TD 9 : Boosting, RL

Exercice 1 – Boosting

Rappel de l'algorithme : on cherche à construire une combinaison de classifieurs faibles $f_T(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$ de manière itérative, de manière à prendre mieux en compte à une itération donnée les erreurs des itérations précédentes. Pour cela, une distribution de poids sur les exemples est considérée et adaptée à chaque itération afin d'augmenter le poids des exemples mal classés, et de baisser le poids des exemples bien classés. Soit $D_t = (w_t(1), \ldots, w_t(n))$ la distribution des poids des exemples au pas t, D_1 correspondant à la distribution uniforme. L'algorithme consiste en l'itération de la procédure suivante :

1. Choisir h_t qui minimise l'erreur selon D_t
2. Calculer l'erreur ϵ_t associé au classifieur h_t selon D_t
3. Fixer $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$
4. Mettre à jour D_{t+1} : $w_{t+1}(i) = \frac{1}{Z_t} w_t(i) e^{(-\alpha_t y_i h_t(x_i))}$, avec Z_t facteur de normalisation

Q 1.1 Introduction

Q 1.1.1 Rappeler le principe et les différences entre le boosting et le bagging. Soit le jeu de données suivant : $Y^+ = \{(-3,-1),(-3,1),(3,-1),(3,1)\}, Y^- = \{(-1,-1),(-1,1),(1,-1),(1,1)\}$. En considérant comme classifieur faible des stumps (fonction de type $\mathbb{1}_{x_i<\theta_i}$, correspondant à un arbre de décision à 2 feuilles), quels sont les deux premiers classifieurs appris ? Sont-ils suffisant pour la classification parfaite ?

Bagging : classifieurs appris par bootstrap indépendant de l'ensemble d'apprentissage (chaque classifieur est appris independamment), boosting modification de la distribution des exemples en fonction des exemples les moins bien classés (pour les faire ressortir et mieux les classer au coup d'apres).

On considère l'erreur 0-1. Le but c'est juste de le faire avec les mains. Donc si on choisit un stump à x = -2, ca fait les 2 points en x=3 mal classifié, donc ceux-la vont grossir et les autres vont tous decroitre (meme poids dailleurs pour tous). au prochain coup, les 2 points en x= 3 doivent etre forcement bien classé, donc pour le classifieur on peut mettre la frontiere a x = 2. Du coup celui-ci classe mal les deux x=-3, tous les negatifs vont avoir des tous petits poids, et les positifs des poids intermediaires. Comme 3eme on peut prendre un classifieur constant (equivalent au biais).... Avec 2 c'est pas suffisant, on va montrer dans la partie apres que le boosting converge vers 0 erreur de train, sil est possible de trouver a chaque coup un classifieur qui fait moins de 50

Q 1.1.2 Soit les problèmes de classification suivants, proposez des classes de classifieurs faibles bien adaptées aux types de données. Combien en faut-il ?

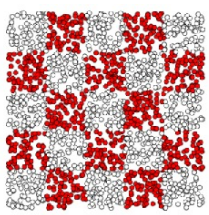
Figure 1: Checkerboard

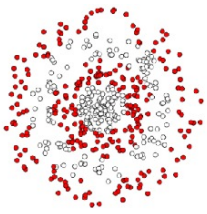
Figure 2: Concentric rings


Figure 3: The serpent

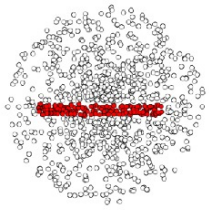
Figure 4: Stop sign

Fig 1 : patch de carré, séparateur linéaire
Fig 2. : patch cercle
Fig3. : un peu tout, en particulier patch carré, cercle
Fig 4. : patch carré, droite, ...
un patch est paramétré generalement par un rayon (cas cercle) ou une longueur (cas carré) et une position

Q 1.1.3 Exprimer l'erreur ϵ_t en fonction d'un coût donné $l(x, y)$ et des $w_t(i)$.

