Tasting Families of Features for Image Classification

Charles Dubout and François Fleuret
Idiap Research Institute
{charles.dubout,francois.fleuret}@idiap.ch

Abstract

Using multiple families of image features is a very efficient strategy to improve performance in object detection or recognition. However, such a strategy induces multiple challenges for machine learning methods, both from a computational and a statistical perspective.

The main contribution of this paper is a novel feature sampling procedure dubbed "Tasting" to improve the efficiency of Boosting in such a context. Instead of sampling features in a uniform manner, Tasting continuously estimates the expected loss reduction for each family from a limited set of features sampled prior to the learning, and biases the sampling accordingly.

We evaluate the performance of this procedure with tens of families of features on four image classification and object detection data-sets. We show that Tasting, which does not require the tuning of any meta-parameter, outperforms systematically variants of uniform sampling and state-of-the-art approaches based on bandit strategies.

1. Introduction

Machine learning methods are frequently used in computer vision for pattern recognition problems that cannot be specified analytically. Detection of faces, pedestrians, vehicles, and recognition of objects or individuals, are almost exclusively done with techniques such as Support Vector Machines [6], Boosting [15], or forests of decision trees [3].

One of the crucial components of any machine learning approach to computer vision problems is the set of image features used as input to the learning core of the system. While machine learning is extremely good at weighting multiple cues optimally, it is weak at building invariant features. The design of measurements able to compensate for variations of the signal due to changes in illumination or geometrical pose is still done by human experts.

It has been demonstrated repeatedly that combining multiple families of features addressing different aspects of the signal is an extremely efficient strategy to improve performance [14, 11]. As shown by our experimental results in § 5, vanilla Boosting of stumps over multiple features reaches state-of-the-art performance. However, such techniques induce two major practical difficulties: the first is the computational cost of the training, which increases linearly with the number of features, and the second is overfitting the training data. Both are related to the number of features which are actually "looked at" during training.

We propose here a straight-forward and original strategy dubbed "Tasting" to deal with that situation, and use it to improve the loss reduction in Boosting. This method samples a few features from every family before the training *per se*, and stores their responses over each training sample. During Boosting, every time we have to sample features to minimize a weighted error, we use these stored features to get an estimate of the expected reduction of the loss for each family, and sample accordingly.

Experiments on four image classification and object detection problems show that Tasting systematically outperforms sophisticated baselines in minimizing both the training loss and the test error, without requiring the tuning of any parameter, contrarily to the most advanced baselines. This makes it the best practical solution to deal with multiple families of features in a context where there are too many features to evaluate and store at once.

We present an overview of existing methods which address the issue of multiple families of features in § 2. Then, in § 3, we formalize Boosting with feature sampling, and describe several baselines. We introduce our own approaches in § 4, and provide experimental results in § 5.

2. Related works

There are essentially two ways to accelerate the training of Boosting algorithms. One is to reduce the number of samples used to train the weak learner at every iteration. Several methods of this kind can be found in [12]. The second is to reduce instead the number of features examined at every Boosting step. The first popular method of this kind is LazyBoosting, which is very simple and efficient in practice [9, 5]. The only difference between this method and AdaBoost is to train the weak learner over a random subset of the features, instead of all of them at every iteration.

While the sampling strategy does not matter much in the usual case, it becomes central when dealing with multiple families of features, and how to do it properly is not clear. This is the subject of section § 3.2.

More recently, several algorithms have been proposed to improve over LazyBoosting [4, 5]. They bias the sampling toward features which have shown to be promising in the past, and use a bandit approach to handle properly the dilemma between exploiting features already known to be good, and exploring the space of features. We describe a few state-of-the-art methods of that kind that we use as baselines in § 3.3.

Bandit-related approaches share similarities with the techniques we introduce in this article, but suffer from several weaknesses. The first one is that they estimate the quality of a feature by making an assumption of stationarity of the loss reduction distribution associated to it, ignoring that it depends on the sample weights. With an aggressive loss such as the exponential one, the weights vary significantly during training, and often become extremely unbalanced.

