Introduction aux méthodes d'apprentissage

Xavier Gendre

ISAE

Préliminaires

Principe général

Une **méthode statistique** se construit à partir des **observations** du phénomène étudié.

Ce cours se concentre sur des méthodes d'**apprentissage supervisé** qui ont comme objectif de mettre en relation des **entrées** et des **sorties**. Les observations sont ainsi des paires entrée-sortie

$$(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$$

où les espaces $\mathcal{X}$ et $\mathcal{Y}$ sont pour le moment quelconque.

Les observations $x_1, \ldots, x_n \in \mathcal{X}$ sont celles d'une **variable d'entrée** x et les observations $y_1, \ldots, y_n \in \mathcal{Y}$ sont celles d'une **variable de sortie** y.

Objectif

Expliquer ou prédire une sortie y en fonction d'une entrée x.

En apprentissage supervisé, cet objectif peut être :

- **descriptif** : comment l'entrée agit sur la sortie **pour les observations**,
- **prédictif** : comment prédire la sortie **pour une nouvelle observation** de la variable d'entrée.

Objectif

Expliquer ou prédire une sortie y en fonction d'une entrée x.

En apprentissage supervisé, cet objectif peut être :

- **descriptif** : comment l'entrée agit sur la sortie **pour les observations**,
- **prédictif** : comment prédire la sortie **pour une nouvelle observation** de la variable d'entrée.

Dans la suite de ce cours, nous nous concentrons principalement sur l'aspect descriptif. Les questions prédictives utilisent un cadre stochastique pour quantifier la qualité de la prédiction mais cela demande un outillage théorique plus conséquent.

Différents problèmes

La variable de sortie y peut être

- **quantitative** : $\mathcal{Y} \subset \mathbb{R}^d$,
- **qualitative / catégorielle** : $\mathcal{Y}$ est fini et sans ordre.

Il existe d'autres types de variables que l'on rencontre moins souvent telles que les variables **ordinales** ($\mathcal{Y}$ fini avec un ordre). Des méthodes d'apprentissage supervisé peuvent généralement s'adapter pour prendre cela en compte.

Différents problèmes

La variable de sortie y peut être

- **quantitative** : $\mathcal{Y} \subset \mathbb{R}^d$,
- **qualitative / catégorielle** : $\mathcal{Y}$ est fini et sans ordre.

Il existe d'autres types de variables que l'on rencontre moins souvent telles que les variables **ordinales** ($\mathcal{Y}$ fini avec un ordre). Des méthodes d'apprentissage supervisé peuvent généralement s'adapter pour prendre cela en compte.

Une façon de reformuler l'objectif de l'apprentissage supervisé est la recherche d'une fonction $f : \mathcal{X} \to \mathcal{Y}$ telle que l'image d'une entrée $x \in \mathcal{X}$ explique aussi bien que possible la sortie $y \in \mathcal{Y}$:

$$f(x) \simeq y.$$

Le sens du symbole "$\simeq$" reste à préciser mais il s'agit de l'idée générale selon laquelle nous voulons expliquer une sortie à partir d'une entrée.

Le vocabulaire dépend de la nature de la sortie :

- lorsque y est quantitative, nous parlons de **régression**,
- lorsque y est qualitative, nous parlons de **classification**.

En pratique, la nature des entrées doit également être prise en compte. Par exemple, il n'est pas possible de faire des combinaisons linéaires de variables d'entrée qualitatives.

Exemple : Ozone

Entrées : mesures quotidiennes de variables météorologiques (température, nébulosité, vent, ...)
Sortie : mesures quotidiennes de la concentration en ozone

Jour	O3	T	V	N
1	63.6	13.4	9.35	7
2	89.6	15.0	5.4	4
3	79.0	7.9	19.3	8
⋮	⋮	⋮	⋮	⋮

Question : peut-on prédire la concentration en ozone du lendemain à partir des prévisions météorologiques ?

Exemple : Spam

Entrées : mesures textuelles sur le contenu de mails (fréquence de certains mots, fréquence de certains caractères, nombre de majuscules, ...)
Sortie : étiquette (*a.k.a.* tag) "Mail" ou "Spam"

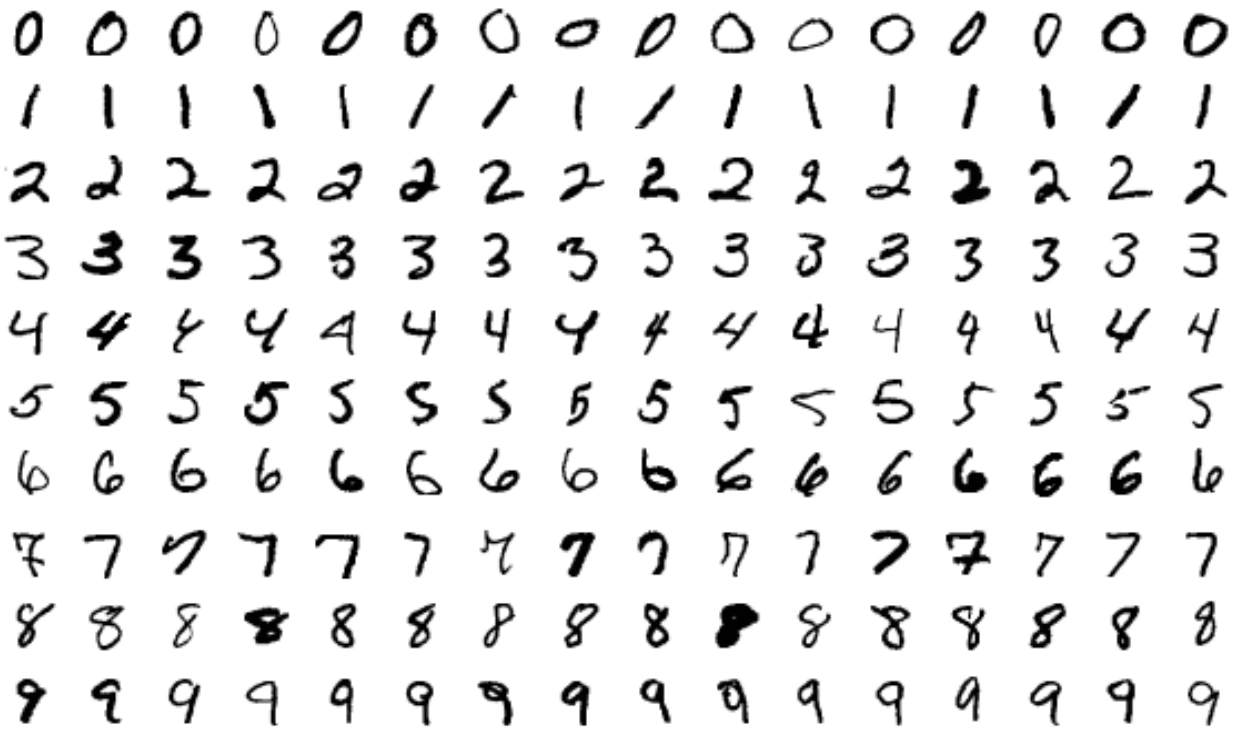
Id	Fenlarge	Fdollar	Fcap	...	Tag
1	0.18	0.06	0.10	...	Spam
2	0.00	0.08	0.09	...	Mail
3	0.01	0.00	0.01	...	Mail
⋮	⋮	⋮	⋮	⋮	⋮

Question : peut-on construire à partir de ces données une méthode de détection automatique de spam ?

Exemple : MNIST

Entrées : des images scannées de chiffres (0 à 9) manuscrits sur des codes postaux

Sortie : la valeur du chiffre inscrit sur l'image



Question : peut-on prédire le chiffre inscrit sur une nouvelle image ?

Exemple	Entrées	Sortie	Problème
Ozone	Données météo	Concentration O_3	Régression
Spam	Mesures textuelles	"Mail" ou "Spam"	Classification
MNIST	Images	Chiffre de 0 à 9	Classification

Formalisation mathématique

À partir d'un **jeu de données** (*a.k.a.* **data set**),

$$(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y},$$

nous cherchons à prédire/expliquer les $y_k \in \mathcal{Y}$ en fonction des $x_k \in \mathcal{X}$. Il s'agit de trouver un **prédicteur** $f : \mathcal{X} \to \mathcal{Y}$ tel que

$$\forall k \in \{1, \ldots, n\}, \ f(x_k) \simeq y_k.$$

Formalisation mathématique

À partir d'un **jeu de données** (*a.k.a.* **data set**),

$$(x_1, y_1), \dots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y},$$

nous cherchons à prédire/expliquer les $y_k \in \mathcal{Y}$ en fonction des $x_k \in \mathcal{X}$. Il s'agit de trouver un **prédicteur** $f : \mathcal{X} \to \mathcal{Y}$ tel que

$$\forall k \in \{1, \dots, n\}, \ f(x_k) \simeq y_k.$$

Il est nécessaire de se donner un **critère** pour mesurer la qualité d'un prédicteur. Pour cela, nous utilisons une **fonction de perte** $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ qui vérifie :

- **Positivité :** $\forall y, y' \in \mathcal{Y}, \ \ell(y, y') \geqslant 0$
- **Séparation :** $\forall y, y' \in \mathcal{Y}, \ \ell(y, y') = 0 \iff y = y'$
- **Symétrie :** $\forall y, y' \in \mathcal{Y}, \ \ell(y, y') = \ell(y', y)$ (en option)

Le choix de la fonction de perte est **crucial**, c'est elle qui donne un sens à la performance de telle ou telle méthode. Un algorithme performant pour un critère ne sera pas forcément performant pour un autre.

Avant de considérer un de construire un algorithme d'apprentissage supervisé, il est **capital** de savoir mesurer la performance d'un prédicteur.

Le choix de la fonction de perte est **crucial**, c'est elle qui donne un sens à la performance de telle ou telle méthode. Un algorithme performant pour un critère ne sera pas forcément performant pour un autre.

Avant de considérer un de construire un algorithme d'apprentissage supervisé, il est **capital** de savoir mesurer la performance d'un prédicteur.

Exemples de fonctions de perte :

- en régression, le critère des **moindres carrés** correspond à

$$\forall y, y' \in \mathbb{R},\ \ell(y, y') = (y - y')^2$$

- en classification, il est possible (mais compliqué comme nous le verrons) de considérer une indicatrice,

$$\forall y, y' \in \mathcal{Y},\ \ell(y, y') = \begin{cases} 1 & \text{si } y = y', \\ 0 & \text{sinon.} \end{cases}$$

Point de vue statistique (Machine Learning)

Pour construire un prédicteur $\hat{f}$ dans une classe de fonction $\mathcal{F}$, le principe consiste généralement à minimiser le **risque empirique**, *i.e.* la moyenne des pertes sur les observations,

$$\hat{f} \in \operatorname*{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{k=1}^{n} \ell(y_k, f(x_k)).$$

Dans un contexte général, il n'y a pas unicité du minimiseur. Il s'agit d'un phénomène qui peut être gênant dans certains cas (identifiabilité).

Point de vue statistique (Machine Learning)

Pour construire un prédicteur $\hat{f}$ dans une classe de fonction $\mathcal{F}$, le principe consiste généralement à minimiser le **risque empirique**, *i.e.* la moyenne des pertes sur les observations,

$$\hat{f} \in \operatorname*{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{k=1}^{n} \ell(y_k, f(x_k)).$$

Dans un contexte général, il n'y a pas unicité du minimiseur. Il s'agit d'un phénomène qui peut être gênant dans certains cas (identifiabilité).

Si $\mathcal{F}$ est l'ensemble des fonctions de $\mathcal{X}$ dans $\mathcal{Y}$, un tel problème d'optimisation est généralement insoluble. Il faut se restreindre à une sous-classe $S \subset \mathcal{F}$, appelé **modèle**, et définir

$$\hat{f}_S \in \operatorname*{argmin}_{f \in S} \frac{1}{n} \sum_{k=1}^{n} \ell(y_k, f(x_k)).$$

En supposant qu'ils existent des fonctions "idéales" $f^* \in \mathcal{F}$ et $f_S^* \in S$ pour prédire la sortie à partir des entrées, nous sommes alors conduits à la décomposition

$$R_n(\hat{f}_S) - R_n(f^*) = \underbrace{R_n(f_S^*) - R_n(f^*)}_{\text{Erreur d'approximation (biais)}} + \underbrace{R_n(\hat{f}_S) - R_n(f_S^*)}_{\text{Erreur d'estimation (variance)}}$$

où $R_n(f) = \frac{1}{n} \sum_{k=1}^{n} \ell(y_k, f(x_k))$.

