Sequential Supervised Learning and Conditional Random Fields

by

Adam J. Ashenfelter

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented December 4th, 2003
Commencement June 2004

Master of Science thesis of Adam J. Ashenfelter presented on December 4th, 2003.

APPROVED:

Thomas G. Dietterich, representing Computer Science

Head of the Computer Science department

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Adam J. Ashenfelter, Author

AN ABSTRACT OF THE THESIS OF

Adam J. Ashenfelter for the degree of Master of Science in Computer Science presented on December 4th, 2003.

Title: Sequential Supervised Learning and Conditional Random Fields.

Abstract approved

Thomas G. Dietterich

Supervised learning is concerned with discovering the relationship between example sets of features and their corresponding classes. The traditional supervised learning formulation assumes that all examples are independent from one another. The order of the examples contains no information. Nonetheless, many problems have a sequential nature. Classifiers for these problems must use the sequence of the examples, as well as the regular features, when generating a prediction. Sequential supervised learning (SSL) algorithms are able to capture both types of information.

A variety of sequential supervised learning methods have already been developed. The conditional random field is a particularly robust SSL algorithm that overcomes some of the limitations of other SSL methods, such as hidden Markov models and recurrent sliding windows. However, the original implementation of the conditional random field explicitly represents a weight for every feature combination. For sequential problems using a window of input features, this can mean a combinatorial explosion. A better way to represent the conditional random field, using regression trees, is introduced. The regression tree conditional random field provides an efficient method for learning feature weights and supports the selection and combination of features. The ability to select feature combinations benefits classification performance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

The author expresses sincere appreciation to Thomas G. Dietterich for the extensive help and guidance he provided during this research. The author also wishes to acknowledge the suggestions and assistance from Diane Damon and Saket Joshi while writing this thesis.

CHAPTER 1

INTRODUCTION

Supervised learning is a methodology that strives to discover the relationship between a set of observations and their corresponding classifications. There are a wide variety of applications for supervised learning, such as speech recognition and medical diagnosis. The basic goal is to learn a function $y = F(x)$, where we provide some input x and receive an answer y. For a task such as face recognition, the input would be an image of someone's face while the classification is the individual's identity. A supervised learning algorithm is trained to make these predictions from a set of examples of the form (x, y). The better a supervised learning algorithm, the more successfully it generalizes from training data and correctly predicts for new queries.

Traditional supervised learning assumes that each example is independent from all others. The order in which the examples exist in the training set has no importance. While true for many problems, not everything fits so nicely into this framework. Consider predicting the pronunciation of an English word, or the problem of diagramming a sentence. Each task can be represented as a sequence of observable features $(x_1, x_2, x_3, ..., x_L)$ for which a series of classes $(y_1, y_2, y_3, ..., y_L)$ must be predicted. The letters of a word correspond to a series of phonemes and stresses. The words in a sentence correspond to a chain of nouns, verbs, etc.

Tasks with a sequential nature can pose a problem for standard supervised learning. To illustrate this, imagine diagramming sentences, also called part-of-speech tagging. Many words can be diagrammed easily, requiring no context from a sentence. "Shoe" is a noun. "Swim" is a verb. Other words are not as simple. Consider the sentences "He will lead the way" and "His feet are like lead". "Lead" can either be a noun or a verb. The part of speech cannot be predicted without taking into account the surrounding words. In this situation, the independence assumption of traditional supervised learning is not valid. The sub-field of sequential supervised learning (SSL) focuses on ways to solve sequential problems, such as part-of-speech tagging.

There are a number of existing methods for sequential supervised learning. Common approaches are sliding windows, recurrent sliding windows, and hidden Markov models. Newer algorithms include maximum entropy Markov models, and conditional random fields. As always in machine learning, each approach has its own strengths and weaknesses. This thesis explores some of the techniques for sequential supervised learning, and introduces an improvement for conditional random fields.

