Fast Boosting Using Adversarial Bandits

Alexis Marion
Encadrante: Cecile Capponi

L'article

Róbert Busa-Fekete[1,2]
Balázs Kégl[1]

[1]LAL/LRI, Université Paris-Sud, CNRS
[2]Groupe de Recherche en Intelligence Artificielle de l'Académie des Sciences de Hongrie et de l'Université de Szeged

International Conference on Machine Learning, Haiva, Israel, 2010.

LAL: Laboratoire de l'Accélérateur Linéaire
LRI: Laboratoire de Recherche en Informatique

Fast boosting using adversarial bandits

Róbert Busa-Fekete[1,2] BUSAROBI@GMAIL.COM
Balázs Kégl[1] BALAZS.KEGL@GMAIL.COM
[1]LAL/LRI, University of Paris-Sud, CNRS, 91898 Orsay, France

[2]Research Group on Artificial Intelligence of the Hungarian Academy of Sciences and University of Szeged, Aradi vértanúk tere 1., H-6720 Szeged, Hungary

Keywords: AdaBoost, multi-armed bandits, large-scale learning

Abstract

In this paper we apply multi-armed bandits (MABs) to improve the computational complexity of AdaBoost. AdaBoost constructs a strong classifier in a stepwise fashion by selecting simple base classifiers and using their weighted "vote" to determine the final classification. We model this stepwise base classifier selection as a sequential decision problem, and optimize it with MABs where each arm represents a subset of the base classifier set. The MAB gradually learns the "usefulness" of the subsets, and selects one of the subsets in each iteration. AdaBoost then searches only this subset instead of optimizing the base classifier over the whole space. The main improvement of this paper over a previous approach is that we use an adversarial bandit algorithm instead of stochastic bandits. This choice allows us to prove a weak-to-strong-learning theorem, which means that the proposed technique remains a boosting algorithm in a formal sense. We demonstrate on benchmark datasets that our technique can achieve a generalization performance similar to standard AdaBoost for a computational cost that is an order of magnitude smaller.

1. Introduction

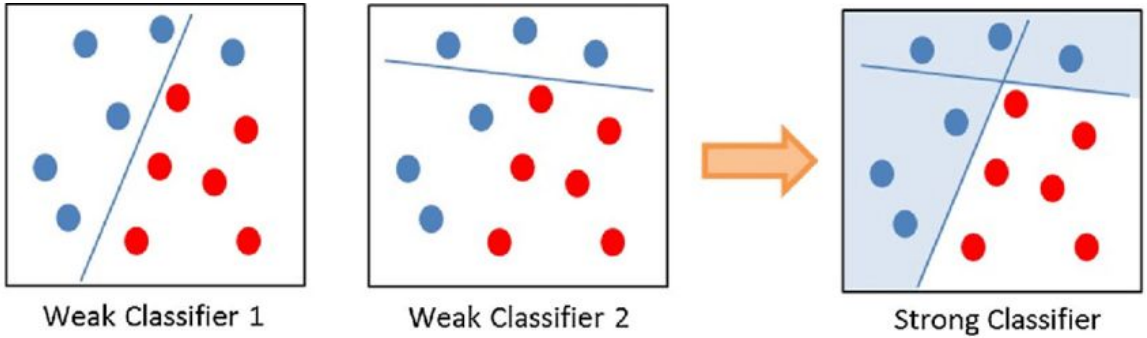
AdaBoost (Freund & Schapire, 1997) is one of the best off-the-shelf learning methods developed in the last decade. It constructs a classifier in a stepwise fashion by adding simple classifiers (called *base classifiers*) to a pool, and using their weighted "vote" to determine the final classification. The simplest base learner used in practice is the *decision stump*, a one-decision two-leaf decision tree. Learning a decision stump means selecting a feature and a threshold, so the running time of AdaBoost with stumps is proportional to the number of data points n, the number of attributes d, and the number of boosting iterations T. When *trees* (Quinlan, 1993) or *products* (Kégl & Busa-Fekete, 2009) are constructed over the set of stumps, the computational cost is multiplied by an additional factor of the number of tree levels $\log N$ (where N is the number of leaves) or the number of terms m. Although the running time is linear in each of these factors, the algorithm can be prohibitively slow if the data size n and/or the number of features d is large.