En supposant qu'ils existent des fonctions "idéales" $f^* \in \mathcal{F}$ et $f_S^* \in S$ pour prédire la sortie à partir des entrées, nous sommes alors conduits à la décomposition

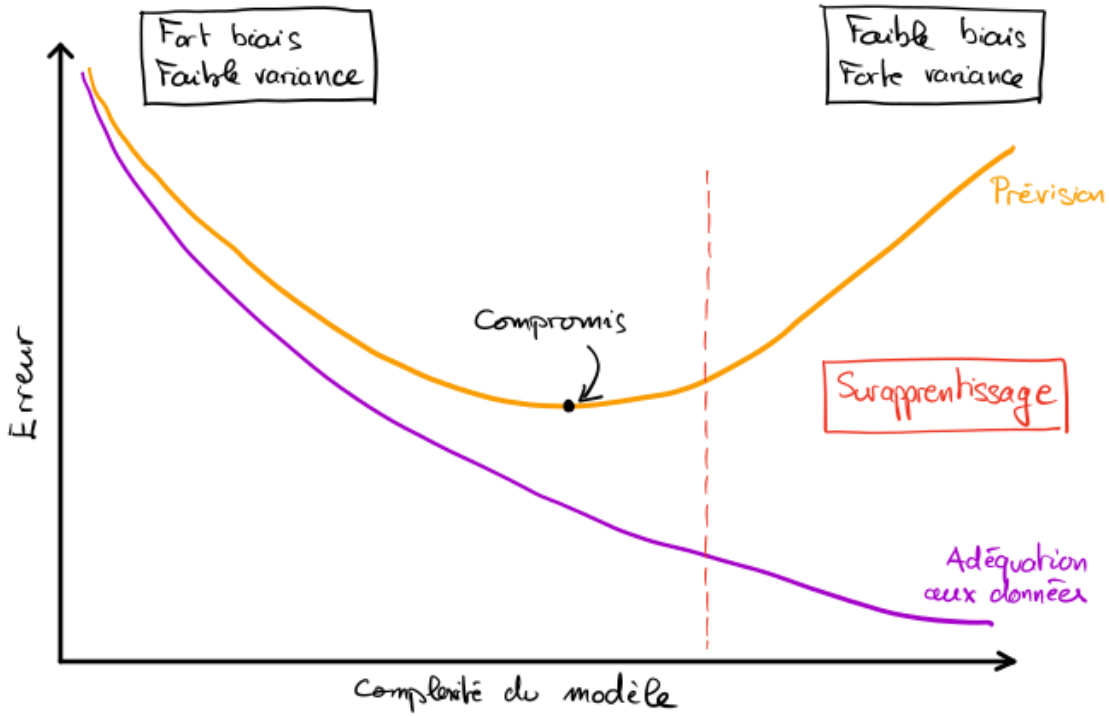
$$R_n(\hat{f}_S) - R_n(f^*) = \underbrace{R_n(f_S^*) - R_n(f^*)}_{\text{Erreur d'approximation (biais)}} + \underbrace{R_n(\hat{f}_S) - R_n(f_S^*)}_{\text{Erreur d'estimation (variance)}}$$

où $R_n(f) = \frac{1}{n}\sum_{k=1}^{n} \ell(y_k, f(x_k))$.

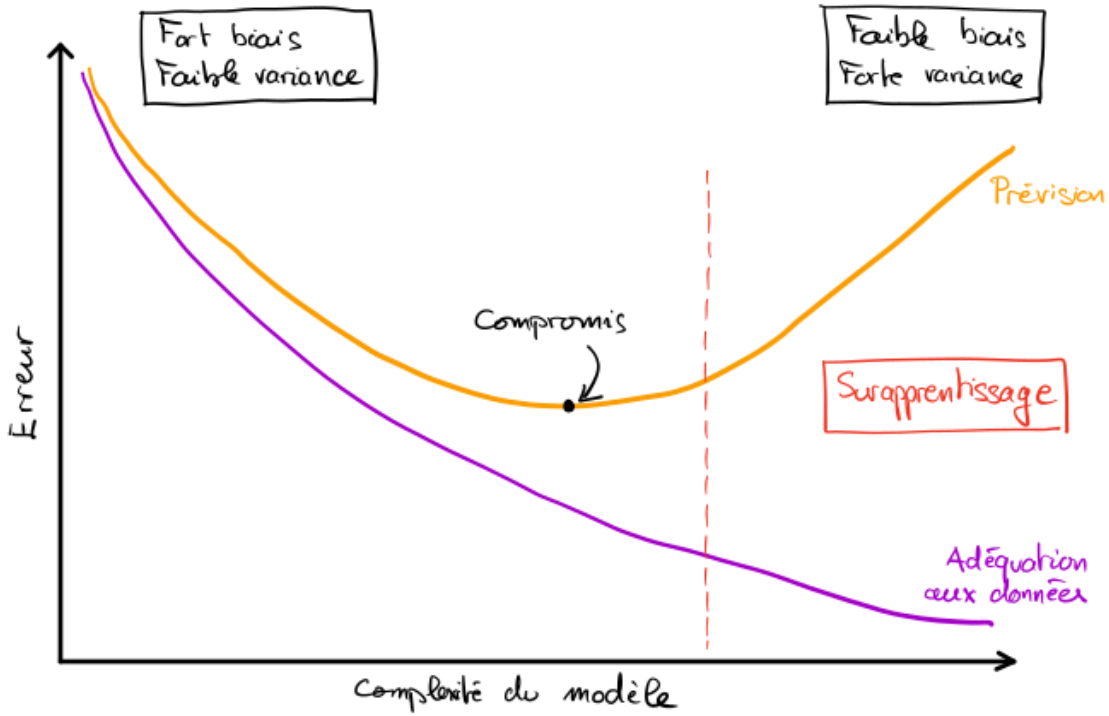
Un modèle riche/complexe aura une erreur d'approximation faible mais son erreur d'estimation sera difficile à contrôler : **faible biais, grande variance**.

Un modèle trop simple aura une erreur d'estimation faible (ou facile à contrôler) mais une erreur d'approximation potentiellement importante : **fort biais, faible variance**.

La recherche d'un **compromis** entre ces situations est le cœur des méthodes d'apprentissage supervisé, nous parlons d'**équilibre biais-variance**.



La recherche d'un **compromis** entre ces situations est le cœur des méthodes d'apprentissage supervisé, nous parlons d'**équilibre biais-variance**.



Lorsque la complexité du modèle est si grande que le critère devient presque nul, le prédicteur donne une adéquation quasiment parfaite sur les données (*i.e.* biais faible). Cette situation, appelée **surapprentissage**, n'est généralement pas souhaitable car elle néglige l'erreur d'estimation qui quantifie la performance du prédicteur à prédire à partir de nouvelles données.

Validation croisée

Comment se prémunir du problème de surapprentissage en pratique ?

Validation croisée

Comment se prémunir du problème de surapprentissage en pratique ?

En effet, lorsque nous disposons d'un jeu de données, nous souhaitons l'utiliser en intégralité pour notre procédure d'apprentissage et nous ne disposons souvent pas de "nouvelles" données.

Validation croisée

Comment se prémunir du problème de surapprentissage en pratique ?

En effet, lorsque nous disposons d'un jeu de données, nous souhaitons l'utiliser en intégralité pour notre procédure d'apprentissage et nous ne disposons souvent pas de "nouvelles" données.

La solution consiste à mettre de côté une partie du jeu de données (**échantillon de validation**) pour mesurer les performances du prédicteur construit à partir du reste des données (**échantillon d'entraînement**). Nous parlons alors de **validation croisée** et plusieurs méthodes (que nous ne présenterons pas) existent pour faire cela intelligemment.

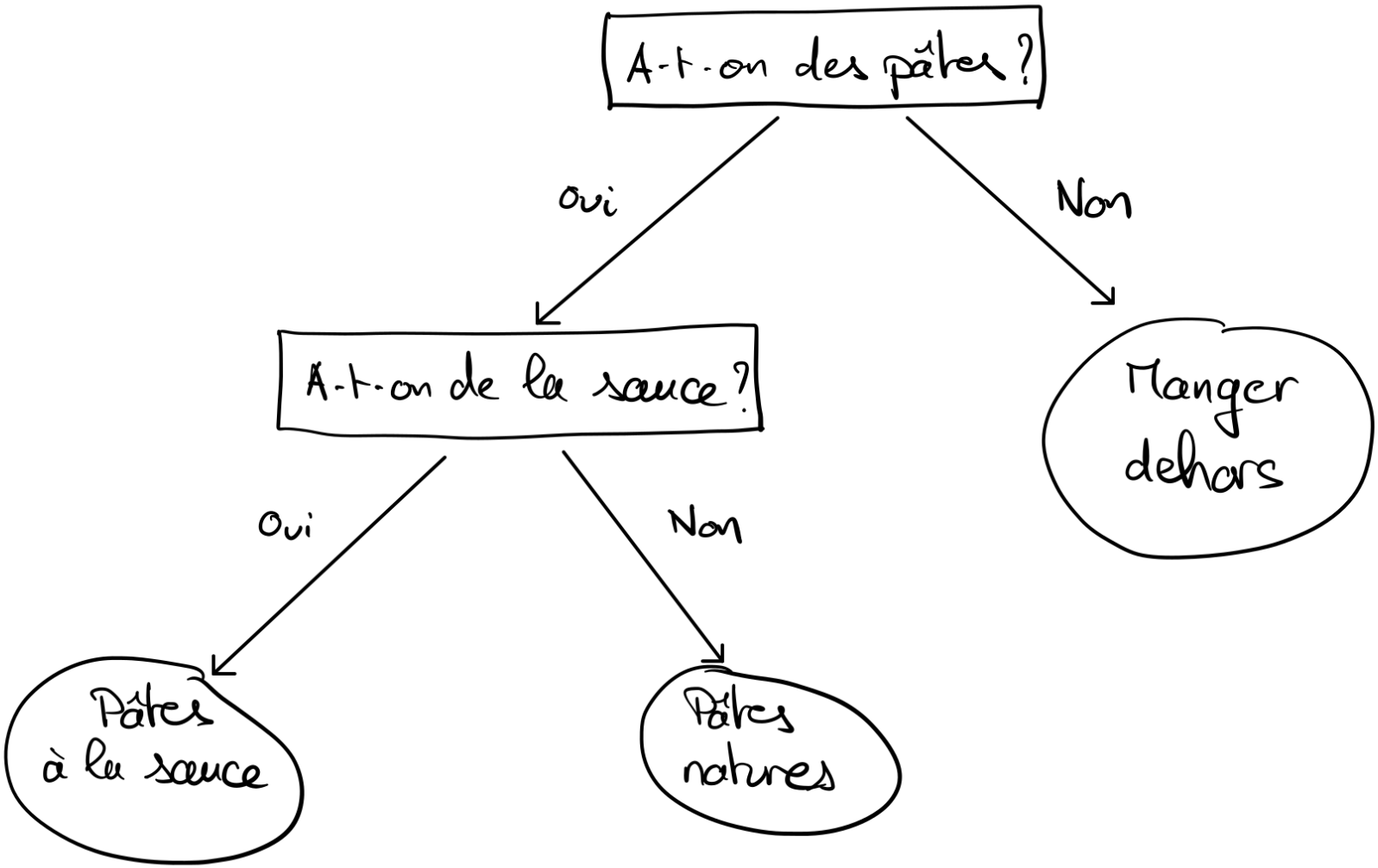
Arbres binaires de décision et agrégation d'arbres

Introduction

Un **arbre (binaire de décision)** est une méthode de partitionnement récursif. Le formalisme des arbres date des années 1980 et il est principalement dû aux travaux de Léo Breiman.

Principe : à chaque étape (nous parlons de **nœud**), une règle de décision binaire est appliquée sur les données d'entrée qui se séparent en deux sous-ensembles, celui pour lequel la règle est satisfaite et celui pour lequel ce n'est pas le cas.

Lorsqu'il n'est plus nécessaire de diviser un groupe de données pour prédire la sortie, il s'agit d'un nœud terminal, appelée **feuille**, qui donne la sortie prédite.


A-t-on des pâtes ?
Oui
Non
A-t-on de la sauce ?
Manger dehors
Oui
Non
Pâtes à la sauce
Pâtes natures

Avantages de la méthode :

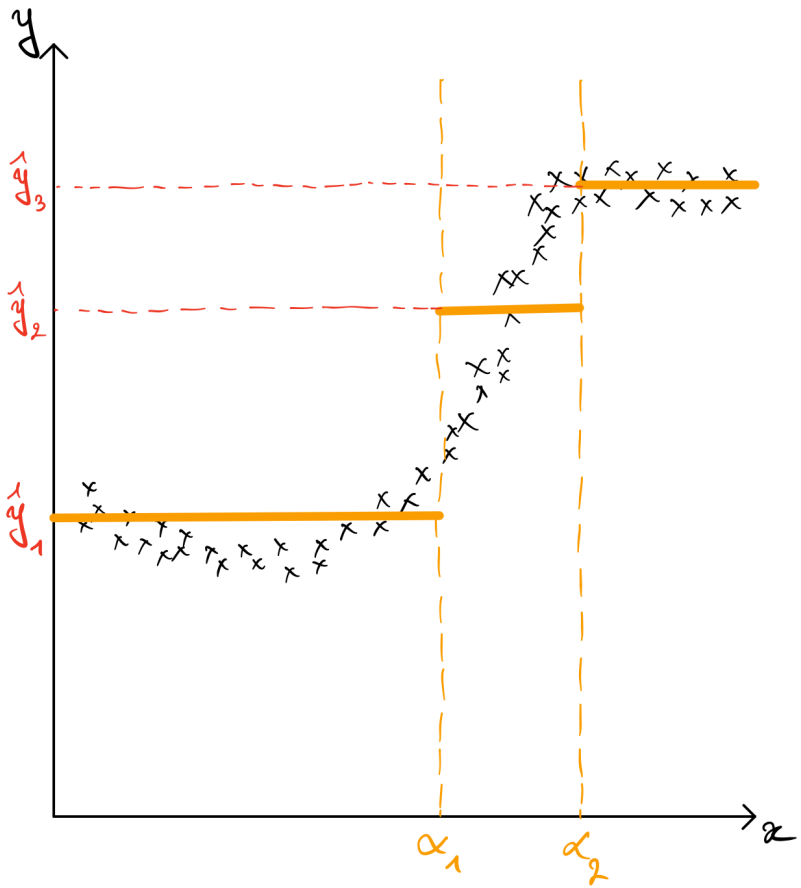
- la solution présentée sous forme graphique est simple à interpréter, même par un néophyte en statistique, et cela donne lieu à une aide à la décision efficace,
- la simplicité de cette procédure permet de manipuler de la même façon des règles construites à partir de variables quantitatives (*e.g.* "Age $\geqslant 18$", "Temp $\leqslant 20$, ...) ou qualitatives (*e.g.* "A-t-on des pâtes ?", "Les yeux sont-ils bleus ?", ...),
- la réponse donnée par une feuille peut être quantitative (arbre de régression) ou qualitative (arbre de décision).

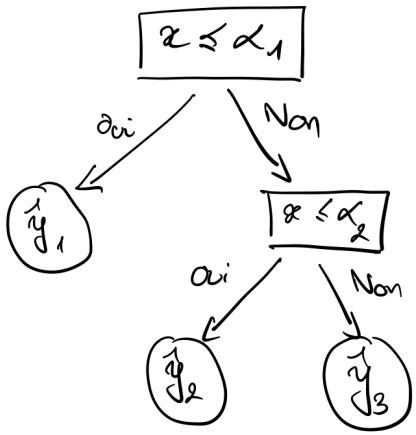
CART = Classification And Regression Tree

Dans le cas simple d'une régression unidimensionnelle,

$$(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2,$$

un arbre n'est rien d'autre qu'une fonction constante par morceaux.

