Boosting Density Function Estimators

Franck Thollard, Marc Sebban, and Philippe Ezequel

EURISE, Department of Computer Science
Université Jean Monnet de Saint-Etienne
{franck.thollard,marc.sebban,ezequel}@univ-st-etienne.fr

Abstract. In this paper, we focus on the adaptation of boosting to density function estimation, useful in a number of fields including Natural Language Processing and Computational Biology. Previously, boosting has been used to optimize classification algorithms, improving generalization accuracy by combining many classifiers. The core of the boosting strategy, in the well-known Adaboost algorithm [4], consists in updating the learning instance distribution, increasing (resp. decreasing) the weight of misclassified (resp. correctly classified) examples by the current classifier. Except in [17, 18], few works have attempted to exploit interesting theoretical properties of boosting (such as *margin maximization*) independently of a classification task. In this paper, we do not take into account *classification errors* to optimize a classifier, but rather *density estimation errors* to optimize an estimator (here a probabilistic automaton) of a given target density. Experimental results are presented showing the interest of our approach.

1 Introduction

Most of the machine learning algorithms in supervised learning aim at providing efficient classification rules, often by optimizing the success rate of a given classifier. However, in some other machine learning areas, such as in Natural Language Processing, the main objective rather consists in correctly estimating probability densities over strings. In such a context, the algorithms aim at modelling a target density from a learning sample, in order to assess the occurrence probability of a new instance. This way to proceed is particularly useful in machine learning areas such as shallow parsing, spelling correction, speech recognition [7, 8, 12, 19] and computational biology [2, 10].

Many algorithms are available for estimating these densities from learning data: Hidden Markov Models [7], probabilistic automata [9, 13, 21], Markov Models and their smoothing techniques [5], etc. Recently, some work has dealt with density estimation by combining some of these models [3, 20]. In this paper, we also use such a strategy by introducing a boosting approach to density estimation. Although during the last decade many papers have shown the interest of voting classification algorithms (such as *boosting* or *bagging*) (see *e.g.* [1, 11, 14]), to the best of our knowledge this is the first attempt to use the optimization properties of boosting in such a context. However, we think that this way to proceed deserves further investigation. Thanks to the margin maximization principle, it not only would allow the optimisation of estimation performance, but also would avoid the tricky use of smoothing techniques, often crucial in density estimation.

By combining many weak hypotheses (usually a hypothesis is a classifier), the algorithm Adaboost [4] generates a relevant final weighted classifier. Recently, many theoretical results have justified the relevance of the boosting in machine learning [15]. But so far, its use has been limited to optimize classification tasks, despite recent original extensions to prototype selection [18] and feature selection [17].

Using boosting for estimating density functions is much more difficult than a *simple* classification optimization. It requires modification of the weight update rule, the core of the Adaboost algorithm, that we recall in section 2. Here, we do not aim to generate an efficient classifier which minimizes the rate of misclassified examples, but rather we aim to automatically and correctly estimate target density function. We show in section 3 that the final combined estimator must then minimize estimation errors (*i.e.* over and under estimations) regarding the original distribution. For the experimental study, a probabilistic automaton is used during the boosting step as a *weak estimator*. We present it in section 4. Finally, its performance is tested on a language modelling task, and results are presented in section 5.

2 Properties of Boosting

Boosting consists in combining many (T) *weak* hypotheses produced from various distributions $D_t(e)$ over the learning set (LS). The pseudo-code of the original boosting algorithm, called Adaboost [4], is described by the Algorithm 1. At the beginning of the process, each instance e is initially distributed according to an uniform density $D_1(e)$ (note that a given example can of course occur many times in LS, resulting in a higher density at this point). At each stage t, Adaboost decreases (resp. increases) the weight of the training instances, *a priori* labeled $y(e)$, correctly (resp. incorrectly) classified by the current weak hypothesis h_t. Boosting thus forces the weak learner to learn the hardest examples. The weighted combination $H(e)$ of all the weak hypotheses results in a better performing model. Schapire and Singer [16] proved that, in order to minimize the training error, one must seek to minimize Z_t (the normalization factor, *i.e.* the sum of the updates) on each round of boosting. It is easy to show that for minimizing the objective function Z_t, the confidence α_t of each weak hypothesis (used in the final combined classifier H) is $\frac{1}{2}\log(\frac{1-\epsilon_t}{\epsilon_t})$.