1.1 Sliding Windows and Recurrent Sliding Windows

Sliding window methods are a simple and popular method for sequential problems. They offer the ability to convert an SSL problem into a format for which any traditional supervised learning algorithm may be applied. Given a sequence of L observable features $\langle x_1, x_2, x_3, \ldots, x_L \rangle$ and a sequence of corresponding classes $\langle y_1, y_2, y_3, \ldots, y_L \rangle$, a window of size d (also known as input context) will slide across the feature sequence, generating an independent example for each class. Given a window of size 3, a sliding window would generate examples $(\langle B, x_1, x_2 \rangle, y_1)$, $(\langle x_1, x_2, x_3 \rangle, y_2)$, $(\langle x_2, x_3, x_4 \rangle, y_3)$, $\ldots$, $(\langle x_{L-1}, x_L, B \rangle, y_L)$, where B symbolizes a null value that pads the ends of our sequence.

The spatial context of the feature windows generated by a sliding window captures some sequential information. However, recurrent sliding windows (RSW) offer a better way to handle sequential data. Very similar to the sliding window, the RSW adds the n previously predicted y's (also known as output context) as features of the current window position. Given an input context of 3 and an output context of 1, a RSW would generate examples $(\langle B, x_1, x_2, B \rangle, y_1)$, $(\langle x_1, x_2, x_3, y_1 \rangle, y_2)$, $(\langle x_2, x_3, x_4, y_2 \rangle, y_3)$, $\ldots$, $(\langle x_{L-1}, x_L, B, y_{L-1} \rangle, y_L)$. During training, the correct y values are used as the output context.

A pitfall of recurrent sliding windows is that sequential information is propagated in only one direction. To clarify, try pronouncing "photograph" and "photography". The pronunciation of the words differ entirely according to the ending "y". Now imagine using a left-to-right RSW with a window size of 7. To the recurrent sliding window, both words

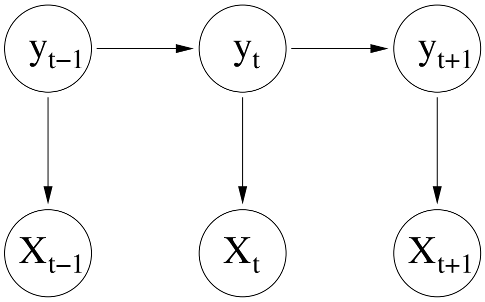

Figure 1: Hidden Markov Model

appear exactly the same until the "y" position moves into the window. At this point, the window is focused on "a" with an input context of $\langle o, g, r, a, p, h, y \rangle$. The phonemes and stresses have already been predicted for every position left of "a". Once the "y" (or lack thereof) is discovered, it is already too late to accurately choose the correct pronunciation at the beginning of the word. In short, a sliding window method must choose between pronouncing "photograph" and "photography" before it knows which word it is observing. A large enough window size could cope with this, but introduces new difficulties. The larger the window size, the more input features for a learning algorithm. With too many inputs, a learning algorithm can have trouble discriminating between informative features and non-informative, or noisy, features. This can hurt the algorithm's ability to generalize effectively.

1.2 Hidden Markov Models

The hidden Markov model (HMM), illustrated in figure 1, is a probabilistic model describing how sequences of observations and class labels are generated. The model assumes that the observations at time t in the sequence are generated according to the class at time t. The class label at time t is generated according to the previous class label, at time $t-1$.

To represent this model, the probabilities $P(y_1|start)$, $P(y_t|y_{t-1})$, and $P(x_t|y_t)$ must be learned. Fortunately, training is simple. The probabilities for the model can be found by counting the frequency of every possible pair of (y_t, y_{t-1}) and (y_t, x_t).

