Journal of Machine Learning Research 15 (2014) 1431-1453 Submitted 1/13; Revised 12/13; Published 4/14

Adaptive Sampling for Large Scale Boosting

Charles Dubout CHARLES@DUBOUT.CH
François Fleuret FRANCOIS.FLEURET@IDIAP.CH
Computer Vision and Learning Group
Idiap Research Institute
CH-1920 Martigny, Switzerland

Editor: Nicolas Vayatis

Abstract

Classical boosting algorithms, such as AdaBoost, build a strong classifier without concern for the computational cost. Some applications, in particular in computer vision, may involve millions of training examples and very large feature spaces. In such contexts, the training time of off-the-shelf boosting algorithms may become prohibitive. Several methods exist to accelerate training, typically either by sampling the features or the examples used to train the weak learners. Even if some of these methods provide a guaranteed speed improvement, they offer no insurance of being more efficient than any other, given the same amount of time.

The contributions of this paper are twofold: (1) a strategy to better deal with the increasingly common case where features come from multiple sources (for example, color, shape, texture, etc., in the case of images) and therefore can be partitioned into meaningful subsets; (2) new algorithms which balance at every boosting iteration the number of weak learners and the number of training examples to look at in order to maximize the expected loss reduction. Experiments in image classification and object recognition on four standard computer vision data sets show that the adaptive methods we propose outperform basic sampling and state-of-the-art bandit methods.

Keywords: boosting, large scale learning, feature selection

1. Introduction

Boosting is a simple and efficient machine learning algorithm which provides state-of-the-art performance on many tasks. It consists of building a strong classifier as a linear combination of weak learners, by adding them one after another in a greedy manner.

It has been repeatedly demonstrated that combining multiple kind of features addressing different aspects of the signal is an extremely efficient strategy to improve performance (Opelt et al., 2006; Gehler and Nowozin, 2009; Fleuret et al., 2011; Dubout and Fleuret, 2011a,b). As shown by our experimental results, vanilla boosting of stumps over multiple image features such as HOG, LBP, color histograms, etc., usually reaches close to state-of-the-art performance. However, such techniques entails a considerable computational cost, which increases with the number of features considered during training.

The critical operations contributing to the computational cost of a boosting iteration are the computations of the features and the selection of the weak learner. Both depend on the number of features and the number of training examples taken into account. While

textbook AdaBoost repeatedly selects each weak learner using all the features and all the training examples for a predetermined number of rounds, one is not obligated to do so and can instead choose to look only at a subset of both.

Since performance increases with both, one needs to balance the two to keep the computational cost under control. As boosting progresses, the performance of the candidate weak learners degrades, and they start to behave more and more similarly. While a small number of training examples is initially sufficient to characterize the good ones, as the learning problems become more and more difficult, optimal values for a fixed computational cost tend to move towards smaller number of features and larger number of examples.

In this paper, we present three new families of algorithms to explicitly address these issues: (1) Tasting (see Section 4 on page 1434) uses a small number of features sampled prior to learning to adaptively bias the sampling towards promising subsets at every step; (2) Maximum Adaptive Sampling (see Section 5.3 on page 1439) models the distribution of the weak learners' performance and the noise in order to determine the optimal trade-off between the number of weak learners and the number of examples to look at; and (3) Laminating (see Section 5.4 on page 1440) iteratively refines the learner selection using more and more examples.

2. Related Works

AdaBoost and similar boosting algorithms estimate for each candidate weak learner a score dubbed "edge", which requires to loop through every training example and take into account its weight, which reflects its current importance in the loss reduction. Reducing this computational cost is crucial to cope with high-dimensional feature spaces or very large training sets. This can be achieved through two main strategies: sampling the training examples, or the feature space, since there is a direct relation between features and weak learners.

Sampling the training set was introduced historically to deal with weak learners which cannot be trained with weighted examples (Freund and Schapire, 1996). This procedure consists of sampling examples from the training set according to their boosting weights, and of approximating a weighted average over the full set by a non-weighted average over the sampled subset. It is related to Bootstrapping as similarly the training algorithm will sample harder and harder examples based on the performance of the previous weak learners. See Section 3 for formal details. Such a procedure has been re-introduced recently for computational reasons (Bradley and Schapire, 2007; Duffield et al., 2007; Kalal et al., 2008; Fleuret and Geman, 2008), since the number of sampled examples controls the trade-off between statistical accuracy and computational cost.

Sampling the feature space is the central idea behind LazyBoost (Escudero et al., 2000), and simply consists of replacing the brute-force exhaustive search over the full feature set by an optimization over a subset produced by sampling uniformly a predefined number of features. The natural redundancy of most type of features makes such a procedure generally efficient. However, if a subset of important features is too small, it may be overlooked during training.

Recently developed algorithms rely on multi-arms bandit methods to balance properly the exploitation of features known to be informative, and the exploration of new features

(Busa-Fekete and Kegl, 2009, 2010). The idea behind those methods is to associate a bandit arm to every feature, and to see the loss reduction as a reward. Maximizing the overall reduction is achieved with a standard bandit strategy such as UCB (Auer et al., 2002), or Exp3.P (Auer et al., 2003).

These techniques suffer from two important drawbacks. First they make the assumption that the quality of a feature — the expected loss reduction of a weak learner using it — is stationary. This goes against the underpinning of boosting, which is that at any iteration the performance of the weak learners is relative to the boosting weights, which evolve over the training (Exp3.P does not make such an assumption explicitly, but still rely exclusively on the history of past rewards). Second, without additional knowledge about the feature space, the only structure they can exploit is the stationarity of individual features. Hence, improvement over random selection can only be achieved by sampling again *the exact same features* already seen in the past. In our experiments, we therefore only use those methods in a context where features can be partitioned into subsets of different types. This allows us to model the quality, and thus to bias the sampling, at a higher level than individual features.

All those approaches exploit information about features to bias the sampling, hence making it more efficient, and reducing the number of weak learners required to achieve the same loss reduction. However, they do not explicitly aim at controlling the computational cost. In particular, there is no notion of varying the number of examples used for the estimation of the loss reduction.

3. Preliminaries

We first present in this section some analytical results to approximate a standard round of AdaBoost — or other similar boosting algorithms — by sampling both the training examples and the features used to build the weak learners. We then precise more formally what we mean by subset of features or weak learners.

3.1 Standard Boosting

Given a binary training set

$$(x_n, y_n) \in \mathcal{X} \times \{-1, 1\}, n = 1, \dots, N,$$

where $\mathcal{X}$ is the space of the "visible" signal, and a set $\mathcal{H}$ of weak learners of the form $h : \mathcal{X} \to \{-1, 1\}$, the standard boosting procedure consists of building a strong classifier