$\epsilon_t = \sum_i w_t(i) l(x_i, y_i)$

Q 1.1.4 Comment varie α_t en fonction de ϵ_t ? Que se passe-t-il pour $w_{t+1}(i)$ si l'exemple i est bien classifié ? mal classifié ?

Si $\epsilon_t < 0.5$, alors $\alpha_t > 0$, si $\epsilon_a < \epsilon_b$ alors $\alpha_a > \alpha_b$.
Le poids decroit si exemple bien classifé, augmente sinon (d'autant plus que loin de la marge d'ailleurs, autre interpretation possible).

On va montrer dans la suite que l'algorithme optimise bien l'erreur d'apprentissage. Le principe de la démonstration consiste à montrer que à chaque pas t, l'erreur est borné par $Z = \prod_{j=1}^{t} Z_j$, et que ce produit converge vers 0.

Q 1.2 Nous allons montrer d'abord que le choix de α_t conduit à minimiser Z_t.

Q 1.2.1 Exprimer Z_t et ϵ_t en fonction de $w_t(i)$, α_t et $y_i h_t(x_i)$.

Q 1.2.2 Exprimer $\frac{\partial Z_t}{\partial a_t}$. En déduire la valeur de α_t qui minimise Z_t.

Q 1.2.3 Donner l'expression de Z_t en fonction de ϵ_t pour α_t optimal.

Q 1.2.4 Soit $\gamma_t = \frac{1}{2} - \epsilon_t$. Sachant que $1 - x \leq e^{-x}$, montrer que Z décroît exponentiellement en fonction de t.

$Z_t = \sum_i w_t(i) e^{(-\alpha_t y_i h_t(x_i))}$, $\epsilon_t = \sum_{i|y_i \neq h_t(x_i)} w_t(i)$
$\frac{\partial Z_t}{\partial a_t} = -\sum_i y_i h_t(x_i) w_t(i) e^{(-\alpha_t y_i h_t(x_i))}$
$= -\sum_{i|y_i = h_t(x_i)} w_t(i) e^{-\alpha_t} + \sum_{i|y_i \neq h_t(x_i)} w_t(i) e^{\alpha_t}$
$= -(1-\epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t}$
$= 0 \iff (1-\epsilon_t) e^{-\alpha_t} = \epsilon_t e^{\alpha_t} \iff \alpha_t = 0.5 ln \frac{1-\epsilon_t}{\epsilon_t}$
$Z_t = \sum_i w_t(i) e^{-\alpha_t y_i h_t(x_i)} = \sum_i w_t(i) e^{0.5 ln \frac{1-\epsilon_t}{\epsilon_t} y_i h_t(x_i)} = \sum_{i|y_i = h_t(x_i)} w_t(i) e^{-0.5 ln \frac{1-\epsilon_t}{\epsilon_t}} +$
$\sum_{i|y_i \neq h_t(x_i)} w_t(i) e^{0.5 ln \frac{1-\epsilon_t}{\epsilon_t}} = (1-\epsilon)\sqrt{\frac{\epsilon_t}{1-\epsilon_t}} + \epsilon\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = 2\sqrt{\epsilon_t(1-\epsilon_t)} = \sqrt{1-4\gamma_t^2} \leq e^{-2\gamma_t^2}$.
Donc $Z = \prod_j Z_j \leq e^{-2\sum \gamma_j^2}$.

Q 1.3 Nous allons montrer maintenant que Z est une borne supérieure de l'erreur 0-1.

