JMLR: Workshop and Conference Proceedings 7: 111-122 KDD cup 2009

Accelerating AdaBoost using UCB

Róbert Busa-Fekete BUSAROBI@GMAIL.COM
LAL, University of Paris-Sud, CNRS
Orsay, 91898, France
Research Group on Artificial Intelligence of the
Hungarian Academy of Sciences and University of Szeged
Aradi vértanúk tere 1., H-6720 Szeged, Hungary

Balázs Kégl BALAZS.KEGL@GMAIL.COM
LAL/LRI, University of Paris-Sud, CNRS
Orsay, 91898, France

Editor: Gideon Dror, Marc Boullé, Isabelle Guyon, Vincent Lemaire, David Vogel

Abstract

This paper explores how multi-armed bandits (MABs) can be applied to accelerate AdaBoost. AdaBoost constructs a strong classifier in a stepwise fashion by adding simple base classifiers to a pool 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. Each arm represents a subset of the base classifier set. The MAB gradually learns the "utility" 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 reward is defined as a function of the accuracy of the base classifier. We investigate how the well-known UCB algorithm can be applied in the case of boosted stumps, trees, and products of base classifiers. The *KDD Cup 2009* was a large-scale learning task with a limited training time, thus this challenge offered us a good opportunity to test the utility of our approach. During the challenge our best results came in the *Up-selling* task where our model was within 1% of the best AUC rate. After more thorough post-challenge validation the algorithm performed as well as the best challenge submission on the small data set in two of the three tasks.

Keywords: AdaBoost, Multi-Armed Bandit Problem, Upper Confidence Bound

1. Introduction

ADABOOST (Freund and 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 to select 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 and 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 N 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 and Schapire, 1997). A recently proposed algorithm of the same kind is FILTERBOOST (Bradley and 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 to accelerate ADABOOST. In particular, the LAZYBOOST algorithm chooses a fixed-size random subset of the features in each boosting iteration, and trains the base learner using only this subset. This technique was successfully applied to face recognition where the number of features can be extremely large (Viola and Jones, 2004).

In this paper we aim to improve the latter approach by "aiding" the random feature selection. It is intuitively clear that certain features are more important than others for classification. In specific applications the utility of features can be assessed a-priori (e.g., on images of characters, we know that background pixels close to the image borders are less informative than pixels in the middle of the images), however, our aim here is to *learn* the importance of features by evaluating their empirical performance during the boosting iterations. Our proposed method is similar in spirit to the feature extraction technique described recently by (Borisov et al., 2006; Tuv et al., 2009). The objective of their method is to use tree-based ensembles for feature selection whereas our goal is more restrictive: we simply want to accelerate ADABOOST. To avoid harming the generalization ability of ADABOOST it is important to keep a high level of base learner diversity, which is the reason why we opted for using *multi-armed bandits* (MAB) that are known to manage the exploration-exploitation trade-off very well.

MAB techniques have recently gained great visibility due to their successful applications in real life, for example, in the game of GO (Gelly and Silver, 2008). In the classical bandit problem the decision maker can select an arm at each discrete time step (Auer et al., 2002b). Selecting an arm results in a random reward, and the goal of the decision maker is to maximize the expected sum of the rewards received. Our basic idea is to partition the base classifier space into subsets and use MABs to learn the utility of the subsets. In each iteration, the bandit algorithm selects an optimal subset, then the base learner finds the best base classifier in the subset and returns a reward based on the accuracy of this optimal base classifier. By reducing the search space of the base learner, we can expect a significant decrease of the complete running time of ADABOOST. We use the UCB algorithm (Auer et al., 2002a) by assigning each feature to a subset. In the case of trees and products we use UCB by considering each tree or product as a sequence of decisions, and using the same partitioning as with decision stumps at each inner node.

The paper is organized as follows. First we describe the ADABOOST.MH algorithm and the necessary notations in Section 2. Section 3 contains our main contribution of using MABs for accelerating the selection of base classifiers. In Section 4 we present experiments conducted during the development period of the competition, our competition results, and some post-challenge analysis. Closing discussions are in Section 5.