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x),$$

by choosing the terms $\alpha_t \in \mathbb{R}$ and $h_t \in \mathcal{H}$ in a greedy manner so as to minimize a loss (for example the empirical exponential loss in the case of AdaBoost) estimated over the training examples. At every iteration, choosing the optimal weak learner boils down to finding the one with the largest edge ϵ, which is the derivative of the loss reduction w.r.t. the weak learner weight α. The higher this value, the more the loss can be reduced locally, and thus

the better the weak learner. The edge is a linear function of the responses of the weak learner over the training examples

$$\epsilon(h) = \sum_{n=1}^{N} \omega_n y_n h(x_n),$$

where the weights ω_n's depend on the loss function (usually either the exponential or logistic loss) and on the current responses of f over the x_n's. We consider without loss of generality that they have been normalized such that $\sum_n \omega_n = 1$. We can therefore consider the weights ω_n's as a distribution over the training examples and rewrite the edge as an expectation over them,

$$\epsilon(h) = \mathrm{E}_{N \sim \omega_n}[y_N h(x_N)], \tag{1}$$

where $N \sim \omega_n$ stands for $\mathrm{P}(N = n) = \omega_n$. The idea of weighting-by-sampling (Fleuret and Geman, 2008) consists of replacing the expectation in Equation (1) with an approximation obtained by sampling. Let $N_1, \ldots, N_S$, be i.i.d. random variables distributed according to the discrete probability density distribution defined by the ω's, we define the approximated edge as

$$\hat{\epsilon}(h) = \frac{1}{S} \sum_{s=1}^{S} y_{N_s} h(x_{N_s}), \tag{2}$$

which follows a binomial distribution centered on the true edge, with a variance decreasing with the number of sampled examples S. It is accurately modeled by the Gaussian

$$\hat{\epsilon}(h) \approx \mathcal{N}\left(\epsilon(h), \frac{1}{S}\right), \tag{3}$$

as the approximation holds asymptotically and the magnitude of the weak learners' edges is typically small, such that $(1 + \epsilon(h))(1 - \epsilon(h)) \approx 1$.

3.2 Feature Subsets

It frequently happens that the features making up the signal space $\mathcal{X}$ can be divided into meaningful disjoint subsets $\mathcal{F}_k$ such that $\mathcal{X} = \cup_{k=1}^{K} \mathcal{F}_k$. This division can for example be the result of the features coming from different sources or some natural clustering of the feature space. In such a case it makes sense to use this information during training, as features coming from the same subset $\mathcal{F}_k$ can typically be expected to be more homogeneous than features coming from different subsets.

4. Tasting

We describe here our approach called Tasting (Dubout and Fleuret, 2011a) which biases the sampling toward promising subsets of features. Tasting in its current form is limited to deal with weak learners looking at a single feature, such as decision stumps. Extending it to deal efficiently with weak learners looking at multiple features is outside of the scope of this work.

4.1 Main Algorithm

The core idea of Tasting is to sample a small number R of features from every subset before starting the training *per se* and, at every boosting step, in using these few features together with the current boosting weights to get an estimate of the best subset(s) $\mathcal{F}_k$(s) to use.

We cannot stress enough that these R features are not the ones used to build the classifier, they are only used to figure out what is/are the best subset(s) at any time during training. As those sampled features are independent and identically distributed samples of the feature response vectors, we can compute the empirical mean of any functional of the said response vectors, in particular the expected loss reduction.

At any boosting step, Tasting requires, for any feature subset, an estimate of the expectation of the edge of the best weak learner we would obtain by sampling uniformly Q features from this subset and picking the best weak learner using one of them,

$$\mathrm{E}_{F_1,\ldots,F_Q \sim \mathcal{U}(\mathcal{F}_k)} \left[\max_{q=1}^{Q} \max_{h \in \mathcal{H}_{F_q}} \epsilon(h) \right], \tag{4}$$

where $\mathcal{F}_k$ are the indices of the features belonging to the k-th subset and $\mathcal{H}_F$ is the space of weak learners looking solely at feature F. Hence $\max_{h \in \mathcal{H}_{F_q}} \epsilon(h)$ is the best weak learner looking solely at feature F_q, and $\max_{q=1}^{Q} \max_{h \in \mathcal{H}_{F_q}} \epsilon(h)$ is the best weak learner looking solely at one of the Q features $F_1, \ldots, F_Q$.

We can build an approximation of this quantity using the R features we have stored. Let $\epsilon_1, \ldots, \epsilon_R$ be the edges of the best R weak learners built from these features. We make the assumption without loss of generality that $\epsilon_1 \leq \epsilon_2 \leq \cdots \leq \epsilon_R$. Let $R_1, \ldots, R_Q$ be independent and identically distributed, uniform over $\{1, \ldots, R\}$. We approximate the quantity in Equation (4) with

$$\begin{aligned}
\mathrm{E}\left[\max_{q=1}^{Q} \epsilon_{R_q} \right] &= \sum_{r=1}^{R} \mathrm{P}\left(\max_{q=1}^{Q} R_q = r \right) \epsilon_r \\
&= \sum_{r=1}^{R} \left[\mathrm{P}\left(\max_{q=1}^{Q} R_q \leq r \right) - \mathrm{P}\left(\max_{q=1}^{Q} R_q \leq r-1 \right) \right] \epsilon_r \\
&= \frac{1}{R^Q} \sum_{r=1}^{R} \left[r^Q - (r-1)^Q \right] \epsilon_r .
\end{aligned} \tag{5}$$

4.2 Tasting Variants

We propose two versions of the Tasting procedure, which differ in the number of feature subsets they visit at every iteration. Either one for Tasting 1.Q or up to Q for Tasting Q.1.

In Tasting 1.Q (Algorithm 1), the selection of the optimal subset k^* from which to sample the Q features is accomplished by estimating for every subset the expected maximum edge, which is directly related to the expected loss reduction, if we were sampling from that subset only. The computation is done over the R features saved before starting training, which serve as a representation of the full set $\mathcal{F}_k$.

In Tasting Q.1 (see Algorithm 2), it is not one but several feature subsets which can be selected, as the algorithm picks the best subset k_q^* for every one of the Q features to

Algorithm 1 The Tasting 1.Q algorithm first samples uniformly R features from every feature subset $\mathcal{F}_k$. It uses these features at every boosting step to find the optimal feature subset k^* from which to sample. After the selection of the Q features, the algorithm continues like AdaBoost.

Input: $\mathcal{F}, Q, R, T$

Initialize: $\forall k \in \{1, \dots, K\}, \forall r \in \{1, \dots, R\}, f_r^k \leftarrow \text{sample}(\mathcal{U}(\mathcal{F}_k))$

for $t = 1, \dots, T$ **do**

$\forall k \in \{1, \dots, K\}, \forall r \in \{1, \dots, R\}, \epsilon_r^k \leftarrow \max_{h \in \mathcal{H}_{f_r^k}} \epsilon(h)$

$k^* \leftarrow \operatorname*{argmax}_k \mathrm{E}\left[\max_{q=1}^{Q} \epsilon_{R_q}^k\right]$ `# Computed using equation (5)`

$\forall q \in \{1, \dots, Q\}, F_q \leftarrow \text{sample}(\mathcal{U}(\mathcal{F}_{k^*}))$

$h_t \leftarrow \operatorname*{argmax}_{h \in \cup_q \mathcal{H}_{F_q}} \epsilon(h)$

$\dots$

end for

Algorithm 2 The Tasting Q.1 algorithm first samples uniformly R features from every feature subset $\mathcal{F}_k$. It uses them to find the optimal subset k_q^* for every one of the Q features to sample at every boosting step. After the selection of the Q features, the algorithm continues like AdaBoost.