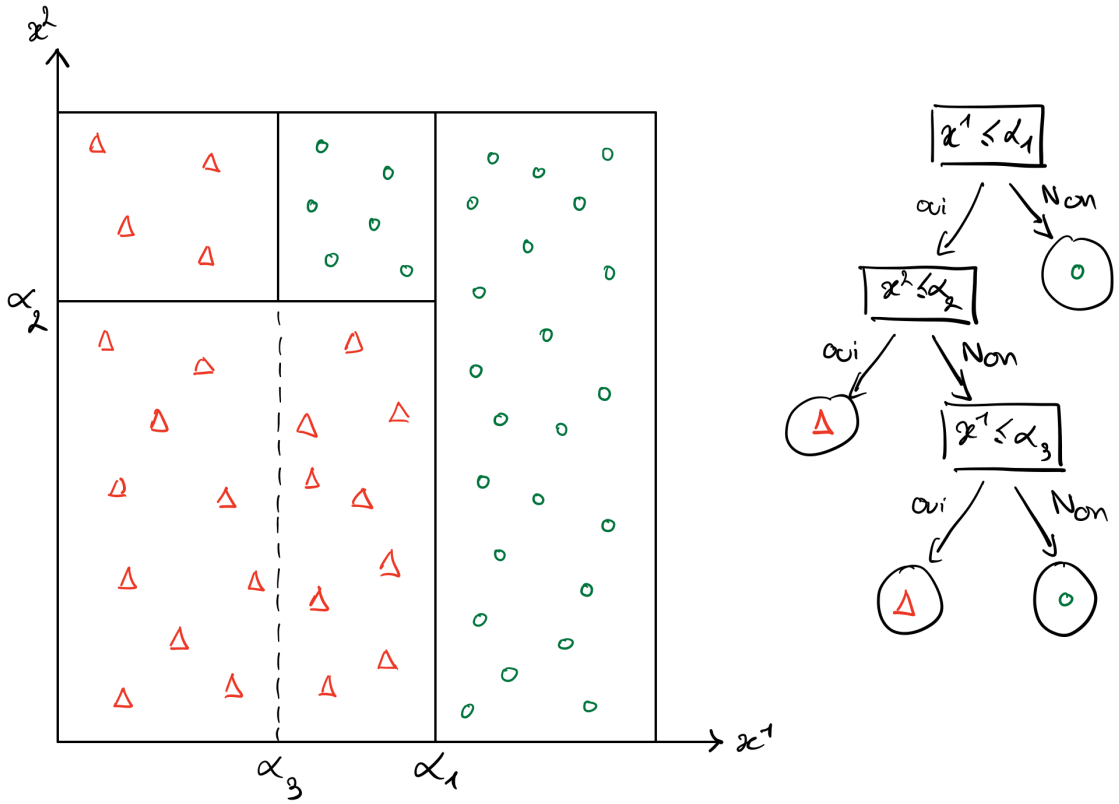




Pour un problème de classification binaire à partir d'entrées quantitatives bidimensionnelles,

$$\left((x_1^1, x_1^2), y_1\right), \ldots, \left((x_n^1, x_n^2), y_n\right) \in \mathbb{R}^2 \times \{-1, +1\},$$

un arbre constitue une **partition dyadique** de l'espace d'entrée.



Ces deux exemples illustrent la simplicité des arbres mais aussi leurs inconvénients :

- **instabilité :** la structure d'un arbre est très sensible à des fluctuations du jeu de données,
- **irrégularité :** en particulier pour la régression, la solution donnée par un arbre est irrégulière par construction même si le phénomène à modéliser est régulier.

Ces deux faiblesses conduisent à considérer des méthodes d'**agrégation** qui feront l'objet de la seconde partie de ce chapitre.

Arbre maximal

Nous nous plaçons dans le cadre de l'apprentissage supervisé :

$$(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}.$$

Pour construire un arbre, il faut déterminer une **séquence de nœuds** :

- un nœud est défini par le choix conjoint d'une **entrée** et d'une **division** qui induit une partition en **deux classes**,
- une **division** est définie par un **seuil** pour une variable d'entrée quantitative ou par un partage en deux **groupes de modalités distinctes** pour une variable d'entrée qualitative,
- la **racine** (*i.e.* nœud initial) correspond à l'ensemble du jeu de données puis la procédure est itérée sur chacun des sous-ensembles obtenus.

Pour cet algorithme, nous avons donc besoin de :

- un **critère** pour choisir la meilleure division parmi tous les choix possibles,
- une **règle d'arrêt** pour décider quand un nœud est une feuille,
- une **méthode de prédiction** pour donner une valeur à une feuille.

Critère de division

Il repose sur une **mesure d'hétérogénéité** de la parition en deux classes induite par une division. L'objectif est de partager l'échantillon en deux groupes les plus homogènes au sens de la variable de sortie.

Critère de division

Il repose sur une **mesure d'hétérogénéité** de la parition en deux classes induite par une division. L'objectif est de partager l'échantillon en deux groupes les plus homogènes au sens de la variable de sortie.

Pour un nœud N, nous notons H_N la mesure d'hétérogénéité du nœud. Nous verrons des exemples dans la suite mais il s'agit d'une quantité $H_N \geqslant 0$ d'autant plus proche de 0 que N est homogène.

Critère de division

Il repose sur une **mesure d'hétérogénéité** de la parition en deux classes induite par une division. L'objectif est de partager l'échantillon en deux groupes les plus homogènes au sens de la variable de sortie.

Pour un nœud N, nous notons H_N la mesure d'hétérogénéité du nœud. Nous verrons des exemples dans la suite mais il s'agit d'une quantité $H_N \geqslant 0$ d'autant plus proche de 0 que N est homogène.

Une division d du nœud N induit une partition en deux sous-groupes $N_1(d)$ et $N_2(d)$ pour lesquels nous pouvons aussi définir les mesures d'hétérogénéité $H_{N_1(d)}$ et $H_{N_2(d)}$.

Le critère de division revient alors à choisir la division d_N^* qui maximise la baisse d'hétérogénéité parmi toutes les divisions possibles,

$$d_N^* = \underset{d \text{ : division de } N}{\operatorname{argmax}} \left\{ H_N - \left(H_{N_1(d)} + H_{N_2(d)} \right) \right\}.$$

De façon équivalente, il s'agit aussi de la division qui induit la partition en classes les plus homogènes,

$$d_N^* = \underset{d \text{ : division de } N}{\operatorname{argmin}} \left\{ H_{N_1(d)} + H_{N_2(d)} \right\}.$$

Règle d'arrêt

Un nœud n'est pas divisé et devient une feuille si il est totalement homogène (*i.e.* il a une mesure d'hétérogénéité nulle) ou si aucune division ne peut faire baisser son hétérogénéité.

Pour des questions algorithmiques, cette règle est souvent relâchée :

- soit en fixant un seuil pour la mesure d'hétérogénéité en dessous duquel nous ne cherchons plus de division,
- soit en fixant une taille minimale telle que si le nombre d'observations est inférieur, le nœud n'est plus divisé.

Méthode de prédiction

La valeur prédite dans une feuille est généralement une statistique simple :

- en régression, ce peut être la moyenne des valeurs de sortie dans l'échantillon,
- en classification, ce peut être la modalité la plus représentée dans les sorties.

Toute autre méthode de prédiction est également acceptable mais, dans le cadre de ce cours, nous ne présenterons pas d'exemples plus avancés.

Exemples classiques de mesures d'hétérogénéité :

- en régression, il est possible d'utiliser la **variance**,

$$H_n = \frac{1}{|N|} \sum_{k \in N} \left(y_k - \frac{1}{|N|} \sum_{k' \in N} y_{k'} \right)^2$$

où $|N|$ désigne la taille de N,

- en classification, l'**entropie** est une mesure utilisée,

$$H_N = -2|N| \sum_m p_m(N) \log \left(p_m(N) \right),$$

où $p_m(N)$ désigne la proportion d'observations de la modalité m dans le nœud N (par convention, $0 \times \log(0) = 0$),

- en classification, l'**indice de Gini** est aussi souvent utilisé,

$$H_n = \sum_m p_m(N) \left(1 - p_m(N) \right).$$

Élagage (pruning)

Par construction, un arbre maximal A_{max} sera excessivement développé car la méthode oblige une forte adéquation aux données. Il s'agit du phénomène de **surapprentissage**.

Élagage (pruning)

Par construction, un arbre maximal A_{max} sera excessivement développé car la méthode oblige une forte adéquation aux données. Il s'agit du phénomène de **surapprentissage**.

Pour limiter l'instabilité due à l'erreur d'estimation (terme de variance), il convient de considérer des modèles d'arbres plus parcimonieux.

Principe de l'élagage : couper certaines branches d'un arbre maximal pour obtenir un sous-arbre moins proche des données d'entraînement afin de chercher un équilibre biais-variance.

Problème

Le nombre de sous-arbres d'un arbre binaire à K feuilles est **exponentiel** en K. Il n'est pas envisageable de faire une exploration exhaustive de tous les sous-arbres de A_{max}.

Problème

Le nombre de sous-arbres d'un arbre binaire à K feuilles est **exponentiel** en K. Il n'est pas envisageable de faire une exploration exhaustive de tous les sous-arbres de A_{max}.

Idée de Breiman : limiter la recherche à une collection de sous-arbres emboîtés de A_{max}. La taille d'une telle collection est linéaire par rapport au nombre K de feuilles. La solution obtenue est **sous-optimale** mais la méthode est fiable et efficace.

Problème

Le nombre de sous-arbres d'un arbre binaire à K feuilles est **exponentiel** en K. Il n'est pas envisageable de faire une exploration exhaustive de tous les sous-arbres de A_{max}.

Idée de Breiman : limiter la recherche à une collection de sous-arbres emboîtés de A_{max}. La taille d'une telle collection est linéaire par rapport au nombre K de feuilles. La solution obtenue est **sous-optimale** mais la méthode est fiable et efficace.

Pour construire cette collection, nous définissons la **qualité d'ajustement** d'un arbre A somme la somme des mesures d'hétérogénéité de ses feuilles,

$$D(A) = \sum_{k=1}^{K_A} H_{N_k}$$

où $N_1, \ldots, N_{K_A}$ sont les feuilles de A.

Soit $\lambda \geqslant 0$, nous cherchons à minimiser le **critère de complexité** d'un sous-arbre $A \subset A_{max}$,

$$\gamma(A) = D(A) + \lambda K_A.$$

Le terme $D(A)$ quantifie la capacité de A à être en **adéquation avec les données** tandis que la **pénalité** λK_A correspond à la complexité de l'arbre.

Soit $\lambda \geqslant 0$, nous cherchons à minimiser le **critère de complexité** d'un sous-arbre $A \subset A_{max}$,

$$\gamma(A) = D(A) + \lambda K_A.$$

Le terme $D(A)$ quantifie la capacité de A à être en **adéquation avec les données** tandis que la **pénalité** λK_A correspond à la complexité de l'arbre.

Pour $\lambda = 0$, le critère est minimal pour $A = A_{max}$ qui a K_{max} feuilles. En faisant augmenter λ, la pénalité va passer au-dessus du **plus petit gain d'hétérogénéité** parmi toutes les feuilles. En coupant les branches associées, nous élaguons deux feuilles en une et nous obtenons un sous-arbre $A_{K_{max}-1}$ à $K_{max} - 1$ feuilles. Ainsi de suite, nous construisons la collection de sous-arbres emboîtés :

$$A_1 \subset A_2 \subset \cdots \subset A_{K_{max}-1} \subset A_{K_{max}} = A_{max}$$

où A_1 est l'arbre trivial réduit à sa racine, *i.e.* le jeu de données initial en entier.

$$\gamma(A) = D(A) + \lambda K_A$$

$$A_1 \subset A_2 \subset \cdots \subset A_{K_{max}-1} \subset A_{K_{max}} = A_{max}$$

La recherche du compromis biais-variance peut se faire dans cette collection de taille raisonnable. Cela revient uniquement à faire un "bon" choix pour la valeur du paramètre λ.

Il n'existe pas de réponse universelle pour choisir cette valeur. En pratique, le paramètre λ est calibré par **validation croisée**.

Agrégation d'arbres

Pour corriger les inconvénients des arbres évoqués en introduction, des méthodes basées sur un ensemble d'arbres (nous parlons de **forêt**) ont été proposées dès les années 2000. Il s'agit de **méthodes d'agrégation d'arbres**.

Agrégation d'arbres

Pour corriger les inconvénients des arbres évoqués en introduction, des méthodes basées sur un ensemble d'arbres (nous parlons de **forêt**) ont été proposées dès les années 2000. Il s'agit de **méthodes d'agrégation d'arbres**.

Il existe deux grandes familles de telles méthodes :

- le **bagging** pour des algorithmes à forte variance,
- le **boosting** pour des algorithmes biaisés de faible variance.

Ces méthodes ne sont pas limitées aux arbres de décisions et peuvent être utilisées avec des prédicteurs plus généraux. Nous nous limiterons ici aux arbres mais les principes sous-jacents restent les mêmes.

Agrégation d'arbres – Bagging

L'idée du bagging est d'imiter un phénomène bien connu de réduction de la variance par moyennisation de résultats sans biais et indépendants.

Agrégation d'arbres – Bagging

L'idée du bagging est d'imiter un phénomène bien connu de réduction de la variance par moyennisation de résultats sans biais et indépendants.

En effet, si nous ne disposions pas que d'un unique jeu de données

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$$

mais de B **répétitions indépendantes** $\mathcal{D}_1, \ldots, \mathcal{D}_B$ de la même expérience, nous pourrions construire des prédicteurs $\hat{f}_1, \ldots, \hat{f}_B$ (par exemple, des arbres) qui seraient également indépendants. Les prédictions pour une nouvelle entrée $x_0 \in \mathcal{X}$ données par $\hat{f}_1(x_0), \ldots, \hat{f}_B(x_0)$ pourraient alors être **agrégées**,

$$\frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x_0).$$

Prédiction agrégée

$$\frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x_0).$$

Les prédictions $\hat{f}_1(x_0), \ldots, \hat{f}_B(x_0)$ ont le même biais (*i.e.* erreur d'approximation) et la même variance σ^2 (*i.e.* erreur d'estimation). Par indépendance, la variance de la prédiction agrégée vaut

$$\frac{\sigma^2}{B}.$$

Ainsi, **agréger réduit la variance**.

