Intelligence Artificielle

Fabien Lauer
fabien.lauer@loria.fr

Master 1 Informatique
Université de Lorraine

UE IA

- 12h CM + 12h TD (3 TD + 3 TP)
- Evaluation : 1 note d'examen + 1 note de TP (projet)

Contenu

- Notion d'agent intelligent
- Algorithmes de recherche pour la résolution de problèmes
- Jeux
- Modélisation de l'incertain (probabilités, réseaux bayesiens...),
- Apprentissage supervisé (discrimination, régression)

Bibliographie

- *Artificial Intelligence: a modern approach* de S. Russel, P. Norvig
- *The Elements of Statistical Learning* de T. Hastie, R. Tibshirani, J. Friedman

Plan

Qu'est-ce que l'Intelligence Artificielle (IA) ?

- IA = étude, analyse et conception de systèmes intelligents

- L'IA reste un domaine aux contours assez flous
- Différentes définitions se distinguent selon deux axes :

Un système intelligent est un système qui est capable...

de penser comme un humain	de raisonner
d'agir comme un humain	d'agir raisonnablement

Agir comme un humain

Test(s) de Turing

- Est-ce que la machine peut avoir une conversion naturelle avec un humain ?
- Est-ce qu'un humain peut faire la différence entre une conversation tenue par un humain ou par une machine ?
- A l'écrit, en face à face, en autorisant le passage d'objets ?

- Pas si intéressant que ça... tant qu'on ne souhaite pas interagir directement avec (ou remplacer) un humain
- Mais inclut beaucoup de problèmes d'IA
  - traitement du langage naturel
  - représentation des connaissances
  - raisonnement automatique
  - apprentissage automatique
  - vision (pour le face à face)
  - robotique

- Cleverbot (en 2011, "59% humain" contre 63% pour les vrais humains)

Penser comme un humain

Sciences cognitives

- Comprendre le fonctionnement du cerveau
- En le simulant avec une machine
- Si les résultats de la simulation coïncident avec les observations sur les humains, alors le modèle implémenté nous apprend peut-être quelque chose sur le cerveau

- La machine n'est qu'un outil de simulation, pas une fin en soi
- On ne cherche pas à résoudre des problèmes ou à "remplacer l'humain"

- Human Brain Project (projet européen d'envergure, 2014–2024)

Penser raisonnablement

Logique

- Implémenter les règles de la pensée rationnelle dans les machines
- Déductions automatiques à partir d'axiomes / de faits

- Limites / difficultés
  - Formulation des connaissances informelles en langage formel
  - Les connaissances sont souvent approximatives plutôt que des faits certains
  - La complexité des problèmes devient vite énorme (explosion du nombre de raisonnements possibles à partir de quelques dizaines de faits)

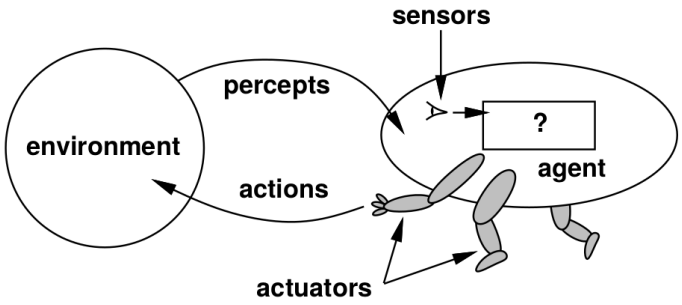
Agir raisonnablement

Théorie de la décision, statistique, apprentissage

- Agir le mieux possible pour atteindre un objectif à partir d'une certaine connaissance/de certaines croyances sur le monde
- Décider qu'une action est meilleure dans certaines circonstances
- Avantages
  - Cette définition est claire et précise
  - Les problèmes peuvent se formuler mathématiquement
  - et être analysés en s'appuyant sur beaucoup d'outils pour construire une théorie solide
  - Nombreuses applications pratiques
- Inconvénients
  - Ne raisonne pas forcément comme un humain
  - Ne permet pas de comprendre le cerveau

Approche suivie dans ce cours

Agents intelligents



Un agent...

- vie/évolue dans un **environnement**
- **perçoit** son environnement grâce à ses **capteurs**
- **agit** sur cet environnement grâce à des actionneurs/effecteurs

Agent rationnel = agent qui exécute la meilleure action possible étant donné sa perception du monde

Mesure de performance

Un agent (ou une séquence d'actions) est "meilleur" qu'un autre = sa mesure de performance est plus grande

Exemples de mesures de performances

La performance d'un aspirateur intelligent se mesure en fonction de

- la quantité de poussière aspirée ?
- et l'électricité consommée ?
- et le niveau de bruit ?

- sur une heure ? sur la journée ? sur l'année ?

Construire une mesure de performance peut être complexe, mais constitue une des étapes les plus importantes (car cela détermine directement ce qu'un agent rationnel fera)

Limitations dues aux perceptions

Agent rationnel vs omniscient

- Un agent rationnel peut décider de traverser la rue et mourir en se prenant un frigo tombé d'un avion sur la tête
- Car ses perceptions ne lui permettent pas de savoir que le frigo arrive
- Son action était rationnelle = idéale étant donné ses connaissances
- MAIS traverser la rue sans regarder à gauche et à droite n'est pas rationnel
  Ne pas oublier qu'acquérir de l'information peut être une action

Agent rationnel idéal

- Pour toute séquence de perception, prend l'action qui **maximise la performance** attendue sur la base des perceptions et de ses connaissances
- Pour pouvoir s'adapter, un agent doit être **autonome** : ses actions ne dépendent pas que de son constructeur mais aussi de son expérience

Agents qui résolvent des problèmes

Formulation d'un problème et d'un objectif

- Mesure de performance : inclue beaucoup de critères
- Dans certaines situations, il faut se concentrer sur 1 objectif précis
- But = ensemble des états dans lesquels l'objectif est atteint
- Solution = suite d'actions menant au but
- Coût d'une solution : certains problèmes ont plusieurs solutions, certaines ont un coût plus élevé, on cherchera donc la solution avec le coût le plus faible

Un agent doit trouver une séquence d'actions qui le mène au but = **recherche**
puis l'exécute = **exécution**

Je suis perdu en Roumanie et...

- je dois prendre l'avion à Bucarest dans 2 jours prendre l'avion à Bucarest dans 2 jours prendre l'avion à Bucarest dans 2 jours ($\sim$ le plus vite possible)
- je souhaiterais visiter des sites touristiques
- je souhaiterais voir un peu la campagne
- je n'aime pas trop les bosses sur la route
- j'aime bien écouter la radio en roulant
- je n'ai pas trop d'argent pour l'essence...

Agents qui résolvent des problèmes

Différents cas de figure

- Cas 1 : monde accessible et effets des actions connus
- Cas 2 : monde partiellement accessible et effets des actions connus
- Cas 3 : monde accessible et effets des actions connus partiellement
- Cas 4 : monde partiellement accessible et effets des actions partiellement connus

Solutions

- Cas 1 : Séquence d'actions = séquence d'états $\Rightarrow$ trouver celle qui conduit au but
- Cas 2 : Idem mais avec une séquence d'ensembles d'états possibles
- Cas 3 : Séquence d'actions conditionnées aux perceptions
- Cas 4 : Pas toujours de solution figée qui garantisse d'arriver au but... il faut itérer entre la recherche d'une solution (partielle) et l'exécution d'actions

Exemple de l'aspirateur...

Algorithmes de recherche et résolution de problèmes

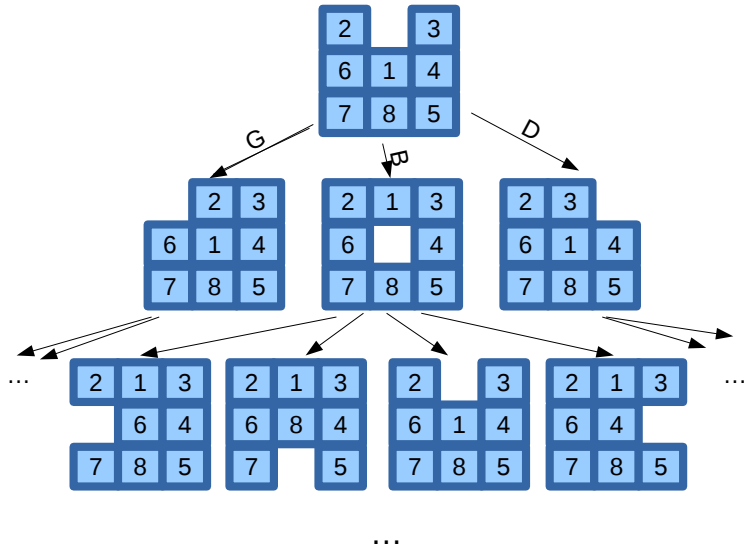
Arbre de recherche

- Racine = état initial
- 1 nœud pour chaque état du problème
- 1 branche pour chaque action possible, pondérée par le coût de l'action
- On descend dans l'arbre en suivant une séquence d'actions jusqu'à un nœud correspondant au but (ensemble d'états satisfaisant l'objectif)
- Le coût d'une solution est la somme des pondérations des branches traversées

Exemple : taquin

- Etat : position des pièces et du trou
- Actions possibles : déplacer le trou vers le haut, le bas, la gauche ou la droite
  (en fonction de l'état, certaines actions sont impossibles)
- Coût d'une action = 1 pour toutes les actions
  on cherche à minimiser le nombre de coups pour arriver au but

Arbre de recherche



Algorithmes de recherche et résolution de problèmes

Algorithmes de recherche "aveugles"

- **Largeur d'abord** : garantie de trouver une solution (avec le moins d'actions possibles) mais demande beaucoup de mémoire
- **Coût uniforme** : idem largeur d'abord, mais trouve la solution de coût minimal (mais peut-être plus d'actions) en creusant les nœuds qui minimisent le coût $g(noeud)$ entre la racine et le nœud
- **Profondeur d'abord** : plus efficace en terme de mémoire, peut trouver une solution rapidement mais peut aussi perdre beaucoup de temps dans des branches inintéressantes
- Mélange : augmenter progressivement la profondeur maximale d'une recherche en profondeur d'abord

Algorithmes de recherche "informés" : le meilleur d'abord

Avec une heuristique $h(noeud)$ qui estime le coût restant pour le meilleur chemin entre *noeud* et *but*

- **Recherche gloutonne** : creuser le nœud qui minimise $h(noeud)$ pour se rapprocher le plus possible du but ; mais sans garantie sur le coût de la solution
- $\boldsymbol{A^*}$: combiner coût uniforme et recherche gloutonne : creuser le nœud qui minimise $f(noeud) = g(noeud) + h(noeud) =$ estimation du coût total d'une solution passant par *noeud*

A^* : heuristiques admissibles

- Une heuristique est dite **admissible** si $h(noeud) \leq$ coût du meilleur chemin de *noeud* à *but* pour tout *noeud*
- Si l'heuristique est admissible alors A^* est garanti de trouver une **solution optimale**

Problème du taquin

- $g(n) =$ nombre de coups joués
- $h_1(n) =$ nombre de pièces mal placées
- $h_2(n) =$ somme des distances des pièces à leur position finale
- h_1 et h_2 admissibles car nb coups qu'il reste à jouer $\geq h_1(n)$ et $\geq h_2(n)$
- Quelle est la meilleure heuristique ?
  celle qui estime le mieux le coût restant
- Puisque coût restant $\geq$ heuristique, celle qui estime le mieux le coût restant est celle qui est la plus grande *pour tous les nœuds*
- $h_2(n) \geq h_1(n)$ pour tout n donc h_2 est meilleure !

Algorithmes de recherche en pratique

Algorithme générique

Squelette identique pour tous les algorithmes de recherche :

1. Prendre un nœud dans la file d'attente
2. Vérifier si le but est atteint à ce nœud (si oui, retourner la solution)
3. Développer le nœud et ajouter les fils à la file d'attente

Un algorithme de recherche = algorithme générique + fonction pour ordonner la file d'attente

- Largeur d'abord : ajouter les nouveaux nœuds à la fin de la file
- Coût uniforme : trier la file par ordre croissant de $g(noeud)$
- Profondeur d'abord : ajouter les nouveaux nœuds au début de la file
- Glouton : trier la file par ordre croissant de $h(noeud)$
- A^* : trier la file par ordre croissant de $f(noeud) = g(noeud) + h(noeud)$

Eviter les états répétés

Problème

- Plusieurs chemins mènent au même état, donc dans un arbre le même état est évalué plusieurs fois...
- En profondeur d'abord, ou glouton, encore plus grave : problème de profondeur infini
- Pour les autres, limiter les répétitions accélère aussi l'algorithme

Solutions

3 niveaux de plus en plus efficaces mais de plus en plus coûteux

1. Ne pas revenir en arrière
2. Ne pas générer un état présent dans le chemin qui y mène (pour éviter les cycles)
3. Ne pas générer d'état présent dans l'arbre

Qu'est-ce qu'un jeu ?