Input: $\mathcal{F}, Q, R, T$

Initialize: $\forall k \in \{1, \dots, K\}, \forall r \in \{1, \dots, R\}, f_r^k \leftarrow \text{sample}(\mathcal{U}(\mathcal{F}_k))$

for $t = 1, \dots, T$ **do**

$\forall k \in \{1, \dots, K\}, \forall r \in \{1, \dots, R\}, \epsilon_r^k \leftarrow \max_{h \in \mathcal{H}_{f_r^k}} \epsilon(h)$

$\epsilon^* \leftarrow 0$

for $q = 1, \dots, Q$ **do**

$k_q^* \leftarrow \operatorname*{argmax}_k \mathrm{E}\left[\max\left(\epsilon^*, \epsilon^k\right)\right]$

$F_q \leftarrow \text{sample}\left(\mathcal{U}\left(\mathcal{F}_{k_q^*}\right)\right)$

$\epsilon^* \leftarrow \max\left(\epsilon^*, \max_{h \in \mathcal{H}_{F_q}} \epsilon(h)\right)$

end for

$h_t \leftarrow \operatorname*{argmax}_{h \in \cup_q \mathcal{H}_{F_q}} \epsilon(h)$

$\dots$

end for

sample, given the best edge ϵ^* achieved so far. Again the computation is done only over the R features saved before starting training.

4.3 Relation with Bandit Methods

The main strength of boosting is its ability to spot and combine complementary features. If the loss has already been reduced in a certain "functional direction", the scores of weak learners in the same direction will be low, and they will be rejected. For instance, the firsts learners for a face detector may use color-based features to exploit the skin color. After a few boosting steps using this modality, color would be exhausted as a source of information, and only examples with a non-standard face color would have large weights. Other features, for instance edge-based, would become more informative, and be picked.

Uniform sampling of features accounts poorly for such behavior since it simply discards the boosting weights, and hence has no information whatsoever about the directions which have "already been exploited" and which should be avoided. In practice, this means that the rejection of bad feature can only be done at the level of the boosting itself, which may end up with a majority of useless features.

Bandit methods (described in Section 6.4) are slightly more adequate, as they model the performance of every feature from previous iterations. However, this modeling takes into account the boosting weights very indirectly, as they make the assumption that the distributions of loss reduction are stationary, while they are precisely not. Coming back to our face-detector example, bandit methods would go on believing that color is informative since it was in the previous iterations, even if the boosting weights have specifically accumulated on faces where color is now totally useless. While the estimate of loss reduction may asymptotically converge to an adequate model, it is a severe weakness while the boosting weights are still evolving.

Tasting addresses this weakness by keeping the ability to properly estimate the performance of every feature subset, *given the current boosting weights*, hence the ability to discard feature subsets redundant with features already picked. In some sense, Tasting can be seen as boosting done at a the subset level.

5. Maximum Adaptive Sampling and Laminating

The algorithms in this section sample both the weak learners and the training examples at every iteration in order to maximize the expectation of the loss reduction, under a strict computational cost constraint.

5.1 Edge estimation

At every iteration they model the expectation of the edge of the selected weak learner. Let $\epsilon_1, \ldots, \epsilon_Q$ stand for the true edges of Q independently sampled weak learners. Let $\Delta_1, \ldots, \Delta_Q$ be a series of independent random variables standing for the noise in the estimation of the edges due to the sampling of only S training examples. Finally $\forall q$, let $\hat{\epsilon}_q = \epsilon_q + \Delta_q$ be the approximated edge. With these definitions, $\operatorname{argmax}_q \hat{\epsilon}_q$ is the selected weak learner. We define ϵ^* as the true edge of the selected weak learner, that is the one

with the highest approximated edge

$$\epsilon^* = \epsilon_{\text{argmax}_q \hat{\epsilon}_q}. \tag{6}$$

This quantity is random due to both the sampling of the weak learners, and the sampling of the training examples. The quantity we want to optimize is $\mathrm{E}[\epsilon^*]$, the expectation of the true edge of the selected learner over all weak learners and over all training examples, which increases with both Q and S. A higher Q increases the number of terms in the maximization of Equation (6), while a higher S reduces the variance of the Δ's, ensuring that ϵ^* is closer to $\max_q \epsilon_q$. In practice, if the variance of the Δ's is of the order of, or higher than, the variance of the ϵ's, the maximization is close to a random selection, and looking at many weak learners is useless. Assuming that the $\hat{\epsilon}_q$'s are all different we have,

$$\begin{aligned}
\mathrm{E}[\epsilon^*] &= \mathrm{E}\Big[\epsilon_{\text{argmax}_q \hat{\epsilon}_q}\Big] \\
&= \sum_{q=1}^{Q} \mathrm{E}\left[\epsilon_q \prod_{i \neq q} \mathbf{1}_{\{\hat{\epsilon}_i < \hat{\epsilon}_q\}}\right] \\
&= \sum_{q=1}^{Q} \mathrm{E}\left[\mathrm{E}\left[\epsilon_q \prod_{i \neq q} \mathbf{1}_{\{\hat{\epsilon}_i < \hat{\epsilon}_q\}} \,\middle|\, \hat{\epsilon}_q\right]\right] \\
&= \sum_{q=1}^{Q} \mathrm{E}\left[\mathrm{E}[\epsilon_q \mid \hat{\epsilon}_q] \prod_{i \neq q} \mathrm{E}\big[\mathbf{1}_{\{\hat{\epsilon}_i < \hat{\epsilon}_q\}} \mid \hat{\epsilon}_q\big]\right],
\end{aligned}$$

where the last equality follows from the independence of the weak learners.

5.2 Modeling the True Edge

If the distributions of the ϵ_q's and the Δ_q's are Gaussians or mixtures of Gaussians, we can derive analytical expressions for both $\mathrm{E}[\epsilon_q \mid \hat{\epsilon}_q]$ and $\mathrm{E}\big[\mathbf{1}_{\{\hat{\epsilon}_i < \hat{\epsilon}_q\}} \mid \hat{\epsilon}_q\big]$, and compute the value of $\mathrm{E}[\epsilon^*]$ efficiently. In the case where weak learners can be partitioned into meaningful subsets, it makes sense to model the distributions of the edges separately for each subset.

As an illustrative example, we consider here the case where the ϵ_q's, the Δ_q's, and hence also the $\hat{\epsilon}_q$'s all follow Gaussian distributions. We take $\epsilon_q \sim \mathcal{N}(0,1)$ and $\Delta_q \sim \mathcal{N}(0,\sigma^2)$ and obtain,

$$\begin{aligned}
\mathrm{E}[\epsilon^*] &= Q\ \mathrm{E}\left[\mathrm{E}[\epsilon_1 \mid \hat{\epsilon}_1] \prod_{i \neq 1} \mathrm{E}\big[\mathbf{1}_{\{\hat{\epsilon}_i < \hat{\epsilon}_1\}} \mid \hat{\epsilon}_1\big]\right] \\
&= Q\ \mathrm{E}\left[\frac{\hat{\epsilon}_1}{\sigma^2+1} \Phi\left(\frac{\hat{\epsilon}_1}{\sqrt{\sigma^2+1}}\right)^{Q-1}\right] \\
&= \frac{Q}{\sqrt{\sigma^2+1}}\ \mathrm{E}\Big[\epsilon_1 \Phi(\epsilon_1)^{Q-1}\Big] \\
&= \frac{1}{\sqrt{\sigma^2+1}}\ \mathrm{E}\left[\max_{1 \leq q \leq Q} \epsilon_q\right],
\end{aligned}$$

