HAL
archives-ouvertes.fr

Complexity Bounds for Batch Active Learning in Classification

Philippe Rolet, Olivier Teytaud

▶ To cite this version:

Philippe Rolet, Olivier Teytaud. Complexity Bounds for Batch Active Learning in Classification. ECML 2010, Oct 2010, Barcelone, Spain. 2010. <inria-00533318>

HAL Id: inria-00533318
https://hal.inria.fr/inria-00533318

Submitted on 5 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Complexity Bounds for Batch Active Learning in Classification

Philippe Rolet[1], Olivier Teytaud[1]

TAO team (Inria, LRI, UMR 8623 (CNRS - Université Paris-Sud)
bat. 490 Université Paris-Sud 91405 Orsay Cedex France,
{name}@lri.fr

Abstract. Active learning [1] is a branch of Machine Learning in which the learning algorithm, instead of being directly provided with pairs of problem instances and their solutions (their labels), is allowed to choose, from a set of unlabeled data, which instances to query. It is suited to settings where labeling instances is costly. This paper analyzes the speed-up of batch (parallel) active learning compared to sequential active learning (where instances are chosen 1 by 1): how faster can an algorithm become if it can query λ instances at once?
There are two main contributions: proving lower and upper bounds on the possible gain, and illustrating them by experimenting on usual active learning algorithms. Roughly speaking, the speed-up is asymptotically logarithmic in the batch size λ (i.e. when $\lambda \to \infty$). However, for some classes of functions with finite VC-dimension V, a linear speed-up can be achieved until a batch size of V. Practically speaking, this means that parallelizing computations on an expensive-to-label problem which is suited to active learning is very beneficial until V simultaneous queries, and less interesting (yet still bringing improvement) afterwards.

1 Introduction

Active learning [1] (AL) is a statistical Machine Learning setting in which data comes unlabeled, the learning algorithm chooses which data points (i.e. instances) are important, and queries an *oracle* to get their labels. Active learning can be particularly useful if the oracle labelling the examples is expensive.

Batch active learning [12, 23, 13] is the particular case of active learning in which the algorithm can choose λ examples at at at a time, meaning the oracle provides λ answers at once—λ is the *batch size*, that is the number of simultaneous requests to the oracle. This setting is for instance suited to cases where the oracle is a computational code (such as in numerical engineering applications): if λ computing units are available, the code can be run simultaneously on each machine.

This paper provides rigorous bounds on the number of iterations before a given precision is reached for batch active learning in binary classification, in particular as a function of λ. This model of complexity, based on the number of iterations only, is relevant for cases in which almost all the cost is in the calls to

the oracle function (*expensive* oracle), and when at least λ computation units are available. The internal cost of the learning algorithm is not taken into account.

The quantity of interest when analyzing batch active learning w.r.t. *sequential* active learning (i.e. when $\lambda = 1$), is the *speed-up* at λ: it is the ratio of the number of iterations (number of calls to the oracle) of a sequential algorithm and the number of iterations of an algorithm using batches of examples of size λ at each iteration.Obviously, under this assumption, *passive* learning—when instances to be queried are selected i.i.d. from the natural input distribution—has a linear speed-up: an algorithm querying λ instances at time in an i.i.d. fashion is λ times faster than the sequential algorithm querying 1 instance at a time. We here investigate to which extent such a good speed-up can be recovered for active learning. The paper is organized as follows:

- Section 2 presents the framework and notations, so that complexity bounds can be properly formalized;
- Section 3 shows bounds for batch active learning using covering and packing numbers. Results include lower and upper bounds on the speed-up of batch active learning, seen as a parallel algorithm;
- Section 4 presents some experiments; these experiments are aimed at comparing predicted speed-ups (for optimal algorithms) to speed-ups of simple or usual algorithms;
- Section 5 concludes.

State of the art

The query learning models introduced by [1] can be viewed as the first attempts of the learning algorithm to directly interact with the oracle. Another early work [15] establishes a lower bound for the instance size in any active learning (AL) classification setting, logarithmic in the ϵ-packing number of the hypothesis space $\mathcal{F}$—the number of disjoint ϵ-radius balls needed that can be put in $\mathcal{F}$(see section 2 below).