Définitions / limitation du cadre d'étude

- Jeux déterministes (dans un premier temps) : pas de hasard
- Jeux à information parfaite : tous les éléments sont connus de tous les joueurs
- Jeux à 2 joueurs (un contre l'autre)

  Remarque : les jeux à plus de 2 joueurs sont forcément à information imparfaite (alliances...)

Exemples

- Dames : Chinook (champion du monde en 1994) (alpha-beta + répertoire de fins de partie)
- Othello/reversi
- Echecs : Kasparov vs IBM DeepBlue (1997) (alpha-beta + répertoires d'ouvertures et de fins + machine parallèle dédiée)
- Jeopardy : IBM Watson (2011) (rien à voir...)
- Go : AlphaGO (champion européen en 2013, champion du monde en 2015) (MCTS + apprentissage)

Jeux à deux joueurs (Max vs Mini)

Formulation d'un jeu

- **Etat initial** : position des pièces, à qui de jouer...
- Ensemble d'**opérateurs** : définit tous les coups possibles pour un joueur (fonction de transition d'un état à l'autre)
- **Test de terminaison** : est-ce que la partie est finie ?
- **Fonction de récompense** : retourne une valeur numérique à la fin de la partie en fonction de l'état du jeu (par ex. $\{+1, 0, -1\}$ ou un score)

Arbre de jeu

- Chaque nœud correspond à un état du jeu
- Chaque branche correspond à un coup possible
- Chaque niveau/profondeur est associé au joueur qui doit jouer

Différent d'un problème de recherche car l'adversaire agit sur le résultat : il ne suffit pas de prendre un chemin qui mène à la meilleure récompense, mais un chemin qui mène à la bonne récompense quelles que soient les actions de l'adversaire

Minimax

Action optimale dans un jeu à deux joueurs

- Sous l'hypothèse que l'autre joueur joue de manière optimale
- Avec une information complète sur l'état du jeu
- En énumérant toutes les séquences de jeu possibles (en développant tout l'arbre de jeu)
- En associant une valeur à chaque nœud

Algorithme Minimax (du point de vue de Max)

- Générer entièrement l'arbre de jeu
- Evaluer la récompense pour chaque nœud terminal
  valeur d'un nœud terminal = récompense
- Remonter l'arbre en propageant
  - la **valeur minimale** sur l'ensemble des fils si **c'est à Mini de jouer**
    (on admet que l'adversaire joue son meilleur coup)
  - la **valeur maximale** sur l'ensemble des fils si **c'est à Max de jouer**
    (on admet que l'on joue notre meilleur coup)
- Jouer le coup correspondant au fils de la racine avec la valeur maximale

En pratique : algorithme récursif avec
$valeurMax(noeud) = \max_{f \in fils(noeud)} valeurMin(f)$ (ou récompense pour un *neoud*
$valeurMin(noeud) = \min_{f \in fils(noeud)} valeurMax(f)$ terminal)

Minimax

Complexité du minimax

- b : facteur de branchement moyen
  $\sim$ 10 au reversi, $\sim$ 35 aux échecs, $\sim$ 360 au Go
- n : profondeur moyenne d'une branche
  60 au reversi, $\sim$ 40 aux échecs, $\sim$ 400 au Go

- Complexité en temps : $O(b^n)$
- Complexité en espace : $O(bn)$
  En pratique, avec une approche en profondeur d'abord il est suffisant de mémoriser uniquement la branche en cours d'exploration

Deux solutions possibles

- Réduire b
- Réduire n

Alpha–beta (pour réduire b)

Elaguer l'arbre pour gagner du temps

- L'arbre est parcouru en profondeur d'abord
  $\Rightarrow$ la valeur des nœuds d'une branche peut être calculée avant de passer à la branche suivante
- Ne pas explorer les branches qui ne seront jamais jouées car elles ne correspondent pas à des stratégies optimales pour les joueurs

Algorithme Alpha-beta : minimax modifié (init : $\alpha = -\infty, \beta = \infty$)

Fonction valeurMax (nœud, α, β)

- Pour chaque fils de nœud
  - calculer sa valeur $v = \mathit{valeurMin}(\mathit{fils}, \alpha, \beta)$
  - $\alpha = \max(\alpha, v)$
  - Si $\alpha \geq \beta$, retourner α
- Retourner α

Fonction valeurMin (nœud, α, β)

- Pour chaque fils de nœud
  - calculer sa valeur $v = \mathit{valeurMax}(\mathit{fils}, \alpha, \beta)$
  - $\beta = \min(\beta, v)$
  - Si $\beta \leq \alpha$, retourner β
- Retourner β

Fonctions d'évaluation et limitation de profondeur

Réduire la profondeur n

- Limiter la profondeur de recherche à n (par exemple $n = 5$ ou $n = 10$)
- Estimer la qualité des nœuds à la profondeur n par une fonction d'évaluation plutôt que de calculer la valeur exacte du nœud à partir de la récompense finale

Fonction d'évaluation

- Doit correspondre à la récompense si appliquée à une position terminale
- Doit être calculable rapidement
- Doit donner une bonne image des "chances" de gagner à partir d'une position
- Pour les jeux "à somme nulle", doit être de la forme $\mathit{evalJoueur}(\mathit{Max}) - \mathit{evalJoueur}(\mathit{Mini})$

Attention : en limitant la profondeur de la recherche, l'algorithme Minimax n'est plus optimal ; il est totalement aveugle à ce qui se passe après n coups

Fonctions d'évaluation et limitation de profondeur

Fonction d'évaluation linéaire

$$\mathit{evaluation}(\mathit{etat}) = \sum_{k=1}^{p} w_k \phi_k(\mathit{etat})$$

Ex. aux échecs : $\phi_k(\mathit{etat})$ = nb de pièces du type k, w_k = valeurs des pièces

- les fonctions de caractéristiques ϕ_k dépendent du jeu...
- les poids w_k peuvent être déterminés automatiquement par "apprentissage" si un expert évalue un ensemble de positions

Jouer contre soi-même pour s'améliorer et apprentissage

- On peut "apprendre" la fonction d'évaluation à partir d'un historique de parties
- Pour chaque position on note l'évaluation donnée par la fraction de parties gagnées à partir de cette position
- Un modèle est entraîné à reproduire ces évaluations (et à généraliser !)
- Cela nécessite un GRAND nombre de parties pour explorer suffisamment de positions suffisamment de fois
- En faisant jouer le programme contre lui-même, un grand nombre de parties peuvent être jouées rapidement !
- Attention : si le programme est mauvais au départ, les parties ne seront pas représentatives des vrais adversaires
- Solution : commencer par "apprendre" à imiter les joueurs humains

Jeux avec du hasard

- Un coup ne donne le résultat prévu qu'avec une certaine probabilité
- Un tirage aléatoire pré-conditionne chaque coup

Arbre probabiliste : arbre de jeu contenant des "nœuds chance"

- nœud chance : nœud intermédiaire dont l'action est aléatoire
- En générale, toutes les positions atteignables à partir d'un état sont atteintes avec une probabilité non nulle
  $\Rightarrow$ il faut explorer toutes les branches (pas d'équivalent d'α-β)
- Une séquence d'actions ne conduit qu'à une certaine probabilité de gagner

Algorithme Expectimax

- Algorithme Minimax modifié tel que la valeur remontée soit
  - la valeur minimale sur l'ensemble des fils si c'est à Mini de jouer
  - la valeur maximale sur l'ensemble des fils si c'est à Max de jouer
  - la **valeur moyenne** sur l'ensemble des fils si c'est un **nœud chance** :

$$\sum_{i \in \mathit{fils}(n)} P(i)\, \mathit{valeurExpectimax}(i), \qquad P(i) \text{ : probabilité d'avoir } i \text{ après } n$$

Algorithmes à base de marches aléatoires

Motivations

- Pas de fonction d'évaluation disponible (par ex. Go)
- Réflexion en temps limité (ne pas dépasser + profiter au max du temps)

Algorithme aléatoire uniforme

- Chaque nœud est évalué selon la probabilité de gagner à partir de là
- On joue le coup qui a la plus grande probabilité de gagner
- Les probabilités sont estimées par simulation :
  Pour chaque coup possible, générer N parties aléatoires et estimer $P(gagner)$ par la fraction de parties gagnées
- Une partie aléatoire = choix aléatoire (uniforme) à chaque coup
- Remarque : les parties sont simulées jusqu'à la fin (nœud terminal)
- Amélioration : modifier le choix aléatoire en introduisant des connaissances sur les bons (ou mauvais) coups
  par ex. toujours choisir un coup gagnant quand cela est possible

Recherche d'arbre Monte Carlo

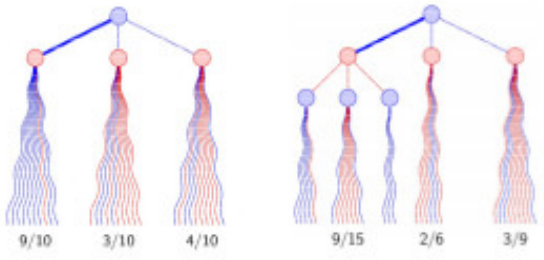
Motivations

- La recherche aléatoire uniforme peut facilement passer à côté des bons coups et sous-estimer les probabilités de gagner
- Idée 1 : Explorer les coups qui n'ont pas été suffisamment simulés pour éviter de passer à côté de quelque chose
- Idée 2 : Exploiter l'information acquise au cours du processus pour améliorer les simulations suivantes et l'estimation des probabilités
- Chaque idée demande du temps $\Rightarrow$ **compromis Exploitation / Exploration**

MCTS (*Monte-Carlo Tree Search*)

- Approche optimiste : on considère que les probabilités sont bien estimées avant la fin des simulations (voire même dès le début)
- Donc on peut exploiter ces probabilités pour améliorer les simulations suivantes (les rendre plus réalistes) : on favorise les simulations qui passent par les nœuds avec une forte probabilité de gagner au lieu de choisir aléatoirement
- On introduit un critère permettant d'explorer les branches qui ne l'ont pas été suffisamment

Recherche d'arbre Monte Carlo vs Algorithme aléatoire uniforme



- L'algorithme aléatoire uniforme génère N simulations indépendantes par coup possible à partir de l'état initial (état courant de la partie)
- MCTS construit un arbre de plus en plus grand où chaque nœud est associé à une estimation de la probabilité de gagner et en se concentrant sur les coups qui paraissent meilleurs

Recherche d'arbre Monte Carlo

Algorithme UCT (*Upper Confidence bounds in Trees*)

Tant que (il reste du temps)

1. **Sélectionner** récursivement à partir de la racine le nœud avec la plus grande B-valeur jusqu'à arriver à un nœud terminal ou un dont tous les fils n'ont pas été développés
2. **Développer** un fils choisi aléatoirement parmi les fils non développés
3. **Simuler** la fin de la partie avec une marche aléatoire
4. **Mettre à jour** les B-valeurs de tous les nœuds sur le chemin de la racine au nœud terminal en remontant la récompense r de la position finale

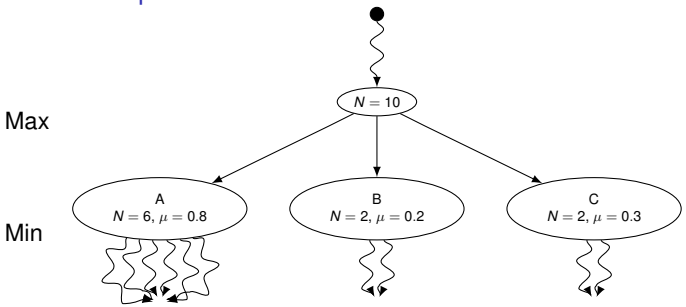
B-valeur du nœud i

$$B(i) = \pm\mu(i) + c\sqrt{\frac{\ln N(parent(i))}{N(i)}}$$

- $\pm$: alternance des joueurs, $+$ si $parent(i)$ est un nœud Max, $-$ si Min
- $\mu(i) = \frac{1}{N(i)} \sum_{\boldsymbol{s}: i \in \boldsymbol{s}} r(\boldsymbol{s})$: moyenne de la récompense $r(\boldsymbol{s})$ sur l'ensemble des simulations $\boldsymbol{s}$ passant par i
  si $r() \in \{0, 1\}$ (perdu ou gagné), alors $\mu(i)$ estime la probabilité de gagner
- $N(i)$: nombres de simulations passant par i
- c : constante permettant de régler le compromis exploitation / exploration

