Approximation de politiques par renforcement et classification

Reinforcement using Supervised Learning for Policy Generalization

Julien Laumônier Brahim Chaib-draa
Laboratoire DAMAS, Département d'informatique et de génie logiciel,
Université Laval, Québec G1K7P4, CANADA
{jlaumoni,chaib}@damas.ift.ulaval.ca

Résumé

La plupart du temps, l'apprentissage par renforcement exact ne permet pas de résoudre efficacement les applications réelles de très grandes tailles. Dès lors, les recherches se sont intéressées aux méthodes d'approximation. Certaines utilisent une fonction de valeur approximée alors que d'autres se concentrent sur les approches de gradient de politique. Parallèlement à ça, de nombreux concepts d'approximation ont été étudiés par la communauté de l'apprentissage supervisé. Il est apparu clairement qu'il fallait lier les deux formes d'apprentissage et ainsi, récemment, des approches faisant le lien entre l'apprentissage par renforcement et l'apprentissage supervisé par l'approximation de la politique optimale par classification, ont fait leur apparition. Dans cet article, nous proposons un algorithme qui combine le Q-Learning, les séparateurs à vaste marge (SVM) et le vote de majorité pour approximer la politique optimale d'un Processus Décisionnel Markovien. Nous étudions l'impact des paramètres résultant de la combinaison des approches sur la performance d'apprentissage et nous montrons empiriquement que notre algorithme est capable d'apprendre une politique proche de l'optimale même avec un grand nombre d'états à traiter.

Mots Clef

Apprentissage par renforcement, classification, approximation

Abstract

In general, exact reinforcement learning algorithms do not provide good solutions to very large Markov Decision Process (MDP). Thus, research has focused on approximation methods. Some of them provide solutions using value function approximation and others using policy gradient approaches. Many concepts for approximation have been intensively studied by supervised learning community. Recently, some works have been done for joining reinforcement and supervised learning. Consequently, in this paper, using basic notion of supervised learning, we propose a reinforcement learning algorithm which combines Q-learning, Support Vector Machines and majority vote. We study the impact of combining algorithms and parameters on learning performance and we show empirically that our algorithm is able to learn a close optimal policy even with a high number of states.

Key Words

Reinforcement Learning, Classification, Approximation

1 Introduction

Utiliser l'apprentissage par renforcement sous tendu par un Processus Décisionnel Markovien (PDM) faisant intervenir un très grand nombre d'états (comme c'est généralement le cas des applications réelles) conduit souvent à une résolution exacte insoluble sur une machine. C'est pourquoi, de nombreuses approches proposent plutôt de calculer une solution approximée soit en utilisant une fonction de valeur paramétrée (comme TD-Gammon [24] ou plus récemment par une approximation du modèle basée sur des instances [13]) soit en approximant directement la politique par une méthode de gradient [21]. L'avantage de ces approches provient du fait qu'elles fournissent une politique sur l'ensemble des états alors que les approches classiques d'apprentissage par renforcement ne garantissent pas l'exploration de l'ensemble des états en un temps fini. Cependant, ces approches approximées ont besoin d'une représentation structurée du problème sous la forme de paramètres.

Parmi les méthodes qui exploitent la connaissance de la structure des problèmes, les méthodes de classification par apprentissage supervisé fournissent de nombreux outils algorithmiques et mathématiques pour trouver et caractériser la solution d'un problème [8]. Ces solutions, tout en se basant sur un petit nombre d'exemples, produisent peu d'erreur globalement. De nombreux travaux ont proposé de combiner l'apprentissage par renforcement et l'apprentissage supervisé pour approximer la fonction de valeur optimale d'un PDM [20]. Cependant, peu de travaux considèrent non pas l'approximation de la fonction de valeur mais bien l'approximation directe de la politique optimale. Le principal argument pour approximer directement la politique est de considérer que l'aspect le plus important dans l'apprentissage par renforcement est d'obtenir la politique optimale, c'est à dire de trouver la meilleure action

dans chaque état. De plus, dans certains problèmes, il peut être plus complexe d'approximer la fonction de valeur que la politique optimale [1]. Langford et Zadrozny [16] ont formalisé ce problème et ont montré que tous problèmes d'apprentissage par renforcement à horizon T peut se transformer en T problèmes de classification. Toutefois les auteurs n'ont pas donné d'algorithme en ligne. D'un point de vue algorithmique, Lagoudakis et Parr [14] ont proposé d'utiliser la classification dans un algorithme d'itération de politique pour trouver une solution approximative.

C'est dans ce contexte que se situe cet article dans la mesure où il propose d'étendre ces approches en fournissant un algorithme qui combine l'apprentissage par renforcement et la classification. Plus précisément, notre algorithme se base sur une fonction de valeur $Q(s, a)$ apprise par Q-Learning qui sert d'oracle à un algorithme de séparateur à vaste marge (SVM) qui approxime la politique optimale. Pour améliorer la convergence de l'algorithme, un vote de majorité est fait sur un ensemble de politiques approximées par le SVM. Nous nous intéressons dans cet article uniquement aux problèmes à deux actions. Nous montrons sur le problème du Mountain Car [23] que cet algorithme converge rapidement vers une politique très proche de l'optimale et ce, même lorsque le nombre d'états est très grand contrairement au Q-Learning classique. Enfin, cet algorithme, de par sa combinaison de plusieurs techniques d'apprentissage, possède un grand nombre de paramètres. C'est pourquoi, en nous basant sur le problème du Mountain Car, nous étudions l'impact des différents paramètres sur la performance de l'apprentissage.

L'article est organisé comme suit. Tout d'abord, nous présentons les bases théoriques des deux formes d'apprentissage : l'apprentissage par renforcement et l'apprentissage supervisé. Puis, après avoir introduit les résultats théoriques de l'approximation de politique par classification, nous proposons un formalisme légèrement différent. Nous présentons ensuite notre algorithme, suivi par nos résultats ainsi qu'une discussion sur les différents paramètres expérimentaux utilisés. Finalement, nous présentons quelques travaux connexes et concluons sur cette approche.

2 Bases théoriques

2.1 Apprentissage par renforcement

Pour un agent, l'apprentissage par renforcement consiste à apprendre à partir des interactions qu'il a avec son environnement. Précisément, il observe l'état courant de l'environnement puis agit. En contrepartie, il reçoit une récompense et l'environnement passe dans un nouvel état. Le Processus Décisionnel Markovien (PDM ou MDP en anglais) est le modèle formel utilisé pour l'apprentissage par renforcement [23]. Un PDM est un n-uplet $< S, A, \mathcal{P}, \mathcal{R} >$ avec

- S, un ensemble d'états,
- A, un ensemble d'actions,
- $\mathcal{P} : S \times A \times S \rightarrow [0, 1]$, la fonction de transition qui donne la probabilité de passer d'un état à un autre en effectuant un action.
- $\mathcal{R} : S \times A \times S \rightarrow \mathbb{R}$, la fonction de récompense.