Q 1.3.1 Exprimer $w_{t+1}(i)$ en fonction de $h_j(x), \alpha_j(i), Z_j,\ 1 \leq j \leq t$, puis en fonction de $f_t(x_i)$. En déduire une expression de $\sum_i w_t(i)$ en fonction des Z_j et $y_i f_t(x_i)$, puis une expression de $Z = \prod_j Z_j$ en fonction de $y_i f_t(x_i)$

$w_{t+1}(i) = \frac{1}{Z_t} w_t(i) e^{(-\alpha_t y_i h_t(x_i))} = \frac{1}{\prod_j Z_j} w_1(i) e^{-y_i \sum_j (\alpha_j h_j(x_i))} = \frac{1}{\prod_j Z_j} w_1(i) e^{-y_i f_t(x_i)}$
$1 = \sum_i w_t(i) = \sum_i \frac{1}{\prod_j Z_j} w_1(i) e^{-y_i f_t(x_i)}$, donc $Z = \sum_i w_1(i) e^{-y_i f_t(x_i)} = \frac{1}{N} \sum_i e^{-y_i f_t(x_i)}$ car w_1 est uniforme.

Q 1.4 Montrez que l'erreur $0-1$ est bornée par le coût exponentiel $l(x,y) = e^{(-yf(x))}$. En déduire que Z est un majorant de l'erreur $0-1$.

On a bien $1 < e^x, x > 0$, et $0 < e^{-x}, x > 0$

Q 1.4.1 Conclure sur la décroissance exponentielle de l'erreur.

Exercice 2 – MDP et programmation dynamique

Rappel :

- un MDP est défini par un ensemble d'états $\mathcal{S}$, un ensemble d'actions $\mathcal{A}$, une fonction de transition $t : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R} : t(s,a,s')$ est la probabilité d'arriver en s' à partir de s en prenant l'action a, une fonction de récompense $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.
- une politique $\pi : \mathcal{S} \to \mathcal{A}$ (cas déterministe) attribue une action à chaque état.
- $V^\pi(s) = \mathbb{E}[\sum_{t \geq 0} \gamma^t r(s_t, \pi(s_t)) | s_0 = s, \pi]$

Q 2.1 Equation de Bellman

Q 2.1.1 Montrer que pour une politique stationnaire $V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} t(s'|s, \pi(s)) V^\pi(s')$.

$$
\begin{aligned}
V^\pi(s) &= \mathbb{E}[\sum_{t\geq 0}\gamma^t r(s_t,\pi(s_t))|s_0=s,\pi] \\
&= r(s,\pi(s)) + \mathbb{E}[\sum_{t\geq 1}\gamma^t r(s_t,\pi(s_t))|s_0=s,\pi] \\
&= r(s,\pi(s)) + \gamma\sum_{s'} t(s'|s,\pi(s))\mathbb{E}[\sum_{t\geq 1}\gamma^{t-1} r(s_t,\pi(s_t))|s_1=s',\pi] \\
&= r(s,\pi(s)) + \gamma\sum_{s'} t(s'|s,\pi(s))V^\pi(s')
\end{aligned}
$$

Q 2.1.2 Montrer l'equation d'optimalité de Bellman : $V^*(s) = max_\pi V^\pi(s) = max_a\ r(s,a) + \gamma\sum_{s'} t(s'|s,a)V^*(s')$.

On considère une politique $\pi = (a,\pi')$ (a est la première action, π' définit les suivantes en fonction des etats)

$$
\begin{aligned}
V^*(s) &= \max_\pi V^\pi(s) \\
&= \max_\pi \mathbb{E}[\sum_{t\geq 0}\gamma^t r(s_t,\pi(s_t))|s_0=s,\pi] \\
&= \max_{(a,\pi')}[r(s,a) + \gamma\sum_{s'} t(s'|s,a)V^{\pi'}(s')] \\
&= \max_a[r(s,a) + \gamma\sum_{s'} t(s'|s,a)\max_{\pi'} V^{\pi'}(s')] \\
&= \max_a[r(s,a) + \gamma\sum_{s'} t(s'|s,a)V^*(s')]
\end{aligned}
$$

Q 2.1.3 On appelle opérateur de Bellman optimal T l'opérateur : $(T\ V)(s) = \max_a r(s,a) + \gamma\sum_{s'} t(s'|s,a)V(s')$. Soit deux ensembles de valeurs V_1 et V_2 définis pour tout etat s. Montrer que $\|T^\pi V_1 - T^\pi V_2\|_\infty \leq \gamma\|V_1 - V_2\|_\infty$.

