Machine Learning : Synthesis

INFOM444
PROF. Benoît Frenay

Sophie Fortz
1st master Computer Science
◂ 2017-2018 ▸

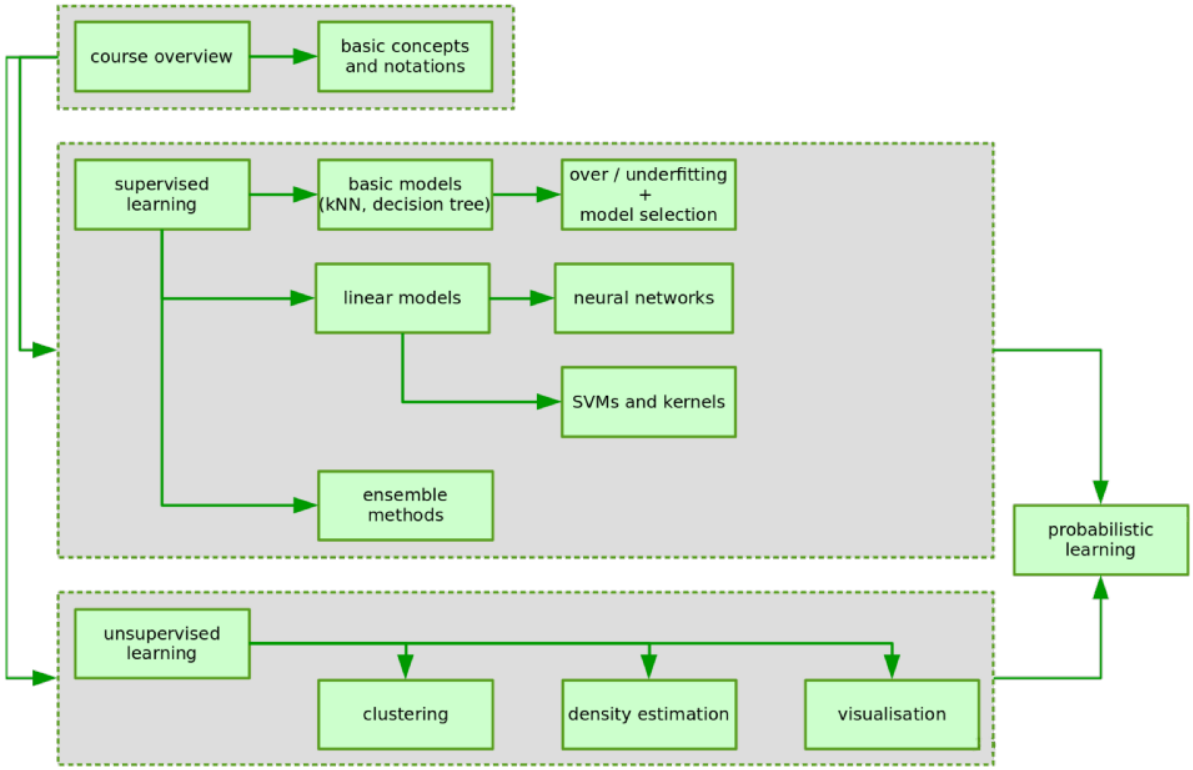
UNIVERSITÉ DE NAMUR

Table des matières

Première partie

Introduction

1 Introduction and Course Overview



1.1 Definition of machine learning

Exemple d'utilisation de Machine Learning :

- Google page rank
- Facebook facial recognition
- Amazon recommendations
- Microsoft Kinect
- Google image caption generator
- etc.

Les données sont partout, en très grande quantité et complexité. La complexité pose de grosse difficulté car elle est souvent quadratique ou cubique, alors que seul le linéaire est abordable actuellement.

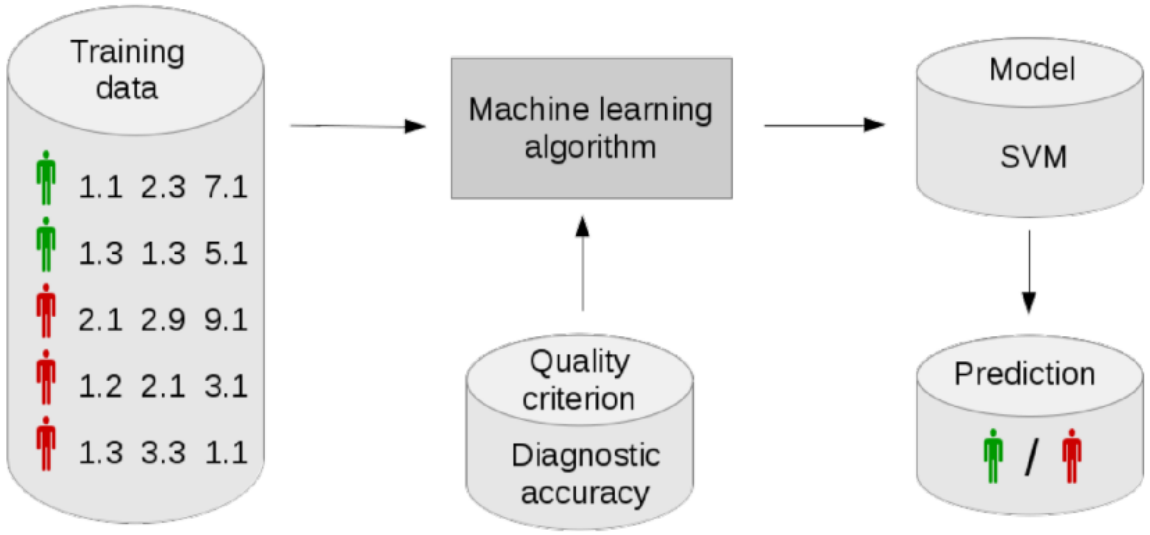
Definition du Machine learning :

Création de modèles descriptifs ou prédictifs permettant l'automatisation de tâches telles que la classification à l'aide de données d'exemple ou d'expérience acquise préalablement.

1.2 Facts and questions about machine learning

Le machine learning est partout dans l'actualité. Pleins de projets sont en cours ou en prévision pour les dizaines d'années à venir. Cette discipline nous renvoie de nombreuses questions

FIGURE 1.1 – Machine learning supervisé



Définit un modèle pour faire une prédiction

FIGURE 1.2 – Machine learning semi-supervisé

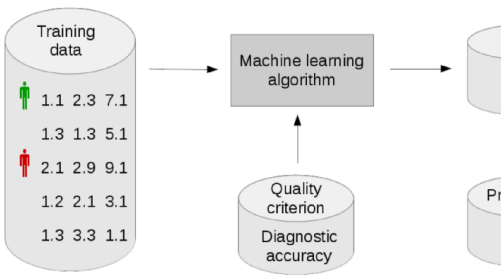
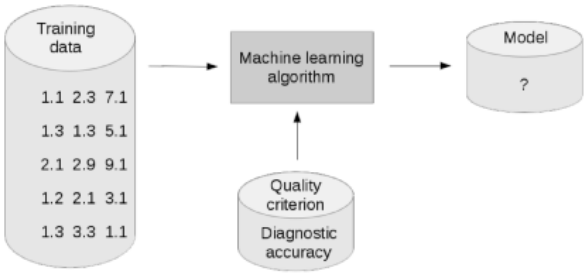


FIGURE 1.3 – Machine learning non supervisé



Les données non labelisées sont regroupées sous-forme d'un nuages de points pour tenter de leur attribuer une étiquette.

d'éthique : peur des robots, science-fiction, perte de contrôle sur la technologie, machine remplaçant les boulots humains, législation, ...

2 Basic Concepts and Notations

2.1 Data, models and learning

Instance : Objet d'intérêt, donnée

Feature : Caractéristique d'une instance (notée $x_1, x_2, ..., x_d$)

Dataset : Ensemble d'instances utilisées pour l'apprentissage :

Supervisé : $\mathcal{D} = \{(\mathbf{x_i}, \mathbf{t_i})\}$

Non supervisé : $\mathcal{D} = \{(\mathbf{x_i})\}$

Pour un data-set de n instances et x_i le vecteur de features du $i^{ème}$ objet et t_i la $i^{ème}$ cible (prédiction réalisée).

Modèle : Approximation du processus de génération des données, caractérisée par ses paramètres

Apprentissage : Exécution d'un algorithme pour optimiser les paramètres du modèle en trois étapes :

Spécification du type de modèle
Spécification du critère de satisfaction
Découverte du meilleur modèle

Description : Obtenir une nouvelle connaissance à propos d'une donnée observée

Prédiction : Prédire le futur d'un objet

Les différents types de tâches :

Classification : Partitionnement d'objets dans différentes classes discrètes (exemple : spam et non spam)

Régression : Associer les objets à une quantité d'intérêt réel (exemple : taille d'un fœtus)

Clustering : Regroupement d'objets non supervisé (exemple : groupement de textes)

Estimation de densité, visualisation, recommandation, graph mining, détection d'anomalies, segmentation d'images, analyse de log, etc.

2.2 Linear regression

(exemple)

2.3 Text classification

(exemple)

Deuxième partie
Supervised learning

3 Introduction to Supervised Learning

3.1 What is supervised learning

Définition de Machine learning supervisé :

L'apprentissage supervisé utilise des vecteurs d'entrées et leur vecteur cible correspondant pour guider un processus d'apprentissage de la fonction de mappage entre entrées et sorties. Ce type de problème se divise en deux catégories : les problèmes de classification, qui ont pour but d'assigner chaque vecteur d'entrée à un nombre fini de catégories discrètes, et les problèmes de régressions qui ont en sortie une ou plusieurs variables continues.

Voici quelques exemples de problèmes supervisés :

Problèmes de classification :

- Reconnaissance faciale
- Reconnaissance d'émotions
- Filtrage de spam
- Diagnostic automatisé
- Détection de chute des personnes âgées

Problèmes de régression :

- Contrôleur de pancréas artificiel
- Estimation du prix de l'immobilier
- Analyse de survie

L'objectif est donc de trouver les meilleurs paramètres du modèles $y = f(x|\theta)$ pour la tâche définie en minimisant le critère d'erreur sur l'ensemble des données d'entraînement $\mathcal{D} = \{(x_i, t_i)\}$ avec :

- $f(x|\theta)$ = la fonction prenant x comme entrée.
- θ = l'ensemble des paramètres du modèle.

La valeur de l'erreur (proche de zéro mais non nulle, à cause du bruit irréductible), définie à l'aide d'une matrice de confusion, dépend donc à la fois de $\mathcal{D}$ et de θ. Les cibles définissent ce que l'on désire apprendre et le critère d'erreur permet d'indiquer l'importance d'une erreur.

Critères d'erreur courants

Un critère d'erreur classique en régression est l'erreur quadratique moyenne (*Mean Square Error*) :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - t_i)^2$$

alors qu'en classification, nous utiliserons plutôt le taux d'erreur de classification :

$$Tauxd'erreur = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}[f(x_i) \neq t_i]$$

3.2 What is generalisation

Définition de la généralisation

La généralisation est la capacité d'un modèle à réaliser des prédiction pour des nouveaux objets jamais rencontrés.

Pour généraliser, on utilise l'hypothèse d'apprentissage inductif (besoin de modèles utiles) : si un modèle se rapproche du processus cible sur un grand nombre d'instances d'apprentissage, il se rapprochera aussi bien des nouvelles instances non observées. Le modèle est une fonction de prédiction ou une fonction de compression qui permet de ne pas tenir compte des données mais uniquement de la fonction définie.

Définition du biais d'apprentissage

Pour généraliser, nous avons besoin d'hypothèses sur le processus de génération. Le biais d'apprentissage (souvent implicite et difficile à remarquer) est l'ensemble des hypothèses qui permettent la généralisation. Sans lui, il est impossible de généraliser.

Un biais peut par exemple être une hypothèse sur la continuité de la fonction, l'indépendance des données, la linéarité de la relation entre x et t, ou que le bruit sur les valeurs cibles t est Gaussien. Ces choix détermineront la qualité du modèle et ne pas choisir le bon biais pourrait avoir des conséquences désastreuses. Une connaissance préalable sur le processus peut être obtenue des experts pour aider à faire des hypothèses appropriées.

4 Basic Models for Supervised Learning

4.1 K-nearest neighbours

Entraînement d'un classificateur k-NN : Enregistrement du dataset pour de futures prédictions.

Input : Le dataset $\mathcal{D} = \{(x_i, t_i)\}$
Output : Le classificateur k-NN

Prédiction à l'aide d'un classificateur k-NN : Détermination des k voisins les plus proches de x dans l'ensemble des données d'entraînement $\mathcal{D} : x_{i_1}, ..., x_{i_k}$ et retour de la classe majoritaire y parmi les labels $t_{i_1}, ...t_{i_k}$ correspondants.

Input : Une nouvelle instance de x

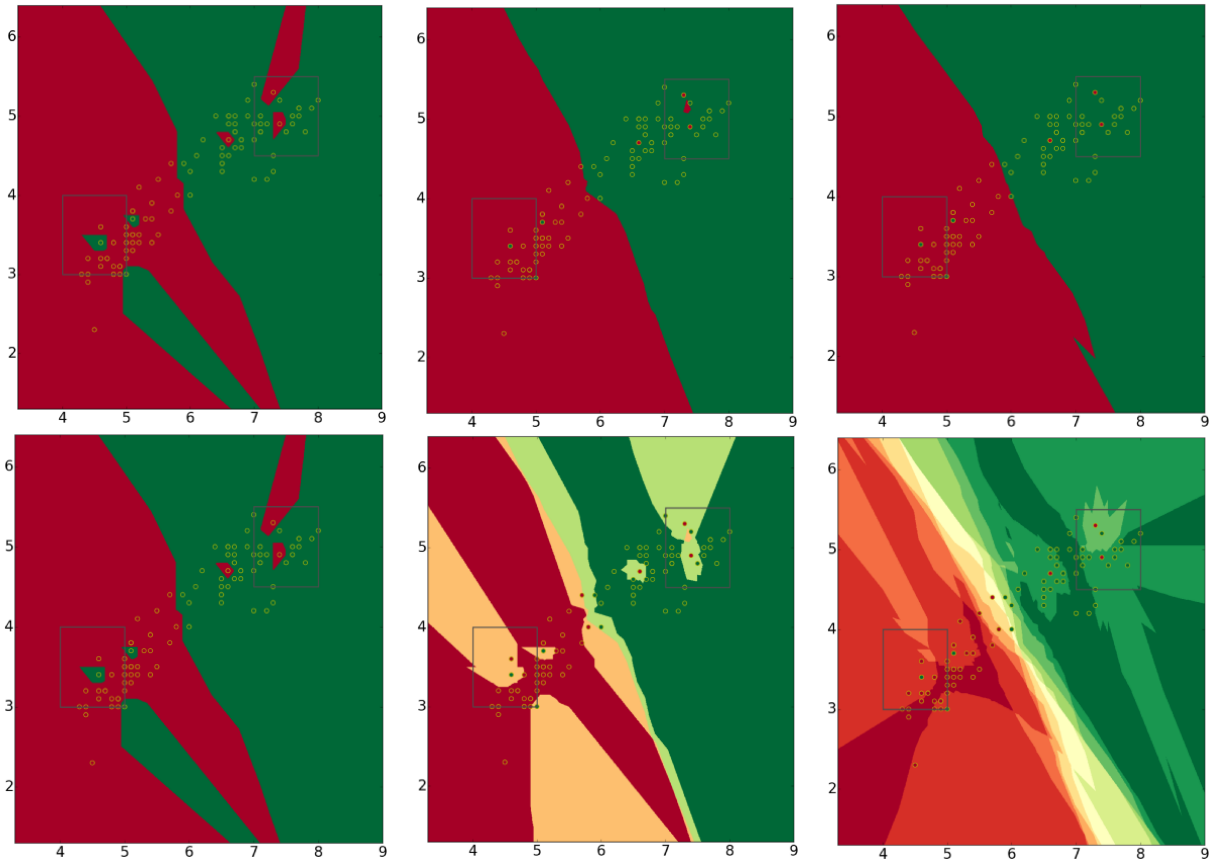
Output : La classe y prédite

Biais d'apprentissage : La classification d'une instance est proche de la classification des instances proches.