Prédiction agrégée

$$\frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x_0).$$

Les prédictions $\hat{f}_1(x_0), \ldots, \hat{f}_B(x_0)$ ont le même biais (*i.e.* erreur d'approximation) et la même variance σ^2 (*i.e.* erreur d'estimation). Par indépendance, la variance de la prédiction agrégée vaut

$$\frac{\sigma^2}{B}.$$

Ainsi, **agréger réduit la variance**.

Problème

En pratique, nous ne disposons pas de répétitions indépendantes du jeu de données.

Problème

En pratique, nous ne disposons pas de répétitions indépendantes du jeu de données.

Pour contourner cette difficulté, nous considérons plutôt B **réplications bootstrap** d'un sous-échantillon de taille $a_n < n$, *i.e.* des tirages aléatoires **sans remise** de a_n éléments parmi les n observations de l'échantillon initial.

Les prédicteurs $\hat{f}_1, \ldots, \hat{f}_B$ obtenus à partir de ces réplications ne sont plus indépendants mais cet **ajout d'aléas** conduit encore à une réduction de la variance pour la prédiction agrégée.

Bagging

- Jeu de données $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
- Taille du bootstrap $a_n < n$

Algorithme :
  Pour b allant de 1 à B :
    1) Tirer un échantillon bootstrap $\mathcal{D}_b$ de taille a_n
    2) Calculer un prédicteur $\hat{f}_b$ à partir de $\mathcal{D}_b$

Sortie : prédicteur agrégé

$$\frac{1}{B} \sum_{b=1}^{B} \hat{f}_b$$

L'étape de bootstrap permet d'évaluer l'erreur sur les observations n'ayant pas été tirées dans l'échantillon comme pour la validation croisée. La moyenne de ces erreurs est facilement calculable au sein de l'algorithme de bagging et elle est souvent retournée par les logiciels statistiques sous le nom de **erreur out-of-bag**.

L'étape de bootstrap permet d'évaluer l'erreur sur les observations n'ayant pas été tirées dans l'échantillon comme pour la validation croisée. La moyenne de ces erreurs est facilement calculable au sein de l'algorithme de bagging et elle est souvent retournée par les logiciels statistiques sous le nom de **erreur out-of-bag**.

En pratique, nous utilisons souvent de petits arbres comme prédicteurs car leur faible taille induit une forte variance. L'agrégation par la moyenne réduit la variance et **lisse la prédiction** (*i.e.* une fonction plus régulière dans le cas de la régression).

Bien que simple, le bagging pose quelques difficultés :

- assurer une erreur out-of-bag faible peut demander un nombre B d'itérations important (*i.e.* du temps de calcul),
- il est nécessaire de stocker tous les prédicteurs pour calculer la prédiction agrégée,
- l'agrégation est beaucoup plus difficile à interpréter que chaque prédicteur seul (particulièrement pour des arbres).

Bien que simple, le bagging pose quelques difficultés :

- assurer une erreur out-of-bag faible peut demander un nombre B d'itérations important (*i.e.* du temps de calcul),
- il est nécessaire de stocker tous les prédicteurs pour calculer la prédiction agrégée,
- l'agrégation est beaucoup plus difficile à interpréter que chaque prédicteur seul (particulièrement pour des arbres).

Dans le cas particulier des arbres, Breiman a proposé au début des années 2000 une amélioration du bagging en ajoutant un peu plus d'indépendance dans la procédure pour favoriser le phénomène de réduction de variance. Il s'agit des **forêts aléatoires** (ou **random forest**).

Forêt aléatoire

- Jeu de données $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
- Taille du bootstrap $a_n < n$
- Nombre de divisions $m > 0$

Algorithme :

Pour b allant de 1 à B :

1) Tirer un échantillon bootstrap $\mathcal{D}_b$ de taille a_n
2) Construire un prédicteur $\hat{f}_b$ à partir de $\mathcal{D}_b$ comme un arbre pour lequel la division optimale dans chaque noeud est recherchée dans un sous-ensemble aléatoire de taille m des divisions possibles

Sortie : prédicteur agrégé

$$\frac{1}{B} \sum_{b=1}^{B} \hat{f}_b$$

En pratique, il est possible de se limiter à de petits arbres (2 à 5 feuilles) et c'est l'agrégation qui assure les bonnes propriétés du prédicteur retourné.

Le choix de m est un sujet de débat encore actif pour lequel nous ne connaissons pas de "bonne" réponse. En pratique, si les entrées sont de dimension p, il est commun de prendre $m = \sqrt{p}$ pour de la classification et $m = p/3$ pour de la régression.

Agrégation d'arbres – Boosting

Le boosting concerne plutôt des prédicteurs $\hat{f}$ avec un fort biais mais une faible variance. Ainsi, après une première étape de prédiction, il reste beaucoup d'information dans les **résidus** $y_1 - \hat{f}(x_1), \ldots, y_n - \hat{f}(x_n)$. L'idée est donc d'estimer encore cette information et de l'**agréger** à l'estimation précédente pour l'améliorer et ainsi de suite par itérations successives. L'objectif est donc d'**améliorer un prédicteur faible**.

Agrégation d'arbres – Boosting

Le boosting concerne plutôt des prédicteurs $\hat{f}$ avec un fort biais mais une faible variance. Ainsi, après une première étape de prédiction, il reste beaucoup d'information dans les **résidus** $y_1 - \hat{f}(x_1), \ldots, y_n - \hat{f}(x_n)$. L'idée est donc d'estimer encore cette information et de l'**agréger** à l'estimation précédente pour l'améliorer et ainsi de suite par itérations successives. L'objectif est donc d'**améliorer un prédicteur faible**.

Contrairement au bagging, les prédicteurs construits au cours du boosting sont calculés de **façon itérative**. Le prédicteur $\hat{f}_m$ invoqué à l'étape m dépend du prédicteur $\hat{f}_{m-1}$ de l'étape $m-1$.

Il existe de très nombreuses variantes de cette approche qu'il serait vain d'énumérer. Il s'agit d'un sujet de recherche très actif et des propositions nouvelles voient régulièrement le jour.

Utilisée avec des arbres simples (*e.g.* arbres à 2 feuilles), ce type d'approche conduit à de meilleurs prédicteurs que ce qui pourrait être obtenu à partir d'arbres plus complexes. Nous nous limiterons ici à présenter deux variantes du boosting :

- **AdaBoost** pour le problème de classification binaire,
- **L2Boost** pour le problème de régression des moindres-carrés.

AdaBoost (Adaptive Boosting)

- Données : $(x_1, y_1), \dots (x_n, y_n) \in \mathcal{X} \times \{-1, +1\}$
- Poids initiaux : $w_1 = \cdots = w_n = 1/n$ (Notation : $W = w_1 + \cdots + w_n$)

Algorithme :

Pour m allant de 1 à M :

1) Construire un prédicteur $\hat{f}_m$ à partir des données pondérées par $w_1, \dots, w_n$
2) Calculer le taux d'erreur : $e_m = \dfrac{1}{W} \displaystyle\sum_{k=1}^{n} w_k \mathbf{1}_{y_k \neq \hat{f}_m(x_k)}$
3) Calculer la correction : $\alpha_m = \log(e_m^{-1} - 1)$
4) Mise à jour des poids : $\forall k,\ w_k = w_k \exp\left(\alpha_m \mathbf{1}_{y_k \neq \hat{f}_m(x_k)}\right)$

Sortie : retourner le signe de

$$\sum_{m=1}^{M} \alpha_m \hat{f}_m.$$

Le principe de AdaBoost est de **donner plus d'importance** aux observations mal classées à l'itération précédente en augmentant leurs poids. Autrement dit, la procédure se focalise sur les observations difficiles à classer pour **corriger le biais** du prédicteur précédent.

Le principe de AdaBoost est de **donner plus d'importance** aux observations mal classées à l'itération précédente en augmentant leurs poids. Autrement dit, la procédure se focalise sur les observations difficiles à classer pour **corriger le biais** du prédicteur précédent.

Plusieurs travaux ont conduit à proposer des versions plus générales de cet algorithme dont ceux de Friedman au début des années 2000. Son point de vue est celui d'une **descente de gradient** et cela permet d'adapter l'approche du boosting à d'autres problèmes comme celui de la régression.

L2Boost (Version simplifiée)

- Données : $(x_1, y_1), \dots (x_n, y_n) \in \mathcal{X} \times \mathbb{R}$
- Prédicteur initial : $\hat{f}_0$
- Pas de descente : $\lambda > 0$

Algorithme :

Pour m allant de 1 à M :

1) Calculer les résidus : $\forall k,\ r_k = y_k - \hat{f}_{m-1}(x_k)$

2) Construire un prédicteur g_m à partir de

$$(x_1, r_1), \dots, (x_n, r_n) \in \mathcal{X} \times \mathbb{R}$$

3) Mettre à jour le prédicteur :

$$\forall x \in \mathcal{X},\ \hat{f}_m(x) = \hat{f}_{m-1}(x) + \lambda g_m(x)$$

Sortie : retourner le prédicteur $\hat{f}_M$

Dans cette version simple de L2Boost, le fait d'expliquer de façon itérative les erreurs commises à l'étape précédente est explicite. Le paramètre $\lambda > 0$ est le **pas de la descente de gradient** sous-jacente. Pour obtenir de bonnes performances en pratique, il est important de bien calibrer λ mais cette discussion dépasse le cadre de ce cours.

Machines à vecteurs supports (SVM)

Introduction

Les SVM sont des méthodes d'apprentissage qui découlent des travaux de Vapnik dans les années 1990. À l'origine, il s'agissait d'algorithmes dédiés à la classification mais ils ont été peu à peu étendus aux problèmes d'apprentissage supervisé en général.

Introduction

Les SVM sont des méthodes d'apprentissage qui découlent des travaux de Vapnik dans les années 1990. À l'origine, il s'agissait d'algorithmes dédiés à la classification mais ils ont été peu à peu étendus aux problèmes d'apprentissage supervisé en général.

Le principe des SVM est la recherche d'un **hyperplan optimal** pour séparer au mieux les données tout en étant le plus éloigné de celles-ci. Cet éloignement a pour objectif d'assurer une **capacité de généralisation** la plus grande possible.

Introduction

Les SVM sont des méthodes d'apprentissage qui découlent des travaux de Vapnik dans les années 1990. À l'origine, il s'agissait d'algorithmes dédiés à la classification mais ils ont été peu à peu étendus aux problèmes d'apprentissage supervisé en général.

Le principe des SVM est la recherche d'un **hyperplan optimal** pour séparer au mieux les données tout en étant le plus éloigné de celles-ci. Cet éloignement a pour objectif d'assurer une **capacité de généralisation** la plus grande possible.

L'aspect linéaire ne doit pas être vu comme une limitation car cette approche peut être étendue à l'aide de **noyaux** dont nous parlerons. Cela confère aux SVM une grande flexibilité qui fait que ces méthodes sont largement utilisées dans de nombreux domaines.

Afin de garder cette présentation simple, nous nous limiterons au problème de classification binaire car ce chapitre est plus technique que les précédents,

$$(x_1, y_1), \dots (x_n, y_n) \in \mathcal{X} \times \{-1, +1\}.$$

Les SVM se généralisent à la classification générale et à la régression mais nous n'aborderons pas ces aspects dans le cadre de ce cours.

SVM linéaire

Nous commençons par le cas linéaire :

$$\mathcal{X} = \mathbb{R}^p.$$

Un problème de classification binaire est dit **linéairement séparable** si les sorties peuvent être discriminées simplement d'après le "côté" où se trouvent les entrées par rapport à un hyperplan. Autrement dit, il s'agit du cas où il existe $v \in \mathbb{R}^p$ et $a \in \mathbb{R}$ tels que

$$\forall k \in \{1, \dots, n\},\ y_k = \begin{cases} +1 & \text{si } v^\top x_k + a > 0, \\ -1 & \text{si } v^\top x_k + a < 0. \end{cases}$$