The second weakness is the application context. Without additional prior knowledge on the features, the only structure one can exploit is a form of stationarity of individual features. Hence, improvement can be achieved only by sampling again *the exact same feature* one has already seen as promising in the past. This pushes toward reusing the same features multiple times, which reduces the information conveyed by the set eventually selected.

We therefore advocate for methods modeling the distribution of the feature responses, from which they compute an empirical distribution of the loss reduction, instead of directly modeling the latter. Also, we apply our approach to a context where the full set of features is the union of multiple families of features addressing different modalities. This allows to model the response, and bias the sampling at the level of sub-families instead of individual features.

3. Notation and Baselines

After introducing our notation, we present in this section a general formulation of the feature sampling in § 3.1, and present several baselines in § 3.2 and § 3.3. For the sake of clarity we provide here the notation and formalization for the Binary case only. However, we used in our experiment AdaBoost.MH [15], which can handle multiple labels.

Let $\mathcal{X}$ stand for the space of images, and

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

a training set of images, with their labels. Let $\mathcal{F}(\mathcal{X}, \mathbb{R})$ be the set of mappings from $\mathcal{X}$ into $\mathbb{R}$, and

$$\mathfrak{F}_k \subset \mathcal{F}(\mathcal{X}, \mathbb{R}),\ k = 1, \dots, K$$

the K families of feature extractors.

$\mathbf{1}_{\{\text{condition}\}}$	is equal to 1 if the condition is true, 0 otherwise
N	Number of training samples
K	Number of families of features
Q	Number of features sampled at every Boosting step
R	Number of features sampled initially for tasting
$\mathcal{X}$	Space of images
$\mathfrak{F}_k$	Family of features
$\mathcal{U}(\mathfrak{F})$	Uniform distribution on $\mathfrak{F}$
$(x_n, y_n) \in \mathcal{X} \times \mathbb{N}$	training samples
$\omega^s \in \mathbb{R}^N$	Weights of the samples at iteration s of boosting
$S(f, \rho; \omega)$	Score of the pair (f, ρ), equal to the absolute value of the derivative of the loss with respect to the weight of the stump defined by f and ρ. This is the quantity Boosting tries to maximize at every step.
$S^*(f; \omega)$	Score of the feature f, equal to $\max_\rho S(f, \rho; \omega)$

Table 1. Notation

Let $\omega^s \in \mathbb{R}^N$ stand for the signed weights of the samples, or more precisely their edges: ω_n^s is the derivative of the loss with respect to the response of the classifier on sample n at step s of the learning. For instance, with the standard exponential loss

$$L(\varphi) = \sum_n \exp(-y_n \varphi(x_n)),$$

if φ_s denotes the strong classifier built at iteration s, then we have

$$\omega_n^s = -y_n \exp(-y_n\, \varphi_s(x_n)).$$

We consider Boosting stumps, which are weak learners of the form

$$2 \cdot \mathbf{1}_{\{f(x) \geq \rho\}} - 1.$$

This means that at every iteration s, Boosting tries to pick a feature f_s and a threshold ρ_s which maximize a score equal to the derivative of the loss reduction $S(f_s, \rho_s;\, \omega_s)$ with respect to the weak-learner weight, which can be expressed as

$$S(f, \rho;\, \omega) = \left| \sum_n \omega_n \left(2 \cdot \mathbf{1}_{\{f(x_n) \geq \rho\}} - 1\right) \right|. \qquad (1)$$

From this quantitiy, we also define the score S^* of a feature f, equal to the best score achievable with f by optimizing the threshold ρ:

$$S^*(f;\, \omega) = \max_\rho\ S(f, \rho; \omega). \qquad (2)$$

Finally, for any k, if F is a random variable uniform on $\mathfrak{F}_k$, we define μ_k^ω as the distribution of $S^*(F; \omega)$. This is the distribution of the (derivative of the) loss reduction when one picks f uniformly at random in $\mathfrak{F}_k$ and then optimizes ρ according to ω.

3.1. Feature sampling

As stated above, a Boosting procedure would ideally compute at every step, given the current weights ω of the samples,

$$\underset{(f,\rho)\in\cup_k \mathfrak{F}_k\times\mathbb{R}}{\operatorname{argmax}} \quad S(f,\rho;\omega). \tag{3}$$

Unfortunately, in most of the real-world problems, particularly in computer vision, one has to cope with very large feature sets, sometimes of infinite cardinality, and (3) cannot be computed explicitly.

