Boosting Active Learning to Optimality: a Tractable Monte-Carlo, Billiard-based Algorithm

Philippe Rolet, Michèle Sebag, and Olivier Teytaud

TAO, CNRS − INRIA − Univ. Paris-Sud
rolet@lri.fr, sebag@lri.fr, olivier.teytaud@inria.fr

Abstract. This paper focuses on Active Learning with a limited number of queries; in application domains such as Numerical Engineering, the size of the training set might be limited to a few dozen or hundred examples due to computational constraints. Active Learning under bounded resources is formalized as a finite horizon Reinforcement Learning problem, where the sampling strategy aims at minimizing the expectation of the generalization error. A tractable approximation of the optimal (intractable) policy is presented, the *Bandit-based Active Learner* (*BAAL*) algorithm. Viewing Active Learning as a single-player game, *BAAL* combines UCT, the tree structured multi-armed bandit algorithm proposed by Kocsis and Szepesvári (2006), and billiard algorithms. A proof of principle of the approach demonstrates its good empirical convergence toward an optimal policy and its ability to incorporate prior AL criteria. Its hybridization with the Query-by-Committee approach is found to improve on both stand-alone *BAAL* and stand-alone QbC.

1 Introduction

Active Learning (AL), a most active topic in supervised Machine Learning (ML) [1–7], aims at accurately approximating a target concept with a limited number of *queries* to an *oracle*; each query asks the oracle to label a point of the problem domain (*instance*) according to the target concept. Through a judicious selection of instances to query, the hope is to learn with a significantly smaller number of queries than in the standard ML setting using iid labeled instances. Prominent AL approaches (section 2) rely on the properties of the hypothesis space (VC dimension or covering numbers) and/or propose various criteria estimating the additional information provided by an instance; they mostly proceed by iteratively selecting the optimal instance in the sense of the considered criterion.

This paper presents a new perspective on AL, formalized as a finite time horizon reinforcement learning problem. Choosing an instance, i.e. making a query is viewed as an action taken by the learner. The reward associated to a sequence of actions is the generalization error of the hypothesis learned from the training set gathered from this sequence of actions. Under mild assumptions (Bayesian realizable setting) detailed in section 3, this paper offers a provably optimal AL strategy, parameterized by the initial training set and the finite horizon T, i.e. the maximum number of queries allowed.

As could have been expected, the formal derivation of this optimal AL strategy is intractable. A tractable approximation thereof is given by the *Bandit-based Active Learner* algorithm (*BAAL*, section 4). This tractable approximation involves two main ingredients. Firstly, the exploration of the AL search space relies on UCT, the tree-structured multi-armed bandit algorithm originally introduced by Kocsis and Szepesvari [8]. The use of UCT is motivated as AL can be formulated as a single player game (section 4) and UCT has been shown quite efficient in learning game strategies [9]. Secondly, an unbiased and frugal sampler of the hypothesis and instance spaces, based on billiard-based mechanisms [10–12] supports the *Monte-Carlo simulations* in UCT. A proof of principle of the *BAAL* algorithm is presented, showing its practical robustness for AL under bounded resource constraints. This bounded resource constraint is motivated by targeted applications, aimed at simplified models in Numerical Engineering: in this context, training sets are limited to a few dozen or hundred labeled instances due to the high computational cost of instance labeling.

The paper is organized as follows. Section 2 briefly introduces the notations used throughout the paper and reviews the state of the art. Section 3 formalizes AL as a reinforcement learning problem, and shows how to tackle it as a single-player game. The online learning algorithm *BAAL*, implementing this single-player game using the UCT algorithm, is described in section 4; it takes as input a training set and the available computational budget, and finds the best instance to label next. Section 5 details two extensions brought to UCT in order to match the AL context. Firstly, as UCT only addresses finite action/state spaces, it needs to be extended to explore the continuous instance space; the proposed approach relies on *progressive widening* [13–15]. A second extension regards the hybridization of *BAAL* with prior knowledge, such as existing AL criteria. The proposed hybridization will be illustrated by embedding a variant of Query-by-Committee (QbC, [16, 17]) within the progressive widening heuristics. An experimental validation of the approach is presented in section 6, using passive learning and QbC as baselines, and the paper concludes with some perspectives for further research.

2 Background and State of the art

Notations and definitions used in the paper are as follows. Let $s_t = \{(x_i, y_i), x_i \in X, y_i \in Y, i = 1 \ldots t\}$ denote a t-size training set, with X the instance space and Y the label space; only the binary classification case ($Y = \{0, 1\}$) will be considered in the paper.

From s_t, a learning algorithm $\mathcal{A}$ extracts some hypothesis h in hypothesis space $\mathcal{H}$, mapping X onto Y. The learning performance most usually refers to the expectation of the loss $\ell(h(x), h^*(x))$ incurred by h over the *target concept* h^* a.k.a. oracle, where the expectation is taken over the joint distribution P_{XY} on the problem domain. Whereas the standard supervised learning setting assumes that training examples (x_i, y_i) are independent and identically distributed (i.i.d.) after P_{XY}, AL selects at time step i some instance x_i in the instance space X

(or in a pool of instances drawn from X), the label y_i of which is determined by the oracle.

A sampler S is a mapping from $\bigcup_{t\in\mathbf{N}}(X\times Y)^t$ to X, also referred to as *policy* or *strategy*, selecting a new instance x to be labeled depending on training set s_t. A learner $\mathcal{A}$ is a mapping from $\bigcup_{t\in\mathbf{N}}(X\times Y)^t$ to $\mathcal{H}$ associating a hypothesis h to any training set s_t. The *Version Space* (VS) associated to a training set s_t, noted $\mathcal{H}(s_t)$, is the set of hypotheses consistent with s_t, i.e. such that $h(x_i) = y_i$ for $1 \leq i \leq t$.

