Stratified Gradient Boosting for Fast Training of CRFs (Extended Abstract)

Bernd Gutmann BERND.GUTMANN@CS.KULEUVEN.BE
Department of Computer Science, Katolieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium

Kristian Kersting KERSTING@CSAIL.MIT.EDU
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, 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 in the regression estimation procedure due to subsampling regression examples.

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 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

Extended abstract in the working notes of MLG-2007, August 1-3, Florence, Italy, 2007. Copyright 2007 by the author(s)/owner(s). This is an extended version of K. Kersting. B. Gutmann. "Unbiased Conjugate Direction Boosting for Conditional Random Fields". Short paper in T. Gärtner, G. C. Garriga, T. Meinl, editors, Working Notes of MLG-06, Berlin, Germany, September 18th, 2006.

of the subsampling, the line search and the gradient method (steepest ascent vs. conjugated gradient) on the predictive performance.

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.

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 $P(Y|X) = Z(X)^{-1} \exp \sum_{t=1}^{T} \Psi_t(y_t, X) \quad + \quad \Psi_{t-1,t}(y_{t-1}, y_t, X)$, where $\Psi_t(y_t, X)$ and $\Psi_{t-1,t}(y_{t-1}, y_t, X)$ are potential functions and $Z(X)$ is a normalization factor over all state sequences X so that each potential contributes overall probability.

Typically, it is assumed that the potentials factorize according to a set of features $\{f_k\}$, which are given and fixed, so that $\Psi(y_t, X) = \sum \alpha_k g_k(y_t, X)$ 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: $\Theta_m = \Theta_0 + \delta_1 + \ldots + \delta_m$ where $\delta_m = \eta_m - M \cdot \partial / \partial\Theta_{m-1} \sum_i \log P(y_i|x_i; \Theta_{m-1})$ is the gradient multiplied by a constant η_m, which is obtained by doing a line search along the gradient.

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

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

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 to these values, one gets an approximation of the expectation $E_{x,y}\left[\partial/\partial\Psi_{m-1}\right]$ 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.

In conjugate direction boosting (Kersting & Gutmann, 2006), 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. 1, 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 $d_t = \sum_{i=1}^{m} \beta_{i,m} \cdot \Delta_i$ where $\beta_{m,m} = 1$ and $\beta_{i,m} = \prod_{j=i+1}^{m} \beta_j$ if $i < m$.

3. Stratified Sampling

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). Although regression tree algorithms are very fast, they still must consider all training examples. 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 not a good idea 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 undesir-

Algorithm 1 Conjugated Gradient Tree Boosting with line search and sampling.

```
1:  function CGTreeBoost($Data, L$)
2:      for $1 \leq m \leq M$ do  ▷ Iterate Functional Gradient
3:          $S := \emptyset$
4:          for $1 \leq k \leq K$ do  ▷ Iterate through the class labels
5:              $(S_{Pos}, S_{Neg}) := (S_{Pos}, S_{Neg}) \cup$
6:                  GenExamples($k, Data, F_{m-1}, m$)
7:          $S_{Sample} :=$ Sample($(S_{Pos}, S_{Neg})$)
8:          $\Delta_m :=$ FitRelRegressTree($S_{Sample}, L$)
9:          if $m = 1$ then
10:             $d_1 = \Delta_1$  ▷ Initial conjugate direction
11:         else
12:             $\beta_m = \frac{\langle \Delta_m, \Delta_m - \Delta_{m-1} \rangle}{\langle \Delta_{m-1}, \Delta_{m-1} \rangle}$  ▷ Polak-Ribiére formula
13:             $d_m = \Delta_m + \beta_m \cdot d_{m-1}$  ▷ Next conjugate direction
14:         $\eta_m :=$ LineSearch($Data, F_{m-1}, d_m$)  ▷ Line Search along $d_m$
15:         $F_m := F_{m-1} + \eta_m \cdot d_m$  ▷ Model udpate
16:     return $F_M$  ▷ Return Potential
```

able property of uniform sampling.

To counter this *selection bias*, Kersting and Gutmann (2006) proposed a simulated annealing approach. The empirical frequency of observed regression examples is quadratically raised in early iterations. 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 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.

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.