FIGURE 4.1 – Classification k-NN pour $k = 1$

FIGURE 4.2 – Classification k-NN pour $k = 3$

FIGURE 4.3 – Classification k-NN pour $k = 10$



Le premier étage représente une classification classique, le second une classification avec niveaux de confiance.

Avantages du classificateur kNN :

- ✓ Facile à comprendre et simple à mettre en œuvre
- ✓ Pas de procédure d'apprentissage fastidieuse (apprentissage *lazy*)
- ✓ Donne (étonnamment) de bons résultats et est plutôt robuste
- ✓ Peut être généralisé à des distances non-euclidiennes
- ✓ Version probabiliste $p(y|x) = \#$ (voisins de x avec étiquette $y)/k$

Problèmes potentiels :

- ✘ Coût de calcul de la prédiction : $\mathcal{O}(n)$
- ✘ Utilisation de la mémoire pour le stockage de données : $\mathcal{O}(n)$
- ✘ Ne convient pas à la modélisation descriptive
- ✘ Que se passe-t-il si l'une des caractéristiques est plus importante ?

Entraînement pour la régression k-NN : Enregistrement du dataset pour de futures prédictions.

Input : Le dataset $\mathcal{D} = \{(x_i, t_i)\}$

Output : Le k-NN pour la régression

Prédiction à l'aide d'un k-NN pour la régression : Détermination des k voisins les plus proches de x dans l'ensemble des données d'entraînement $\mathcal{D}$: $x_{i_1}, ..., x_{i_k}$ et retour de la valeur cible moyenne pour les voisins $y = \frac{1}{k} \sum_{s=1}^{k} t_{i_s}$.

Input : Une nouvelle instance de x

Output : La valeur cible y prédite

Biais d'apprentissage : La valeur cible d'une instance est proche de la valeur cible des instances proches.

FIGURE 4.4 – Régression k-NN pour $k = 1$

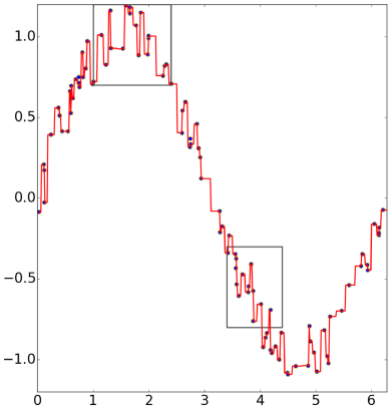


FIGURE 4.5 – Régression k-NN pour $k = 3$

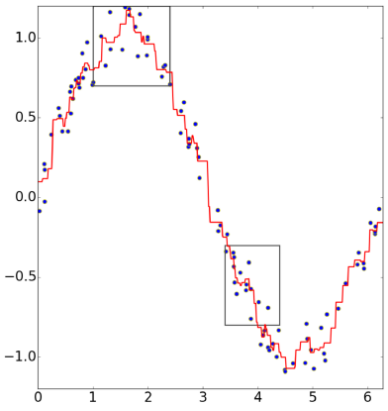
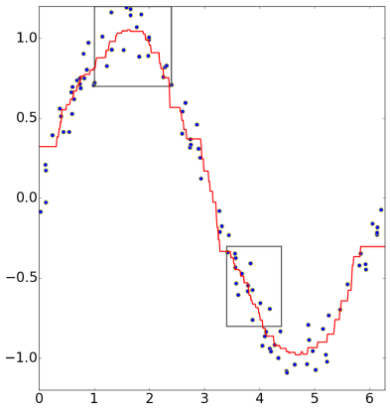


FIGURE 4.6 – Régression k-NN pour $k = 10$



Aux extrémités, l'interpolation pose problème car se sont toujours les même données prises en compte. Pour corriger se problème, il faut extrapoler.

4.2 Decision trees

L'être humain classifie à l'aide de règles, un moyen puissant d'exprimer la connaissance d'un expert dans des modèles descriptifs. Cependant, ces règles sont difficiles à lire et surtout, difficiles à définir. Les arbres de décisions partent de l'idée que c'est règles pourraient être définies directement à partir des données.
Un arbre de décision contient trois types de nœuds : la racine, les nœuds internes et les feuilles. Une prise de décision démarre à la racine qui correspond à la première caractéristique. Chaque arête sortante correspond à une valeur de cette caractéristique et chaque nœud interne est également une caractéristique. Chacune des feuille correspond à l'une des décisions possibles.
Pour pouvoir extraire les règles logique d'un arbre de décision, il suffit de considérer chaque chemin comme une conjonction termes. Ces termes sont tests sur la valeur d'une caractéristique particulière (nœud interne). Chaque chemin peut alors être écris sous la forme d'une règle :

if *(conjonction)* ***then*** *décision* (feuille).

Rappel : Définition de l'entropie

L'entropie (ou incertitude) de Shannon de la distribution de la probabilité $p(Y)$ sur la classe Y se définit comme :

$$H[p] = -\sum_{y \in Y} p(y) \log p(y)$$

Algorithm 1: Algorithme ID3($\mathcal{D}, \mathcal{F}$)

Input: Le dataset $\mathcal{D} = \{x_i, t_i\}$ et l'ensemble $\mathcal{F}$ de caractéristiques
Output: L'arbre de décision récursif classifiant $\mathcal{D}$ selon les caractéristiques dans $\mathcal{F}$

if *toutes les instances ont le même label* t **then**
  return un nœud avec le label t
else if *l'ensemble des features* $\mathcal{F}$ *est vide* **then**
  return un nœud avec le label t le plus présent dans $\mathcal{D}$
else
  création d'un nœud où les décisions utiliseront la meilleure caractéristique $X_j \in \mathcal{F}$ en fonction de $\mathcal{D}$
  for *chaque valeur* v *de la caractéristique* X_j **do**
    if $\mathcal{D}_v = \{x_i \in \mathcal{D} | x_{ij} = v\} \neq \varnothing$ **then**
      ajout d'un enfant ID3($\mathcal{D}_v, \mathcal{F} \backslash \{X_j\}$) au nœud courant
    else
      ajout d'un enfant au nœud courant avec le label t le plus présent dans $\mathcal{D}$
    end if
  end for
  return le nœud courant
end if

L'algorithme d'apprentissage ID3 (Algorithm 1) n'explique pas comment choisir le meilleur critère $X_j \in \mathcal{F}$ qui est pourtant déterminant dans la qualité de l'arbre de décision. Le gain d'information est une mesure de la façon dont une caractéristique donnée sépare les instances d'entraînement (réduction de l'impureté). Pour chaque caractéristique X_j, le gain peut se calculer comme :

$$gain(\mathcal{D}, X_j) = H[p] - \sum_{v \in X_j} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} H[p_v]$$

ou p est la distribution de classe dans $mathcal D$ et pour chaque valeur v de la caractéristique X_j :

$$p_v(t|x) = \text{ le pourcentage des instances de la classe } t \in \mathcal{D} \text{ avec } X_j = v$$

Algorithm 2: decision_tree_classify(r, x)

Input: La racine r de l'arbre de décision, une nouvelle instance x
Output: La classe y prédite

if r *est une feuille avec le label* t **then**
  return la classe $y = t$
else
  Soit X_j la caractéristique de décision associée à r
  Soit c le fils de r sur la branche $X_j = x_j$
  return la classe $y = decision\_tree\_classify(c, x)$
end if

Il est important de noter que l'algorithme de prédiction (Algorithm 2) est indépendant de l'algorithme d'apprentissage (Algorithm 1).
Avantages des arbres de décision :

- ✓ Facile à comprendre, simple à mettre en œuvre
- ✓ Procédure d'apprentissage efficace, peut être effectuée en ligne
- ✓ Peut être utilisé pour la modélisation prédictive/descriptive
- ✓ Facile à expliquer aux non-experts en machine learning
- ✓ Peut être étendu à des variables continues (par exemple, division binaire $x > v$)

Problèmes potentiels :

✘ Le nombre de nœuds peut augmenter très rapidement pour les grands ensembles de données

✘ Trouver le plus petit arbre est un problème NP-complet (ID3 est une heuristique gloutonne)

FIGURE 4.7 – Régression k-NN pour $k = 1$

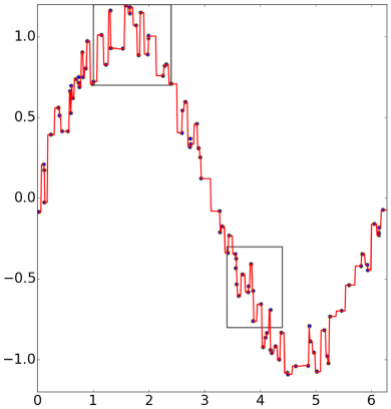


FIGURE 4.8 – Régression k-NN pour $k = 3$

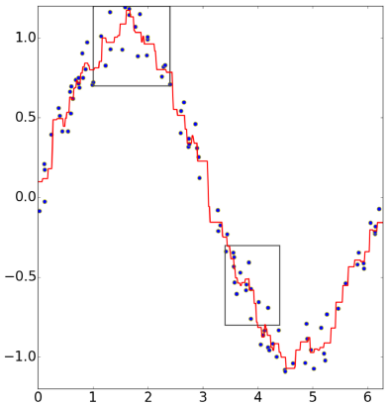
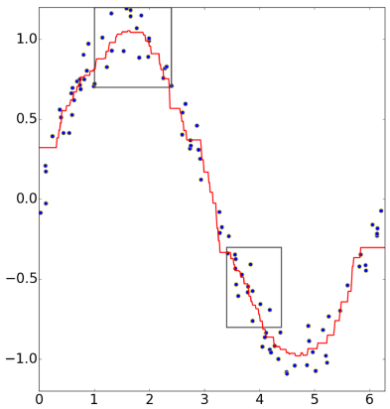


FIGURE 4.9 – Régression k-NN pour $k = 10$



Aux extrémités, l'interpolation pose problème car se sont toujours les même données prises en compte. Pour corriger se problème, il faut extrapoler.

✘ Expressivité limitée (une seule variable à la fois)

5 Overfitting, Underfitting and Model Selection

5.1 Overfitting/Underfitting

Définition de la complexité et de (méta-)paramètre

La complexité d'un modèle est la capacité de la classe de fonctions qui peut être approximée par le modèle. Un méta-paramètre défini la complexité du modèle et doit donc être choisit avant la sélection du modèle (*learning*). Un paramètre détermine la fonction particulière que le modèle définit (obtenu par l'apprentissage, en fonction des méta-paramètres).

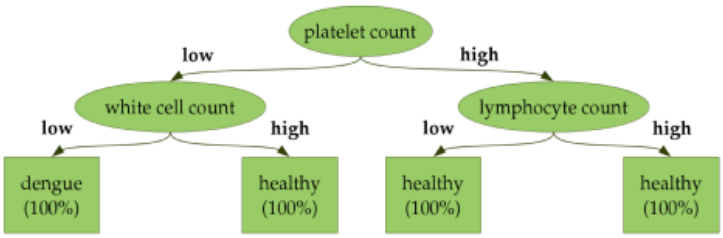
model	meta-parameters	parameters
kNN classifier	number k of neighbours	-
decision tree	depth / number of nodes	nodes (decisions)
polynomial	order (largest exponent)	coefficients
neural network	number of neurons	synaptic weights
clustering	number of clusters	center/size of clusters

Définition d'overfitting et underfitting

Un modèle *overfit* s'il possède une complexité trop importante pour un nombre de données d'entraînement faible. Le modèle sera alors beaucoup trop souple et aura tendance à coller fortement au bruit. L'*underfitting* est le phénomène inverse, qui entraîne un pauvre généralisation des données et ne permet donc pas de réaliser de bonnes prédictions.

FIGURE 5.1 – Too Complex Models : the Overfitting Phenomenon

Arbre de décision pour $d = 2$ et $n = 4$



Régression linéaire
pour $p = 9$ et $n = 20$

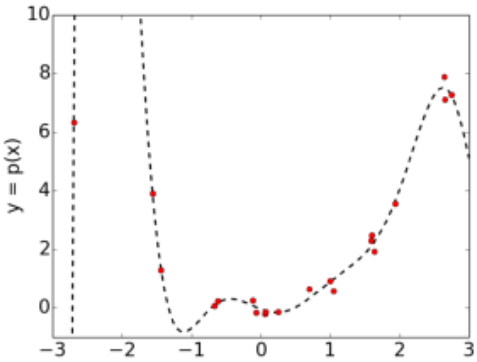
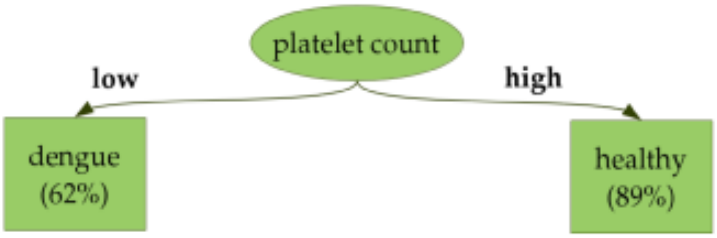
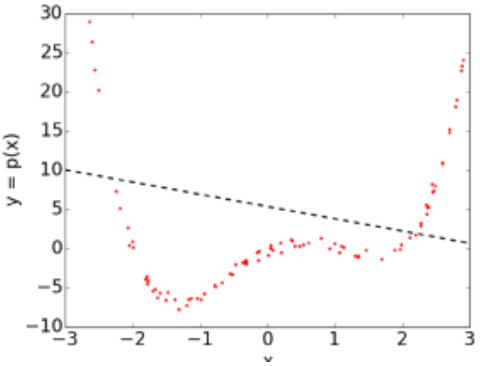


FIGURE 5.2 – Too Simple Models : the Underfitting Phenomenon

Arbre de décision pour $d = 2$ et $n = 1200$



Régression linéaire
pour $p = 3$ et $n = 100$



Le méta-paramètre doit donc être sélectionné consciencieusement en fonction du nombre de données et de la qualité pour éviter d'*overfit* ou d'*underfit*.