2. ADABOOST.MH

For the formal description let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ be the $n \times d$ *observation matrix*, where $x_i^{(j)}$ are the elements of the d-dimensional observation vectors $\mathbf{x}_i \in \mathbb{R}^d$. We are also given a *label matrix* $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$ of dimension $n \times K$ where $\mathbf{y}_i \in \{+1, -1\}^K$. In *multi-class* classification one and only one of the elements of $\mathbf{y}_i$ is $+1$, whereas in *multi-label* (or *multi-task*) classification $\mathbf{y}_i$ is arbitrary, meaning that the observation $\mathbf{x}_i$ can belong to several classes at the same time. In the former case we will denote the index of the correct class by $\ell(\mathbf{x}_i)$.

ADABOOST.MH$(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(1)}, \text{BASE}(\cdot,\cdot,\cdot), T)$

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

2 $\quad \mathbf{h}^{(t)}(\cdot) \leftarrow \alpha^{(t)}\mathbf{v}^{(t)}\varphi^{(t)}(\cdot) \leftarrow \text{BASE}\big(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)}\big)$

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

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

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

Figure 1: The pseudocode of the ADABOOST.MH algorithm. $\mathbf{X}$ is the observation matrix, $\mathbf{Y}$ is the label matrix, $\mathbf{W}^{(1)}$ is the initial weight matrix, $\text{BASE}(\cdot,\cdot,\cdot)$ is the base learner algorithm, and T is the number of iterations. $\alpha^{(t)}$ is the base coefficient, $\mathbf{v}^{(t)}$ is the vote vector, $\varphi^{(t)}(\cdot)$ is the scalar base classifier, $\mathbf{h}^{(t)}(\cdot)$ is the vector-valued base classifier, and $\mathbf{f}^{(T)}(\cdot)$ is the final (strong) classifier.

The goal of the ADABOOST.MH algorithm ((Schapire and Singer, 1999), Figure 1) is to return a vector-valued classifier $\mathbf{f} : \mathcal{X} \to \mathbb{R}^K$ with a small *Hamming loss*

$$R_H\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big) = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(1)} \mathbb{I}\left\{\operatorname{sign}\big(f_\ell^{(T)}(\mathbf{x}_i)\big) \neq y_{i,\ell}\right\}^1$$

by minimizing its upper bound (the exponential margin loss)

$$R_e\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big) = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(1)} \exp\big(-f_\ell^{(T)}(\mathbf{x}_i) y_{i,\ell}\big), \tag{1}$$

where $f_\ell(\mathbf{x}_i)$ is the ℓth element of $\mathbf{f}(\mathbf{x}_i)$. The user-defined weights $\mathbf{W}^{(1)} = \big[w_{i,\ell}^{(1)}\big]$ are usually set either uniformly to $w_{i,\ell}^{(1)} = 1/(nK)$, or, in the case of multi-class classification, to

$$w_{i,\ell}^{(1)} = \begin{cases} \frac{1}{2n} & \text{if } \ell = \ell(\mathbf{x}_i) \text{ (i.e., if } y_{i,\ell} = 1), \\ \frac{1}{2n(K-1)} & \text{otherwise (i.e., if } y_{i,\ell} = -1) \end{cases} \tag{2}$$

1. The indicator function $\mathbb{I}\{A\}$ is 1 if its argument A is true and 0 otherwise.

to create K well-balanced one-against-all classification problems. ADABOOST.MH builds the final classifier $\mathbf{f}$ as a sum of *base classifiers* $\mathbf{h}^{(t)} : \mathcal{X} \to \mathbb{R}^K$ returned by a *base learner* algorithm $\text{BASE}(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)})$ in each iteration t. In general, the base learner should seek to minimize the *base objective*

$$E\big(\mathbf{h}, \mathbf{W}^{(t)}\big) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell}^{(t)} \exp\big(-h_\ell(\mathbf{x}_i) y_{i,\ell}\big). \tag{3}$$

Using the weight update formula of line 4 (Figure 1), it can be shown that

$$R_e\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big) = \prod_{t=1}^{T} E\big(\mathbf{h}^{(t)}, \mathbf{W}^{(t)}\big), \tag{4}$$

so minimizing (3) in each iteration is equivalent to minimizing (1) in an iterative greedy fashion. By obtaining the multi-class prediction

$$\widehat{\ell}(\mathbf{x}) = \arg\max_{\ell} f_\ell^{(T)}(\mathbf{x}),$$

it can also be proven that the "traditional" multi-class loss (or *one-error*)

$$R\big(\mathbf{f}^{(T)}\big) = \sum_{i=1}^{n} \mathbb{I}\left\{\ell(\mathbf{x}_i) \neq \widehat{\ell}(\mathbf{x}_i)\right\} \tag{5}$$

has an upper bound $K R_e\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big)$ if the weights are initialized uniformly, and $\sqrt{K-1} R_e\big(\mathbf{f}^{(T)}, \mathbf{W}^{(1)}\big)$ with the multi-class initialization (2). This justifies the minimization of (1).