Given only a sequence of observations, a Hidden Markov model can find the most likely class label at any time t through using the forward-backward algorithm, also known as the

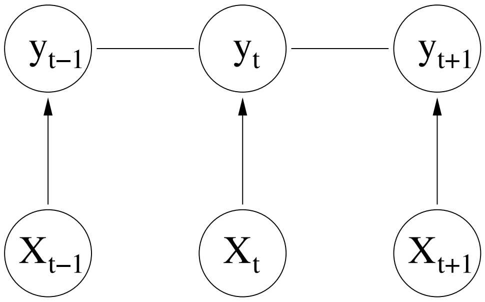

Figure 2: Conditional Random Field

Baum-Welch algorithm (see section 1.3.2). Alternatively, the HMM can find the overall most likely sequence of class labels by searching over all possible sequences with the Viterbi algorithm (see section 1.3.3). Both methods are dynamic programs which carry information in each direction over the sequence, therefore avoiding the problem inherent in recurrent sliding windows.

While hidden Markov models work well for many applications, the model is limited by its assumptions. If long distance relationships in the data exist (e.g. y_{t-5} influences y_t), the Markov property prevents capturing the relationship. There may also be overlapping features, where x_t is influenced by more than just y_t. A larger input context can cope with this, but it requires learning much larger probability distributions. For a input window of size 3, we must learn $P(x_{t-1}, x_t, x_{t+1}|y_{t-1}, y_t, y_{t+1})$ instead of $P(x_t|y_t)$. This increase in the order of the model can quickly make the HMM computationally impractical.

1.3 Conditional Random Fields

Introduced by Lafferty, McCallum, and Pereira [10], the conditional random field (CRF) is a Markov random field where y-to-y relationships are conditioned by the observable x's. With the y transitions conditioned on x , the model can accept an arbitrary set of input features. Overlapping features or features that capture long distance relationships do not pose the problems they did for hidden Markov models.

$$M_t(y_t, y_{t-1}|w_t) = \exp\left(\sum_{\alpha} \lambda_\alpha f_\alpha(y_t, y_{t-1}, w_t) + \sum_{\beta} \mu_\beta g_\beta(y_t, w_t)\right) \tag{1}$$

Conditional random fields are represented as a set of potentials, $M_t(y_t, y_{t-1}|w_t)$. We denote the window of input features at time t as w_t. The function $f(y_t, y_{t-1}, w_t)$ generates a matrix of boolean features related to the interactions between y_t, y_{t-1}, and w_t. The second function, $g(y_t, w_t)$, generates a matrix of boolean features regarding the y_t to w_t relationship. The learned parameters of the CRF are represented by the matrices λ and μ. The λ matrix consists of a weight for every element in $f(y_t, y_{t-1}, w_t)$, while μ consists of a weight for every $g_\beta(y_t, w_t)$ element.

The probability of a label sequence Y given a feature sequence X is calculated using the potentials. Let L be the length of the sequence, y_0 is a special starting state, and y_{l+1} is a stopping state.

$$P(Y|X) = \frac{\prod_{t=1}^{L+1} M_t(y_t, y_{t-1}|w_t)}{Z} \tag{2}$$

Z is a normalizer, the total of all potentials,

$$\begin{aligned} Z &= \left(\prod_{t=1}^{L+1} M_t(w_t)\right)_{start,stop} \\ &= (M_1(w_1) * M_2(w_2) * M_3(w_3) \cdots M_L(w_L) * M_{L+1}(w_{L+1}))_{start,stop}. \end{aligned}$$

After finding the matrix product of all M_t potential matrices, Z is equal to the element that represents the potential for beginning at the start state at $t = 0$ and finishing at the stop state at $t = L + 1$.

1.3.1 Training the CRF

In the original CRF formulation, iterative scaling was used to learn the weights for λ and μ. To speed up training time, we implemented a version of conditional random fields with gradient descent. Let ρ be the scalar step size for a gradient update. After each iteration, the weight matrices are updated using:

$$\lambda_i = \lambda_i + (\rho * \partial\lambda_i)$$

$$\mu_i = \mu_i + (\rho * \partial\mu_i)$$

Let N be the entire set of training sequences, and K is the set of all possible classes. $\hat{y}_t$ and $\hat{y}_{t-1}$ denote the true classes from the current training sequence at time t and $t - 1$.