A first research direction focuses on the *uncertainty region* (set of examples where VS hypotheses disagree). Cohn et al. [2] reduce the VS by characterizing the uncertainty region via neural nets, and selecting new instances in this region. Well-known methods such as Query-by-Committee (QbC) algorithms [16, 17] directly reduce the VS volume; Freund et al. [17] proved that these methods can lead to an exponentially smaller number of queries than the random querying strategy (passive learning), at least in the case of perceptrons. Dasgupta [4] has established the quasi-optimality of these methods in the realizable classification case (i.e. when h^* belongs to $\mathcal{H}$) for a finite instance pool. More generally, when there exists a probability measure P_H on $\mathcal{H}$, usual AL strategies are concerned with reducing either the measure of the Version Space, or the variance of the VS hypothesis labels. Otherwise, the reduction of the Version Space can be expressed in terms of its diameter, that is the measure of points where hypotheses in the VS differ.

A related research direction focuses on *error reduction*, meant as the expected generalization error improvement brought by an instance. Many criteria reflecting various measures of the expected error reduction have been proposed [18–21]; corresponding AL strategies proceed by greedily selecting the optimal instances in the sense of the considered criterion. Other approaches exploit prior, learner-dependent knowledge about what makes an instance informative, such as its margin [3, 22, 23]. Hoi et al. [6] consider *batch* active learning, querying a subset of instances that results in the largest reduction of the Fisher information. Some of these approaches however happen to face learning instabilities, which might require to mix the AL procedure with a uniform instance selection [24]. Such instabilities suggest that in some cases an optimally efficient AL system can hardly be based on a greedy selection strategy, at least using the criteria considered so far.

On the theoretical side, significant results have been obtained in terms of lower and upper bounds on the reduction of the sample complexity brought by AL, e.g. depending on the complexity of the hypothesis search space measured through covering numbers or Kolmogorov complexity [1, 25, 5]; the appropriateness of hypothesis spaces to AL has been studied [26, 27, 7] and an "almost" optimal (though intractable) algorithm has proposed by [27] in the realizable setting for finite VC-dimension.

3 An optimal active learning strategy: AL as a Markov decision process

This section presents a theoretically optimal strategy for Active Learning in *finite time horizon* T, where T corresponds to the total number of instances to be labeled along the AL process. As in [17, 4], the proposed approach is built on a Bayesian setting [29, 28]; prior knowledge about the target concept is accounted for by a probability distribution P_H on the hypothesis space[1] $\mathcal{H}$. Further, the approach relies on the *realizable* assumption, i.e. the target concept h^* to be learned is assumed to be deterministic and to belong to $\mathcal{H}$ (how to relax this assumption will be discussed in section 7).

Let us formalize AL as a Markov Decision Process (MDP). MDPs are classically described in terms of states, actions, reward, policy and transition functions [30]. In the AL context, the state space $\mathcal{S}$ consists of all possible training sets s_t. An action corresponds to the selection of a new instance to be labeled; the set of actions noted A thus coincides with the instance space $\mathcal{X}$ or a subset thereof.

The reward function associated to state s_t corresponds to the generalization error of the hypothesis $\mathcal{A}(s_t)$ learned from s_t by learner $\mathcal{A}$. In many finite time horizon settings, and particularly so for the targeted applications, the reward at the final state of the learning is the only one that matters. Accordingly, the reward function will be defined for *horizon* states only, and assimilated to the *value function* of those states (see below).

The transition function $p : \mathcal{S} \times A \times \mathcal{S} \rightarrow \mathbb{R}_+$ defines the probability of arriving at some state s_{t+1} by selecting action x in state s_t (see below). Lastly, in the AL context a MDP policy is a sampler S, mapping a state s_t onto an action, i.e. a new instance x_{t+1}.

Considering horizon T, let $S_T(h)$ denote the training set built by applying T times policy S when learning some target concept h, and let $\mathbf{Err}(\mathcal{A}(S_T(h)), h)$ denote the generalization error of the hypothesis learned from $S_T(h)$. It comes naturally that an optimal AL strategy is one minimizing the expectation of $\mathbf{Err}(\mathcal{A}(S_T(h)), h)$ when h ranges in $\mathcal{H}$:

$$S_T^* = \arg\min_S \mathbb{E}_{h \sim \mathcal{H}} \mathbf{Err}(\mathcal{A}(S_T(h)), h). \tag{1}$$

The main motivation behind the presented MDP framework is to build a policy with optimal behavior in the sense of Eq. (1). In order to do so, it remains to find the appropriate reward and transition probability functions, and such that the latter accounts for the actual behavior of the AL process.

Regarding the reward function, as the goal is to minimize the generalization error of the hypothesis learned from s_T, the reward is only known at horizon T. After the realizable assumption, the target concept h is bound to be in the Version Space of s_T, noted $\mathcal{H}(s_T)$. The reward function $V(s_T)$ (value at horizon T) therefore is the expectation of the generalization error of the learner $\mathcal{A}$, taken over $\mathcal{H}(s_T)$:

$$V(s_T) = \mathbb{E}_{h \sim \mathcal{H}(s_T)} \mathbf{Err}(\mathcal{A}(s_T), h). \tag{2}$$

[1] P_H is set to the uniform distribution on $\mathcal{H}$ in the absence of prior knowledge.