5.2 Definition of model selection

La sélection du modèle permet de trouver les meilleures méta-paramètres possibles. En pratique, la sélection du modèle est effectuée avant l'optimisation des paramètres. Cette sélection ne peut se faire manuellement (intraitable) et elle dépend des caractéristique du data-set d'entraînement (taille, qualité, etc.).
Pour ce faire, on utilise une généralisation (estimée) de l'erreur. L'erreur d'entraînement est un estimateur trop optimiste et ne peut donc être utilisée (biaisée).
Voici les formules des estimateurs pour les deux critères d'erreur vu précédemment :

Estimateurs d'erreur courants

L'estimateur de l'erreur quadratique moyenne pour le modèle f peut se calculer comme : Un critère d'erreur classique en régression est l'erreur quadratique moyenne (*Mean Square Error*) :

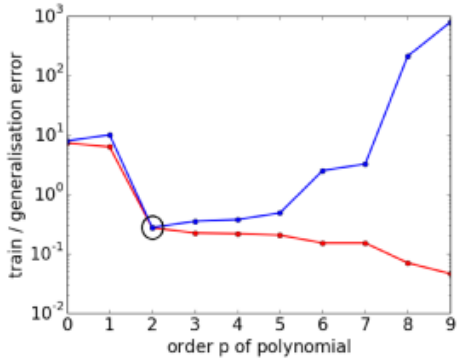
$$\mathop{\mathbb{E}}_{x}\left[\mathop{\mathbb{E}}_{t|x}[(f(x)-t)^2]\right] = \mathop{\mathbb{E}}_{x,t}[(f(x)-t)^2] \approx \frac{1}{n}\sum_{i=1}^{n}(f(x_i)-t_i)^2$$

Et l'estimateur du taux d'erreur de classification pour le modèle f peut se calculer comme :

$$\mathop{\mathbb{E}}_{x}\left[\mathop{\mathbb{E}}_{t|x}[\mathbb{I}[f(x)\neq t]]\right] = \mathop{\mathbb{E}}_{x,t}[\mathbb{I}[f(x)\neq t]] \approx \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}[f(x_i)\neq t_i]$$

Ces deux estimateurs convergent vers l'erreur généralisation lorsque $n \to \infty$, mais en pratique la taille de l'ensemble de validation n est finie , il faut donc être très prudent.

p	E_{train}	$\hat{E}_{\text{gen}}$
0	7.291	7.927
1	6.288	10.013
* **2**	**0.272**	**0.275**
3	0.223	0.351
4	0.217	0.374
5	0.205	0.484
6	0.151	2.511
7	0.151	3.243
8	0.070	211.622
9	0.046	791.109

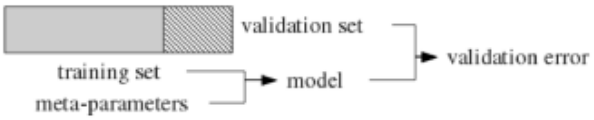


5.3 Validation-based model selection

5.3.1 Simple validation

Procédure :

- Diviser les données en un ensemble d'apprentissage et un ensemble de validation
- Utiliser l'ensemble d'entraînement pour former le modèle avec des méta-paramètres
- Utiliser l'ensemble de validation pour estimer l'erreur de généralisation



Avantages et inconvénients :

✓ Facile et rapide, intuitif, convergent lorsque $n \to \infty$ (estimateur sans biais)

✘ Peu fiable : une seule répétition, on risque donc d'être (mal)chanceux

5.3.2 Cross-Validation

Procédure :

- Simple validation répétée
- Ensemble de données mélangé avant chaque répétition
- Les estimations d'erreur de généralisation pour chaque répétition sont moyennées

Avantages et inconvénients :

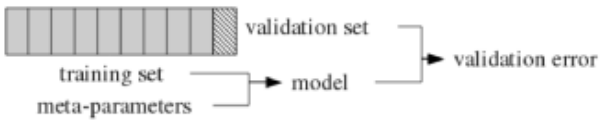
✓ Plus fiable : peu susceptible d'être (mal)chanceux si suffisamment de répétitions

✘ Chevauchement potentiellement important entre les ensembles d'entraînement et de validation

5.3.3 K-Fold Cross-Validation

Procédure :

- L'ensemble de données est divisé en k plis (fixés pour toutes les répétitions), habituellement $k = 10$
- A chaque répétition, entraînement = k - 1 plis et validation = 1 pli
- Chaque pli n'est utilisé qu'une seule fois comme jeu de validation



Avantages et inconvénients :

✓ Mêmes avantages que la validation croisée, petit nombre k de répétitions

✓ Pas de chevauchement (les estimations d'erreur de généralisation sont $\pm$ indépendantes)

✘ Les répétitions prennent un temps considérable

5.3.4 Grid Search for Meta-Parameter Optimisation

Algorithm 3: grid_search($\mathcal{D}, k$)

Input: Le dataset $\mathcal{D} = \{x_i, t_i\}$ et le nombre k de plis
Output: Le modèle avec les méta-paramètres optimaux

for *chaque valeur* α *de méta-paramètres possibles* **do**
    $\hat{E}_{gen}(\alpha) = compute\_kfcv\_error(\mathcal{D}, k, \alpha)$
end for
return le modèle appris avec $\mathcal{D}$ et les méta-paramètres $\alpha^* = \arg\min_\alpha \hat{E}_{gen}(\alpha)$

Algorithm 4: compute_kfcv_error($\mathcal{D}, k, \alpha$)

Input: Le dataset $\mathcal{D} = \{x_i, t_i\}$, le nombre k de plis et les méta-paramètres α
Output: L'erreur de généralisation estimée du meilleur modèle avec les méta-paramètres α

$\hat{E}_{gen}(\alpha) = 0$
for *chaque plis* **do**
    Diviser les k plis de $\mathcal{D}$ en $\mathcal{D}_{train}$ et $\mathcal{D}_{val}$
    Entraîner le modèle avec $\mathcal{D}_{train}$ et les méta-paramètres α
    $\hat{E}_{gen}(\alpha) += \frac{1}{k}$ (erreur de prédiction du modèle sur $\mathcal{D}_{val}$)
end for
return $\hat{E}_{gen}(\alpha)$

5.4 Advanced techniques and model testing

Pour pouvoir valider un modèle, il est nécessaire de le tester. Les erreurs d'entraînement et de validation sont biaisées car elles ont été utilisées pour sélectionner les paramètres et méta-paramètres respectivement. En pratique, il faut utiliser un nouveau jeu de données appelées données de test et évaluer le modèle dans un contexte réel. L'étape de test est d'importance cruciale.

6 Linear Models for Regression and Classification

6.1 Regression with linear models

La régression linéaire fait comme hypothèse que les valeurs des caractéristiques de x et que la valeur cible t sont linéairement reliées :

$$f(x_1, ..., x_d) = w_1 x_1 + ... + w_d x_d + w_0$$

avec $x \in \mathbb{R}^d$, $t \in \mathbb{R}$ et chaque contribution de la caractéristique x_j dans t a un poids w_j.
Il faut cependant faire attention à deux choses :

- Les vecteurs de caractéristiques peuvent être définis selon des échelles (unité de mesure) différentes. Pour normaliser les valeurs :
  Soit $(x - x_{min})/(x_{max} - x_{min})$ et $x \in [0, 1]$
  Soit $(x - \mu_x)/(\sigma_x)$ (ou sa valeur corrigée en remplaçant σ_x par σ_{x_0,x_n}), et $x \in \mathcal{N}[0, 1]$
- Certaines caractéristiques fournies n'ont pas de sens ou ne sont pas connues et n'influencent donc pas la cible.

Pour tenir compte des relations partiellement non-linéaire entre x et t et du bruit, on peut utiliser un critère d'erreur, comme le MSE. Les modèles linéaires peuvent être optimisés pour la régression (avec le MSE) grâce à deux algorithmes :

- La méthode pseudo-inverse : Solution analytique, exacte en une étape, rapide, facile à implémenter et peu coûteuse, elle est cependant sensible aux grandes dimensions (complexité en temps = $\mathcal{O}(d^3)$ et en espace = $\mathcal{O}(d^2)$) et la matrice $\mathbf{X^T X}$ peut ne pas être inversible.

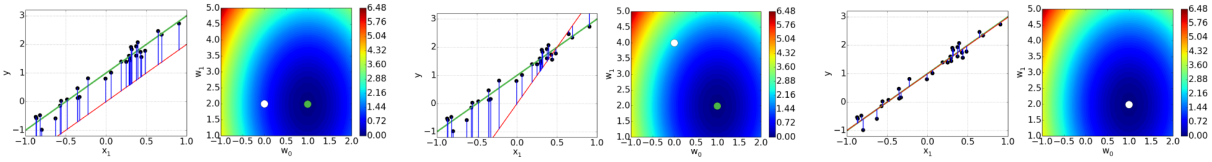
Algorithm 5: pseudo-inverse method

Input: Le data-set $\mathcal{D} = \{(x_i, t_i)\}$
sous la forme matricielle $\mathbf{X}$/vectorielle $\mathbf{t}$ **Output**: Les poids optimaux pour la régression linéaire (tenant compte de la MSE)

return $\mathbf{w} = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (t_i - f(x_{i1}, ..., x_{id}))^2 = (\mathbf{X^T X})^{-1} \mathbf{X^T t}$

- Les algorithmes itératifs comme la descente de gradient (stochastique) :

FIGURE 6.1 – $f(x) = 2x + 1 + \varepsilon \sim \mathcal{N}(0, 0.2)$ et $n = 30$



$f'(x) = 2x$ $\quad$ $f'(x) = 4x$ $\quad$ $f'(x) = 2x + 1$

6.2 Classification with linear models

Avantages de la régression linéaire pour la classification :

- ✓ Les classes peuvent être facilement converties en nombres
- ✓ Techniquement, rien ne nous empêche d'utiliser la régression linéaire
- ✓ La régression linéaire est très efficace et facile à comprendre

Désavantages de la régression linéaire pour la classification :

- ✘ N'a pas vraiment de sens : les cibles sont des classes, pas des nombres réels
- ✘ La fonction objectif ne convient pas (LA MSE sur les classes n'a aucun sens)
- ✘ Les résultats changeront si nous changeons l'ordre des classes

- ✘ La classification est un problème très différent qui nécessite des techniques spécifiques (par exemple, classes déséquilibrées, label noise, coûts de mauvaise classification, etc.)

Cependant, un variante appelée régression logistique offre de nettement meilleurs résultats. Le log-ratio des probabilités de classes conditionnelles est supposé linéaire :

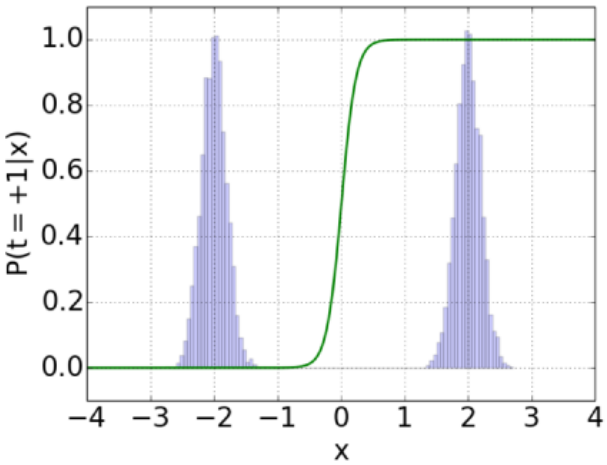
$$\log\left(\frac{p(t=+1|x)}{p(t=-1|x)}\right) = w_1x_1 + ... + w_dx_d + w_0$$

où chaque poids w_j modélise la contribution de la caractéristique x_j au log-ratio par exemple, si w_ix_i augmente de 0.69, $\frac{p(t=+1|x)}{p(t=-1|x)}$ est doublée car $\exp 0.69 = 2$).
On utilise la régression logistique quand :

- ✘ Les distributions de classes normales ont des matrices de covariance égales
- ✘ Le nombre de caractéristiques est important (trop d'effets croisés à prendre en compte)
- ✘ Seulement certaines caractéristiques sont pertinente et qu'il faut faire une sélection

Exemples typiques d'applications : text mining, analyse de données génétiques, etc.

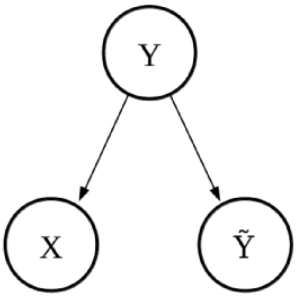
$$p(t=+1|x) = \frac{1}{1+\exp(-\mathbf{w^T x}-w_0)}$$



6.3 Outliers in regression and classification

Pour pouvoir tenir compte du bruit, on va créé un deuxième modèle. Par exemple, le simple modèle probabiliste ci-dessous :

$$P(\tilde{Y} = \tilde{y}|Y = y) = \begin{cases} 0.9 & \text{si } \tilde{y} = y, \text{ c-à-d si la labellisation est correcte} \\ 0.1 & \text{si } \tilde{y} \neq y, \text{ c-à-d si le label est incorrect} \end{cases}$$

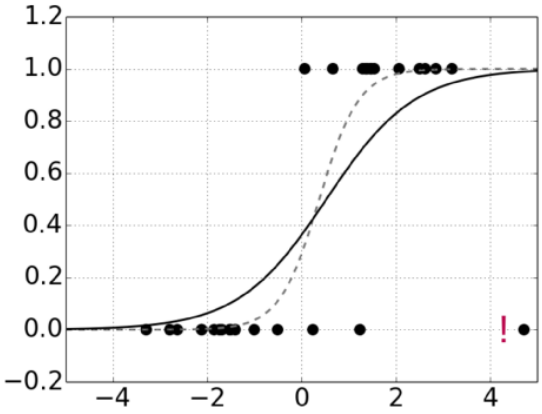


Le *label noise* peut provenir de plusieurs sources :

- Information insuffisante fournie à l'expert
- Erreurs dans l'étiquetage des experts lui-même
- Subjectivité de la tâche d'étiquetage
- Problèmes de communication/d'encodage

Il peut avoir des effets divers également :

- Diminuer les performances de classification
- Augmenter la complexité des modèles appris
- Constituer une menace pour des tâches (ex : la sélection des caractéristique)



En tenant compte des *outliers*, on tendra plutôt vers la courbe en pointillé.