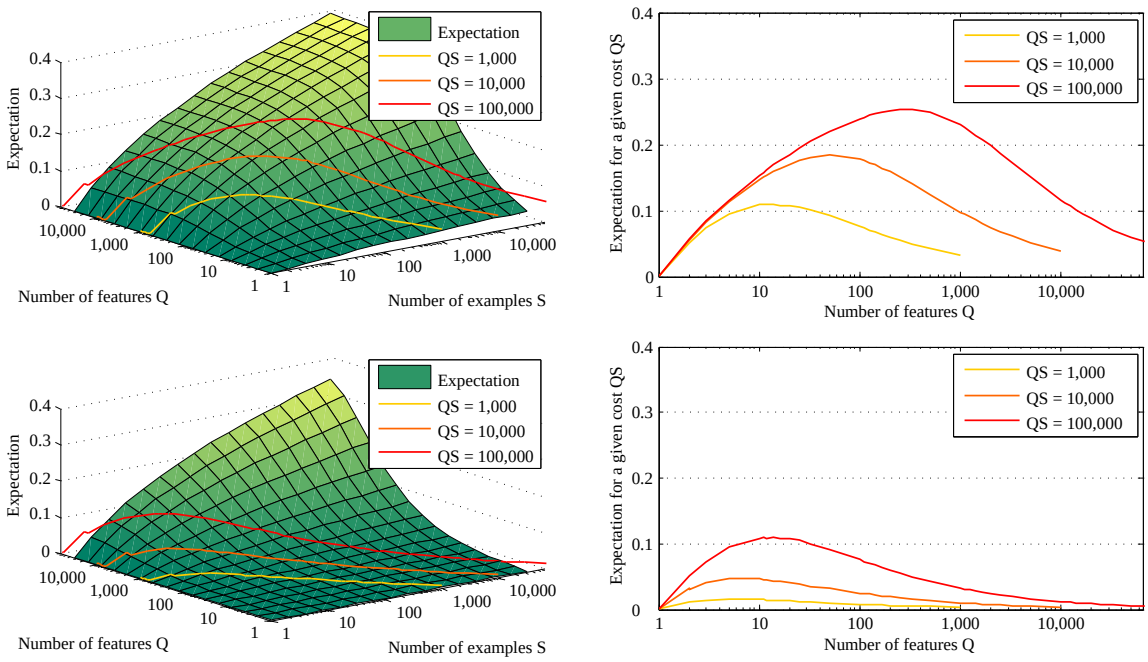

Figure 1: Simulation of the expectation of ϵ^* in the case where both the ϵ_q's and the Δ_q's follow Gaussian distributions. Top: $\epsilon_q \sim \mathcal{N}(0, 10^{-2})$. Bottom: $\epsilon_q \sim \mathcal{N}(0, 10^{-4})$. In both simulations $\Delta_q \sim \mathcal{N}(0, \frac{1}{S})$. Left: expectation of ϵ^* vs. the number of sampled weak learners Q and the number of examples S. Right: same value as a function of Q alone, for different fixed costs (product of Q and S). As these graphs illustrate, the optimal value for Q is greater for larger variances of the ϵ_q's. In such a case the ϵ_q's are more spread out, and identifying the largest one can be done despite a large noise in the estimations, hence with a limited number of training examples.

where Φ stands for the cumulative distribution function of the unit Gaussian, and σ^2 is of order $\frac{1}{S}$. See Figure 1 for an illustration of the behavior of $\mathrm{E}[\epsilon^*]$ for two different variances of the ϵ_q's and a cost proportional to QS, the total number of features computed.

There is no reason to expect the distribution of the ϵ_q's to be Gaussian, contrary to the Δ_q's, as shown in Equation (3), but this is not a problem as it can usually be approximated by a mixture, for which we can still derive analytical expressions, even if the ϵ_q's or the Δ_q's have different distributions for different q's.

5.3 M.A.S. Variants

We created three algorithms modeling the distribution of the ϵ_q's with a Gaussian mixture model, and $\Delta_q = \hat{\epsilon}_q - \epsilon_q$ as a Gaussian. The first one, M.A.S. naive, is described in Algorithm 3, and fits the model to the edges estimated at the previous iteration.

Algorithm 3 The M.A.S. naive algorithm models the current edge distribution with a Gaussian mixture model fitted on the edges estimated at the previous iteration. It uses this density model to compute the pair (Q^*, S^*) maximizing the expectation of the true edge of the selected learner $\mathrm{E}[\epsilon^*]$, and then samples the corresponding number of weak learners and training examples, before keeping the weak learner with the highest approximated edge. After the selection of the Q features, the algorithm continues like AdaBoost.

Input: $gmm, Cost$

for $t = 1, \ldots, T$ **do**

$(Q^*, S^*) \leftarrow \underset{\mathrm{cost}(Q,S) \leq Cost}{\mathrm{argmax}} \mathrm{E}_{gmm}[\epsilon^*]$

$\forall q \in \{1, \ldots, Q^*\}, H_q \leftarrow \mathrm{sample}(\mathcal{U}(\mathcal{H}))$

$\forall s \in \{1, \ldots, S^*\}, N_s \leftarrow \mathrm{sample}(\mathcal{U}(\{1, \ldots, N\}))$

$\forall q \in \{1, \ldots, Q^*\}, \hat{\epsilon}_q \leftarrow \frac{1}{S^*} \sum_{s=1}^{S^*} y_{N_s} H_q(x_{N_s})$ `# Similar to equation (2)`

$h_t \leftarrow H_{\mathrm{argmax}_q \hat{\epsilon}_q}$

$gmm \leftarrow \mathrm{fit}(\hat{\epsilon}_1, \ldots, \hat{\epsilon}_{Q^*})$

...

end for

The second one, M.A.S. 1.Q, takes into account the decomposition of the weak learners into K subsets, associated to different kind of features. It models the distributions of the ϵ_q's separately for each subset, estimating the distribution of each on a small number of weak learners and examples sampled at the beginning of each boosting iteration, chosen so as to account for 10% of the total computational cost. From these models, it optimizes Q, S, and the index k of the subset to sample from. Unlike M.A.S. naive, it has to draw a small number of weak learners and examples in order to fit the model since the edges estimated at the previous iterations came from a unique subset.

Finally M.A.S. Q.1 similarly models the distributions of the ϵ_q's, but it optimizes $Q_1, \ldots, Q_K$ greedily, starting from $Q_1 = 0, \ldots, Q_K = 0$, and iteratively incrementing one of the Q_k so as to maximize $\mathrm{E}[\epsilon^*]$. This greedy procedure is repeated for different values of S and ultimately the $Q_1, \ldots, Q_K, S$ leading to the maximum expectation are selected.

5.4 Laminating

The last algorithm we have developed tries to reduce the requirement for a density model of the ϵ_q's. At every boosting iteration it iteratively reduces the number of considered weak learners, and increases the number of examples taken into account.

Given fixed Q and S, at every boosting iteration, the Laminating algorithm first samples Q weak learners and S training examples. Then, it computes the approximated edges and keeps the $\frac{Q}{2}$ best learners. If more than one remains, it samples $2S$ examples, and re-iterates. The whole process is described in Algorithm 4. The number of iterations is bounded by $\lceil \log_2(Q) \rceil$.

Algorithm 4 The Laminating algorithm starts by sampling Q weak learners and S examples at the beginning of every boosting iteration, and refine those by successively halving the number of learners and doubling the number of examples until only one learner remains. After the selection of the Q features, the algorithm continues like AdaBoost.