By construction, the transition function $p(s_t, x, s_{t+1})$ is such that p is zero for all s_{t+1} except the ones satisfying $s_{t+1} = (s_t, (x_{t+1} = x, y_{t+1}))$ for some y_{t+1}. In the latter case, p reflects the probability for the label of x to be y_{t+1}. $p(s_t, x, s_{t+1})$ is thus expressed as a function of the probability of the label of x, conditionally to the fact that the target hypothesis belongs to $\mathcal{H}(s_t)$:

$$p(s_t, x, (s_t, (x_{t+1} = x, y_{t+1}))) = p(h(x) = y_{t+1} | h \in \mathcal{H}(s_t)) \tag{3}$$

The above transition and value functions guarantee that the optimal MDP strategy achieves the AL goal and minimizes the expected generalization error (Eq. (1)):

Theorem 1 (Optimal AL policy). *Let $\mathbb{E}$ denote the expectation operator defined after Eq. (3). Let value function $V^{*,T}$ be recursively defined as follows, where $|s|$ denotes the number of examples in training set s:*

$$V^{*,T}(s) = \begin{cases} \mathbb{E}_{h \sim \mathcal{H}(s)} \mathbf{Err}(\mathcal{A}(s), h) & \textit{if } |s| = T \\ \inf_{x \in X} \mathbb{E}_{s' \sim p(s'|x, s_t)} V^{*,T}(s') & \textit{otherwise} \end{cases} \tag{4}$$

Define $S^{*,T}$ as $S^{*,T}(s) = \arg\inf_{x \in X} \mathbb{E}_{s' \sim p(s'|x,s_t)} V^{*,T}(s')$ (Bellman optimal strategy). Then $S^{*,T}$ is optimal in the sense of Eq. (1).

The proof is given in [31] due to space limitations. It essentially shows that the value function of a policy at the initial state coincides with the criterion in Eq. (1). The rest of the proof is a direct application of Bellman's optimality principle [32].

After this MDP formulation, AL can be seen as a one-player game. The active learner plays against the (unknown) target hypothesis h, belonging to the version space $\mathcal{H}(\mathbf{s}_0)$ of the initial training set[2] s_0. Upon each move (selection of some instance x), oracle h provides the label $y = h(x)$. At the end of the game − that is, after T examples have been picked, defining training set s_T − the reward is the generalization error of the hypothesis $\mathcal{A}(s_T)$ learnt from s_T. The *real* learning game is indeed played against oracle h.

It is however possible to train the AL player, and therefore devise a good AL policy beforehand, by mimicking the above game and playing against a "surrogate" oracle, made of a hypothesis h uniformly selected in the VS. The reward of such a game is computed as in the real game: it is the generalization error of the learned hypothesis w.r.t. the (surrogate) oracle. This reward implements a uniform draw of the random variable $\mathbf{Err}(\mathcal{A}(s), h)$ for h ranging in $\mathcal{H}$.

By construction, the average empirical reward collected by the AL player after many such games asymptotically converges toward the true expectation of this random variable, that is, the desired reward function (Eq. (2)).

[2] For the sake of simplicity and when no confusion is to fear, the initial training set $\mathbf{s}_0$ is omitted and $\mathcal{H}$ is used instead of $\mathcal{H}(\mathbf{s}_0)$.

4 Tractable Approximations of Optimal AL strategy

This section presents a consistent and tractable approximate resolution of the above MDP, the *BAAL* algorithm. *BAAL* relies on two main ingredients. Firstly the tree-structured multi-armed bandit UCT [8] is extended to the one-player game of AL. Secondly, a fair and frugal billiard-based algorithm [10, 11] is used to sample the instance and hypothesis spaces.

4.1 Bandit-based Active Learning

UCT (Upper Confidence Tree) is a Monte-Carlo tree-search algorithm [8, 13], where the selection of a child node is cast into a multi-armed bandit problem [33]. UCT notably became famous in the domain of strategic games as it inspired the computer-Go program MoGo, first to ever win over professional human players [9].

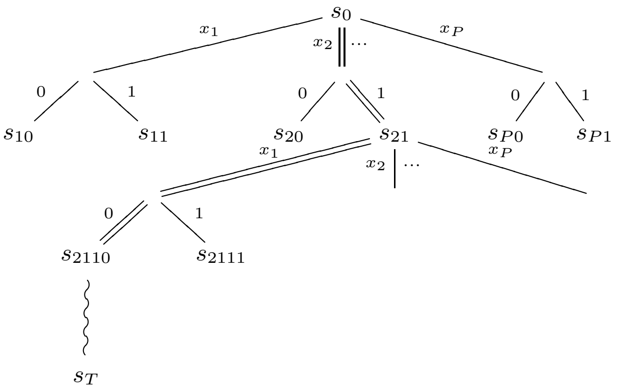


Fig. 1. Search Tree developed by *BAAL*, the Bandit Based Active Learner, in the binary classification case. The root of the tree is the initial training set s_0. Each tree walk is based on selecting some hypothesis h in the Version Space of s_0. The tree walk proceeds by iteratively selecting an instance x, whose label is set to $h(x)$. Ultimately, the tree walk is assessed from the generalization error $Err(\mathcal{A}(s_T), h)$ of the learned hypothesis w.r.t. h.

The UCT-based game strategy provides the basis for *BAAL* (Fig. 4). Let $\mathcal{T}$ denote the "game" tree of AL; its root node is the initial training set $\mathbf{s}_0$. It is worth noting that *BAAL* actually explores a directed graph (a given node can be reached along different paths) although it is presented as a tree-search algorithm for simplicity.

Each tree-walk (aka game or simulation) is indexed by a surrogate hypothesis h (see below) and proceeds as follows. Each node corresponding to a training set s_t is called a *state* node. Its child nodes, referred to as *decision* nodes, stand for the actions of the AL player, that is, the possible instances to be selected. A decision node thus is made of the training set s_t (parent node), and the chosen instance x_{t+1}.