A powerful strategy is to approximate it by sampling features: instead of looking exhaustively at $\cup_k \mathfrak{F}_k$, one can sample Q features in this set, and pick the best among them according to S^*.

3.2. Uniform sampling baselines

A naive strategy would pick these Q features uniformly in $\cup_k \mathfrak{F}_k$. However, this does not distribute the sampling properly among the $\mathfrak{F}_k$. In the extreme case, if one of the $\mathfrak{F}_k$ had a far greater cardinality than the others, all features would come from it. And in most of the contexts, mixing features from the different $\mathfrak{F}_k$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 family** picks Q features at random in a fixed family, the one with the smallest final Boosting loss.
- **Uniform Naive** picks Q features at random, uniformly in $\cup_k \mathfrak{F}_q$.
- **Uniform 1.Q** picks one of the feature families at random, and then samples the Q features from that single family.
- **Uniform Q.1** picks at random, uniformly, Q families of features (with replacement if $Q > K$), and then picks one feature uniformly in each family.

The cost of running **Best family** is K times higher than running the other three strategies. Also, as it makes use of one family only, we can expect its final performance to be lower than the others. It was included for comparison only.

3.3. 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 [4], and extended in [5]. The driving idea of these papers is to entrust a multi-armed bandits (MAB) algorithm (respectively **UCB** [1] and **Exp3.P** [2]) with the mission to sample useful features.

The multi-armed bandits problem is defined as follows: there are M gambling machines (i.e. 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 sample weights, and thus can only rely on the history of past rewards.

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

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 algorithm will not learn anything, as it expects rewards to make full use of the range $[0,1]$.

For this reason we also used a third multi-armed bandit algorithm in our experiments, **ϵ-greedy** [1], which does not suffer from that problem.

Hence, we use in our experiments the three following baselines, using the same reward as in [5]:

- **UCB** picks Q features from the family that maximizes $\bar{r}_j + \sqrt{(2\log n)/n_j}$, where $\bar{r}_j$ is the current average reward of family j, n_j is the number of times family j was chosen so far, and n is the current Boosting round.
- **Epx3.P** maintains a distribution of weights over the families of features, and at every round picks one family accordingly, obtains a reward, and updates the distribution. For the precise definition of the algorithm, see [2, 5].
- **ϵ-greedy** picks Q features from the family with the highest current average reward with probability $1-\epsilon_n$, or from a random family with probability ϵ_n, where $\epsilon_n = cK/(d^2 n)$, and c and d are parameters of the algorithm.

4. Feature Tasting

We describe here our approach called Tasting which biases the feature sampling toward promising families of features. This is achieved by sampling a small number R of features from each family before starting the training *per se*, and at every Boosting step, in using these few features with the current sample weights to get an estimate of the best

family of $\mathcal{F}_k$ to use. We cannot stress enough that those features are not the ones used to build the classifier, they are only used to get an estimate of the best family. 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.

4.1. Description

As for bandit-related methods, Tasting exploits the fact that the full feature set is a heterogeneous union of somehow homogeneous subsets of features. Given ω, the distribution of the loss reduction associated to one specific family of features (i.e. the distribution μ_k^ω introduced in § 3) has lower variance than if the feature was picked uniformly over the full set $\cup_k \mathfrak{F}_k$.

In practice, this means that if a few features picked at random in $\mathfrak{F}_k$ perform poorly for the current ω, it is likely that sampling from that family is not a good choice. Note that this is a more realistic assumption than the one justifying the bandit approaches described above, since it takes into account the value of ω.

Since the number R of features required for such an estimate is far smaller than the full cardinality of the feature families (experiments with $R = 10$ and $R = 100$ give roughly the same results, see § 5.3), they can be stored in memory. Hence, doing so avoids the requirement of running the full pre-computations and computations of every family at every Boosting step, except for the families actually sampled. Also, the samples can be pre-sorted according to each one of the R pre-sampled features, which speeds up the selection of the optimal stumps.