4. 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 [npSeg(unk('rat7p')), vpSeg(cop('is')), vpSeg(v('located')), ppSeg(prep('at')), npSeg(art('the')), npSeg(adj('nuclear')), npS eg(unk('rim')),...] [protein, none, none, none, none, location, ...]

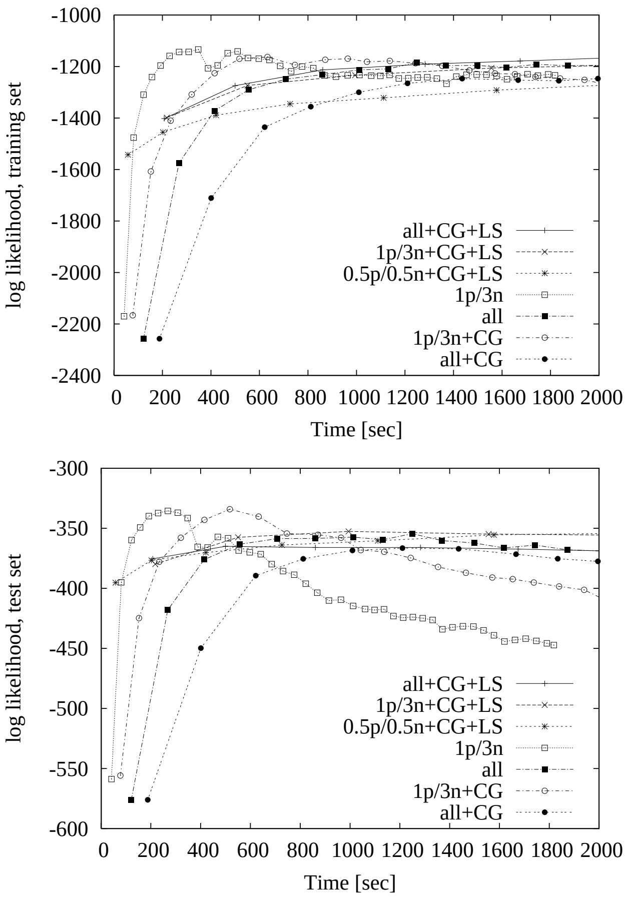

Figure 1. The log likelihood on training and test set; all: all regression examples were used, 1p/3n: all positive and 3 times more negative regression examples were sampled, 0.5p/0.5n: only half of the positive and half of the negative examples were sampled, CG: conjugated gradients were used, LS: a line search was used

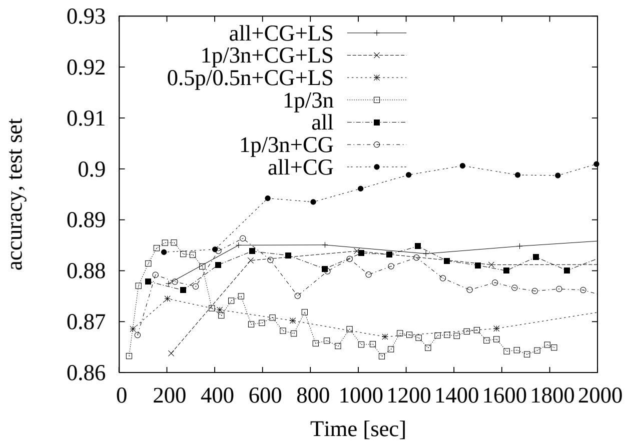

Figure 2. The accuracy on the test set.

We removed all training sequences that do not contain a protein or location tag. Because of memory limitations, we only took 25% of the resulting dataset, namely 195 input/output sequence pairs. 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 1 shows the results plotted against the training time. Strategy 1p/3n, which uses all positive and 3 times as many negative regression examples, 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.

Figure 2 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 first few iterations and afterwards all+CG outperforms all the other strategies. This supports our results from (Kersting & Gutmann, 2006). Furthermore, one can see that the line search does not really 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 shown as they follow the general picture: the less examples are used, the worse the results become.

5. Conclusions and Future Work

The experimental results show that a stratified sampling scheme can indeed speed-up computations and improve performance. It actually performs better than uniform sampling. More experiments on more data sets have to be done to confirm this result. It could also be interesting to stratify on each output class, not only on positive/negative examples.

Acknowledgments The authors thank Luc De Raedt for his support and Vítor Santos Costa for his help with the Prolog system YAP.

References

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*.

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).*

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).*