Input: Q, S

for $t = 1, \ldots, T$ **do**

$\forall q \in \{1, \ldots, Q\}, h_q \leftarrow \text{sample}(\mathcal{U}(\mathcal{H}))$

while $Q > 1$ **do**

$\forall s \in \{1, \ldots, S\}, N_s \leftarrow \text{sample}(\mathcal{U}(\{1, \ldots, N\}))$

$\forall q \in \{1, \ldots, Q\}, \hat{\epsilon}_q \leftarrow \frac{1}{S} \sum_{s=1}^{S} y_{N_s} H_q(x_{N_s})$ `# Similar to equation (2)`

$\text{sort}(h_1, \ldots, h_Q)$ s.t. $\hat{\epsilon}_1 \geq \cdots \geq \hat{\epsilon}_Q$ `# Order the weak learners s.t.`

$Q \leftarrow \frac{Q}{2}$ `# the best half comes first`

$S \leftarrow 2S$

end while

...

end for

We have the following results on the accuracy of this Laminating procedure (the proof is given in Appendix A):

Lemma 1 *Let $q^\star = \text{argmax}_q \epsilon_q$ and $\delta > 0$. The probability for an iteration of the Laminating algorithm to retain only weak learners with edges below or equal to $\epsilon_{q^\star} - \delta$ is*

$$\mathrm{P}\left(\left|\left\{q : \epsilon_q \leq \epsilon_{q^\star} - \delta,\, \hat{\epsilon}_q \geq \max_{r:\epsilon_r \geq \epsilon_{q^\star} - \delta} \hat{\epsilon}_r\right\}\right| \geq \frac{Q}{2}\right) \leq 4 \exp\left(-\frac{\delta^2 S}{2}\right).$$

This holds regardless of the independence of the ϵ_q's and/or the Δ_q's.

Since at each iteration the number of examples S doubles the lemma implies the following theorem:

Theorem 1 *The probability for the full Laminating procedure starting with Q weak learners and S examples to end up with a learner with an edge below or equal to $\epsilon_{q^*} - \delta$ (the edge of the optimal weak learner at the start of the procedure minus δ) is upper bounded by (the proof is given in Appendix B)*

$$\frac{4}{1 - \exp\left(-\frac{\delta^2 S}{69}\right)} - 4.$$

The theorem shows that as the number of samples grows, the probability to retain a bad weak learner eventually goes down exponentially with the number of training examples, as in this case the bound can be approximated by $4 \exp\left(-\frac{\delta^2 S}{69}\right)$. This confirms the usual relation between the number of examples and the complexity of the space of predictors in learning theory.

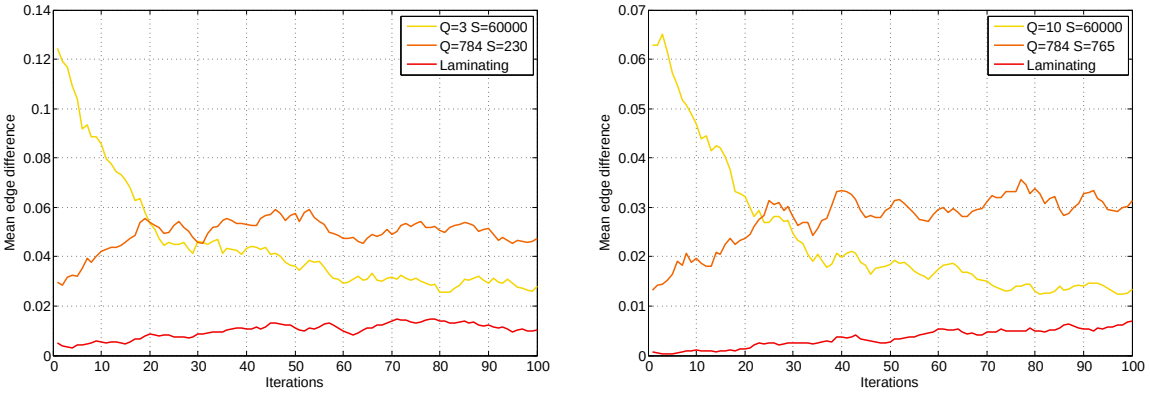

Figure 2: Difference between the maximum edge and the best edge found by 3 different sampling strategies on the MNIST data set using the original features. The algorithm used is AdaBoost.MH using $T = 100$ decision stumps as weak learners, and the results were averaged over 10 randomized runs. The first strategy samples uniformly a small number of features Q and determines the best one using all $S = 60,000$ training examples. The second strategy samples all $Q = 784$ features and determines the best one using a small number of training examples S. The third strategy is Laminating, starting from all the features and a suitable number of training examples chosen so as to have the same cost as the first two strategies. The cost is the product of Q and S and is set to $QS = 180,000$ for the left figure and $QS = 600,000$ for the right one.

In practice the difference between the maximum edge $\epsilon_{q^\star}$ and the edge of the final weak learner selected by Laminating is typically smaller than the difference with the edge of a learner selected by a strategy looking at a fixed number of weak learners and training examples, as can be observed in Figure 2.

6. Experiments

We demonstrate the effectiveness of our approaches on four standard image classification and object detection data sets, using 19 kinds of features (33 on Caltech 101) divided in as many subsets. We used the AdaBoost.MH algorithm (Schapire and Singer, 1999) with decision stumps as weak learners to be able to use all methods in the same conditions.

6.1 Features

The features used in our experiments with all but the Caltech 101 data set can be divided into three categories. (1) Image transforms: identity, grayscale conversion, Fourier and Haar transforms, gradient image, local binary patterns (ILBP/LBP). (2) Intensity histograms: sums of the intensities in random image patches, grayscale and color histograms of the entire image. (3) Gradient histograms: histograms of (oriented and non oriented) gradients, Haar-like features.

The features from the first category typically have a large dimensionality, usually proportional to the number of pixels in the image. Some of them do not pre-process the images (identity, grayscale conversion, LBP, etc.) while some pre-transform them to another space, prior to accessing any feature (typically the Fourier and Haar transforms).

Features from the second and third categories being histograms, they are usually much smaller (containing typically of the order of a few hundreds to a few thousands coefficients), but require some pre-processing to build the histograms.

For the Caltech 101 data set we used the same features as (Gehler and Nowozin, 2009) in their experiments. They used five type of features: PHOG shape descriptors, appearance (SIFT) descriptors, region covariance, local binary patterns, and V1S+, which are normalized Gabor filters. More details can be found in the referenced paper. Those features are computed in a spatial pyramid, where each scale of the pyramid is considered as being part of a different subset, leading to a total of 33 features. The number of features used in our experiments (33) differ from (Gehler and Nowozin, 2009) as they also compute a 'subwindow-kernel' of SIFT features which we did not use.

6.2 Data sets

Figure 3: Example images from the four data sets used for the experimental validation. Top left: first image of every digit taken from the MNIST database. Top right: images from the INRIA Person data set. Bottom left: random images from the Caltech 101 data set. Bottom right: some of the first images of the CIFAR-10 data set.

The first data set that we used is the MNIST handwritten digits database (LeCun et al., 1998). It is composed of 10 classes and its training and testing sets consist respectively of 60,000 and 10,000 grayscale images of resolution 28×28 pixels (see the upper left part of Figure 3 for some examples). The total number of features on this data set is 16,775.

The second data set that we used is the INRIA Person data set (Dalal and Triggs, 2005). It is composed of a training and a testing set respectively of 2,418 and 1,126 color images of

pedestrians of dimensions 64×128 pixels cropped from real-world photographs, along with 1,219 and 453 "background" images not containing any people (see the upper right part of Figure 3 for some examples). We extracted 10 negative samples from each one of the background image, following the setup of (Dalal and Triggs, 2005). The total number of features on this data set is 230,503.