Classification. [4] devised a heuristic for error-free learning (i.e., in the realizable setting) of a binary classifier (i.e. a $\{0, 1\}$-valued target function to learn), considering a large pool of unlabeled examples and selecting the best example to be labeled in each time step (pool-based adaptive sampling). Considering the set of hypotheses compatible with the available examples—the *version space* (VS) defined in [18]—the selected examples were meant to prune the version space.

[10] analyzed another algorithm based on a Bayesian prior on the hypothesis space called *Query-by-committee* (QBC) from [22]; it directly reduces the VS volume. 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 [5, 14, 19, 17, 8], with sometimes encouraging results in, for instance,

pharmaceutical industry [26]. Specific algorithms and methods have been developed for active learning in linear and kernel spaces, either heuristically [21]or with theoretical foundations [3, 8, 2].

On the theoretical side, [10] related the efficiency of QBC to a statistical criterion called *Information Gain*, measuring how efficiently the VS can be divided. [6] has shown that with a Bayesian prior, greedily choosing examples that most evenly divide the VS is an almost optimal AL strategy.

Dasgupta also studied the non-Bayesian setting [7], deriving upper and lower complexity bounds based on a criterion called *splitting index*.

Batch active learning for classification. Batch active learning has received less attention. [12] assesses the information brought by batches of examples via a criterion based on Fisher information matrix reduction. [11] seeks sets of examples with low uncertainty; they phrase this as an optimization problem (NP-hard), and devise a method to find an acceptable approximation of the solution. Both works provide empirical evidence of the soundness of their strategies. However, they do not provide any formal proof guaranteeing their behavior. Further, we are not aware of any theoretical study of the speed-up of batch Active Learning over sequential Active Learning, in terms of sample complexity bounds (speed-up is in terms of gain with respect to the number of iterations, see section 2 below). Clearly, batch active learning can not reduce the overall number of evaluations when compared to sequential active learning; the advantage is only in the case of a parallel use of the oracle querying λ instances at a time.

2 Framework

In all the paper, log refers to the binary logarithm. If $a, b \in \mathbb{N}$, $[[a, b]] = \mathbb{N} \bigcap [a, b]$. If $x, y \in [0, \infty[^d$, then we note $[x, y] = \{a \in \mathbb{R}^d; \forall i, x_i \leq a_i \leq y_i\}$. The *speedup* of a parallel algorithm $\mathcal{A}_{\lambda}$ over its sequential counterpart $\mathcal{A}$ (or equivalently $\mathcal{A}_1$) is the ratio of $\mathcal{A}$'s complexity in terms of number of iterations (i.e. of batch calls to the oracle) on $\mathcal{A}_{\lambda}$'s complexity.

Only deterministic algorithms are considered here; the lower bounds can be extended, nonetheless, to stochastic cases within logarithmic dependencies on the risk δ (i.e. case with confidence $1 - \delta$)[1] and upper bounds (Theorem 2, referred to as a *simulation* result) can also be extended to the stochastic case. The framework of batch active learning is presented in Algorithm 1. A batch active learning algorithm $\mathcal{A}_{\lambda}$ is defined by the triplet $(learn_{\lambda}, generate_{\lambda}, update_{\lambda})$. Let D be a domain, P_D a probability measure on this domain, and $f^* : D \to \{0, 1\}$ be the unknown oracle, supposed to be deterministic and to belong to some set $\mathcal{F} \subset \{0, 1\}^D$. The *generalization error* $\mathbf{d}(f, f^*)$ of an approximation $f \in \mathcal{F}$ of f^* is defined as $P_D(\{x | f(x) \neq f^*(x)\})$. Note that $\mathbf{d}$ is a distance for the space $\mathcal{F}$[2].

[1] Precisely, the sample complexity is increased by a factor $-O(\log(\delta))$ if we request that the algorithm finds the solution with probability $1 - \delta$.

[2] More precisely, a pseudo-metric.

