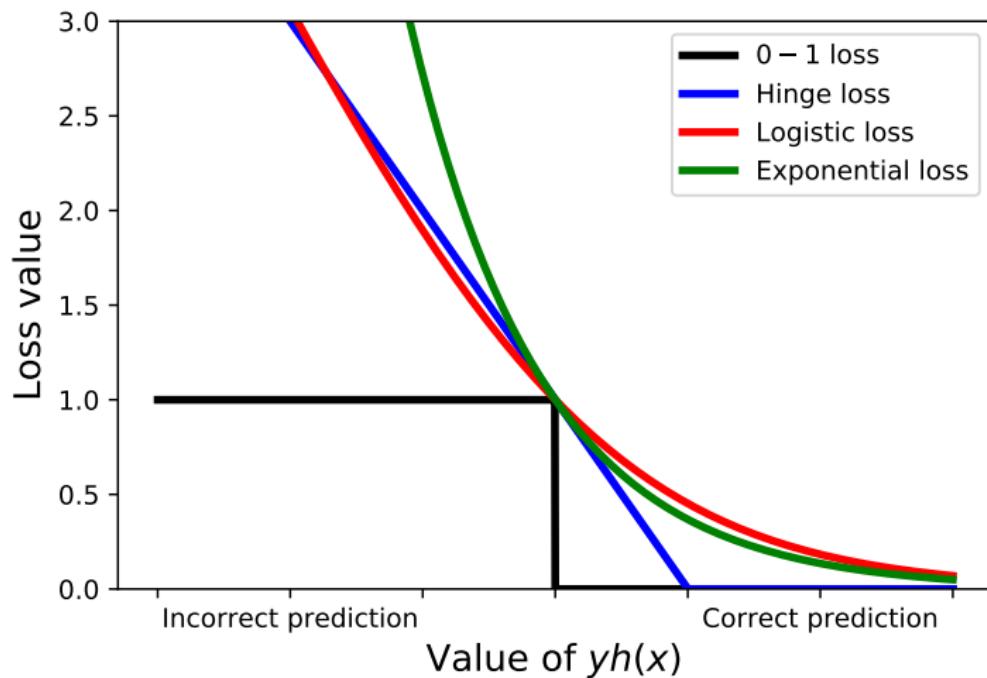
Cette loss présente de nombreux inconvénients : elle n'est pas convexe, elle n'est pas dérivable, donc peu intéressante pour des **algorithmes d'optimisation**.

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

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

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

Fonctions de loss III



Fonctions de loss IV

- **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)

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

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

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

Exercice

Considérez que votre h s'écrit sous la forme $h(\mathbf{x}) = \beta^T \mathbf{x}$.

Montrez que les loss précédentes sont bien des bornes supérieures convexes de la 0 – 1 loss.

Redéfinition du risque

Définition 2.3: Risque Réel/Empirique

Le risque réel d'une hypothèse h selon une loss ℓ , notée $\mathcal{R}\ell_S(h)$ se définit comme la valeur moyenne de la loss sur la distribution. $\mathcal{R}_S(h)$,

$$\mathcal{R}_S^\ell(h) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}} [\ell(h(\mathbf{x}) \neq y)].$$

Soit $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$. Le risque empirique $\mathcal{R}_S(h)$ d'une hypothèse h correspond à l'erreur moyenne de ℓ sur S

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

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

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

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

Cross validation I

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 utilisera pour faire une première évaluation (entraînement - validation) mais on peut faire encore mieux.

Définition 2.4: *k*-fold Cross-Validation

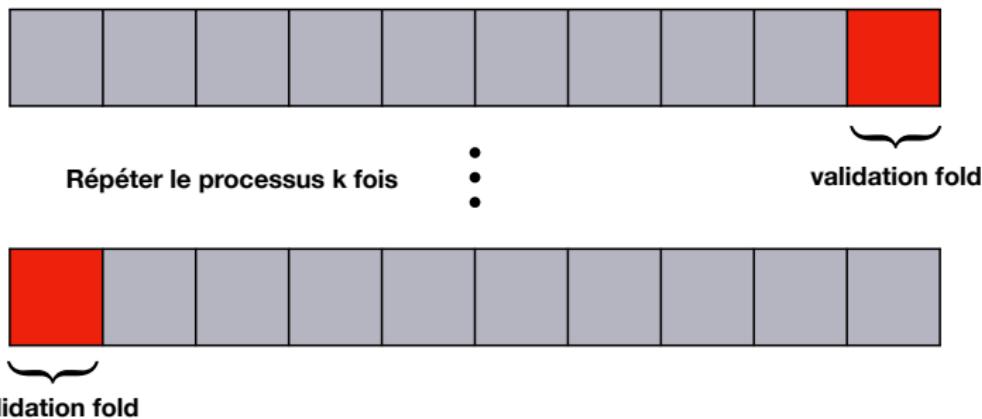
Il s'agit d'un moyen algorithmique permettant d'estimer les performances en généralisation d'un algorithme donné sur des données inconnues.

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.

Cross validation II

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

k fold cross-validation



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

Cross validation III

Erreur moyenne en CV.