There are essentially two ways to accelerate AdaBoost in this setting: one can either limit the number of data points n used to train the base learners, or one can cut the search space by using only a subset of the d features. Although both approaches increase the number of iterations T needed for convergence, the net computational time can still be significantly decreased. The former approach has a basic version when the base learner is not trained on the whole weighted sample, rather on a small subset selected randomly using the weights as a discrete probability distribution (Freund & Schapire, 1997). A recently proposed algorithm of this kind is FilterBoost (Bradley & Schapire, 2008), which assumes that an oracle can produce an unlimited number of labeled examples, one at a time. In each boosting iteration, the oracle generates sample points that the base learner can either accept or reject, and then the base learner is trained on a small set of accepted points. The latter approach was proposed by (Escudero et al., 2000) which introduces several feature selection and ranking methods used

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Haifa, Israel, 2010.

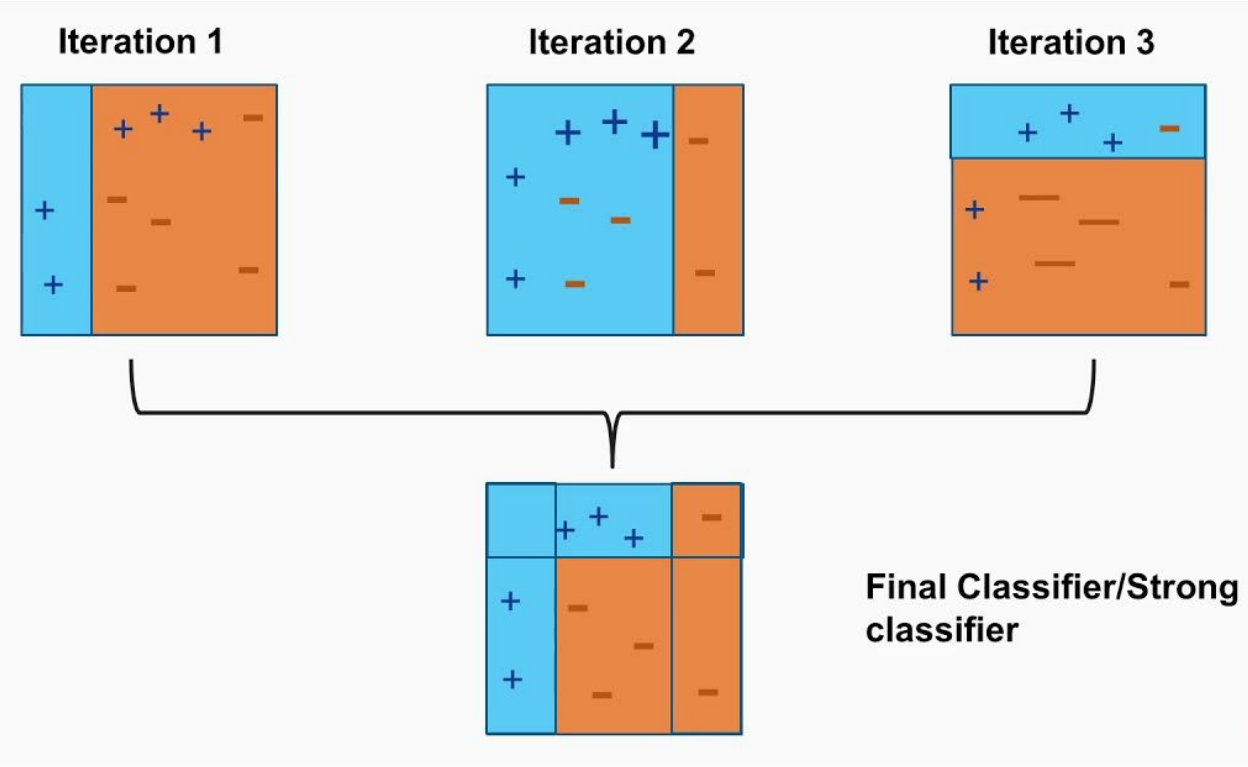
AdaBoost

Le Boosting



AdaBoost

Le Boosting Pondéré



AdaBoost

$\textsc{AdaBoost.MH}(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(1)}, \textsc{Base}(\cdot,\cdot,\cdot), T)$

1 **for** $t \leftarrow 1$ **to** T

2 $\quad \mathbf{h}^{(t)}(\cdot) \leftarrow \alpha^{(t)}\mathbf{v}^{(t)}\varphi^{(t)}(\cdot) \leftarrow \textsc{Base}\left(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)}\right)$