We assume that the considered concept class $\mathcal{F}$ has a finite VC-dimension V. VC-dimension is a classical measure of complexity for classes of functions, for which the reader is referred to [24, 9]. It is common to consider finite VC-dimension in active learning settings since the improvement over passive learning is potentially much bigger in this case [15]. Indeed, the number of examples required to learn f^* with precision ϵ, i.e. to find f such that $\mathbf{d}(f^*, f) \leq \epsilon$, is $N = \Theta(V \log(1/\epsilon))$ in good cases, see for instance [7, 10].

Algorithm 1 Batch active learning algorithm $\mathcal{A}_\lambda = (learn_\lambda, generate_\lambda, update_\lambda)$. λ is the number of visited points per iteration.

$I_0 = initial\ state$ // *Global state of the algorithm*
$n \leftarrow 0$
while true **do**
  $f_n \leftarrow learn_\lambda(I_n)$
  $(x_{n\lambda+1}, \ldots, x_{n\lambda+\lambda}) = generate_\lambda(I_n)$
  for $i \in [[1, \lambda]]$ **do**
    $y_{n\lambda+i} = f(x_{n\lambda+i})$ // *label λ instances at once*
  end for
  $I_{n+1} \leftarrow update_\lambda(I_n, x_{n\lambda+1}, \ldots, x_{(n+1)\lambda}, y_{n\lambda}, \ldots, y_{(n+1)\lambda})$
  $n \leftarrow n + 1$
end while

For a given λ and a given algorithm $\mathcal{A}$, the number of iterations for reaching precision ϵ is noted

$$N_\lambda^{\mathcal{A}}(\epsilon) = \sup_{f \in \mathcal{F}} \min\{n; ||f_n[\mathcal{A}_\lambda] - f||_1 \leq \epsilon\}$$

with $||.||_1$ the L_1 norm and $f_n[\mathcal{A}_\lambda]$ the approximation learned after n iterations via the framework presented in Algorithm 1. It will be convenient to note the best number of iterations achievable

$$L_\lambda(\epsilon) = \inf_{\mathcal{A}} N_\lambda^{\mathcal{A}}(\epsilon).$$

L_λ depends on the considered function class $\mathcal{F}$ (so does $N_\lambda^{\mathcal{A}}$), and will be noted $L_\lambda^{\mathcal{F}}$ when it is needed to make explicit which $\mathcal{F}$ is under study. Let $pack_{\mathcal{F}}(\epsilon)$ be the maximum number of points in $\mathcal{F}$ with pairwise distance $\mathbf{d}$ at least 2ϵ for the L_1 norm. In the sequel, for some of our results, the following equation will be assumed:

$$\forall \epsilon, pack_{\mathcal{F}}(\epsilon) \geq (M/\epsilon)^{(C \times V)} \tag{1}$$

for constants C and M. It states that the log-packing numbers of target function class $\mathcal{F}$ are at least $-CV \log(\epsilon/M)$. The product $C \times V$ in Eq. 1 stems from many results emphasizing some constant C, and the VC-dimension V [7], such as, for

example, the well-known case of homogeneous linear separators [3] of the sphere with homogeneous distribution[10, 8, 2], in which case V is equal to the dimension of the domain.

3 Covering Numbers and Batch Active Learning

Eq. 1 has the following consequence (see [16, 25]):

$$L_1(\epsilon) \geq \lceil CV \log(M/\epsilon) \rceil. \tag{2}$$

Eq. 2 states a lower bound on the sample complexity of AL, and has the following consequence (which is a lower bound on the sample complexity of batch AL):

$$L_\lambda(\epsilon) \geq \lceil CV \log(M/\epsilon)/\lambda \rceil \tag{3}$$

Eq. 3 is the ultimate limit for batch active learning: it is the case of a linear speed-up. The following explores the extent to which it can be reached, w.r.t. λ. In a parallel setting, in which λ calls to the oracle are performed in parallel, Eq. 3 refers to a linear speed-up for the parallel (i.e. batch) form of active learning.

