Stratified Gradient Boosting for Fast Training of Conditional Random Fields *

Bernd Gutmann[1] and Kristian Kersting[2]

[1] Department of Computer Science, Katolieke Universiteit Leuven
Celestijnenlaan 200A, 3001 Heverlee, Belgium
[2] CSAIL, Massachusetts Institute of Technology
32 Vassar Street, Cambridge, MA, 02139-4307, USA

Abstract. Boosting has recently been shown to be a promising approach for training conditional random fields (CRFs) as it allows to efficiently induce conjunctive (even relational) features. The potentials are represented as weighted sums of regression trees that are induced using gradient tree boosting. Its large scale application such as in relational domains, however, suffers from two drawbacks: induced trees can spoil previous maximizations and the number of generated regression examples can become quite large. In this paper, we propose to tackle the latter problem by injecting randomness into the regression estimation procedure by subsampling regression examples. Experiments on a real-world data set show that this sampling approach is comparable with more sophisticated boosting algorithms in early iterations and, hence, provides an interesting alternative as it is much simpler to implement.

1 Introduction

Sequential data are ubiquitous and are of interest to many communities. Such data can be found in virtually all application areas of machine learning including computational biology, activity recognition, information extraction. As an example, consider the task of marking all proteins in abstracts of biological publications. One appealing approach to such sequence labeling problems are (Lafferty et al., 2001)'s *conditional random fields* (CRFs). They are undirected models encoding the conditional dependency $P(Y|X)$ and have outperformed HMMs (Rabiner, 1989) on language processing tasks such as information extraction and shallow parsing. In contrast to generatively trained HMMs, the discriminatively trained CRFs are designed to handle non-independent input features such as such as the molecular weight and the neighboring acids of an amino acid.

The flexibility, however, comes at the expense of severe training costs. To this end, fast and integrated feature induction and parameter estimation techniques

* An earlier version of this work appeared as 4-pages extended abstract in the electronic working notes of the 5th International Workshop on Mining and Learning with Graphs (MLG'07), August 1–3, 2007, Universita degli Studi di Firenze, Florence, Tuscany, Italy.

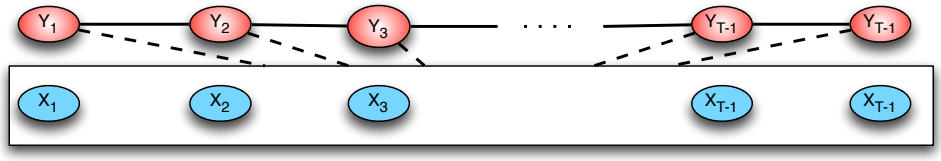


Fig. 1. Graphical representation of linear-chain CRF.

have been proposed. (McCallum, 2003)'s Mallet system employs the BFGS algorithm, which is a second-order parameter optimization method that deals with parameter interactions, and induces features iteratively. Starting with a single feature, conjunctions of features are iteratively constructed that significantly increase conditional log-likelihood if added to the current model. Recently, (Dietterich et al., 2004) proposed a boosting approach, called TreeCRF, which is competitive to Mallet. TreeCRF follows (Friedman, 2001)'s gradient tree boosting algorithm, i.e., the potential functions are represented by sums of regression trees, which are grown stage-wise in the manner of Adaboost (Freund & Schapire, 1996). Each regression tree can be viewed as defining several new feature combinations, one for each path in the tree from the root to a leaf. Thus, the features can be quite complex; even relational conjunctions as shown by Gutmann and Kersting (2006)'s TildeCRF.

One major drawback of the gradient tree boosting approach is that the number of generated regression examples can become very large. If we have 3 labels and 100 training sequences of length 200, then the number of training examples for each label k is $3 \cdot 100 \cdot 200 = 60,000$. To get around this, we propose a sampling strategy tailored to gradient tree boosting for (relational) CRFs. More precisely, we introduce a stratified sampling approach for CRFs, conduct an experimental evaluation, and examine the influence of the subsampling, the line search and the gradient method (steepest ascent vs. conjugated gradient) on the predictive performance.