In order to introduce our boosting approach to density estimation, we recall here notations already proposed in Schapire and Singer [16]. Suppose that $y(e) \in \{-1, +1\}$ and that the range of each weak hypothesis h_t is restricted to $-1, 0, +1$.

Algorithm 1: Pseudo-code for AdaBoost.

```
Data   : A learning sample LS, a number of iterations T, a weak learner WL
Result : An aggregated classifier $H$
Initialize distribution: $\forall e \in LS,\ D_1(e) = \frac{1}{|LS|}$ ;
for $t = 2$ to $T$ do
    $h_t$ = WL ($LS$,$D_t$);
    $\epsilon_t = \sum_{e:y(e)\neq h_t(e)} D_t(e)$ ;
    $\alpha_t = \frac{1}{2}\log(\frac{1-\epsilon_t}{\epsilon_t})$ ;
    Distribution Update        /* $\forall e \in LS, D_{t+1}(e) = \frac{D_t(e)e^{-\alpha_t y(e) h_t(e)}}{Z_t}$ */;
Return $H$ s.t. $H(e) = \frac{1}{T}(\sum_{t=1}^{T} \alpha_t h_t(e))$ ;
```

Let W^{-1}, W^0 and W^{+1} be defined by

$$W^b = \sum_{e \in LS: y(e)h_t(e)=b} D_t(e)$$

Using symbols + and - for +1 and -1, the following property is then satisfied:

$$W^+ + W^- + W^0 = 1$$

W^+ (resp. W^-) describes then the sum of the weights of the correctly (resp. incorrectly) classified instances. W^0 describes the part of the instances unclassified by the current classifier (for example, a point located on a linear separator).

3 Boosting a Density Estimator

In our framework, the weak hypothesis h_t is not a classifier which labels an example e, by giving it a negative or positive label (-1 or $+1$). In Natural Language modelling, the examples are not split into two negative and positive classes. e is usually described by a symbol and its context, *i.e.* the beginning of the string in which it appears. Hence, h_t is now a model which must provide an *occurrence probability* for a given example e. The objective is then to compare the current inferred distribution D_t with the original density D_1 which is the target distribution. D_1 is not the uniform distribution anymore (as in Adaboost), but rather the distribution over the data. $D_1(e)$ describes in fact a conditional probability, to observe in the learning sample a symbol given its context. We aim to fit the original distribution D_1 and the distribution D_t estimated by h_t. In such a context, we cannot use the instance classes ($y(e)$ in Adaboost). The use of the weight update rule according to the correct or incorrect prediction of $h(t)$, is then impossible. To allow this adaptation of boosting to density estimation, we must solve the following problems:

1. We must redefine W^+ and W^- which describe respectively the proportion of correctly and incorrectly classified examples in the standard ADABOOST. What is now a good or a bad prediction of the weak hypothesis ?
2. Are there examples not modelled by a given weak hypothesis (corresponding to the quantity W^0 in ADABOOST) ?
3. We must redefine the weight update rule, taking into account the quality of the current estimator.

As to the first problem, we can enumerate three mutually exclusive cases for each learning instance e (see the example described on figure 1):

1. The weak hypothesis h_t provides a good estimate of the probability of e. The weights of such points will be describe by W^+ in our approach, defined as follows: $W^+ = \sum_{e/D_t(e)=D_1(e)} D_t(e)$
2. The weak hypothesis h_t under-estimates the probability to have e. The weights of such points will be described by W_1^-, the first part of W^-, the weighted sum of instances incorrectly treated by the weak hypothesis. W_1^- is then defined as follow: $W_1^- = \sum_{e/(D_t(e)-D_1(e))<0} D_t(e)$.
3. The algorithm over-estimates the probability to have e. The weights of such points will be denoted W_2^-, the second part of W^-. W_2^- is then defined as follow: $W_2^- = \sum_{e/(D_t(e)-D_1(e))>0} D_t(e)$.