Recherche d'arbre Monte Carlo – UCT

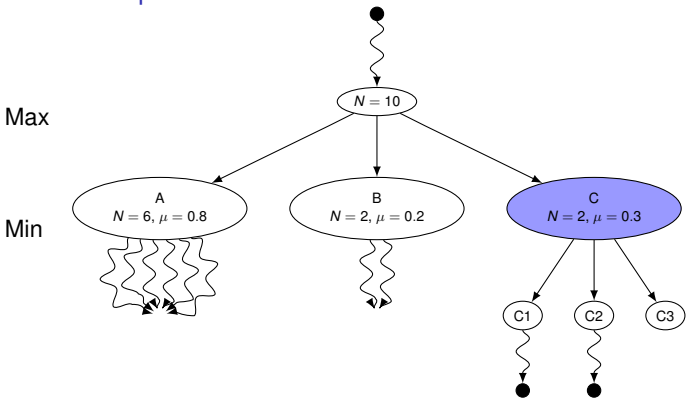
Exemple d'itération



1. **Sélection** de A, B ou C

Recherche d'arbre Monte Carlo – UCT

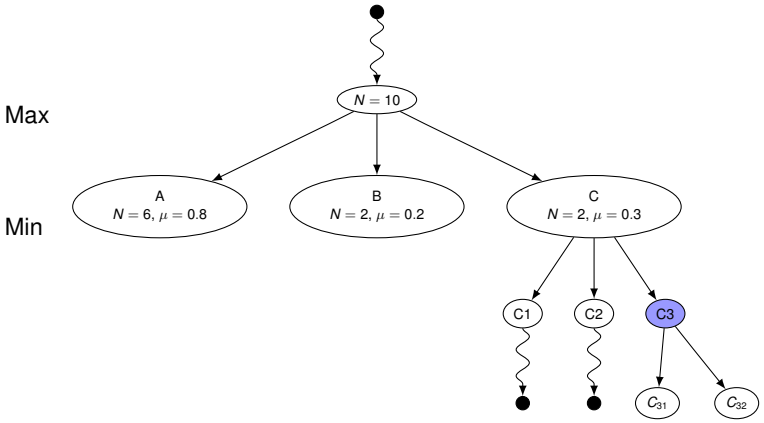
Exemple d'itération



1. **Sélection** du nœud C (avec $c = 2$) :

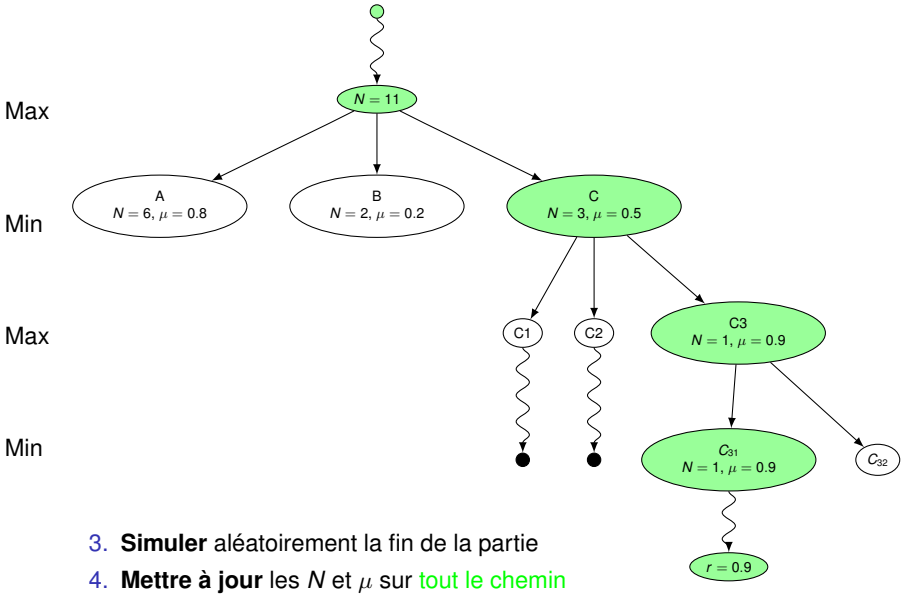
$$B(A) = 0.8 + 2\sqrt{\frac{\ln 10}{6}} \approx 2.04, \quad B(B) \approx 2.35, \quad B(C) \approx 2.45$$

Recherche d'arbre Monte Carlo – UCT



1. **Sélection** du nœud C
2. **Développer** le dernier fils C3 du nœud C

Recherche d'arbre Monte Carlo – UCT



3. **Simuler** aléatoirement la fin de la partie
4. **Mettre à jour** les N et μ sur tout le chemin

Remarque : si $r \in \{0, 1\}$, $\mu(i) = \frac{\text{nb de victoires passant par } i}{N}$

Recherche d'arbre Monte Carlo

Mise à jour des B-valeurs en pratique

$$B(i) = \pm\mu(i) + c\sqrt{\frac{\ln N(parent(i))}{N(i)}} \quad \text{avec} \quad \mu(i) = \frac{1}{N(i)} \sum_{\boldsymbol{s}:i\in\boldsymbol{s}} r(\boldsymbol{s})$$

- Mémoriser $\mu(i)$ et $N(i)$ au lieu de $B(i)$
- Mise à jour :

$$\mu(i) \leftarrow \frac{N(i)\mu(i) + r}{N(i) + 1} \quad ; \quad N(i) \leftarrow N(i) + 1$$

ou alors :

- Mémoriser $S(i) = \sum_{\boldsymbol{s}:i\in\boldsymbol{s}} r(\boldsymbol{s})$ et calculer $\mu(i) = \frac{S(i)}{N(i)}$
- Mise à jour plus simple : $S(i) \leftarrow S(i) + r$
- Si la récompense est binaire (0 : perdu ou 1 : gagné), alors $S(i) =$ nb de victoires passant par i

Remarque : le signe codant l'alternance entre Max et Mini peut être pris en compte soit dans $B(i) = \pm\mu(i) + ...$, soit dans $S(i) = \sum_{\boldsymbol{s}:i\in\boldsymbol{s}} \pm r(\boldsymbol{s})$

Recherche d'arbre Monte Carlo

Choix du coup à jouer

Tant que (il reste du temps)
...
Retourner le coup qui maximise

- soit la récompense attendue $\mu(i)$ (règle "max")
- soit le nombre de simulations $N(i)$ (règle "robuste")
- soit la récompense attendue $\mu(i)$ et le nombre de simulations (règle "max-robuste") (si aucun coup ne maximise les deux, on continue à chercher)

Améliorations

- Biaiser la sélection et/ou les simulations
  par ex. en choisissant toujours les coups gagnants ou qui empêchent l'autre joueur de gagner en un coup ; ou en repérant des motifs particuliers
- Utiliser une fonction d'évaluation pour limiter la profondeur des simulations
- Si l'ordre des coups n'a pas d'importance : mettre à jour les valeurs de tous les nœuds correspondant aux actions choisies pendant la simulation
- Beaucoup d'autres techniques plus ou moins spécifiques à chaque jeu

Succès de l'IA dans les jeux

IBM DeepBlue

- Aux échecs, $b \approx 30$, $n \approx 40$
- Au début et à la fin : répertoires d'ouvertures et de fin de partie
- Au milieu : alpha-beta
- Fonction d'évaluation avec +8000 paramètres + machine parallèle dédiée

AlphaGO

- Au GO, $b \approx 360$, $n \approx 400$, pas de fonction d'évaluation connue
- La machine doit apprendre la fonction d'évaluation

1. Apprentissage d'un modèle imitateur (réseau de neurones profond) à partir d'une collection de parties jouées par des humains
2. Le modèle joue contre lui-même pour s'améliorer et générer des millions de parties
3. Apprentissage de la fonction d'évaluation à partir de ces parties
4. MCTS biaisé par la fonction d'évaluation + simulations orientées par le modèle imitateur
   + beaucoup de CPU et GPU en parallèle

Recherche d'arbre Monte Carlo UCT et bandits manchots

Bandits manchots à K bras (machines à sous)

- Chaque bras a une probabilité **inconnue** de gagner
- A chaque étape, l'algorithme choisit un bras à tirer
- BUT : maximiser les gains en essayant de tirer le plus souvent possible le meilleur bras
- **Compromis Exploration / Exploitation** : tirer le bras que l'on pense

Rappels de probabilités

Espace probabilisé $(\Omega, \mathcal{A}, P)$

- **Univers** Ω : ensemble des résultats d'expérience possibles
- **Tribu (σ-algèbre)** $\mathcal{A}$ **de** Ω : ensemble de tous les événements
  tribu de Ω = ensemble de sous-ensembles de Ω tel que i) $\Omega \in \mathcal{A}$, ii) $A \in \mathcal{A} \Rightarrow \overline{A} = (\Omega \setminus A) \in \mathcal{A}$, iii) $A, B \in \mathcal{A} \Rightarrow A \cup B \in \mathcal{A}$, iv) si $A_i \in \mathcal{A}$ pour tout $i \in \mathbb{N}$ alors $\cup_{i \in \mathbb{N}} A_i \in \mathcal{A}$
- **Evénement** A : $A \subseteq \Omega$ et $A \in \mathcal{A}$
- **Mesure de probabilité** P : fonction associant une probabilité entre 0 et 1 à un événement

$$P : \mathcal{A} \to [0, 1]$$

  telle que
  - $P(\Omega) = 1$ et $P(\emptyset) = 0$
  - Pour un nombre fini d'événements A_i disjoints : $P(\bigcup_i A_i) = \sum_i P(A_i)$

Propriétés d'une probabilité

- $P(\overline{A}) = 1 - P(A)$ $\qquad\qquad$ $\overline{A} = \Omega \setminus A$
- $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- Si A et B disjoints : $P(A \cup B) = P(A) + P(B)$

Rappels de probabilités

Exemple : le lancé d'un dé

- $\Omega = \{$⚀,⚁,⚂,⚃,⚄,⚅$\}$
- $\mathcal{A} = \mathcal{P}(\Omega) = \{\{$⚀$\}, \{$⚁$\}, \ldots, \{$⚀,⚁$\}, \ldots, \{$⚀,⚁,⚂$\}, \ldots, \ldots\}$
  $\mathcal{A}$ est l'ensemble de toutes les sous-parties de Ω
- 1 événement : $\{$⚀,⚁$\}$, 1 autre événement : $\{$⚂$\}$
- Probabilité : $P(\{\omega\}) = 1/6$ pour tout $\omega \in \Omega$
- Union : $P(\{$⚀,⚁$\}) = P(\{$⚀$\} \cup \{$⚁$\}) = P(\{$⚀$\}) + P(\{$⚁$\}) = 2/6$
- Complément : $P(\{$⚀,⚁,⚂,⚃,⚄$\}) = P(\Omega) - P(\{$⚅$\}) = 1 - 1/6 = 5/6$

Rappels de probabilités

Variables aléatoires (v.a.)