SVM linéaire

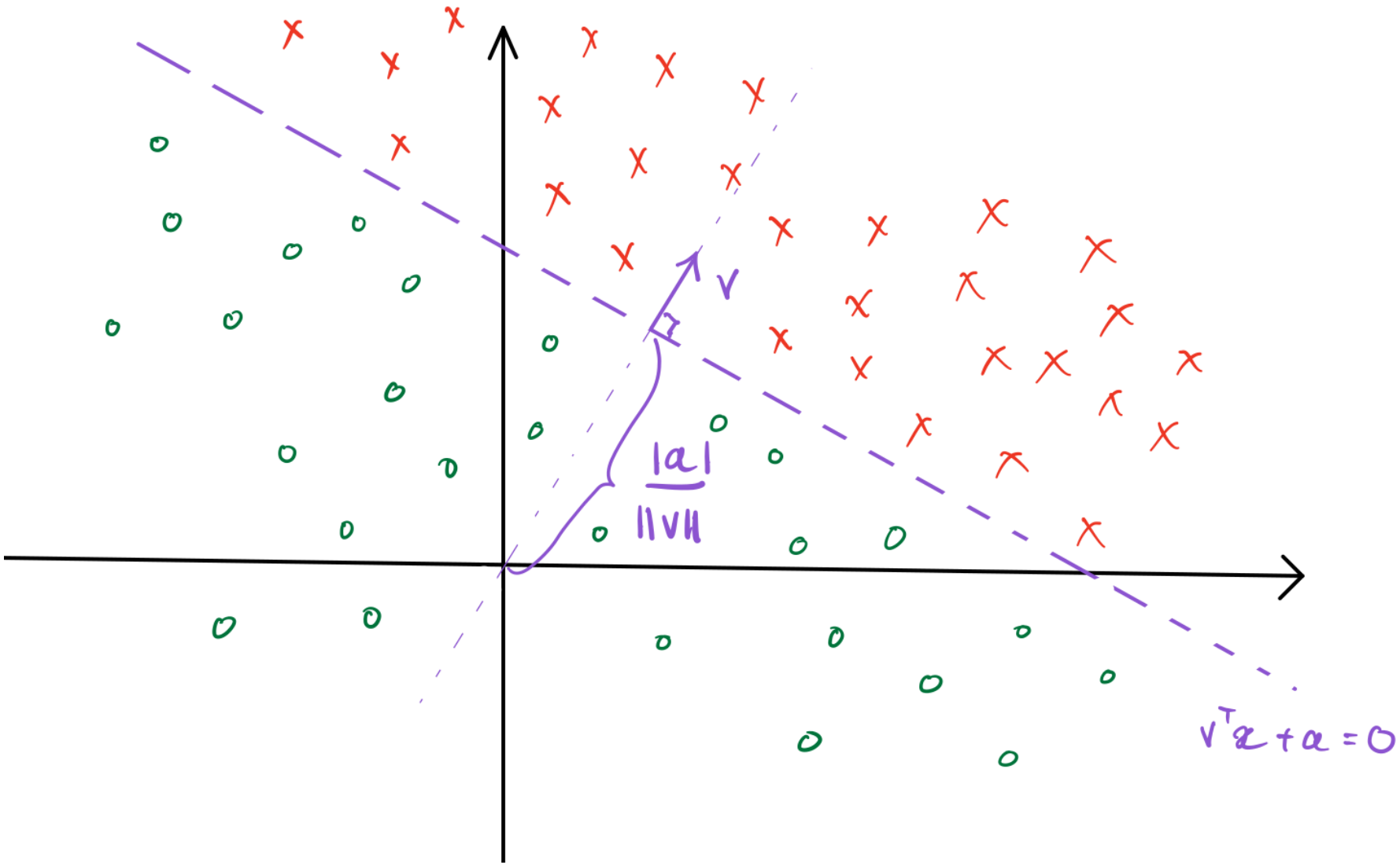
Nous commençons par le cas linéaire :

$$\mathcal{X} = \mathbb{R}^p.$$

Un problème de classification binaire est dit **linéairement séparable** si les sorties peuvent être discriminées simplement d'après le "côté" où se trouvent les entrées par rapport à un hyperplan. Autrement dit, il s'agit du cas où il existe $v \in \mathbb{R}^p$ et $a \in \mathbb{R}$ tels que

$$\forall k \in \{1, \dots, n\},\ y_k = \begin{cases} +1 & \text{si } v^\top x_k + a > 0, \\ -1 & \text{si } v^\top x_k + a < 0. \end{cases}$$

Il s'agit d'un cas très particulier (et très simple) mais qui présente un **grand intérêt pédagogique**.

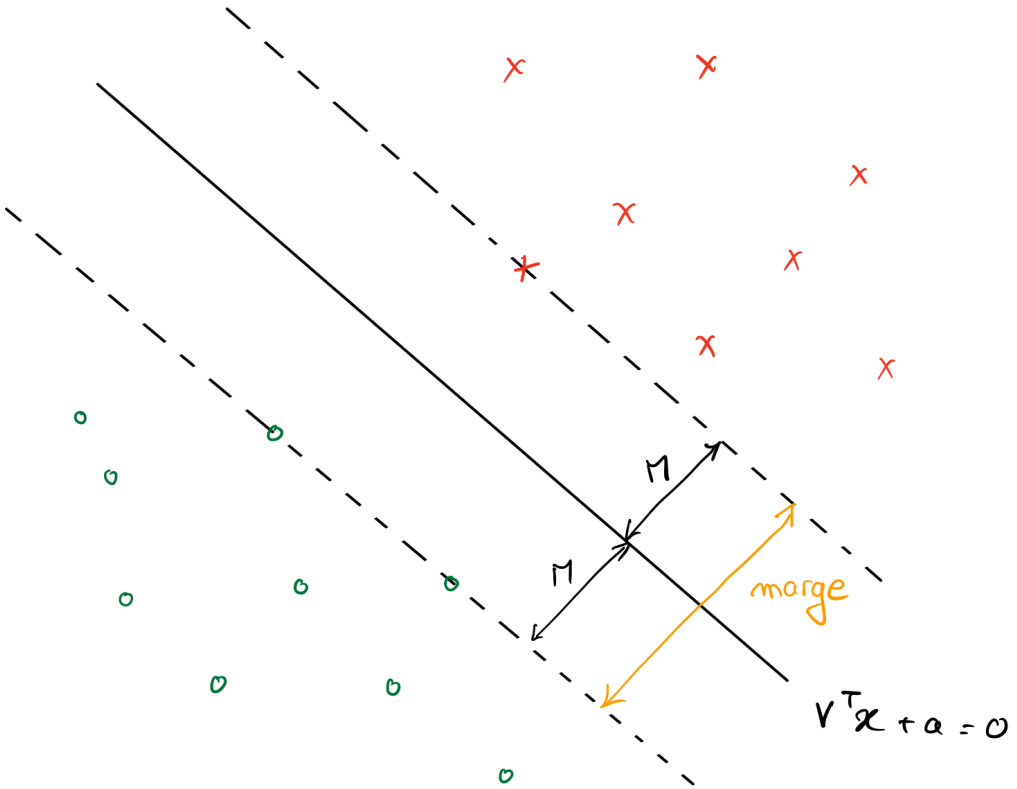

v
|a|
||v||
v^T x + a = 0

Sauf cas particulier, il existe une infinité de tels hyperplans séparateurs dans le cas linéairement séparable.

Sauf cas particulier, il existe une infinité de tels hyperplans séparateurs dans le cas linéairement séparable.

Idée de Vapnik

Choisir l'hyperplan de **marge** maximale.

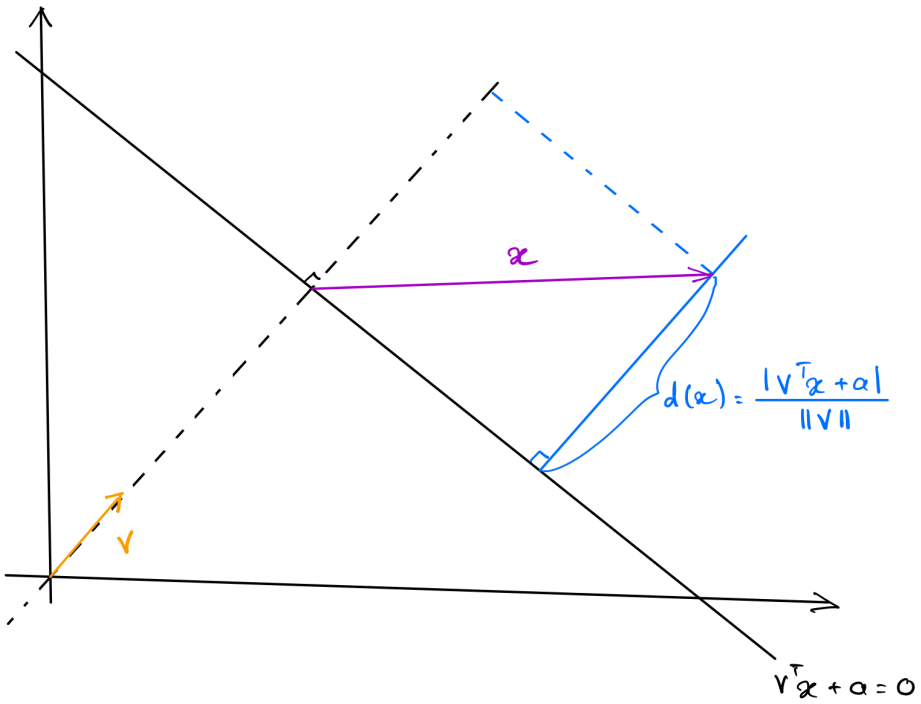


Distance à une droite

$$d(x) = \frac{|v^\top x + a|}{\|v\|}$$

Puisque les y_k sont dans $\{-1, +1\}$, la marge d'un hyperplan séparateur donné par $(v, a) \in \mathbb{R}^p \times \mathbb{R}$ vaut

$$\min_{1 \leqslant k \leqslant n} \frac{y_k(v^\top x_k + a)}{\|v\|}$$



Nous devons donc résoudre un problème d'optimisation sous contrainte : trouver $(v, a) \in \mathbb{R}^p \times \mathbb{R}$ pour maximiser la marge M sous la contrainte

$$\min_{1 \leqslant k \leqslant n} y_k(v^\top x_k + a) \geqslant M\|v\|$$

Nous devons donc résoudre un problème d'optimisation sous contrainte : trouver $(v, a) \in \mathbb{R}^p \times \mathbb{R}$ pour maximiser la marge M sous la contrainte

$$\min_{1 \leqslant k \leqslant n} y_k(v^\top x_k + a) \geqslant M\|v\|$$

Pour assurer l'unicité de la solution, il faut fixer la valeur de $\|v\|$ sinon tout vecteur colinéaire à v sera aussi solution. En prenant $\|v\| = 1/M$, le problème se reformule de façon quadratique,

$$(v, b) \in \mathbb{R}^p \times \mathbb{R} \text{ qui minimise } \frac{1}{2}\|v\|^2 \text{ tel que } \min_{1 \leqslant k \leqslant n} y_k(v^\top x_k + a) \geqslant 1.$$

Ce problème se résout avec le Lagrangien

$$L_p(v, a, \lambda_1, \ldots, \lambda_n) = \frac{1}{2}\|v\|^2 - \sum_{k=1}^{n} \lambda_k \left(y_k(v^\top x_k + a) - 1 \right)$$

dont les dérivées partielles s'annulent si

$$v = \sum_{k=1}^{n} \lambda_k x_k y_k \quad \text{et} \quad \sum_{k=1}^{n} \lambda_k y_k = 0.$$

Ce problème se résout avec le Lagrangien

$$L_p(v, a, \lambda_1, \ldots, \lambda_n) = \frac{1}{2}\|v\|^2 - \sum_{k=1}^{n} \lambda_k \left(y_k(v^\top x_k + a) - 1 \right)$$

dont les dérivées partielles s'annulent si

$$v = \sum_{k=1}^{n} \lambda_k x_k y_k \quad \text{et} \quad \sum_{k=1}^{n} \lambda_k y_k = 0.$$

En substituant ces deux équations dans L_p, nous obtenons le **problème dual** qui consiste à maximiser

$$(\lambda_1, \ldots, \lambda_n) \longmapsto L(\lambda_1, \ldots, \lambda_n) = \sum_{k=1}^{n} \lambda_k - \frac{1}{2} \sum_{k=1}^{n} \sum_{\ell=1}^{n} \lambda_k \lambda_\ell y_k y_\ell x_k^\top x_\ell$$

sous les contraintes $\lambda_1, \ldots, \lambda_n \geqslant 0$ et $\sum_{k=1}^{n} \lambda_k y_k = 0$.

Le problème dual est simple à résoudre avec n'importe quel logiciel scientifique. Si $(\lambda_1^*, \ldots, \lambda_n^*)$ est solution, alors

$$v^* = \sum_{k=1}^{n} \lambda_k^* x_k y_k.$$

Le problème dual est simple à résoudre avec n'importe quel logiciel scientifique. Si $(\lambda_1^*, \ldots, \lambda_n^*)$ est solution, alors

$$v^* = \sum_{k=1}^{n} \lambda_k^* x_k y_k.$$

Les conditions de Karush, Kuhn et Tucker (KKT) assurent que

$$\lambda_1^*, \ldots, \lambda_n^* \geqslant 0$$

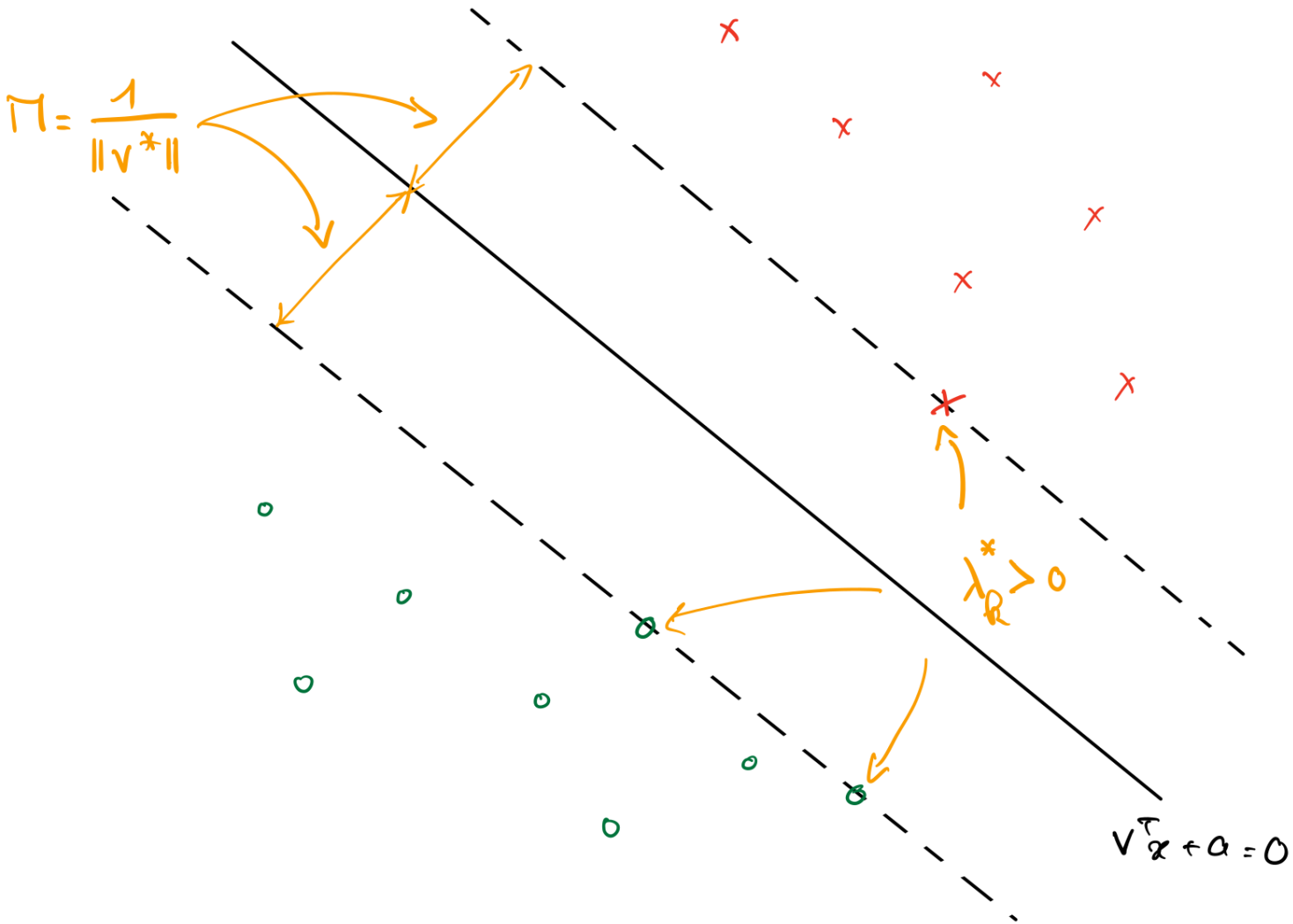
et nous avons

$$\forall k \in \{1, \ldots, n\}, \ \lambda_k^* \Big(y_k ({v^*}^\top x_k + a) \Big) = 0.$$