2.1 Learning the base classifier

In this paper we use *discrete* ADABOOST.MH in which the vector-valued base classifier $\mathbf{h}(\mathbf{x})$ is represented as

$$\mathbf{h}(\mathbf{x}) = \alpha \mathbf{v} \varphi(\mathbf{x}),$$

where $\alpha \in \mathbb{R}^+$ is the *base coefficient*, $\mathbf{v} \in \{+1, -1\}^K$ is the *vote vector*, and $\varphi(\mathbf{x}) : \mathbb{R}^d \to \{+1, -1\}$ is a *scalar* base classifier. It can be shown that for minimizing (3), one has to choose φ that maximizes the *edge*

$$\gamma = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} v_\ell \varphi(\mathbf{x}_i) y_{i,\ell}, \tag{6}$$

using the votes

$$v_\ell = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_{i,\ell} \mathbb{I}\{\varphi(\mathbf{x}_i) = y_{i,\ell}\} > \sum_{i=1}^{n} w_{i,\ell} \mathbb{I}\{\varphi(\mathbf{x}_i) \neq y_{i,\ell}\}, \\ -1 & \text{otherwise,} \end{cases} \quad \ell = 1, \ldots, K. \tag{7}$$

The optimal coefficient is then

$$\alpha = \frac{1}{2} \ln \frac{1+\gamma}{1-\gamma}.$$

It is also well known that the base objective (3) can be expressed as

$$E\big(\mathbf{h}, \mathbf{W}\big) = \sqrt{(1+\gamma)(1-\gamma)} = \sqrt{1-\gamma^2}. \tag{8}$$

The simplest scalar base learner used in practice is the *decision stump*, a one-decision two-leaf decision tree of the form

$$\varphi_{j,b}(\mathbf{x}) = \begin{cases} 1 & \text{if } x^{(j)} \geq b, \\ -1 & \text{otherwise,} \end{cases}$$

where j is the index of the selected feature and b is the decision threshold. If the features are pre-ordered before the first boosting iteration, a decision stump maximizing the edge (6) can be found very efficiently in $\Theta(ndK)$ time (making the total running time $\Theta\big(nd(\log n + KT)\big)$).

Although boosting decision stumps often yields satisfactory results, state-of-the-art performance of ADABOOST is usually achieved by using decision trees as base learners. In this paper we use an "in-house" implementation that calls the decision stump optimizer as a subroutine. The learner is similar to Quinlan's C4.5 algorithm (Quinlan, 1993), except that we use the edge improvement (instead of Quinlan's entropy-based criterion) to select the next node to split and the threshold b. The base learner has one hyperparameter, the number of the leaves N, which also shows up as a linear factor in the running time.

We also test our approach using a recently proposed base learner that seems to outperform boosted trees (Kégl and Busa-Fekete, 2009). The goal of this learner is to optimize products of simple base learners of the form

$$\mathbf{h}(\cdot) = \alpha \prod_{j=1}^{m} \mathbf{v}_j \varphi_j(\cdot), \tag{9}$$

where the vote vectors $\mathbf{v}_j$ are multiplied element-wise. The learner also calls the decision stump optimizer as a subroutine but in an iterative rather than a recursive fashion. The hyperparameter m, again, appears as a linear factor in the total running time.

3. Using multi-armed bandits to reduce the search space

In this section we will first describe the MAB framework and next we show how bandit algorithm UCB can be used to accelerate the base learning step in ADABOOST.

3.1 Multi-armed bandits

In the classical bandit problem there are M arms that the decision maker can select at discrete time steps. Selecting arm j in iteration t results in a random reward $r_j^{(t)} \in [0, 1]$ whose (unknown) distribution depends on j. The goal of the decision maker is to maximize the expected sum of the rewards received. Intuitively, the decision maker's policy has to balance between using arms with large past rewards (exploitation) and trying arms that have not been tested enough times (exploration). The UCB algorithm (Auer et al., 2002a) manages this trade-off by choosing the arm that maximizes the sum of the average reward

$$\overline{r}_j^{(t)} = \frac{1}{T_j^{(t)}} \sum_{t'=1}^{t} \mathbb{I}\left\{\text{arm } j \text{ is selected}\right\} r_j^{(t')}$$

and a confidence interval term

$$c_j^{(t)} = \sqrt{\frac{2 \ln t}{T_j^{(t)}}},$$

where $T_j^{(t)}$ is the number of times when arm j has been selected up to iteration t. To avoid the singularity at $T_j^{(t)} = 0$, the algorithm starts by selecting each arm once. We use a generalized version, denoted by UCB(k), in which the best k arms are selected for evaluation, and the one that maximizes the actual reward $r_j^{(t)}$ is finally chosen.