The third data set that we used is Caltech 101 (Fei-Fei et al., 2004) due to its wide usage and the availability of already computed features (Gehler and Nowozin, 2009). It consists of color images of various dimensions organized in 101 object classes (see the bottom left part of Figure 3 for some examples). We sampled 15 training examples and 20 distinct test examples from every class, as advised on the data set website. The total number of features on this data set is 360,630.

The fourth and last data set that we used is CIFAR-10 (Krizhevsky, 2009). It is a labeled subset of the 80 tiny million images data set. It is composed of 10 classes and its training and testing sets consist respectively of 50,000 and 10,000 color images of size 32×32 pixels (see the bottom left part of Figure 3 for some examples). The total number of features on this data set is 29,879.

6.3 Uniform Sampling Baselines

A naive sampling strategy would pick the Q features uniformly in $\cup_k \mathcal{F}_k$. However, this does not distribute the sampling properly among the $\mathcal{F}_k$'s. In the extreme case, if one of the $\mathcal{F}_k$ had a far greater cardinality than the others, all features would come from it. And in most contexts, mixing features from the different $\mathcal{F}_k$'s in an equilibrate manner is critical to benefit from their complementarity. We propose the four following baselines to pick a good feature at every boosting step:

- **Best subset** picks Q features at random in a fixed subset, the one with the smallest final boosting loss.
- **Uniform Naive** picks Q features at random, uniformly in $\cup_k \mathcal{F}_q$.
- **Uniform 1.Q** picks one of the feature subsets at random, and then samples the Q features from that single subset.
- **Uniform Q.1** picks at random, uniformly, Q subsets of features (with replacement if $Q > K$), and then picks one feature uniformly in each subset.

The cost of running **Best subset** is K times higher than running the other three strategies since the subset leading to the smallest final boosting loss is not known a priori and requires to redo the training K times. Also, since it makes use of one subset only we can expect its final performance to be lower than the others. It was included for comparison only.

6.4 Bandit Sampling Baselines

The strategies of the previous section are purely random and do not exploit any kind of information to bias their sampling. Smarter strategies to deal with the problem of exploration-exploitation trade-off were first introduced in (Busa-Fekete and Kegl, 2009), and extended in (Busa-Fekete and Kegl, 2010). The driving idea of these papers is to

entrust a multi-armed bandits (MAB) algorithm (respectively UCB in Auer et al. (2002) and Exp3.P in Auer et al. (2003)) with the mission to sample useful features.

The multi-armed bandits problem is defined as follows: there are M gambling machines (the "arms" of the bandits), and at every time-step t the gambler chooses an arm j_t, pulls it, and receives a reward $r^t_{j_t} \in [0,1]$. The goal of the algorithm is to minimize the weak-regret, that is the difference between the reward obtained by the gambler and the best fixed arm, retrospectively.

The first weakness of these algorithms in the context of accelerating boosting, that we have identified in Section 2, is the assumption of stationarity of the loss reduction, which cannot be easily dealt with. Even though the Exp3.P algorithm does not make such an assumption explicitly, it still ignores the boosting weights, and thus can only rely on the history of past rewards.

The second weakness, the application context, can be addressed in our setting by learning the usefulness of the subsets instead of individual features.

A third weakness is that in boosting one aims at minimizing the loss (which translates into maximizing the sum of the rewards for the bandit algorithm), and not at minimizing the weak-regret.

Finally, another issue arises when trying to use those algorithms in practice. As they use some kind of confidence intervals, the scale of the rewards matters greatly. For example, if all the rewards obtained are very small ($\forall t, r^t \leq \epsilon \ll 1$), the algorithms will not learn anything, as they expect rewards to make full use of the range $[0,1]$.

For this reason we set the bandit baselines' meta-parameters to the ones leading to the lowest loss a posteriori, as explained in Section 6.5, and use a third multi-armed bandit algorithm in our experiments, ϵ-greedy (Auer et al., 2002), which does not suffer from this problem.

Hence, we use in our experiments the three following baselines, using the same reward as in (Busa-Fekete and Kegl, 2010):

- **UCB** picks Q features from the subset that maximizes $\bar{r}_j + \sqrt{(2\log n)/n_j}$, where $\bar{r}_j$ is the current average reward of subset j, n_j is the number of times subset j was chosen so far, and n is the current boosting round.

- **Epx3.P** maintains a distribution of weights over the feature subsets, and at every round picks one subset accordingly, obtains a reward, and updates the distribution. For the precise definition of the algorithm, see (Auer et al., 2003; Busa-Fekete and Kegl, 2010).

- **ϵ-greedy** picks Q features from the subset with the highest current average reward with probability $1-\epsilon_n$, or from a random subset with probability ϵ_n, where $\epsilon_n = \frac{cK}{d^2 n}$, and c and d are parameters of the algorithm.

6.5 Results

We tested all the proposed methods of Sections 4, 5.3, and 5.4 against the baselines described in Sections 6.3 and 6.4 on the four benchmark data sets described above in Section 6.2 using the standard train/test cuts and all the features of Section 6.1. We report the results of doing

MNIST								
Methods	T (# boosting steps)							
	10		100		1,000		10,000	
	loss	test error	loss	test error	loss	test error	loss	test error
Best family*	-0.43	36.48	-0.95	5.77	-1.84	1.47	-4.84	0.92
Uniform Naive	-0.38	45.3	-0.85	7.79	-1.74	1.64	-5.37	0.93
Uniform 1.Q	-0.36	49.4	-0.75	10.8	-1.51	2.18	-3.90	1.08
Uniform Q.1	-0.38	43.0	-0.86	7.40	-1.72	1.71	-5.06	0.97
UCB*	-0.40	41.9	-0.79	9.67	-1.64	1.86	-5.54	0.94
Exp3.P*	-0.36	47.9	-0.77	10.3	-1.66	1.79	-5.42	0.92
ϵ**-greedy***	-0.37	45.9	-0.88	7.04	-1.78	1.57	-5.45	**0.88**
Tasting 1.Q	-0.43	36.1	-0.96	5.38	-1.91	1.41	-5.90	0.92
Tasting Q.1	-0.44	34.8	-0.97	5.31	-1.91	1.36	**-5.91**	0.94
M.A.S. Naive	-0.51	26.3	-1.01	4.78	-1.80	1.54	-5.06	0.96
M.A.S. 1.Q	-0.48	29.9	-0.98	5.21	-1.74	1.63	-4.15	1.04
M.A.S. Q.1	-0.43	35.7	-0.98	5.21	-1.78	1.68	-4.51	1.01
Laminating	**-0.55**	**21.9**	**-1.10**	**3.85**	**-2.00**	**1.35**	-5.87	0.96

Table 1: Mean boosting loss ($\log_{10}$) and test error (%) after various number of boosting steps on the MNIST data set with all families of features. Methods highlighted with a $\star$ require the tuning of meta-parameters, which have been optimized by training fully multiple times.