Let

$$f_k^r \in \mathfrak{F}_k,\ r = 1, \dots, R,\ k = 1, \dots, K$$

be the features whose responses are stored before starting the Boosting iterations. Then, $\forall \omega \in \mathbb{R}^N$, we have,

$$\hat{E}_{F \sim \mathcal{U}(\mathfrak{F}_k)}(S^*(F; \omega)) = \frac{1}{R} \sum_{r=1}^{R} S^*(f_k^r; \omega).$$

and given any $\sigma \in \mathbb{R}$, we have

$$\hat{E}_{F \sim \mathcal{U}(\mathfrak{F}_k)}(\max(\sigma, S^*(F; \omega))) = \frac{1}{R} \sum_{r=1}^{R} \max(\sigma, S^*(f_k^r; \omega)).$$

The latter quantity stands for (the expectation of) what we eventually achieve by picking a feature at random in $\mathfrak{F}_k$, that is, the maximum between the score σ already achieved and the score of the sampled feature.

From this, we define two strategies to sample $F_1, \dots, F_Q$, which correspond to the two variants of the uniform sampling:

- **Tasting 1.Q** computes the optimal family at the beginning of every Boosting step, and then samples Q features uniformly from it:

  $k \leftarrow \text{argmax}_k \hat{E}_{F_1, \dots, F_Q \sim \mathcal{U}(\mathfrak{F}_k)}(\max_q(S^*(F_q; \omega^s)))$
  for $q = 1, \dots, Q$ **do**
  $F_q \leftarrow \text{SampleUniformly}(\mathfrak{F}_k)$,
  end for

  Numerically, given $F_1, \dots, F_Q$ independent, and identically distributed, uniform over $\{1, \dots, R\}$. And $v_1 \leq v_2 \leq \dots \leq v_R$, we use:

  $$\begin{aligned}
  & E\left(\max_{q=1}^{Q} v_{F_q}\right) \\
  &= \sum_{r=1}^{R} P\left(\max_{q=1}^{Q} F_q = r\right) v_r \\
  &= \sum_{r=1}^{R} \left(P\left(\max_{q=1}^{Q} F_q \leq r\right) - P\left(\max_{q=1}^{Q} F_q \leq r-1\right)\right) v_r \\
  &= \frac{1}{R^Q} \sum_{r=1}^{R} \left(r^Q - (r-1)^Q\right) v_r \\
  &= v_R + \frac{1}{R^Q} \sum_{r=1}^{R-1} r^Q (v_r - v_{r+1})
  \end{aligned}$$

- **Tasting Q.1** selects the optimal family for every one of the Q features to sample, given the best score σ_q achieved so far in this iteration. With the convention that $\max(\emptyset) = 0$, we have:

  for $q = 1, \dots, Q$ **do**
  $\sigma_q \leftarrow \max(S^*(F_1; \omega^s), \dots, S^*(F_{q-1}; \omega^s))$
  $k_q \leftarrow \text{argmax}_k \hat{E}_{F \sim \mathcal{U}(\mathfrak{F}_k)}(\max(\sigma_q, S^*(F; \omega^s)))$
  $F_q \leftarrow \text{SampleUniformly}(\mathfrak{F}_{k_q})$
  end for

4.2. Analysis

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 weak learners for a face detector may use color-based features to exploit the skin color. After color has been optimally exploited, only samples with a non-standard face color would have large weights, and other feature, for instance edge-based, would become more informative, and be picked by Boosting.

Uniform sampling and bandit-methods account poorly for such behavior at the family level. Uniform sampling simply discards the sample weights, hence has no information whatsoever about the directions which have "already

Figure 1. 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.

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 are slightly more adequate, as they model a score for each family from previous iterations. However, this modeling takes into account the weighting of the samples very indirectly, as they make the assumption that the distribution 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, because it was in the previous iterations, even if the sample 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, as the weight of the samples do not fluctuate much, it is a severe weakness when they are still evolving.

Tasting addresses this weakness by keeping the ability to properly estimate the value of each family, *given the current sample weights*, hence the ability to discard families of features redundant with features already chosen. In some sense, Tasting can be seen as a Boosting at the family level.