7 Single and Multilayer Artificial Neural Networks

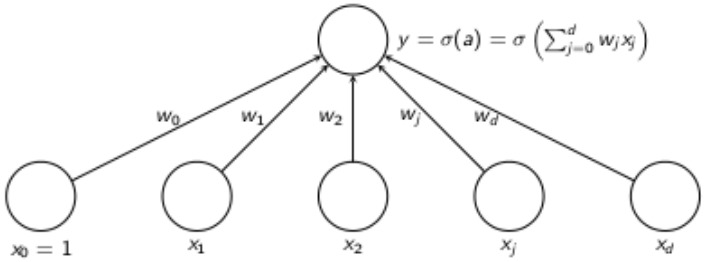
7.1 A bit of history

	Ordinateur numérique	Cerveau humain
Traitement de l'information	Séquentiel (sans threading et multi-cœur)	Parallèle ($10^1$1 neurones, 10^4 connexions)
Vitesse	Rapide ($\pm$ 0,3 ns)	Lent (min 1 ms)
Taille	Assez importante ($720m^2$ pour Tianhe-2)	Petit ($1.5kg$ et $1200cm^3$)
Énergie	Énergivore (24 MW pour Tianhe-2)	Économe (12 W, total 20%)
Résistance aux dommages physiques	Faible	Peu récupérer
Méthode d'apprentissage	Programmation soigneuse	Par expérience
Tâches optimisées	Effectuer des tâches basiques et du calcul	Décision, Contrôle, Vision

Définition d'un réseau de neurones artificiels (ANN)

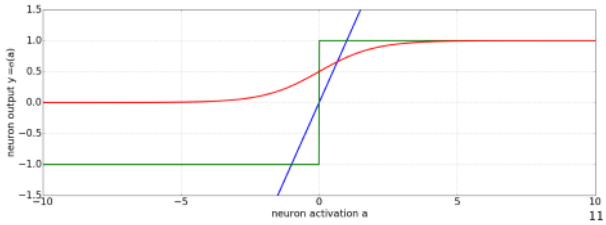
Un réseau de neurones artificiels est un logiciel d'architectures matérielles qui imite (vaguement) le comportement et les capacités de vrais réseaux de neurones biologiques, dans le but de construire de puissants outils de calcul pour résoudre des problèmes.

7.2 single-layer neural networks



Analogie avec les vrais réseaux de neurones :

- Les unités d'input x_j --→ neurones sensoriels (sauf x_0 pour simplifier la notation)
- Connexion --→ synapse de dendrite à axone (sauf le biais w_0)
- Poids --→ poids synaptique/force d'influence (positive ou négative)
- Activation --→ activation totale de l'axone résultant de la sommation



D'un point de vue biologique, on retrouve également un phénomène de saturation. Le modèle ReLU est une approche intermédiaire : il reprend le tracé vert mais avec un $2^{ème}$ segment oblique.

Simple perceptron :

- Modèle linéaire avec fonction d'activation d'identité : $\sigma(a) = a$
- Unité sigmoïde avec fonction d'activation sigmoïde : $\sigma(a) = \frac{1}{1+\exp(-a)}$
- Discriminant linéaire :

$$\sigma(a) = sign(a) = \begin{cases} +1 & \text{si } a > 0 \\ -1 & \text{sinon} \end{cases}$$

7.2.1 Simple adaptation rules

Pour les tâches de **classification**, on utilise une règle d'apprentissage par perceptron : on apprend les poids des unités seuillées (signe). Pour les tâches de classification binaires, on peut utiliser un simple discriminant ($\sigma(a) = sign(a)$) comme règle d'apprentissage du perceptron. Pour simplifier la notation, on ajoute un feature fictif $x_0 = 1$. La fonction y dépend de $\sum_{j=0}^{d} w_j x_j = \mathbf{w} \cdot \mathbf{x}$, c-à-d l'alignement (la similarité) de $\mathbf{w}$ et $\mathbf{x}$. w peut être compris comme "le" pattern qui sera détecté par apprentissage. A noter que si les deux ensembles se chevauchent, le taux d'erreurs sera positif.

Algorithm 5: Perceptron Learning Rule

Input: Le data-set $\mathcal{D} = \{(x_i, t_i)\}$
Output: Les poids qui séparent linéairement les deux classes

`/* astuce pour ne pas tenir compte du biais */`
Ajouter $x_0 = 1$ au début de chaque vecteur x
Initialiser les poids $\mathbf{w}_t = (w_0, w_1, ..., w_d)$ du perceptron de manière aléatoire

while *au moins une instance est mal classée* **do**
  for *chaque instance x_i de cible t_i du data-set* **do**
    Calculer la prédiction du perceptron $y_i = \text{sign } \mathbf{w}_t^T \mathbf{x}_i$
    if *l'instance est mal classifiée par le perceptron, c-à-d si $y_i \neq t_i$* **then**
      $mathbf w_{t+1} = mathbf w_t + \alpha_t(t_i - y_i)x_i$
    end if
  end for
end while

return $\mathbf{w}_t$

Pour les tâches de **régression**, on utilise la règle d'apprentissage Delta : on apprend les poids

d'une fonction d'activation σ (non-)linéaire et continue. Il en existe deux version : la version *"Batch"* qui prend tous les points en compte à chaque itération et la version *"Online"* qui met les points à jour l'un après l'autre.

Algorithm 6: Delta Rule (batch version)

Input: Le data-set $\mathcal{D} = \{(x_i, t_i)\}$
Output: Les poids optimaux pour la régression (non-)linéaire, tenant compte de la MSE

`/* astuce pour ne pas tenir compte du biais */`
Ajouter $x_0 = 1$ au début de chaque vecteur x
Initialiser les poids $\mathbf{w}_t = (w_0, w_1, ..., w_d)$ du perceptron de manière aléatoire

while *la MSE continue à changer de manière significative (ou autre critère de convergence)* **do**
  if *la fonction d'activation est linéaire* **then**
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \frac{\alpha}{n} \sum_{i=1}^{n} (t_i - y_i) x_i$
  else if *la fonction d'activation est sigmoïde* **then**
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \frac{\alpha}{n} \sum_{i=1}^{n} (t_i - y_i) y_i (1 - y_i) x_i$
  end if
end while

return $\mathbf{w}_t$

Pour trouver le minima de la MSE, il faudrait pouvoir dériver sa formule. Malheureusement, il n'existe pas de méthode "pseudo-inverse non-linéaire" et ce calcul n'est donc pas réalisable. Cependant, on peut en réaliser une méthode d'approximation itérative : la descente de gradient. dans la version *Batch*, on utilise la descente de gradient classique. L'idée est de mettre à jour les poids synaptiques dans la direction de la MSE la plus basse :

$$w_j \leftarrow w_j - \alpha \frac{\delta E}{\delta w_j} = w_j + \frac{\alpha}{n} \sum_{i=1}^{n} (t_i - \sigma(a_i)) \sigma'(a_i) x_j$$

Pour une fonction d'activation linéaire $\sigma(a) = a$, la règle de mise-à-jour devient :

$$w_j \leftarrow w_j + \frac{\alpha}{n} \sum_{i=1}^{n} (t_i - y_i) x_j$$

Et pour une fonction sigmoïde $\sigma(a) = 1/(1 + \exp(-a))$, la règle de mise-à-jour devient :

$$w_j \leftarrow w_j + \frac{\alpha}{n} \sum_{i=1}^{n} (t_i - y_i) y_i (1 - y_i) x_j$$

Toutes les instances doivent donc être connues avant la mise-à-jour des poids.
Dans la version *Online*, c'est un peu différent puisque l'on ne peut pas réutiliser l'ensemble des instances :

Algorithm 7: Delta Rule (batch version)

Input: Le data-set $\mathcal{D} = \{(x_i, t_i)\}$
Output: Les poids optimaux pour la régression (non-)linéaire, tenant compte de la MSE

`/* astuce pour ne pas tenir compte du biais */`
Ajouter $x_0 = 1$ au début de chaque vecteur x
Initialiser les poids $\mathbf{w}_t = (w_0, w_1, ..., w_d)$ du perceptron de manière aléatoire

while *la MSE continue à changer de manière significative (ou autre critère de convergence)* **do**
  if *la fonction d'activation est linéaire* **then**
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + (t_i - y_i) x_i$
  else if *la fonction d'activation est sigmoïde* **then**
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + (t_i - y_i) y_i (1 - y_i) x_i$
  end if
end while

return $\mathbf{w}_t$

Dans la version *Online*, on utilise la descente de gradient stochastique qui choisit de manière itérative une instance x_i et met à jour les poids en fonction du composant d'erreur $E_i = \frac{1}{2}(t_i - y_i)^2$:

$$w_j \leftarrow w_j - \alpha \frac{\delta E}{\delta w_j} = w_j + \frac{\alpha}{n} \sum_{i=1}^{n} (t_i - \sigma(a_i))\sigma'(a_i)x_j$$

Pour une fonction d'activation linéaire $\sigma(a) = a$, la règle de mise-à-jour devient :

$$w_j \leftarrow w_j + \alpha(t_i - y_i)x_j$$

Et pour une fonction sigmoïde $\sigma(a) = 1/(1 + \exp(-a))$, la règle de mise-à-jour devient :

$$w_j \leftarrow w_j + \alpha(t_i - y_i)y_i(1 - y_i)x_j$$

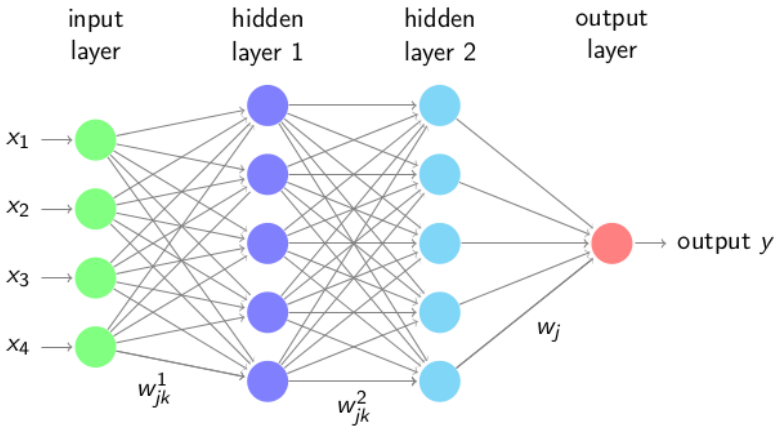
Cette version est beaucoup plus proche de la règle du Perceptron (excepté que $\sigma \neq sign$).

	Descente de gradient classique	Descente de gradient stochastique
Direction de chaque itération	Estimation de la MSE la plus faible	Chacune dans un sens différent
Vitesse	Chaque itération doit prendre toutes les données en compte	Une seule instance nécessaire à chaque itération (parfait pour les grand data-set ou les data streams)
Nombre d'itérations	Quelques dizaine, centaine ou milliers	Très grand, ne dépend pas de n

7.2.2 The first AI winter

Les réseaux à un seul neurone sont assez limités : ils ne peuvent pas résoudre les tâches de classification XOR et certaines tâches nécessitent une couche cachée de neurones entièrement connectés. C'est pourquoi entre 1969 et le milieu des années '80, on remarque une nette perte d'intérêt pour les réseau de neurones, jusqu'à l'avènement de la rétro-propagation (*back-propagation*).

7.3 Multi-layer neural networks



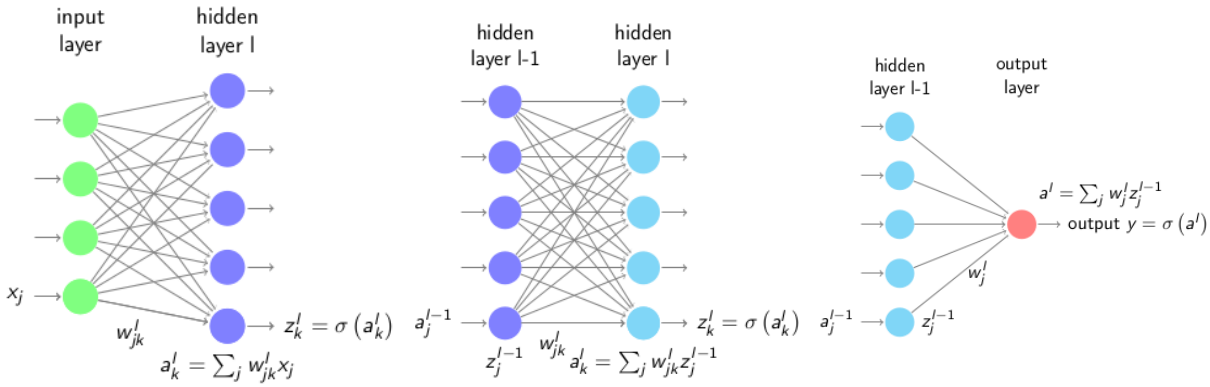
Les couches cachées augmentent l'expressivité des perceptrons. La découverte la retro-propagation permet de créer de nouveaux algorithme plus efficaces et de remettre les ANN sur le devant de la scène du machine learning.

En Deep learning, on utilise 30 à 40 couches cachées.

La complexité des Perceptrons multi-couches (MLP) dépend du nombre de couches cachées (chacune correspond à un "niveau de représentation") et du nombre de neurones cachés dans chaque couche cachée (tâche complexe ⇒ plus de neurones nécessaires).

Un MLP avec deux couches de neurones (dont une cachée) et une activation linéaire à la sortie est suffisant pour approximer les fonctions continues. En pratique cependant, l'utilisation de plus de deux couches peut être nécessaire pour traiter des images.

7.3.1 Error back-propagation



Propagation des activations neuronales dans le réseau pour l'instance x d'entraînement. Les indices j et k représentent les neurones quand l'indice l représente la couche considérée.