We proceed as follows. After reviewing CRFs and gradient tree boosting, we discuss our stratified sampling scheme. Before concluding, we we evaluate our approach on a real-world information extraction dataset.

2 Gradient Tree Boosting for CRFs

CRFs are undirected graphical models that encode conditional probability distributions using a given set of features. We will focus on *linear-chain* CRF models, cf. Figure 1.

Let G be an undirected graphical model over sets of random variables X and Y. For linear-chain CRFs, $X = \langle x_{i,j} \rangle_{j=1}^{T_i}$ and $Y = \langle Y_{i,j} \rangle_{j=1}^{T_i}$ correspond to the input and output sequences such that Y is a labeling of an observed sequence X. The conditional probability of a state sequence given the observed sequence

is defined as

$$P(Y|X) = Z(X)^{-1} \exp \sum_{t=1}^{T} \Psi_t(y_t, X) + \Psi_{t-1,t}(y_{t-1}, y_t, X),$$

where $\Psi_t(y_t, X)$ and $\Psi_{t-1,t}(y_{t-1}, y_t, X)$ are potential functions [3] and $Z(X)$ is a normalization factor over all state sequences X so that each potential contributes overall probability.

2.1 Training

Typically, it is assumed that the potentials factorize according to a set of features $\{f_k\}$, which are given and fixed, so that

$$\Psi(y_t, X) = \sum \alpha_k g_k(y_t, X)$$
$$\text{and } \Psi(y_{t-1}, y_t, X) = \sum \beta_k f_k(y_{t-1}, y_t, X)$$

respectively. The model parameters are now a set of real-valued weights α_k, β_k; one weight for each feature. To estimate them, a conditional maximum likelihood approach is typically followed. That is, the (conditional) likelihood of the training data given the current parameter Θ_{m-1} is used to improve the parameters. Normally, one uses some sort of gradient search for doing this:

$$\Theta_m = \Theta_0 + \delta_1 + \ldots + \delta_m$$
$$\text{where } \delta_m = \eta_m - M \cdot \frac{\partial}{\partial \Theta_{m-1}} \sum_i \log P\left(y_i | x_i; \Theta_{m-1}\right)$$

is the gradient multiplied by a constant η_m, which is obtained by doing a line search along the gradient.

2.2 Training via Gradient Tree Boosting

Gradient tree boosting interleaves parameter estimation and feature selection. More precisely, one starts with some initial potential Ψ_0, e.g. the zero function, and adds iteratively corrections

$$\Psi_m = \Psi_0 + \Delta_1 + \ldots + \Delta_m.$$

In contrast to the standard gradient approach, Δ_i denotes the so-called functional gradient, i.e.,

$$\Delta_m = \eta_m \cdot E_{x,y} \left[\frac{\partial}{\partial \Psi_{m-1}} \log P(y|x; \Psi_{m-1}) \right].$$

Since the joint distribution $P(x, y)$ is unknown, one cannot evaluate the expectation $E_{x,y}$. Instead ones evaluates the gradient function at every position in every

[3] A *potential function* is a real-valued function that captures the degree to which the assignment y_t to the output variable fits the transition from y_{t-1} and X.

Algorithm 1 The inner loop of gradient tree boosting for generating the regression examples

function GENEXAMPLES($k, Data, Pot_m$)
  $S := \emptyset$ ▷ Initialize relational regression examples
  for all $(X_i, Y_i) \in Data$ **do** ▷ Iterate over all training examples
    $\big(\alpha, \beta, Z(X_i)\big) =$ FORWARDBACKWARD(X_i, T, K) ▷ Compute forward and backward probabilities
    for $1 \leq t \leq T_i$ **do** ▷ Iterate over all positions
      for $1 \leq k' \leq K$ **do** ▷ Iterate over all class labels
        ▷ Compute value of gradient at position t for class label k
        $P(y_{t-1} = k', y_t = k|X_i) := \dfrac{\alpha(k', t-1) \cdot \exp(F_m^k(k', w_t(X)) \cdot \beta(k, t)}{Z(X_i)}$
        $\Delta(k, k', t) := I(y_{t-1} \subseteq_\Theta k', y_t \subseteq_\Theta k) - P(y_{t-1} \subseteq_\Theta k', y_t \subseteq_\Theta k|X_i)$
        $S := S \cup \{((w_t(X_i), k'), \Delta(k, k', t))\}$ ▷ Update set of relational regression examples
      end for
    end for
  end for
  return S
end function