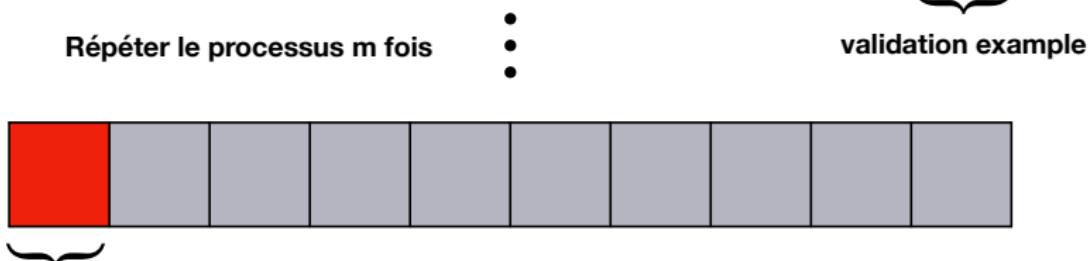
L'évaluation de $\frac{1}{k} \sum_{j=1}^k \mathcal{R}_{S_j}(h)$, appelée erreur moyenne en cross-validation permet de quantifier à quel point notre algorithme est capable de généraliser sur des nouvelles données issues de la même distribution.

C'est donc un bon indicateur pour nous permettre de vérifier si on peut potentiellement mettre notre algorithme en production ou non. On verra que ce n'est pas son seul usage ... De plus, il serait également bon de pouvoir tester notre procédure sur des données non rencontrées par le modèle jusqu'à présent.

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).

Leave and One Out



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 ?

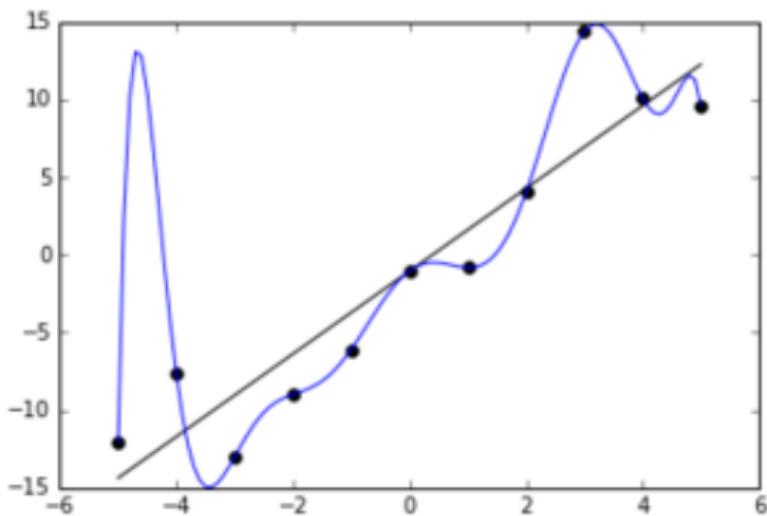
- si l'erreur en validation est **plus faible** que l'erreur sur l'ensemble d'entraînement,
- ou inversement, si l'erreur en validation est **plus** grande que l'erreur sur l'ensemble d'entraînement.

Overfitting ou Sur-Apprentissage I

Définition 2.5: Overfitting

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).

Overfitting ou Sur-Apprentissage II

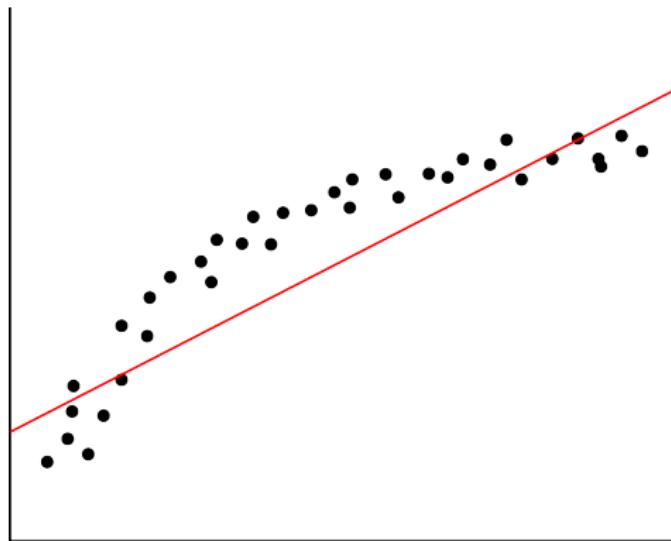


Underfitting ou Sous-Apprentissage I

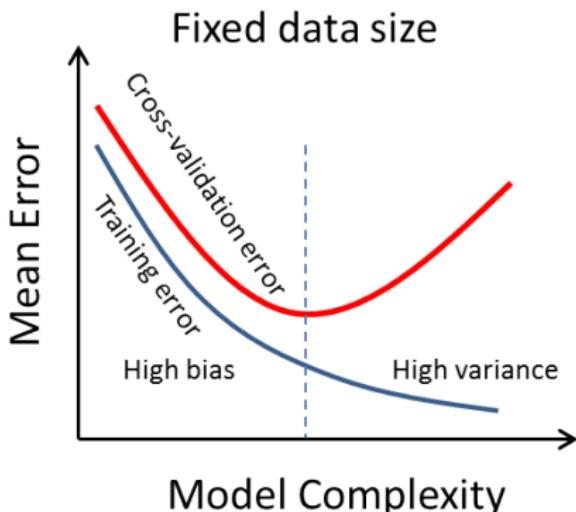
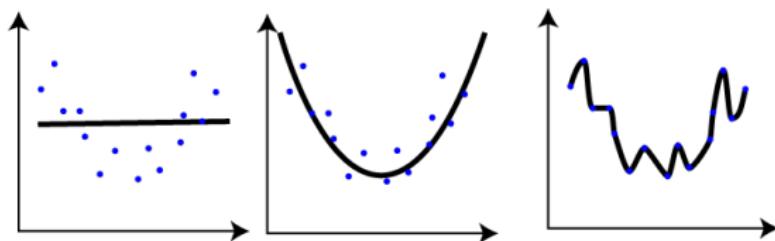
Définition 2.6: Underfitting