- Variable aléatoire réelle X : **fonction** associant une valeur numérique au résultat d'une expérience

$$X : \Omega \to \mathbb{R}$$

- Pour $A \subseteq \mathbb{R}$, l'ensemble des résultats d'expérience conduisant à une valeur de la v.a. dans A est un événement :

$$\{\omega \in \Omega \mid X(\omega) \in A\} \in \mathcal{A}$$

- On notera simplement X pour $X(\omega)$ et $\{X \in A\}$ pour $\{\omega \in \Omega \mid X(\omega) \in A\}$

Loi d'une variable aléatoire

- On définit une mesure de probabilité P_X sur $\mathbb{R}$ muni de la tribu $\mathcal{B}$ par

$$P_X : \mathcal{B} \to [0, 1]$$
$$P_X(A) = P(X \in A)$$

- P_X **est la loi de** X
- P_X est la probabilité image de P par la fonction X
- En pratique : utilisation de l'espace probabilisé $(\mathbb{R}, \mathcal{B}, P_X)$

Rappels de probabilités

Exemple : tirage aléatoire d'un individu dans une population

- Ω : ensemble des individus de la population
  (chaque individu est un résultat possible)
- X : taille de l'individu en cm
- $\mathcal{B}$: tribu borélienne de $\mathbb{R}$
  ensemble de toutes les parties de $\mathbb{R}$ engendrées par union, intersections et compléments d'événements du type $(X < x)$
- Un exemple d'événement pour l'espace probabilisé $(\mathbb{R}, \mathcal{B}, P_X)$: $[170, 180]$
- Correspond à l'événement $\{\omega \in \Omega | X(\omega) \in [170, 180]\}$, noté de manière courte $\{X \in [170, 180]\}$, pour l'espace probabilisé $(\Omega, \mathcal{A}, P)$
- Probabilité : $P_X([170, 180]) = P(X \in [170, 180]) = 0.3$
  (probabilité de tiré aléatoirement un individu dont la mesure de la taille soit entre 170 et 180 cm)

Rappels de probabilités

Variables aléatoires discrètes

- Une v.a. discrète Y ne peut prendre qu'un nombre fini de valeurs distinctes
$$Y \in \mathcal{Y} = \{y_k\}_{1 \leq k \leq n}$$
- La loi P_Y d'une v.a. discrète est donnée par la somme des probabilités de chacune des valeurs possibles :
$$P_Y(A) = \sum_{y \in \mathcal{Y} \cap A} P(Y = y) = \sum_{y_k \in A} P(Y = y_k)$$

Variables aléatoires continues

- Une v.a. continue X prend ses valeurs dans $\mathbb{R}$
- Densité de probabilité $p : \mathbb{R} \to \mathbb{R}^+$
- X suit la loi de **densité** p si :
$$P_X(A) = P(X \in A) = \int_A p(x)dx$$
- Remarque :
$$\text{pour } a \in \mathbb{R}, \quad P_X(a) = P(X = a) = \int_a^a p(x)dx = 0$$

Lois usuelles

Exemples de lois discrètes

- Loi uniforme sur $\{1, \ldots, n\}$:
$$\forall y \in \{1, \ldots, n\}, \quad P(Y = y) = \frac{1}{n}$$
- Loi de Bernoulli de paramètre $b \in [0, 1]$:
$$P(Y = 1) = b, \quad \text{et} \quad P(Y = 0) = 1 - b$$

Exemples de densités de probabilité

- Densité de la loi uniforme sur $[a, b]$:
$$p(x) = \begin{cases} \frac{1}{b-a}, & \text{si } x \in [a, b] \\ 0, & \text{sinon} \end{cases}$$
- Densité de la loi gaussienne (ou normale) $\mathcal{N}(\mu, \sigma^2)$:
$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$

Rappels de probabilités

Espérance : valeur "en moyenne" d'une variable aléatoire

Espérance d'une v.a. discrète

$$\mathbb{E}[Y] = \sum_{y \in \mathcal{Y}} y\, P(Y = y)$$

$$\mathbb{E}[f(Y)] = \sum_{y \in \mathcal{Y}} f(y)\, P(Y = y)$$

Espérance d'une v.a. continue à densité

$$\mathbb{E}[X] = \int_{\mathbb{R}} x\, p(x)\, dx$$

$$\mathbb{E}[f(X)] = \int_{\mathbb{R}} f(x)\, p(x)\, dx$$

Linéarité de l'espérance

$$\mathbb{E}[aX] = a\mathbb{E}[X] \quad \textit{et} \quad \mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$

Rappels de probabilités

Fonction indicatrice $\mathbb{I}$

- $\mathbb{I}_X(A) = \mathbb{I}(X \in A) = 1$ si l'événement A est observé, 0 sinon
- Exemple : $\mathbb{I}(X > 5)$

Espérance de l'indicatrice d'un événement = Probabilité de l'événement

En discret :

$$\mathbb{E}_Y[\mathbb{I}_Y(A)] = \sum_{y \in \mathcal{Y}} \mathbb{I}_Y(A) P(Y = y) = \sum_{y \in \mathcal{Y} \cap A} P(Y = y) = P_Y(A)$$

En continu :

$$\mathbb{E}_X[\mathbb{I}_X(A)] = \int_{\mathbb{R}} \mathbb{I}_X(A)\, p(x)\, dx = \int_A 1\, p(x)\, dx + \int_{\overline{A}} 0\, p(x)\, dx = P_X(A)$$

Bandits manchots à K bras (machines à sous)

- v.a. X_i : gain du bras i, de loi **inconnue** P_i de moyenne $\mathbb{E}[X_i] = \mu(i)$
- Le meilleur bras i^* a la plus grande moyenne : $i^* = \arg\max_{i=1,\dots,K} \mu(i)$
- Un algorithme/une politique $A : t \mapsto i$ choisi un bras i à chaque étape t
- Si on tire T fois le bras i, on gagne

$$\sum_{t=1}^{T} X_{i,t} \qquad X_{i,t} \text{ : copie indépendante de } X_i$$

Donc on peut espérer gagner au mieux

$$Gain^*(T) = \mathbb{E}\left[\sum_{t=1}^{T} X_{i^*,t}\right] = \sum_{t=1}^{T} \mathbb{E}[X_{i^*,t}] = \sum_{t=1}^{T} \mu(i^*) = T\mu(i^*)$$

Si A est déterministe et indépendant des $X_{i,t}$, on espère gagner

$$Gain(T) = \mathbb{E}\left[\sum_{t=1}^{T} X_{A(t),t}\right] = \sum_{t=1}^{T} \mu(A(t))$$

- Le **regret** est la somme de ce qu'on aurait pu espérer gagner en plus :

$$Regret(T) = \mathbb{E}\left[Gain^*(T) - Gain(T)\right] = \mathbb{E}\left[\sum_{t=1}^{T} (\mu(i^*) - \mu(A(t)))\right]$$

On prend l'espérance car A dépend des résultats aléatoires précédents

Bandits manchots à K bras (machines à sous)

- Comment choisir quel bras tirer pour **minimiser le regret** ?
- Au début : n'importe lequel (aucune information permettant de choisir)
- Mais ensuite ? et les 100 coups suivants ? Toujours tirer celui qui gagne le plus souvent ? Essayer un autre ?
- Le regret se reformule en fonction du nombre $N_T(i)$ d'essais de chaque bras parmi T et des différences $\Delta_i = \mu(i^*) - \mu(i)$:

$$\begin{aligned} Regret(T) &= \mathbb{E}\left[\sum_{t=1}^{T} (\mu(i^*) - \mu(A(t)))\right] \\ &= \mathbb{E}\left[\sum_{i=1}^{K} N_T(i)(\mu(i^*) - \mu(i))\right] \\ &= \sum_{i=1}^{K} \mathbb{E}\left[N_T(i)\right] \Delta_i \end{aligned}$$

A devrait conduire à des petits $N_T(i)$ pour les grands Δ_i

Bandits manchots à K bras – Algorithmes gloutons

Algorithme glouton (*exploiter au maximum*)

- Jouer au moins une fois chaque bras pour observer les gains $x_{i,j}$
- Estimer $\mu(i)$ par $\hat{\mu}(i) = \frac{1}{N_t(i)} \sum_{j=1}^{t} x_{i,j} \mathbb{I}(A(j) = i)$
- Choisir ensuite le bras qui maximise $\hat{\mu}(i)$
- Risque de toujours choisir le même bras sous-optimal
  $\Rightarrow$ Regret linéaire en T

Algorithme ϵ-glouton (*explorer à l'infini*)

- A chaque étape,
  - avec une probabilité $1 - \epsilon$, appliquer l'algorithme glouton
  - avec une probabilité ϵ, tirer un bras aléatoirement
- Mais à chaque étape, le regret instantané est

$$\begin{aligned} \mathbb{E}[\mu(i^*) - \mu(A(t))] &= \sum_{i=1}^{K} \mathbb{E}\left[(\mu(i^*) - \mu(i))\mathbb{I}(A(t) = i)\right] = \sum_{i=1}^{K} \Delta_i P(A(t) = i) \\ &\geq \frac{\epsilon}{K} \sum_{i=1}^{K} \Delta_i = O(1) \end{aligned}$$

- $\Rightarrow$ Regret total linéaire en T

En faisant décroître ϵ avec t, on peut obtenir un regret logarithmique *à partir de la connaissance des* Δ_i

Bandits manchots à K bras – Algorithme UCB

Idée

- Imaginons que l'on dispose d'une estimation $\hat{\mu}(i)$ de $\mu(i)$ et d'un semi-intervalle de confiance $C(i)$ tel que :

$$\mu(i) \leq \hat{\mu}(i) + C(i) = B(i)$$

- Alors on pourrait choisir le bras qui maximise $B(i)$ pour
  - favoriser les bras que l'on croit bons (grand $\hat{\mu}(i)$)
  - sans oublier les bras qui *pourraient* être meilleurs (grand $C(i)$)

Algorithme UCB (*Upper Confidence Bound*)

Tirer chaque bras une fois pour $t \leq K$, puis le bras i qui maximise

$$B_t(i) = \hat{\mu}_{N_t(i)}(i) + \sqrt{\frac{3 \ln t}{2 N_t(i)}}$$

- $\hat{\mu}_n(i) = \frac{1}{n} \sum_{j=1}^{n} x_{i,j}$: estimation de $\mu(i)$ à partir de n observations $x_{i,j}$ des gains du bras i
- $N_t(i)$: Nombre de tirages du bras i sur les t premiers coups

MCTS-UCT : appliquer UCB à chaque nœud de l'arbre

Explication de la formule de $B(i)$ (UCB)

Théorème (inégalité de Hoeffding)

Soit une suite (X_j) de n v.a. indépendantes et identiquement distribuées à valeurs dans $[0,1]$ et de moyenne μ, alors

$$P\left\{\mu > \frac{1}{n}\sum_{j=1}^{n} X_j + \epsilon\right\} \leq e^{-2n\epsilon^2}$$

Pour UCB

On souhaite être de plus en plus sûr de ce que l'on dit, et donc on veut une probabilité de plus en plus faible que μ dépasse l'intervalle de confiance. Soit δ_t une borne que l'on se fixe sur cette probabilité à l'instant t, on pose

$$e^{-2n\epsilon_t^2} = \delta_t \qquad \text{et par exemple} \qquad \delta_t = t^{-3}$$

On en déduit le (semi)-intervalle de confiance

$$\epsilon_t = \sqrt{\frac{3\ln t}{2n}}$$

Plus t est grand, plus on est sûr que $\mu \leq \hat{\mu} + \epsilon_t$. Cependant, l'intervalle de confiance ϵ_t augmente, mais de façon raisonnable (logarithmique)

Borne sur le regret de UCB

Théorème

Le regret de la politique UCB sur T tirages est **logarithmique en** T

$$Regret(T) \leq \left[\sum_{\Delta_i \neq 0} \frac{6}{\Delta_i}\right] \ln T + K\left(1 + \frac{\pi^2}{3}\right) = O(\ln T)$$

- Il suffit de montrer que $\mathbb{E}[N_T(i)] \leq \frac{6\ln T}{\Delta_i^2} + 1 + \frac{\pi^2}{3}$ pour $\Delta_i \neq 0$
- Calculons le nombre de tirages du bras $i \neq i^*$: $N_T(i) = \sum_{t=1}^{T} \mathbb{I}(A(t) = i)$

  $N_T(i) \leq n +$ nb de tirages après n tirages

$$\leq n + \sum_{t=n+1}^{T} \mathbb{I}(A(t) = i,\ N_{t-1}(i) \geq n)$$

$$\leq n + \sum_{t=n+1}^{T} \mathbb{I}(\underbrace{\hat{\mu}_{t-1}(i^*) + C_{t-1,N_{t-1}(i^*)} \leq \hat{\mu}_{t-1}(i) + C_{t-1,N_{t-1}(i)}}_{\text{nécessaire pour avoir } A(t)=i},\ N_{t-1}(i) \geq n)$$

$$\leq n + \sum_{t=n}^{T-1} \mathbb{I}(\hat{\mu}_t(i^*) + C_{t,N_t(i^*)} \leq \hat{\mu}_t(i) + C_{t,N_t(i)},\ N_t(i) \geq n)$$

Avec $C_{t,N} = \sqrt{\frac{3\ln t}{2N}}$

Borne sur le regret de UCB I

- $\hat{\mu}_{N_t(i^*)}(i^*) + C_{t,N_t(i^*)} \leq \hat{\mu}_{N_t(i)}(i) + C_{t,N_t(i)}$ nécessite soit $\mu(i^*)$ très sous-estimé, soit $\mu(i)$ très sur-estimé, soit une petite différence Δ_i :

$$\hat{\mu}_{N_t(i^*)}(i^*) \leq \mu(i^*) - C_{t,N_t(i^*)} \quad (1)$$

$$\hat{\mu}_{N_t(i)}(i) \geq \mu(i) + C_{t,N_t(i)} \quad (2)$$

$$\Delta_i = \mu(i^*) - \mu(i) < 2C_{t,N_t(i)} \quad (3)$$

  Par ex., si pas (1) ni (3), $\hat{\mu}_{N_t(i^*)}(i^*) + C_{t,N_t(i^*)} > \mu(i^*) \geq \mu(i) + 2C_{t,N_t(i)}$ et il faut $\hat{\mu}_{N_t(i)}(i) + C_{t,N_t(i)} \geq \mu(i) + 2C_{t,N_t(i)} \Leftrightarrow$ (2) ; ...