training example and fit a regression tree to these derived examples. More precisely, setting $F(y_{t-1}, y, t, X) = \Psi(y_t, X) + \Psi(y_{t-1}, y_t, X)$, the gradient becomes (see (Dietterich et al., 2004; Gutmann & Kersting, 2006) for more details),

$$\frac{\partial \log P(Y|X)}{\partial F(u, v, w_d(X))} = I(y_{d-1} \subseteq_\Theta u, y_d \subseteq_\Theta v) - P(y_{d-1} \subseteq_\Theta u, y_d \subseteq_\Theta v|w_d(X)), \tag{1}$$

where I is the indicator function, $\subseteq_\Theta$ denotes that u matches/subsumes y, and $P(y_{d-1} \subseteq_\Theta u, y_d \subseteq_\Theta v|w_d(X))$ is the probability that class labels u, v fit the class labels at positions $d, d-1$. By evaluating the gradient at every known position in the training data and fitting a regression model such as a relational regression tree, cf. Figure 2, to these values, one gets an approximation of the expectation $E_{x,y}[\partial/\partial\Psi_{m-1}]$ of the gradient. In order to speed-up computations, not the complete input X is typically used but only a window $w_d(X) = x_{d-s}, \ldots, x_d, \ldots, x_{d+s}$, where s is a fixed window size.

2.3 Conjugate Direction Boosting

Reconsider the basic gradient-ascent optimization approach. One of the problems with choosing the step size doing a line search is that a maximization in one direction could spoil past maximizations. To avoid this, conjugate gradient boosting methods (Kersting & Gutmann, 2006) compute so-called conjugate directions in the function space, which are orthogonal and, hence, do not spoil previous maximizations. The step size is estimated along these directions doing line searches.

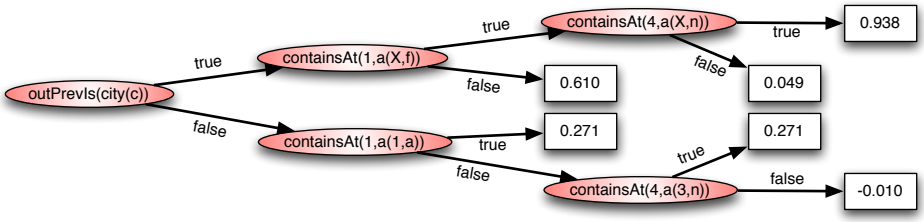
Algorithm 2 Conjugated gradient tree boosting with line search and sampling.

function CGTreeBoost($Data, L$)
  for $1 \leq m \leq M$ **do** ▷ Iterate Functional Gradient
    $S := \emptyset$
    for $1 \leq k \leq K$ **do** ▷ Iterate through the class labels
      $(S_{Pos}, S_{Neg}) := (S_{Pos}, S_{Neg}) \cup$
        GenExamples($k, Data, F_{m-1}, m$)
    end for
    S_{Sample} := Sample((S_{Pos}, S_{Neg}))
    Δ_m := FitRelRegressTree(S_{Sample}, L)
    if $m = 1$ **then**
      $d_1 = \Delta_1$ ▷ Initial conjugate direction
    else
      $\beta_m = \frac{\langle \Delta_m, \Delta_m - \Delta_{m-1} \rangle}{\langle \Delta_{m-1}, \Delta_{m-1} \rangle}$ ▷ Polak-Ribiére formula
      $d_m = \Delta_m + \beta_m \cdot d_{m-1}$ ▷ Next conjugate direction
    end if
    η_m :=LineSearch($Data, F_{m-1}, d_m$) ▷ Line Search along d_m
    $F_m := F_{m-1} + \eta_m \cdot d_m$ ▷ Model udpate
  end for
  return F_M ▷ Return Potential
end function