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 ou Sous-Apprentissage II



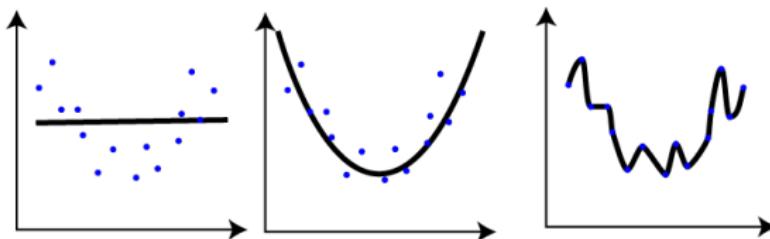
Underfitting-Overfitting I



Comment savoir s'il y a overfitting ou underfitting ?

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

Underfitting-Overfitting II



Considérons que nos hypothèses sont issues d'un ensemble \mathcal{H} , appelé espace d'hypothèses

Overfitting : il intervient lorsque l'espace d'hypothèses est **trop riche**, i.e. si notre espace d'hypothèses est beaucoup **trop grand** et contient **des modèles potentiellement complexes**.

Underfitting : il intervient lorsque l'espace d'hypothèses est **pauvre**, i.e. si notre espace d'hypothèses est beaucoup **trop petit** et contient des modèles qui sont **simples**.

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 **pénalité** dans notre problème d'optimisation.

Pénalité

L'introduction d'un terme de pénalité dans notre problème d'optimisation implique la recherche d'une solution qui trouvera un compromis entre *performances* et *simplicité* :

$$h^* = \arg \min_{h \in \mathcal{H}} \mathcal{R}_S(h) + pen(n, d)$$

Ce terme de pénalité dépend à la fois de dimension d des données et de la taille du jeu de données n . Il est, en général croissant par rapport n .

$$pen(n, d) \Leftrightarrow \mathcal{H}_n$$

$$\mathcal{H}_1 \subset \mathcal{H}_2 \subset \dots \subset \mathcal{H}_n$$

Cette notion est en générale complexe à déterminer, surtout lorsqu'elle évolue en fonction de taille de l'échantillon.

Pour cette raison, on ne cherche que très rarement à passer par cette approche pour trouver un modèle à la fois simple et performant.

Régularisation

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

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

Définition 2.7: Régularisation

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 I

Le nouveau problème d'optimisation s'écrit alors :

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

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.

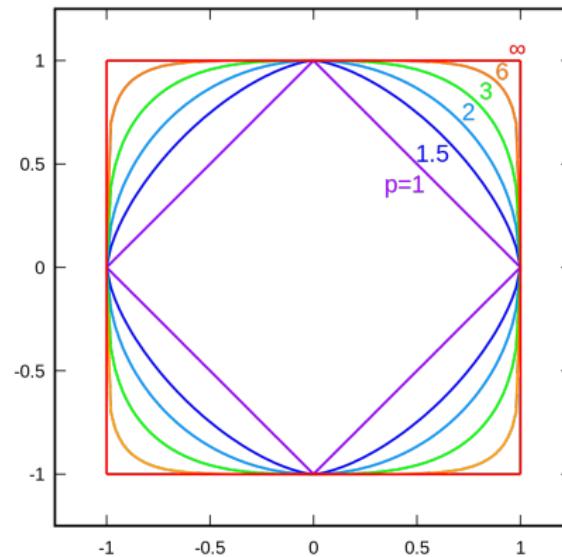
Risque empirique régularisé II

Ce type de problème a déjà été rencontré dans le cadre de modèle linéaire, plus précisément lorsque l'on a cherché à apprendre un modèle linéaire qui soit **parcimonieux**, *i.e.* dont les solutions ne dépendent que d'un nombre restreint de variables, *i.e.*

$$\min_{\beta \in \mathbb{R}^{d+1}} \|\mathbf{y} - \mathbf{X}\beta\| + \lambda \|\beta\|_1$$

Normes usuelles

Quelques exemples de normes : $\|h\|_p = \left(\sum_{i=1}^d h_i^p \right)^{\frac{1}{p}}$.



Exercices I

- Montrez que la fonction $\mathbf{x} \mapsto \|\mathbf{x}\|_2^2$ est convexe.
- Faire de même pour n'importe quel entier p de la définition précédente (la preuve de l'inégalité triangulaire se fait à l'aide de la question suivante).
- Soit p et q deux nombres vérifiant

$$\frac{1}{p} + \frac{1}{q} = 1$$

et deux vecteurs \mathbf{x} et \mathbf{y} de \mathbb{R}^d , $\alpha_1, \dots, \alpha_d > 0$ tels que $\sum_{k=1}^d \alpha_k = \alpha$.
Montrer que l'on a

$$\sqrt[d]{\prod_{k=1}^d x_i} \leq \frac{1}{d} \sum_{k=1}^d x_i.$$

Exercices II

En déduire que l'on a, plus généralement

$$\sqrt[\alpha]{\prod_{k=1}^d x_i^{\alpha_i}} \leq \frac{1}{\alpha} \sum_{k=1}^d \alpha_i x_i$$