Contrary to ADABOOST, which accepts instances unclassified (those described by W^0) by the weak hypothesis, the estimator provides a given density for each learning example, resulting in $W^0 = 0$. *That deals with the second problem.*

We then handle three quantities, which satisfy the following properties:

$$W^+ + W_1^- + W_2^- = 1$$

Finally, in order to correct the estimation error of h_t, we will increase (resp. decrease) the density of examples under-estimated (resp. over-estimated) by the

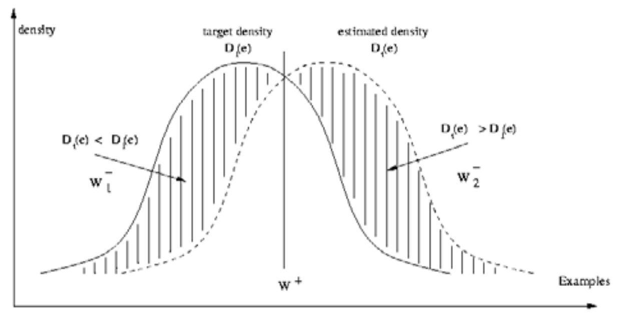

Fig. 1. Estimation errors

hypothesis. The weight of the correctly estimated examples remains the same. Then, we will use the following general weight update rule:

$$D_{t+1}(e) = \frac{D_t(e).e^{-\alpha_t(D_t(e)-D_1(e))}}{Z_t}$$

where Z_t is the normalization factor: $Z_t = \sum_e D_t(e).e^{-\alpha_t(D_t(e)-D_1(e))}$.

According to Adaboost, the confidence level α_t of the current weak hypothesis can be assessed by minimizing Z_t. Actually, minimizing Z_t results in minimizing the error rate of the weak hypothesis, since the main part of the Z_t quantity is due to misclassified instances. In our adaptation to density estimation, *mis-estimated* instances can be either under or over-estimated. Minimizing Z_t would attribute more relevance to under-estimated instances than to over-estimated ones. In such a context, a better optimization would consist in minimizing the following objective function,

$$Z_t^* = \sum_e D_t(e).e^{-\alpha_t|D_t(e)-D_1(e)|}$$

The confidence α_t of the weak hypothesis is determined by minimizing Z_t^*.

$$\frac{\partial Z_t^*}{\partial \alpha_t} = -\sum_e |D_t(e) - D_1(e)|.D_t(e)e^{-\alpha_t|D_t(e)-D_1(e)|}$$

Replacing $e^{-\alpha_t|D_t(e)-D_1(e)|}$ by its power series, we obtain

$$\frac{\partial Z_t^*}{\partial \alpha_t} = \sum_{n\geq 0} \frac{-(-\alpha_t)^n \sum_e |D_t(e) - D_1(e)|^{n+1} D_t(e)}{n!}$$

Since $|D_t(e) - D_1(e)| \in [0,1]$, we can assume that, for $n \geq 2$, $|D_t(e) - D_1(e)|^{n+1}$ is negligible, and then

$$\frac{\partial Z_t^*}{\partial \alpha_t} \simeq -\sum_e |D_t(e) - D_1(e)|.D_t(e) + \alpha_t \sum_e (|D_t(e) - D_1(e)|^2.D_t(e)$$

The value of α_t for which $\frac{\partial Z_t}{\partial \alpha_t} = 0$ is then

$$\alpha_t = \frac{\sum_e |D_t(e) - D_1(e)|.D_t(e)}{\sum_e |D_t(e) - D_1(e)|^2.D_t(e)} = \frac{E(\delta_t(e))}{E(\delta_t(e)^2)}$$

where $\delta_t(e) = |D_t(e) - D_1(e)|$, $E(\delta_t(e))$ is its first statistical moment, and $E(\delta_t(e)^2)$ its second statistical moment. Despite our approximation, we can note that α_t keeps the following interesting properties:

1. α_t tends to ∞ when the estimated density tends towards the initial distribution. We then highly weight such a good weak hypothesis.
2. α_t tends to 0 for estimated densities which do not have overlap with the initial distribution.