Every decision node has two child nodes (the possible labels of the instance x_{t+1}). The one selected during the tree-walk indexed by h obviously corresponds to $h(x_{t+1})$, thus leading to the next state node $s_{t+1} = s_t \cup (x_{t+1}, h(x_{t+1}))$.

More precisely, each tree-walk proceeds as follows (Fig. 4; *BAAL* pseudo-code is given in Alg. 1):

- Firstly, a surrogate hypothesis h is uniformly drawn in the version space $\mathcal{H}(s_0)$ of the initial training set s_0;
- In each state node (i.e. some training set s_t), *BAAL* uses the famed Upper Confidence Bounds (UCB) criterion (Eq. (5), see below) to select some decision node, i.e. some instance x_{t+1} to be labeled. The associated label is set to $h(x_{t+1})$, where h is the surrogate hypothesis. The next state node thus is $s_{t+1} = s_t \cup (x_{t+1}, h(x_{t+1}))$.
- The tree-walk proceeds until arriving in a tree leaf, i.e. a state node s_{t_0} not yet visited. At this point, $T - t_0$ additional instances are uniformly selected, labeled after h, and added to the training set s_{t_0}, forming a T-size training set s_T.
- From training set s_T, a hypothesis $\hat{h}$ is learned by *BAAL*[3].
- The reward of the tree walk is the generalization error of $\hat{h}$ with respect to the surrogate hypothesis h: this reward is used to update the value of every relevant node.

The value $\hat{\mu}(s_t)$ of state node s_t is computed by averaging all rewards gathered for nodes s_T such that $s_t \subseteq s_T$. These values are exploited by the UCB criterion [33], determining which arm (instance aka decision node) should be selected. The arm selected is the one maximizing the sum of the empirical reward $\hat{\mu}_i$ (exploitation term), plus an exploration term depending on the number n_i of times arm i has been selected. The exploration vs exploitation tradeoff is adjusted using some tuned constant C:

$$\arg \max_{i \in \text{child nodes}} \hat{\mu}_i + C \sqrt{\frac{\log(\sum_{j \in \text{child nodes}} n_j)}{n_i}} \tag{5}$$

In the multi-armed bandit setting, this formula guarantees a provably optimal convergence towards the arms with maximal value, under the assumption that arm rewards are independent random variables. (Clearly this assumption does not hold in the AL game, no more than in the game of Go [34]).

The memory-wise computational tractability of *BAAL* is ensured by gradually developing the tree, initially made of the root node and its child nodes only. After each tree-walk, the first randomly selected node is stored in memory with its child nodes: the current leaf node will no longer be a leaf node, and

[3] The learning algorithm actually is a parameter of *BAAL*. In the experiments, we used a uniform sampler of the version space of s_T.

Algorithm 1 The ***BAAL*** algorithm:

Input: measure P_H on hypothesis space $\mathcal{H}$; initial training set $\mathbf{s}_0$; time horizon T; number N of allowed tree-walks;

Output: an instance x to be labelled by the oracle.

```
$\boldsymbol{BAAL}(P_H, \mathbf{s}_0, T, N)$
for i=1 to N do
    $h$ = DrawHypothesis($P_H, \mathbf{s}_0$)
    Tree-Walk($\mathbf{s}_0, T, h$)
end for
Return $x = \arg\max_{x' \in \mathcal{X}} \{n(s \bigcup \{x'\})\}$

Tree-Walk($\mathbf{s}, t, h$)
Increment $n(\mathbf{s})$
if t==0 then
    Compute $r = Err(\mathcal{A}(\mathbf{s}), h)$
else
    $\mathcal{X}(\mathbf{s})$ = ArmSet($\mathbf{s}, n(\mathbf{s})$)
    Select $x^* = UCB(\mathbf{s}, \mathcal{X}(\mathbf{s})))$
    $r$ = Tree-Walk($\mathbf{s} \bigcup \{(x^*, h(x^*))\}, t-1, h$)
end if
$r(\mathbf{s}) \leftarrow (1 - \frac{1}{n(\mathbf{s})}) r(\mathbf{s}) + \frac{1}{n(\mathbf{s})}\ r$
Return $r$
```

the next time it is encountered, the selection among its child nodes will rely on the UCB formula. Thereby the tree is asymmetrically grown, developing more the subtrees where nodes have better values − since those will be selected more often.

BAAL (Alg. 1) implements the UCT scheme with two specific ingredients. The `DrawHypothesis` function selects the surrogate hypothesis h attached to each tree-walk using billiard algorithms (see below). The `ArmSet` function (section 5.1) extends UCT to deal with an infinite set of actions (the continuous instance space), using the so-called progressive widening heuristics.

4.2 Billiard algorithms

As already mentioned, each tree-walk in *BAAL* is indexed by a hypothesis h uniformly sampled in version space $\mathcal{H}(s_0)$. The most straightforward sampling algorithm is based on *rejection*: hypotheses are uniformly drawn in $\mathcal{H}$ and rejected if they do not belong to the VS (if they are inconsistent with the training set s_0). Unfortunately, the rejection algorithm, although sound, is hardly tractable. Alternative algorithms such as Gibbs sampling or more generally Monte-Carlo Markov Chains (MCMC) methods, involve quite a few free parameters and might scale poorly with respect to the dimensionality of the search space and the size of the training set s_0. We thus used a billiard (a.k.a. ray-tracing) algorithm inspired from [10, 35, 11], assuming that the hypothesis space $\mathcal{H}$ can be parameterized by $\mathbb{R}^d$. Actually, the experimental validation (section 6) considers linear hypotheses; how to go beyond the linear case will be discussed in section 7.