Se servir de la question précédente pour montrer que

$$\|\mathbf{x}\mathbf{y}\|_1 \leq \|\mathbf{x}\|_p \|\mathbf{y}\|_q$$

En déduire l'inégalité triangulaire

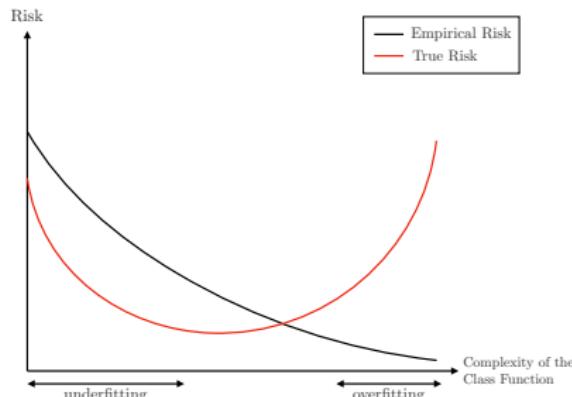
$$\|\mathbf{x} + \mathbf{y}\|_p \leq \|\mathbf{y}\|_p + \|\mathbf{x}\|_p.$$

Rôle et choix du paramètre λ

Retournons à notre problème d'optimisation

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

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

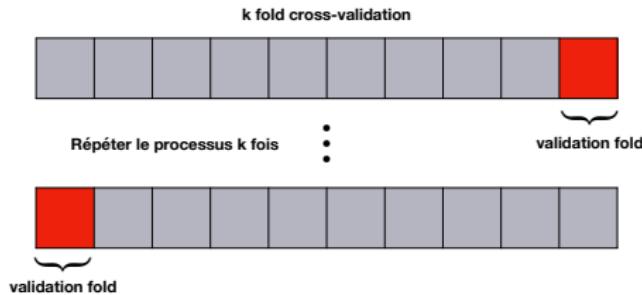
Notre ensemble d'apprentissage peut aussi servir à vérifier les capacités qu'à notre modèle **à généraliser pour le choix d'un hyper-paramètre en particulier**

- 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 :
 $\lambda \in (\lambda_1, \lambda_2, \dots, \lambda_p)$
- 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

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

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

Bornes en généralisation I

Comme nous venons de la voir, l'objectif de ces bornes est de mesurer d'évaluer la probabilité de mesurer un certain écart entre le risque empirique et le risque généralisé.

Ces études sont relativement anciennes et sont issues de l'apprentissage PAC (Probably Approximately Correct) et notamment des bornes PAC [Valiant, 1984].

$$\mathbb{P}(|\mathcal{R}(\cdot) - \mathcal{R}_S(\cdot)| \geq \varepsilon) \leq \delta,$$

où $\varepsilon > 0$ et $\delta \in [0, 1]$.

Bornes en généralisation II

Les bornes sont généralement présentées sous la forme suivante

$$|\mathcal{R}(\cdot) - \mathcal{R}_S(\cdot)| \leq \varepsilon(\delta, m),$$

où $1 - \delta$ s'interprète alors comme un niveau de confiance dans votre borne. La fonction ε est alors décroissante en le nombre d'exemples m mais croissante en δ . On pourra montrer que le taux de convergence est en général en $\mathcal{O}(\ln(m)\sqrt{m})$ ou parfois même en $\mathcal{O}(1\sqrt{m})$.

Bornes en généralisation III

Déviation Uniforme

La construction de telles bornes repose sur la convergence du risque empirique vers le risque généralisé (y voir la loi forte des grands nombres)

$$\lim_{m \rightarrow \infty} \frac{1}{m} \sum_{i=1}^m \ell(y_i, h(\mathbf{x}_i)) = \mathbb{E}_{(y, \mathbf{x}) \sim \mathcal{D}} [\ell(y, h(\mathbf{x}))].$$

On parlera de bornes *uniformes* car elles sont valables pour toutes les hypothèses $h \in \mathcal{H}$.

$$\left\{ \sup_{h \in \mathcal{H}} \left| \mathcal{R}^\ell(h) - \mathcal{R}_S^\ell(h) \right| \geq \varepsilon \right\}.$$

Bornes en généralisation IV

Ces bornes s'établissent en général avec des résultats probabilistes comme les inégalités de concentration. De plus, certaines de ces bornes peuvent dépendre de la complexité de l'espace d'hypothèses [Bousquet et al., 2004].

Théorème 2.1: Uniform Generalization Bound

Soit \mathcal{H} un espace d'hypothèses de taille finie, S un ensemble d'apprentissage de taille m obtenu de façon *i.i.d.* de \mathcal{D} , ℓ une loss prenant ses valeurs dans $[0, 1]$ et $\delta > 0$. Alors, pour tout $h \in \mathcal{H}$, avec probabilité au moins $1 - \delta$, nous avons :

$$\mathcal{R}^\ell(h) \leq \mathcal{R}_S^\ell(h) + \sqrt{\frac{\ln |\mathcal{H}| + \ln(2/\delta)}{2m}}.$$

Bornes en généralisation V

En général l'espace \mathcal{H} est rarement de taille finie, il suffit de s'imaginer l'ensemble des droites de régression dans un plan ou encore l'ensemble des droites qui pourraient séparer une collection de points avec deux étiquettes différentes.

Il existe des mesures qui permettent d'évaluer la complexité des espaces d'hypothèses comme la *Dimension de Vapnik-Chervonenkis* (VC-dim) [Vapnik and Chervonenkis, 1971] ou encore la *Complexité de Rademacher* [Koltchinskii and Panchenko, 2000].

Ces notions sont plutôt difficiles à manipuler on va plutôt se concentrer sur une autre approche dans le cadre de ce cours.

Bornes en généralisation VI

Stabilité Uniforme

C'est une approche plus récente pour établir ces bornes probabilistes et qui ne se passe pas par l'évaluation de la complexité de l'espace d'hypothèses.