Une hypothèse très importante de l'apprentissage par renforcement, appelée la *propriété de Markov*, est définie sur la fonction de transition des états de l'environnement. Un environnement est markovien si la fonction de transition vers un état ne dépend que de l'état précédent.

Rappelons que l'objectif de l'agent est de maximiser le total des récompenses reçues. Dans ce contexte, on introduit la notion de *coefficient d'actualisation*, noté γ, qui permet de donner plus d'importance aux récompenses proches plutôt qu'à celles qui sont plus lointaines. La somme des récompenses est alors : $R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$, avec $0 \leq \gamma < 1$. Dans le cas particulier où $\gamma = 0$, l'agent maximise la récompense immédiate alors que, lorsque $\gamma \rightarrow 1$, l'agent prend de plus en plus en compte les récompenses à long terme. T représente la durée d'un *épisode* qui correspond à une sous séquence d'interaction entre l'agent et l'environnement.

La fonction de valeur $V^\pi(s)$ représente la qualité de l'état s en terme de récompense espérée en suivant la politique π. Formellement, cette fonction de valeur est définie par

$$V^\pi(s) = E_\pi[R_t | s_t = s] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right].$$

De même, on définit la *fonction de valeur* $Q^\pi(s, a)$ comme l'espérance de récompense si l'agent choisit l'action a dans l'état s et s'il suit la politique π par la suite.

Trouver une solution à un problème d'apprentissage par renforcement revient à trouver une politique qui maximise la somme des récompenses sur le long terme. L'équation optimale de Bellman pour $V^*(s)$ décrit le fait que la valeur optimale d'un état est égale à la récompense attendue pour la meilleure action dans cet état. Plus, formellement :

$$V^*(s) \quad = \quad \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^*(s') \right]. \qquad (1)$$

Le lien entre $V(s)$ et $Q(s, a)$ est représenté par l'équation $V^*(s) = \max_a Q^*(s, a)$. À partir de la fonction de valeur optimale $Q^*(s, a)$, on définit la politique optimale par $\pi^*(s) = \arg\max_a Q^*(s, a)$. Si la dynamique du système est connue ($\mathcal{P}^a_{ss'}$ et $\mathcal{R}^a_{ss'}$), il est possible de résoudre le problème en utilisant la programmation dynamique pour trouver la politique optimale. Dans les cas où l'on ne connaît pas la dynamique du système, il faut soit utiliser une estimation soit utiliser un ensemble de méthodes, appelées apprentissage aux différences temporelles (TD-Learning), qui sont capables d'évaluer la politique optimale grâce aux expériences générées par une interaction avec le système.

Un des principaux algorithmes de calcul de la politique optimale, Q-Learning, utilise la fonction de valeur Q. Cet algorithme fonctionne par interaction avec l'environnement de la manière suivante : à chaque épisode, l'agent choisi une action a en fonction d'une politique π dérivée des

valeurs courantes de Q. Il exécute l'action a, reçoit la récompense et observe l'état suivant. Puis il met à jour les valeurs de $Q(s,a)$ par la formule

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

où r est la récompense immédiate, s' est le nouvel état, α est le taux d'apprentissage. La convergence de cette algorithme vers la fonction de valeur optimale a été démontrée par Watkins et Dayan [26]. Le choix de l'action à chaque étape s'effectue grâce à des fonctions d'exploration. Parmi ces méthodes on peut citer l'approche *voraces*, *ϵ-voraces* et *softmax* qui choisissent une action au hasard selon une certaine probabilité. Pour une description plus exhaustive des fonctions d'exploration, on pourra se référer à [25].

2.2 Apprentissage supervisé

Le problème de l'apprentissage supervisé (ou classification) est de prédire la classe de chaque exemple d'un domaine particulier à partir d'un petit échantillon de ces exemples. Formellement, le problème de l'apprentissage supervisé est de trouver une fonction $h(\mathbf{x})$ à partir d'exemples d'un ensemble d'apprentissage $\mathbf{TS} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ telle que, idéalement, $h(\mathbf{x}) = y$ pour tous les exemples non observés $(\mathbf{x}, y)$. Dit autrement, h doit minimiser l'erreur

$$E_{x,y\sim D}[I(h(x) \neq y)]$$

avec $I(p) = 1$ si p est vrai, 0 sinon. Une autre forme de classification, la classification à importance pondérée, consiste à trouver une fonction h qui minimise l'erreur $E_{x,y\sim D}[wI(h(x) \neq y)]$ où chaque exemple (x,y) est associé à un poids w.

Plusieurs algorithmes ont été proposés pour calculer cette fonction. En général, ces algorithmes de classification supposent que les exemples sont distribués indépendamment et identiquement (iid) à partir d'une distribution D. Cette hypothèse est nécessaire pour garantir des bornes sur la généralisation que propose h. Parmi ces algorithmes, le Perceptron et les séparateurs à vaste marge (en anglais Support Vector Machine ou SVM) apprennent un hyperplan qui sépare linéairement les exemples [8]. Le Perceptron apprend un hyperplan séparateur quelconque alors que les SVM apprennent l'hyperplan qui maximise la marge de séparation. Les paramètres du Perceptron (β) et des SVM (C) représentent le niveau d'erreur de séparation autorisé pendant l'apprentissage. Ces paramètres permettent d'obtenir une meilleure généralisation sur le problème. Parmi les améliorations des SVM, Scholkopf et al. [22] ont proposé l'algorithme ν-SVM. Le paramètre ν de cet algorithme contrôle le nombre d'erreurs autorisé ainsi que le nombre de vecteurs de support, c'est-à-dire, les exemples effectivement utilisés pour approximer la fonction h. Plus particulièrement, ν est à la fois une borne supérieure sur la fraction de la marge d'erreur et une borne inférieure sur la fraction des vecteurs de support. Nous verrons dans la section 4.2 comment utiliser ce paramètre pour améliorer l'apprentissage.

Tous les algorithmes d'apprentissage supervisé précédemment introduits fournissent un classificateur linéaire. Cependant, en général, les fonctions à approximer ne sont pas linéaires. C'est pourquoi pour permettre d'apprendre des séparateurs non linéaires, il est possible d'utiliser des méthodes à base de noyaux. Un noyau est une fonction K qui calcule un produit scalaire de deux vecteurs dans un espace augmenté. En utilisant un noyau, les algorithmes d'apprentissage supervisés calculent un séparateur linéaire dans l'espace augmenté qui correspond à un séparateur non linéaire dans l'espace d'origine. Parmi les noyaux disponibles, les plus communs sont le noyau polynomial $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^p$ et le noyau RBF (Radial Basis Function) $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2)}{2\sigma^2}}$. Dans cet article, nous utilisons un noyau RBF puisqu'il permet de travailler dans des espaces de très grande dimension et peut potentiellement approximer des séparateurs très complexes.

