Unbiased Conjugate Direction Boosting for Conditional Random Fields

Kristian Kersting and Bernd Gutmann

University of Freiburg, Institute for Computer Science, Machine Learning Lab,
Georges-Koehler-Allee, Building 079, 79110 Freiburg, Germany
{kersting,bgutmann}@informatik.uni-freiburg.de

Abstract. Conditional Random Fields (CRFs) currently receive a lot of attention for labeling sequences. To train CRFs, Dietterich *et al.* proposed a functional gradient optimization approach: the potential functions are represented as weighted sums of regression trees that are induced using Friedman's gradient tree boosting method. In this paper, we improve upon this approach in two ways. First, we identify an expectation selection bias implicitly imposed and compensate for it. Second, we employ a more sophisticated boosting algorithm based on conjugate gradients in function space. Initial experiments show performance gains over the basic functional gradient approach.

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, speech recognition, activity recognition, information extraction. Consider the protein secondary structure prediction problem [10], where the task is to assign a secondary structure class to each amino acid residue in the protein sequence. This is an instance of the general sequence labeling problem: assign labels $Y = \langle y_1 \ldots y_n \rangle$ to a sequence $X = \langle x_1 \ldots x_n \rangle$ of objects.

One appealing approach to label sequences are Lafferty *et al.*'s *conditional random fields* (CRFs) [6]. They are undirected models encoding the conditional dependency $P(Y|X)$ and have outperformed HMMs [11] 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 great flexibility to include a wide array of features raises one important questions: where do the features and the parameters come from? In many practical situations, even extremely simple undirected models such as the linear-chain CRFs considered in the present paper models can have severe training costs. Early approaches built large sets of feature conjunctions according to hand-built, general heuristics. This can result in extremely large feature sets, with millions of parameters trained on hundreds of thousands of training examples; parameter estimation can literally take days [9]. Lafferty *et al.*'s [6] originally introduced iterative scaling algorithm for parameter estimation of a given CRF was reported to be exceedingly slow. Naive implementations of gradient ascent methods for maximizing the conditional likelihood $P(Y|X)$ have been proposed but they are typically quite slow too, because parameters highly interact. Therefore, it is not surprising that fast and integrated feature induction and parameter estimation techniques have been proposed.

McCallum's Mallet system [9] 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. Mallet's method is akin to the boosting idea [3] in that it creates new conjunctions (weak learners) based on a collection of misclassified instances, and assigns weights to the new conjunctions. Indeed, boosting has been applied to CRF-like models [1] and recently Dietterich *et al.* [2] presented a boosting approach, which is competitive to Mallet. It follows Friedman's gradient tree boosting algorithm [4], i.e., the potential functions are represented by sums of regression trees, which are grown stage-wise in the manner of Adaboost [3]. Each regression tree can be viewed as defining several new feature combinations, one corresponding to each path in the tree from the root to a leaf. Thus, the features can be quite complex; even relational conjunctions [5].

Despite elegancy, good performance, and flexibility, the gradient tree boosting approach, however, has two drawbacks. First, it imposes an expectation selection bias. In each boosting iteration, it generates functional gradient training examples for all possible label-label pairs at a sequence position. Thus, there are potentially quadratically many more expected than actually observed regression examples. We propose to quadratically raise the empirical frequency of observed label-label pairs. Second, only simple gradient ascent is employed so that maximization in one direction could spoil past maximizations. We propose the use of a more sophisticated boosting algorithm using conjugate directions. This way, we incorporate one of Mallet's major advantages into the functional gradient boosting approach: second-order information is used to adjust search directions so that previous maximizations are not spoiled. Experiments show for both modifications performance gains over the basic functional gradient approach.

We proceed as follows. After briefly reviewing CRFs and Dietterich *et al.*'s gradient tree boosting for training them, we show how to account for its expectation selection bias in Section 3. In Section 4, we devise the conjugate gradient variant. Before concluding, we experimentally evaluate our method in Section 5.