3 $\quad$ **for** $i \leftarrow 1$ **to** n **for** $\ell \leftarrow 1$ **to** K

4 $$w_{i,\ell}^{(t+1)} \leftarrow w_{i,\ell}^{(t)} \frac{\exp\left(-h_\ell^{(t)}(\mathbf{x}_i) y_{i,\ell}\right)}{\sum_{i'=1}^{n}\sum_{\ell'=1}^{K} w_{i',\ell'}^{(t)} e^{-h_{\ell'}^{(t)}(\mathbf{x}_{i'}) y_{i',\ell'}}}$$

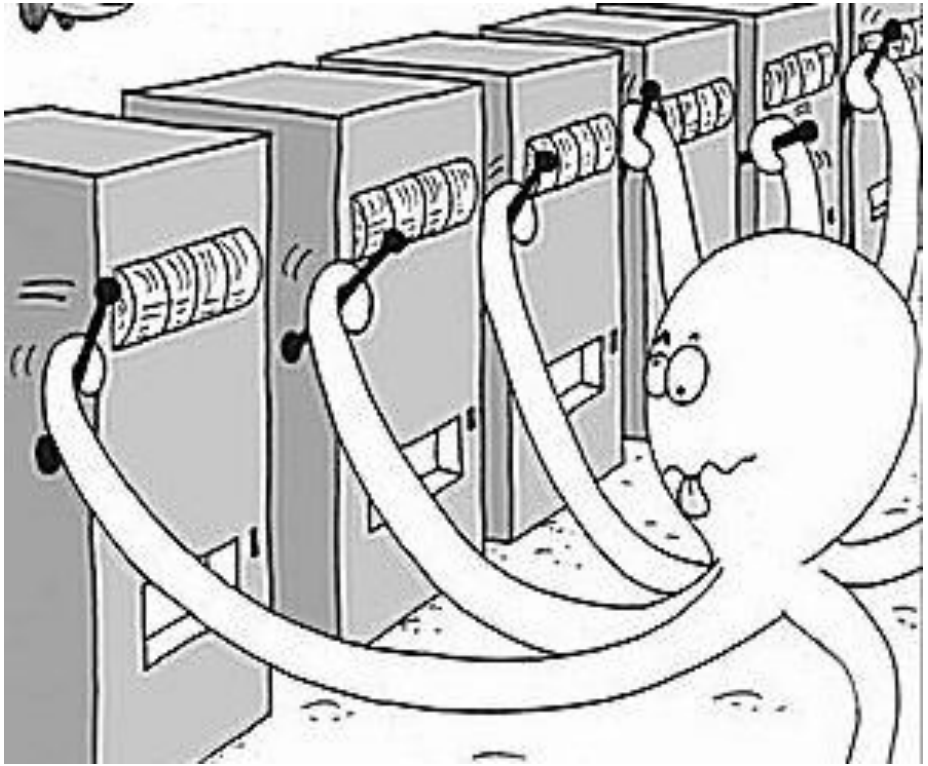
5 **return** $\mathbf{f}^{(T)}(\cdot) = \sum_{t=1}^{T} \mathbf{h}^{(t)}(\cdot)$

Entrainement du modèle de base

Mise à jour des poids des exemples

Construction du classifieur final

Bandit Multi-bras



Bandit Multi-bras

EXP3.P(η, λ, T)

1 **for** $j \leftarrow 1$ **to** M ▷ *initialization*

2 $\quad \omega_j^{(1)} \leftarrow \exp\left(\frac{\eta\lambda}{3}\sqrt{\frac{T}{M}}\right)$

3 **for** $t \leftarrow 1$ **to** T

4 $\quad$ **for** $j \leftarrow 1$ **to** M

5 $\quad\quad p_j^{(t)} \leftarrow (1-\lambda)\frac{\omega_j^{(t)}}{\sum_{j'=1}^{M}\omega_{j'}^{(t)}} + \frac{\lambda}{M}$

6 $\quad j^{(t)} \leftarrow \text{RANDOM}\left(p_1^{(t)}, \ldots, p_M^{(t)}\right)$

7 $\quad$ Receive reward $r_{j^{(t)}}^{(t)}$

8 $\quad$ **for** $j \leftarrow 1$ **to** M

9 $\quad\quad \widehat{r}_j^{(t)} \leftarrow \begin{cases} r_j^{(t)}/p_j^{(t)} & \text{if } j = j^{(t)} \\ 0 & \text{otherwise} \end{cases}$