More precisely, the empirical angle β_m between Δ_m and Δ_{m-1} on the training examples given the current gradient Δ_m is computed. As shown in Alg. 2, the current gradient plus the old weighted gradient multiplied by the calculated angle is added to the current model, $d_m = \Delta_m + \beta_m \cdot d_{m-1}$. The angle β_m can be calculated by evaluating the Polak-Ribiére formula for each example. Every weighted gradient d_m is a linear combination of the gradients $\Delta_1, \ldots, \Delta_m$. It can be shown that

$$d_t = \sum_{i=1}^{m} \beta_{i,m} \cdot \Delta_i$$
$$\text{where} \;\; \beta_{m,m} = 1 \text{ and } \beta_{i,m} = \prod_{j=i+1}^{m} \beta_j \text{ if } i < m.$$

Fig. 2. A (relational) regression tree. Nodes denote tests; leaves are the predicted values.



Algorithm 3 Stochastic gradient tree boosting.

```
1: function CGTreeBoost(Data, L, N)
2:     for 1 ≤ m ≤ M do                                  ▷ Iterate Functional Gradient
3:         S := ∅
4:         Data' := randomly sample a subset of size N from Data
5:         for 1 ≤ k ≤ K do                              ▷ Iterate through the class labels
6:             S := S ∪ GenExamples(k, Data, F_{m-1}, m)
7:         end for
8:         Δ_m := FitRelRegressTree(S_Sample, L)
9:         F_m := F_{m-1} + d_m                          ▷ Model udpate
10:    end for
11:    return F_M                                        ▷ Return Potential
12: end function
```

3 Stochastic gradient tree boosting

One major drawback of the gradient tree boosting approach is that the number of generated regression examples can become very large. In every iteration, $k^2 \times n \times l$ regression examples are generated (k is the number of labels, n the number of training examples, l the average sequence length).

An obvious modification to speed up gradient tree boosting is to use only a subset of the original data. This subset is drawn randomly in every iteration which ensures that the complete training data contributes to the learned model. This modification, *stochastic gradient tree boosting*, was originally proposed by Friedman (2002). Algorithm 3 shows the pseudocode, where N is the size of the sample. For $N = \text{size}(Data)$ the algorithm behaves like standard gradient tree boosting. Stochastic gradient tree boosting is a kind of bagging approach. Experimental results show that it improves the runtime (which is an obvious result) and can also increase the accuracy of the learned model.

Indeed, stochastic gradient tree boosting can directly be used for fast training of conditional random fields. It is, however, a general-pupose technique and methods tailored to the problem at hand, namely training conditional random fields are likely to improve performance. In (Kersting & Gutmann, 2006) we showed that gradient tree boosting for CRFs induces an expectation bias on the generated regression examples: For every observed (called positive) example in the training data $k-1$ unobserved (negative) regression examples are generated (k is the number of possible labels in the output sequence). For higher k, the regression tree learner spends a lot of time to fit the tree to unobserved examples, whereas the predictive accuracy on positive examples is lower. We proposed to reweight the examples such that the empirical ratio of positive to negative examples is equal. However, the problem remained that both the regression tree learner and the generating step of gradient tree boosting hat to consider all data.