3.2 The application of UCB(k) for accelerating ADABOOST

The general idea is to partition the base classifier space into (not necessarily disjunct) subsets and use MABs to learn the utility of the subsets. In each iteration, the bandit algorithm selects an optimal subset (or, in the case of UCB(k), a union of subsets). The base learner then finds the best base classifier in the subset, and returns a reward based on this optimal base learner. By reducing the search space of the base learner, we can expect a significant decrease of the complete running time of ADABOOST.

The upper bound (4) together with (8) suggest the use of $-\frac{1}{2}\log(1-\gamma^2)$ for the reward. In practice we found that

$$r_j^{(t)} = 1 - \sqrt{1-\gamma^2}$$

works as well as the logarithmic reward; it was not surprising since the two are almost identical in the lower range of the $[0,1]$ interval where the majority of the edges are. The latter choice has another advantage of always being in the $[0,1]$ interval which is a formal requirement in MABs.

The actual partitioning of the base classifier set depends on the particular base learner. In the case of decision stumps, the most natural choice for UCB is to assign each feature to a subset, i.e., *jth* subset is $\{\varphi_{j,b}(\mathbf{x}) : b \in \mathbb{R}\}$. In principle, we could also further partition the threshold space but that would not lead to further savings in the linear computational time since, because of the changing weights $w_{i,\ell}$, all data points and labels would have to be visited anyway. On the other hand, subsets that contain more than one feature can be efficiently handled by UCB(k).

In the case of trees and products we use UCB by considering each tree or product as a sequence of decisions, and using the same partitioning as with decision stumps at each inner node. In this setup we lose the information in the dependence of the decisions on each other *within* a tree or a product.

4. Experiments

4.1 Data set description and data preparation

In *KDD Cup 2009* we were provided with two data sets referred as *Small* and *Large*. These two data sets differed only in the number of features they consisted of. In the *Small* data set there were 190 numerical and 40 categorical features and the *Large* data consisted of 14740 numerical and 260 categorical features for a total of $d = 15000$. Both data sets contained the same 50000 training and 50000 test instances. Each instance had three different labels corresponding to the three tasks of *Churn*, *Appetency*, and *Up-selling*. About 65.4% (2%) of the values were missing in the *Small* (*Large*); we treated them by using an out-of-range value (i.e., we set all missing value to ∞).

Since the three tasks used the same instances, we experimented with both a *single-task* and a *multi-task* approach. In the former, the three classifiers were trained completely separately, whereas in the latter we trained one classifier with a three-element binary label.

First, we trained all of our models using the large feature set, only deleting features with one singular value. We also investigated the utility of the features. Using the info-gain based feature ranker of WEKA package (Witten and Frank, 2005), we found that only a relatively small number of features have positive score for any of task. In the single-task setup we used only those features which had positive score for the given task, and in the multi-task case we used those features which had positive score for at least one of the three tasks. The numbers of remaining features after feature selection are shown in Table 1. We also performed experiments where we applied PCA, but this do not results improvement in performance.

	single-task			*multi-task*
	Churn	*Appetency*	*Up-selling*	
Small	51	45	65	71
Large	2839	2546	4123	5543

Table 1: The number of features after applying feature selection.

All of the three KDD tasks were very imbalanced in size of classes: all three label sets contained only a small number of positive labels compared to the size of the whole data set. In order to handle this imbalance problem we tried a few initial weighting scheme beside the uniform weighting described in Eq. 2. We found that the best-performing weighting scheme was when both classes received half of the total weight, which meant that the instances from the positive class had higher initial weights than the instances form the negative class.