A la place on se concentrera uniquement sur la valeur de l'hyper-paramètre qui contrôle la complexité de l'espace d'hypothèses. L'inconvénient de cette méthode est qu'elle ne s'applique qu'à des **problèmes d'optimisation convexes**.

En quelques mots, on va regarder l'impact d'une modification de l'échantillon d'apprentissage sur la loss. Plus formellement

Bornes en généralisation VII

Définition 2.8: Uniform Stability

Un algorithme d'apprentissage \mathcal{A} a une constante de stabilité uniforme en $\frac{\beta}{m}$ vis à vis de la fonction de coût ℓ et de l'ensemble des paramètres θ , où $\beta > 0$ si :

$$\forall S, \forall i, 1 \leq i \leq m, \sup_{\mathbf{x}} |\ell(\boldsymbol{\theta}_S, \mathbf{x}) - \ell(\boldsymbol{\theta}_{S^i}, \mathbf{x})| \leq \frac{\beta}{m},$$

où S est notre ensemble d'apprentissage de taille m , $\boldsymbol{\theta}_S$ les paramètres du modèle appris avec S , $\boldsymbol{\theta}_{S^i}$ les même paramètres appris avec S^i où S^i est obtenu en remplaçant le i -ème exemple \mathbf{x}_i de S par un autre exemple \mathbf{x}'_i tiré selon \mathcal{D} . $\ell(\boldsymbol{\theta}_S, \mathbf{x})$ est la valeur de loss évaluée en \mathbf{x} .

Bornes en généralisation VIII

Théorème 2.2: Borne via Stabilité Uniforme

Soit $\delta > 0$ et $m > 1$. Soit un algorithme ayant une constante de stabilité uniforme en β/m vis à vis d'une fonction de coût ℓ bornée par K . Alors, avec probabilité au moins $1 - \delta$ selon le tirage de S , nous avons :

$$\mathcal{R}^\ell(\theta_S) \leq \mathcal{R}_S^\ell(\theta_S) + \frac{2\beta}{m} + (4\beta + K)\sqrt{\frac{\ln 1/\delta}{2m}},$$

où $\mathcal{R}^\ell(\cdot)$ est le risque réel $\mathcal{R}_S^\ell(\cdot)$ est le risque empirique évalué sur S .

Quelques algorithmes

Sommaire

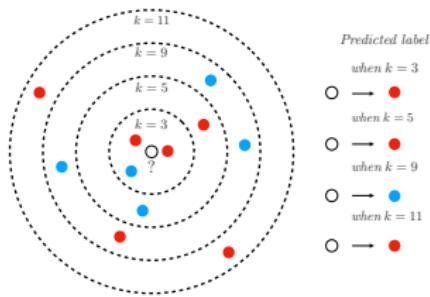
On va rapidement présenter quelques algorithmes assez simples :

- l'algorithme du plus proche voisin
- la régression linéaire (simple et multiple)
- la régression logistique
- les séparateurs à vaste marge
- arbres de décisions et forêts aléatoires
- réseaux de neurones

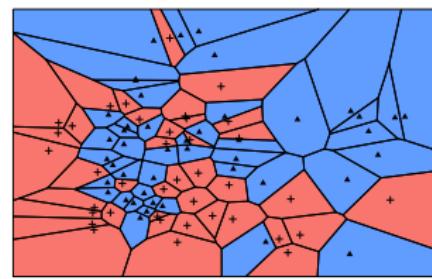
Certains de ces algorithmes et parfois d'autres, non mentionnés ici, seront abordés/traités en TD.

k -NN I

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

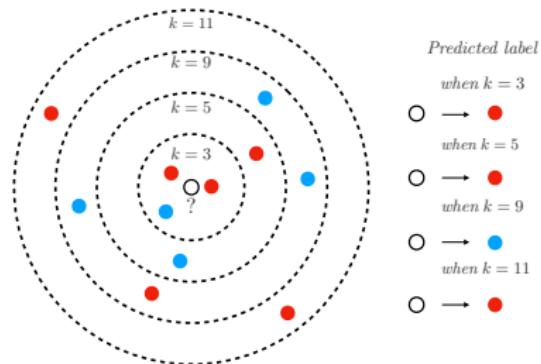


Régions de Voronoï, $k = 1$

k -NN II

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^* si

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



k -NN III

Définition 3.1: 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

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

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

k -NN IV

- 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 V

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 → **Metric Learning**

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

k -NN VI

Il existe des méthodes de pré-traitement qui permettent de réduire le temps de calcul de cet algorithme, en supprimant des données en ou en créant des groupes ou encore en réduisant la dimension.

Condensed Nearest Neighbor

Input: Echantillon d'apprentissage S

Output: Un échantillon S' plus petit que S

begin

 Séparer S en deux ensembles aléatoires S_1 et S_2

while S_1 et S_2 sont modifiés **do**

 Retirer de S_1 tous les exemples mal classifiés à l'aide de S_2 et
 d'un 1-NN

 Retirer de S_2 tous les exemples mal classifiés à l'aide de S_1 et
 d'un 1-NN

$S' = S_1 \cup S_2$;

return S'

On peut aussi faire clustering (créer des modèles locaux) ou encore de l'ACP (réduction de dimension).