Les classificateurs de Bayes sont des classificateurs particuliers qui utilisent plusieurs classificateurs pour choisir la classe d'un exemple par la règle de majorité. Chaque classificateur choisit une classe et la classe majoritaire (possiblement avec pondération) est choisie par le classificateur de Bayes. Parmi les algorithmes qui apprennent des classificateurs de Bayes, on peut citer les algorithmes de Bagging et de boosting [2]. Fondamentalement, l'algorithme de boosting change la distribution de l'ensemble d'apprentissage en fonction des performances des classificateurs appris précédemment. De son coté, l'algorithme de Bagging ne change pas la distribution. Nous verrons dans la section 4.1 comment utiliser le vote de majorité pour améliorer la converge de l'approximation de la politique optimale.

3 Formalisation du problème

Comme il a été dit précédemment, nous nous intéressons à trouver une politique approximée d'un PDM par l'utilisation de l'apprentissage supervisé à l'intérieur de l'apprentissage par renforcement. Le formalisme suit celui du PDM factorisé : Un ensemble d'états S ($|S| = n$) où chaque état est représenté par v variable d'états $\mathbf{s}_i = (s_1, \ldots, s_v)_i$ qui peuvent chacune prendre leur valeur dans $\mathcal{D}_v$. Dans le même temps, l'agent peut effectuer $p = |A|$ actions $a_1, \ldots, a_p$. Cependant, dans cet article, on ne s'intéresse qu'aux cas à deux actions. Une fonction de récompense ainsi qu'une fonction de transition peuvent être définies. Cependant, comme nous travaillons dans le contexte du Q-Learning, la connaissance de ces deux fonctions n'est pas nécessaire. Une politique dans un PDM est représentée par une fonction $\pi : S \rightarrow A$ qui peut être vue comme un classificateur qui doit prédire une action a dans chaque état s. Le problème d'approximation de politique consiste donc à trouver un classificateur capable d'approximer la politique optimale d'un PDM.

3.1 Formalismes existants

Langford et Zadrozny ont proposé une réduction de l'apprentissage par renforcement à horizon T vers T problèmes de classification [16]. Cette réduction se décompose en deux parties :

1. Une réduction du problème de l'apprentissage par renforcement vers un problème de classification à importance pondérée.
2. Une réduction de la classification à importance pondérée vers la classification multi-classe.

La première partie est une réduction du problème d'apprentissage par renforcement à horizon T vers T instances de classification qui consiste à définir les distributions $D_1, \ldots, D_T$ sur les états. Cette première partie dépend de la connaissance que l'on possède sur le modèle de l'environnement : soit on possède un modèle génératif d'états soit un modèle génératif de traces. Le modèle génératif d'états suppose que l'on est capable d'obtenir, à partir d'un état s et d'une action a, l'état suivant s'. Le modèle génératif de traces permet uniquement d'obtenir une trace d'exécution a partir d'un état initial s_0 et d'une politique π. Le modèle de traces est donc plus faible que le modèle d'états. Concrètement, grâce au modèle génératif, on génère les états possibles à partir de l'état initial et on évalue la fonction de valeur Q pour chaque paire états-action (x, y). Cette fonction de valeur est utilisée pour le poids w.
Comme précisé plus haut, la deuxième partie transforme les problèmes de classification à importance pondérée en problèmes de classification multi-classe. Elle suppose que l'on possède une distribution D_t sur (s, a, w) à partir de laquelle on peut construire une distribution $\hat{D}_t$ sur les paires états-action avec

$$\hat{D}_t(s, a) = \frac{w}{E_{s,a,w\sim D_t}[w]} D_t(s, a, w)$$

. Cette formulation implique minimiser l'erreur d'un classificateur appris à partir d'exemples générés par $\hat{D}_t$ revient à minimiser $E_{s,a\sim D_t}[wI(h(s) \neq a)]$ sur des exemples générés par D_t. Il suffit ensuite d'utiliser n'importe quel algorithme de classification sur les exemples générés par les distribution $\hat{D}_t$.
En utilisant un modèle génératif d'états, Langford et Zadrozny ont démontré que si chaque classificateur possède un risque d'au plus ε alors la politique π satisfait

$$V^{\pi^*} - V^{\pi} \leq \frac{T+1}{2}\varepsilon$$

avec π^* la politique optimale.
Les algorithmes généraux proposés par les auteurs sont soit à base de traces ou à base d'états. L'idée générale des ces algorithmes est la suivante :

1. Générer des exemples (s, a, w) à partir de π et du modèle (états ou trace),
2. Utiliser ces exemples pour apprendre un classificateur,
3. Utiliser ce classificateur pour améliorer π,
4. Recommencer jusqu'à ce que π n'évolue plus.

Similairement, Blatt et Hero [5] ont proposé une réduction et un algorithme similaires à ceux de Langford et Zadrozny [16]. Toutefois, il existe une petite différence entre les deux réductions qui réside dans le fait que l'approche de Blatt et Hero [5], selon les auteurs, génère une distribution D qui donne plus d'importance aux états dont l'écart de valeur entre l'action optimale et les autres actions est grand, alors que l'approche de Langford et Zadrozny [16] se concentre simplement sur les états dont la valeur est grande.
Enfin, Langford et Zadrozny [17] ont proposé une autre réduction de l'apprentissage par renforcement à horizon T vers un ensemble de T problèmes de classification binaire. Cette réduction utilise une autre réduction appelée SECOC (Sensitive Error Correcting Output Code [15]) qui permet de passer d'un problème de classification sensible aux coûts multiclasse à un problème de classification binaire. Langford et Zadrozny [17] définissent le regret d'une politique π comme la perte de récompense que donne π par rapport à la politique optimale. En utilisant cette définition, ils ont proposé une borne sur le regret de la politique en fonction du regret des classificateurs binaires associés.