5. Experiments

We demonstrate the effectiveness of our approach on four standard image classification and object detection data-sets, using nineteen to thirty-three families of features. We compare the Tasting approach with the baselines of § 3.

The implementations of both the families of features, and of the feature sampling and Boosting algorithms will be published under the GPL v2.0 open-source license at the time of publication of this article. Together with the freely available data-sets used for the experiments, this will ensure the reproducibility of the results presented here.

5.1. Feature families

The feature families used in our experiments on 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 families from the first category typically have a large number of features, 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).

Families from the second and third categories being histograms, they usually contain much fewer features (typically of the order of a few hundreds to a few thousands), but require some pre-processing to build the histograms.

For the Caltech 101 data-set we used the same features as [11] in their experiments. They used five type of features (PHOG Shape descriptors, Appearance (SIFT) descriptors, Region Covariance, Local Binary Patterns, V1S+ (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 family, leading to a total of 39 feature families. The number of features families used in our experiments (33) differ from [11] as they also compute a 'subwindow-kernel' of SIFT features that we did not use.

Final Boosting loss ($\log_{10}$)							
Methods	Data-sets						
	MNIST	INRIA Person		Caltech 101		CIFAR-10	
	$Q = 10$	$Q = 10$	$Q = 100$	$Q = 10$	$Q = 100$	$Q = 10$	$Q = 100$
Best family*	-4.911 / 0.005	-9.05 / 0.11	-12.97 / 0.07	-52.36 / 0.27	-57.80 / 0.31	-0.6784 / 0.0004	-0.6935 / 0.0005
Uniform Naive	-5.319 / 0.013	-10.00 / 0.17	-21.14 / 0.14	-50.21 / 0.26	-57.44 / 0.30	-0.9305 / 0.0006	-1.0137 / 0.0012
Uniform 1.Q	-3.802 / 0.021	-10.96 / 0.14	-19.04 / 0.21	-43.13 / 0.41	-48.21 / 0.39	-0.8485 / 0.0012	-0.9096 / 0.0010
Uniform Q.1	-5.038 / 0.014	-12.52 / 0.22	-24.42 / 0.15	-48.23 / 0.34	-55.40 / 0.24	-0.9142 / 0.0011	-1.0058 / 0.0011
UCB*	-5.261 / 0.012	-14.11 / 0.21	-26.14 / 0.19	-48.21 / 0.35	-53.82 / 0.31	-0.9007 / 0.0007	-0.9737 / 0.0016
Exp3.P*	-5.350 / 0.045	-13.30 / 0.41	-23.84 / 0.41	-46.65 / 0.66	-52.24 / 0.51	-0.9080 / 0.0040	-0.9850 / 0.0040
ϵ**-greedy***	-5.380 / 0.092	-15.41 / 0.22	-27.14 / 0.22	**-53.24 / 0.32**	**-58.91 / 0.32**	-0.9054 / 0.0046	-0.9724 / 0.0037
Tasting 1.Q	**-5.968 / 0.022**	**-17.01 / 0.31**	**-28.54** / 0.56	-52.70 / 0.59	-58.05 / 0.67	**-0.9399 / 0.0090**	**-1.0230 / 0.0008**
Tasting Q.1	**-5.970 / 0.014**	**-17.01 / 0.40**	**-28.63** / 0.37	-52.88 / 0.64	-57.35 / 0.79	**-0.9400 / 0.0059**	**-1.0231 / 0.0012**

Table 2. Mean and standard deviation of the final Boosting loss ($\log_{10}$) on the four data-sets for each method with two different values of Q. Methods highlighted with a $\star$ require the tuning of meta-parameters which have been optimized by training fully multiple times.

5.2. Data-Sets

The first data-set that we used is the MNIST handwritten digits database [13]. It contains 10 classes and its training and testing sets consist respectively of 60,000 and 10,000 grayscale images of resolution 28×28 pixels. The total number of features on this data-set is 16,451.

The second data-set that we used is the INRIA Person data-set [8]. 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. We extracted 10 negative samples from each one of the background image, following the setup of [8]. The total number of features on this data-set is 230,503.