k -NN VII

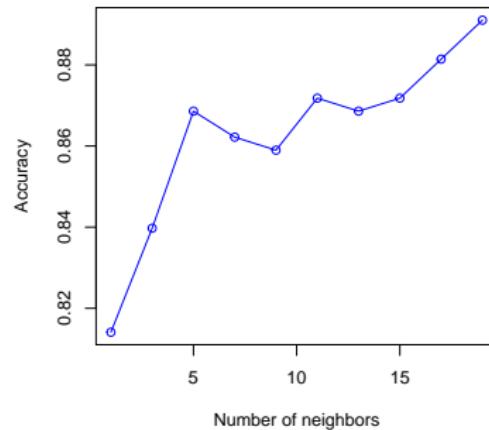
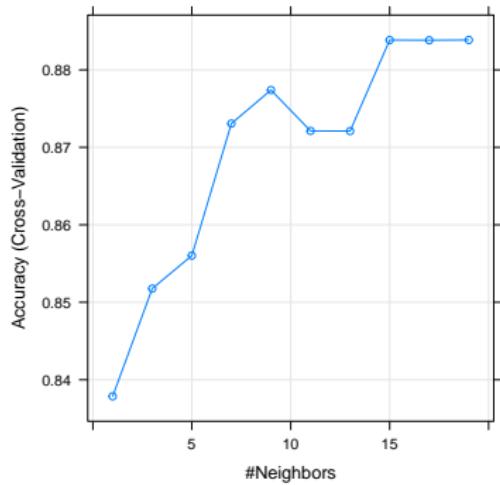
En pratique

C'est un algorithme non paramétrique, *i.e.* il n'y aucun paramètre à apprendre ! Il y a simplement un hyper-paramètre à tuner : k , le nombre de voisins.

On pourra regarder les packages **caret** de  ou encore la librairie **model_selection** de Python pour effectuer cette cross-validation.

k -NN VIII

Un exemple en application de la cross-validation



k -NN IX

Pour finir

C'est un algorithme simple à implémenter et qui peut parfois fournir de très bons résultats.

Il est à la base de plusieurs autres algorithmes plus complexes basées sur le Metric Learning (apprentissage de représentation) mais peut aussi servir à faire du pré-traitement sur les données (e.g. des méthodes d'échantillonnage comme on l'a vu avec CNN par exemple).

En revanche, brute, il ne sera que très peu utile/efficace lorsque l'on travaille avec une quantité importante de données.

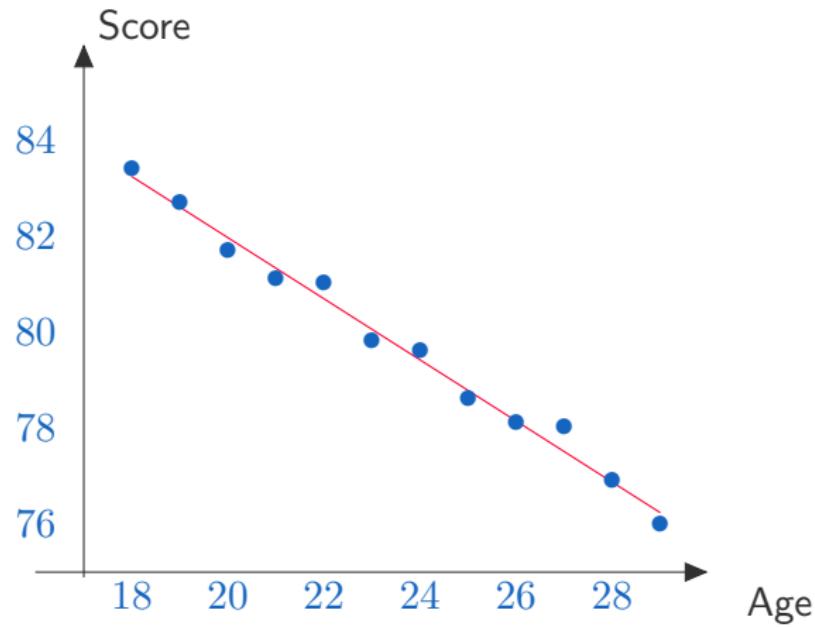
Régression linéaire |

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 le score obtenu par une personne à un test donné en fonction de son âge.

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

Régression linéaire II



Régression linéaire III

Définition 3.2: Modèle Linéaire 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.

Régression linéaire : formalisation I

Objectif : Minimiser l'erreur de prédiction, i.e. l'écart entre la valeur réelle et la valeur prédictée.

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)**

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

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

Régression linéaire : formalisation II

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

Régression linéaire : formalisation III

L'estimation des paramètres du modèle se fait par la **méthodes 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 et que le problème est relativement simple, on peut aisément trouver une solution sous forme close, *i.e.* une expression des solutions.

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

- Rappeler les hypothèses du modèle linéaires gaussien.
- Montrer que le problème d'optimisation est bien convexe.
- Résoudre le problème de la méthode des moindres carrés dans le cas où notre modèle ne comprend que deux paramètres β_0 et β_1 .
- Rappeler comment tester la significativité des paramètres de la droite de régression.