The pseudo-code of our boosted-algorithm, called PdfBoost, is presented in the Algorithm 2.

Algorithm 2: PdfBoost.

```
$D_1(e)$ is the probability to observe $e$ in the learning sample, $\forall e \in LS$
for $t = 2$ to $T$ do
    Build an automaton $h_t$ using $D_{t-1}$ ;
    Get an estimation $D_t$ of the probability density;
    Compute the confidence $\alpha_t = \frac{E(\delta_t(e))}{E(\delta_t(e)^2)}$ where $\delta_t(e) = |D_t(e) - D_1(e)|$;
    Update: $\forall e \in LS$: $D_{t+1}(e) = \frac{D_t(e)e^{-\alpha_t(D_t(e)-D_1(e))}}{Z_t}$;
    /*$Z_t$ is a Normalization Factor*/;
Return the final model aggregating the $T$ weighted-distributions:
    $$D^*(e) = \frac{1}{\sum_t \alpha_t}(\sum_{t=1}^{T} \alpha_t D_t(e))$$
```

4 Probabilistic Automaton as Weak Hypothesis

We recall that boosting can aggregate many weak hypothesis, *i.e.* many models. This feature induces two kind of constraints on the type of model used: on the one hand, each model must be compact, on the other hand its use must be very efficient. We decided to use, for the experimental study, Probabilistic Deterministic Finite States Automata, since they are, on the one hand, more compact than, say the N-gram model, and on the other hand, the determinism makes them faster than non deterministic probabilistic automata (*i.e.* Hidden Markov Models) when used in real applications. We present here the formal definition of the model and the inference algorithm.

A *Probabilistic Finite Automaton* (PFA) A is a 7-tuple $(\Sigma, Q^A, q_I^A, \xi^A, \delta^A, \gamma^A, F^A)$ where Σ is the alphabet, *i.e.* a finite set of symbols Q^A is the *set of states*, $q_I^A \in Q$ is the *initial* state, $\xi^A \subset Q \times \Sigma \times Q \times (0,1]$ is a set of probabilistic transitions. F^A: $Q^A \rightarrow [0,1]$ is the "end of parsing" probabilistic function. Functions δ^A and γ^A, from $Q \times \Sigma$ to Q and $(0,1]$ respectivelly are defined as: $\delta^A(q_i, \sigma) = q_j$ iff $\exists\ p \in (0,1]\ : (q_i, \sigma, q_j, p) \in \xi$ and $\gamma^A(q_i, \sigma) = p$ iff $\exists\ q_j \in Q\ : (q_i, \sigma, q_j, p) \in \xi$. These functions can be trivially extended to $Q \times \Sigma^\star$.

We require that for all states q, $\sum_\sigma \xi^A(q, \sigma, q') + F^A(q) = 1$. We assume that all states are reachable from the start state with non-zero probability, and that the automaton terminates with probability one. This then defines a distribution over $\Sigma^\star$.

$Pr_A(x) = \gamma^A(q_I, x) \times F^A(\delta(q_I, x))$ will be the probability of x w.r.t. the automaton A.

These automata differ from the one used in many papers in that they define a probability distribution over $\Sigma^\star$ and not over Σ^n with n a constant.