The third data-set that we used is Caltech 101 [10] due to its wide usage and the availability of already computed features [11]. It consists of color images of various dimensions organized in 102 visual classes (101 objects plus an additional background class). 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 [7]. It is a labeled subset of the 80 tiny million images data-set. It contains 10 classes and its training and testing sets consist respectively of 50,000 and 10,000 color images of size 32×32 pixels. The total number of features on this data-set is 29,879.

5.3. Results

We tested the proposed Tasting methods of § 4, against the baselines described in § 3.2 and § 3.3 on the four benchmark data-sets described above in § 5.2 using the standard train/test cuts. We report the results of doing 10,000 Boosting rounds using AdaBoost.MH on all data-sets but INRIA Person, for which we report only 3,000, having observed nearly no gain in doing more. All experiments were averaged through ten randomized runs.

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. We set the values of the parameters of **Exp3.P** to $\eta = 0.3$ and $\lambda = 0.15$ as recommended in [5].

Our methods require only one parameter to be set, the number R of features to initially store from each family. We used the value $R = 100$ in all our experiments, but we observed that setting it to 10 only marginally affects them, increasing the test error by less than 0.02% on average, and reducing the (logarithm of) the loss by less than 3%.

Table 2 shows the final Boosting loss after 10,000 iterations (3,000 for INRIA Person), while figure 2 shows some of the test errors during training. The results we obtain are close to state-of-the-art. On the MNIST data-set, we obtain 0.85% classification error. The best results on this base (0.39%) are achieved with methods tuned for character recognition, and training with synthetic deformations or unsupervised learning with additional data. We are not aware of a generic and non-tuned learning procedure that outperforms ours.

On the INRIA pedestrian data-set, we achieve 0.30% classification error rate, again without any form of tuning or bootstrapping. Taking the best point on the ROC curves for six methods designed for pedestrian detection [16], we get with our number of negative samples: $(0.01 \cdot 1126 + 0.001 \cdot 4530)/(1126 + 4530) = 0.28\%$.

On Caltech 101 we get 35% classification error using 15 training examples, second among the eight methods reported in [11]. Finally on CIFAR-10 our results (30% classification error) are the best among the ones listed in [7] (35% classification error at best, ignoring the ones obtained