Régression linéaire multiple I

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

- $Y = (y_1, \dots, y_m)$ est un vecteur de \mathbb{R}^m est **la variable à expliquer**
- $X = (1, \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_d)$ 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
- $\beta = (\beta_0, \beta_1, \dots, \beta_d)$ 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 II

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

On cherche à résoudre le problème d'optimisation suivant :

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

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

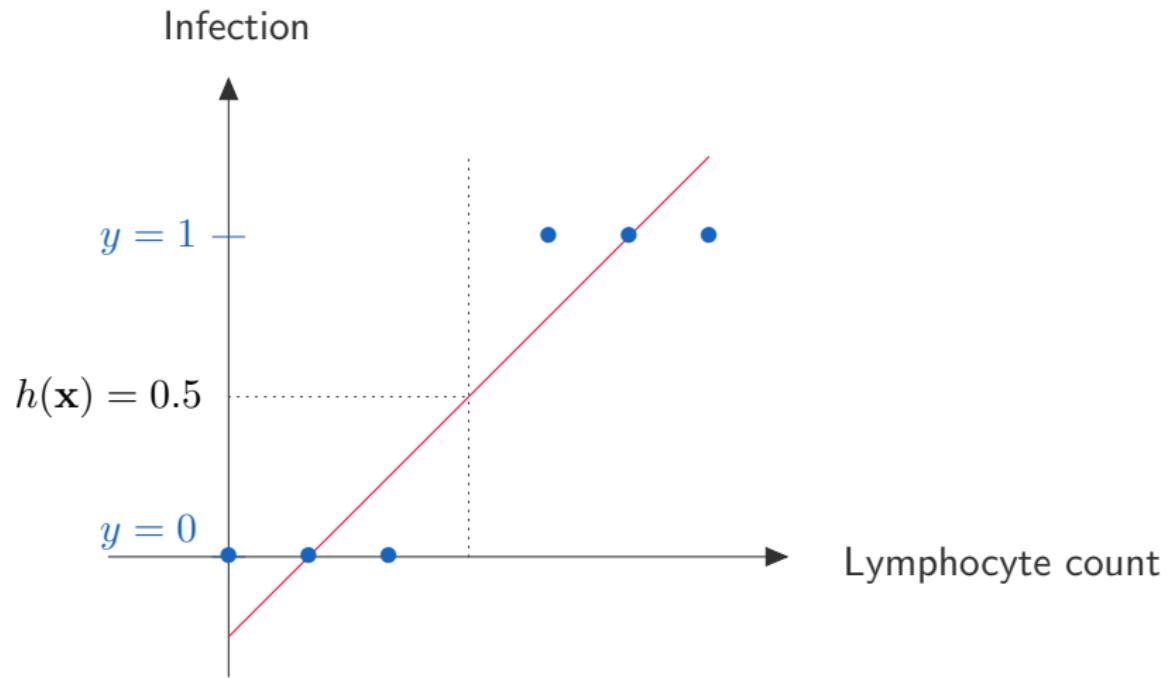
$$\beta = (X^T X)^{-1} X^T Y,$$

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

Exercice

- Retrouver l'expression du paramètre β donnée dans le slide précédent, en résolvant le problème des MCO.
- Retrouver le même résultat en passant par la calcul de la vraisemblance de votre modèle.

De la régression vers la classification I



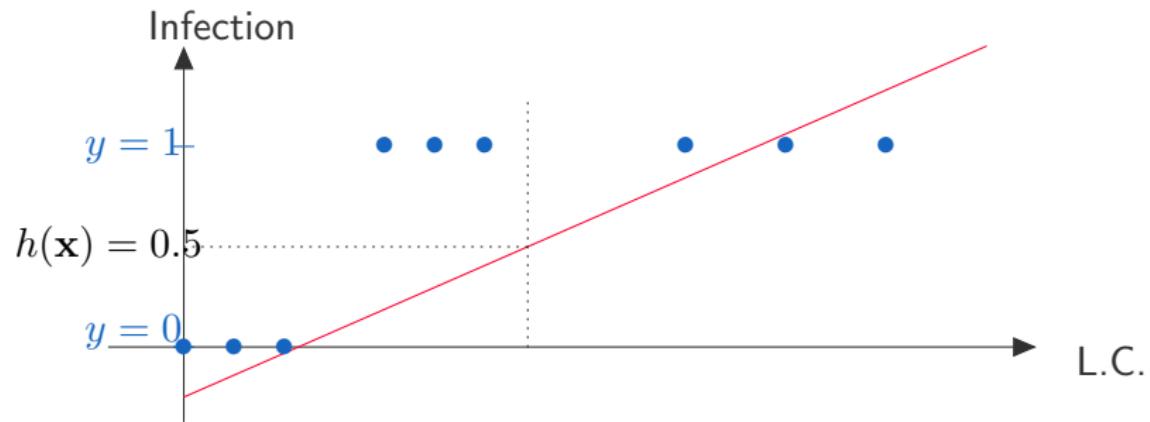
De la régression vers la classification II

Dans cet exemple, on souhaite déterminer si une personne présente une infection en fonction de sa numération en lymphocytes.

Le problème que l'on considère devient alors un problème de classification, mais qu'est-ce qui nous empêche de le voir a priori comme un problème de régression ?

Si on reprend le jeu de données présentés précédemment, tout semble se passer pour le mieux. Mais est-ce toujours le cas si on modifie un peu ces données ?

De la régression vers la classification III



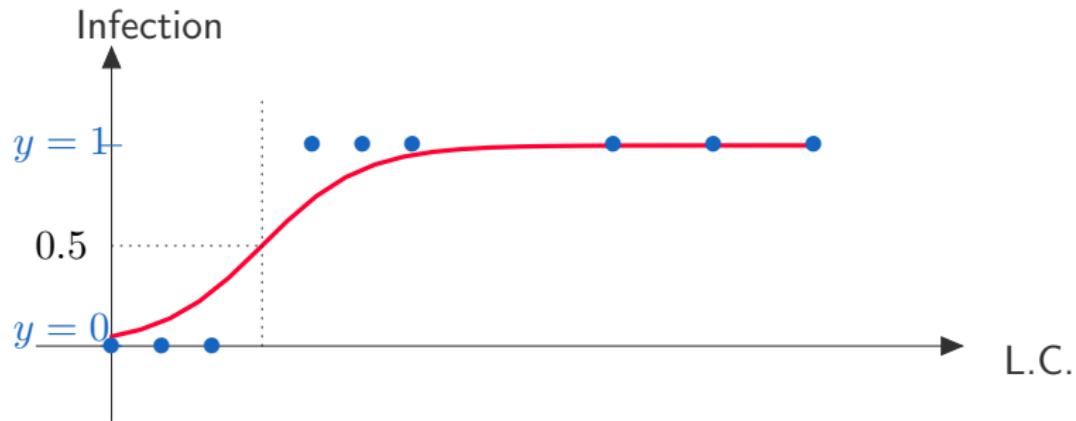
De la régression vers la classification IV