3.2 Formalisme proposé

Notre approche utilise un modèle génératif de traces. Cependant, elle est différente de l'approche de Langford et Zadrozny [16] du point de vue de la génération de l'ensemble d'apprentissage. Notre algorithme ne passe pas par la classification à importance pondérée mais génère directement des exemples qui indiquent quelle est l'action optimale de chaque état sans spécifier de poids à une mauvaise classification. D'un point de vue de classification à importance pondérée, cela revient à donner un poids $w = 1$ pour les exemples qui donne la mauvaise action et $w = 0$ pour les exemples qui donnent l'action optimale. L'avantage de cette approche est qu'elle permet de générer plus simplement des exemples pour l'ensemble d'apprentissage sans attendre la fin d'une trace. L'inconvénient est que l'on perd les propriétés théoriques présentés précédemment puisque les exemples générés ainsi ne sont plus iid. De plus, notre approche a pour but de calculer une politique stationnaire approximée sur un horizon infini et non pas à horizon fini.
Formellement, l'objectif est d'apprendre une approximation de la politique optimale π^* qui va minimiser le nombre d'erreurs faites sur l'action prédite dans l'ensemble des états. Pour cela, nous définissons la fonction de perte suivante :

$$l(\pi(\mathbf{s}), a) = \begin{cases} 0 & \text{if } \pi(s) = a \\ 1 & \text{if } \pi(s) \neq a. \end{cases}$$

La politique apprise π doit minimiser la fonction de perte pour tous les états de S. Le risque d'une politique $R(\pi)$ est défini comme la fraction des états dans lesquels π ne prédit

pas l'action optimale :

$$R(\pi) = \mathop{\mathrm{E}}_{S}[l(\pi(S), \pi^*(S))] = \frac{1}{n}\sum_{i=1}^{n} l(\pi(s_i), \pi^*(s_i)).$$

La politique optimale π^* est la politique qui minimise le risque pour toutes les politiques Π :

$$\pi^* = \arg\min_{\pi \in \Pi} R(\pi).$$

Par définition le risque de la politique optimale est zéro. Cependant, en pratique, il n'est pas possible de calculer le risque d'une politique puisqu'on ne connaît pas la politique optimale. L'algorithme de classification doit donc utiliser un ensemble d'apprentissage $\mathbf{TS}$, idéalement un sous ensemble de π^* de taille m. Le risque empirique est défini comme la fraction des états où la politique π se trompe sur l'ensemble d'apprentissage :

$$R_{emp}(\pi) = \frac{1}{m}\sum_{i=1}^{m} l(\pi(S), A).$$

Ce nombre d'erreurs est calculé grâce à un oracle qui fournit l'action optimale dans chaque état. Notre algorithme utilise comme oracle $\arg\max_a Q(s,a)$ avec Q une fonction de valeur tabulaire apprise par un Q-Learning. Il faut noter que lorsqu'un état s n'a pas été visité, $\arg\max_a Q(s,a)$ va retourner une action aléatoire. La politique tabulaire va donc de temps en temps faire des actions aléatoires. En généralisant, on espère faire mieux que la politique aléatoire dans les états non visités.
Cette approche nécessite plusieurs hypothèses pour fonctionner de manière efficace. Tout d'abord, il faut supposer que, même si le nombre d'états est très grand, l'action optimale donnée par la fonction de valeur après un petit nombre de mises à jour est la véritable action optimale. Idéalement, l'oracle doit être capable d'apprendre une partie de la solution rapidement sans explorer tout l'espace d'états. Dans le cas contraire, plus le nombre de mise à jour est grand pour obtenir l'action optimale, plus l'ensemble d'apprentissage de l'algorithme de classification contiendra des erreurs. De plus, compte tenu du fait qu'on s'intéresse au risque d'une politique, il est nécessaire que le fait de faire quelques erreurs dans le choix de l'action pour certains états n'influence pas trop la valeur de la politique.

4 Algorithme MajSPRL

L'algorithme, appelé Majority Vote Supervised Policy Reinforcement Learning (MajSPRL), que nous proposons dans cet article combine plusieurs idées. Tout d'abord, il est basé sur le Q-Learning qui sert de mise à jour pour l'oracle. Ensuite, il utilise un classificateur pour approximer la politique optimale (ν-SVM à marge floue avec un noyau RBF). Enfin, il réalise un vote de majorité sur un ensemble de classificateur pour garantir l'amélioration de la politique. L'algorithme 1 peut se décomposer en quatre parties :

1. **Génération des exemples** : de la ligne 3 à la ligne 8, l'algorithme, qui correspond au Q-Learning, permet de générer des exemples,
2. **Mise à jour de l'ensemble d'apprentissage** : les lignes 10 à 16 correspondent à la génération de l'ensemble d'apprentissage,
3. **Entraînement** : la ligne 18 fait apprendre le nouveau classificateur,
4. **Amélioration de la politique** : enfin, les lignes 23 à 27 améliorent la politique π.

Algorithm 1 Q-Learning à politique supervisée par vote de majorité (MajSPRL)

1: **for all** épisodes **do**
2: Initialiser $s = s_0$
3: **while** épisode non terminé **do**
4: {***Générations des exemples***}
5: $a = \pi(s) = Vote(h_i(s), \alpha_i)$ avec expl. ε_1
6: Faire l'action a
7: Observer r et le nouvel état s'
8: $Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')]$
9: {***Mise à jour des ensembles d'apprentissage***}
10: **if** $\pi(s) \neq \arg\max_{a'} Q(s,a')$ **then**
11: avec probabilité ε_2,
12: **if** $\mathbf{TS}_i$ est plein **then**
13: Enlever un exemple au hasard avec prob. ε_3
14: ou enlever le pire exemple avec prob. $1-\varepsilon_3$
15: **end if**
16: ajouter $(s, \arg\max_{a'} Q(s,a'))$ dans $\mathbf{TS}_i$
17: {***Entraînement***}
18: Entraîner h_i avec $\mathbf{TS}_i$
19: **end if**
20: $s = s'$
21: **end while**
22: {***Amélioration de la politique***}
23: Tester la valeur V^{h_i} du classificateur h_i.
24: **if** $V^{h_i} > \theta$ **then**
25: $\alpha_i = 2$
26: Ajouter un votant h_{i+1} avec $\alpha_{i+1} = 1$
27: $\theta = V^{h_i}$
28: **end if**
29: **end for**

4.1 Description

Génération d'exemples. La génération des exemples se fait grâce à la boucle de l'algorithme du Q-Learning. L'agent choisit une action en fonction de la politique actuelle, agit, observe le nouvel état et la récompense. L'oracle est mis à jour par la mise à jour habituelle de la fonction de valeur. On utilise $\arg\max_a Q(s,a)$ pour l'oracle. La politique $\pi(s)$ correspond à un vote de majorité pondéré par des poids α_k sur les classificateurs $h_1(s), \ldots h_i(s)$.

Mise à jour de l'ensemble d'apprentissage. Normalement, approximer la politique optimale par un algorithme de classification nécessiterait de garder toutes les paires états-action (s, a) visitées. Cependant, cette approche est irréalisable en pratique car, non seulement elle utilise beaucoup de mémoire mais aussi les premières paires visitées par l'agent ne sont pas optimales. Ainsi, toute la difficulté de l'approximation par un algorithme de classification est de trouver un sous ensemble de paire états-action (s, a) qui va permettre de généraliser correctement la politique optimale. Récemment, plusieurs modifications de l'algorithme du Perceptron ont été proposées pour limiter le nombre d'exemples nécessaire pour apprendre [9] [27]. Ces méthodes gardent un sous ensemble d'exemples en effaçant certains quand un nouvel exemple provoque une erreur de classification. Les deux stratégies employées dans notre approche pour choisir l'exemple à enlever sont les suivantes :

1. Test de la marge [9] : Cette méthode enlève l'élément qui, une fois enlevé, garde la marge de séparation maximum sur l'ensemble d'apprentissage. Formellement, l'élément i est enlevé si

$$i = \arg\max_j \{a_j(w_{t-1} - \alpha_j a_j s_j) \cdot s_j\}.$$

   où $a_j(w_{t-1} - \alpha_j a_j s_j)$ correspond à la marge du classificateur entraîné avec l'ensemble d'apprentissage $\mathbf{TS}$ sans s_j.