Let Ω be a connected subset of $\mathbb{R}^d$, defined by a set of constraints $g_1, \ldots, g_n$: $\Omega = \{x \in \mathbb{R}^d \; s.t. \; g_i(x) \geq 0, i = 1 \ldots n\}$. The billiard algorithm considers a point $z \in \mathbb{R}^d$ (not necessarily in Ω) and a direction $\boldsymbol{v}$ ($\boldsymbol{v} \in \mathbb{R}^d, ||\boldsymbol{v}|| = 1$). The trajectory followed by z is such that: i) the set of constraints satisfied by z does not decrease; ii) z "bounces" when it meets an active constraint g, i.e. its direction v is changed to point inside the domain (see below); iii) the trajectory is stopped when the computational resources are exhausted, that is when its total length reaches some user-defined parameter L, and the final point is returned. Under a few regularity conditions on the constraints, a billiard trajectory is ergodic, i.e. it covers the whole domain when L goes to infinity [36]; the distribution of the final trajectory point converges toward the uniform distribution on Ω. Billiard algorithms have been successfully used in Machine Learning, e.g. to estimate the Bayes classifier in a (high-dimensional) kernel feature space [10, 35].

Rebound policy
Let g_i be the saturated constraint and z_i the rebound point. The rebound policy sets the current direction $\boldsymbol{v}$ to a unit vector randomly drawn in the half sphere centered on z_i and defined from the hyperplane tangent to g_i in z_i. Experimentally, this policy efficiently approximates the Knudsen law advocated by Comets et al. [36], which is more expensive to compute. This matter is however outside the scope of this work.

5 UCT with continuous domain and AL criteria

This section is devoted to specific adaptations of UCT involved in BAAL: how to deal with a continuous action set (the instances) and how to take advantage of existing AL criteria, such as Query-by-Committee [16].

5.1 Progressive Widening

UCB originally requires each arm to be selected at least once for Eq. (5) to be computed. When the number of arms is large with respect to the number N of simulations, UCB thus tends to degenerate into pure exploration. UCT faces the same limitation, and even more so since many arms need be considered at each node of the tree, strongly biasing the search towards exploration.

The *progressive widening* (PW) technique has been proposed by [13] to handle such cases. Let n_s denote the number of times node s has been visited so far; then the number of arms that can be considered from s is limited to a fraction m of n_s. Empirical and theoretical studies suggest $m = O(n_s^{1/4})$ [13–15]. In practice, the `ArmSet` procedure implementing PW in *BAAL* (Algorithm 1), considers a finite set of options for each node (see below), and a new option is added to this set whenever $\lfloor n_s^{1/4} \rfloor$ is incremented by one. Typically, a single arm will be explored from the root node in the first fifteen tree-walks; an additional arm is considered in the sixteenth random walk; UCB will be used to select among both arms during random walks 16 to 80; a third arm is considered in random walk 81, etc. The rationale behind progressive widening is that the better (i.e. the more visited) s, the more careful the investigation of its subtrees should be.

5.2 Integrating AL criteria in *BAAL*

The selection of the actions (instances) considered at a given time step by `ArmSet` offers some room for the use of prior knowledge. The simplest selection procedure indeed is based on the uniform sampling of the instance space. This procedure reflects an agnostic viewpoint, making no assumption about the utility of instances. It must be noted however that *BAAL* convergence might be delayed (like other UCT-based algorithms) if the optimal options are considered late during the search: instances introduced later on are clearly disadvantaged compared to the earlier, more investigated, instances.

Furthermore, the AL literature suggests that some selection criteria − although not optimal − offer generally sound indications in order to select informative instances (section 2). Such criteria can seamlessly be integrated in *BAAL* through the progressive widening procedure. Formally, a new instance is added to `ArmSet` when it satisfies the considered criterion, instead of being uniformly selected in the pool of instances[4].

5.3 QbC-*BAAL*: *BAAL* with Maximal Uncertainty

Maximal uncertainty (MU) (section 2) is a criterion stemming from Query-by-committee (QbC) algorithms. QbC works by randomly sampling hypotheses in the Version Space, and choosing a given instance only if enough hypotheses disagree about its label. An analysis of this strategy with a committee of size 2 is for instance presented in [17]. In the case of large-sized committees, one proceeds by ranking instances based on the committee disagreement: the top-ranked instances indeed correspond to those with maximal uncertainty.

The QbC approach is integrated within BAAL as follows. At each node (training set s_t) a committee of hypotheses is built by uniformly sampling the Version Space of s_t. Whenever a new instance is to be added to ArmSet, one selects the one maximizing the committee disagreement.

MU is an *aggressive* criterion, thus holding a higher AL potential than random progressive widening. In counterpart it leads to a less diversified sample; whenever the criterion is under-optimal, the limited exploration will yield poorer performance. It is also more demanding computationally.

6 Experimental validation

This section reports on the experimental study of *BAAL*. After describing the experimental goal and setting, empirical results are discussed comparatively to ran-

[4] Let us emphasize that hybridizing *BAAL* with any such criterion does not depend on the true instance labels.

dom active learning, referred to as passive learning, and the Query-by-Committee (QbC) approach [16, 17].

6.1 Goal of experiments

The main questions investigated in the experiments regard the theoretical and computational performance of *BAAL*. Specifically:
Question 1 (Optimality): Does the stand-alone *BAAL*, where the progressive widening heuristics involves a uniform instance selection, converge to an optimal strategy; does it match QbC results when these are known to be quasi-optimal after Dasgupta [4] ?
Question 2 (Flexibility): As shown in section 5.1, *BAAL* can embed existing AL criteria, e.g. QbC, within the progressive widening heuristics. Does the use of QbC within *BAAL* improve i) on stand-alone *BAAL* ? ii) on stand-alone QbC ?