Pour tout etat s on a :

$$
\begin{aligned}
|TV_1 - TV_2| &= |\max_a r(s,a) + \gamma\sum_{s'} t(s'|s,a)V_1(s') - \max_{a'} r(s,a') + \gamma\sum_{s'} t(s'|s,a')V_2(s')| \\
&\leq \max_a |r(s,a) + \gamma\sum_{s'} t(s'|s,a)V_1(s') - r(s,a) + \gamma\sum_{s'} t(s'|s,a)V_2(s')| \\
&= \gamma\max_a \sum_{s'} t(s'|s,a)|V_1(s') - V_1(s')| \\
&\leq \gamma\|V_1 - V_2\|_\infty \max_a \sum_{s'} t(s'|s,a) \\
&=\leq \gamma\|V_1 - V_2\|_\infty
\end{aligned}
$$

Où la première inégalité est due au fait que $\max_a f(a) - \max_b g(b) \leq \max_a(f(a) - g(a))$

Q 2.1.4 L'algorithme value iteration applique, pendant un nombre donné d'itérations, l'opérateur optimal de Bellman pour mettre à jour les valeurs des etats : $V^{(k+1)} = TV^{(k)}$, avec $V^{(k)}$ l'ensemble des valeurs à l'itération k. Déduire de la propriété de contraction démontrée à la question précédente la convergence de cet algorithme (en supposant la propriété de point fixe $TV^* = V^*$).

$\|V^{(k+1)} - V^*\|_\infty = \|TV^{(k)} - TV^*\|_\infty \leq \gamma\|V^{(k)} - V^*\|_\infty \leq ... \leq \gamma^{(k+1)}\|V^{(0)} - V^*\|_\infty \to 0$

Q 2.1.5 On appelle opérateur de Bellman T^π l'opérateur : $(T^\pi\, V)(s) = r(s, \pi(x)) + \gamma \sum_{s'} t(s'|s, \pi(x))V(s')$. Montrer que si $V_1(s) \leq V_2(s)\ \forall s$, alors pour tout s on a : $T^\pi V_1(s) \leq T^\pi V_2(s)$.

$V_1(s) \leq V_2(s)\ \forall s$, donc pour tout s' et a on a : $r(s,a) + \gamma \sum_{s'} t(s'|s,a)V_1(s') \leq r(s,a) + \gamma \sum_{s'} t(s'|s,a)V_2(s')$
Donc, en prenant $a = \pi(s)$ en chaque s, on a : $r(s,\pi(s)) + \gamma \sum_{s'} t(s'|s,\pi(s))V_1(s') \leq r(s,\pi(s)) + \gamma \sum_{s'} t(s'|s,\pi(s))V_2(s')$
Soit pour tout s : $(T^\pi V_1(s)) \leq (T^\pi V_2(s))$

Q 2.1.6 L'algorithme policy iteration travaille, à chaque itération k, en deux temps :

1. Évaluation de $V^{(k)}$ selon la politique courante stationnaire π_k.
2. Mise à jour de la politique π_{k+1} selon une stratégie greedy. Pour tout etat s : $\pi_{k+1}(s) = arg\max_a r(s,a) + \gamma \sum_{s'} t(s'|s,a)V^{(k)}(s')$

Déduire de la propriété de monotonicité démontrée à la question précédente la convergence de cet algorithme.