21-> 25/29

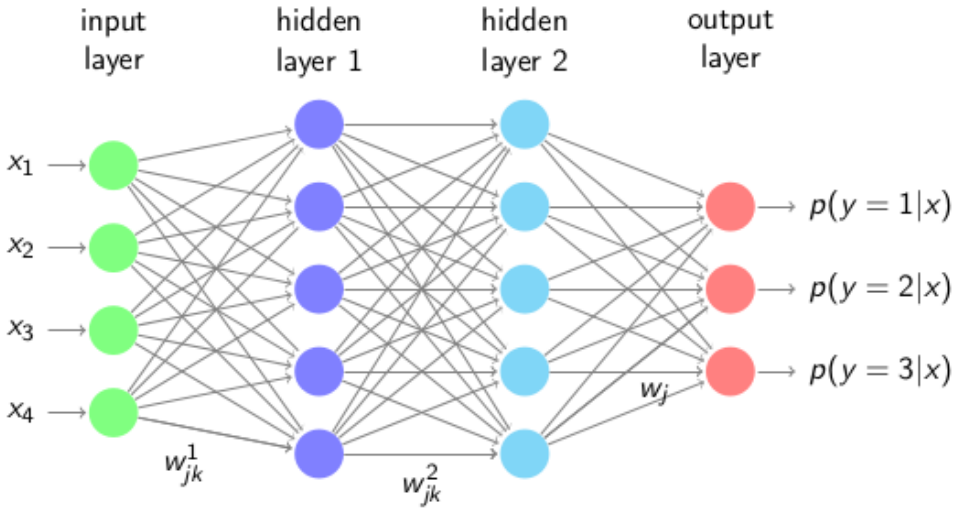
Avantages des MLP avec rétro-propagation :

- ✓ Pas si difficile à mettre en œuvre, une fois que les bases sont comprise
- ✓ Chaque étape est bon marché : seuls δ^l_k et $\sigma'(a^{l-1}_j)$ doivent être calculés

Limitations :

- ✘ Beaucoup d'étapes sont nécessaires pour converger (des milliers ou pire)
- ✘ La descente en gradient est susceptible de rester bloquée dans les minima locaux de la MSE
- ✘ La MSE a de nombreux optima locaux (en raison de la symétrie dans les MLP notamment)
- ✘ Il est très difficile de trouver un bon choix initial de poids

7.3.2 Classification with multi-layer perceptrons



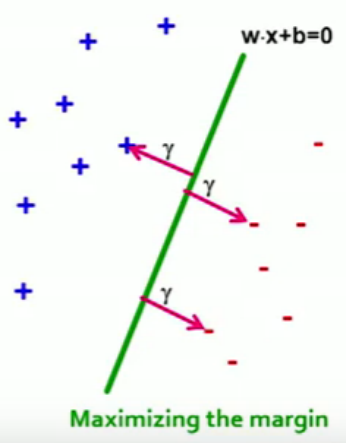
Chaque sortie k correspond à le probabilité conditionnelle de cette classe $k = \frac{1}{1+\exp(-a^l_k)}$.

7.4 The future of neural networks

Les réseaux de neurones artificiels subissent une "deuxième mort" dans les années 2000, suite à l'arrivée de nouveaux modèles. Depuis le milieu des années 90, les SVM apportent beaucoup d'avantages : elles sont plus faciles à optimiser car leur fonction objectif est convexe, pas de minimum local, pas de descente de gradient infinie, pas de réglage de régime α_t et leurs noyaux peuvent être utilisés pour apprendre à partir de données non numériques (par exemple, avec un graphe en entrée).
Cependant, les récente recherches en Deep learning ouvrent la voie pour une renaissance des réseaux de neurones. Déjà présent dans les années 80-90, le deep learning est aujourd'hui soutenu par les capacités hardware, le nombre de données grandissantes et de nouveaux algorithmes. Il utilise ses couches cachées pour effectuer l'extraction de caractéristiques et apprends les poids cachés de manière non supervisée. Contrairement aux réseaux de neurones traditionnels, chaque couche de deep learning possède une fonction différente des autres. Il apporte des résultats fructueux pour la vision par ordinateur, la reconnaissance vocale et la NLP (Natural Language Processing).

8 Support Vector Machines and Kernels

Contrairement aux réseaux de neurones, les SVM (*Support Vector Machines* ou séparateurs à vaste marge) ne permettent de construire qu'un seul modèle optimal, grâce à la convergence de w.L'idée est de trouver une "ligne" séparant les données avec la marge la plus grande possible. La distance (la marge γ) entre un pont et cette ligne définit notre niveau de confiance envers la prédiction émise sur cette instance. Cette idée semble admissible d'un point de vue de l'intuition, mais aussi théorique et pratique.



8.1 Maximum margin classifiers

La prédiction peut se calculer simplement comme :

$$f(x) = sign(\mathbf{w} \cdot x + b)$$

Et la confiance de la prédiction s'obtient de la façon suivante :

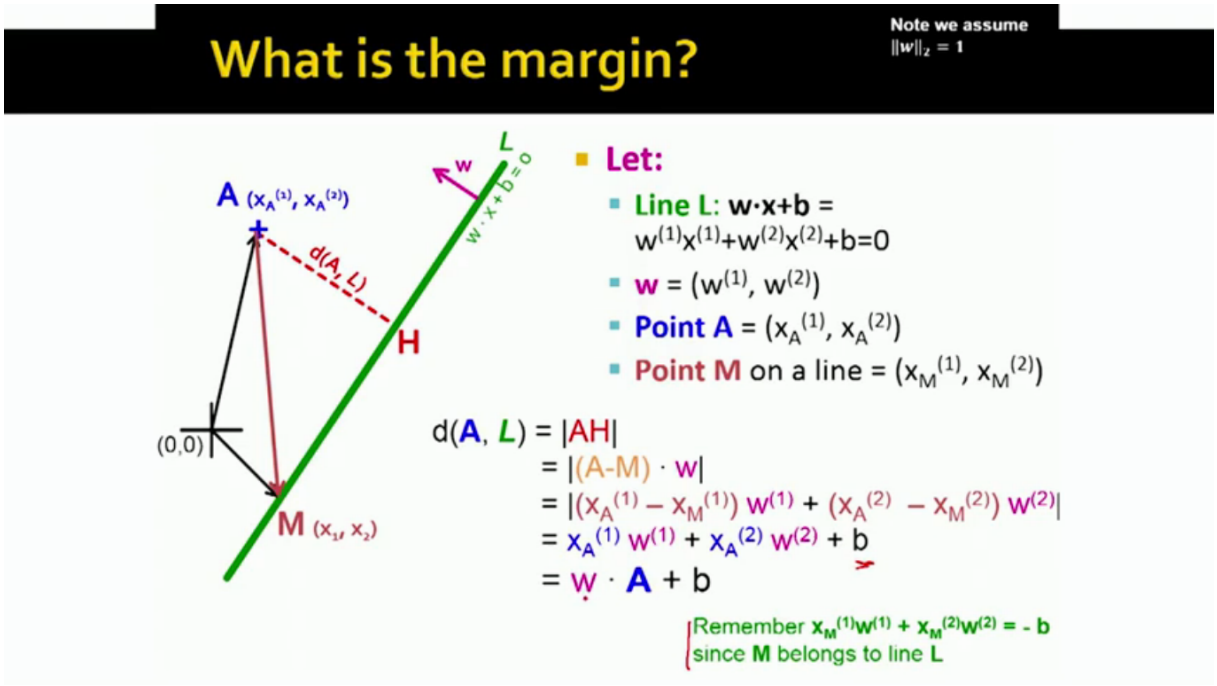
$$\gamma_i = (\mathbf{w} \cdot x_i + b) y_i$$

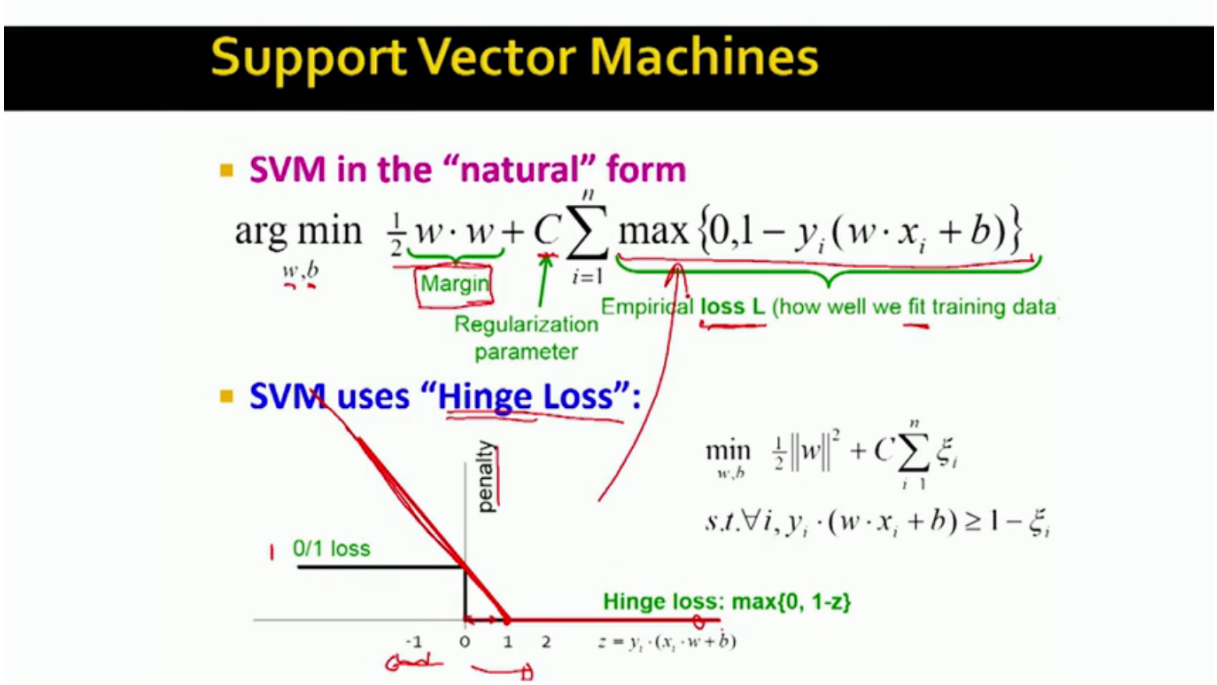
Notre objectif est donc de trouver ce γ tel que la plus petite marge est maximisée :

$$\max_{w,\gamma}$$
$$\text{tel que} \qquad \forall i, y_i(\mathbf{w} \cdot x_i + b) \geqslant \gamma$$

Pour déterminer la ligne de séparation, il est inutile de considérer tous les points : seule une minorité, appelée vecteurs de support suffit. Si l'on ignore les dégénérescences, il suffit de $d+1$ vecteurs de support pour déterminer la ligne d'un espace en d dimensions.
Remarque : $\mathbf{w}$ a été normalisé pour que seule sa direction soit prise en compte, et non sa longueur.
La SVM à forte contrainte se définit donc comme :

$$\min_{w} \frac{1}{2}||\mathbf{w}||^2$$
$$\text{tel que} \qquad \forall i, y_i(\mathbf{w} \cdot x_i + b) \geqslant 1$$





8.2 Linear SVMs for separable data

8.3 Linear SVMs for non-separable data

8.4 Kernels and similarity measures

3 -> 11/16

8.5 Kernelised SVMs for non-linear data

Forme duale des SVM kernelisés pour les données non séparables Pour les données non linéairement non séparables, la solution est donnée par :

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k(x_i, x_j)$$

$$\text{Avec} \qquad C \geqslant \alpha_i \geqslant 0 \qquad \text{et} \qquad \sum_{i=1}^{n} \alpha_i t_i = 0$$

Les prédictions peuvent se faire avec $f(x) = \mathbf{w} \cdot \mathbf{x} + b = \sum_{i=1}^{n} \alpha_i t_i k(x_i, x) + b$
La complexité de la SVM dépend également du paramètre de noyau, par exemple pour le noyau RBF :

$$k(x, z) = \exp(-\gamma ||x - z||^2) \in [0, 1]$$

Si le paramètre γ est petit, le focus est mis sur les grandes structures et s'il est grand sur les petits détails.

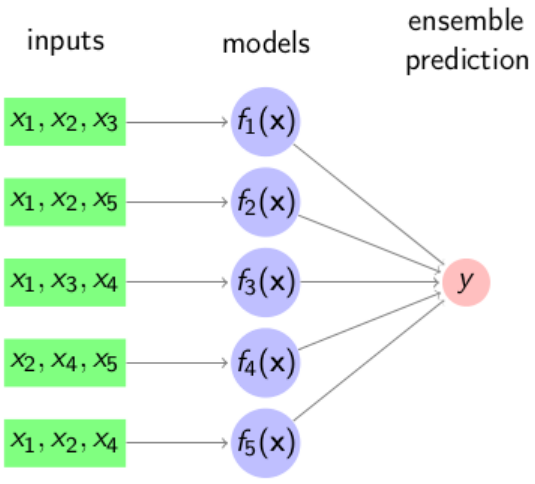
8.6 Going further with kernels

Si la complexité est contrôlée correctement à l'aide de techniques de régularisation, toutes les méthodes dont les instances apparaissent à travers des produits scalaires (régression linéaire et logistique, techniques de clustering, etc.) peuvent être "*kernelisées*".

9 Ensemble Methods : from Single Models to Ensemble of Models

9.1 Ensemble learning

Il existe des ensembles de modèles de machine learning pour la classification avec des faiblesses et capacités différentes. Cet ensemble pourrait construire un modèle plus fort que chacun des modèles le constituant pris individuellement. Cependant, le théorème NFL (*No free lunch*) nous dit qu'il n'existe pas de "super-algorithme" : si un algorithme A surpasse un algorithme B selon une certaine fonction de coût, alors il existe également une autre fonction de coût par laquelle B dépasse A.



Avantages et inconvénients des méthodes ensemblistes (fusion de classificateurs, système de classification multiple, etc.) :

- ✓ Les méthodes d'ensemble sont puissantes
- ✘ Elles peuvent facilement *overfit*
- ✘ Elles sont également affectées par le théorème NFL
- ✘ Certaines combinaisons n'ont pas de sens (exemple : une combinaison linéaire de modèles linéaires, un ensemble de réseaux de neurones à sortie linéaire, etc.)

Les modèles ensemblistes n'ont de sens que si les modèles assemblés sont différents. La stratégie la plus simple consiste à utiliser différents ensembles de données d'entraînement (ou différentes parties du même set) pour obtenir de la variabilité. Il est aussi possible de jouer sur l'architecture

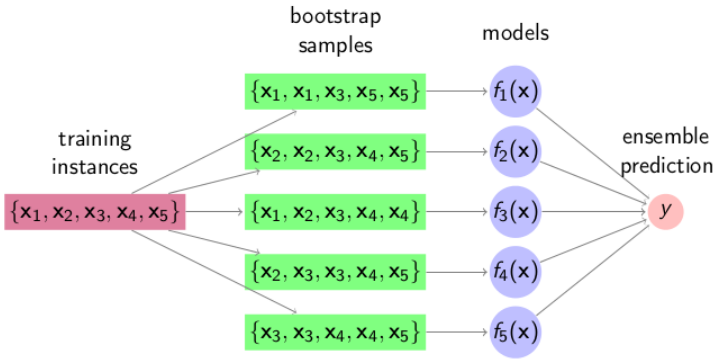
du classificateur lui-même en choisissant différents modèles (linéaires, NN, DT, k-NN, etc.) ou d'utiliser différentes initialisations pour des fonctions objectives non-convexes. Chaque modèle peut être très faible individuellement (faible complexité, haut taux d'erreur).