6.2 Experimental setting

Following [17, 21, 23], the instance space $\mathcal{X}$ considered in these experiments is the unit sphere of $\mathbb{R}^d$. The hypothesis space $\mathcal{H}$ is restricted to the linear classifiers, a.k.a. separating hyperplanes. The choice of this search space is motivated as it has been thoroughly studied from a theoretical perspective [17, 21, 23] although these results did not lead to experimental studies to our best knowledge. Accordingly, some upper and lower bounds on the optimal performance are available, and will be used to assess *BAAL* performance. The goal of this experimental study is to provide a proof of principle regarding the validity of this new AL approach; still, it must be emphasized that *BAAL* is not limited to linear hypothesis spaces (a kernelized extension will be discussed in section 7).

Two variants of *BAAL* will be considered. *BAAL* stand-alone (or *BAAL* for short), uses a uniform selection of instances within progressive widening, whereas QbC-*BAAL* uses a committee-based selection of instances. More precisely, in each node (training set s_t), a committee of 100 hypotheses uniformly selected in $\mathcal{H}(s_t)$ is built, and a set of $10,000$ instances uniformly drawn from $\mathcal{X}$ is sampled and ordered after the committee disagreement. Whenever a new action is to be considered ($\lfloor n(s_t)^{1/4} \rfloor$ increased by one), the first instance not yet considered in the ordered set is returned by `ArmSet`.

A hypothesis h is characterized as a unit vector w_h ($w_h \in \mathbb{R}^d,\ ||w_h||_2 = 1$), where $h(x)$ is positive if and only if the scalar product $< w_h, x >$ is positive. The lack of prior knowledge is accounted for by setting distribution P_H to the uniform distribution on the unit sphere of $\mathbb{R}^d$.

A *BAAL* run proceeds as follows:

1. The target concept h^* is uniformly selected in $\mathcal{H}$.
2. For $t = 0$ to T, where T is the horizon time and the initial training set s_0 is empty:
   (a) The *BAAL* tree rooted on s_t is constructed using N tree-walks, respectively using a randomly (a QbC-) ordered instance pool in the progressive widening heuristics for *BAAL* (resp. QbC-*BAAL*);

(b) The best instance, i.e. the most visited one at the first level of the s_t rooted tree, noted x_{t+1}, is selected and labeled after the target concept (oracle) h^*;

(c) $s_{t+1} = s_t \cup \{(x_{t+1}, h^*(x_{t+1})\}$.

3. From s_T, the T-size training set built by $BAAL$ (QbC-$BAAL$) and labeled after the current target concept h^*, a hypothesis h_T is learned (uniformly sampled in the Version Space $\mathcal{H}(s_T)$).
4. The performance of the run, that is, the generalization error $\mathbf{Err}(h, h^*)$ defined as $P_{x\sim\mathcal{X}}(h^*(x) \neq h(x))$, is computed as the dot product of the associated vectors w_h and w_{h^*}.

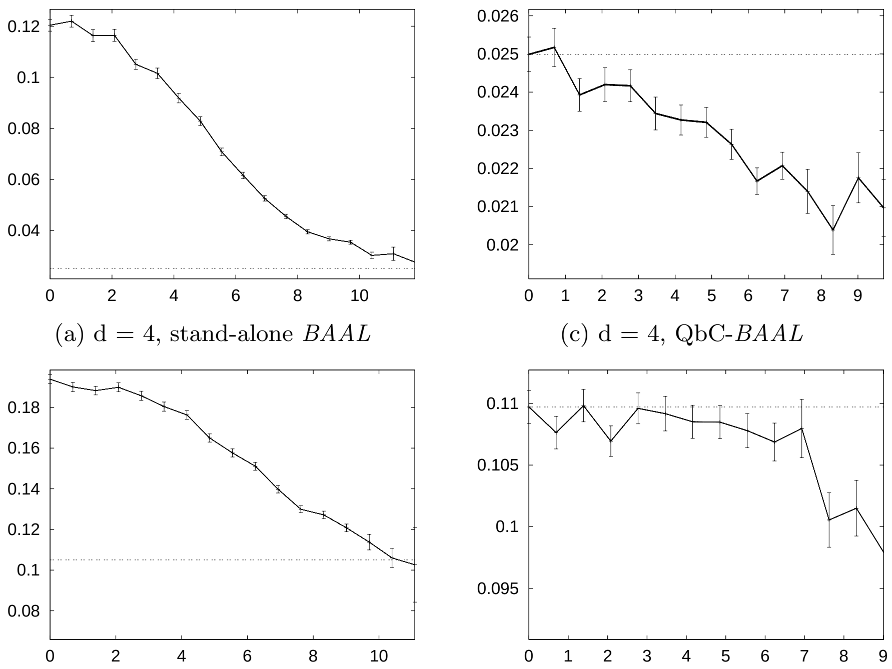
$BAAL$ and QbC-$BAAL$ performances are averaged over 400 independent runs for each 3-uple {d,T,N} (dimension, horizon, simulation number). The number N of simulations, controlling the computational cost, ranges in $\{1, 2, 4, \ldots N = 2^{12}\}$. The reported experiments consider ($d = 4, T = 15$) and ($d = 8, T = 20$) to assess the scalability of the approach. The performance is plot against the number N of tree-walks, Fig. 2, indicating the average performance (plain line) and the standard deviation (vertical bars). The performance of $BAAL$ stand-alone (respectively QbC-$BAAL$) is assessed comparatively to the passive learning (resp. the QbC greedy AL) baseline. The baseline performances correspond to those obtained for $N = 1$. Actually, $BAAL$ can be viewed as an algorithm providing an “educated” sampling strategy, where the “educated sampler” is based on a computational budget of N simulations; the baseline, non-educated, sampler accordingly corresponds to $N = 1$.