- (3) $\Rightarrow N_t(i) < 6\ln t/\Delta_i^2 \leq 6\ln T/\Delta_i^2$, donc avec $N_t(i) \geq n = 6\ln T/\Delta_i^2 + 1$, il faut (1) ou (2) pour avoir $A(t+1) = i$
- Avec la linéarité de l'espérance et la propriété $\mathbb{E}[\mathbb{I}(E)] = P(E)$:

$$\mathbb{E}[N_T(i)] \leq \mathbb{E}\left[n + \sum_{t=n}^{T-1} \mathbb{I}\left((1)\ ou\ (2)\right)\right]$$

$$\leq n + \sum_{t=n}^{T-1} \mathbb{E}\left[\mathbb{I}\left((1)\ ou\ (2)\right)\right]$$

$$\leq n + \sum_{t=n}^{T-1} P\left((1)\ ou\ (2)\right)$$

Borne sur le regret de UCB II

- "Union bound" :

$$P\left((1)\ ou\ (2)\right) \leq P(1) + P(2)$$

$$P(1) \leq P\left(\bigcup_{N=1}^{t} \{(1), N_t(i^*) = N\}\right) = \sum_{N=1}^{t} P\left((1), N_t(i^*) = N\right)$$

- Inégalité de Hoeffding : $P\left((1), N_t(i^*) = N\right) \leq e^{-2NC_{t,N}^2}$

$$\Rightarrow \quad P(1) \leq \sum_{N=1}^{t} e^{-2NC_{t,N}^2} = \sum_{N=1}^{t} e^{-3\ln t} = \sum_{N=1}^{t} t^{-3} = t^{-2}$$

- De même, $P(2) \leq t^{-2}$, donc

$$\mathbb{E}[N_T(i)] \leq \frac{6\ln T}{\Delta_i^2} + 1 + \sum_{t=n}^{T-1} 2t^{-2}$$

- La série peut être bornée par une constante (grâce au problème de Bâle) :

$$\sum_{t=n}^{T-1} 2t^{-2} \leq 2\sum_{t=1}^{\infty} \frac{1}{t^2} = \frac{\pi^2}{3}$$

- Donc

$$\mathbb{E}[N_T(i)] \leq \frac{6\ln T}{\Delta_i^2} + 1 + \frac{\pi^2}{3} = O\left(\frac{6\ln T}{\Delta_i^2}\right) = O(\ln T)$$

L'intelligence artificielle face à l'incertain

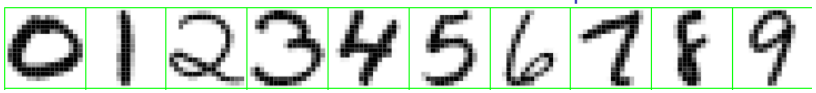
Deux grands cas de figure

L'incertain dont on ne pourra jamais être sûr du résultat mais que l'on connaît bien

- Exemple : on "sait" que le résultat d'un lancer de dé est incertain et on "sait" (on arrive facilement à se convaincre que cela est assez proche de la réalité) que le ⚀ va sortir avec une probabilité $1/6$
- On connaît avec "certitude" un modèle de l'incertain (basé sur des probabilités)
- Mais on ne pourra jamais prédire le résultat avec certitude

$\Rightarrow$ Modéliser efficacement l'incertitude à partir de nos connaissances

L'incertain dont le résultat est certain mais sur lequel on ne sait rien



- Exemple : ces images représentent des chiffres de manière certaine, mais on ne sait pas comment, on ne connaît rien sur la fonction f dans $f(\textit{image}) = \textit{chiffre}$
- On ne sait pas comment modéliser le résultat d'une expérience
- Mais on sait que résultat n'est pas (ou peu) aléatoire

$\Rightarrow$ Apprendre une bonne approximation de f à partir d'exemples

Modéliser l'incertain

Réseaux bayésiens

- Permettent de modéliser des phénomènes aléatoires complexes impliquant de nombreuses variables aléatoires
  Une loi jointe sur n variables à m valeurs possibles s'exprime a priori avec $O(m^n)$ nombres
- Par une approche graphique mettant en valeur les dépendances entre variables et limitant la taille de la représentation

Rappels de probabilités

Couple de v.a. discrètes $(X, Y) \in \mathcal{X} \times \mathcal{Y}$, $|\mathcal{X}| < \infty, |\mathcal{Y}| < \infty$

- Loi jointe du couple : pour $A \subseteq \mathcal{X} \times \mathcal{Y}$

$$P_{X,Y}(A) = P((X, Y) \in A) = \sum_{(x,y)\in A} P(X = x, Y = y)$$

- Espérance :

$$\mathbb{E}_{(X,Y)}[f(X, Y)] = \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} f(x, y)\, P(X = x, Y = y)$$

Couple de v.a. continues $(X, Y) \in \mathcal{X} \times \mathcal{Y} \subseteq \mathbb{R}^2$

- Densité de la loi jointe : $p_{X,Y}(x, y)$
- Loi jointe du couple : pour $A \subseteq \mathcal{X} \times \mathcal{Y}$

$$P_{X,Y}(A) = P((X, Y) \in A) = \int_A p_{X,Y}(x, y)\, dxdy$$

- Espérance : $\mathbb{E}_{(X,Y)}[f(X, Y)] = \iint_{\mathbb{R}^2} f(x, y)\, p_{X,Y}(x, y)\, dxdy$

Rappels de probabilités

Probabilités conditionnelles

- Probabilité de A sachant B : $P(A|B) = \frac{P(A,B)}{P(B)}$ (si $P(B) \neq 0$)
- Pour des v.a. discrètes : $P(X = x|Y = y) = \frac{P(X=x,Y=y)}{P(Y=y)}$
- Pour des v.a. continues : $p_{X|Y}(x|y) = \frac{p_{X,Y}(x,y)}{p_Y(y)}$

Factorisation d'une loi jointe

- En discret :

  $P(X = x, Y = y) = P(X = x|Y = y)P(Y = y) = P(Y = y|X = x)P(X = x)$
- En continu : $p_{X,Y}(x, y) = p_{X|Y}(x|y)\, p_Y(y) = p_{Y|X}(y|x)\, p_X(x)$

Lois marginales et probabilités totales

- En discret :

$$P(X = x) = \sum_{y\in\mathcal{Y}} P(X = x, Y = y) = \sum_{y\in\mathcal{Y}} P(X = x|Y = y)P(Y = y)$$

Rappels de probabilités

Indépendance et indépendance conditionnelle

- X et Y sont indépendantes si et seulement si

$$P(X = x, Y = y) = P(X = x)P(Y = y)$$

- X et Y sont **conditionnellement indép.** sachant Z si et seulement si

$$P(X = x, Y = y|Z = z) = P(X = x|Z = z)P(Y = y|Z = z)$$

- Si X et Y sont conditionnellement indép. sachant Z, on a aussi

$$P(X = x|Y = y, Z = z) = P(X = x|Z = z)$$

Car

$$\begin{aligned} P(X = x|Y = y, Z = z) &= \frac{P(X = x, Y = y, Z = z)}{P(Y = y, Z = z)} = \frac{P(X = x, Y = y|Z = z)P(Z = z)}{P(Y = y, Z = z)} \\ &= \frac{P(X = x, Y = y|Z = z)P(Z = z)}{P(Y = y|Z = z)P(Z = z)} \\ &= \frac{P(X = x, Y = y|Z = z)}{P(Y = y|Z = z)} = \frac{P(X = x|Z = z)P(Y = y|Z = z)}{P(Y = y|Z = z)} \\ &= P(X = x|Z = z) \end{aligned}$$

Réseaux bayésiens

Vecteurs aléatoires

- $\boldsymbol{X} \in \mathbb{R}^n$ est un vecteur aléatoire contenant n v.a. continues
- $\boldsymbol{X} \in \{0, 1\}^n$ est un vecteur aléatoire binaire contenant n v.a. binaires
- La loi de $\boldsymbol{X}$ est la loi de jointe de ses composantes :

$$P(\boldsymbol{X} = \boldsymbol{x}) = P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$$

Représenter des lois jointes en grande dimension

- Pour des vecteurs aléatoires binaires, $P(\boldsymbol{X} = \boldsymbol{x})$ peut s'écrire comme un tableau de $2^n - 1$ lignes contenant $2^n - 1$ paramètres

x_1	x_2	$\ldots$	x_n	$P(\boldsymbol{X} = (x_1, x_2, \ldots, x_n))$
0	0		0	0.1
0	0		1	0.8
$\vdots$				$\vdots$

- Difficile à implémenter pour n grand
- Besoin d'inclure des hypothèses/contraintes supplémentaires

Réseaux bayésiens

Représentation graphique d'une loi jointe

- Chaque nœud correspond à une variable aléatoire
- Les arcs orientés déterminent les (in)dépendances conditionnelles
  chaque v.a. est conditionnellement indépendante de ses prédécesseurs sachant ses parents :
  $P(X_i = x_i|\boldsymbol{X}_{1:i-1} = \boldsymbol{x}_{1:i-1}) = P(X_i = x_i|\boldsymbol{X}_{parents(X_i)} = \boldsymbol{x}_{parents(X_i)})$

Factorisation d'une loi jointe par un réseau bayésien

$$P(\boldsymbol{X} = \boldsymbol{x}) = \prod_{i=1}^{n} P(X_i = x_i \mid \boldsymbol{X}_{parents(X_i)} = \boldsymbol{x}_{parents(X_i)})$$

Si les X_i sont ordonnées de telle sorte que $j \in parents(X_i) \Rightarrow j < i$,

$$\begin{aligned} P(\boldsymbol{X} = \boldsymbol{x}) &= P(X_n = x_n|\boldsymbol{X}_{1:n-1} = \boldsymbol{x}_{1:n-1})P(\boldsymbol{X}_{1:n-1} = \boldsymbol{x}_{1:n-1}) \\ &= P(X_n = x_n|\boldsymbol{X}_{1:n-1} = \boldsymbol{x}_{1:n-1})P(X_{n-1} = x_{n-1}|\boldsymbol{X}_{1:n-2} = \boldsymbol{x}_{1:n-2})P(\boldsymbol{X}_{1:n-2} = \boldsymbol{x}_{1:n-2}) \\ &= \ldots \\ &= \prod_{i=1}^{n} P(X_i = x_i|\boldsymbol{X}_{1:i-1} = \boldsymbol{x}_{1:i-1}) = \prod_{i=1}^{n} P(X_i = x_i|\boldsymbol{X}_{parents(X_i)} = \boldsymbol{x}_{parents(X_i)}) \end{aligned}$$

avec $1 : 0 = \emptyset$ et $parents(X_1) = \emptyset$

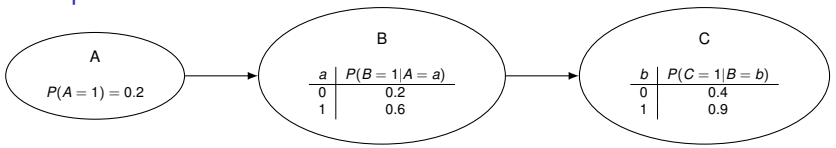
Réseaux bayésiens

Table de probabilités conditionnelles

- Chaque nœud correspond à une variable aléatoire X_i
- On associe une table de probabilités conditionnelles à chaque nœud pour définir

$$P(X_i = x_i \mid \boldsymbol{X}_{parents(X_i)} = \boldsymbol{x}_{parents(X_i)})$$