Indeed, this approach rebalances the positive and negative examples. The regression tree learner, however, must still consider all examples. To this end, we propose to use stratified sampling. This means that we use two different sampling strategies, one for the observed regression examples and one for the

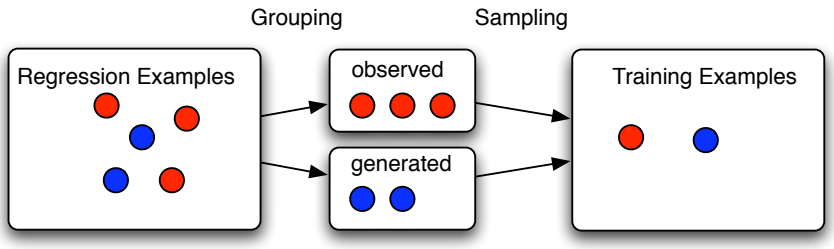


Fig. 3. Stratified sampling methodology.

negative examples. The idea is to use reduce the difference between the number of positive and negative examples. Doing so, we increase both the predictive performance on the positive examples and we reduce the time needed by the regression tree learner.

4 Stratified Sampling

There are several other possible sampling approaches for gradient tree boosting. (Friedman, 2001) suggested two techniques to speed up the learning process by reducing the number of regression examples. *Influence Trimming* ignores regression examples with absolute value smaller ϵ. We do not investigate this method but instead consider *sampling*:

> *use only a randomly drawn subset of the generated examples to train the regression trees.*

Friedman's original proposal is to subsample uniformly from all training examples. This strategy is suboptimal for training CRFs. Most of the generated examples are likely to have never been observed. Recall that for each *observed* y_{t-1} we generate $k-1$ many *expected* labels y_t. Thus, with an increasing number of labels, the probability of sampling an observed regression examples decreases (keeping the training set fixed); roughly at the scale of $\mathcal{O}(\frac{1}{k^2})$. In turn, the influence of the expected examples, the ones we have not observed, increases and gradient boosting is likely to induce meaningless models. This is truely an undesirable property of uniform sampling. To overcome this, we propose to a stratified sampling scheme.

Stratified sampling, cf. Figure 3 is a method of sampling in which it is desirable to maintain certain characteristics of the data set in any subset sampled. It is achieved by firstly partitioning the data set into a number of mutually exclusive subsets of cases (strata), each of which is representative of some aspect of the real-world process involved. Sampling from the population is then achieved by randomly sampling from the various subsets so as to achieve representative proportions.

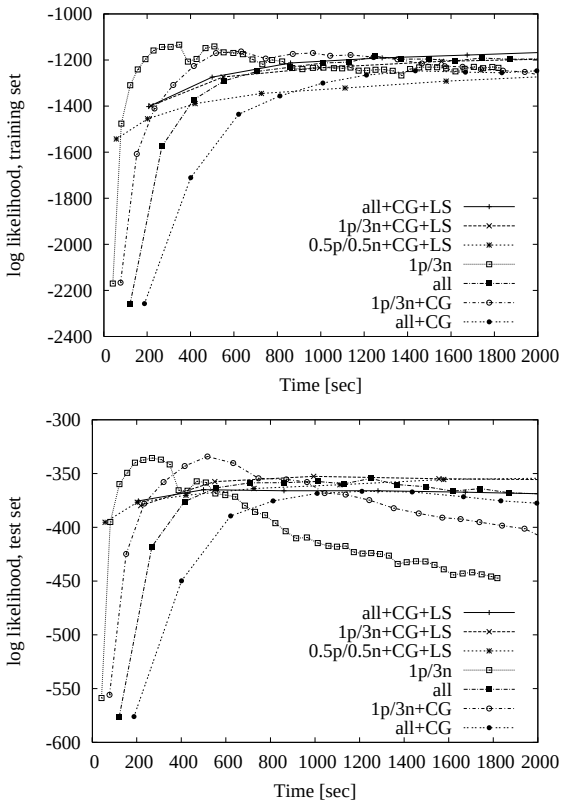


Fig. 4. 5-fold cross-validated log likelihoods on training and test set; all: all regression examples were used, 1p/3n: all positive and 3 times more negative regression examples were sampled, 0.5p/0.5n: only half of the positive and half of the negative examples were sampled, CG: conjugated gradients were used, LS: a line search was used

In our case at hand there are two natural strata: the observed and the generated regression examples. Thus, when we generate the regression examples, we mark those examples as positive for which the identity function I returnes 1.0 (see Equation (1)). Then, we sample with different strategies for the positive and negative examples S_{Pos} and S_{Neg}. For instance, we can take all positive examples and sample a fraction of the negative examples that is 3 times bigger than the positive examples. We denote this strategy by 1p/3n.