The first contribution of this work is the following extension of the classical bound 2:

Theorem 1 (Lower bound for batch AL). *If $\mathcal{F}$ has packing number*

$$pack_{\mathcal{F}}(\epsilon) \geq (M/\epsilon)^{C \cdot V}, \tag{4}$$

then the following holds:

$$L_\lambda(\epsilon) \geq CV \log(M/\epsilon)/\log(K) \tag{5}$$

where $K = \lambda^V$ if $V \geq 3$, $\lambda^V + 1$ if $V \leq 2$, i.e.

$$L_\lambda(\epsilon) \geq C \log(M/\epsilon)/(\log(\lambda)). \tag{6}$$

Remark. K is an upper bound on the number of possible classifications of λ points, given a class of function with VC-dimension V. $K = \lambda^V$ stems from Sauer's lemma (see [20]). K is exponential in V, but finite, thanks to the finite number of possible classifications of λ points when the VC-dimension is V. Having this upper bound on the number of reachable states for one batch, the number of possible branches in a run of the algorithm can be bounded accordingly.

Proof: Consider an algorithm realizing $L_\lambda(\epsilon)$.

As the algorithm is deterministic, there is one and only one possible value for $x_1, \ldots, x_\lambda$ for this algorithm, independently of f:

$$(x_1, \ldots, x_\lambda) = generate(I_0).$$

[3] that is, linear separators whose value in the null vector is 0

Thanks to the finite VC-dimension and to Sauer's lemma, there are at most K possible values for $y_1, \ldots, y_\lambda$; therefore there are at most K possible values for I_1 (since the algorithm is assumed to be deterministic).

Similarly, for each possible value of I_1, there are at most K possible values for I_2; therefore the total number of possible values for I_2 is at most K^2.

By induction, there are at most K^i possible values for I_i. After $L_\lambda(\epsilon)$ iterations, each possible state $I_{L_\lambda(\epsilon)}$ corresponds to a function learned with $\lambda L_\lambda(\epsilon)$ examples. Since the algorithm realizes the bound $L_\lambda(\epsilon)$, for any 2 oracle functions distant of ϵ or more, the algorithm must have 2 different states. Thus, the final number of states is at least as big as the packing number: $K^{L_\lambda(\epsilon)} \geq pack_{\mathcal{F}}(\epsilon)$. As a consequence,

$$L_\lambda(\epsilon) \geq \log(pack_{\mathcal{F}}(\epsilon))/\log(K). \tag{7}$$

Eqs. 7 and 4 yield the expected result. □

The next result shows that this bound is tight, at least asymptotically ($\lambda \to \infty$).

Theorem 2 (Upper bound for batch AL). *In AL framework of Algo. 1), assume $\mathcal{F}$ has VC-dimension V. Define $K = \lambda^V + 1$ and*

$$\lambda' = \lambda \frac{K^D - 1}{K - 1}. \tag{8}$$

Then the following holds for all $D \geq 1$:

$$L_{\lambda'}(\epsilon) \leq \lceil L_\lambda(\epsilon)/D \rceil \tag{9}$$

Remark. Eq. 9 leads to

$$L_{\lambda'}(\epsilon) \leq \Omega\left(\lceil L_\lambda(\epsilon)/\log(\lambda') \rceil\right) \tag{10}$$

for fixed V and λ. This is a logarithmic speed-up: see for instance that if $\lambda = 1$, then $\forall D, L_{2^D} = \Omega(\frac{L_1(\epsilon)}{D})$

Proof: The proof exhibits an algorithm realizing Eq. 9. Consider an algorithm $\mathcal{A}_\lambda = (learn_\lambda, generate_\lambda, update_\lambda)$ realizing $L_\lambda(\epsilon)$ and consider some $D \geq 1$. Define $\lambda' = \lambda \frac{K^D - 1}{K - 1}$. Consider, then, another algorithm $\mathcal{A}_{\lambda'} = (learn_{\lambda'}, generate_{\lambda'}, update_{\lambda'})$ which generates λ' points by simulating $\mathcal{A}_\lambda$ on D steps; if $\mathcal{A}_\lambda$ has internal state I_n, then $\mathcal{A}_{\lambda'}$ has n^{th} internal state $I'_n = I_{Dn}$. At each iteration:

- $generate_{\lambda'}$ simulates the K^D possible internal paths

$$(I_{Dn}, I_{Dn+1}, \ldots, I_{Dn+D}.)_k \text{ with } k \in [[0, K^D]] \tag{11}$$

  and generates for each iteration all the λ' points visited in any of those paths. At state I_{Dn} λ points are to be labeled. Then, there are K possible states I_{Dn+1} resulting in Kλ other points. Repeating the process for $I_{Dn+2}, \ldots$, one can see that λ' as in Eq. 8 is enough;

– the target f is computed at these λ' points. It is then possible to figure out which path among the K^D possible paths is the one that would actually have happened during D steps of $\mathcal{A}_\lambda$
– $update_{\lambda'}$ is the result of $update_\lambda$ for the path selected in $generate_{\lambda'}$[4];
– the output of $learn_{\lambda'}$ is the output of $learn_\lambda$ on all points visited in the selected path[5]. □

Eqs. 6 and 9 show that $\log(\lambda)$ is the optimal speed-up when no assumption on λ are made: theorem 2 shows that in all cases, a logarithmic speed-up is achievable and theorem 1 shows that we cannot do much better for λ large.

The remaining question is what happens for moderate values of λ, and in particular how many simultaneous queries we need for removing the dependency in V. We now show that $\lambda = V$ leads to a nearly linear speed-up, for some families $\mathcal{F}$. This removes the dependency in V in runtimes—this means that the curse of dimensionality can be broken with a batch size of V, whereas $\lambda > V$ will only provide a logarithmic speed-up.

Let us remind that for binary classification, function classes of a domain X can equivalently be described as sets of subsets of X. In our case, the convention is that a set describes all the instances on which the function values are 1 (the complement of the set is thus where the function values are 0).

Theorem 3 (Linear speed-up until $\lambda = V$). *Consider $\mathcal{F}_V = \{[0, x], x \in [0, 1]^V\}$. Then, for some $M > 0, M' > 0$,*

$$\exists C > 0, \forall V, \exists \epsilon_0, \forall \epsilon < \epsilon_0, L_1^{\mathcal{F}_V}(\epsilon) \geq CV \log(M/\epsilon) \tag{12}$$

and

$$\exists C'; \forall V, \exists \epsilon_0, \forall \epsilon < \epsilon_0, L_V^{\mathcal{F}_V}(\epsilon) \leq C' \log(M'/\epsilon). \tag{13}$$

It is a classical result that $VC - dim(\mathcal{F}_V) = V$. Eqs. 12 and 13 state the linear speed-up for batch active learning with $\lambda = V$ for this family of functions (within the constants C and C').

Proof: We first show that the following holds:

$$\exists C > 0, \forall V, \exists \epsilon_0, \forall \epsilon < \epsilon_0, pack_{\mathcal{F}_V}(\epsilon) \geq M/\epsilon^{(C \times V)}. \tag{14}$$

Eq. 14 is a version of Eq. 1 modified for considering only ϵ small; it is weaker than Eq. 1 and sufficient for our purpose.

Eq. 14 is proved as follows:

– For x and y in $[\frac{1}{2}, 1]^V$, the L^1 distance between $[0, x]$ and $[0, y]$ is lower bounded by $\Theta(||x - y||_1)$.

[4] Therefore, only $D\lambda$ points are actually used among the λ' labeled points

[5] A big part of the points for which the target value has been computed is discarded. This is necessary for the formal proof of tightness of complexity bounds. For real-world applications, we guess that applying $learn_{\lambda'}$ on all points might be much better, within constant factors however.

– Therefore, a regular grid of edge $\Theta(\epsilon)$ can be constructed in $[0,1]^V$, yielding $\Theta(1/\epsilon^V)$ points for the sole $[1/2,1]^V$ part. Consequently, the packing number of $\{[0,x]; x \in [0,1]^V\}$ is $\Theta(1/\epsilon^V)$ and is therefore $\omega(1/\epsilon^{V/2})$.
– This shows Eq. 14 for $C = \frac{1}{2}$.