Let LS denote a *positive sample*, *i.e.* a set of strings belonging to the probabilistic language we are trying to model. Let $PTA(LS)$ denote the *prefix tree*

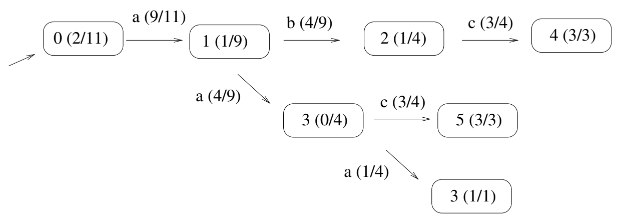

Fig. 2. PPTA built with $LS = \{$aac,λ, aac, abd, aac, aac, abd, abd, a, ab, $\lambda\}$

acceptor built from a positive sample LS. The prefix tree acceptor is an automaton that only accepts the strings in the sample and in which common prefixes are merged together resulting in a tree-shaped automaton. Let $PPTA(LS)$ denote the *probabilistic prefix tree acceptor*. It is the probabilistic extension of the $PTA(LS)$ in which each transition has a probability related to the number of times it is used while generating, or equivalently parsing, the positive sample. Let $C(q)$ denote the count of state q, that is, the number of times the state q was used while generating LS from $PPTA(LS)$. Let $C(q,\#)$ denote the number of times a string of LS ended on q. Let $C(q,a)$ denote the count of the transition (q,a) in $PPTA(LS)$. The $PPTA(LS)$ is the maximal likelihood estimate built from LS. In particular, for $PPTA(LS)$ the probability estimates are $\overset{\wedge}{\gamma}(q,a) = \frac{C(q,a)}{C(q)}$, $a \in \Sigma \cup \{\#\}$.

Figure 2 exhibits a $PPTA$ and the learning set it is built from.

We now present the second tool used by the generic algorithm: the state merging operation. This operation induces two modifications to the automaton: (i) it modifies the structure (figure 3, left) and (ii) the probability distribution (figure 3, right). It applies to two states. Merging two states can lead to non-determinism. The states that create non-determinism are then recursively merged. When state q results from the merging of the states q' and q'', the following equality must hold in order to keep an overall consistent model:

$$\gamma(q,a) = \frac{C(q',a)+C(q'',a)}{C(q')+C(q'')}\ , \forall a \in \Sigma \cup \{\#\}$$

One can note two properties of the update of the probabilities: (i) $\frac{C(q',a)+C(q'',a)}{C(q')+C(q'')}$ is included in $[\frac{C(q',a)}{C(q')}, \frac{C(q'',a)}{C(q'')}]$ which means that the probability of an after merge transition has a value bounded by the two values of the transitions it comes from; (ii) the merge naturally weights more the probability of the transition that holds the more information. For instance, $\frac{20+100}{1000+105} = \frac{120}{1105}$ is closer to $\frac{20}{1000}$ than to $\frac{100}{105}$.

These remarks hold for each pair of transitions that takes part in the merge. Let us merge states q_i and q_j and define P_{q_i} (resp. P_{q_j}) as the probability distribution defined by considering state q_i (resp. q_j) as the initial state. Since the merge is recursively applied (see figure 3), the probability distribution after merging states q_i and q_j will be a kind of weighted mean between the distributions P_{q_i} and P_{q_j}.

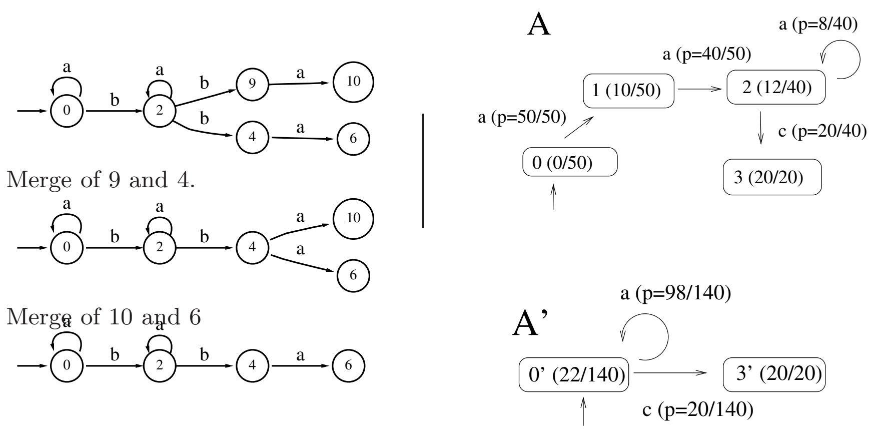

Fig. 3. Merging states 5 and 2

We are now in a position to present the MDI algorithm itself for which we recall in the next section the main features.