2. Test de l'erreur [27] : Cette méthode enlève l'élément i qui, une fois enlevé, garde le nombre d'erreur minimale sur l'ensemble d'apprentissage. Formellement,

$$i = \arg\min_j \{\sum_{k=1}^{t} l(h_{\mathbf{TS}-s_j}(s_j), a_j\},$$

   où $h_{\mathbf{TS}-s_j}$ est le classificateur entraîné avec l'ensemble d'apprentissage $\mathbf{TS}$ sans s_j.

Concrètement, à chaque fois que la politique se trompe, si l'ensemble d'apprentissage est plein, on enlève un élément selon une des deux méthodes précédentes. Puis on ajoute dans l'état courant s avec l'action que l'oracle prédit comme optimale. Il faut noter que pendant l'apprentissage, on modifie seulement l'ensemble d'apprentissage de h_i lorsque π se trompe. De plus, on se garde toujours une probabilité ε_2 de ne pas changer l'ensemble d'apprentissage pour ne pas ajouter un exemple qui ne servirait à rien. De plus, on s'autorise toujours une probabilité ε_3 d'enlever un élément au hasard puisque l'heuristique de l'élément à enlever peut ne pas être bonne ainsi que pour éviter le surapprentissage. Cependant, ces probabilités doivent diminuer avec le temps pour garantir une stabilité dans la politique apprise.

Entraînement. On entraîne le classificateur en cours d'apprentissage h_i avec $\mathbf{TS}_i$. Ici, nous avons utilisé l'algorithme du ν-SVM. L'intérêt de cet algorithme par rapport au SVM classique est que son paramètre ν prend ses valeurs dans $[0, 1]$ alors que le paramètre d'un SVM standard C prend ses valeurs dans $[0, \infty[$. Nous pouvons donc autoriser de moins en moins d'erreur en faisant tendre ν vers 0 en partant de 1. Ce qui est plus difficile à faire avec le paramètre C.

Amélioration la politique. La dernière partie de l'algorithme proposé correspond à l'amélioration de la politique. Pour cela, nous utilisons un vote de majorité sur un ensemble de classificateurs. Pour améliorer la politique, après un certain nombre d'épisodes d'apprentissage, on teste le classificateur en cours d'apprentissage h_i seul sur un ensemble d'épisodes. Si la valeur moyenne de h_i est supérieur à un seuil θ, on modifie le poids de h_i (de 1 à 2), on ajoute un nouveau classificateur h_{i+1} (avec un poids de 1) et on augmente θ à V_{h_i}. Sinon on continue à améliorer h_i. Enfin, afin d'avoir une politique globale qui soit efficace et qui ne soit pas trop influencée par le classificateur en train d'apprendre, le poids de chaque classificateur fixé $h_1, \ldots, h_{i-1}$ doit être supérieur au poids du classificateur h_i.

4.2 Analyse de l'algorithme MajSPRL

Nous présentons une analyse de chaque partie de l'algorithme. Nous discutons de la complexité et de la convergence globale de l'algorithme dans la section 6.

Génération des exemples. L'utilisation d'un Q-Learning comme mise à jour de l'oracle va faire en sorte que cet oracle prédit de mauvaises actions au début de l'apprentissage, mais que, après un nombre infini d'épisodes, l'oracle ne fait plus d'erreur. Le fait que l'oracle tende vers l'oracle parfait va faire en sorte que les ensembles d'apprentissage ne vont plus contenir d'erreur. Donc, l'algorithme de classification trouvera des classificateurs plus efficaces et ce, grâce à la diminution du taux d'erreur permise ν.

Mise à jour de l'ensemble d'apprentissage. Les deux stratégies proposées (marge et erreur) permettent d'enlever les exemples non pertinents pour l'apprentissage du classificateur. Cependant, un inconvénient de ces stratégies est leur complexité, qui reste polynomiale dans la taille de l'ensemble d'apprentissage mais avec un haut degré qui dépend de la complexité de l'algorithme de classification.

Amélioration la politique. La justification de l'utilisation du vote de majorité vient du fait que chaque classificateur utilise un ensemble d'apprentissage de taille fini. Il existe donc un risque de surapprentissage. Pour éviter le surapprentissage de la politique π, on autorise possiblement un surapprentissage pour chaque classificateur h_i mais sur des ensembles d'états différents. En effet, puisque l'ajout d'un élément dans l'ensemble d'apprentissage de h_i se produit lorsque la politique globale se trompe, on garantie que les exemples d'apprentissage de h_i seront ceux pour lesquels π est mauvaise et donc que h_i compensera les erreurs de $h_1, \ldots, h_{i-1}$. Enfin, on fixe l'ensemble d'appren-

tissage de h_i s'il donne une bonne valeur sur l'ensemble du problème. Ainsi, on s'assure qu'un nouveau classificateur ne peut pas empirer (en moyenne) la politique π. Le fait de diminuer le seuil pour évaluer si un classificateur donne une bonne valeur, assure qu'on va finir par converger vers la meilleure politique approximable par le classificateur avec une taille donnée de l'ensemble d'apprentissage.

5 Expérimentations

Pour les expérimentations, nous avons utilisé PIQLE [7] pour les algorithmes d'apprentissage par renforcement et LIBSVM [6] pour les algorithmes de classification. Nous présentons des résultats sur le problème du Mountain Car [23] où nous nous sommes limités aux deux actions d'accélération. À chaque épisode, l'état initial est choisi uniformément sur l'ensemble des états possibles. L'agent apprend sur 2000 épisodes. Nous utilisons un facteur d'exploration $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = \frac{100}{idEpisode}$, un coefficient d'actualisation $\gamma = 0.99$ et un taux d'apprentissage constant $\alpha = 0.9$. Nous présentons les résultats sur le nombre d'étapes de chaque épisode puisque, dans ce problème, la récompense est inversement proportionnel au nombre d'étapes. L'environnement du Mountain car est continu. C'est pourquoi, il est nécessaire de le discrétiser pour utiliser des algorithmes d'apprentissage par renforcement, . Pour nos tests, nous sommes allés d'une précision de 10^{-2} sur chacune des deux variables d'état jusqu'à une précision de 10^{-5}. Il faut noter que pour une discrétisation de 10^{-2}, le nombre d'états est de 2380 alors que pour une précision de 10^{-5} il est de $2,38 \times 10^9$.