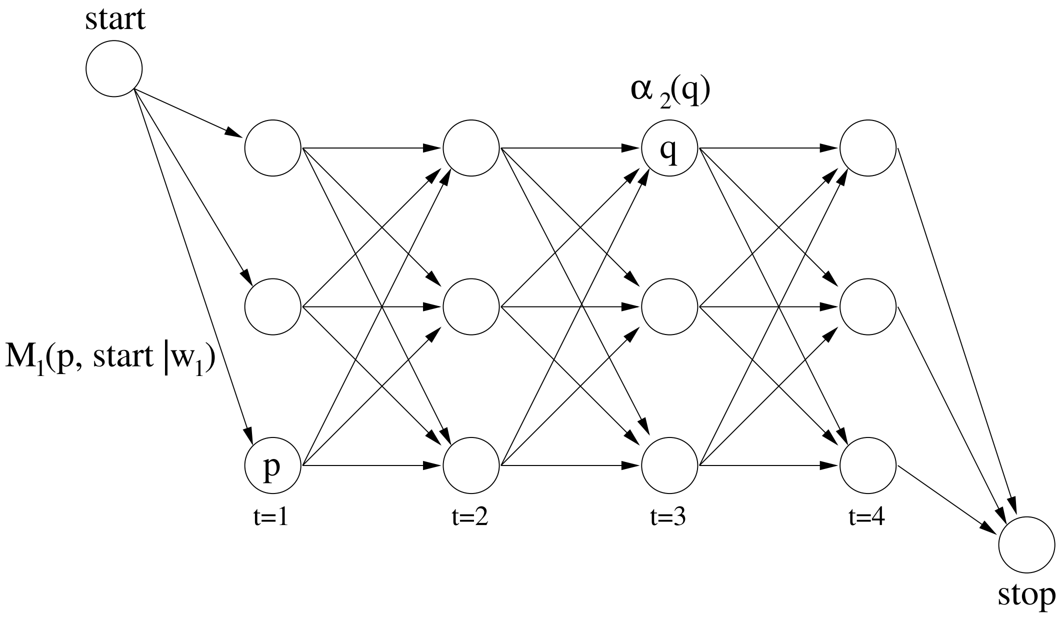

Figure 3: Forward Pass of the Forward-Backward Algorithm

The gradients are calculated by:

$$\partial\lambda = \sum_{N}\sum_{t=1}^{L}\left(f(\hat{y}_t, \hat{y}_{t-1}, w_t) - \sum_{p,q\in K} f(q,p,w_t) * P(y_t = q, y_{t-1} = p|w_t)\right)$$

$$\partial\mu = \sum_{N}\sum_{t=1}^{L}\left(g(\hat{y}_t, w_t) - \sum_{q\in K} g(q,w_t) * P_t(y_t = q|w_t)\right)$$

The optimal step size, ρ, for the gradient update can be found using a line search over the training data, or a hold out dataset. The probabilities used in the gradient updates are computed using the forward-backward algorithm, described in the next section.

1.3.2 Forward-Backward Algorithm

The forward-backward algorithm uses the potentials to calculate the flow of probability over all possible classes at each time step in a sequence. Two passes, forward and backward, ensure that sequential information flows in both directions. After the forward and backward passes, the potential values are normalized to find the final probabilities.

A common way to visualize this algorithm is as a graph of the possible classes at each time step in a sequence. This is referred to as the trelice. Figure 3 depicts the forward pass of the algorithm over the trelice. There are three classes in this example, excluding the special start and stop states. The columns correspond to the time steps, and the nodes correspond to classes. Each edge represents a potential (e.g. $M_1(p, start|w_1)$).