The MDI algorithm [21] (algorithm 3) takes two arguments: the learning set LS and a tuning parameter β. It looks for an automaton, that is the result of a tradeoff between a small size and a small distance to the data. The distance measure used is the Kullback-Leibler divergence. The data are represented by the PPTA as it is the maximum likelihood estimate of the data. While merging two states, the distance between the automaton and the data, in general, increases and, at the same time, the number of states and the number of transitions, in general, decreases. Two states will be set compatible if the impact in terms of divergence of their merge divided by the gain of size is smaller than the parameter β.

Algorithm 3: MDI (LS, β).

```
A ← Numbering_in_Breadth_First_Order($PPTA$);
for $q_i = 1$ to Nb_State (A) do
    for $q_j = 0$ to $i - 1$ do
        if Compatible $(A, q_i, q_j) < \beta$ then Merge (A,$q_i$,$q_j$) ;
Return A ;
```

The MDI algorithm tries to infer a small automaton that is close to the data. In fact, the bigger β, the more general (and small with respect to the number of states) the resulting automaton should be. The parameter β hence controls the level of generalization of the algorithm. Since boosting is known to overfit

Table 1. Probability updates, where $D_i(\text{"}to\text{"}) = D_i(\text{"}to|I'd\ like...C\text{"})$

e	D_1	D_2	D_3	D_4	D_5	$D_1^\star$	$D_2^\star$	$D_3^\star$	$D_4^\star$	$D_5^\star$
`C|I'd .. D`	1	.4961	.8754	.9012	1	1	.7407	.7864	.8155	.8532
`to| I'd..C`	1	.4304	.5931	.9254	.9265	.6061	.5157	.5419	.6391	.6977
`Atlanta|I'd..to`	1	.1682	.0910	.0621	.5640	.2791	.2220	.1775	.1483	.2331

with noisy data, having such a controlling parameter seems to be crucial in this context.

Usually, a cross validation protocol is used to assess the right value for the parameter β. The learner is then considered as the algorithm with β estimated on a held-out development set. Another point of view is to consider that each value of β defines a particular algorithm that belongs to a family of learner parameterized by β. Note that the point here is not to optimally define the value of the parameter β, because the algorithm is still a heuristic method. Moreover the boosting just needing a weak learner, we will consider the MDI algorithm, used with a fixed value of the parameter β, as a given weak learner. We will see in the next section that it deserves further investigation.

5 Experimental Issues

We used for this experimental study a very noisy database coming from a language modeling task, *i.e.* the ATIS task. The Air Travel Information System (ATIS) corpus [6] was developped under a DARPA speech and natural language program that focused on developing language interfaces for information retrieval systems. The corpus consists of speakers of American English making information requests such as: `‘‘I’d like to find the cheapest flight from Washington D C to Atlanta’’`.

Since the probabilities of the whole sentences are here very small (as a product of all the conditional probabilities), we decided to deal with the conditional probabilities. Since, the PPTA represents the maximum likelihood of the data, it will describe the target density D_1 in PdfBoost. At each step t, D_{t+1} is not an update of LS, but rather an update of the PPTA transition probabilities.

Before studying the PdfBoost behavior on the whole database, we decided to test it on some conditional probabilities to see whether our intuitions are corroborated.

The PdfBoost behavior is shown on table 1 where we provide the evolution of some conditional probabilities taken from the example sentence given above (”`... D C to Atlanta`)”. In this table, D_i is the probability provided at iteration i; $D_i^\star$ is the probability provided by the first i-aggregated models.

As one can see, the probabilities tend rather quickly to the value of the training sample (column D_1 in the table). This is true either for the non aggregate models (columns D_i) and for the aggregate ones (columns $D_i^\star$). One can note that aggregate models converge slower, hopefully leading to less overfit than the models taken alone.