Au moins un des λ_k^* est non-nul et nous obtenons a^* en résolvant

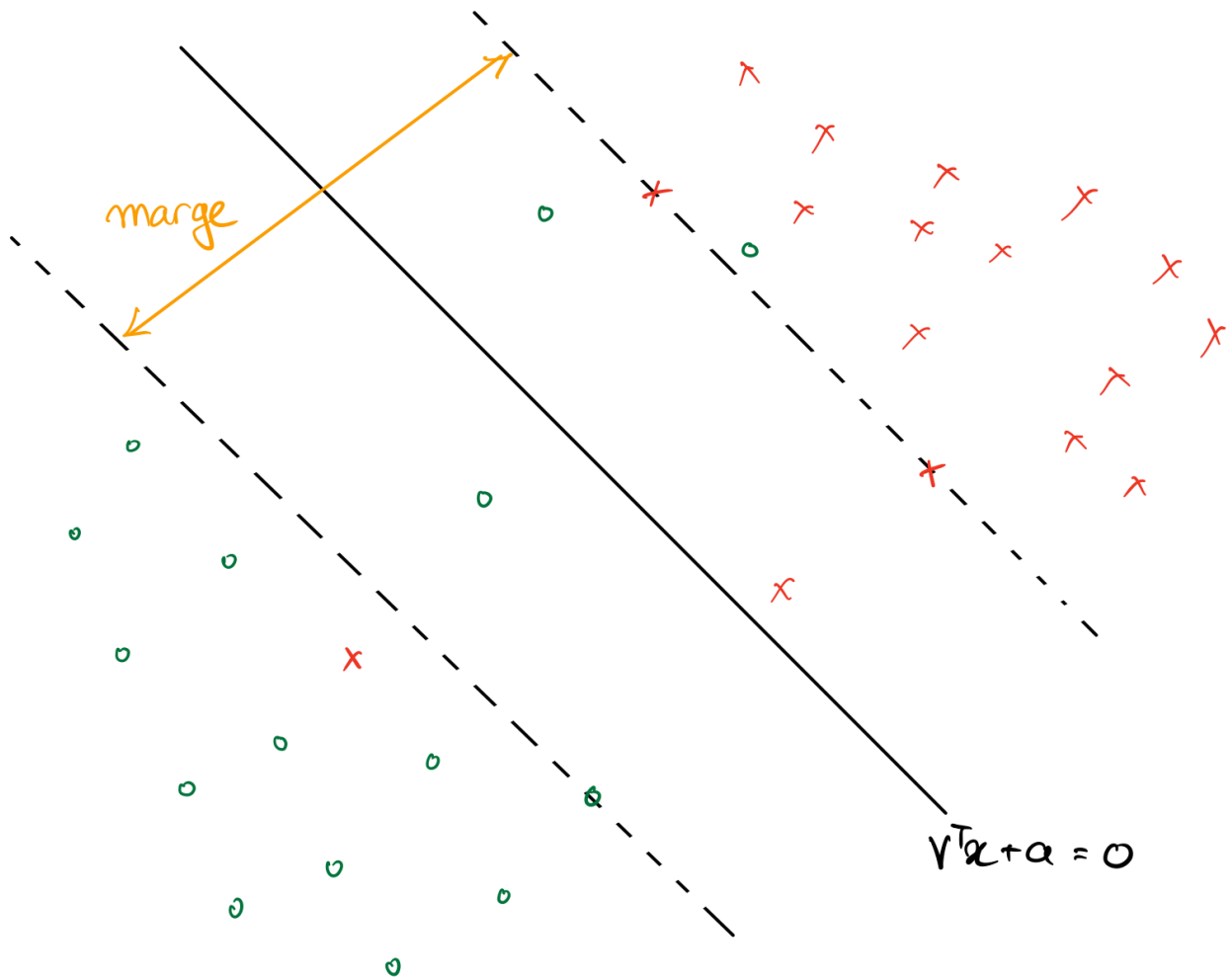
$$\lambda_k^* \Big(y_k ({v^*}^\top x_k + a) - 1 \Big) = 0$$

Les points x_k tels que $\lambda_k^* > 0$ vérifient $y_k(v^{*\top} x_k + a^*) = 1$. Ces points sont sur la frontière de la marge maximale, nous les appelons les **vecteurs supports**.



Retour à la dure réalité

Les données ne sont (presque) jamais linéairement séparables.



Il faut relâcher la construction précédente pour autoriser certains points :

- à être **bien classés** mais à l'intérieur de la marge,
- à être **mal classés** (mais le moins possible).

Il faut relâcher la construction précédente pour autoriser certains points :

- à être **bien classés** mais à l'intérieur de la marge,
- à être **mal classés** (mais le moins possible).

Il n'y a **plus de solution** au problème de minimisation de

$$\frac{1}{2}\|v\|^2 \text{ tel que } \min_{1\leqslant k\leqslant n} y_k(v^\top x_k + a) \geqslant 1.$$

Pour autoriser certains points à être dans la marge et/ou mal classés, nous introduisons des **variables ressort** (*slack variables*) $\xi_1, \ldots, \xi_n \geqslant 0$ telles que

$$\forall k \in \{1, \ldots, n\},\ y_k(v^\top x_k + a) \geqslant 1 - \xi_k.$$

Il faut relâcher la construction précédente pour autoriser certains points :

- à être **bien classés** mais à l'intérieur de la marge,
- à être **mal classés** (mais le moins possible).

Il n'y a **plus de solution** au problème de minimisation de

$$\frac{1}{2}\|v\|^2 \text{ tel que } \min_{1\leqslant k\leqslant n} y_k(v^\top x_k + a) \geqslant 1.$$

Pour autoriser certains points à être dans la marge et/ou mal classés, nous introduisons des **variables ressort** (*slack variables*) $\xi_1, \ldots, \xi_n \geqslant 0$ telles que

$$\forall k \in \{1, \ldots, n\},\ y_k(v^\top x_k + a) \geqslant 1 - \xi_k.$$

- $\xi_k \in [0, 1]$: point bien classé mais dans la marge,
- $\xi_k > 1$: point mal classé.

Bien entendu, si $\xi_k > 0$, nous souhaitons que cette variable ressort demeure la plus petite possible. Plus généralement, nous voulons qu'un maximum de ξ_k soient **nulles**.

Bien entendu, si $\xi_k > 0$, nous souhaitons que cette variable ressort demeure la plus petite possible. Plus généralement, nous voulons qu'un maximum de ξ_k soient **nulles**.

Cela se traduit par l'apparition d'une **pénalité** afin de minimiser

$$(v, a, \xi_1, \ldots, x_n) \longmapsto \frac{1}{2}\|v\|^2 + C\sum_{k=1}^{n} \xi_k$$

sous les contraintes, pour tout $k \in \{1, \ldots, n\}$,

$$y_k(v^\top x_k + a) \geqslant 1 - \xi_k \quad \text{et} \quad \xi_k \geqslant 0.$$

Bien entendu, si $\xi_k > 0$, nous souhaitons que cette variable ressort demeure la plus petite possible. Plus généralement, nous voulons qu'un maximum de ξ_k soient **nulles**.

Cela se traduit par l'apparition d'une **pénalité** afin de minimiser

$$(v, a, \xi_1, \ldots, x_n) \longmapsto \frac{1}{2}\|v\|^2 + C\sum_{k=1}^{n} \xi_k$$

sous les contraintes, pour tout $k \in \{1, \ldots, n\}$,

$$y_k(v^\top x_k + a) \geqslant 1 - \xi_k \quad \text{et} \quad \xi_k \geqslant 0.$$

Lorsque le paramètre $C > 0$ augmente, il est plus difficile de faire entrer un point dans la marge ou de mal le classer. En pratique, ce paramètre se calibre par **validation croisée**.

Ce nouveau problème d'optimisation sous contraintes se traite comme le précédent avec le Lagrangien et le problème dual.

Les conditions KKT donnent, pour tout $k \in \{1, \dots, n\}$,

- $0 \leqslant \lambda_k^* \leqslant C$,
- $y_k({v^*}^\top x_k + a^*) \geqslant 1 - \xi_k^*$,
- $\lambda_k^* \left(y_k({v^*}^\top x_k + a) + \xi_k^* - 1 \right) = 0$,
- $\xi_k^*(\lambda_k^* - C) = 0$.

Les vecteurs supports sont encore les x_k tels que $\lambda_k^* > 0$,

- ceux sur la frontière de la marge vérifient $\xi_k^* = 0$,
- ceux en dehors vérifient $\xi_k^* > 0$ et $\lambda_k^* = C$.

Les vecteurs non-supports sont tels que $\lambda_k^* = 0$ et $\xi_k^* = 0$.

Comme dans le cas séparable, l'hyperplan optimal est donné par (v^*, a^*) tel que

$$v^* = \sum_{k=1}^{n} \lambda_k^* x_k y_k$$

et $y_k({v^*}^\top x_k + a^*) = 1$ donne a^* avec n'importe quel k tel que $0 < \lambda_k^* < C$.

Comme dans le cas séparable, l'hyperplan optimal est donné par (v^*, a^*) tel que

$$v^* = \sum_{k=1}^{n} \lambda_k^* x_k y_k$$

et $y_k({v^*}^\top x_k + a^*) = 1$ donne a^* avec n'importe quel k tel que $0 < \lambda_k^* < C$.

Nous en déduisons le **prédicteur SVM** pour la classification binaire,

$$\forall x \in \mathbb{R}^p,\ \hat{f}(x) = \text{signe}\left({v^*}^\top x + a^*\right),$$

et la marge maximale vaut

$$\frac{1}{\|v^*\|} = \left(\sum_{k=1}^{n} {\lambda_k^*}^2\right)^{-1/2}.$$

SVM non-linéaire

Pour aller au-delà du cas linéaire, il faut invoquer des objets mathématiques appelés **noyaux**. Il s'agit de fonctions qui transportent les observations d'une situation non-linéaire dans **un plus grand espace qui n'a pas à être caractérisé mais qui est linéaire**. Ainsi, tout ce qui a été fait dans le cas linéaire restera valable et les SVM sont encore utilisables.

SVM non-linéaire

Pour aller au-delà du cas linéaire, il faut invoquer des objets mathématiques appelés **noyaux**. Il s'agit de fonctions qui transportent les observations d'une situation non-linéaire dans **un plus grand espace qui n'a pas à être caractérisé mais qui est linéaire**. Ainsi, tout ce qui a été fait dans le cas linéaire restera valable et les SVM sont encore utilisables.

Noyau

Notons $\mathcal{X}$ l'espace des entrées et $\mathcal{H}$ un espace de Hilbert muni d'un produit scalaire $\langle \cdot, \cdot \rangle$. Soit $\phi : \mathcal{X} \to \mathcal{H}$, le **noyau** K associé à ϕ entre deux points x et x' de $\mathcal{X}$ est le produit scalaire de $\phi(x)$ et $\phi(x')$,

$$\begin{array}{rccc} K : & \mathcal{X} \times \mathcal{X} & \longrightarrow & \mathbb{R} \\ & (x, x') & \longmapsto & \langle \phi(x), \phi(x') \rangle \end{array}$$

Par exemple, si $\mathcal{X} = \mathcal{H} = \mathbb{R}^2$ et $\phi(x) = \phi\left((x_1, x_2)\right) = (x_1^2, x_2^2)$, alors

$$K(x, x') = x_1^2 {x'_1}^2 + x_2^2 {x'_2}^2.$$

L'idée générale est que SVM fonctionnera bien dans $\mathcal{H}$ si il est bien choisi au sens où $(\phi(x_1), y_1), \ldots, (\phi(x_n), y_n)$ soient (presque) linéairement séparable.

L'idée générale est que SVM fonctionnera bien dans $\mathcal{H}$ si il est bien choisi au sens où $(\phi(x_1), y_1), \ldots, (\phi(x_n), y_n)$ soient (presque) linéairement séparable.

Tout cela fonctionne grâce au **kernel trick** : le problème dual consiste à maximiser

$$L(\lambda_1, \ldots, \lambda_n) = \sum_{k=1}^{n} \lambda_k - \frac{1}{2} \sum_{k=1}^{n} \sum_{\ell=1}^{n} \lambda_k \lambda_\ell y_k y_\ell \langle \phi(x_k), \phi(x_\ell) \rangle$$
$$= \sum_{k=1}^{n} \lambda_k - \frac{1}{2} \sum_{k=1}^{n} \sum_{\ell=1}^{n} \lambda_k \lambda_\ell y_k y_\ell K(x_k, x_\ell)$$

sous les mêmes contraintes que dans le cas linéaire.