The value $\alpha_t(y)$ symbolizes the sum of potentials flowing into class y at time t. When starting the forward pass, the initial alpha vector is:

$$\alpha_0(y) = \{^{1 \;\; if \; y=start}_{0 \;\; otherwise}.$$

The recurrence for computing the alphas over the sequence:

$$\alpha_t = \alpha_{t-1} M_t(w_t),$$

which is equivalent to:

$$\alpha_t(y) = \sum_{i \in K} \alpha_{t-1}(i) M_t(i, y|w_t).$$

Similarly, the initial beta vector:

$$\beta_{L+1}(y) = \{^{1 \;\; if \; y=stop}_{0 \;\; otherwise}.$$

The recurrence computing the betas over the sequence:

$$\beta_t^T = M_{t+1}(w_{t+1}) \beta_{t+1},$$

which is equivalent to:

$$\beta_t(y) = \sum_{i \in K} M_{t+1}(y, i|w_{t+1}) \beta_{t+1}(i).$$

After completing both passes, we can compute the probability for any state at any time in the sequence:

$$P(y_t = q|w_t) = \frac{\alpha_t(q)\beta_t(q)}{Z}, \tag{3}$$

where Z is the normalizer found through summing the alpha-beta values of each class at any time t,

$$Z = \sum_{i \in K} \alpha_t(i)\beta_t(i).$$

For training a conditional random field we must also compute the probability of a transition between specific states, $P(y_t, y_{t-1}|w_t)$. Shown in figure 4, this is found with:

$$P(y_t = q, y_{t-1} = p|w_t) = \frac{\alpha_{t-1}(p) M_t(q, p|w_t) \beta_t(q)}{Z}. \tag{4}$$

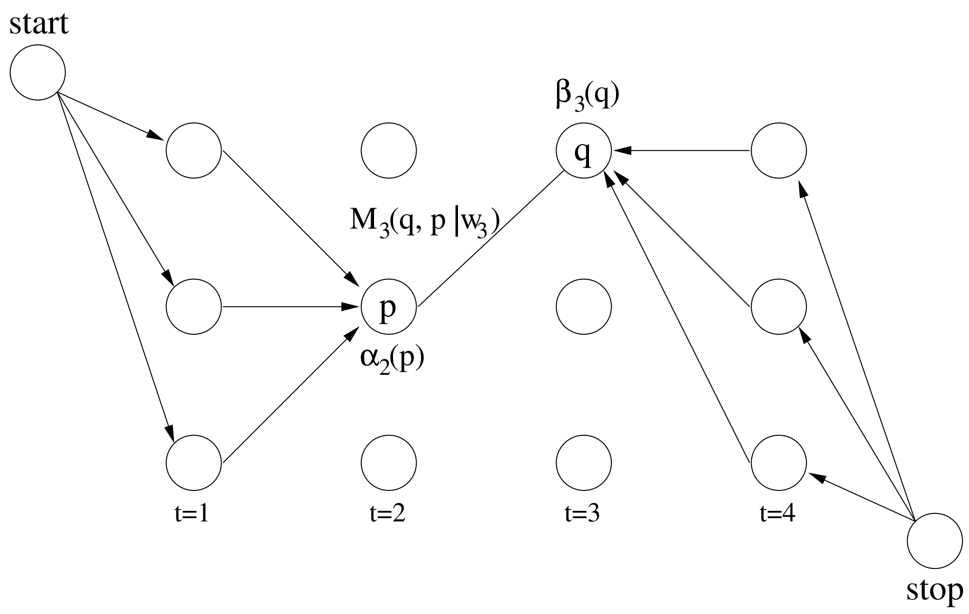

Figure 4: Finding a Transition Probability

We note that the normalizer can also be found by summing the entire potential flow between the classes at any time $t-1$ and t,

$$Z = \sum_{i,j \in K} \alpha_{t-1}(i) M_t(j, i|w_t) \beta_t(j).$$

While the forward-backward algorithm is used during training of conditional random fields, it is also useful for classification of new data. Since the forward-backward algorithm takes the CRF's potentials and finds $P(y_t = u|w_t)$, we can predict the most likely class at any time step t. This is valuable if we are interested in maximizing the number of individual classes predicted correctly. If instead we want to maximizing the probability of predicting an entire sequence correctly, we use the Viterbi algorithm.