5.1 Résultats

Nous présentons dans cette section, les résultats de l'algorithme MajSPRL. Sauf indication contraire, ces résultats ont été obtenus avec une discrétisation de 10^{-3}. Tout d'abord, la figure 1 présente la comparaison de l'approximation par MajSPRL avec un Q-Learning standard et ce, pour différent degré de discrétisation. On peut observer que quelle que soit la discrétisation du problème, MajSPRL fait mieux que le Q-Learning et converge vers une politique proche de l'optimale. On peut voir que le Q-Learning ne parvient à s'approcher des résultats de MajSPRL que pour une discrétisation de 10^{-2}. Cependant, on remarque que dans les premiers épisodes, le Q-Learning fait en moyenne moins d'étapes que l'approximation. Mais une fois que l'ensemble d'apprentissage contient les éléments nécessaires à une bonne généralisation, l'approximation par MajSPRL reste très proche de la politique optimale.

La figure 2 présente l'impact de la taille de l'échantillon d'apprentissage sur la converge de l'apprentissage de MajSPRL. Globalement, on peut voir que plus la taille de l'échantillon d'apprentissage est grande, meilleure est la convergence. Cependant, à partir d'une taille de 100, nous n'avons constaté aucun gain significatif. Un autre point important est d'observer la courbe qui correspond à un ensemble d'apprentissage de taille 10. La politique avec

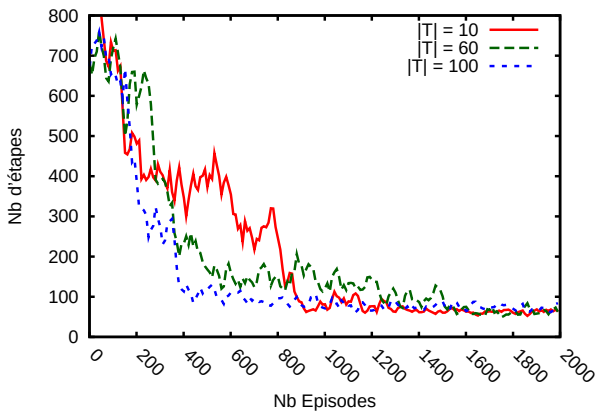
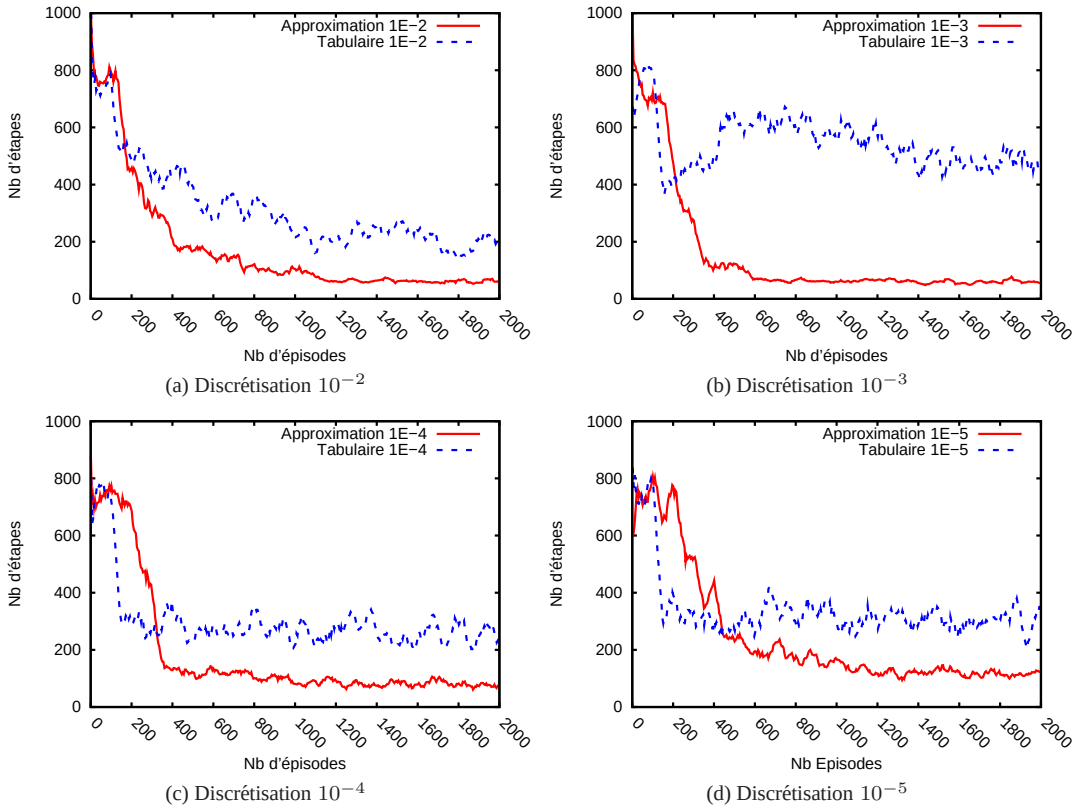


FIG. 2: Influence de la taille de l'ensemble d'apprentissage **TS**

MajSPRL obtenue après 2000 épisodes est aussi efficace qu'avec qu'un ensemble d'apprentissage plus grand bien que la convergence soit plus longue à obtenir.

Il est intéressant de noter que dans l'ensemble d'apprentissage, tous les exemples ne sont pas importants pour trouver une bonne politique approximée. Les vecteurs de support retournés par l'algorithme SVM correspondent aux exemples effectivement utilisés pour généraliser la politique. Le tableau 5.1 indique le nombre de vecteurs de support utilisé selon la taille des ensembles d'apprentissage. Ce nombre de vecteur de support est en relation avec la performance moyenne testé après 2000 épisodes d'apprentissage, le nombre moyen de votants ayant convergé ainsi que le nombre d'épisode moyen pour atteindre la convergence du premier classificateur. On peut constater que le nombre de vecteur de support n'influence pas la performance à long terme. Il peut d'ailleurs être étonnant que, même avec 2 vecteurs de support la performance soit aussi bonne. Ceci s'explique par le fait que la politique optimale du Mountain Car possède une structure très simple [19] facilement approximable par un noyau RBF. Cette même simplicité de la structure explique également le faible nombre de votant nécessaires pour une bonne approximation de la politique. En effet, la plupart du temps, dans ce problème le premier classificateur qui converge, converge vers la valeur proche de l'optimale. Dans ce cas, aucun autre classificateur suivant n'arrive à augmenter cette performance. Pourtant, utiliser un plus grand nombre d'exemple d'apprentissage a un intérêt pour ce problème. On peut voir que la vitesse de convergence moyenne du premier classificateur augmente avec la taille de l'ensemble d'apprentissage. Ceci peut s'expliquer par le fait que plus la taille de **TS** est grande plus il y a de chance d'avoir les bons vecteurs de support dans cet ensemble. Cependant, cette vitesse de convergence n'est plus significativement meilleure après une taille de 60.