Cet exemple très simple montre que le modèle de régression employé n'est pas approprié et qu'il va certainement falloir voir la valeur que l'on cherche à prédire y non pas comme un nombre réel, plutôt comme un identifiant.

Problème : le modèle actuel renvoie des valeurs réelles alors que ce que l'on cherche à prédire prend ses valeurs dans l'ensemble $\{0, 1\}$.

Solution : il faudrait trouver une fonction qui permet de modéliser le problème et qui prend ses valeurs dans l'intervalle $[0, 1]$, *i.e.* transformer les valeurs de notre modèle linéaire pour qu'elles se retrouvent dans l'intervalle $[0, 1]$.

De la régression vers la classification V



De la régression vers la classification VI

Ces valeurs comprises dans l'intervalle $[0, 1]$ peuvent s'interpréter comme des probabilités !

Ce que l'on va donc faire, c'est estimer la probabilité qu'un patient soit malade en fonction de sa numération en lymphocytes. Pour cela on va utiliser une transformation à l'aide d'une fonction logistique

$$\frac{1}{1 + \exp(-h(\mathbf{x}))}.$$

On parle alors de **Régression Logistique**

Régression Logistique I

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 à la classe positive, *i.e.* que $y = 1$.

Régression Logistique II

On va donc chercher un modèle qui va approximer la probabilité suivante

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

Pour faire cela, on va chercher un modèle linéaire qui va approximer l'*odd ratio* entre la classe positive et la classe négative.

Principe 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 :

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

Régression Logistique III

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**.

$$\mathcal{L}(\mathbf{w}, b, S) = \prod_{i=1}^m \mathbb{P}(Y = y_i \mid X = x_i),$$

↓ separer $y_i = 0$ and $y_i = 1$

$$= \prod_{i=1, y_i=1}^m \mathbb{P}(Y = y_i \mid X = x_i) \times \prod_{i=1, y_i=0}^m \mathbb{P}(Y = y_i \mid X = x_i),$$

↓ Utiliser la loi de Bernoulli

Régression Logistique IV

$$= \prod_{i=1}^m \left(\frac{1}{1 + \exp(-h(\mathbf{w}, b, \mathbf{x}_i))} \right)^{y_i} \times \left(\frac{1}{1 + \exp(h(\mathbf{w}, b, \mathbf{x}_i))} \right)^{(1-y_i)}$$

Plutôt que de chercher à maximiser la vraisemblance, on parfois chercher à minimiser l'opposé de la log-vraisemblance des données.

$$\begin{aligned}\ell(\mathbf{w}, b, S) &= -\ln (\mathfrak{L}(\mathbf{w}, b, S)), \\ &= -\sum_{i=1}^m y_i \ln \left(\frac{1}{1 + \exp(-(\langle \mathbf{w}, \mathbf{x}_i \rangle + b))} \right) \\ &\quad + (1 - y_i) \ln \left(1 - \frac{1}{1 + \exp(-(\langle \mathbf{w}, \mathbf{x}_i \rangle + b))} \right).\end{aligned}$$

Régression Logistique V

On passe les détails mathématiques concernant le calcul de la fonction à optimiser.

Ce qu'il faut retenir dans le cas présent c'est que, contrairement au modèle linéaire standard, il n'existe pas de solutions en forme close pour le problème de la régression logistique.

La fonction à optimiser reste cependant convexe, il est donc possible de trouver son minimum à l'aide d'un algorithme de descente de gradient comme la méthode de Newton ou d'approximation de Newton (voir la librairie **Scipy** sous Python et plus précisément la fonction *optimize*).

Régression Logistique VI

Règle de prédiction

Une fois que l'on dispose de notre modèle, nous sommes donc capable d'évaluer la probabilité qu'une nouvelle donnée 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 :

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

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

A propos des SVM I

Certainement l'un des algorithmes les plus répandus 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 : $\{-1, +1\}$:

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

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

Références I

-  Bousquet, O., Boucheron, S., and Lugosi, G. (2004).
Introduction to Statistical Learning Theory, pages 169–207.
Springer Berlin Heidelberg.
-  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.
-  Koltchinskii, V. and Panchenko, D. (2000).
Rademacher processes and bounding the risk of function learning.
In Giné, E., Mason, D. M., and Wellner, J. A., editors, *High Dimensional Probability II*, pages 443–457. Birkhäuser Boston.

Références II

-  Valiant, L. G. (1984).
A theory of the learnable.
Communication of the ACM, 27(11) :1134–1142.
-  Vapnik, V. and Chervonenkis, A. (1971).
On the uniform convergence of relative frequencies of events to their probabilities.
Theory of Probability & Its Applications, 16(2) :264–280.
-  Vapnik, V. and Cortes, C. (1995).
Support-vector networks.
Machine Learning, 20 :273–297.