Puisqu'on a fait une évaluation complète de $V^{(k)}$ selon la politique π_k, on a : $V^{(k)} = T^{\pi_k}V^{(k)}$ (on est dans un etat stable après convergence)
Or : $T^{\pi_k}V^{(k)} \leq TV^{(k)} = T^{\pi_{k+1}}V^{(k)}$ (du fait de la mise à jour greedy qui choisit l'action max)
On alors, grâce à la propriété de monotonicité :

$$V^{(k)} \leq T^{\pi_{k+1}}V^{(k)}, \tag{1}$$
$$T^{\pi_{k+1}}V^{(k)} \leq (T^{\pi_{k+1}})^2V^{(k)}, \tag{2}$$
$$... \leq ..., \tag{3}$$
$$(T^{\pi_{k+1}})^{n-1}V^{(k)} \leq (T^{\pi_{k+1}})^nV^{(k)}. \tag{4}$$

Si on assemble toutes ces inégalités (qui correspondent en quelque sorte aux différentes passes de l'algo d'évaluation de la politique), on a : $V^{(k)} \leq \lim_{n\to\infty}(T^{\pi_{k+1}})^nV^{(k)} \leq V^{(k+1)}$
On a donc une séquence de valeurs qui s'accroissent à chaque itération jusqu'à stabilité. Avec un nombre de politiques fini, cette stabilité est sûre d'être atteinte au bout d'un certain temps, c'est l'avantage de policy iteration (contrairement à value iteration qui ne converge qu'asymptotiquement). Le problème de cet algo c'est qu'il demande à chaque itérations un évaluation complète des valeurs, ce qui peut être très coûteux.

Q 2.2 On considère le MDP déterministe suivant :
- 6 états $s_1, \ldots, s_6$
- s_1 et s_6 comme états finaux.
- deux actions possibles g et d
- les transitions : $t(s_i, g) = s_{i-1}$ et $t(s_i, d) = s_{i+1}$ pour $1 < i < 6$
- La récompense est définie par $r(s_i, a) = -10$ pour $1 < i < z$ pour $a \in \{g, d\}$, et $r(s_2, g) = 50$ et $r(s_5, d) = 100$.

Q 2.2.1 Représenter le mdp graphiquement. A quel problème plus général il correspond ?

Labyrinthe

Q 2.2.2 Soit la politique non déterministe uniforme $\pi(s_i, d) = \pi(s_i, g) = 0.5$ pour $1 < i < 6$. Déterminer $V^\pi(s)$ en utilisant un algorithme d'évaluation de cette politique.

Q 2.2.3 Déterminer π^* politique optimale en utilisant l'algorithme Policy iteration.

Q 2.2.4 Déterminer π^* politique optimale en utilisant l'algorithme Value iteration.

Q 2.3 On considère maintenant que l'on ne connaît pas la fonction de transition t

Q 2.3.1 Dire quels problèmes se posent alors avec les algorithmes précédents

V pas calculable en prenant l'action max
Il faut sampler

Q 2.3.2 Si on considère que $V^\pi(s_t) = r(s, \pi(s_t)) + \gamma V^\pi(s_{t+1})$, proposer un algorithme pour evaluer une politique π

Temporal Difference

Q 2.3.3 Si on souhaite maintenant chercher la politique maximisant V^π. Comment procéder ?

Soit $Q^\pi(s, a) = \mathbb{E}[\sum_{t \geq 0} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a, a_t = \pi(s_t)]$ l'espérance de gains d'une politique π après avoir réalisé l'action a à partir de l'état s.
Politique online =¿ Sarsa
Politique offline =¿ Q-Learning

$$Q_{t+1}(s_t, a_t) = \underbrace{Q_t(s_t, a_t)}_{\text{actuel}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{vitesse d'apprentissage}} \cdot \left(\overbrace{\underbrace{R_{t+1}}_{\text{recompense}} + \underbrace{\gamma}_{\text{facteur d'actualisation}} \underbrace{\max_a Q_t(s_{t+1}, a)}_{\text{valeur optimale estimee}}}^{\text{valeur apprise}} - \underbrace{Q_t(s_t, a_t)}_{\text{actuel}} \right)$$

Réponses intuitives dans https://www.tu-chemnitz.de/informatik/KI/scripts/ws0910/ml09_6.pdf

Q 2.3.4 Et si on a beaucoup trop d'états pour tout stocker en mémoire / tout considérer ?

On passe à de l'approché avec représentation de $Q^\pi(s,a)$ par une fonction $f_\theta(s,a)$