2 Linear-Chain CRFs

CRFs (see [6] for more details) are undirected graphical models that encode conditional probability distributions using a given set of features. In the present paper, we will focus on *linear-chain* CRF models.

Representation: 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. Now, they define the conditional probability of a state sequence given the observed sequence 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[1] and $Z(X)$ is a normalization factor over all state sequences X. Due to the global normalization by $Z(X)$, each potential has an influence on the overall probability.

Training: To apply CRFs, one most choose the representation for the potentials $\Psi_t(y_t, X)$ and $\Psi_{t-1,t}(y_{t-1}, y_t, X)$. 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. Furthermore, one must estimate the weights α_k, β_k. To do so, 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. The parameter in the next iteration are the current plus the gradient of the conditional likelihood function: $\Theta_m = \Theta_0 + \delta_1 + \ldots + \delta_m$ where $\delta_m = \eta_m - M \cdot \partial/\partial\Theta_{m-1} \sum_i \log P\left(y_i|x_i; \Theta_{m-1}\right)$ is the gradient multiplied by a constant η_m, which is obtained by doing a line search along the gradient.

Training via Gradient Tree Boosting: Dietterich *et al.*'s non-parametric approach interleaves both steps. 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 $E_{x,y}$. Dietterich *et al.* suggested to evaluate 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 (see [2, 5] for more details),

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

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

[1] 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.

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 the following, we will present two improvements of the boosting approach.

3 Expectation Selection Bias

Reconsider the functional gradient. The expectation basically consist of two terms. The first term is the expected value of the potential function under the empirical distribution and the second term, which arises from the derivative of the normalization constant $Z(X)$, is the expectation of the potential function under the current model distribution. Due to the second term, the set S of generated examples the number K of given classes. For example, if we have 2 classes and 100 training sequences of length 200, then the number of training examples is $2^2 \times 200 = 80,000$. For 4 classes, there are already $4^2 \times 200 = 320,000$ examples. Most of these examples are likely to have never been observed. In the worst case, for each observed y_{t-1}, y_t pair, we additionally generate $K^2 - 1$ expected examples. Thus, the regression tree learner is biased towards reducing the variance of expected functional gradient training examples.

This suggests to quadratically raise the empirical frequency[2] of observed examples in S. To do so, we augment each functional gradient training example $(((w_t(X_i), k', \Delta(k, k', t))$ with an additional frequency weight f. We set f to $K^2 - 1$ if $y_{t-1} = k'$ and $y_t = k$, and to 1.0 otherwise. Furthermore, as we get more confident of the estimated model with increasing numbers of iterations t, we impose a decay of f using the inverted logistic function $(K^2-2)\cdot\exp[-c\cdot(t-1)^b]+1$, where c and b are constants; in our experiments $C = 0.007$ and $b = 4$. Within the regression tree learner, these weights are treated strictly as example multipliers. That is, the weighted variance is identical to the same analysis when the examples are replicated the specific number f of time.

4 Conjugate Direction Boosting

Reconsider the basic gradient-ascent optimization approach. One of the problems with choosing the step size η_m doing a line search is that a maximization in one direction could spoil past maximizations. This problem is solved by conjugate gradient boosting methods [7, 8]. Conjugate gradient boosting methods compute so-called conjugate directions $d_1, d_2, \ldots$ 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.

More precisely, following Li *et al.*'s notation [7], conjugate gradient boosting iteratively performs two steps starting with setting the first direction to Δ_1:

[2] Weighting the empirical frequency has the appealing feature that it does not change the functional gradient values $\Delta(k, k', t)$. We only enforce a lower prediction variance over 'observed' training examples. This nicely fits our intuition that we are more confident in observed training examples in early iterations.