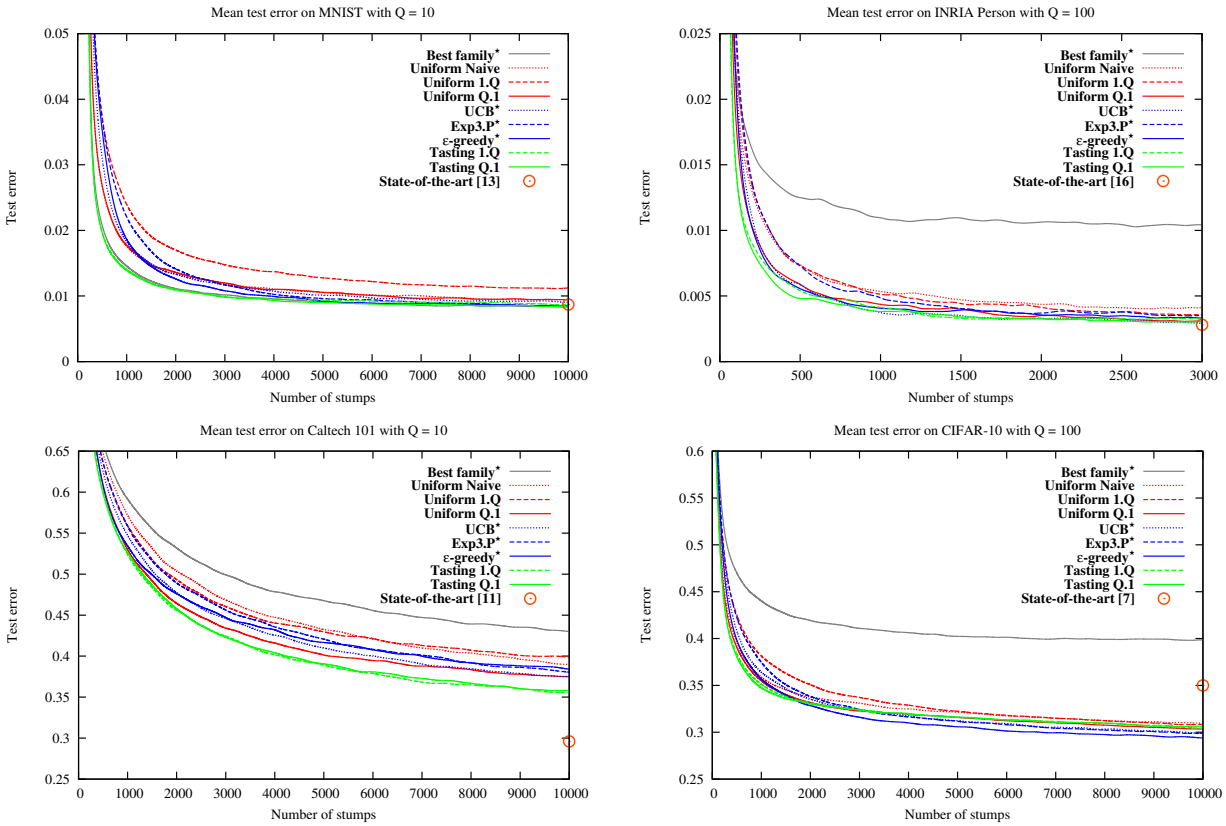

Figure 2. Mean test error on the four data-sets for each method with two different values of Q. The two versions of Tasting provide very similar test errors on the four data-sets, are the best on three, and perform well on the fourth. State-of-the-art results do not include methods using additional training data.

by pre-training with massive amounts of additional data).

The two methods presented in this paper, **Tasting 1.Q** and **Tasting Q.1** obtain virtually identical results, and always lead to the fastest loss reduction, at the exception of the Caltech 101 data-set, where they rank second after the ϵ**-greedy** method. They also obtain the best or close to the best test errors, with remarkable stability. The methods managing to marginally beat them only do it after a large number of iterations, and are beaten by a large margin in some other conditions. Beside, the meta-parameters of the bandit baselines were optimized by running the training fully twelve times. The good performance of the **Uniform** methods can be attributed to the fact that most of the features that we used are relevant to the problem, but their performance can become arbitrarily bad in the presence of uninformative feature families.

Using two standard deviations as a significant delta, out of the 49 comparisons (7 baselines and 7 setups): with 1000 stumps, **Tasting 1.Q** is significantly better for loss reduction in 42 comparisons, and worst in 0 (resp. 25 and 0 for the test error). With 10,000 stumps (3,000 for INRIA), **Tasting 1.Q** is significantly better for loss reduction in 45 comparisons, and significantly worst in 0 (resp. 11 and 7 for the test error).

Table 3 shows the proportions of each family of features used in the classifier on the pedestrian task, when built using the different sampling strategies. The Tasting procedures concentrate the sampling over a restricted sub-set of feature families. While it contains only 5 families of features out of 19, and 19,300 features out of 230,503, the third group of feature families constitutes 69% of the features in the classifier with **Tasting 1.Q** and 61% with **Tasting Q.1**.

6. Conclusion

The Tasting approach presented in this paper is extremely straight-forward and avoids the need for setting multiple parameters, in particular to control the trade-off between exploration and exploitation. In practice, the only parameter to set is the number of features R sampled initially to estimate the expected loss reduction during training. As stated in § 5.3, the process is remarkably stable to changes of that value.