6 Behavior on the Full Task

The criterion used to estimate the quality of a model is the perplexity. Probabilistic models cannot be evaluated by classification error rate, as the fundamental problem has become the estimation of a probability distribution over the set of possible strings.

The quality of the model is measured by the *per symbol log-likelihood* of strings x belonging to a test sample according to the distribution defined by the hypothesis $P_A(x)$ computed on a test sample S :

$$LL = -\frac{1}{\|S\|} \sum_{j=1}^{|S|} \sum_{i=1}^{|x|} \log P(x_i^j | q^i)$$

where $P(x_i^j|q^i)$ denotes the probability of generating x_i^j, the i-th symbol of the j-th string in S, given that the generation process was in state q^i.

The *test sample perplexity* PP is most commonly used for evaluating language models of speech applications. It is given by $PP = 2^{LL}$. The minimal perplexity $PP = 1$ is reached when the next symbol x_i^j is always predicted with probability 1 from the current state q^i (i.e. $P(x_i^j|q^i) = 1$) while $PP = |\Sigma|$ corresponds to random guessing from an alphabet of size $|\Sigma|$.

We aim here to check if the behavior of PdfBoost is coherent with the behavior of the standard AdaBoost algorithm. With the standard algorithm, the classifier gets closer and closer to the training set. Figure 4 shows the training-set perplexity of the aggregated automata. As expected, the perplexity goes down as the number of iterations grows. It stabilizes at around 100 iterations.

This means that the update function chosen is adequate with respect to the perplexity.

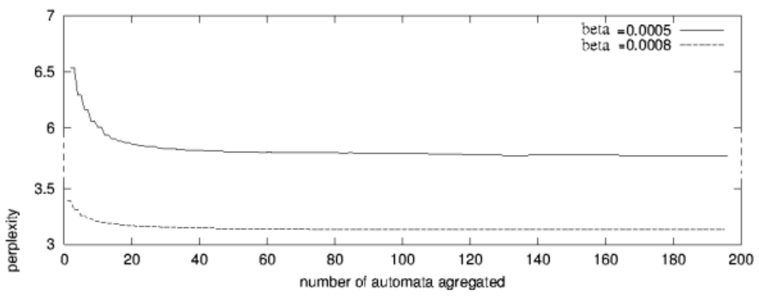

Fig. 4. Behavior of the aggregated model on the training set

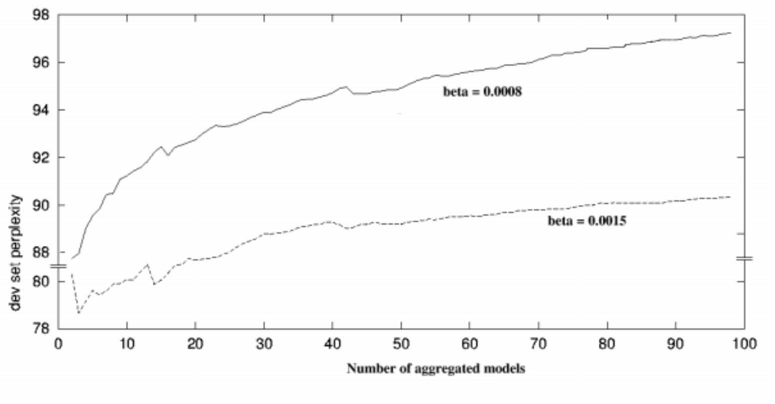

Fig. 5. Behavior of the aggregated model on the development set

Figure 5 shows the behavior of the method on the development set, *i.e.* its behavior in generalization. As one can see, the two curves are rather different depending on the value of the tuning parameter β. It is interesting to notice that the parameter which performs best on the training set performs worse on the development one. From our point of view, this means that having a parameter that prevents generalization will cause over-fitting. Actually, during the four first boosting steps the development-set perplexity goes down, which shows the interest of our approach. The curves then raise, which, from our point of view, proves over-fitting. We think that tuning the MDI parameter β at each boosting step could prevent over-fitting and thus lead to better results in generalization.