Then, Eq. 14 classically leads to Eq. 12 (this is analogous to the proof of Eq. 2 from Eq. 1, see section 2). This is the first part of the theorem (Eq. 12). Let us now show Eq. 13, by considering the following algorithm described at iteration n, with $\lambda = V$:

– $generate_\lambda$ prepares the batch $((x_{n\lambda+1}, \ldots, x_{(n+1)\lambda})$ as follows: for each $x_{n\lambda+i}$, all coordinates $j \neq i$ are set to 0. The i^{th} coordinate of $x_{n\lambda+i}$ is chosen by looking at the $n-1$ previous points in position i of each of the $n-1$ previous batches. It is defined as the middle of the segment defined by the lowest previously-observed i^{th} coordinate whose label is 0, and the highest previously observed i^{th} coordinate whose label is 1. More formally: [6]

$$(x_{n\lambda+i})_i = \frac{1}{2}\left(\min_{n' \leq n}\{(x'_{n'\lambda+i})_i | y'_{n'\lambda+i} = 0\} + \max_{n' \leq n}\{(x'_{n'\lambda+i})_i | y'_{n'\lambda+i} = 1\}\right). \quad (15)$$

– $learn_\lambda$ selects any function $f_n \in \mathcal{F}_V$ which is consistent with $x_1, \ldots, x_{n\lambda}$.

At a given iteration n each point $x_{n,i}$ of the batch of size λ makes sure that the domain will be halved along the i^{th} coordinate. Thus, after N iterations, it is known that the target oracle/classifier is in a square of edge size 2^{-N}. As a consequence, precision ϵ is reached in at most $\Theta(\log(1/\epsilon))$ iterations, which shows Eq. 13. □

This theorem shows that, at least for $\mathcal{F}$ as above, we can have a linear speed-up until $\lambda = V$; this is the tightness of Eq. 3 for $\lambda \leq V$—similarly to the tightness of Eq. 6 (*i.e.* logarithmic speed-up) shown by Eq. 10 for λ large.

4 Experiments

We have formally proved both lower and upper bounds on batch AL. The following shows that both a simple algorithm and a more sophisticated (yet usual) AL algorithm behave as predicted by the theorems of previous sections when adapted to the batch setting.

4.1 Experiments with Naive AL

We here experiment a simple batch AL algorithm for $\mathcal{F} = \{[0,x]; x \in [0,1]^d\}$ (VC-dimension $V = d$). The new sample(s) $x_{n\lambda+1}, \ldots, x_{(n+1)\lambda}$ are λ points

[6] In Eq. 15, if no point $x_{n'\lambda+i}$ has been labeled as 0, the minimum is set to 1; equivalently, if no point has been labeled as 1, the maximum is set to 0.

randomly drawn in v where

$$v = \{x \in [0,1]^d; \forall j \in [[1, n\lambda]] y_j = 0 \Rightarrow \neg(x_j \leq x)\}$$

$$\cap \{x \in [[0,1]^d; \forall j \in [[1, n\lambda]] y_j = 1 \Rightarrow x_j \leq x\}.$$

This means that we randomly sample the version space. Note that this parallel algorithm is straightforward to derive from its sequential counterpart that queries one random sample from the version space at each iteration. This is not true of all active learning algorithms: in many cases, it is not clear how to efficiently turn a sequential active learning into a parallel one.

The plot shows the inverse of the number of iterations for reaching precision $0.001d^2$, depending on λ; this means that what lies on the Y-axis are *rates* of convergence. Results are presented in Fig. 1. Each point is averaged over 33 runs.

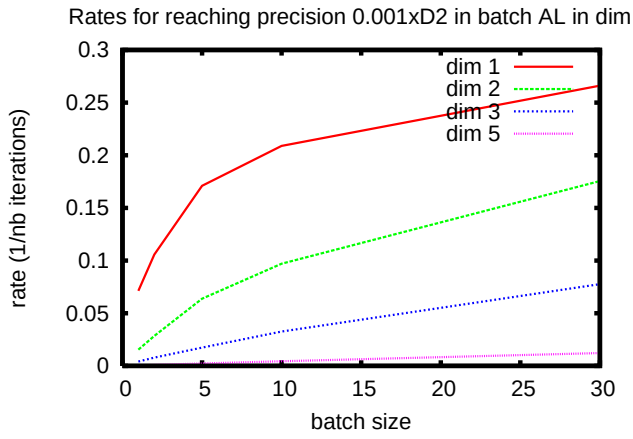

Fig. 1. Speed-up of batch AL for a simple AL algorithm (see text).