INRIA								
Methods	T (# boosting steps)							
	10		100		1,000		10,000	
	loss	test error	loss	test error	loss	test error	loss	test error
Best family*	-0.34	12.2	-0.93	3.29	-3.72	1.20	-26.9	1.00
Uniform Naive	-0.31	13.4	-0.86	4.87	-3.92	1.27	-31.9	0.53
Uniform 1.Q	-0.30	14.2	-0.95	3.99	-4.26	0.89	-34.3	0.37
Uniform Q.1	-0.30	14.0	-1.01	3.92	-4.86	0.69	-40.0	0.33
UCB*	-0.35	12.1	-1.08	3.17	-5.47	0.61	-49.3	0.30
Exp3.P*	-0.31	13.6	-0.91	4.09	-4.53	0.79	-44.7	0.32
ϵ**-greedy***	-0.34	12.9	-1.11	2.89	-5.92	0.54	-49.3	0.34
Tasting 1.Q	-0.39	10.9	-1.30	2.14	-6.44	0.49	-54.1	0.30
Tasting Q.1	-0.40	11.2	-1.30	2.33	-6.54	0.57	-55.1	0.32
M.A.S. Naive	-0.46	8.80	-1.50	1.66	-7.23	0.44	-60.4	**0.27**
M.A.S. 1.Q	-0.41	10.1	-1.45	1.82	-6.87	0.50	-55.9	0.28
M.A.S. Q.1	-0.44	9.43	-1.51	1.65	-6.97	0.42	-57.1	**0.27**
Laminating	**-0.56**	**6.85**	**-2.05**	**1.12**	**-11.2**	**0.39**	**-99.8**	0.30

Table 2: Mean boosting loss ($\log_{10}$) and test error (%) after various number of boosting steps on the INRIA data set with all families of features. Methods highlighted with a $\star$ require the tuning of meta-parameters, which have been optimized by training fully multiple times.

Caltech 101								
Methods	T (# boosting steps)							
	10		100		1,000		10,000	
	loss	test error	loss	test error	loss	test error	loss	test error
Best family*	-0.80	95.2	-1.44	79.4	-7.17	56.7	-65.5	41.9
Uniform Naive	-0.79	95.8	-1.40	80.3	-6.81	55.6	-61.8	38.8
Uniform 1.Q	-0.79	95.9	-1.36	79.0	-5.84	54.2	-49.6	40.8
Uniform Q.1	-0.81	94.2	-1.44	76.5	-6.74	51.8	-59.2	37.6
UCB*	-0.81	94.2	-1.40	78.6	-6.46	52.6	-61.6	37.0
Exp3.P*	-0.79	95.8	-1.34	80.3	-5.89	54.7	-54.4	40.6
ϵ**-greedy***	-0.81	94.8	-1.42	76.7	-7.26	50.6	-67.1	37.4
Tasting 1.Q	**-0.82**	**93.8**	**-1.50**	**74.2**	**-7.47**	50.7	**-68.1**	**35.3**
Tasting Q.1	**-0.82**	93.9	**-1.50**	74.5	-7.46	**50.5**	**-68.1**	35.5
M.A.S. Naive	-0.80	94.3	-1.43	76.2	-6.70	51.8	-59.1	37.9
M.A.S. 1.Q	-0.78	96.4	-1.01	90.5	-2.04	85.9	-29.5	53.6
M.A.S. Q.1	-0.79	95.8	-1.21	85.6	-5.01	58.7	-42.7	44.5
Laminating	-0.81	94.3	-1.43	77.0	-6.33	53.0	-54.4	38.4

Table 3: Mean boosting loss ($\log_{10}$) and test error (%) after various number of boosting steps on the Caltech 101 data set with all families of features. Methods highlighted with a $\star$ require the tuning of meta-parameters, which have been optimized by training fully multiple times.

CIFAR-10								
Methods	T (# boosting steps)							
	10		100		1,000		10,000	
	loss	test error	loss	test error	loss	test error	loss	test error
Best family	-0.27	73.6	-0.33	57.4	-0.43	44.8	-0.67	40.2
Uniform Naive	-0.26	74.9	-0.34	55.9	-0.48	38.9	-0.93	32.2
Uniform 1.Q	-0.26	76.6	-0.33	57.5	-0.47	39.9	-0.84	31.3
Uniform Q.1	-0.27	74.3	-0.34	53.8	-0.49	37.6	-0.91	30.9
UCB*	-0.27	73.3	-0.34	56.2	-0.49	37.7	-0.90	30.6
Exp3.P*	-0.26	77.2	-0.33	58.0	-0.47	38.9	-0.86	30.3
ϵ-greedy*	-0.26	75.8	-0.35	53.4	-0.49	37.09	-0.88	**30.0**
Tasting 1.Q	-0.28	72.6	-0.36	50.9	**-0.50**	**36.2**	**-0.95**	31.7
Tasting Q.1	-0.28	71.8	-0.36	50.9	**-0.50**	36.3	**-0.95**	31.5
M.A.S. Naive	-0.28	71.9	-0.35	52.5	-0.49	37.5	-0.91	31.0
M.A.S. 1.Q	-0.28	70.7	-0.35	53.3	-0.45	40.5	-0.63	33.8
M.A.S. Q.1	-0.28	71.4	-0.35	52.7	-0.45	40.4	-0.62	34.1
Laminating	**-0.29**	**67.8**	**-0.37**	**49.1**	**-0.50**	36.8	-0.88	31.5

Table 4: Mean boosting loss ($\log_{10}$) and test error (%) after various number of boosting steps on the CIFAR-10 data set with all families of features. Methods highlighted with a $\star$ require the tuning of meta-parameters, which have been optimized by training fully multiple times.

up to 10,000 boosting rounds averaged through ten randomized runs in Tables 1—4. We used as cost for all the algorithms the number of evaluated features, that is for each boosting iteration QS, the number of sampled features times the number of sampled examples. For the Laminating algorithm we multiplied this cost by the number of iterations $\lceil \log_2(Q) \rceil$. We set the maximum cost of all the algorithms to $10N$, setting $Q = 10$ and $S = N$ for the baselines, as this configuration leads to the best results after 10,000 boosting rounds.

The parameters of the baselines—namely the scale of the rewards for UCB and Exp3.P, and the c/d^2 ratio of ϵ-greedy—were optimized by trying all values of the form $2^n, n = \{0, 1, ..., 11\}$, and keeping the one leading to the smallest final boosting loss on the training set, which is unfair to the uniform baselines as well as our methods. We set the values of the parameters of Exp3.P to $\eta = 0.3$ and $\lambda = 0.15$ as recommended in (Busa-Fekete and Kegl, 2010).

These results illustrate the efficiency of the proposed methods. Up to 1,000 boosting rounds, the Laminating algorithms is the clear winner on three out of the four data sets. Then come the M.A.S. and the Tasting procedures, still performing far better than the baselines. On the Caltech 101 data set the situation is different. Since it contains a much smaller number of training examples compared to the other data sets (1515 versus several tens of thousands), there is no advantage in sampling examples. It even proves detrimental as the M.A.S. and Laminating methods are beaten by the baselines after 1,000 iterations.

The performance of all the methods tends to get similar for 10,000 stumps, which is unusually large. The Tasting algorithm appears to fare the best, sampling examples offering no speed gain for such a large number of boosting steps, except on the INRIA data set. On this data set the Laminating algorithm still dominates, although its advantage in loss reduction does not translate into a lower test error anymore.

7. Conclusion

We have improved boosting by modeling the statistical behavior of the weak learners' edges. This allowed us to maximize the loss reduction under strict control of the computational cost. Experiments demonstrate that the algorithms perform well on real-world pattern recognition tasks.