10 $\quad\quad \omega_j^{(t+1)} \leftarrow \omega_j^{(t)} \exp\left(\frac{\lambda}{3M}\left(\widehat{r}_j^{(t)} + \frac{\eta}{p_j^{(t)}\sqrt{MT}}\right)\right)$

Initialisation des poids des bras

Mise à jour de la distribution de probabilité du choix du bras en fonction des poids des bras et des paramètres du bandit

Choix du bras relativement à la distribution de probabilité

Entrainement du modèle et réception de la reward

Mise à jours des poids des bras

Bandit Multi-bras

La reward

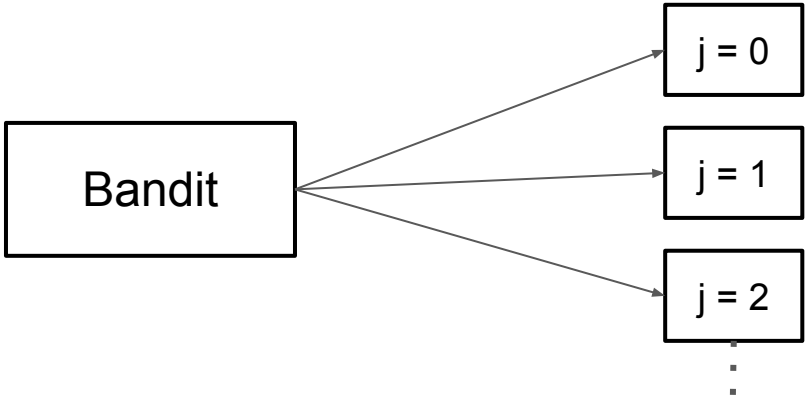
$$r_j^{(t)} = \min\left(1, -\log\sqrt{1-\gamma_{\mathcal{H}_j}^{(t)\,2}}\right)$$

$$\gamma = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell} v_\ell \varphi(\mathbf{x}_i) y_{i,\ell}$$

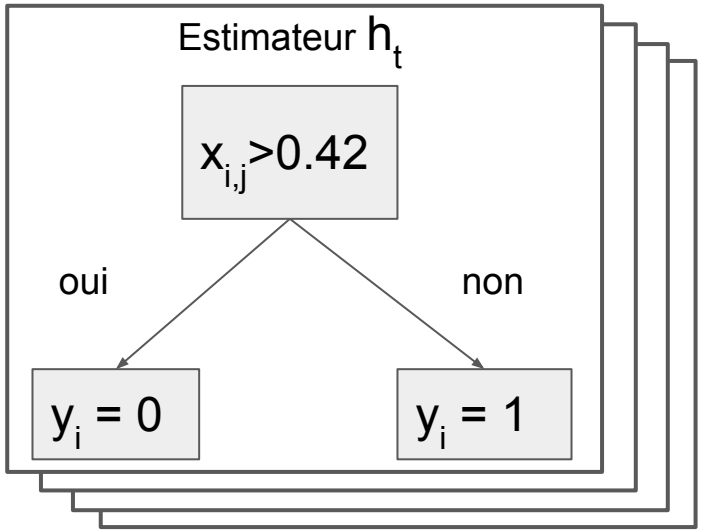
Intégration du bandit dans AdaBoost

L'espace des estimateurs est partitionné en sous-ensembles. Chaque sous-ensemble correspond aux estimateurs entrainés à l'aide d'une seule feature.

Le rôle du bandit est de sélectionner la feature qui servira au prochain estimateur.