5 Experiments

Our intention was to examine the influence of the sampling rate, the line search and gradient method on the predictive performance. To this aim, we integrated

stratified sampling in TildeCRF for boosting relational CRFs, which is implemented in YAP Prolog. The regression tree learner was implemented in hipP Prolog.

We ran several experiments on a subset of (Skounakis et al., 2003)'s *subcellular-location* data set. The data set consists of sentences of medical abstracts, where each word in the input sequence is augmented with the word type and the phrase type. The task is to find proteins and their location within the DNA. More precisely, the output sequence consists of sequences over `protein, location, none`. An example of for a training example is

$$[\texttt{npSeg}(\texttt{unk}('\texttt{rat7p}')), \texttt{vpSeg}(\texttt{cop}('\texttt{is}')), \texttt{vpSeg}(\texttt{v}('\texttt{located}')),$$
$$\texttt{ppSeg}(\texttt{prep}('\texttt{at}')), \texttt{npSeg}(\texttt{art}('\texttt{the}')), \texttt{npSeg}(\texttt{adj}('\texttt{nuclear}')),$$
$$\texttt{npSeg}(\texttt{unk}('\texttt{rim}')), \ldots\}$$
$$[\texttt{protein}, \texttt{none}, \texttt{none}, \texttt{none}, \texttt{none}, \texttt{location}, \ldots]$$

We removed all training sequences that do not contain a protein or location tag. Because of memory limitations[4], we only took 25% of the resulting data set, namely 195 input/output sequence pairs. and only sampling techniques were running with. to still compare to the deterministic approach. For all our experiments, we did a 5-fold cross validation, 156 examples for training and 39 for testing the trained model. We tried several sampling strategies and ran the experiments with and without line search and conjugated gradients. We used a window size of 5 elements, the tree learner learned the regression trees up to depth 20 but it stop splitting a node (pre-pruning) when the variance was smaller than 0.0001.

Figure 4 shows the results plotted against the training time. Strategy 1p/3n, which uses all positive and 3 times as many negative regression examples as positive ones, outperforms all other approaches in early iterations. In this case we used only 4/9 of the training examples, since 1/9 of the examples where positive and 8/9 were negative ($k = 3$). It is remarkable that 1p/3n is even better than the 0.5p/0.5n+CG+LS strategy. Where we use 50% of the examples and run furthermore a more sophisticated training algorithm. To understand why, one has to see that the line search is rather slow and furthermore 50% of the observed data is not considered. Whereas in the 1p/3n one uses all the observed data. Furthermore, all methods actually achieve similar performances in terms of likelihood, i.e., the objective function we are maximizing in later iterations. Note, however, that only the stratified sampling approach achieves the highest training and test likelihoods already after about 400 sec. This would also coincide with the stopping point of the optimization if we had implemented a stopping rule. We did not do so in order to see the long term behaviors of the boosting algorithms. To summarize,

[4] For larger data sets the deterministic boosting technique ran out of memory. Only the stochastic versions had a chance to keep running due to the reduction in memory consumptions.

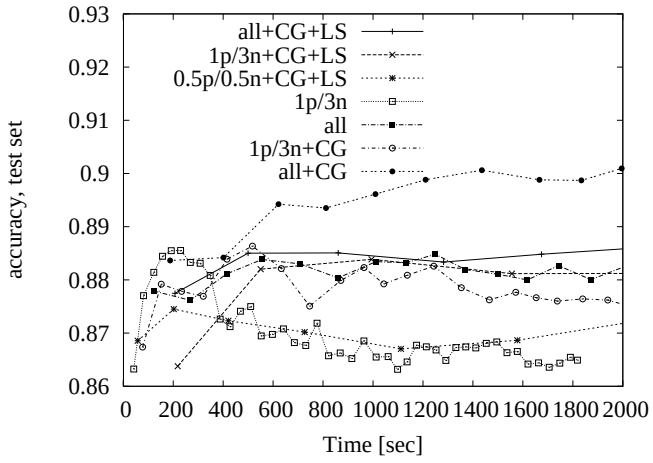


Fig. 5. 5-fold cross-validated accuracy on the test set.

stratified sampling speeds up computations and achieves higher log-likelihood estimates.