4.1.1 VALIDATION

During the challenge we validated only the number of iterations using a $60\% - 40\%$ simple validation on the training set. Figure 2 shows the AUC curves for the three tasks. Due to the time limit in the Fast Track we did not validate the number of tree leaves N and number of product terms m. We set the $N = 8$ and $m = 3$ based on the former experiments using our program package (Kégl and Busa-Fekete, 2009). The only remaining hyperparameter we had to choose was the number of best arms to be evaluated in the case of $\mathrm{UCB}(k)$. We set k to 50. We also carried out some experiments with a lower value ($k = 20$), but we found that this only slightly influenced the results.

Table 2 shows our official results. The *Churn* and *Up-selling* tasks were evaluated by ROC analysis and the *Appetency* task was evaluated using Balanced Accuracy[2].

learner \ data set	Churn (AUC)		Appetency (B. Acc.)		Up-selling (AUC)	
	Validation	Evaluation	Validation	Evaluation	Validation	Evaluation
STUMP	0.7424	0.6833	0.7051	0.7359	0.8938	**0.8917**
PRODCT	0.6702	0.6377	0.6999	0.6398	0.8888	0.8665
TREE	0.7088	**0.6819**	0.7424	**0.7216**	0.8956	0.8891
BEST	–	0.7611	–	0.883	–	0.9038

Table 2: The validation and evaluation results in Fast Track. The bold face values indicate our final submission.

2. http://www.kddcup-orange.com/

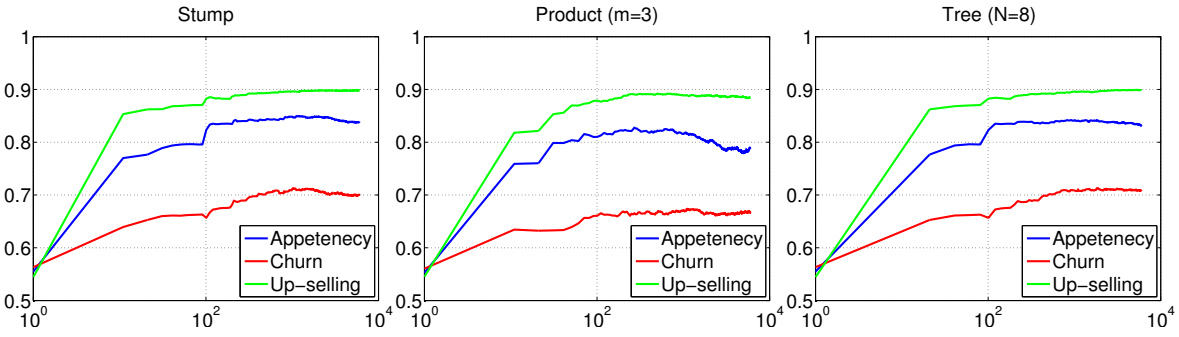

Figure 2: The ROC values vs. number of iterations on the validation set using ADABOOST.MH with Stump, Product, and Tree base learners.

As a post-challenge work we carried out a more comprehensive validation. We used the same $60\% - 40\%$ validation scheme but this time we validated all the parameters in a wide range: $2 \leq N \leq 32$, $2 \leq m \leq 10$, $T \leq 10000$. Similarly to other teams (Miller et al., 2009; IBM Research, 2009), we found that, relatively to other benchmarks, smaller trees and products worked better on this challenge. In the case of products, it turned out that the optimal number of terms m is in only a few times more than two (see in Tables 3 and 5). The optimal number of iterations was chosen based on the maximum of the ROC values on the validation set calculated in each iteration. In the multi-task case we used the average of the ROC values of the three tasks. Both in the single- and multi-task setup, the ROC values can have a large fluctuation from one iteration to another so we smoothed the learning curves using a moving average filtering with a relative window size of 20%. In general, the optimal numbers of iterations are also relatively small compared to the parameters of the experiments of (Kégl and Busa-Fekete, 2009).

4.2 Combination technique

We also tried to combine the three (stump, product, tree) learners. In case of *Appetency* we applied a simple majority voting. In the tasks where scores were required, we used the discriminant output $\mathbf{f}(\mathbf{x})$ rescaled into $[0, 1]$ of the trained models as posterior probabilities and we simply multiplied them.

4.3 Performance evaluation

4.3.1 LARGE DATA SET

Our official results in Fast Track on the *Large* data set using multi-task approach are shown in Table 2. Without the full knowledge of the validation we found that the multi-task approach using feature selection achieves better than single-task one.