La figure 3 présente une comparaison entre les différentes méthodes de mise à jour de l'ensemble d'apprentissage :



(a) Discrétisation 10^{-2}

(b) Discrétisation 10^{-3}

(c) Discrétisation 10^{-4}

(d) Discrétisation 10^{-5}

FIG. 1: Comparaison de Q-Learning tabulaire et de MajSPRL avec plusieurs discrétisations

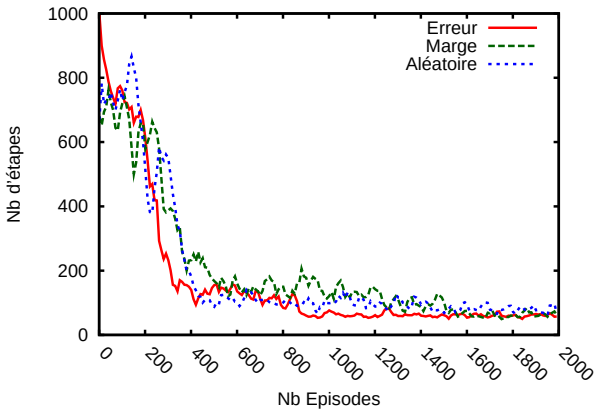
$\|\mathbf{TS}\|$	Épisode de convergence	Nombre de votants ayant convergé	Nombre total de vecteurs de support	Nombre d'étapes moyen
2	427 ± 332.46	1.12 ± 0.48	2.24 ± 0.96	95.26 ± 38.96
5	442 ± 184.71	1.10 ± 0.42	3.24 ± 1.42	100 ± 44.86
10	350 ± 180.42	1.08 ± 0.34	5.62 ± 2.18	90.80 ± 39.34
60	330 ± 174.09	1.56 ± 0.88	30.98 ± 13.69	86.64 ± 34.74
100	325 ± 171.80	1.34 ± 0.63	45.66 ± 21.09	92.30 ± 31.71

TAB. 1: Relation entre vecteurs de support, vitesse de convergence et performance après 2000 épisodes sur 50 apprentissages indépendants.

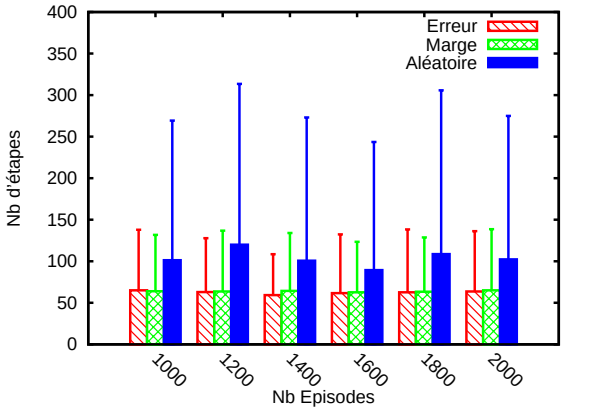
test de la marge, du nombre d'erreur ainsi qu'un choix aléatoire. Comme on peut le voir sur la figure 3a, la convergence de la politique ne varie pas vraiment d'une stratégie à l'autre. Cependant, lorsqu'on regarde un peu plus en détail et qu'on teste la politique apprise sur 500 épisodes à intervalle régulier, on remarque, comme la figure 3b le montre, que la valeur moyenne de la politique est moins bonne pour stratégie aléatoire que la valeur moyenne des deux autres stratégies. De même, la variance est supérieure pour le choix aléatoire. Pourtant, étant donnée la complexité des heuristiques de marge et d'erreur par rapport au choix aléatoire, ce dernier peut être intéressant à prendre en compte lors de la conception de l'algorithme.

6 Discussion

Les résultats que nous obtenons en utilisant le Q-Learning, les SVM et le vote de majorité sont très bons même si nous n'obtenons pas des résultats aussi bons que les meilleurs algorithmes d'apprentissage en terme de rapidité de convergence [10]. Cependant, l'algorithme en ligne présenté dans cet article est capable d'apprendre une politique proche de l'optimale sans connaissance du modèle de l'environnement. De plus, cet algorithme est très flexible et rien n'empêche d'utiliser une autre méthode pour obtenir un meilleur oracle que le Q-Learning ou même un autre algorithme de classification qui est capable de mieux généraliser que les SVM. En effet, nous sommes limités par le taux de convergence de l'oracle même s'il n'est pas nécessaire qu'il converge exactement. Seule l'action prédite doit correspondre à la véritable action optimale.



(a) Apprentissage



(b) Test

FIG. 3: Influence de la stratégie de mise à jour de l'ensemble d'apprentissage

Un des aspects important de notre algorithme est la place nécessaire pour stocker la politique généralisée. Comme toutes les méthodes d'approximation, il n'est nécessaire de stocker que les paramètres. Dans notre cas, la complexité en espace de la politique approximée est au plus $O(v \times SV_i)$ où v est le nombre de votants et SV_i le nombre de vecteurs de support du votant i. Pendant l'apprentissage, cependant, la complexité en espace doit en plus prendre en compte la taille de l'oracle (ici, $O(SA)$). Enfin, la complexité en temps est principalement dominée par la complexité de mise à jour de l'algorithme de classification.

Les résultats présentés ici ne garantissent aucune convergence théorique. Cependant nous avons de bonnes raisons de croire que notre algorithme va trouver une politique proche de l'optimale et ce, grâce au fait que la fonction de valeur va converger vers la fonction de valeur optimale et que, grâce au vote de majorité et au seuil qui diminue, on s'assure que la politique apprise va en moyenne converger vers la meilleure politique approximable par l'algorithme de classification. De nombreux travaux restent à faire sur l'absence de l'hypothèse iid des ensembles d'apprentissages de notre approche. Cependant, il existe plusieurs mécanismes possibles pour tenter de remédier à ce type de problèmes [17] [3].

Les algorithmes de classification utilisés pour approximer la politique optimale sont généralement définis pour deux classes seulement. C'est pourquoi, dans les problèmes à deux actions, MajSPRL utilise un ensemble de classificateurs binaires ainsi qu'un simple vote de majorité entre ces classificateurs pour approximer la politique optimale. Cependant, pour les problèmes à plus de deux actions, une adaptation de notre algorithme serait nécessaire. Pour le problème de classification, de nombreuses approches ont été proposées pour gérer plus de deux classes. L'approche la plus commune consiste à combiner plusieurs classificateurs binaires pour choisir parmi plus de deux classes. Hsu et Lin [12] présentent certaines de ces approches telles que le un-contre-tous ou bien l'approche un-contre-un.

