Apprentissage Supervisé :
Le perceptron

Apprendre la fonction d'évaluation

- Définition de la fonction d'évaluation

$$Eval(s) = w_1\, f_1(s) + w_2\, f_2(s) + \ldots + w_n\, f_n(s)$$

  - Fonction dépendante ou non du temps (du n° de coup)

$$Eval(t, s) = w_1(t)\, f_1(s) + w_2(t)\, f_2(s) + \ldots + w_n(t)\, f_n(s)$$

- Apprendre les w ou les w(t)

Exploration locale

- A partir d'une situation donnée:
  - Modification d'un paramètre (+ ou – epsilon)
  - Estimation de la qualité de la configuration selon simulations
- ➢ Peu efficace :
  - Estimations coûteuses : nécessitent un grand nombre de simulations
  - Estimations peu fiables : large part d'aléatoire => mauvaises décisions probables
  - Interdépendance des paramètres => chaque décision sur un paramètre peut impliquer de reconsidérer tous les autres
  - Nombreux optima locaux

Apprentissage supervisé

- Principe
  - Apprentissage à partir d'une base d'exemples
    - On fournit un ensemble de situations étiquetées par un expert
      - Apprentissage automatique de la fonction d'évaluation *f(s)*
      - Algorithmes
        - Perceptron
        - Réseaux de neurones

Apprentissage à partir d'un ensemble d'apprentissage

- Base de coups idéaux E= {($etat_i$, $coup_i$), i=1..N}
  - On cherche la fonction *f* qui va choisir dans chaque situation $etat_i$ le coup $coup_i$
  - Il faut que l'ensemble d'apprentissage soit représentatif de toutes les situations
- Généralement algorithmes itératifs

- Pb : Avoir l'ensemble E
  - Plusieurs possibilités :
    1. Parser des parties diffusées sur Internet

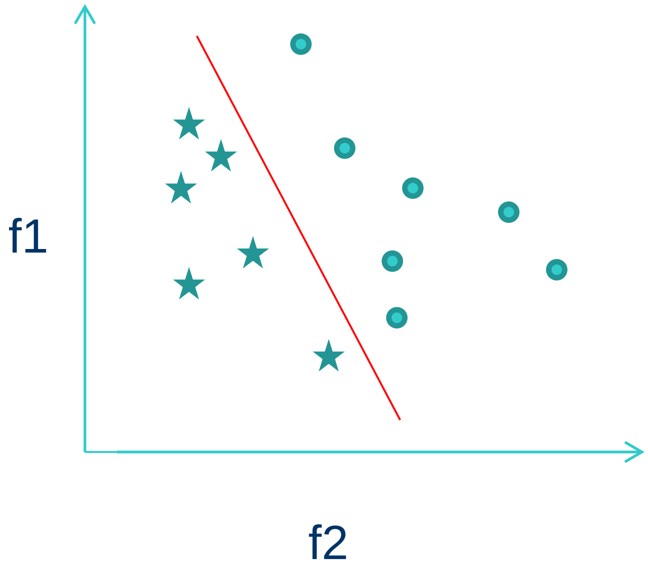
    1. Imiter des décisions prises par des algos plus fiables mais plus complexes => par exemple : choisir en H1 le coup qu'un minmax profondeur 6 aurait choisi

Apprentissage supervisé

- Classification
- Regression
- Ranking

Classification binaire

- On cherche à déterminer la frontière entre deux classes (+1 et -1) selon l'espace des caractéristiques des éléments à classer
  – Par exemple : trouver la frontière en bons et mauvais coups



Classification binaire

- Trouver la fonction f minimisant les erreurs de classification
- Dans le cas linéaire : $f(x) = \sum_i w_i * x_i + b$
- Avec deux classes -1 et +1, classe de x déterminée par $signe(f(x))$

- Problème : trouver w minimisant les erreurs de classification

Classification binaire

- Problème : trouver w minimisant les erreurs de classification
- Fonction de coût permettant de mesurer l'erreur de prédiction sur un exemple:

  $l(f(x), y) = \max(1 - f(x) * y, 0)$ (Hinge Loss)

$\Rightarrow$ Trouver w minimisant la somme des loss sur tous les exemples de la base d'apprentissage :

$w^* = argmin_w \sum_i l\left(f\left(x^i\right), y^i\right)$ avec y^i la classe de x^i

$$= argmin_w \sum_i \max(1 - (\sum_j x_j^i * w_j + b) * y^i, 0)$$

Régression

- Trouver la fonction f qui minimise pour tous les exemples d'apprentissage x, l'écart entre valeur observée et valeur prédite
- Par exemple coût moindres carrés :

$$l(f(x), y) = (f(x) - y)^2$$

- Dans le cas linéaire :

$$w^* = argmin_w \sum_i \left(f\left(x^i\right) - y^i\right)^2$$

Ranking (ordonnancement)