Table 3 shows our post-challenge results and the validated parameters. The first four blocks (single/multi-task, feature selection on/off) are followed by results obtained using the combined models. We tried combining only the multi-task or only the single-task classifiers, then we combined all of them. The results revealed a few general trends. The single-task approach was superior in solving the *Churn* task whereas the *Appetency* task seemed to prefer the multi-task approach. In the

Up-selling problem both were competitive, and compared to the winning results we scored the best on this task.

	Churn (AUC)		Appetency (B. Acc.)		Up-selling (AUC)	
	Parameters	Evaluation	Parameters	Evaluation	Parameters	Evaluation
MULTI-TASK						
STUMP	$T = 1671$	0.6808		0.7176		0.8844
PRODUCT	$T = 1664$ $m = 2$	0.6877		**0.7388**		0.8817
TREE	$T = 4090$ $N = 3$	0.6918		0.7158		0.8870
MULTI-TASK/FEATURE SELECTION						
STUMP	$T = 369$	0.6802		0.7514		0.8804
PRODUCT	$T = 946$ $m = 2$	0.6749		0.6406		0.8818
TREE	$T = 1501$ $N = 4$	0.6871		0.6165		0.8822
SINGLE-TASK						
STUMP	$T = 1528$	0.7002	$T = 682$	0.5403	$T = 431$	0.7685
PRODUCT	$T = 448$ $m = 2$	0.6873	$T = 401$ $m = 2$	0.5726	$T = 344$ $m = 4$	0.7558
TREE	$T = 1362$ $N = 2$	0.6350	$T = 241$ $N = 5$	0.6719	$T = 1201$ $N = 2$	0.7784
SINGLE-TASK/FEATURE SELECTION						
STUMP	$T = 427$	0.7052	$T = 255$	0.5981	$T = 2260$	0.6449
PRODUCT	$T = 348$ $m = 2$	0.6887	$T = 9177$ $m = 4$	0.6875	$T = 410$ $m = 2$	0.8840
TREE	$T = 296$ $N = 2$	0.71	$T = 2514$ $N = 3$	0.6836	$T = 296$ $N = 2$	0.8762
COMBINED CLASSIFIERS						
MULTI-TASK		0.7197		0.7362		0.8920
SINGLE-TASK		0.7126		0.6312		0.8832
ALL		**0.7245**		0.7013		**0.8944**
WINNERS						
BEST/Fast[3]	–	0.7611	–	0.8830	–	0.9038
BEST/Slow[4]	–	0.7570	–	0.8836	–	0.9048

Table 3: The post-challenge results for *Large* data set with the validated parameters.

4. IBM Research (IBM Research, 2009)
5. University of Melbourne (Miller et al., 2009)

4.3.2 SMALL DATA SET

In the Slow Track we concentrated only on the *Small* data set. During the challenge our best results were obtained by single-task models using feature selection. Table 4 shows our official submission results and Table 5 shows our post-challenge results together with the validated parameters. We found that the multi-task approach did not work very well; our explanation for this is that the three tasks had different complexities and they needed very different number of iterations, so a single shared stopping time harmed the results. On the other hand, the single task approach worked very well in these experiments. In fact, the combined single-task post-challenge classifiers outperformed the best official results (among teams that did not used unscrambling).

learner \ data set	Churn (AUC)		Appetency (B. Acc.)		Up-selling (AUC)	
	Validation	Evaluation	Validation	Evaluation	Validation	Evaluation
STUMP	0.716862	**0.7258**	0.6712	0.7243	0.85747	**0.8582**
PRODCT	0.692894	0.6816	0.78055	**0.6915**	0.842176	0.8512
TREE	0.695429	0.7158	0.778751	0.6174	0.840568	0.8549
BEST	–	0.7375	–	0.8245	–	0.8620

Table 4: The validation and evaluation results in the Slow Track. The bold face values indicate our final submission.

4.4 Time complexity

Since out goal was to accelerate ADABOOST, we show in Table 6 the time (in minutes) needed for training and testing for 8000 iterations. The numbers indicate that all the models can be trained and tested in less than 10 hours on the *Large* database. For the *Small* data set the whole training time is typically less than an hour.