7 Travaux connexes

Peu d'approches algorithmiques qui combinent l'apprentissage par renforcement en ligne et la classification ont été proposées. Parmi les travaux existants, Lagoudakis et Parr ont proposé un algorithme qui combine l'apprentissage par renforcement et la classification [14]. Cet algorithme se base sur l'algorithme d'itération de politique approximé (API) qui utilise un Rollout [4] pour estimer la fonction de valeur de la paire (s, a) courante. Puis, un SVM est utilisé pour approximer la politique. Fern *et. al* [11] ont proposé également un algorithme basé sur API et un Rollout qui approxime la politique mais plus d'un point de vue planification et non du point de l'apprentissage comme on le fait ici. De par l'utilisation de l'algorithme de Rollout, ces algorithmes utilisent un modèle génératif d'états contrairement à notre algorithme qui utilise un modèle à base de traces dans lequel on ne peut pas initialiser l'environnement à un état quelconque désiré.

8 Conclusion

Cet article a présenté une approche qui approxime la politique optimale d'un PDM en combinant un algorithme d'apprentissage par renforcement, un algorithme de classification et un vote de majorité. Nous avons montré empiriquement que cette combinaison permet d'obtenir une politique approximée très proche de la politique optimale tout en faisant mieux que l'apprentissage par Q-valeur tout seul. Cependant, de nombreux travaux sont nécessaires pour déterminer dans quelles conditions approximer la politique est plus avantageux que d'approximer la fonction de valeur. Enfin, pour tester l'extensibilité de notre algorithme, nous prévoyons d'étendre notre approche à un nombre quelconque d'actions et de la tester sur des problèmes réels et multiagents comme le problème de coordination de véhicules [18].

Remerciements

Nous remercions François Laviolette, Camille Besse ainsi que les évaluateurs anonymes pour leurs commentaires sur ce problème.

Références

[1] C. W. ANDERSON : Approximating a policy can be easier then approximating a value function. Rap. tech., Colorado State University, 2000.

[2] E. BAUER et R. KOHAVI : An empirical comparison of voting classification algorithms : Bagging, boosting and variants. *Machine Learning*, 36:105–142, 1999.

[3] S. BEN-DAVID, J. BLITZER, K. CRAMMER et F. PEREIRA : Analysis of representation for domain adaptation. *In Neural Information Processing Conference (NIPS)*, 2006.

[4] D. P. BERTSEKAS et D. A. CASTANON : Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5:89, 1999.

[5] D. BLATT et A. O. HERO : From weighted classification to policy search. *In Advances in Neural Information Processing Systems (NIPS)*, vol. 18, p. 139–146, 2006.

[6] C.-C. CHANG et C.-J. LIN : *LIBSVM : a library for support vector machines*, 2001. Software available at http ://www.csie.ntu.edu.tw/ cjlin/libsvm.

[7] F. D. COMITÉ : Piqle : http ://sourceforge.net/projects/piqle, 2006.

[8] A. CORNUÉJOLS et L. MICLET : *Apprentissage artificiel, concepts et algorithmes*. Eyrolles, 2002.

[9] K. CRAMMER, J. KANDOLA et Y. SINGER : Online classification on a budget. *In* S. THRUN, L. SAUL et B. SCHOLKOPF, éds : *Sixteenth Annual Conference on Neural Information Processing Systems (NIPS)*, 2003.

[10] A. DUTECH, T. EDMUNDS, J. KOK, M. LAGOUDAKIS, M. LITTMAN, M. RIEDMILLER, B. RUSSELL, B. SCHERRER, R. SUTTON, S. TIMMER, N. VLASSIS, A. WHITE et S. WHITESON : Reinforcement learning benchmarks and bake-offs ii. Workshop at the 2005 NIPS conference, 2005.

[11] A. FERN, S. YOON et R. GIVAN : Approximate policy iteration with a policy language bias. *In Neural Information Processing Conference (NIPS)*, 2003.

[12] C.-W. HSU et C.-J. LIN : A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, March 2002.

[13] N. K. JONG et P. STONE : Model-based function approximation in reinforcement learning. *In The Sixth Internationnal Conference on Autonomous Agents and Multiagent Systems (AAMAS07)*, 2007.

[14] M. G. LAGOUDAKIS et R. PARR : Reinforcement learning as classification : Leveraging modern classifier. *In ICML-2003*, Washington DC, 2003.

[15] J. LANGFORD et A. BEYGELZIMER : Sensitive error correcting output codes. *In Proceedings of the Eighteenth Annual Conference on Learning Theory (COLT-2005)*, 2005.

[16] J. LANGFORD et B. ZADROZNY : Reducing T-step reinforcement learning to classification. *In Proc. of Machine Learning Reductions Workshop*, 2003.

[17] J. LANGFORD et B. ZADROZNY : Relating reinforcement learning performance to classification performance. *In Proceedings of International Conference on Machine Learning 2005*, 2005.

[18] D. E. MORIARTY et P. LANGLEY : Distributed learning of lane-selection strategies for traffic management. Rap. tech., Daimler-Benz Research & Technology Center, Palo Alto, CA, Palo Alto, CA., 1998. 98-2.

[19] R. MUNOS et A. W. MOORE : Variable resolution discretization for high-accuracy solutions of optimal control problems. *In IJCAI*, p. 1348–1355, 1999.

[20] D. ORMONEIT et S. SEN : Kernel-based reinforcement learning. Rap. tech. TR 1999-8, Statistics, Stanford University, 1999.

[21] S. J. RUSSELL et P. NORVIG : *Artificial Intelligence : A Modern Approach (2nd Edition)*. Prentice Hall, 2002.

[22] B. SCHOLKOPF, A. SMOLA, R. WILLIAMSON et P. BARTLETT : New support vector algorithms. *Neural Computation*, 12:1083–1121, 2000.

[23] R. S. SUTTON et A. G. BARTO : *Reinforcement Learning : An Introduction*. MIT Press, Cambridge, MA, 1998.

[24] G. TESAURO : Temporal difference learning and TD-gammon. *Communication of the ACM*, 38(3):58–68, 1995.

[25] S. THRUN : The role of exploration in learning control. *In* D. WHITE et D. SOFGE, éds : *Handbook for Intelligent Control : Neural, Fuzzy and Adaptive Approaches*. Van Nostrand Reinhold, Florence, Kentucky 41022, 1992.

[26] C. WATKINS et P. DAYAN : Q-learning. *Machine Learning*, 8:279–292, 1992.

[27] J. WESTON, A. BORDES et L. BOTTOU : Online (and offline) on an even tighter budget. *In AISTATS*, 2005.