7 Conclusion and Further Work

The preliminary results presented in Figure 5 seem promising to us in that they tend to show that the behavior of PdfBoost is coherent with the one usually observed in the classical boosting. The next step will be to have a complete study of the behavior on the development set, *e.g.* tuning the inference algorithm at each boosting step in order to prevent overfitting.

Since the boosting has been already applied to prototype selection [18], another further work should be to see if there exists a *unified boosting* that could include the three frameworks known at the moment, *i.e.* the classic boosting, the boosting applied to protototype selection, and the boosting of density function estimation.

Acknowledgments

We wish to thank Alexander Clark for useful remarks on preliminary version.

References

[1] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, 1999. 431

[2] P. Bladi, Y. Chavin, and T. Hunkapillerand McClure. Hidden markov models of biological primary sequence information. In *National Academy of Science*, pages 1059–1063, USA, 1991. 431

[3] Andrew Brown and Geoffrey Hinton. Products of hidden markov models. In T. Jaakkola and T. Richardson, editors, *Artificial Intelligence and Statistics*, pages 3–11. Morgan Kaufmann, 2001. 431

[4] Y. Freund and R. E. Shapire. A decision theoretic generalization of online learning and an application to boosting. *Intl. Journal of Computer and System Sciences*, 55(1):119–139, 1997. 431, 432

[5] Joshua Goodman. A bit of progress in language modeling. Technical report, Microsoft Reserach, 2001. 431

[6] L. Hirschman. Multi-site data collection for a spoken language corpus. In *DARPA Speech and Natural Language Workshop*, pages 7–14, 1992. 439

[7] Frederick Jelinek. *Statistical Methods for Speech Recognition*. The MIT Press, Cambridge, Massachusetts, 1998. 431

[8] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, Englewood Cliffs, New Jersey, 2000. 431

[9] M. J. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. E. Schapire, and L. Sellie. On the learnability of discrete distributions. In *Proc. of the 25th Annual ACM Symposium on Theory of Computing*, pages 273–282, 1994. 431

[10] Anders Krogh, Michael Brown, I. Saira Mian, Kimmen Sjolander, and David Haussler. Hidden markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994. 431

[11] R. Maclin and D. Opitz. An empirical evaluation of bagging and boosting. In *Proc. of the Fourteenth Natl. Conf. on Artificial Intelligence*, pages 546–551, 1997. 431

[12] E. Roche and Yves Schabes. *Finite-State Language Processing*. MIT Press, 1997. 431

[13] D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. In *ACM*, pages 31–40, Santa Cruz, 1995. COLT'95. 431

[14] E. Tjong Kim Sang. Text chunking by system combination. In *CoNLLL-2000 and LLL-2000*, pages 151–153, Lisbon, Portugal, 2000. 431

[15] R. E. Schapire, Y. Freund, P. Bartlett, and W. Sun Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of Statistics*, 1998. 432

[16] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, 1998. 432

[17] M. Sebban and R. Nock. Contribution of boosting in wrapper models. In *Proc. of the Thirth European Conf. on Principles and Practice of Knowledge Discovery in Databases*, pages 214–222, 1999. 431, 432

[18] M. Sebban, R. Nock, and S. Lallich. Boosting neighborhood-based classifiers. In *Proc. of the Seventeenth Intl. Conf. on Machine Learning*, 2001. 431, 432, 441

[19] A. Stolcke and S. Omohundro. Inducting probabilistic grammars by bayesian model merging. In Lecture Notes in Artifitial Intelligence, editor, *Second Intl Collo. on Gramatical Inference*, 862, pages 106–118. ICGI-94, 1994. 431

[20] F. Thollard. Improving probabilistic grammatical inference core algorithms with post-processing techniques. In *Eighth Intl. Conf. on Machine Learning*, pages 561–568, Williams, July 2001. Morgan Kauffman. 431

[21] F. Thollard, P. Dupont, and C. de la Higuera. Probabilistic dfa inference using kullback-leibler divergence and minimality. In Pat Langley, editor, *Seventh Intl. Conf. on Machine Learning*, San Francisco, June 2000. Morgan Kaufmann. 431, 438