Algorithm 1 Conjugated Gradient Tree Boosting with line search.

```
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:             $S := S \cup$GenWeightedExamples($k, Data, F_{m-1}, m$)    ▷ Generate examples
6:         $\Delta_m :=$FitRelRegressTree($S, L$)                          ▷ Functional gradient
7:         if $m = 1$ then
8:             $d_1 = \Delta_1$                                      ▷ Initial conjugate direction
9:         else
10:            $\beta_m = \frac{\langle \Delta_m, \Delta_m - \Delta_{m-1} \rangle}{\langle \Delta_{m-1}, \Delta_{m-1} \rangle}$             ▷ Polak-Ribiére formula
11:            $d_m = \Delta_m + \beta_m \cdot d_{m-1}$                  ▷ Next conjugate direction
12:        $\eta_m :=$LineSearch($Data, F_{m-1}, d_m$)                    ▷ Line Search along $d_m$
13:        $F_m := F_{m-1} + \eta_m \cdot d_m$                              ▷ Model udpate
14:    return $F_M$                                                   ▷ Return Potential
```

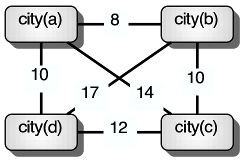
1. **(Conjugate directions)** Given the current gradient Δ_m, compute the empirical angle β_m between Δ_m and Δ_{m-1} on the training examples. The current gradient plus the old weighted gradient multiplied by the calculated angle is added to the current model, $d_m = \Delta_m + \beta_m \cdot d_{m-1}$. The angle β_m can be calculated by evaluating the Polak-Ribiére formula $\beta_m = \frac{\langle \Delta_m, \Delta_m - \Delta_{m-1} \rangle}{\langle \Delta_{m-1}, \Delta_{m-1} \rangle}$ 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 $i < m$.
2. **(Line search)** Compute the next model F_m by maximizing along the direction of d_m, i.e.

$$F_m = \sum_{k=1}^{m} \eta_k \cdot d_k = \sum_{i=1}^{m} \left(\sum_{j=1}^{m} \eta_j \cdot \beta_{i,j} \right) \Delta_i \; .$$

Training CRFs using conjugate gradient boosting is realized in CGTreeBoost in Alg. 1. Note that it uses the example weighting scheme discussed in the previous section. Furthermore, for the sake of simplicity and to stay close to traditional conjugate gradient for function optimization, we present the model update step as simply adding a single function. In fact, d_m is a linear combination of all previously computed functional gradients and one needs to keep track of the $\beta_{i,m}$. For a detailed discussion, we refer to [7, 8].

5 Preliminary Experiments

We implemented our approach in Yap 5.1.0 Prolog and investigate the following two questions: **Q1** *Can unbiased boosting speed-up the training of CRFs, i.e., faster improvements of the objective score, which is the conditional log-likelihood?* **Q2** *Can conjugate gradient boosting improve the training of CRFs?* To answer both questions, we carried out experiments on two domains, traveling salesman and protein secondary structure.



act	1	2	3	4	5	6	7	8
`city(a)`	-	7	2	1	10	-	-	2
`city(b)`	11	-	3	-	5	-	-	-
`city(c)`	12	8	-	-	5	8	10	3
`city(d)`	13	9	-	-	5	8	-	10

	Extra
`city(a)`	22
`city(b)`	50
`city(c)`	12
`city(d)`	10

Fig. 1. Traveling salesman instance used in the experiment.(Left) The map where nodes denote cities and edges transitions with associated costs. (Middle) Costs of activities. (Right) Costs of doing an activity fast in one city.

(Traveling Salesman ^{3}**)** There are n cities and actions $\mathcal{A}$, which can be done with normal speed or fast. All these actions have different costs. The task is, given is a sequence of activities $a_1, \ldots, a_T$, find a sequence of cities $c_1, \ldots, c_T$ such that the overall costs are minimized. We considered the instance with 4 cities and 8 actions as described in Figure 1. We randomly generated 100 independent activity sequences of length 15 and searched brute force for an optimal travel sequence. This yield relational sequences such as $X = \langle \texttt{act}(4, \texttt{normal}), \texttt{act}(1, \texttt{fast}), \texttt{act}(8, \texttt{normal}), \texttt{act}(7, \texttt{normal}), \ldots \rangle$ and $Y = \langle \texttt{city}(\texttt{a}), \texttt{city}(\texttt{d}), \texttt{city}(\texttt{c}), \texttt{city}(\texttt{c}), \ldots \rangle$. We refer to [5] for more details.