5. Discussion and Conclusions

The goal of this paper was to accelerate ADABOOST using multi-armed bandits. Recently, machine learning applications have become the center of interest in which millions of training examples and thousands of features are not uncommon. In this scenario, fast optimization becomes more important than the asymptotic statistical optimality (Bottou and Bousquet, 2008). From this point of view, our approach has solved the tasks well because it reduced greatly the computational complexity of the learning phase. In the official competition our approach achieved competitive results only on the *Up-selling* task. Based on our post-challenge analysis it seems that on small data set we could also have been competitive also on the *Churn* task, but since there was a confusion during the competition (people merged the small and large data sets), we decided to concentrate on the Large set. Our post-challenge results nevertheless confirmed that ADABOOST is among the best generic classification methods on large, real-world data sets.

6. University of Melbourne (Miller et al., 2009)

	Churn (AUC)		Appetency (B. Acc.)		Up-selling (AUC)	
	Parameters	Evaluation	Parameters	Evaluation	Parameters	Evaluation
MULTI-TASK						
STUMP	$T = 600$	0.7046		0.7045		0.8542
PRODUCT	$T = 150$ $m = 2$	0.7128		0.6677		0.8500
TREE	$T = 600$ $N = 2$	0.7255		0.7214		0.8578
MULTI-TASK/FEATURE SELECTION						
STUMP	$T = 740$	0.6273		0.5000		0.6872
PRODUCT	$T = 191$ $m = 2$	0.6188		0.5000		0.6805
TREE	$T = 194$ $N = 3$	0.6223		0.5006		0.6842
SINGLE-TASK						
STUMP	$T = 465$	0.7303	$T = 20$	0.5000	$T = 250$	0.8590
PRODUCT	$T = 191$ $m = 2$	0.7210	$T = 10$ $m = 2$	0.6135	$T = 174$ $m = 2$	0.8543
TREE	$T = 200$ $N = 2$	0.7278	$T = 100$ $N = 3$	0.6494	$T = 185$ $N = 2$	0.8571
SINGLE-TASK/FEATURE SELECTION						
STUMP	$T = 300$	0.7353	$T = 20$	0.5162	$T = 540$	0.8612
PRODUCT	$T = 154$ $m = 2$	0.7237	$T = 10$ $m = 4$	0.5318	$T = 345$ $m = 2$	0.8551
TREE	$T = 188$ $N = 2$	0.7317	$T = 191$ $N = 2$	**0.7207**	$T = 203$ $N = 2$	0.8606
COMBINED CLASSIFIERS						
MUTLI-TASK		0.7040		0.6600		0.8241
SINGLE-TASK		**0.7369**		0.6033		**0.8630**
ALL		0.7316		0.5918		0.8538
WINNERS						
BEST[5]	–	0.7375	–	0.8245	–	0.8620

Table 5: The post-challenge results for *Small* data set with the validated parameters.

Base learner	STUMP	PRODUCT	TREE
Time requirements	274	456	384

Table 6: Training and testing running times (in minutes) on the *Large* data set. The number of iterations is $T = 8000$.

References

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002a.

P. Auer, N. Cesa-Bianchi, Y. Freund, and R.E. Schapire. The non-stochastic multi-armed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002b.

A. Borisov, A. Eruhimov, and E. Tuv. *Tree-Based Ensembles with Dynamic Soft Feature Selection*, volume 207, pages 359–374. Springer, 2006.

L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, volume 20, pages 161–168, 2008.

J.K. Bradley and R.E. Schapire. FilterBoost: Regression and classification on large datasets. In *Advances in Neural Information Processing Systems*, volume 20. The MIT Press, 2008.

G. Escudero, L. Màrquez, and G. Rigau. Boosting applied to word sense disambiguation. In *Proceedings of the 11th European Conference on Machine Learning*, pages 129–141, 2000.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

S. Gelly and D. Silver. Achieving master level play in 9 x 9 computer go. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1537–1540, 2008.

IBM Research. Winning the KDD Cup Orange Challenge with ensemble selection. In *this volume*, pages 0–0, 2009.

B. Kégl and R. Busa-Fekete. Boosting products of base classifiers. In *Proceedings of the 26th International Conference on Machine Learning*, 2009.

H. Miller, S. Clarke, Lane S., A. D. Lazaridis, Petrovski S., and Jones O. Predicting customer behaviour: The University of Melbourne's KDD Cup report. In *this volume*, pages 0–0, 2009.

J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

E. Tuv, A. Borisov, G. Runger, and K. Torkkola. Feature selection with ensembles, artificial variables, and redundancy elimination. *Journal of Machine Learning Research*, 10:1341–1366, 2009.

P. Viola and M. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57:137–154, 2004.

I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.