Exemple 1 : réseau en forme de chaîne avec v.a. binaires

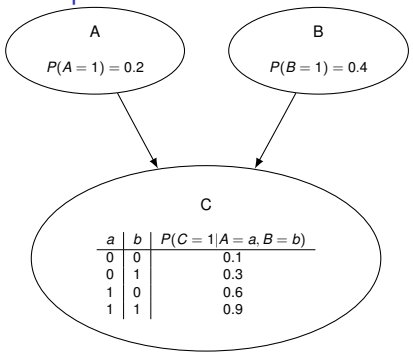


- 5 nombres suffisent ici pour coder la loi jointe de 3 variables binaires au lieu de $2^3 - 1 = 7$

$$\begin{aligned} P(A = 0, B = 1, C = 1) &= P(C = 1|B = 1)P(B = 1|A = 0)P(A = 0) \\ &= 0.9 \times 0.2 \times 0.8 = 0.144 \end{aligned}$$

Réseaux bayésiens – exemples avec des v.a. binaires

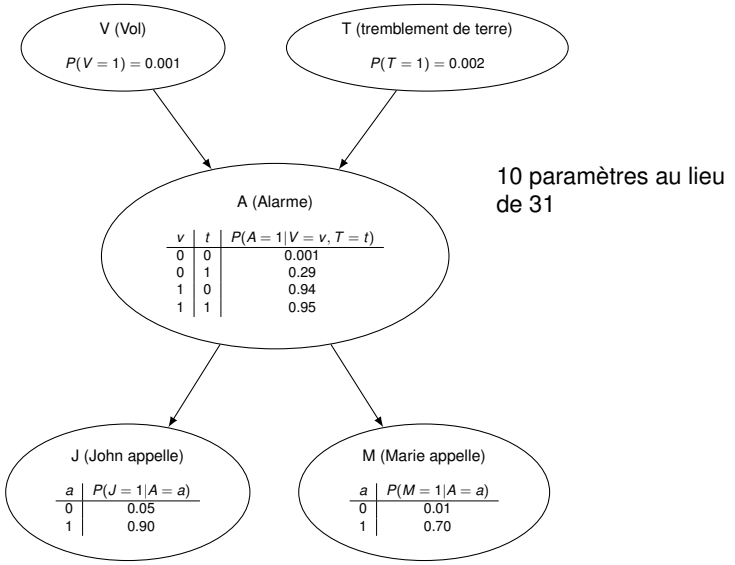
Exemple 2 : réseau en forme de V



- 6 nombres suffisent ici pour coder la loi jointe de 3 variables binaires au lieu de $2^3 - 1 = 7$

$$P(A=0,B=1,C=1) = P(C=1|A=0,B=1)P(A=0)P(B=1)$$
$$= 0.3 \times 0.8 \times 0.4 = 0.288$$

Réseaux bayésiens



10 paramètres au lieu de 31

$P(V=1,T=0,A=1,J=1,M=0)$
$= P(V=1)P(T=0)P(A=1|V=1,T=0)P(J=1|A=1)P(M=0|A=1)$

Réseaux bayésiens

Construction d'un réseaux

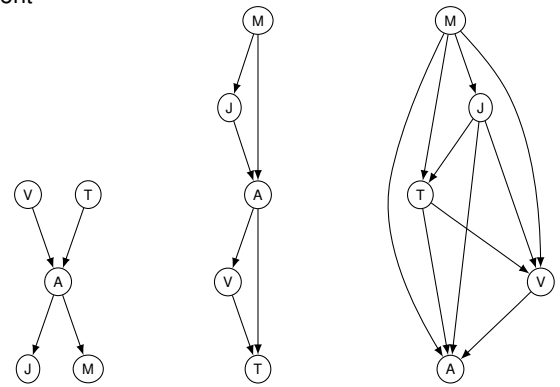
- Plusieurs topologies possibles pour la même loi jointe
- Certaines plus complexes et moins intuitives
- Dans un réseau simple, les arcs modélisent les liens de **cause à effet directs**
- Commencer le réseaux avec les causes premières (les événements qui semblent indépendants du reste)
- Ajouter les nœuds un par un dans l'ordre des causes à effets et en créant un jeu minimal d'arcs

Procédure itérative

1. Choisir l'ensemble des variables et les ordonner : $X_1, X_2, \ldots, X_n$
2. Pour $i = 1, \ldots, n$
   - 2.1 ajouter un nœud pour la variable X_i
   - 2.2 ajouter des arcs en provenance des $parents(X_i)$ **déjà présents dans le réseau** en respectant les hypothèses d'indépendances conditionnelles $P(X_i = x_i | \boldsymbol{X}_{parents(X_i)} = \boldsymbol{x}_{parents(x_i)}) = P(X_i = x_i | \boldsymbol{X}_{1:i-1} = \boldsymbol{x}_{1:i-1})$
   - 2.3 remplir la table des probabilités conditionnelles de X_i
     valeurs de $P(X_i = x_i | \boldsymbol{X}_{parents(X_i)} = \boldsymbol{x}_{parents(x_i)})$ pour tout $\boldsymbol{x}_{parents(x_i)}$

Réseaux bayésiens

Exemple de topologies obtenues en ajoutant les nœuds dans un ordre différent



V, T, A, J, M
10 paramètres

M, J, A, V, T
?? paramètres

M, J, T, V, A
?? paramètres

D-séparation et indépendances conditionnelles

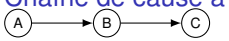
Notations

- $A \perp B$: A et B sont indépendantes
- $A \perp B \mid C$: A et B sont conditionnellement indépendantes sachant C

- Evidence $\boldsymbol{E} = \boldsymbol{e}$: ensemble de v.a. dont la valeur est connue / observée
- Comment savoir si $A \perp B \mid \boldsymbol{E}$?

- d-séparation : notion de séparation "directionnelle" ou "orientée"
- A et B sont d-séparées par $\boldsymbol{E} \Rightarrow A \perp B \mid \boldsymbol{E}$

Chaîne de cause à effet

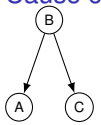


- $A \perp B$? Non
- $A \perp C$? Non
- $A \perp C \mid B$? Oui

$$\begin{aligned} P(A=a, C=c|B=b) &= \frac{P(A=a, B=b, C=c)}{P(B=b)} \\ &= \frac{P(A=a)P(B=b|A=a)P(C=c|B=b)}{P(B=b)} \\ &= P(A=a|B=b)P(C=c|B=b) \end{aligned}$$

- Après avoir observé B, A n'a plus d'influence sur ma croyance en C

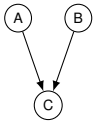
Cause commune



- $A \perp C$? Non
- $A \perp C|B$? Oui

$$\begin{aligned} P(A=a, C=c|B=b) &= \frac{P(A=a, B=b, C=c)}{P(B=b)} \\ &= \frac{P(A=a|B=b)P(B=b)P(C=c|B=b)}{P(B=b)} \\ &= P(A=a|B=b)P(C=c|B=b) \end{aligned}$$

Effet commun (structure en V)



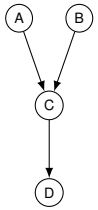
- $A \perp B$? Oui

$$\begin{aligned} P(A=a, B=b) &= \sum_c P(A=a, B=b, C=c) \\ &= P(A=a)P(B=b)\sum_c P(C=c|A=a, B=b) \\ &= P(A=a)P(B=b) \end{aligned}$$

- $A \perp B|C$? NON : après avoir observé C, A et B deviennent liées
  Par exemple : A = il fait beau ? B = j'ai gagné au loto ? C = je suis content ?
- Utile pour le diagnostic
  Par exemple : A = panne de batterie ? B = panne d'essence ? C = voiture ne démarre pas ?

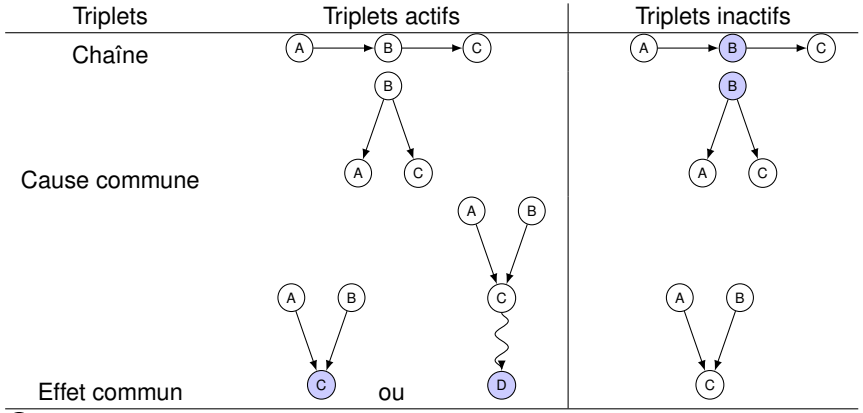
Réseaux bayésiens – triplets élémentaires

Effet commun (structure en V) observé "à distance"



- $A \perp B$? Oui
- $A \perp B|C$? Non
- $A \perp B|D$? Non : observer D me donne de l'information sur C qui me permet de lier A et B
- Exemples :
  A = note $<$ 6 en IA ? B = $<$ 6 en LMC ? C = semestre non validé ? D = étudiant au rattrapage de Réseau ?
  A = câble branché ? B = IP configurée ? C = connexion internet fonctionnelle ? D = page web s'affiche ?

Réseaux bayésiens – triplets élémentaires



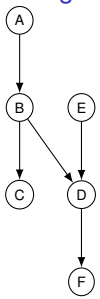
B : évidence (nœud observé)

Cas général : $X \perp Y|Z$?

- $X \perp Y|Z$ si et seulement si il n'existe pas de chemin (indépendamment de l'orientation des arcs) actif entre X et Y
- Chemin actif = chemin le long duquel tous les triplets sont actifs

Réseaux bayésiens

Cas général

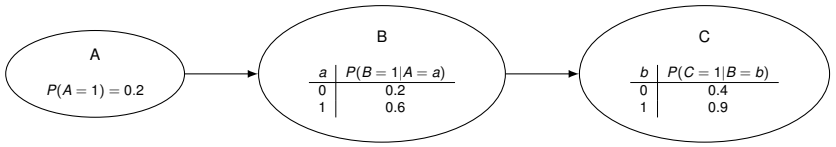


- $A \perp F$?
- $A \perp F|D$?
- $A \perp E$?
- $A \perp E|D$?
- $A \perp E|F$?
- $A \perp E|B, D$?

Réseaux bayésiens

Inférence

- Calcul d'une probabilité conditionnelle pour une certaine **évidence**
- Evidence = ensemble de variables à valeurs connues / observées



- $P(B=1|A=1)$?$=0.6 \rightarrow$ lecture directe depuis la table
- $P(C=1|A=1)$?$\rightarrow$ utilisation des probabilités totales
  $= P(C=1, B=0|A=1) + P(C=1, B=1|A=1)$
  $= P(C=1|A=1, B=0)P(B=0|A=1) + P(C=1|A=1, B=1)P(B=1|A=1)$
  $= P(C=1|B=0)P(B=0|A=1) + P(C=1|B=1)P(B=1|A=1)$

Règle de Bayes

$$P(X=x|Y=y) = \frac{P(Y=y|X=x)P(X=x)}{P(Y=y)}$$

- $P(A=1|B=1)$?$= \frac{0.6\times0.2}{P(B=1)} = \frac{0.12}{P(B=1|A=1)P(A=1)+P(B=1|A=0)P(A=0)} = \frac{0.12}{0.12+0.2\times0.8} = \frac{0.12}{0.28} \approx 0.43$

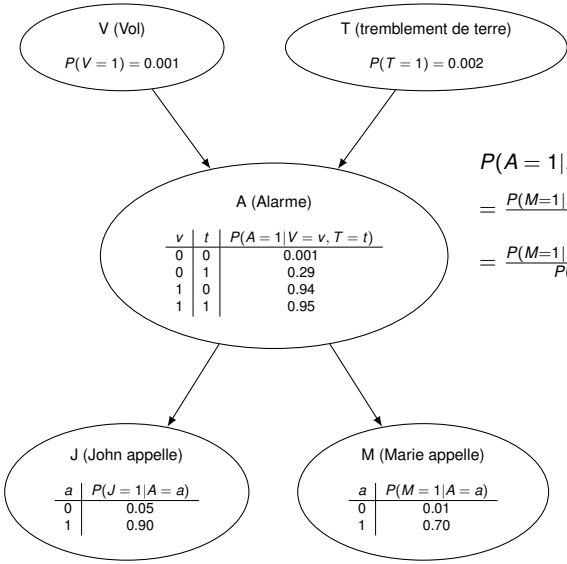
Inférence en général

- Calcul d'une probabilité marginale : $P(X=1) = \sum_{\boldsymbol{v}} P(X=1, \boldsymbol{V}=\boldsymbol{v})$
- Calcul de $P(X=1|\boldsymbol{E}=\boldsymbol{e})$
- Si $\mathit{parents}(X) \subseteq \boldsymbol{E} \subseteq \mathit{predecesseurs}(X)$, alors lire la table des probabilités cond. de X
- Sinon, il y a d'autres variables $\boldsymbol{V}$ et on passe par la loi jointe

$$\begin{aligned} P(X=1|\boldsymbol{E}=\boldsymbol{e}) &= \frac{P(X=1,\boldsymbol{E}=\boldsymbol{e})}{P(\boldsymbol{E}=\boldsymbol{e})} = \sum_{\boldsymbol{v}} \frac{P(X=1,\boldsymbol{V}=\boldsymbol{v},\boldsymbol{E}=\boldsymbol{e})}{P(\boldsymbol{E}=\boldsymbol{e})} \\ &= \sum_{\boldsymbol{v}} P(X=1,\boldsymbol{V}=\boldsymbol{v}|\boldsymbol{E}=\boldsymbol{e}) \\ &= \sum_{\boldsymbol{v}} P(X=1|\boldsymbol{V}=\boldsymbol{v},\boldsymbol{E}=\boldsymbol{e}) P(\boldsymbol{V}=\boldsymbol{v}|\boldsymbol{E}=\boldsymbol{e}) \end{aligned}$$