To complete the view on the performance, we investigated the 5-fold cross-validated accuracy. Figure 5 shows the accuracy of the trained model on the test data. The accuracy is defined as c/a where c is the number of correctly predicted positions in all sequences and a is the number of all positions in all sequences. One can see that strategy 1p/3n performs best for the early iterations and afterwards all+CG outperforms all the other strategies. In other words,

subsampling negative examples yields slightly lower accuracies but does not degrade performance.

Basically all methods are all in close range and comparable. This supports our results from (Kersting & Gutmann, 2006). Furthermore, one can see that the line search does not significantly increase the accuracy. We believe that the line search actually tends to overfit .

Finally, we ran several other sampling strategies, 1p/2n, 0.25p/0.25 and so on. The results are not presented here as they follow the general picture:

- Stratified sampling taking all positive examples but subsampling negative ones is much faster than deterministic gradient tree boosting and achieves comparable performance.
- The less examples (in particular positive ones) are used, the worse the performance.

6 Conclusions and Future Work

We have presented a stochastic gradient tree boosting approach to training (relational) CRFs. In contrast to existing stochastic techniques, we do not follow a uniform sampling strategy but sample positive/observed and negative/generated regression examples following different strategies. The experimental results have shown that this stratified sampling scheme indeed speeds up computations and can improve performance. It actually performs better than uniform sampling. Moreover, compared to more advanced boosting techniques such as conjugate direction boosting, it is much simpler to implement. Thus, we view stochastic boosting based on stratified sampling as an attractive and promising approach to scaling up CRF training to large data sets.

Indeed more experiments on larger data sets have to be conducted to confirm the scale up behavior. The reduction of examples on our rather small data set are encouraging; for larger data sets, one can expect higher reduction rates. It is also interesting to investigate stratification of output class, not only of positive/negative regression examples.

Acknowledgments

The authors would like to thank the anonymous reviewers for the valuable comments, Luc De Raedt for his support, and Vítor Santos Costa for his help with the Prolog system YAP. This work has been supported by the Research Foundation-Flanders (FWO-Vlaanderen)

Bibliography

Dietterich, T., Ashenfelter, A., & Bulatov, Y. (2004). Training conditional random fields via gradient tree boosting. *Proc. 21st International Conf. on Machine Learning* (pp. 217–224). ACM.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. *In Proceedings of ICML-96* (pp. 148–156). Morgan Kaufman.

Friedman, J. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, *29*.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, *38*, 367–378.

Gutmann, B., & Kersting, K. (2006). Tildecrf: Conditional random fields for logical sequences. *Proc. of the 15th European Conference on Machine Learning (ECML-2006)* (pp. 174–185). Berlin, Germany: Springer.

Kersting, K., & Gutmann, B. (2006). Unbiased conjugate direction boosting for conditional random fields. *Working Notes of the ECML-2006 Workshop on Mining and Learning with Graphs (MLG 2006)* (pp. 157–164). Berlin, Germany. Short paper.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proc. 18th Int. Conf. on Machine Learning (ICML-01)* (pp. 282–289).

McCallum, A. (2003). Effciently inducing features of conditional random fields. *Proc. of the 21st Conference on Uncertainty in Artificial Intelligence (UAI-03)* (pp. 403–410). Edinburgh, Scotland.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE* (pp. 257–285).

Skounakis, M., Craven, M., & Ray, S. (2003). Hierarchical hidden markov models for information extraction. *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)* (pp. 427–433).