On a random 92/8 training/test split, we ran conjugate gradient tree (CGT) and gradient tree (GT) boosting with and without (,i.e., $\eta_m = 1.0$) line search on biased and unbiased examples. Figure 2 summarizes the results. As it can readily be seen, both algorithms overfit on this data set when doing a line search, cf. Fig 2 **(b)** and **(d)**. However, compensating for the expectation selection bias, cf. Fig 2 **(d)**, prevents CGT in contrast to GT from overfitting. Without line search, cf. Fig 2 **(a)** and **(c)**, CGT yields better performance than GT. This affirmatively answers **Q2**. More over, compensating for the expectation selection bias, cf. Fig. 2 **(c)**, results in larger improvements in early iterations of CGT and again prevents CGT from overfitting. This affirmatively answers **Q1**. The accuracies on the test set were in the same range across all algorithms.

(Protein Secondary Structure) This propositional data set was originally published by Qian and Sejnowski [10]. A protein consists of a sequence of amino acid residues. Each residue is represented by a single feature with 20 possible values (corresponding to the 20 standard amino acids). There are three classes: alpha helix, beta sheet, and coil (everything else). There is a training set of 111 sequences and a test set of 17 sequences. We used the same set up as in [5] and ran several variants of the basic algorithms.

The results summarized in Figure 3 **(a)** support the results of the first experiment: CGT performs better than GT and unbiased variants better than biased ones. Although doing a line search did not yield overfitting, unbiased CGT without line search still achieved the best predictive performance, partic-

^{2} This domain has originally been used in [5] to show that CRFs for sequences of relational symbols can significantly outperform CRFs for sequences of flat symbols.

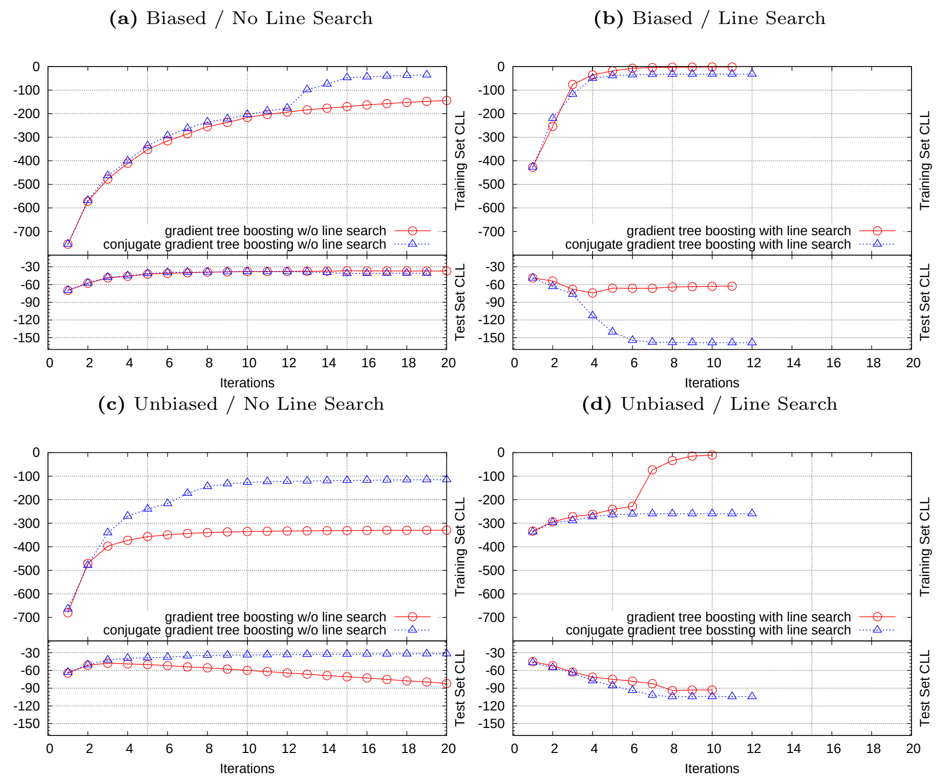


Fig. 2. Learning curves on the job scheduling domain. **(c, d)** compensated for the expectation selection bias in contrast to **(a, b)**. Cases **(b, d)** used a line search whereas **(a,c)** did not

ularly compared to the original GT, cf. Figure 3 **(b)**. This affirmatively answers **Q1** and **Q2**.