6.3 Performance and Scalability

Fig. 2.(a) and (c) display the overall performance of stand-alone $BAAL$ and QbC-$BAAL$ versus the computational budget N (in log scale).

The competence of $BAAL$ in terms of Active Learning is visible as stand-alone $BAAL$ strongly outperforms the passive learning baseline ($N = 1$): the generalization error significantly decreases as N increases. For a given computational budget N, the improvement on passive learning is higher in small dimension, as could have been expected. Nevertheless, the improvement is significant both in dimensions 4 and 8 even for a small computational budget ($N > 2^6$).

The performance of stand-alone $BAAL$ is further assessed by comparison with that of a stand-alone QbC using a large-sized committee. After [4, 17], the Maximum Uncertainty heuristics (approximated by a large-sized QbC selection) is almost optimal (up to logarithmic terms) in the linear setting. It is thus satisfactory to see that stand-alone $BAAL$ steadily approaches the QbC reference performance (the fact that it can even outperform the QbC reference in dimension 8 is discussed below). These results suggest a positive answer to the *Optimality* question (section 6.1): stand-alone $BAAL$ is a competent, criterion-agnostic Active Learner, which might work well in the absence of prior knowledge about the instance selection, and which matches the known optimal performance in a simple hypothesis space.



(a) d = 4, stand-alone $BAAL$

(c) d = 4, QbC-$BAAL$

(c) d = 8, stand-alone $BAAL$

(d) d = 8, QbC-$BAAL$

Fig. 2. $BAAL$ performances: Generalization error vs the number of simulations N in log scale. Top rows report the results obtained for instance space dimension $d = 4$ and time horizon $T = 15$; bottom rows report the results for $d = 8, T = 20$. Left rows report the performance of the stand-alone $BAAL$. Right rows report the performance of QbC-$BAAL$. The performance of the baseline correspond to the performance obtained for $N = 1$ (at the origin of the curve). The horizontal line reports the results obtained for the stand-alone QbC strategy.

Another picture is provided by the performance of QbC-$BAAL$ (Fig. 2.(c) and (d)), suggesting that QbC-$BAAL$ can get the “best of both worlds”. In dimensions 4 and 8, QbC-$BAAL$ outperforms the QbC baseline in a statistically significant way; it overcomes the theoretical limitations of QbC with regards to optimality (i.e. multiplicative factors w.r.t. optimality logarithmic in the precision and linear in the dimension). These results thus suggest a positive answer to the *Flexibility* question (section 6.1): $BAAL$ can be hybridized with the QbC criterion, and this hybridization can improve on both stand-alone $BAAL$ and stand-alone QbC. Although $BAAL$ flexibility remains to be confirmed by considering other AL criteria, QbC is among the most widely used criteria in the AL literature.

Due to space limitations, the tractability of $BAAL$ is detailed in [31]. Overall, billiard algorithms allow for a time complexity lower by several orders of magnitude than the rejection algorithm, with an outstanding scalability w.r.t. horizon and dimension of the instance space. For $(d = 8, T = 20, N = 16,000)$, the computational time is 17 seconds on Pentium PC vs > 3 hours for the rejection-based *BAAL*.

7 Conclusion and Perspectives

This paper focuses on Active Learning with a limited number of queries, motivated by application domains such as Numerical Engineering. In such domains, computing the response of an example might require several days of computation, limiting the training set size to a few dozen or hundred examples.

Within this bounded resource setting, Active Learning was tackled as a Reinforcement Learning problem: find the sampling strategy aimed at minimizing the overall generalization error for a finite time horizon. This ideal and utterly intractable formalization leads to the proposed *BAAL* algorithm: an approximation of the optimal sampling strategy is learned within a one-player game setting. *BAAL* is inspired from an earlier work devoted to Computer Go [9], building upon the tree-structured bandit-based search algorithm UCT [37] and the progressive widening heuristics to deal with a continuous search space [13–15].

The experimental validation of *BAAL* investigates three main questions: the convergence towards the optimal performance when it is known; the ability to take advantage of competent AL criteria, such as Maximum Uncertainty [38], and improve on the greedy use of these criteria; the computational tractability of the overall AL scheme. A proof of principle, the experiments discussed in the paper show that the answer to all three questions is positive in the simple case of a realizable setting with linear hypotheses.

Further research aims at extending the proposed method beyond this simple setting. Getting rid of the realizable setting (considering label noise, or the case where the target concept does not belong to the hypothesis space $\mathcal{H}$) will be investigated through relaxing the billiard-based exploration of the Version Space. The billiard-based sampling of the hypotheses will likewise be extended to accommodate the priors on the hypothesis space (set to the uniform density in the present paper).

Another perspective for further study is to extend *BAAL* to non-linear hypothesis spaces, thanks to the famed kernel trick. It must be noted that billiard-based algorithms have been investigated for kernel spaces [11, 10, 35], featuring good theoretical and computational results. The extension of *BAAL* to Active Learning with kernels thus defines a new and appealing objective.

Acknowledgments

This work was supported in part by the Région Ile de France and Digiteo, and the PASCAL network of excellence. The authors acknowledge fruitful discussions with Jean-Marc Martinez, DM2S CEA, regarding applications of this work

to Numerical Engineering, and Benoit Kloeckner, from the Mathematics Department of the University of Grenoble, regarding the theoretical properties of billiards.