9.2 Bootstrap aggregating

Pour obtenir $B >> 1$ training sets similaires et pourtant différents :

- Dessiner n fois avec le remplacement de l'ensemble d'entraînement original de taille n
- Les instances peuvent apparaître plusieurs fois (ou pas) dans l'échantillon bootstrap

En moyenne, 63% de patterns-sont uniques dans chaque échantillon bootstrap (pour un grand n). Les 36% des instances non-sélectionnées peuvent être utilisées pour effectuer des tests.
Un classifcateur est entraîné pour chaque échantillon bootstrap. La prédiction correspond à la règle de majorité appliquée aux prédictions B.



9.3 Random forests

Une forêt aléatoire est un ensemble très grand d'arbres de décision (un millier ?) avec une haute variabilité (caractéristique aléatoires et bagging). Un arbre de décision est entraîné grâce à un échantillon bootstrap et à chaque nœud de décision, la meilleure caractéristique est choisie parmi un sous-ensemble aléatoire de caractéristiques. La randomisation de ce choix conduit à une grande diversité dans les arbres.
Avantages des Random Forests :

- ✓ Très bonnes performances sur de grands ensembles de données
- ✓ Facile à paralléliser (utile si beaucoup de processeurs à disposition)
- ✓ Peut tirer parti des techniques avancées comme les arbres Hoeffding

9.4 Adaptive boosting

AdaBoost est une construction itérative d'un ensemble de modèles très faibles (à peine mieux que random) qui sont pondérés par une coefficient α_i déterminé une fois pour toute. A chaque itération, le modèle f_i apprend sur une version allégée du data-set. Les poids des instances précédemment mal classées augmentent ensuite (et inversement), ce qui oblige les classificateurs successifs à se concentrer dessus. ci-contre pour les poids d'instances préalablement classées correctement

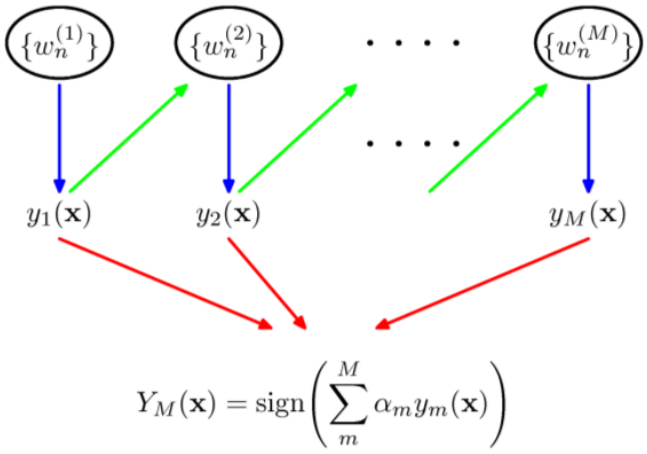
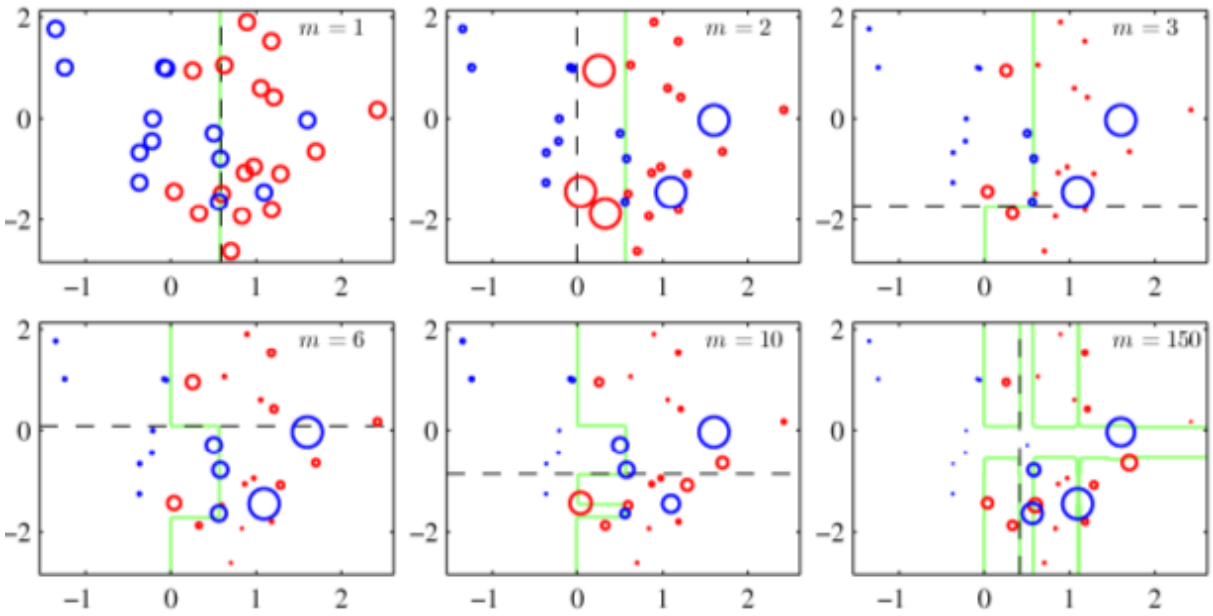
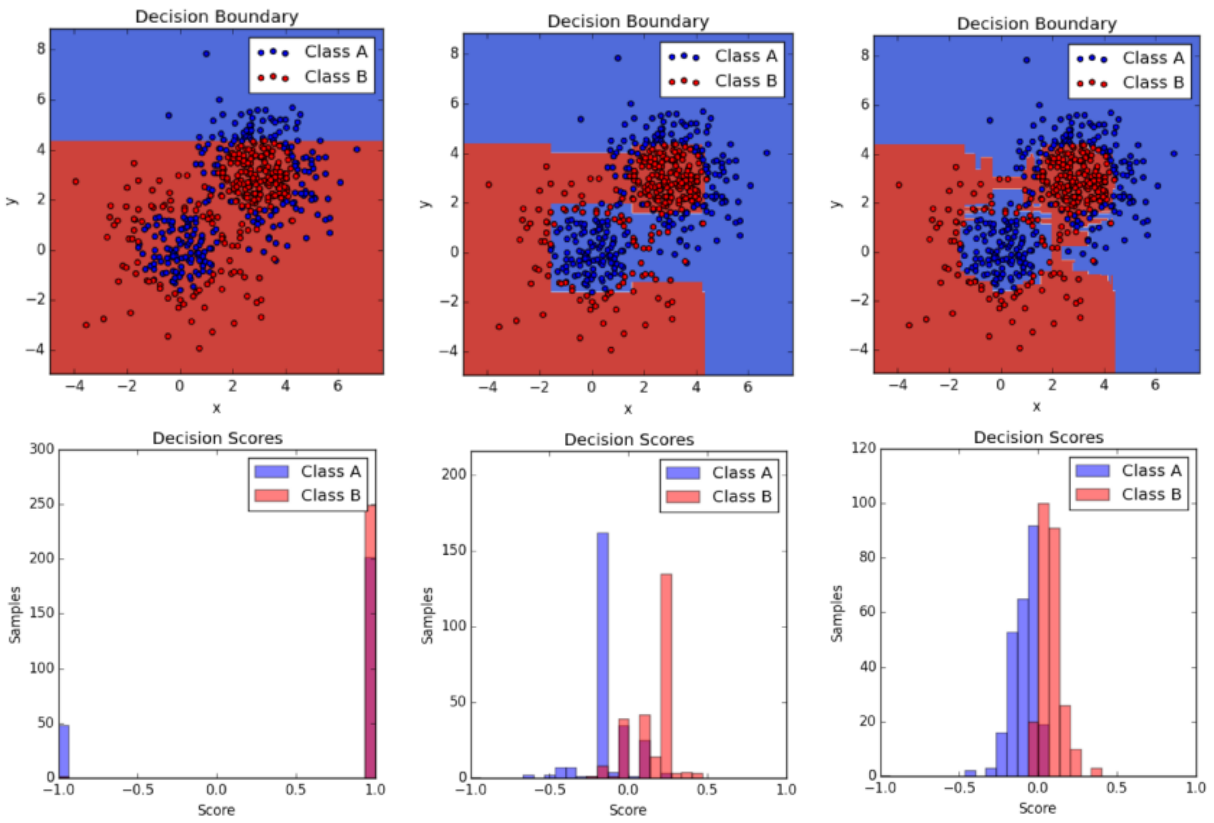


FIGURE 9.1 – Intuition Behind Adaptive Boosting



Single decision stump | Ensemble of 20 decision stumps | Ensemble of 1000 decision stumps



Troisième partie
Unsupervised learning

10 Introduction to Unsupervised Learning

10.1 Generalities about unsupervised learning

Definition du Machine learning non supervisé :

Recherche de patterns, d'une structure de données en limitant l'interaction humaine pour pouvoir faciliter une future analyse et gagner de l'information sur des données existantes. En pratique, il fonctionne par essai/erreur et nécessite donc un feedback humain. Il regroupe des tâches de clustering, estimation de densité et visualisation.

Attention ! Il ne s'agit pas d'apprendre sans fonction objectif, mais sans cible. Voici quelques exemples de fonctions objectifs :

- Clustering : Minimiser les distances intra-clusters
- Estimation de densité : maximiser la vraisemblance des données
- Visualisation : Minimiser la perte d'information après projection

10.2 Comparison with supervised learning

	Apprentissage supervisé	Apprentissage non supervisé
But	Spécifié par la cible et le contexte (tâche +/- objective)	Pas de vérité absolue (souvent définie pas l'utilisateur et subjective)
Fonction objectif	Peu (substituts)	Beaucoup (problème mal défini)
Objectif	Prédiction de la cible	Nouvelle connaissance ou étape intermédiaire d'apprentissage
Feedback	Fournie par le "apprentisseur" sous forme de paire "instance-cible"	Interactif (contraintes, préférences, validation de l'utilisateur)
Sélection du modèle	Basée sur une validation	Requiert un feedback humain ou des étapes ultérieures d'apprentissage
Test du modèle	basée sur des tests	Requiert un feedback humain ou des étapes ultérieures d'apprentissage
Interprétabilité	Pas toujours nécessaire (l'objectif est de prédire une cible)	Critique pour la découverte de connaissances & le feedback

11 Clustering

11.1 Problem statement

Definition du Clustering :

Groupement d'individus d'une population multidimensionnelle qui permet de définir une structure dans un ensemble de données. Un cluster doit contenir des données similaires entre elle, mais divergentes de celles des autres clusters.

Le problème du clustering, c'est qu'il n'existe aucune vérité universelle quant au nombre de cluster ou à leur nature.

11.2 The k-means algorithm

Algorithme K-Means :

Procédure itérative permettant de retrouver les k clusters les plus représentatifs des données à partir de k centroïdes ou prototypes. Cet algorithme possède de nombreuses extensions : fuzzy k-means, k-medoids, ...

On note $\mathcal{C} = \{z_j\}$ le dictionnaire des centroïdes et $y(x)$ l'index du centroïde auquel est assigné l'instance x. $d(x, z_{y(x)})$ est donc la distance entre le cluster et son prototype.
Définition de la fonction objectif qui permet de minimiser l'erreur de reconstruction grâce au

dictionnaire des centroïdes :

$$\mathcal{J}(\mathcal{C}) = \int_x p(x) d(x, z_{y(x)})^2 dx$$

Cette fonction est malheureusement impossible à calculer en pratique. On l'approxime donc par la fonction suivante, utilisant l'erreur d'entraînement :

$$\mathcal{J}(\mathcal{C}) = \frac{1}{n} \sum_{i=1}^{n} d(x_i, z_{y(x_i)})^2 dx$$

$$x_i \longrightarrow \boxed{\text{encodeur}} \longrightarrow \text{index } y(x_i) \longrightarrow \boxed{\text{décodeur}} \longrightarrow \text{ centroïde } z_{y(x)}$$

Cette vue "encodeur-décodeur" tente de minimiser l'erreur de reconstruction d'entraînement mais ne possède pas de solution analytique. Le k-means est un algorithme itératif qui part d'un *codebook* (dictionnaire) de départ et l'améliore par "*greedy*" (gloutonne) approche.
L'algorithme se découpe en deux étapes dont l'objectif est de minimiser l'erreur de reconstruction :

1. **Étape d'encodage :** Connaissant le décodeur (*codebook*), quelle est les meilleur encodeur/la meilleure assignation ?

   $$x_i \longrightarrow \boxed{\textbf{encodeur}} \longrightarrow \text{index } y(x_i) \longrightarrow \boxed{\text{décodeur}} \longrightarrow \text{ centroïde } z_{y(x)}$$

   Solution : assigner x_i au centroïde $z_{y(x_i)}$ le plus proche :

   $$\mathcal{J}(\mathcal{C}) = \frac{1}{n} \sum_{i=1}^{n} d(x_i, z_{y(x_i)})^2 dx$$
   $$y(x_i) = \arg \min_{j=1\ldots k} d(x_i, z_j)$$

2. **Étape de décodage :** Connaissant l'encodeur (assignation), quelle est les meilleur décodeur/*codebook* ?

   $$x_i \longrightarrow \boxed{\text{encodeur}} \longrightarrow \text{index } y(x_i) \longrightarrow \boxed{\textbf{décodeur}} \longrightarrow \text{ centroïde } z_{y(x)}$$

   Solution : déplacer le centroïde z_j jusqu'au centre de gravité du cluster $\mathcal{C}_j$:

   $$\mathcal{J}(\mathcal{C}) = \frac{1}{n} \sum_{i=1}^{n} d(x, z_{y(x)})^2 dx$$
   $$z_j = \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} x_i$$

Le critère de terminaison peut être soit un certain nombre d'itérations, soit un changement des valeurs successives du codebook $\mathcal{J}(\mathcal{C})$ Avantages et inconvénients du k-mens :

- ✓ Simple à comprendre et implémenter
- ✓ Très rapide (les seules opérations sont des calculs de distances, des minimisations et des moyennes)
- ✓ Converge seulement vers un minimum local de $\mathcal{J}(\mathcal{C})$, et dépend donc de l'initialisation
- ✓ En pratique, une centaine de passes sont nécessaires pour obtenir des résultats cohérents
- ✓ Assez rigide : chaque instance n'appartient qu'à un seul cluster (solution : variante "fuzzy")

Algorithm 8: Algorithme k-means

Input: Le dataset $\mathcal{D} = \{x_i\}$ et le nombre k de clusters
Output: Le codebook $\mathcal{C} = \{z_j\}$ et la fonction d'assignation y

while *critère de terminaison non rencontré* **do**
  // étape d'encodage/d'assignation **for** *chaque instance* x_i **do**
    $y(x_i) = \arg\min_{j=1...k} d(x_i, z_j)$
  end for
  // étape de décodage et de mise-à-jour du codebook
  for *chaque centroïde* z_j **do**
    $z_j = \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} x_i$
  end for
end while

11.3 Choosing the number of clusters

Choisir le bon nombre de cluster est difficile et il n'existe pas de méthode universelle. La méthode "du coude" permet une approximation, mais parfois elle donne une valeur trop élevée, voire il est impossible de déterminer exactement le coude. Dans le cas d minima locaux,même si le bon nombre de prototypes a été choisi, il est possible de ne pas trouver les "bons" clusters car, lors de la première itération, ceux-ci ont été mal répartis. Le nombre de clusters doit être définit par un être humain, car il dépend de l'objectif visé.Le clustering est souvent utilisé pour explorer des données, comme étape préliminaire avant un autre traitement ou encore pour valider un résultat obtenu par d'autres méthodes.