4.2 Experiments with Max-Uncertainty

This part of the experiments is concerned with a straightforward adaptation of a good, classical active learning heuristic that we call *Maximum Uncertainty* to the batch setting. The idea is to choose the most *uncertain* examples, meaning the ones for which many approximations of the target function that are still good candidates disagree on the label.

The possible approximations lie in the *version space*, the space of all possible functions of $\mathcal{F}$ consistent with the examples observed so far. Each example x splits the version space in two: the functions labeling x by 1, and those labeling x by 0. Thus, the goal is to find examples separating the version space the most

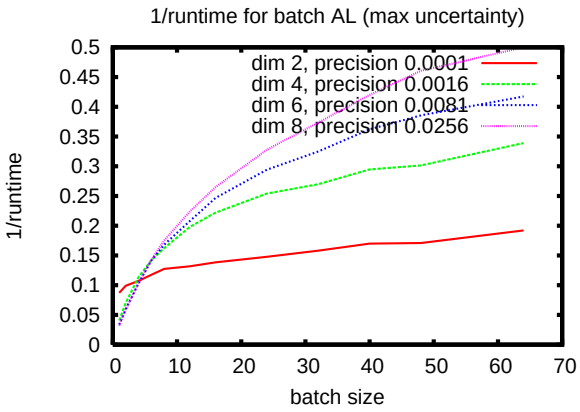

Fig. 2. Speed-up of batch maximum uncertainty active learning.

evenly. This criterion has been studied empirically and theoretically; multiple algorithms enforcing this criterion or related criteria have been proposed[22, 6].

Experiments learn homogeneous linear separators of $\mathbb{R}^d$, where examples lie on the hypersphere $\mathbb{S}^{d-1}$ for dimensions $d = 2, 4, 6, 8$. This setting has been widely studied for sequential active learning[2, 8, 10] and is thus fitted to a speed-up analysis for the batch setting.

In such a setting, for $d > 2$, an infinite number of points of the hypersphere maximize uncertainty given previously witnessed instances—whereas if $d = 2$, the maximum is unique. Consequently, a possible batch strategy may consist in selecting λ of those points maximizing uncertainty, at each iteration. Note that contrary to the algorithm of the preceding subsection, it is less obvious, at first sight, that this parallelization will be an efficient one (although a posteriori the results emphasize good speed-ups).

Batch sizes are $\lambda = 1, 2, 4, 6, 8, 12, 16, 20, 24, 32, 40, 48, 64$. Precision is set to $0.0001(d/2)^4$. For each (d, λ), the rate are averaged over 160 runs.

Interpretation

In both cases, the results resemble the expected behavior: a steady (linear) speed-up for small batch sizes, when $\lambda < d$ (where d is the dimension), and a slow speed-up when λ becomes much bigger than d, that somewhat resembles a logarithmic speed-up. Thus, as expected, the gain of parallelization in high dimension is bigger.

Remarks

- We did not study the behavior of the speed-up for values of λ in-between d and λ large; it might be that the speed-up can remain good even for a while when $\lambda > d$, but asymptotically it will end in a logarithmic improvement;

– The rates for high dimensions seem low (although linear) on the first figure, while they seem higher in the second. This is due to the fact that the precision to be reached is bigger in the second figure than in the first, in an attempt to be more "fair" to high dimensions—since for a given number of examples, a concept is harder to learn if the dimension of the domain is higher.

5 Discussion

This paper shows that batch active learning exhibits:

– a linear speed-up until $\lambda = V$ for some families of target functions;
– a speed-up at least logarithmic in all cases;
– and a logarithmic speed-up at most for λ large.

Please note that the logarithmic speed-up is a simulation result. The point is not to analyze the convergence rate of active learning in general, but to emphasize that *any* active learning algorithm can be transformed into a batch active learning algorithm (with λ computation units) which simulates it with speedup D, where D is logarithmic as a function of λ.