L'idée générale est que SVM fonctionnera bien dans $\mathcal{H}$ si il est bien choisi au sens où $(\phi(x_1), y_1), \ldots, (\phi(x_n), y_n)$ soient (presque) linéairement séparable.

Tout cela fonctionne grâce au **kernel trick** : le problème dual consiste à maximiser

$$L(\lambda_1, \ldots, \lambda_n) = \sum_{k=1}^{n} \lambda_k - \frac{1}{2} \sum_{k=1}^{n} \sum_{\ell=1}^{n} \lambda_k \lambda_\ell y_k y_\ell \langle \phi(x_k), \phi(x_\ell) \rangle$$
$$= \sum_{k=1}^{n} \lambda_k - \frac{1}{2} \sum_{k=1}^{n} \sum_{\ell=1}^{n} \lambda_k \lambda_\ell y_k y_\ell K(x_k, x_\ell)$$

sous les mêmes contraintes que dans le cas linéaire.

Prédicteur SVM (Inutile de connaître $\mathcal{H}$ ou ϕ, le noyau K suffit !)

$$x \in \mathcal{X} \longmapsto \text{signe}\left(\sum_{k=1}^{n} \lambda_k^* y_k K(x_k, x) + a^* \right)$$

Tout cela est bien sympathique **à condition de savoir construire des noyaux**. Le théorème suivant donne pour cela une condition nécessaire et suffisante mais sa preuve dépasse le cadre de ce cours.

Théorème

Une fonction $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ est un noyau si et seulement si elle est symétrique définie positive :

- $\forall x, x' \in \mathcal{X},\ K(x, x') = K(x', x)$,
- $\forall N \in \mathbb{N},\ \forall x_1, \ldots, x_N \in \mathcal{X},\ \forall c_1, \ldots, c_N \in \mathbb{R}$,

$$\sum_{k=1}^{n} \sum_{\ell=1}^{n} c_k c_\ell K(x_k, x_\ell) \geqslant 0.$$

Autrement dit, n'importe quelle **fonction symétrique définie positive** fait l'affaire ! Cela ouvre la possibilité de travailler sur des objets complexes (courbes, images, séquences de lettres, ...).

Quelques exemples de noyaux :

- Linéaire :
$$\forall x, x' \in \mathbb{R}^p, \ K(x, x') = x^\top x'$$
- Polynomial :
$$\forall x, x' \in \mathbb{R}^p, \ K(x, x') = (x^\top x' + 1)^d$$
- Gaussien :
$$\forall x, x' \in \mathbb{R}^p, \ K(x, x') = \exp\left(-\|x - x'\|^2 / 2\sigma^2\right)$$
- Laplace :
$$\forall x, x' \in \mathbb{R}, \ K(x, x') = \exp\left(-\gamma |x - x'|\right)$$
- Min :
$$\forall x, x' \geqslant 0, \ K(x, x') = \min\{x, x'\}$$
- ...

Réseaux de neurones

Introduction

La brique élémentaire d'un réseau de neurones est le **neurone formel**. Cette notion fut introduite dans les années 1940 pour simuler le comportement du cerveau humain puis formalisé et mis en réseau dans les travaux de Rosenblatt à la fin des années 1950.

À l'origine, cela s'appelait un **perceptron** et l'objectif était la reconnaissance de formes. Cette approche a connu un fort essor au fil des décennies suivantes pour se généraliser aux problèmes d'apprentissage supervisé en général.

Introduction

La brique élémentaire d'un réseau de neurones est le **neurone formel**. Cette notion fut introduite dans les années 1940 pour simuler le comportement du cerveau humain puis formalisé et mis en réseau dans les travaux de Rosenblatt à la fin des années 1950.

À l'origine, cela s'appelait un **perceptron** et l'objectif était la reconnaissance de formes. Cette approche a connu un fort essor au fil des décennies suivantes pour se généraliser aux problèmes d'apprentissage supervisé en général.

Cette évolution n'a pas été linéaire car plusieurs obstacles théoriques et pratiques ont dû être surmontés :

- dans les années 1980, la complexité des algorithmes d'entraînement de tels réseaux les rendait inutilisables avec les moyens de l'époque,
- dans les années 1990, le développement d'algorithmes d'apprentissage tels que le boosting ou les SVM qui présentaient une grande simplicité a mis en sommeil les travaux sur les réseaux de neurones.

À partir des années 2000, des avancées théoriques (optimisation stochastique, . . .), des moyens de calculs beaucoup plus performants (GPU, . . .) et un grand battage médiatique (reconnaissance faciale, jeu de go, jeux vidéo, . . .) ont remis ces méthodes en lumière.

À partir des années 2000, des avancées théoriques (optimisation stochastique, ...), des moyens de calculs beaucoup plus performants (GPU, ...) et un grand battage médiatique (reconnaissance faciale, jeu de go, jeux vidéo, ...) ont remis ces méthodes en lumière.

Un **neurone formel** est un objet mathématique qui possède des **entrées** (réelles) $x^1, \ldots, x^p$ (*signaux* ou *stimuli*) et une **sortie** y (*réponse*). Selon sa réponse, un neurone peut être **actif ou inactif**.

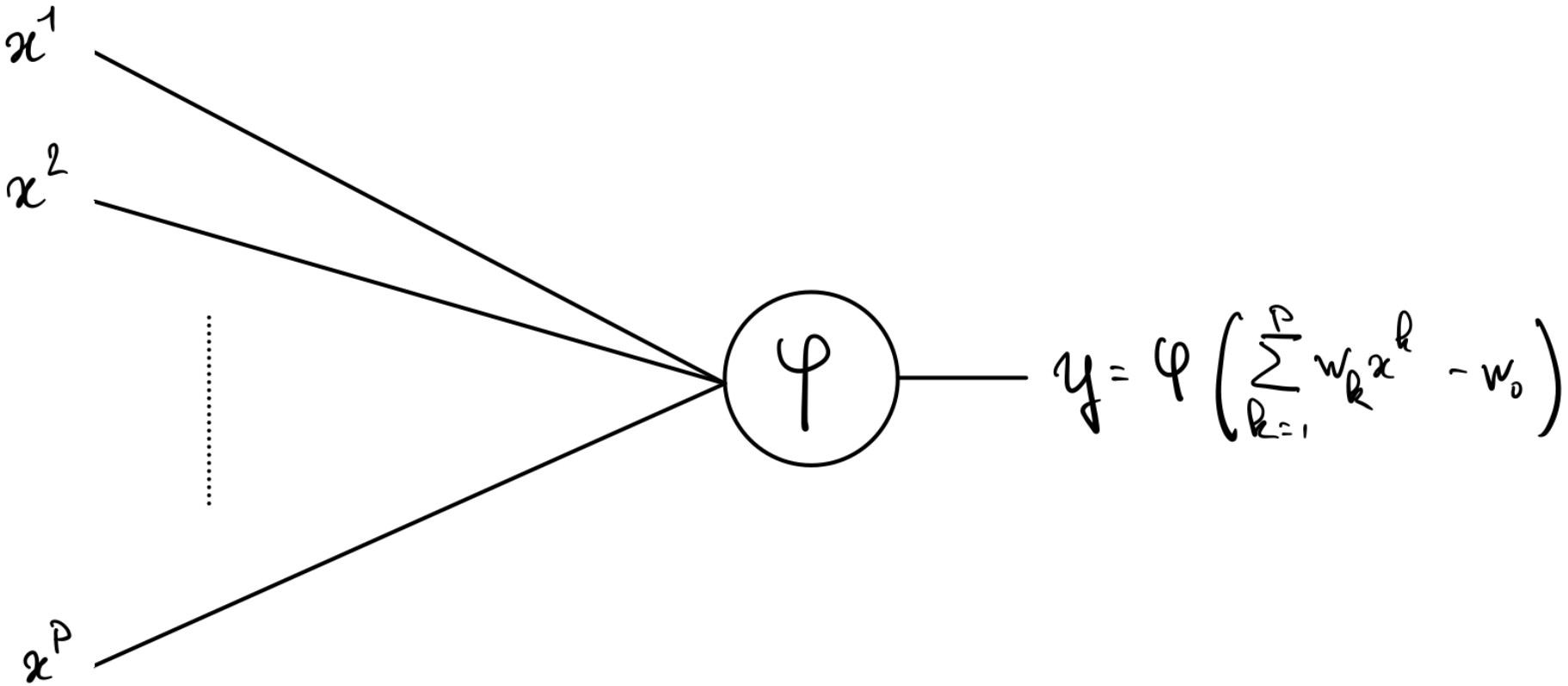
À partir des années 2000, des avancées théoriques (optimisation stochastique, ...), des moyens de calculs beaucoup plus performants (GPU, ...) et un grand battage médiatique (reconnaissance faciale, jeu de go, jeux vidéo, ...) ont remis ces méthodes en lumière.

Un **neurone formel** est un objet mathématique qui possède des **entrées** (réelles) $x^1, \dots, x^p$ (*signaux* ou *stimuli*) et une **sortie** y (*réponse*). Selon sa réponse, un neurone peut être **actif ou inactif**.

Plus précisément, un neurone est constitué par

- des coefficients $w_1, \dots, w_p \in \mathbb{R}$ pour les entrées,
- un **coefficient de biais** $w_0 \in \mathbb{R}$,
- une **fonction d'activation** $\varphi : \mathbb{R} \to \mathbb{R}$.