Extensions of the proposed methods could be investigated along two axes. The first one is to merge the best two methods by adding a Tasting component to the Laminating procedure, in order to bias the sampling towards promising feature subsets. The second is to add a bandit-like component to the methods by adding a variance term related to the lack of samples, and their obsolescence in the boosting process. This would account for the degrading density estimation when subsets have not been sampled for a while, and induce an exploratory sampling which may be missing in the current algorithms.

Acknowledgments

We are grateful to Gilles Blanchard for suggesting an improvement of Theorem 1 to make the bound independent of Q.

Charles Dubout was supported by the Swiss National Science Foundation under grant 200021-124822 — VELASH, and François Fleuret was supported in part by the European Community's 7th Framework Programme under grant agreement 247022 — MASH.

Appendix A

Proof of Lemma 1:

Since

$$\max_{r:\epsilon_r \geq \epsilon_{q^\star} - \delta} \hat{\epsilon}_r \geq \hat{\epsilon}_{q^\star} \tag{7}$$

Defining $B_q = \mathbf{1}_{\{\Delta_q - \Delta_{q^\star} \geq \delta\}}$, we have

$$\begin{aligned}
&\mathrm{P}\left(\left|\left\{q:\ \epsilon_q \leq \epsilon_{q^\star} - \delta,\ \hat{\epsilon}_q \geq \max_{r:\epsilon_r \geq \epsilon_{q^\star} - \delta} \hat{\epsilon}_r\right\}\right| \ \geq\ \frac{Q}{2}\right) \\
\leq &\mathrm{P}\left(\left|\{q:\ \epsilon_q \leq \epsilon_{q^\star} - \delta,\ \hat{\epsilon}_q \geq \hat{\epsilon}_{q^\star}\}\right| \ \geq\ \frac{Q}{2}\right) && (8)\\
\leq &\mathrm{P}\left(\left|\{q : \epsilon_q \leq \epsilon_{q^\star} - \delta, \Delta_q - \Delta_{q^\star} \geq \delta\}\right| \geq \frac{Q}{2}\right) && (9)\\
\leq &\mathrm{P}\left(\sum_{q,\, \epsilon_q \leq \epsilon_{q^\star} - \delta} B_q \geq \frac{Q}{2}\right) && (10)\\
\leq &\mathrm{P}\left(\sum_q B_q \geq \frac{Q}{2}\right) && (11)\\
\leq &\mathrm{P}\left(\frac{2\sum_q B_q}{Q} \geq 1\right) && (12)\\
\leq &\frac{2\,\mathrm{E}\left[\sum_q B_q\right]}{Q} && (13)\\
\leq &2\,\mathrm{E}[B_q] && (14)\\
\leq &2\,\mathrm{E}\left[\mathbf{1}_{\{(\Delta_q \geq \frac{\delta}{2}) \cup (\Delta_{q^\star} \leq -\frac{\delta}{2})\}}\right] && (15)\\
\leq &4\exp\left(-\frac{\delta^2 S}{2}\right). && (16)
\end{aligned}$$

■

Equation (7) is true since $q^\star$ is among the $\{r : \epsilon_r \geq \epsilon_{q^\star} - \delta\}$ and δ is positive. Equations (8) to (12) are true since we relax conditions on the event. Equation (13) is true since $\mathrm{P}(X \geq 1) \leq \mathrm{E}(X)$ for $X \geq 0$. Equations (14) and (15) are true analytically, and equation (16) follows from Hœffding's inequality.

Appendix B

Proof of Theorem 1:

Defining $\delta_k = \frac{1}{C}\delta\sqrt{\frac{k}{2^{k-1}}}$ where $C = \sum_{k=1}^{\lceil \log_2(Q) \rceil} \sqrt{\frac{k}{2^{k-1}}}$ is a normalization constant such that the δ_k's sum to the original δ, i.e. $\sum_{k=1}^{\lceil \log_2(Q) \rceil} \delta_k = \delta$.

We apply Lemma 1 with constant δ_k for each of the k Laminating iterations, $1 \leq k \leq \lceil \log_2(Q) \rceil$. Since each iteration samples twice as many training examples as the previous one, and the δ_k's sum to the original δ, the probability to end up with a weak learner with an edge below or equal to $\epsilon_{q^*} - \delta$ is upper bounded by

$$
\begin{aligned}
& 4 \sum_{k=1}^{\lceil \log_2(Q) \rceil} \exp\left(-\frac{\delta_k^2 \, S \, 2^{k-1}}{2}\right) \\
\leq & 4 \sum_{k=1}^{\infty} \exp\left(-\frac{\delta^2 S k}{2C^2}\right) && (17) \\
\leq & 4 \left(\frac{1}{1 - \exp\left(-\frac{\delta^2 S}{2C^2}\right)} - 1\right) && (18) \\
\leq & 4 \left(\frac{1}{1 - \exp\left(-\frac{\delta^2 S}{69}\right)} - 1\right). && (19)
\end{aligned}
$$

■

Equation (17) is true analytically, Equation (18) follows from the formula for geometric series, and Equation (19) is true due to the fact that the constant C is upper bounded by $\sqrt{2}$ times the polylogarithm $\mathrm{Li}_{-\frac{1}{2}}\left(\frac{1}{\sqrt{2}}\right) = \sum_{k=1}^{\infty} \sqrt{\frac{k}{2^k}} \approx 4.15$.

References

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2003.

J. Bradley and R. Schapire. Filterboost: Regression and classication on large datasets. In *Neural Information Processing Systems*, 2007.

R. Busa-Fekete and B. Kegl. Accelerating AdaBoost using UCB. *JMLR W&CP*, 2009.

R. Busa-Fekete and B. Kegl. Fast Boosting using adversarial bandits. In *ICML*, 2010.

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Conference on Computer Vision and Pattern Recognition*, 2005. `http://pascal.inrialpes.fr/data/human/`.

C. Dubout and F. Fleuret. Tasting families of features for image classification. In *International Conference on Computer Vision*, 2011a.

C. Dubout and F. Fleuret. Boosting with maximum adaptive sampling. In *Neural Information Processing Systems*, 2011b.

N. Duffield, C. Lund, and M. Thorup. Priority sampling for estimation of arbitrary subset sums. *J. ACM*, 54, December 2007.

G. Escudero, L. Màrquez, and G. Rigau. Boosting applied to word sense disambiguation. *Machine Learning: ECML 2000*, pages 129–141, 2000.

L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories. In *CVPR, Workshop on Generative-Model Based Vision*, 2004. `http://www.vision.caltech.edu/Image_Datasets/Caltech101/`.

F. Fleuret and D. Geman. Stationary features and cat detection. *Journal of Machine Learning Research*, 9:2549–2578, 2008.

F. Fleuret, T. Li, C. Dubout, E. K. Wampler, S. Yantis, and D. Geman. Comparing machines and humans on a visual categorization test. *Proceedings of the National Academy of Sciences*, 108(43):17621–17625, 2011.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann, 1996.

P. Gehler and S. Nowozin. On feature combination for multiclass object classification. In *International Conference on Computer Vision*, 2009.

Z. Kalal, J. Matas, and K. Mikolajczyk. Weighted sampling for large-scale Boosting. *British machine vision conference*, 2008.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. `http://www.cs.toronto.edu/~kriz/cifar.html`.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86(11), pages 2278–2324, 1998. `http://yann.lecun.com/exdb/mnist/`.

A. Opelt, A. Pinz, M. Fussenegger, and P. Auer. Generic object recognition with Boosting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:416–431, 2006.

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