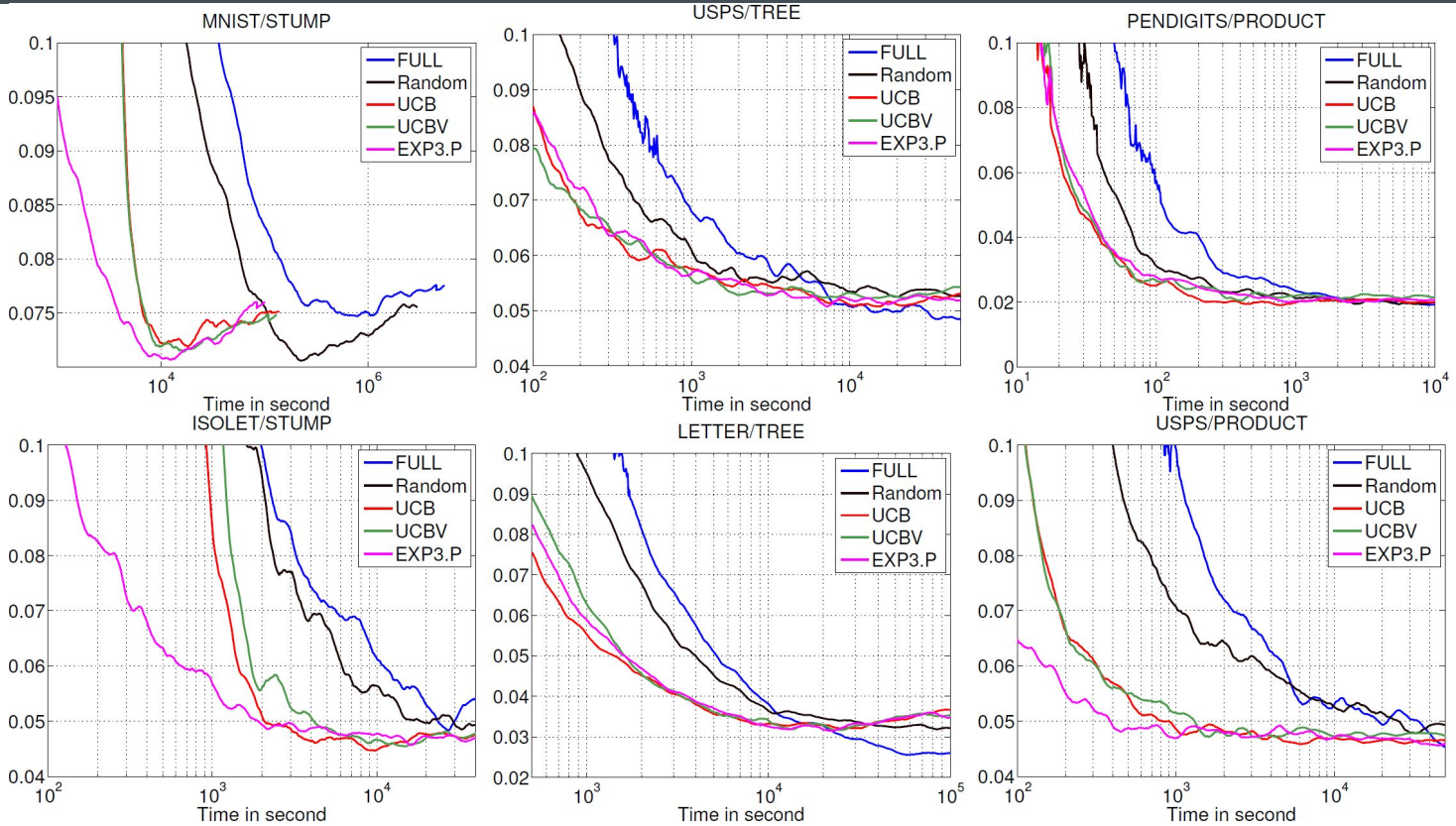
Le bandit choisit la feature sur laquelle sera entraîné le prochain estimateur (un arbre de décision de profondeur 1)



Résultats

Performances (Article)

FULL / Random / UCB / UCBV / EXP3.P



Implémentation

Algorithme SAMME

sklearn.ensemble.AdaBoostClassifier

class sklearn.ensemble. **AdaBoostClassifier** (*base_estimator=None*, *n_estimators=50*, *learning_rate=1.0*, *algorithm='SAMME.R'*, *random_state=None*) [source]

An AdaBoost classifier.

An AdaBoost [1] classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

This class implements the algorithm known as AdaBoost-SAMME [2].

Read more in the User Guide.

Parameters:

base_estimator : *object, optional (default=None)*

The base estimator from which the boosted ensemble is built. Support for sample weighting is required, as well as proper `classes_` and `n_classes_` attributes. If `None`, then the base estimator is `DecisionTreeClassifier(max_depth=1)`

n_estimators : *integer, optional (default=50)*

The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.

learning_rate : *float, optional (default=1.)*

Learning rate shrinks the contribution of each classifier by `learning_rate`. There is a trade-off between `learning_rate` and `n_estimators`.

algorithm : *{'SAMME', 'SAMME.R'}, optional (default='SAMME.R')*

If 'SAMME.R' then use the SAMME.R real boosting algorithm. `base_estimator` must support calculation of class probabilities. If 'SAMME' then use the SAMME discrete boosting algorithm. The SAMME.R algorithm typically converges faster than SAMME, achieving a lower test error