$$y = \varphi\left(\sum_{k=1}^{p} w_k x^k - w_0\right)$$

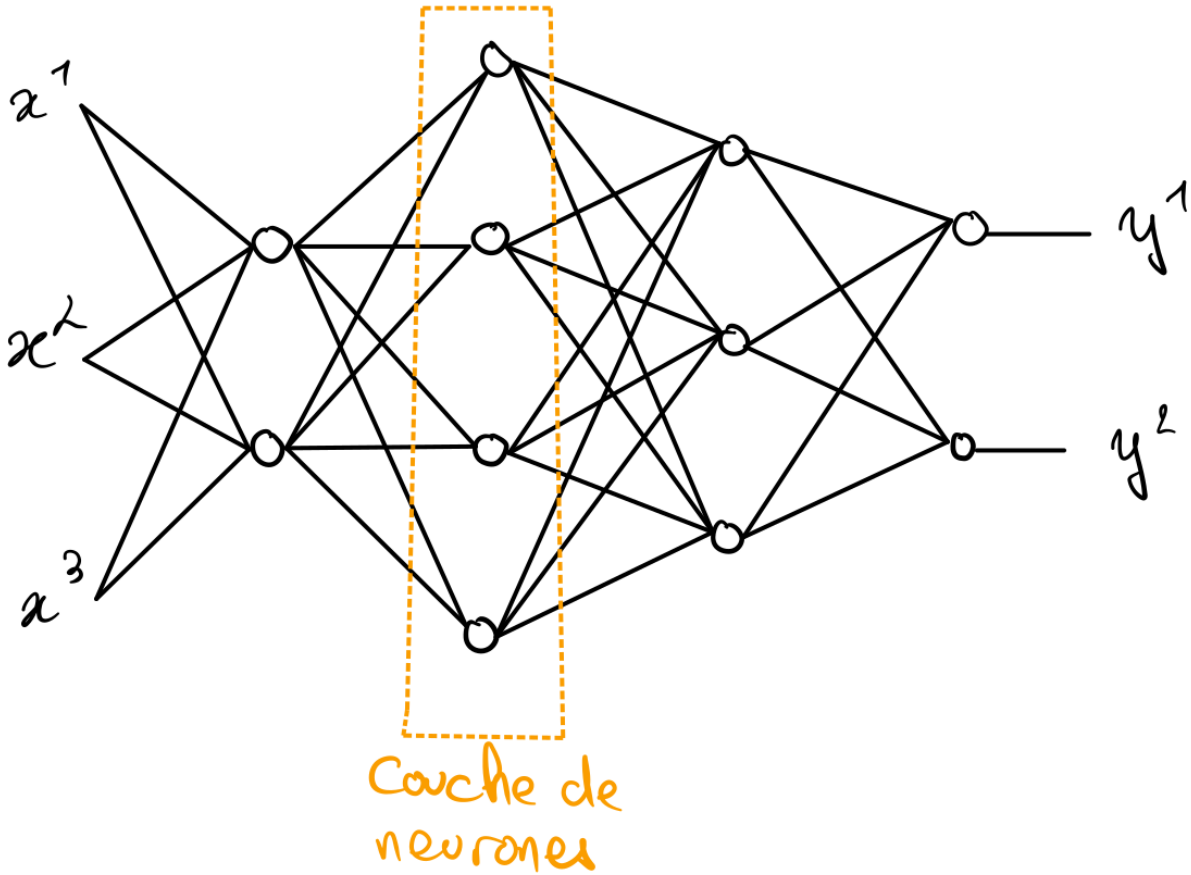


$$y = \varphi\left(\sum_{k=1}^{p} w_k x^k - w_0\right)$$

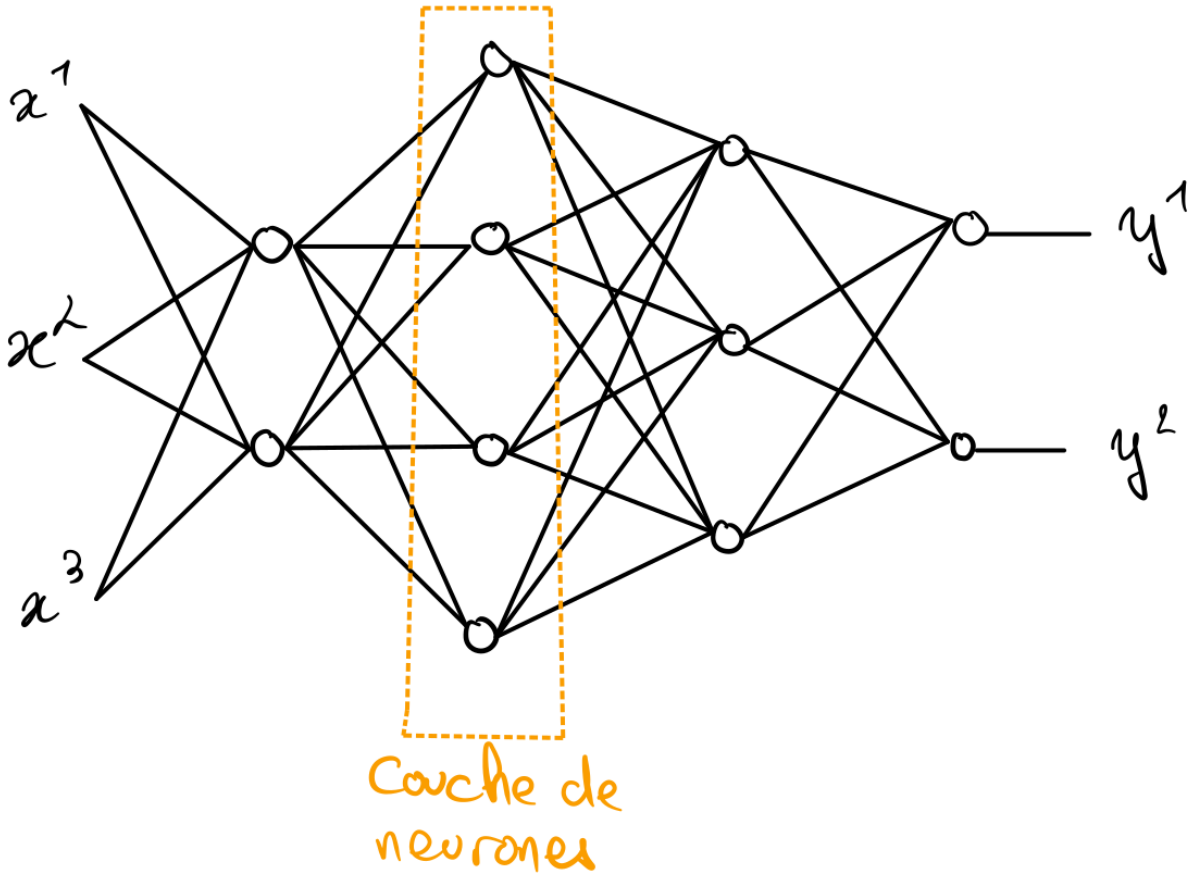
La fonction d'activation φ transforme une **combinaison affine** des entrées en une sortie. Voici quelques exemples de fonctions utilisées en pratique :

- Linéaire : $\varphi(x) = x$
- Seuil : $\varphi(x) = \mathbf{1}_{x \geqslant 0}$
- Sigmoïde : $\varphi(x) = \dfrac{1}{1+e^x}$
- ReLU (*Rectified Linear Unit*) : $\varphi(x) = \max\{0, x\}$
- Radiale : $\varphi(x) = \exp\left(-\dfrac{x^2}{2\sigma^2}\right)$ avec $\sigma^2 > 0$
- Stochastique : $\varphi(x) \sim \text{Ber}\left(\dfrac{1}{1+e^{-x/H}}\right)$ à température $H > 0$

Un réseau de neurones est une association de neurones en un graphe plus ou moins complexe. En apprentissage, de tels réseaux sont souvent **organisés en couches**.



Un réseau de neurones est une association de neurones en un graphe plus ou moins complexe. En apprentissage, de tels réseaux sont souvent **organisés en couches**.



Entraîner un réseau de neurones = Entraîner **tous** ses neurones

Perceptron multicouche

Dans le cadre de ce cours, nous nous limiterons à une structure élémentaire de réseau appelée **perceptron multicouche**. D'autres structures sont possibles en intégrant des boucles de rétroaction, ...

Dans le perceptron multicouche, les neurones sont organisés en couches successives où toutes les sorties de la couche précédente sont les entrées de la suivante.

Perceptron multicouche

Dans le cadre de ce cours, nous nous limiterons à une structure élémentaire de réseau appelée **perceptron multicouche**. D'autres structures sont possibles en intégrant des boucles de rétroaction, ...

Dans le perceptron multicouche, les neurones sont organisés en couches successives où toutes les sorties de la couche précédente sont les entrées de la suivante.

Les couches entre les entrées et les sorties du perceptron multicouche sont appelées des **couches cachées** (*hidden layers*).

Un perceptron multicouche à N couches de $p_1, \ldots, p_N$ neurones respectivement est donc un prédicteur $\hat{f}_W$ qui relie des entrées aux sorties et qui est paramétré par ses coefficients,

$$W = \left\{ (w_0^k, w_1^k, \ldots, w_{p_k}^k) \text{ avec } k \in \{1, \ldots, N\} \right\}.$$

D'un point de vue théorique, il a été démontré qu'une seule couche cachée suffit pour prendre en compte les problèmes classiques d'apprentissage.

D'un point de vue théorique, il a été démontré qu'une seule couche cachée suffit pour prendre en compte les problèmes classiques d'apprentissage.

Ce résultat ne contredit pas l'usage d'un (très) grand nombre de couches car il masque les difficultés d'entraînement et de stabilité pour des problèmes complexes de grande dimension.

D'un point de vue théorique, il a été démontré qu'une seule couche cachée suffit pour prendre en compte les problèmes classiques d'apprentissage.

Ce résultat ne contredit pas l'usage d'un (très) grand nombre de couches car il masque les difficultés d'entraînement et de stabilité pour des problèmes complexes de grande dimension.

Configurations usuelles :

- en régression, la dernière couche contient des neurones avec la fonction d'activation linéaire pour retourner une combinaison des sorties précédentes,
- en classification, nous pouvons utiliser la fonction d'activation sigmoïde et voir les sorties du réseau comme des scores pour les différentes modalités.

Prenons un jeu de données,

$$\Big((x_1^1, \ldots, x_1^p), y_1), \ldots, (x_n^1, \ldots, x_n^p), y_n)\Big) \in \mathbb{R}^p \times \mathcal{Y},$$

et une fonction de perte $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$. L'entraînement du réseau revient à minimiser la fonction des coefficients

$$R_n(W) = \frac{1}{n} \sum_{k=1}^{n} \ell\left(y_k, \hat{f}_W(x_k)\right).$$

Prenons un jeu de données,

$$\Big((x_1^1, \ldots, x_1^p), y_1), \ldots, (x_n^1, \ldots, x_n^p), y_n)\Big) \in \mathbb{R}^p \times \mathcal{Y},$$

et une fonction de perte $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$. L'entraînement du réseau revient à minimiser la fonction des coefficients

$$R_n(W) = \frac{1}{n} \sum_{k=1}^{n} \ell\left(y_k, \hat{f}_W(x_k)\right).$$

La structure du réseau et le nombre de coefficients à calibrer rendent ce problème non-trivial et plusieurs algorithmes d'optimisation sont proposés. Ces algorithmes sont généralement basés sur une évaluation du gradient par **rétro-propagation** (*backpropagation*).

Compte tenu du grand nombre de paramètres, un réseau de neurones est souvent sujet au phénomène de surapprentissage. Pour limiter cela, il est courant de **régulariser** le critère à l'aide d'une pénalité similaire à celle de la **régression ridge** :

$$R_n(W) + \lambda \|W\|^2$$

où $\lambda > 0$ est un paramètre à calibrer.

Compte tenu du grand nombre de paramètres, un réseau de neurones est souvent sujet au phénomène de surapprentissage. Pour limiter cela, il est courant de **régulariser** le critère à l'aide d'une pénalité similaire à celle de la **régression ridge** :

$$R_n(W) + \lambda \|W\|^2$$

où $\lambda > 0$ est un paramètre à calibrer.

Une difficulté rencontrée par les utilisateurs des réseaux de neurones est la grande diversité des paramètres de ces méthodes :

- choix des variables d'entrée et de sortie,
- topologie : nombre de couches cachées, de neurones par couche, ...
- rétro-propagation du gradient et descente : taux d'apprentissage, nombre d'itérations, erreur tolérée, ...
- paramètres divers (λ, ...)
- ... (taille des batchs, ...)

En pratique, entraîner un réseau de neurones est plus coûteux en temps et en ressources que des approches plus classiques. Il s'agit du prix à payer pour la flexibilité des approches neuronales.

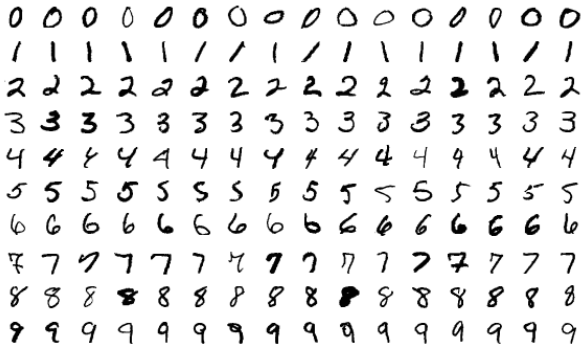
Apprentissage profond (Deep learning)

Profitant du développement des unités de calcul graphique (GPU) pour paralléliser massivement les calculs d'entraînement des réseaux de neurones, de nouvelles approches ont vu le jour depuis le début des années 2010 en empilant **un nombre de plus en plus important de couches**.

Apprentissage profond (Deep learning)

Profitant du développement des unités de calcul graphique (GPU) pour paralléliser massivement les calculs d'entraînement des réseaux de neurones, de nouvelles approches ont vu le jour depuis le début des années 2010 en empilant **un nombre de plus en plus important de couches**.

Parmi ces approches, nous pouvons citer les **réseaux de neurones convolutionnels** pour l'analyse d'image. Ce type d'approche a permis de passer d'un taux d'erreur de 12% avec un perceptron à une couche à 0.3% en 2012 sur la base MNIST. L'idée était d'utiliser des couches neuronales pour manipuler des images à un très grand nombre d'échelles et d'exploiter les propriétés locales d'invariance par translation.

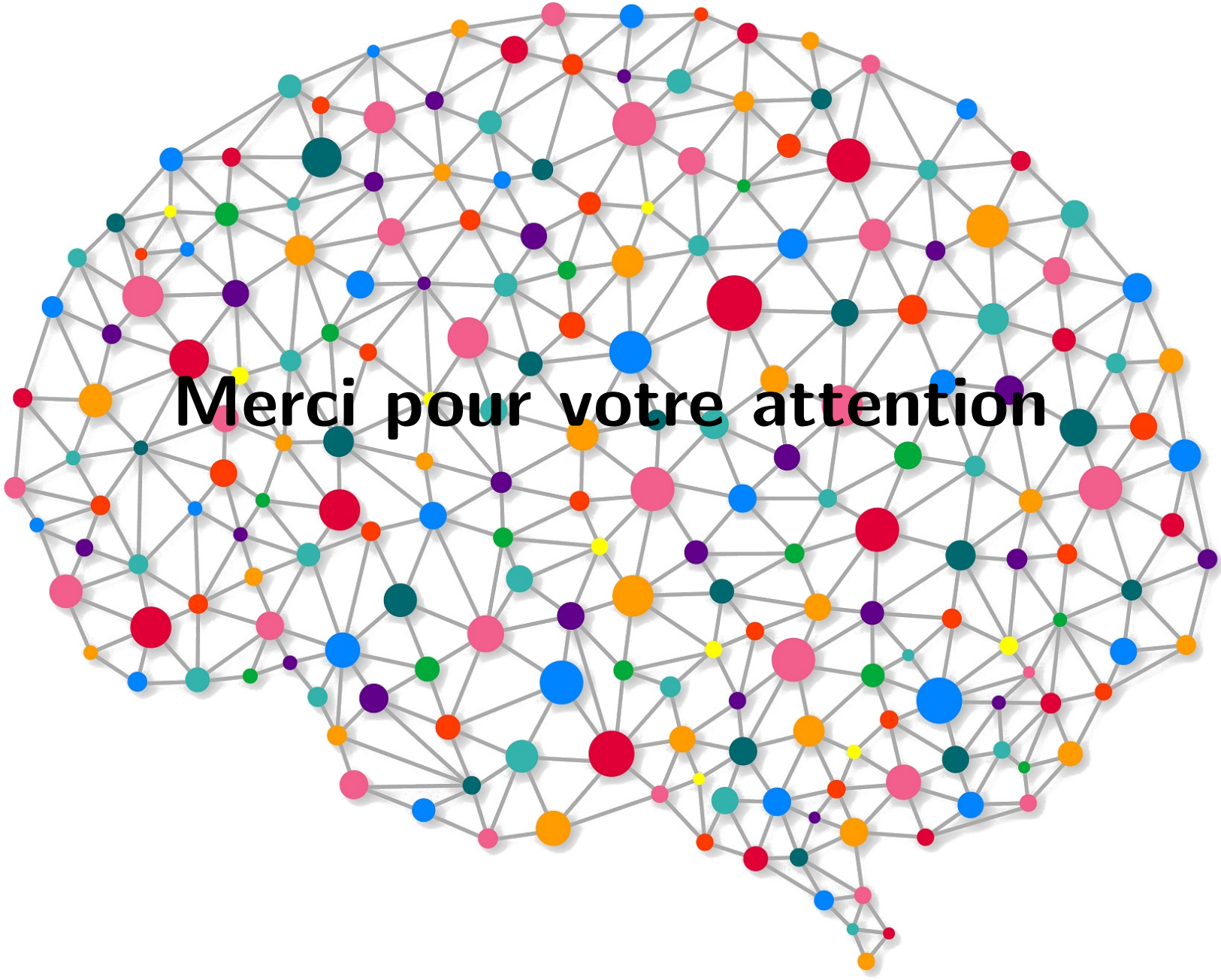


Un réseau de neurones profond implique des milliers de coefficients et son entraînement nécessite des bases de données très volumineuses et des moyens de calcul important.

Un réseau de neurones profond implique des milliers de coefficients et son entraînement nécessite des bases de données très volumineuses et des moyens de calcul important.

Du point de vue architecture, un réseau profond est organisé avec des motifs aux propriétés spécifiques :

- **Fully connected :** couches classiques de perceptron,
- **Convolution :** convolution d'une partie des entrées (caractère d'échelle, réduction de la dimension, ...),
- **Pooling :** résumé d'entrées par une seule valeur (*e.g.* max)
- **Dropout :** mise en sommeil aléatoire de certains neurones,
- ...


Merci pour votre attention

Licence