- La règle de Bayes permet d'inverser une condition :

$$P(X=x|(\boldsymbol{E}_1,\boldsymbol{E}_2)=(\boldsymbol{e}_1,\boldsymbol{e}_2)) = \frac{P(\boldsymbol{E}_1=\boldsymbol{e}_1|X=x,\boldsymbol{E}_2=\boldsymbol{e}_2)P(X=x|\boldsymbol{E}_2=\boldsymbol{e}_2)}{P(\boldsymbol{E}_1=\boldsymbol{e}_1|\boldsymbol{E}_2=\boldsymbol{e}_2)}$$



$$P(A=1|M=1,V=1) = \frac{P(M=1|A=1,V=1)P(A=1|V=1)}{P(M=1|V=1)} = \frac{P(M=1|A=1)P(A=1|V=1)}{P(M=1|V=1)}$$

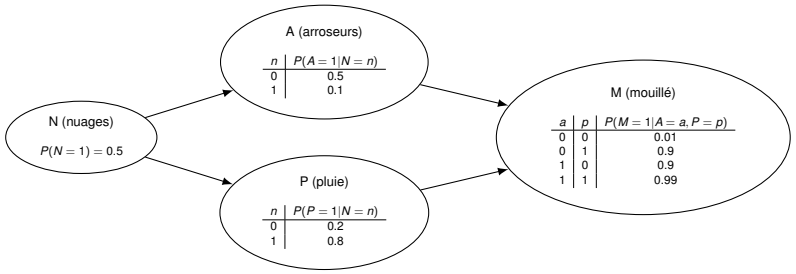
$$\begin{aligned} P(M=1|V=1) &= P(M=1|V=1,A=1)P(A=1|V=1) + P(M=1|V=1,A=0)P(A=0|V=1) \\ &= P(M=1|A=1)P(A=1|V=1) + P(M=1|A=0)P(A=0|V=1) \end{aligned}$$

$$P(A=1|V=1) = P(A=1|V=1,T=0)P(T=0) + P(A=1|V=1,T=1)P(T=1)$$

$$P(A=0|V=1) = 1 - P(A=1|V=1)$$

Inférence approximative

- Inférence exacte souvent trop complexe (NP-difficile)
- Approximation par échantillonnage (en commençant par les nœuds dont les parents sont inexistants ou déjà échantillonnés)
- $P(\boldsymbol{X}=\boldsymbol{x}) = \mathbb{E}[\mathbb{I}(\boldsymbol{X}=\boldsymbol{x})] \approx \frac{\text{nb d'échantillons avec } \boldsymbol{X}=\boldsymbol{x}}{\text{nb total d'échantillons}}$

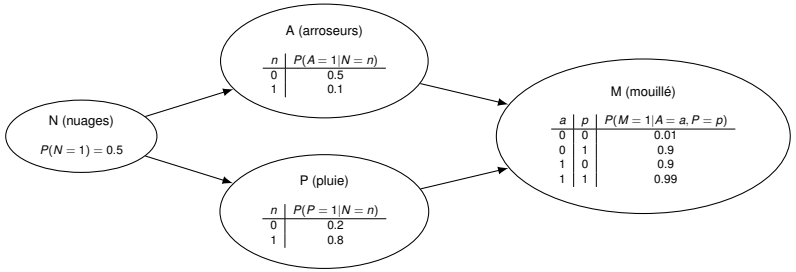


$P(N=1,A=0,P=1,M=1)$?

Inférence approximative de lois conditionnelles

- $P(X=x|\boldsymbol{E}=\boldsymbol{e}) \approx \frac{\text{nb d'échantillons avec } X=x \text{ et } \boldsymbol{E}=\boldsymbol{e}}{\text{nb d'échantillons avec } \boldsymbol{E}=\boldsymbol{e}}$
  $P(M=1|N=0)$?
- On rejette certains échantillons et on perd beaucoup de temps (surtout si $\boldsymbol{E}=\boldsymbol{e}$ est rare)
  $P(P=1|A=1,M=1)$?
- Pondération par la vraisemblance : forcer l'évidence $\boldsymbol{E}=\boldsymbol{e}$ au lieu de tirer aléatoirement $\boldsymbol{E}$ et pondérer les échantillons par la vraisemblance v_i d'avoir $\boldsymbol{E}=\boldsymbol{e}$ pour chaque échantillon i :

$$P(X=x|\boldsymbol{E}=\boldsymbol{e}) \approx \frac{\sum_i v_i \mathbb{I}(X=x)}{\sum_i v_i}$$



Apprentissage automatique

Définition de A. Samuel (1959)

Apprentissage automatique (*Machine Learning*) : champ d'étude donnant aux ordinateurs la capacité d'apprendre sans être explicitement programmés

Apprentissage par l'exemple

- Méthode privilégiée par l'apprentissage automatique
- Exemple : reconnaissance de chiffres manuscrits

Définition

Apprendre = Acquérir la capacité de généraliser à partir d'exemples
Généraliser = Prédire correctement un phénomène non observé auparavant

Apprendre par cœur ne permet pas de généraliser

- Exemple : apprendre les préférences d'un utilisateur pour les touches du clavier

Motivations / Applications

Vision par ordinateur

- Reconnaissance de formes (ex. codes postaux)
- Suivi de trajectoires
- Segmentation d'images / de vidéo

Bioinformatique

- Prédiction de la structure secondaire des protéines
- Philogénie
- . . .

Biomédical

- Aide au diagnostic, pronostics
- Imagerie médicale

Physique / ingénierie / automatique

- Modélisation de processus complexes
- Simulation temps réel
- Lois de commandes

Informatique

- Détection d'intrusion
- Détection de SPAM
- Applications web (propositions orthographiques, détection de langue, publicité ciblée)

Robotique

- Perception artificielle
- Prise de décisions

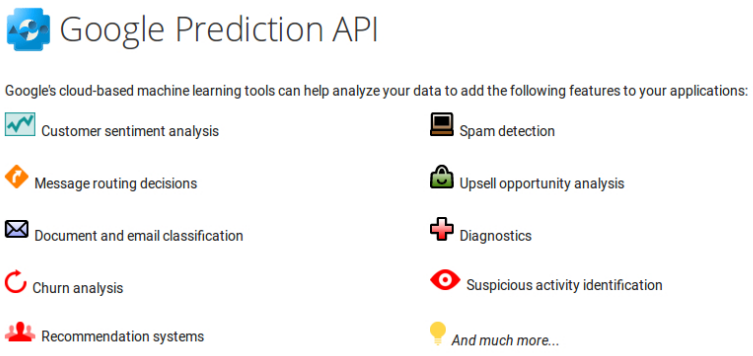
Fouille de données

- Extraction de connaissances

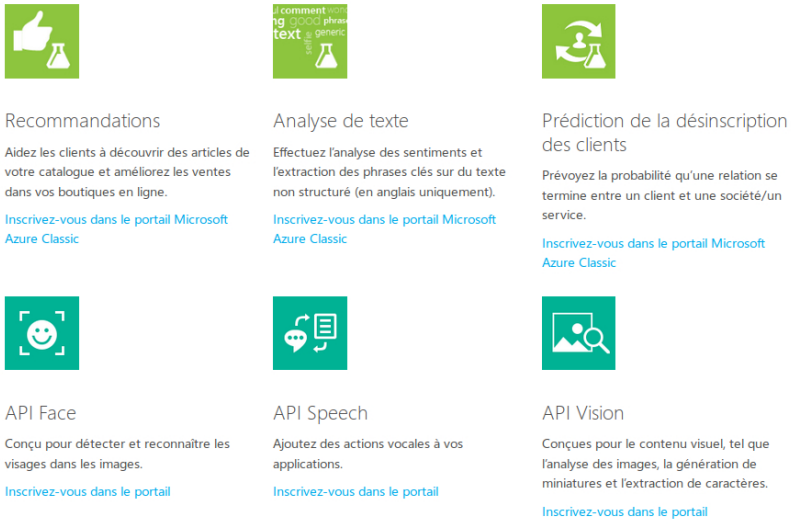
Séries temporelles

- Finance
- Météo
- Charge du réseau (EDF)

Google Prediction API



Microsoft Azure Machine Learning API



Représentation des exemples

Un exemple est un couple $(\boldsymbol{x}, y) \in \mathcal{X} \times \mathcal{Y}$

Espace de représentation : $\mathcal{X}$

- Les objets pour lesquels on souhaite prédire quelque chose sont représentés par un ensemble de descripteurs (*features*) ou attributs

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_d]^T \in \mathcal{X} \subseteq \mathbb{R}^d$$

- d : dimension de l'espace de représentation

Ensemble des étiquettes possibles : $\mathcal{Y}$

Etant donné la représentation $\boldsymbol{x}$ d'un objet, on cherche à prédire son **étiquette** $y \in \mathcal{Y}$

- Discrimination (ou classification) binaire : $\mathcal{Y} = \{-1, +1\}$
- Discrimination multi-classe : $\mathcal{Y} = \{1, \ldots, Q\}$
- Régression : $\mathcal{Y} = \mathbb{R}$

Le choix des descripteurs est pertinent si ces variables sont liées à l'étiquette

De quoi dispose-t-on pour prédire ?

Ensemble des exemples disponibles S

$$S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$$

- S est la **base d'apprentissage**
- m : taille de la base d'apprentissage = nombre d'exemples disponibles
- S contient (souvent) les seules informations disponibles

Modèle de prédiction f

$$f : \mathcal{X} \to \mathcal{Y}$$

- Prédire = calculer la sortie $f(\boldsymbol{x})$ du modèle
- $\mathcal{F}$: ensemble des modèles possibles ($f \in \mathcal{F}$)
  - $\mathcal{F}$ est une classe de fonctions (= famille de fonctions = ensemble de fonctions)
  - La recherche du modèle de prédiction f est limitée à l'ensemble $\mathcal{F}$

Comment apprendre à prédire ?

Algorithme d'apprentissage $\mathcal{A}$

$$\mathcal{A} : (\mathcal{F}, S) \mapsto \hat{f} \in \mathcal{F}$$

Apprendre = sélectionner un modèle $\hat{f}$ parmi les modèles f de $\mathcal{F}$ après observation d'un ensemble d'exemples S

- But : *sélectionner un modèle de prédiction $\hat{f}$ capable de "prédire correctement" l'étiquette y pour tout $\boldsymbol{x} \in \mathcal{X}$ et pas uniquement pour les exemples de la base d'apprentissage*

Généralisation

$$\forall\, \boldsymbol{x} \in \mathcal{X}, \quad \hat{f}(\boldsymbol{x}) \stackrel{?}{=} y$$

- Généralisation : capacité de $\hat{f}$ à donner des prédictions $\hat{f}(\boldsymbol{x})$ proches de y pour $\boldsymbol{x} \notin S$

Paramètres et hyperparamètres

Paramètres

- Ensemble des variables paramétrant les fonctions f d'une classe $\mathcal{F}$ donnée
- Exemples :

$$\mathcal{F}_1 = \left\{ f \;\middle|\; f(x) = x^\theta,\ \theta \in \{1, 2, 3\} \right\}$$

$$\mathcal{F}_2 = \left\{ f \;\middle|\; f(x) = \theta_1 x + \theta_2,\ (\theta_1, \theta_2) \in \mathbb{R}^2 \right\}$$

  Apprendre = déterminer les valeurs des paramètres θ

Hyperparamètres

- Ensemble des variables paramétrant un algorithme d'apprentissage ou une classe de fonctions $\mathcal{F}$
- Exemple : $\mathcal{F}^\gamma$ est l'ensembles des fonctions linéaires de x^γ

$$\mathcal{F}^\gamma = \{ f \mid f(x) = \theta x^\gamma,\ \theta \in \mathbb{R} \}$$

- Autre exemple : l'algorithme $\mathcal{A}^\gamma$ stoppe après γ itérations
- L'apprentissage des paramètres θ peut être réalisé pour différentes valeurs de γ
- Chaque valeur de γ conduit à un apprentissage et un modèle $\hat{f}$ différents

Comment valider l'apprentissage ?