6 Conclusions

Training CRFs can be viewed as gradient ascent in function space. In this paper, we devised a novel algorithm for training CRFs, which employs conjugate directions in function space. Furthermore, we identified an expectation selection bias when training CRFs and presented an example weighting approach to compensate for it. Preliminary experiments are encouraging: the resulting approach can indeed perform better than simple gradient tree boosting. To validate this, further experiments should be conducted.

Acknowledgments The authors thank Luc De Raedt for his support, Alan Fern for interesting discussions, and Tom Dietterich for providing his version of

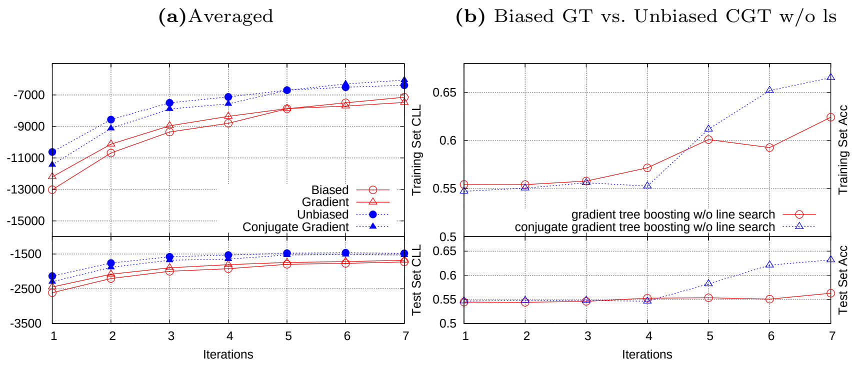


Fig. 3. Learning curves on the protein secondary structure domain. **(a)** Average over several variants of biased/unbiased and GT/CGT. **(b)** Predictive accuracies of unbiased CGT and biased GT both w/o line search.

Qian and Sejnowski's benchmark. The research was supported by the European Union IST programme: FP6-508861, *Application of Probabilistic ILP II*.

References

1. Y. Altun, T. Hofmann, and M. Johnson. Discriminative learning for label sequences via boosting. In *Advances in Neural Inf. Proc. Systems (NIPS-15)*, 2003.
2. T. Dietterich, A. Ashenfelter, and Y. Bulatov. Training conditional random fields via gradient tree boosting. In *Proc. 21st International Conf. on Machine Learning*, pages 217–224. ACM, 2004.
3. Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *In Proceedings of ICML-96*, pages 148–156. Morgan Kaufman, 1996.
4. J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 29, 2001.
5. B. Gutmann and K. Kersting. TildeCRF: Conditional Random Fields for Logical Sequences. In *Proc. of the 17th European Conference on Machine Learning (ECML-06)*, 2006. (To appear).
6. J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th Int. Conf. on Machine Learning (ICML-01)*, pages 282–289, 2001.
7. L. Li, Y. S. Abu-Mostafa, and A. Pratap. CGBoost: Conjugate Gradient in Function Space. Technical Report CaltechCSTR:2003.007, Calfornia Institute of Technology, August 2003.
8. R. W. Lutz and P. Bühlmann. Conjugate direction boosting. *Journal of Computational and Graphical Statistics*, 15(2):287–311, 2006.
9. A. McCallum. Effciently inducing features of conditional random fields. In *Proc. of the 21st Conference on Uncertainty in Artificial Intelligence (UAI-03)*, 2003.
10. N. Quian and T. J. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *JMB*, 202:865–884, 1988.
11. L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–285, 1989.