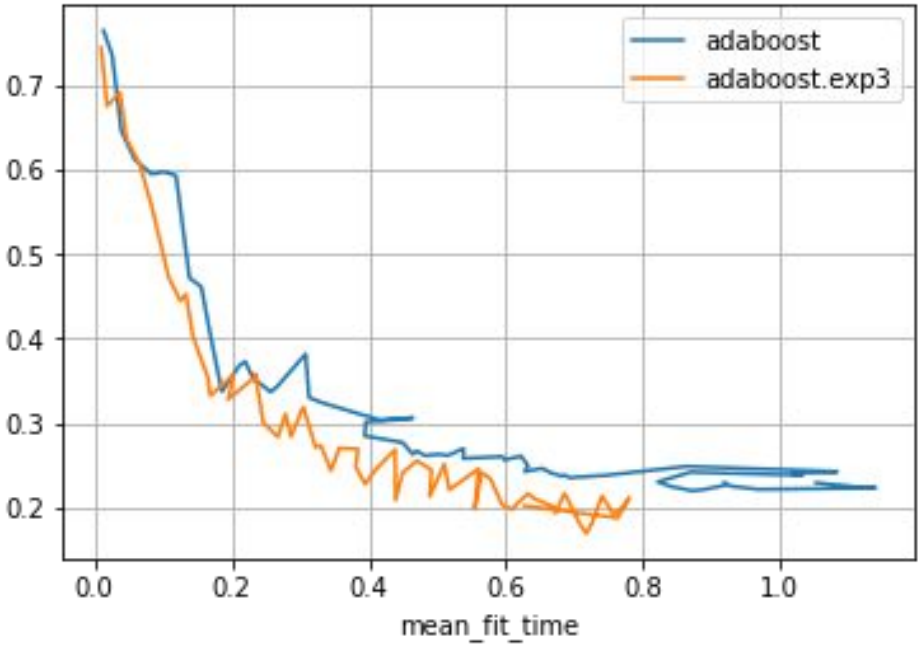
Résultats

Performances (Implémentation Sklearn)

FULL / EXP3.P

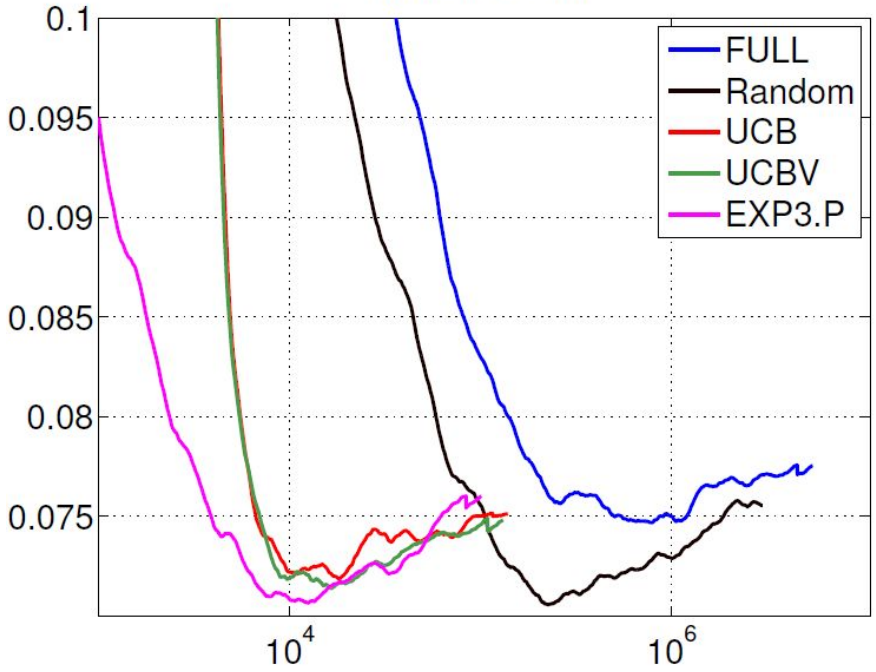
Erreur

digits



Erreur

MNIST/STUMP



Conclusion

Bilan de l’algorithme

- Améliore la complexité de calcul
- Généralisation au moins aussi performante
- L’erreur de test asymptotique est difficile à battre

La suite

- Exploiter l’état d’AdaBoost (les poids)
- Utiliser des bandits multi-bras capables d’exploiter des espaces de features structurés