11.4 The DBSCAN algorithm

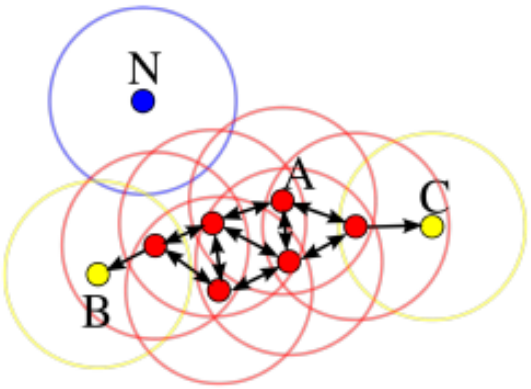
Algorithme Density-Based Spatial Clustering of Applications with Noise :

Procédure de recherche des régions de haute densité et l'une des techniques de clustering les plus utilisées. Il a l'avantage de ne pas nécessité la définition d'un nombre de clusters, mais nécessite d'autres méta-paramètres. Il permet égaiement de mettre en évidence des *outliers* (points qui n'appartiennent à aucun cluster). Il s'agit plus d'une technique de data mining que de machine learning puisqu'il se déduit directement des données.

- L'ε-voisinage de x = l'ensemble des points x_i tels que $d(x, x_i) \leqslant \varepsilon$
- x = point de base (*core point*) si son ε-voisinage contient au moins $n_{min} \neq$ points
- $x_i \in \varepsilon$-voisinage du point de base x est directement accessible depuis x
- x_n est atteignable depuis x_i s'il existe un chemin $x_1 \rightarrow x_2 \rightarrow ... \rightarrow x_n$ où $x_i + 1$ est directement joignable depuis x_i
- Les points non-joignables sont des *outliers*
- Le cluster auquel un point de base appartient correspond à l'ensemble des points atteignables.
- Les points qui ne sont pas des points de bases $\approx$ limite du cluster (ils ne peuvent pas atteindre d'autres points)
- L'accessibilité est transitive mais non symétrique (excepté pour les points de base)
- Les points qui ne sont pas des points de bases mais appartiennent au même cluster ne sont pas atteignable l'un de l'autre
- x_i et x_j sont connectés si $\exists x_k$ depuis lequel x_i et x_j sont atteignables
- Tout point atteignable depuis un point du cluster appartient au cluster

Hypothèses d'apprentissages (biais) :

- Les régions à haute densité ont des densités similaires(ε et n_{min} sont globaux)
- Les clusters ne se chevauchent pas trop (séparées par des régions à faible densité)



Les core points sont en rouge.

- tous les points du cluster sont mutuellement connectés

Limites :

- Les résultats dépendent de ε (difficile à estimer) et n_{min} (dépendant de la taille du data-set)
- Algorithme pas entièrement déterministe (points non basiques)
- Problèmes si grandes différences de densité entre les clusters
- Les clusters se chevauchants sont susceptibles d'être fusionnés
- Pas de points représentatifs ⇒ pas d'interprétation

12 Density Estimation

12.1 Problem statement

Definition de l'estimation de densité :

Recherche de la probabilité $p(x)$ d'observer x étant donné un data-set $\mathcal{D} = \{x_i\}$. Cette méthode permet de résumer la distribution des données et de résoudre les tâches qui y sont liées (par exemple, la détection d'outliers).

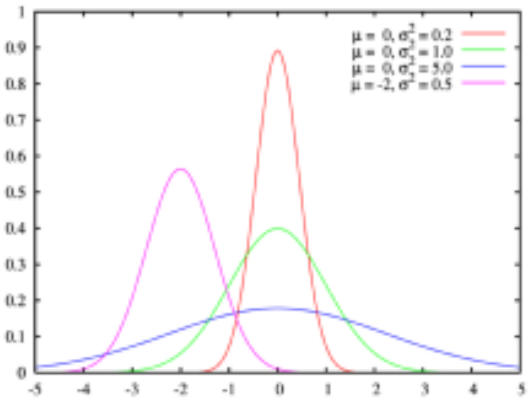
Il existe pour ce faire différents types d'estimateurs :

- Estimateurs paramétriques : méthode qui propose de très bon résultats, mais qui nécessite que les données appartiennent à une famille de distribution paramétrique connue. (exemple : Gaussienne, Gamma, Weilbull, etc.).
- Estimateurs non-paramétriques : nombres de paramètres proportionnels au nombre d'instances ; méthode très flexible, mais ouvert à l'overfitting.
- Estimateurs semi-paramétriques : compromis entre les deux méthodes et dont la complexité (le nombre de paramètre) est fixé par validation de l'utilisateur.

12.2 Parametric estimators

La distribution Gaussienne est souvent une bonne approximation pour des données telles au la taille, le poids, des mesures de bruit, etc. Pour représenter une somme des variables aléatoires, on utilise généralement une distribution Gaussienne (théorème central limite).

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$



Définissons maintenant notre fonction objectif d'un point de vue d'une solution générique. La log-vraisemblance de l'ensemble des données $\mathcal{D} = \{x_i\}$ se définit comme :

$$\begin{aligned}\mathcal{L}(x_1, ...x_n|\theta) &= \log p(x_1, ...x_n|\theta) \\ &= \log \prod_{i=1}^{n} p(x_i|\theta) \\ &= \sum_{i=1}^{n} \log p(x_i|\theta)\end{aligned}$$

Pour la maximiser, on utilise l'approche suivante :

$$\hat{\theta} = \arg\max \theta \mathcal{L}(x_1, ...x_n|\theta) = \arg\max \theta \sum_{i=1}^{n} \log p(x_i|\theta)$$

Si l'on se contente d'une distribution Gaussienne, nous obtenons cette expression de la log-vraisemblance :

$$\begin{aligned}\mathcal{L}(x_1, ...x_n|\mu,\sigma) &= \sum_{i=1}^{n} \log \mathcal{N}(x_i|\mu,\sigma^2) \\ &= \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \\ &= \frac{1}{n}\log\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2\end{aligned}$$

Et l'approche maximale nous donne :

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad\qquad \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2$$

Avantages des estimateurs de densité de noyau :

- ✓ Facile à utiliser et à comprendre
- ✓ Standard en statistique
- ✓ Peu de paramètres intuitifs, faible coût de calcul
- ✓ Bonne solution pour les petits ensembles de données ou si on possède une connaissance antérieure

Limitations :

- ✘ Uni-modal, pas très flexible
- ✘ Le nombre de paramètres $= d + \frac{d(d+1)}{2} = \mathcal{O}(d^2)$
- ✘ Peut *overfit* de petits ensembles de données de grande dimension

Ces conclusions sont similaires pour la plupart des estimateurs paramétriques (non analytiques).

12.3 Non-parametric estimators

L'idée de base est de placer un noyau sur chaque point de données et d'additionner les contributions :

$$p(x) = \frac{1}{n}\sum_{i=1}^{n} k(x|x_i, \theta)$$

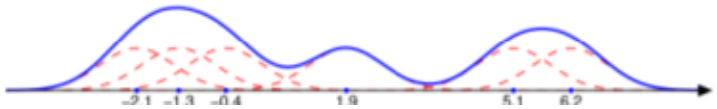
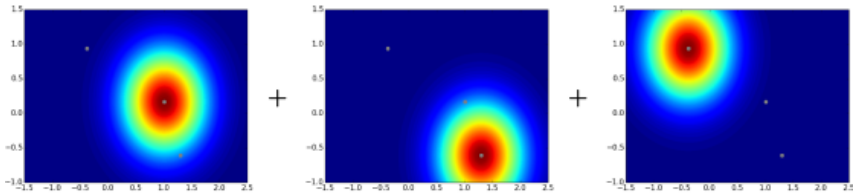
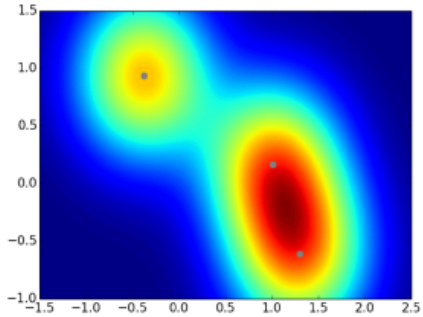


FIGURE 12.1 – Estimateur de densité : exemple de noyaux Gaussien



$$\begin{aligned} p(\mathbf{x}) &= \frac{1}{n}\sum_{i=1}^{n} k(\mathbf{x}|\mathbf{x}_i, \boldsymbol{\theta}) \\ &= \frac{1}{n}\sum_{i=1}^{n} \mathcal{N}(\mathbf{x}|\mathbf{x}_i, \sigma^2 \mathbf{I}_d) \\ &= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{(\sqrt{2\pi}\sigma)^d} e^{-\frac{\|\mathbf{x}_i - \mu\|_2^2}{2\sigma^2}} \end{aligned}$$



Deux choix de conception sont nécessaires :

- Le choix du type de noyau, le choix le plus courant étant $k(x|x_i, \theta) = \mathcal{N}(x|x_i, \sigma^2 \mathbf{I}_d)$
- Le choix du paramètre du noyau, par exemple, pour un noyau Gaussien, la largeur du kernel sera σ

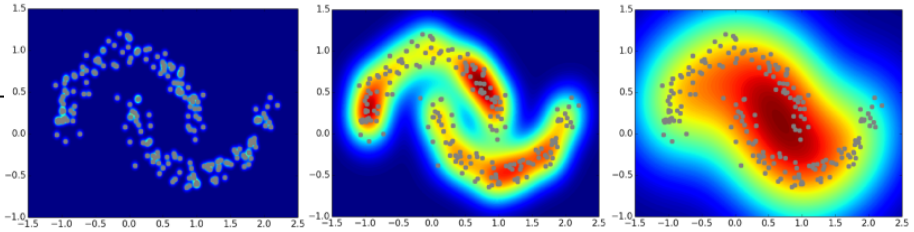
La Gaussienne a pour avantage de ne jamais tomber à zéro. La largeur σ du noyau peut être estimée par validation croisée (*"leave-one-out"*) :

$$p(x) = \frac{1}{n}\sum_{i=1}^{n} k(x|x_i, \theta) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{(\sqrt{2\pi}\sigma)^d} \exp^{-\frac{||x_i - \mu||_2^2}{2\sigma^2}}$$

Résultat avec simple validation (200/100)

σ	L_{train}	$\hat{L}_{\text{gen}}$
0.03	0.24	−2.72
0.20	−1.36	−1.44
0.50	−1.99	−2.00

$\sigma = 0,03$ $\sigma = 0,20$ $\sigma = 0,50$



Avantages des estimateurs de densité de noyau :

- ✓ Facile à utiliser et à comprendre
- ✓ Standard en analyse de données
- ✓ Pas de paramètre, très flexible et seulement un méta-paramètre
- ✓ Bonne solution si nous ne possédons pas de connaissances antérieures

Limitations :

- ✘ Coût de calcul élevé : exigences en temps et espace = $\mathcal{O}(n \times d)$ car on doit enregistrer toutes les données
- ✘ Le méta-paramètre doit être réglé, ce qui requiert du temps ($\mathcal{O}(n^2 \times d)$)
- ✘ Overfit facile, il ne convient donc pas pour les ensembles de données en haute dimension

Ces estimateurs se basent sur de nombreuses notions de physique.

12.4 Semi-parametric estimators

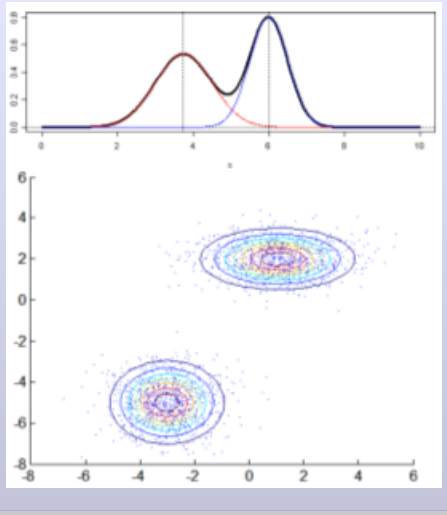
Les estimateurs semi-paramétriques permettent de réguler la complexité de manière à créer des modèles plus complexe et nécessitant moins de connaissances antérieures que les modèles paramétriques, mais moins complexes et moins *"overfit"* que les modèles non-paramétriques. Par exemple, le *"mixture model"* est une somme pondérée de k estimateurs paramétriques, qui déterminent sa complexité. Les paramètres et le poids de chaque composant doivent être appris.

Multivariate Gaussian mixture model (GMM)

$$p(x) = \sum_{j=1}^{k} \pi_j \mathcal{N}(x|\mu_j, \Sigma_j)$$

Avec les paramètres :

- π_j = probabilité antérieure d'appartenir au j^{ime} composant
- μ_j, Σ_j les paramètres du j^{ime} composant



Difficultés d'implémentation :

- Estimation des $k + kd + k\frac{d(d+1)}{2}$ paramètres
- Choix du nombre k de paramètres Gaussiens (méta-paramètre)

Pour estimer les paramètres du GMM, il faudrait maximiser la fonction suivante :

$$\mathcal{L}(x_1, ..., x_n|\theta) = \sum_{i=1}^{n} \log \left(\sum_{j=1}^{k} \pi_j p(x_i|\mu_j, \Sigma_j) \right)$$

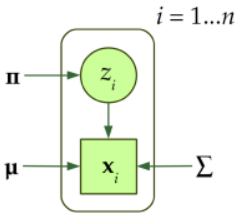
avec l'ensemble des paramètres $\theta = (\pi_1, ..., \pi_k, \mu_i, ..., \mu_k, \Sigma_1, ...\Sigma_k)$.
Cette fonction étant non-convexe, elle ne possède cependant pas de solution en forme fermée. Une alternative est d'utiliser l'algorithme de maximisation des attentes (EM). Cet algorithme permet de comparer le GMM à une version plus souple du k-means, dans laquelle une instance peut appartenir à deux clusters en même temps. Il se découpe en deux étapes itératives :

1. Calcul de l'appartenance de chaque instance aux k composants
2. Mise-à-jour des paramètres de chaque composant avec les instances correspondantes (qui ont une appartenance importante à ce composant)

Avantages du GMM :

- ✓ Modèles de distributions multimodales
- ✓ Flexible ($k + kd + k\frac{d(d+1)}{2}$ paramètres)
- ✓ Interprétation des composants comme des clusters ($\approx$ *"soft k-means"*)

FIGURE 12.3 – Vue générative du GMM et de ses variables cachées



1. Choix du composant $z_i = j$ qui générera x_i avec $p(z_i = j) = \pi_j$
2. Génération de x_i avec une moyenne μ_j et une matrice de covariance Σ_j

✓ semi-paramétrique ($\mathcal{O}(k \times d^2)$) → compromis entre les modèles paramétriques ($\mathcal{O}(d^2)$) et non-paramétriques ($\mathcal{O}(n^2 \times d)$) du point de vue de la complexité temporelle

Limitations :

✘ Trop flexible ⇒ matrices de covariance contraintes (diagonale, partagée, etc.)