- Ordonnancement par paires : pour chaque paire d'élements (e1,e2) avec e1 devant être rangé avant e2, chercher la fonction f qui donne un score plus élevé à e1 qu'à e2
- Coût pairwise (avec e1 devant être rangé avant e2):

$$l\big(f(e1), f(e2)\big) = \max(1 - f(e1) + f(e2), 0)$$

$$w^* = argmin_w \sum_{x^i, x^j : y^i > y^j} l\left(f\left(x^i\right), f\left(x^j\right)\right)$$

Algorithme d’apprentissage linéaire : le perceptron

Le perceptron : classification

- L'algorithme du perceptron (2 classes)
- Sortie désirée = ± 1

Initialiser W

Répéter

Pour i = 1 à N

Si $y^i * (\sum_j x_j^i * w_j + w_0) \leq 1$

Pour j = 1 à M

$w_j \leftarrow w_j + \alpha * y^i x_j^i$

$w_0 \leftarrow w_0 + \alpha * y^i$

Jusqu'à convergence

- C'est un algorithme de correction d'erreur (par descente de gradient)
- α est le pas d'apprentissage (diminue en fonction du temps)
- Revient à résoudre: $\mathrm{w}^* = argmin_w \sum_i \max(1 - (\sum_j x_j^i * w_j + w_0) * y^i, 0)$

Le perceptron : adaptation à la regression (moindres carrés)

- Sortie désirée = valeur y

Initialiser W
Répeter
Pour i = 1 à N
Pour j = 1 à M

$$w_j \leftarrow w_j - \alpha * 2x_j^i(\sum_j x_j^i * w_j + w_0 - y^i)$$

$$w_0 \leftarrow w_0 - \alpha * 2 (\sum_j x_j^i * w_j + w_0 - y^i)$$

Jusqu'à convergence

- C'est un algorithme de correction d'erreur (par descente de gradient)
- α est le pas d'apprentissage (diminue en fonction du temps)
- Revient à résoudre: $w^* = argmin_w \sum_i ((\sum_j x_j^i * w_j + w_0) - y^i)^2$

Le perceptron : adaptation au ranking pairwise

- Sortie désirée = score respectant un ordonnancement prédéfini selon valeurs y

Initialiser W

Répeter

Pour i = 1 à N

Pour j = 1 à N

Si $y^i > y^j$ et $\sum_k x_k^i * w_k - \sum_k x_k^j * w_k \leq 1$

Pour k = 1 à M

$$w_k \leftarrow w_k - \alpha * (x_k^j - x_k^i)$$

Jusqu'à convergence

- C'est un algorithme de correction d'erreur (par descente de gradient)
- α est le pas d'apprentissage (diminue en fonction du temps)
- Revient à résoudre: $w^* = argmin_w \sum_{x^i, x^j : y^i > y^j} \max(1 - \sum_k x_k^i * w_k + \sum_k x_k^j * w_k, 0)$

Apprendre à imiter un joueur

Apprendre à imiter un joueur

- Oracle : joueur alpha beta profondeur > 1
- Joueur paramétrique
  - Cherche à apprendre à réaliser les mêmes choix que l’Oracle à profondeur 1 (entre les coups possibles pour le joueur courant)
- Application des poids appris dans un joueur à profondeur > 1 => évaluation des feuilles selon une fonction proche de ce qu’aurait fait un joueur de profondeur supérieure

Apprendre à imiter un joueur

- Classification : coup choisi par l’oracle étiqueté +1, les autres -1
- Regression : prédire le score d’évaluation déterminé par l’oracle pour les différents coups
- Ranking : donner un meilleur score au coup choisi par l’oracle (plus adapté à ce qu’on cherche à faire)

Apprendre à imiter un joueur

Pseudo – Algo :

Init W

Repeter

- Tant que pas fin de la partie

- Demander les estimations d'utilité des différents coups à l'Oracle et au joueur paramétrique

- Utiliser les évaluations de l'oracle pour mettre à jour W

- Faire jouer le joueur paramétrique

- Faire jouer l'adversaire

- Faire légèrement baisser le pas d'apprentissage α

Jusqu'à convergence

Apprendre à imiter un joueur

Mise à jour des poids :

$o =$ scores donnés par l'Oracle aux différents coups

opt = coup joué par l'oracle

Pour c = coup possible pour l'oracle tel que o(c) < o(opt)

$o = \sum_j w_j * sc_j(opt);$ $s = \sum_j w_j * sc_j(c);$

Si (o-s)<1

Pour tout j = 1 à N

$sc_j(opt)$ = score de opt selon la fonction j du joueur

$sc_j(c)$ = score de c selon la fonction j du joueur

$w_j \leftarrow w_j - \alpha * \left(sc_j(c) - sc_j(opt)\right)$