References

1. Kulkarni, S.R., Mitter, S.K., Tsitsiklis, J.N.: Active learning using arbitrary binary valued queries. Mach. Learn. **11**(1) (1993) 23–35
2. Cohn, D., Atlas, L., Ladner, R.: Improving generalization with active learning. Mach. Learn. **15**(2) (1994) 201–221
3. Schohn, G., Cohn, D.: Less is more: Active learning with support vector machines. Int. Conf. on Machine Learning **282** (2000) 285–286
4. Dasgupta, S.: Analysis of a greedy active learning strategy. In NIPS 17. MIT Press, (2005) 337–344
5. Castro, R., Willett, R., Nowak, R.: Faster rates in regression via active learning. In NIPS 18. MIT Press, (2006) 179–186
6. Hoi, S.C.H., Jin, R., Zhu, J., Lyu, M.R.: Batch mode active learning and its application to medical image classification. Int. Conf. on Machine Learning, ACM (2006) 417–424
7. Hanneke, S.: A bound on the label complexity of agnostic active learning. Int. Conf. on Machine Learning, ACM (2007) 353–360
8. Kocsis, L., Szepesvari, C.: Bandit-based monte-carlo planning. Eur. Conf. on Machine Learning, Springer Verlag (2006) 282–293
9. Gelly, S., Silver, D.: Combining online and offline knowledge in UCT. Int. Conf. on Machine Learning, ACM (2007) 273–280
10. Ruján, P.: Playing billiards in version space. Neural Computation **9**(1) (January 1997) 99–122
11. Herbrich, R., Graepel, T., Campbell, C.: Bayes point machines. Journal of Machine Learning Research **1** (2001) 245–279
12. Warmuth, M.K., Liao, J., Rätsch, G., Mathieson, M., Putta, S., Lemmen, C.: Support vector machines for active learning in the drug discovery process. Journal of Chemical Information Sciences **43** (2003) 667–673
13. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In P. Ciancarini and H. J. van den Herik, editors, Proc. of the 5th Int. Conf. on Computers and Games (2006)
14. Chaslot, G., Winands, M., Uiterwijk, J., van den Herik, H., Bouzy, B.: Progressive strategies for Monte-Carlo tree search. In Wang, P., et al., eds.: Proc. of the 10th Joint Conf. on Information Sciences, World Scientific Publishing (2007) 655–661
15. Wang, Y., Audibert, J.Y., Munos, R.: Algorithms for infinitely many-armed bandits. In NIPS 21. (2009) 1729–1736
16. Seung, H.S., Opper, M., Sompolinsky, H.: Query by committee. In: COLT '92, ACM (1992) 287–294
17. Freund, Y., Seung, H.S., Shamir, E., Tishby, N.: Selective sampling using the query by committee algorithm. Mach. Learn. **28**(2-3) (1997) 133–168
18. Cohn, D., Ghahramani, Z., Jordan, M.: Active Learning with Statistical Models. Journal of Artificial Intelligence Research **4** (1996) 129–145
19. Roy, N., McCallum, A.: Toward optimal active learning through sampling estimation of error reduction. Int. Conf. on Machine Learning, Morgan Kaufmann (2001) 441–448

20. Lindenbaum, M., Markovitch, S., Rusakov, D.: Selective sampling for nearest neighbor classifiers. Machine Learning **54** (2004) 125–152
21. Dasgupta, S., Kalai, A.T., Monteleoni, C.: Analysis of perceptron-based active learning. In COLT'05. (2005) 249–263
22. Cesa-Bianchi, N., Conconi, A., Gentile, C.: Learning probabilistic linear-threshold classifiers via selective sampling. In COLT'03, Springer (2003) 373–386
23. Florina Balcan, M., Broder, A., Zhang, T.: Margin based active learning. In COLT'07 (2007).
24. Xiao, G., Southey, F., Holte, R.C., Wilkinson, D.: Software testing by active learning for commercial games. In: AAAI-05. (2005) 609–616
25. Vidyasagar, M.: A Theory of Learning and Generalization, with Applications to Neural Networks and Control Systems. Springer-Verlag (1997)
26. Hegedüs, T.: Generalized teaching dimensions and the query complexity of learning. In COLT '95, ACM (1995) 108–117
27. Dasgupta, S.: Coarse sample complexity bounds for active learning. In NIPS 18. MIT Press (2006) 235–242
28. Haussler, D., Kearns, M., Schapire, R.E.: Bounds on the sample complexity of bayesian learning using information theory and the VC dimension. Mach. Learn. **14**(1) (1994) 83–113
29. Mackay, D.J.C.: Bayesian interpolation. Neural Computation **4** (1992) 415–447
30. Sutton, R., Barto, A.: Reinforcement learning: An introduction. MIT Press., Cambridge, MA (1998)
31. Rolet, P., Sebag, M., Teytaud, O.: Boosting active learning to optimality: some results on a tractable Monte-Carlo, billiard-based algorithm. Technical report, Laboratoire de Recherche en Informatique, Univ. Paris Sud (2009)
32. Bellman, R.: Dynamic Programming. Princeton Univ. Press (1957)
33. Auer, P.: Using confidence bounds for exploitation-exploration trade-offs. The Journal of Machine Learning Research **3** (2003) 397–422
34. Wang, Y., Gelly, S.: Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In: IEEE Symposium on Computational Intelligence and Games, Honolulu, Hawaii. (2007) 175–182
35. Ruján, P., Marchand, M.: Computing the bayes kernel classifier (1999)
36. Comets, F., Popov, S., Schütz, G.M., Vachkovskaia, M.: Billiards in a General Domain with Random Reflections. Archive for Rational Mechanics and Analysis **191** (March 2009) 497–537
37. Kocsis, L., Szepesvari, C.: Bandit Based Monte-Carlo Planning. Eur. Conf. on Machine Learning, Springer-Verlag (2006) 282-293
38. Freund, Y., Schapire R.: Large margin classification using the perceptron algorithm. In COLT'98, Morgan Kaufmann (1998)