All proofs have been made for deterministic algorithms. Their extensions to stochastic cases, however, is straightforward.

Experiments have been performed only in moderate dimension and for easy families of functions; the extension of the experiments to bigger dimensions and to other families of functions, is a possible further work.

Acknowledgements. This work was supported by the French National Research Agency (ANR) grant No. ANR-08-COSI-007-12, and through COSINUS program (project EXPLO-RA n ANR-08-COSI-004).

References

1. D. Angluin. Queries and concept learning. *Mach. Learn.*, 2(4):319–342, 1988.
2. M.-F. Balcan, A. Broder, and T. Zhang. Margin based active learning. In *Proc. of the 20 th Conference on Learning Theory*, 2007.
3. N. Cesa-bianchi, A. Conconi, and C. Gentile. Learning probabilistic linear-threshold classifiers via selective sampling. In *In Proc. 16th COLT*, pages 373–386. Springer, 2003.
4. D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Mach. Learn.*, 15(2):201–221, 1994.
5. D. Cohn, Z. Ghahramani, and M. Jordan. Active Learning with Statistical Models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.
6. S. Dasgupta. Analysis of a greedy active learning strategy. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 337–344. MIT Press, Cambridge, MA, 2005.
7. S. Dasgupta. Coarse sample complexity bounds for active learning. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 235–242. MIT Press, Cambridge, MA, 2006.

8. S. Dasgupta, A. T. Kalai, and C. Monteleoni. Analysis of perceptron-based active learning. In *In COLT*, pages 249–263, 2005.
9. L. Devroye, L. Györfi, and G. Lugosi. *A probabilistic Theory of Pattern Recognition*. Springer, 1997.
10. Y. Freund, H. S. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Mach. Learn.*, 28(2-3):133–168, 1997.
11. Y. Guo and D. Schuurmans. Discriminative batch mode active learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 593–600, Cambridge, MA, 2008. MIT Press.
12. S. C. H. Hoi, R. Jin, J. Zhu, and M. R. Lyu. Batch mode active learning and its application to medical image classification. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 417–424, New York, NY, USA, 2006. ACM.
13. S. C. H. Hoi, R. Jin, J. Zhu, and M. R. Lyu. Semisupervised svm batch mode active learning with applications to image retrieval. *ACM Trans. Inf. Syst.*, 27(3):1–29, 2009.
14. V. S. Iyengar, C. Apte, and T. Zhang. Active learning using adaptive resampling. In *Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 91–98, 2000.
15. S. R. Kulkarni, S. K. Mitter, and J. N. Tsitsiklis. Active learning using arbitrary binary valued queries. *Mach. Learn.*, 11(1):23–35, 1993.
16. S. R. Kulkarni, S. K. Mitter, and J. N. Tsitsiklis. Active learning using arbitrary binary valued queries. *Mach. Learn.*, 11(1):23–35, 1993.
17. M. Lindenbaum, S. Markovitch, and D. Rusakov. Selective sampling for nearest neighbor classifiers. *Machine Learning*, 54:125–152, 2004.
18. T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.
19. N. Roy and A. McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proc. 18th International Conf. on Machine Learning*, pages 441–448. Morgan Kaufmann, San Francisco, CA, 2001.
20. N. Sauer. On the density of families of sets. *J. Comb. Theory, Ser. A*, 13(1):145–147, 1972.
21. G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. *Proceedings of the Seventeenth International Conference on Machine Learning*, 282:285–286, 2000.
22. H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, New York, NY, USA, 1992. ACM.
23. M. Sugiyama and N. Rubens. A batch ensemble approach to active learning with model selection. *Neural Netw.*, 21(9):1278–1286, 2008.
24. V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, N.Y., 1995.
25. M. Vidyasagar. *A Theory of Learning and Generalization*. Springer-Verlag, New York, New York, 1997.
26. M. K. Warmuth, J. Liao, G. Rätsch, M. Mathieson, S. Putta, and C. Lemmen. Support vector machines for active learning in the drug discovery process. *Journal of Chemical Information Sciences*, 43:667–673, 2003.