✘ EM converge vers les minima locaux ⇒ plusieurs redémarrages sont nécessaires

✘ EM peut "crash" (matrices de covariance mal conditionnées) ⇒ utiliser des astuces

✘ Coût de calcul de la procédure EM

✘ Choix du nombre de composants

13 Visualising High-dimensional Data

13.1 Motivation and definition

Definition de la visualisation :

Méthode pour visualiser une large quantité de données en grande dimension, en minimisant la perte d'information.

La visualisation pose énormément de questions, entre autre sur la validation qui est liée à la psychologie de l'utilisateur. D'un point de vue du machine learning, l'objectif est de projeter en 2 (ou 3) dimensions pour pouvoir afficher le résultat sur un écran d'ordinateur. Ces projections doivent être validées par des critères objectifs et la perte d'information peut concerner la variance, mais pas le voisinage.

13.2 Stochastic neighbour embedding

Algorithme t-SNE :

Représentation ξ_i conservant (presque) le même voisinage que x_i. Il s'agit d'une approche non-paramétrique et non-linéaire.

La difficulté réside dans la définition de voisinage, de la comparaison de ces voisinage, de la définition d'une fonction objectif et de son calcul (pour pouvoir l'optimiser).

Voisinage dans l'espace original : La probabilité conditionnelle que x_i choisissent x_j comme voisin peut se définir comme :

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

Plus cette probabilité est grande, plus x_j est proche de x_i et inversement.

Voisinage dans l'espace de projection : La probabilité conditionnelle que ξ_i choisissent ξ_j comme voisin peut se définir comme :

$$q_{j|i} = \frac{exp(-||\xi_i - \xi_j||^2)}{\sum_{k \neq i} exp(-||\xi_i - \xi_k||^2)}$$

Cette définition est donc très semblable, excepté que l'on fixe arbitrairement $2\sigma_i^2 = 1$ et que l'on travaille avec les ξ_i et non plus les x_i. Pour pouvoir calculer ces probabilité, nous avons besoin de voisinages similaires pour ξ_i et x_i, c'est à dire que $p_{j|i}$ et $q_{j|i}$ et une distribution semblable. L'algorithme t-SNE choisit le ξ_i qui maximise la similarité.

La dissimilarité de Kullback-Leibler : La dissimilarité des distribution de de p et de q peut se calculer comme :

$$D_{KL}(p||q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \geqslant 0$$

Avec p la distribution de référence (la "vraie" distribution) de x. Cette valeur est donc très grande si q est très différent de p (et inversement).

Quatrième partie
Probabilistic learning

14 Probabilistic Models : Maximum Likelihood and Bayesian Approaches

14.1 Probabilistic learning

Définition des modèles probabilistes

Un modèle probabiliste est livré avec une fonction de vraisemblance $p(\mathcal{D}|\theta)$ qui donne la probabilité d'observer un jeu de données $\mathcal{D}$ pour chaque ensemble possible de paramètres du modèle θ, c-à-d qui estime comment θ est bon pour expliquer $\mathcal{D}$. La fonction ´ $p(\mathcal{D}|\theta)$ de log-vraisemblance est souvent utilisé comme fonction objectif.

Pour un data-set $\mathcal{D} = \{x_i\}$ (de variables indépendantes et identiquement distribuées), $\log p(\mathcal{D}|\theta)$ se factorise en

$$\log p(x_1, ..., x_n|\theta) = \log \prod_{i=1}^{n} p(x_i|\theta) = \sum_{i=1}^{n} \log p(x_i|\theta)$$

et comme beaucoup de modèles appartiennent à la famille des exponentielle, ils sont facilement optimisables. Avantages des modèles probabilistes :

- ✓ Permet d'intégrer facilement les connaissances antérieures et d'ajuster les modèles
- ✓ Donne un niveau de confiance pour la prédiction (par exemple, l'appartenance à une classe)
- ✓ Détecte les observations anormales (*outliers*) avec une probabilité très faible

Si $p(\theta)$ est supposé uniforme, la vraisemblance maximum peut inférer que $\hat{\theta} = \arg\max p(\theta|\mathcal{D}) = \arg\max \mathcal{D}|\theta$. Si $p(\theta)$ n'est pas uniforme, on peut utiliser la solution MAP $\hat{\theta} = \arg\max p(\theta|\mathcal{D})$.

Théorème d'inférence de Bayes

$$\underbrace{p(\boldsymbol{\theta}|\mathcal{D})}_{\text{posterior}} = \frac{\overbrace{p(\mathcal{D}|\boldsymbol{\theta})}^{\text{likelihood}} \overbrace{p(\boldsymbol{\theta})}^{\text{prior}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}$$

- prior $p(\boldsymbol{\theta})$ = what kind of parameters $\boldsymbol{\theta}$ you expect to get
- likelihood $p(\mathcal{D}|\boldsymbol{\theta})$ = how well data are explained by parameters $\boldsymbol{\theta}$
- evidence $p(\mathcal{D})$ = marginal likelihood (= constant w.r.t. parameters $\boldsymbol{\theta}$)
- posterior $p(\boldsymbol{\theta}|\mathcal{D})$ = how well parameters $\boldsymbol{\theta}$ are supported by data

Les hypothèses antérieures se déduisent de la nature-même des données, alors que les postérieures sont émises après. Pour de petits échantillon, notre connaissance des données est également très importante.

L'inférence MAP peut être interprétée comme la vraisemblance maximale avec un terme de pénalité $\log p(\theta)$ car :

$$\underbrace{\log p(\boldsymbol{\theta}|\mathcal{D})}_{\text{log-posterior}} = \underbrace{\log p(\mathcal{D}|\boldsymbol{\theta})}_{\text{log-likelihood}} + \underbrace{\log p(\boldsymbol{\theta})}_{\text{log-prior}} - \underbrace{\log p(\mathcal{D})}_{\text{log-evidence}}$$

L'inférence avec estimation ponctuelle de θ (solutions ML et MAP) se divise en deux étapes :

- L'étape d'inférence : choisit la meilleure estimation de θ
- L'étape de prédiction : prend des décisions à l'aide des probabilités $p(x|\theta)$

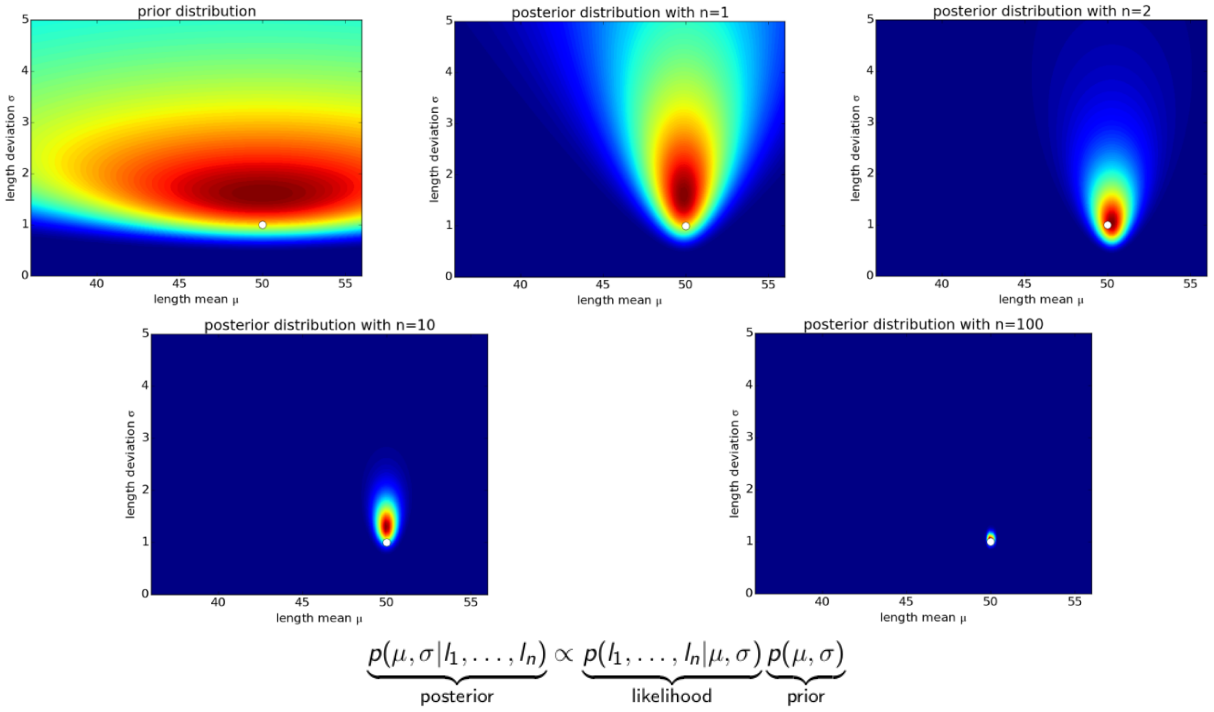
S'il existe plusieurs solutions ML/MAP équivalente, une seule est utilisée, ou la même chose mais avec des ensembles de paramètres légèrement moins bons que la solution ML/MAP. L'inférence "*Full Bayes*" calcule la distribution postérieur $p(\theta|\mathcal{D})$ de θ et prédit en utilisant la marginalisation :

$$p(x|\mathcal{D}) = \sum_{\theta} p(x|\theta)p(\theta|\mathcal{D}).$$

L'idée est d'utiliser plusieurs modèles valides et très proches l'un de l'autre pour en former un seul.

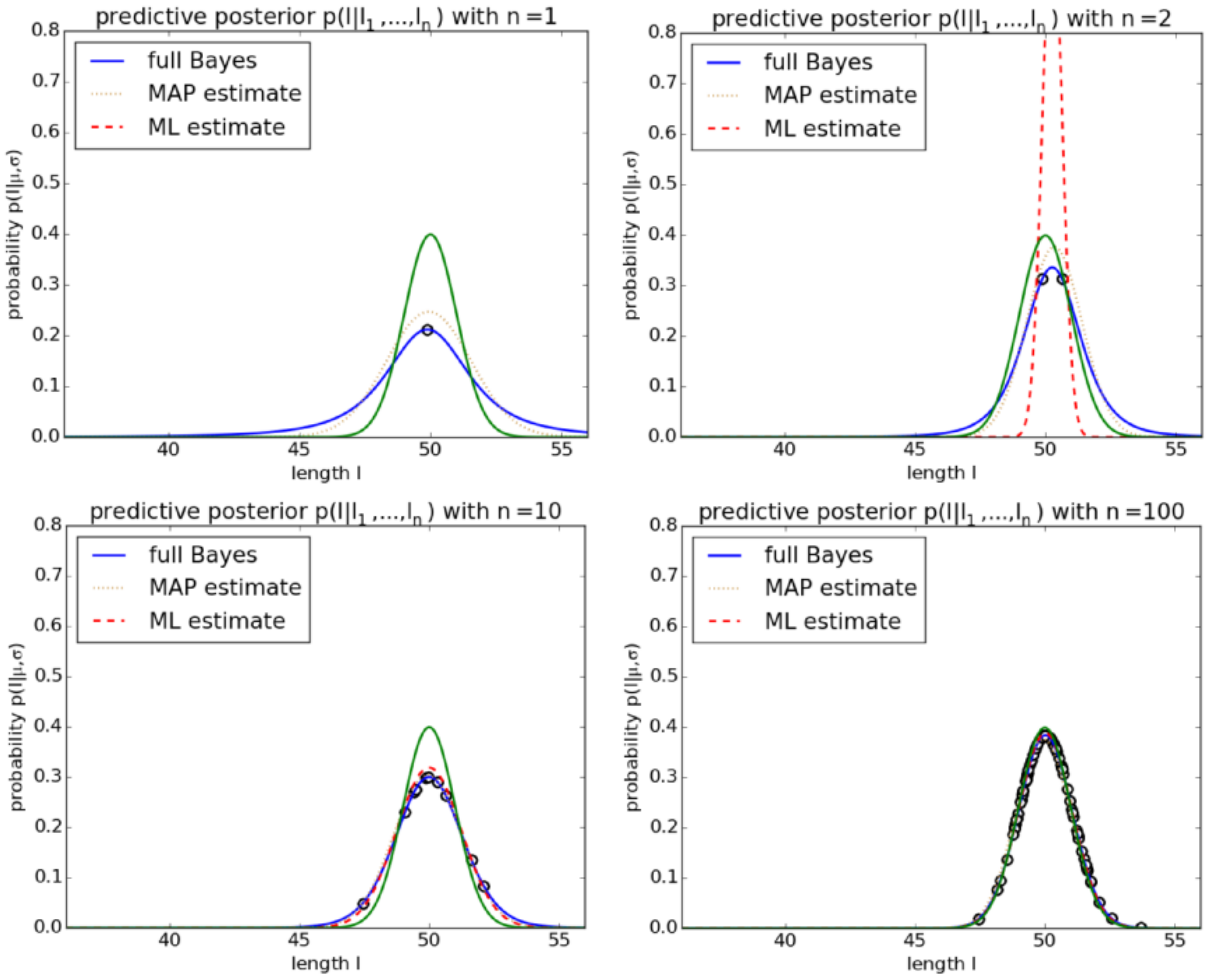
14.2 Illustration : newborn length

FIGURE 14.2 – From prior to posterior



Plus on est fiable, plus on peut resserer le prior, jusqu'à arriver au postérior

FIGURE 14.3 – From prior to posterior



$$p(l|l_1,\ldots,l_n) = \int_{\mu,\sigma} p(l|\mu,\sigma)p(\mu,\sigma|l_1,\ldots,l_n)d\mu d\sigma$$

La vraissemblance maximale n'utilise pas de prior, le MAP si et full Bayes tiet compte de plusieurs modèles avec prior et posterior.

14.3 Naive Bayes and Dirichlet priors

14.4 Hidden Markov models

14.5 Markov random fields

14.6 Gaussian processes for regression

17-> 34/37