Validation

- Etape destinée au **réglage des hyperparamètres** γ (sélection de modèle)
- Evaluation de la qualité d'une valeur de γ *après* l'apprentissage du modèle $\hat{f}$
- En pratique : évaluer la qualité du modèle $\hat{f}$ sur des données supplémentaires dites de validation

Test

- **Etape finale** de la procédure destinée à **estimer la qualité du modèle** $\hat{f}$ *après* réglage des éventuels hyperparamètres
- La qualité du modèle correspond à sa **capacité à généraliser**
- Cette estimation est indispensable pour déterminer si le modèle est utilisable et le risque lié à cette utilisation
- En pratique (dans les cas simples) : estimation de la qualité du modèle $\hat{f}$ sur des données supplémentaires dites de test

NE PAS utiliser les données de test pour régler les hyperparamètres
(cela conduirait à une surestimation des performances du modèle)

Modélisation probabiliste de l'apprentissage

Cadre probabiliste pour l'apprentissage

- L'espace $(\mathcal{X}, \mathcal{Y})$ est muni d'une mesure de probabilité P **inconnue**
- $(X, Y) \in (\mathcal{X}, \mathcal{Y})$: couple de variables aléatoires de loi P $(= P_{X,Y})$
- Jeu de données $S = \{(x_i, y_i)\}_{1 \leq i \leq m}$:
  m réalisations **indépendantes et identiquement distribuées** (i.i.d.) de (X, Y)
- Notation : X et x_i peuvent être des vecteurs de dimension d

Remarques

- **indépendantes** : chaque exemple (x_i, y_i) apporte le maximum d'information
- **identiquement distribuées** : chaque exemple (x_i, y_i) suit la loi P inconnue et toute observation future du couple (X, Y) est supposée suivre la même loi
  la base d'apprentissage est donc sensée être représentative du problème
- **Seules les régularités des données peuvent être apprises**
- **On ne peut pas prédire l'imprévisible**
  (ex : crack boursier)

Mesures de performance et apprentissage

Apprentissage d'un modèle de prédiction

- **Modèle de prédiction** : $f : \mathcal{X} \to \mathcal{Y}$
- **Algorithme d'apprentissage** : $\mathcal{A} : (\mathcal{F}, S) \mapsto \hat{f} \in \mathcal{F}$

Mesure de l'erreur de prédiction sur un exemple : la fonction de perte

- **Fonction de perte** $\ell : (\mathcal{F}, \mathcal{X}, \mathcal{Y}) \to \mathbb{R}^+$
- Propriété : $f(x) = y \;\Rightarrow\; \ell(f, x, y) = 0$

Le risque d'un modèle de prédiction

Risque de f = **erreur de généralisation** de f = espérance de la fonction de perte calculée en f :

$$R(f) = \mathbb{E}_{(X,Y)}\left[\ell(f, X, Y)\right]$$

Risque = erreur (mesurée par la fonction de perte) probable ou "en moyenne" sur un exemple tiré aléatoirement

But de l'apprentissage : minimiser le risque

Risque de f = **erreur de généralisation** de f :

$$R(f) = \mathbb{E}_{(X,Y)}\left[\ell(f, X, Y)\right]$$

Minimisation du risque

- Minimiser le risque revient à minimiser l'erreur probable de prédiction
- Apprentissage idéal : retenir le modèle f avec le risque $R(f)$ minimal
- Mais le risque $R(f)$ dépend de la loi P du couple (X, Y) qui est inconnue
- Approche : minimiser une estimation du risque

Estimer le risque à partir de données empiriques

- **Erreur d'apprentissage** (= erreur empirique = risque empirique) :

$$R_{emp}(f) = \frac{1}{m}\sum_{i=1}^{m} \ell(f, x_i, y_i)$$

- Moyenne *empirique* de la fonction de perte sur la base d'apprentissage

Principe de minimisation du risque empirique

Principe inductif ERM (*empirical risk minimization*)

- Choisir le modèle qui minimise $R_{emp}(f)$ au lieu de $R(f)$
- Soit l'algorithme :

$$\hat{f} = \arg\min_{f \in \mathcal{F}} R_{emp}(f)$$

$\hat{f}$: modèle de prédiction retenu par l'algorithme d'apprentissage parmi l'ensemble des modèles possibles $\mathcal{H}$

Remarques

- Les algorithmes d'apprentissage prennent souvent la forme de problèmes de minimisation
- On appelle *algorithme d'apprentissage* soit la définition du problème d'optimisation (ci-dessus) soit l'algorithme permettant de le résoudre
- Le principe ERM est simple mais malgré tout souvent efficace (si $\mathcal{F}$ est choisi correctement !)

Discrimination binaire : $\mathcal{Y} = \{-1, +1\}$

- Prédire Y revient à classer X ; f est appelé *classifieur*

Risque de classification

- Fonction indicatrice : $\mathbb{I}(A) = 1$ si A est observé, 0 sinon
- Fonction de perte standard : $\ell(f, X, Y) = \mathbb{I}(f(X) \neq Y)$
- **Risque = probabilité de mal classer un exemple** = "taux d'erreur" :

$$R(f) = \mathbb{E}_{(X,Y)}[\mathbb{I}(f(X) \neq Y)] = P(f(X) \neq Y)$$

- Taux de reconnaissance = $1 - P(f(X) \neq Y)$
- Sur la base d'apprentissage :
  taux de reconnaissance = $1 - R_{emp}(f) = 1 - \frac{1}{m}\sum_{i=1}^{m} \mathbb{I}(f(x_i) \neq y_i)$

Classifieur optimal : le classifieur de Bayes

Associer l'exemple à la classe la plus probable :

$$f_{Bayes}(x) = \begin{cases} +1, \text{ si } P(Y=1|X=x) \geq P(Y=-1|X=x) \\ -1, \text{ si } P(Y=1|X=x) < P(Y=-1|X=x) \end{cases}$$

Non applicable en pratique sans la connaissance de P

Régression : $\mathcal{Y} \subseteq \mathbb{R}$

- $X \in \mathbb{R}^d$: vecteur aléatoire continu (entrées, régresseurs)
- Y : v.a. continue (sortie)
- Loi jointe P à densité $p(x, y) = p(y|x)p(x)$

Risque de régression

- Fonction de perte quadratique : $\ell(f, X, Y) = (Y - f(X))^2$
- **Risque = erreur quadratique moyenne** (EQM ou MSE en anglais) :

$$R(f) = \mathbb{E}_{(X,Y)}[(Y - f(X))^2] = \int_{\mathcal{X}\times\mathcal{Y}} (y - f(x))^2 p(x, y)\, dxdy$$

Modèle optimal : la fonction de régression

La fonction de régression est l'espérance conditionnelle de Y sachant X :

$$\forall x \in \mathcal{X}, \quad f_{reg}(x) = \mathbb{E}_{Y|X=x}[Y|X = x]$$

Décomposition de l'erreur

Par rapport au meilleur modèle de la classe f^*

- Soit f^* la meilleure fonction de $\mathcal{F}$: $f^* = \arg\min_{f \in \mathcal{F}} R(f)$
- $R^* = \inf_f R(f)$: risque du modèle optimal
  (sans obligation d'avoir $f \in \mathcal{F}$)
- Le risque d'un modèle $\hat{f}$ sélectionné par un algorithme se décompose en

$$R(\hat{f}) - R^* = \underbrace{[R(f^*) - R^*]}_{\text{erreur d'approximation}} + \underbrace{\left[R(\hat{f}) - R(f^*)\right]}_{\text{erreur d'estimation}}$$

- **Erreur d'approximation** : erreur due au choix de $\mathcal{F}$ ($= 0$ si f_{Bayes} ou $f_{reg} \in \mathcal{F}$)
- **Erreur d'estimation** : erreur dépendant des données

Dilemme

Plus la classe de fonctions $\mathcal{F}$ est étendue, plus l'erreur d'approximation peut être faible, mais plus l'erreur d'estimation peut être grande
(il est plus difficile de trouver f^* dans un ensemble plus vaste de modèles)

Surapprentissage (*overfitting*)

ERM *a priori* efficace si le modèle optimal est dans $\mathcal{F}$, alors

Pourquoi restreindre la classe de fonctions $\mathcal{F}$?

- Imaginons un problème de discrimination binaire déterministe avec $Y = f_{Bayes}(X)$
- Sans contrainte sur $\mathcal{F}$, il est toujours possible de choisir f telle que pour une base d'apprentissage S:
  - $\forall (x_i, y_i) \in S,\ f(x_i) = y_i$
  - et $f(x) \neq f_{Bayes}(x)$ sur tous les autres $x \in \mathcal{X}$

  (*cas extrême d'apprentissage par cœur*)

  ce qui conduit à
  $$R_{emp}(f) = 0, \quad \text{et} \quad R(f) = 1$$

But : trouver le bon compromis conduisant à une erreur empirique faible sans surapprentissage

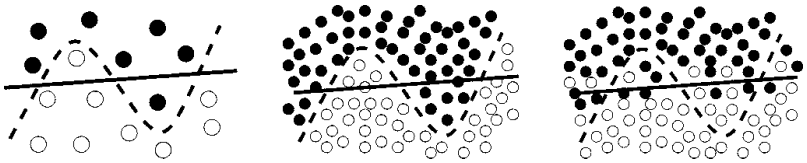
Surapprentissage (*overfitting*)

Le bon compromis

- Surapprentissage $\approx$ apprentissage d'une fonction trop complexe
- Trouver le compromis entre l'erreur d'apprentissage faible et la simplicité du classifieur

Difficile à trouver

- Dépend du problème
- Dépend des données



En pratique : trouver le bon compromis = régler un (ou plusieurs) hyperparamètres

Régression linéaire

Restreindre la classe de fonctions

- Pour éviter le surapprentissage ; et pour faciliter l'apprentissage
- Classe des modèles linéaires :

$$\mathcal{F} = \left\{ f \mid f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle = \boldsymbol{w}^T \boldsymbol{x} = \sum_{j=1}^{d} w_j x_j,\ \boldsymbol{w} \in \mathbb{R}^d \right\}$$

Méthode des moindres carrés

- Application directe du principe ERM : minimiser l'EQM empirique

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w} \in \mathbb{R}^d} \sum_{i=1}^{m} (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2$$

- Forme analytique de la solution :

$$\hat{\boldsymbol{w}} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}, \quad \text{avec } \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_m^T \end{bmatrix} \in \mathbb{R}^{m \times d},\ \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \in \mathbb{R}^m$$

- Pour les modèles affines, $\boldsymbol{x}$ contient une composante constante ($= 1$)

Classification linéaire

Classifieurs binaires ($\mathcal{Y} = \{-1, +1\}$) et linéaires

$$\mathcal{F} = \left\{ f \mid f(\boldsymbol{x}) = \text{signe}\left(\langle \boldsymbol{w}, \boldsymbol{x} \rangle\right) = \text{signe}\left(\boldsymbol{w}^T \boldsymbol{x}\right),\ \boldsymbol{w} \in \mathbb{R}^d \right\}$$

Apprentissage

- Dans le cas général : application directe du principe ERM "impossible" car la fonction de perte $\mathbb{I}(f(x_i) \neq y_i)$ est non convexe et non différentiable
  $\Rightarrow$ on considère souvent une approximation de la fonction de perte
- Dans le cas linéairement séparable ($\exists f \in \mathcal{F},\ R_{emp}(f) = 0$) : problème de programmation linéaire

  $$\forall (\boldsymbol{x}_i, y_i) \in S, \quad y_i \boldsymbol{x}_i^T \boldsymbol{w} > 0$$

  ou

  $$\forall (\boldsymbol{x}_i, y_i) \in S, \quad y_i \boldsymbol{x}_i^T \boldsymbol{w} \geq 1$$