The distribution required to select a good family of fea-

INRIA Person (Q = 10)								
Best	**Uniform**			**Bandits**			**Tasting**	
family	**Naive**	**1.Q**	**Q.1**	**UCB**	**Exp3.P**	**ϵ-greedy**	**1.Q**	**Q.1**
-	4%	5%	3%	3%	3%	1%	-	-
-	12%	5%	3%	3%	4%	1%	-	-
-	1%	5%	1%	2%	2%	1%	-	-
-	2%	6%	2%	2%	3%	1%	-	-
-	9%	5%	6%	7%	7%	7%	6%	6%
-	9%	5%	6%	4%	5%	1%	1%	2%
-	27%	5%	10%	9%	9%	14%	12%	18%
-	6%	5%	3%	3%	4%	1%	-	-
-	6%	5%	4%	3%	3%	1%	1%	-
-	5%	5%	3%	3%	3%	1%	-	-
-	-	5%	2%	2%	2%	1%	-	1%
-	-	5%	3%	3%	3%	-	1%	1%
-	-	5%	6%	5%	5%	3%	3%	4%
-	-	5%	4%	6%	5%	6%	7%	5%
-	-	5%	5%	4%	4%	1%	3%	4%
-	2%	5%	10%	9%	8%	9%	9%	11%
-	2%	5%	11%	8%	10%	7%	8%	11%
-	7%	5%	9%	11%	9%	15%	23%	15%
100%	8%	5%	9%	13%	11%	31%	26%	20%

Table 3. Proportions of features from each family used by each method. Each row corresponds to one of the 19 families of features, and horizontal lines separate the three groups defined in § 5.1. For clarity, percentages have been rounded to their closest integral values, and zeros have been replaced by hyphens.

tures at every step is that of the loss reduction μ_k^ω, which is a distribution on $\mathbb{R}$. It is the projection of a distribution on $\mathbb{R}^N$, projection which is not known *a priori*. As described in § 4.1 and justified in § 4.2, Tasting relies on the ability to estimate the loss reduction given any weighting of the training samples. We have chosen to use an empirical model, that is to store actual responses over samples, instead of fitting an analytical density model. It may be possible to choose the later strategy, and summarize the information provided by feature responses for instance with a Gaussian model. However, it is not clear how such a model could lead to a proper estimate of the distribution of the loss reduction when ω is strongly unbalanced. A third option would be to use an analytical model of the loss reduction itself, estimated on-the-fly at every Boosting step, given ω. This may be a nice approach to dealing with long tails, or other behavior poorly reflected with a small sample set.

From a computational perspective, the methods we have proposed here have a cost composed of a dominant term proportional to the number of families actually used at every step, but still have a dominated term linear with the number K of feature families itself. We would like to deal eventually with hundreds or thousands of such families, for which the second term will be of importance. Such a context will require to handle properly the exploration/exploitation dilemma: The approximation of μ_k^ω could be done in an adaptive manner, both in term of computation and memory usage, by investing more resources on the families of features which have proved to be useful in the previous iterations. This would in practice add to Tasting a bandit-like component which is currently missing.

Acknowledgments

This work was supported by the European Community's 7th Framework Programme under grant agreement 247022 – MASH, and by the Swiss National Science Foundation under grant 200021-124822 – VELASH.

References

[1] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002. 3

[2] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2003. 3

[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. 1

[4] R. Busa-Fekete and B. Kegl. Accelerating AdaBoost using UCB. *JMLR W&CP*, Jan 2009. 2, 3

[5] R. Busa-Fekete and B. Kegl. Fast Boosting using adversarial bandits. In *ICML*, 2010. 1, 2, 3, 6

[6] N. Christiani and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000. 1

[7] The CIFAR-10 data-set. http://www.cs.toronto.edu/~kriz/cifar.html. 6

[8] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Conference on Computer Vision and Pattern Recognition*, 2005. 6

[9] G. Escudero, L. Màrquez, and G. Rigau. Boosting applied to word sense disambiguation. *Machine Learning: ECML 2000*, pages 129–141, 2000. 1

[10] 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. 6

[11] P. Gehler and S. Nowozin. On feature combination for multiclass object classification. In *International Conference on Computer Vision*, 2009. http://www.vision.ee.ethz.ch/~pgehler/projects/iccv09/. 1, 5, 6

[12] Z. Kalal, J. Matas, and K. Mikolajczyk. Weighted sampling for large-scale Boosting. *British machine vision conference*, 2008. 1

[13] The MNIST data-set. http://yann.lecun.com/exdb/mnist/. 6

[14] 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. 1

[15] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336, 1999. 1, 2

[16] W. Schwartz, A. Kembhavi, D. Harwood, and L. S. Davis. Human detection using partial least squares analysis. In *ICCV*, 2009. 6