1.3.3 Viterbi Algorithm

Similar to the forward-backward algorithm, Viterbi searches for the maximum transition into each state y_t instead of summing the potentials. Sweeping across the trelice, the algorithm calculates the most probable path from the start state to any y_t. When Viterbi reaches the end of the sequence, it has found the overall most likely classification sequence.

Initializing the Viterbi algorithm:

$$\gamma_0(y) = \{ {}^{1 \;\; if\ y=start}_{0 \;\; otherwise} .$$

The recurrence, calculating the best path to y_t:

$$\gamma_t(y) = \max_i \gamma_{t-1}(i) M_t(y, i|w_t).$$

The forward-backward and Viterbi algorithms are computationally efficient. Although there are K^L possible classification sequences, the algorithms run in $O(L * K^2)$.

1.3.4 Normalization

SSL problems with long sequence lengths pose a computational problem for the forward-backward and Viterbi algorithms. The length of a protein secondary structure sequence may extend up to 500 time steps. The potential values across a long sequence, $\prod_{t=1}^{L+1} M_t(y_t, y_{t-1}|w_t)$, can grow prohibitively large. A solution is to normalize during the recurrence:

$$\hat{\alpha}_t(y) = \frac{\alpha_t(y)}{Q_t}.$$

Q_t is the sum of all α_t:

$$Q_t = \sum_i \alpha_t(i).$$

The original α_t is found in the regular fashion, but using the previously calculated $\hat{\alpha}_{t-1}$,

$$\alpha_t(y) = \sum_{i \in K} \hat{\alpha}_{t-1}(i) M_t(i, y|w_t).$$

The backward pass calculates $\hat{\beta}$ values in the same way as $\hat{\alpha}$. Probabilities are calculated using:

$$\begin{aligned} P(y_t = q|w_t) &= \frac{\hat{\alpha}_t(q)\hat{\beta}_t(q)}{\hat{Z}_t} \\ P(y_t = q, y_{t-1} = p|w_t) &= \frac{\hat{\alpha}_{t-1}(p) M_t(q, p|w_t)\hat{\beta}_t(q)}{\hat{Z}_{t,t-1}} \end{aligned}$$

The probability normalizers are now be calculated at every time across the sequence:

$$\begin{aligned} \hat{Z}_t &= \sum_{i \in K} \hat{\alpha}_t(i)\hat{\beta}_t(i) \\ \hat{Z}_{t,t-1} &= \sum_{i,j \in K} \hat{\alpha}_{t-1}(i) M_t(j, i|w_t)\hat{\beta}_t(j) \end{aligned}$$

1.3.5 Performance of the CRF

Lafferty, McCallum, and Pereira [10] compared the performance of the conditional random fields, hidden Markov models, and maximum entropy Markov models on the part-of-speech tagging problem. The maximum entropy Markov model (MEMM) was a precursor to the CRF [13].

In one experiment, the classifiers all used the same feature configuration and were tested on sentences that only included words seen in training sentences (in vocabulary). In another experiment, they added spelling related features that could not be used by the HMM because of its restrictions. They then tested the learning algorithms on sentences that included previously unseen words (out of vocabulary).

	In Vocabulary Error	Out of Vocabulary Error
HMM	5.69%	45.99%
MEMM	4.81%	26.99%
CRF	4.27%	23.76%

Table 1: HMM, MEMM, and CRF results on the part-of-speech tagging problem.

The conditional random field shows strong results. This is a reflection of the CRF's abilities to incorporate any arbitrary feature and use information from over the entire sequence for classification. CRFs have also succeded in noun phrase segmentation [18], and table extraction from documents [14]. The CRF appears to be an elegant solution to the problems posed in sequential supervised learning. Nonetheless, the CRF still struggles when faced with certain tasks.

The CRF uses the λ weight table to learn the probability of class transitions given some features, $P(y_t, y_{t-1}|w_t)$. To learn this, λ must contain a weight for every combination of transitions and features. The table must be of size $(K * K * F)$, where K is the number of classes and F is the number of features.

For the basic representation for a problem with 3 classes and 20 possible feature values, λ will have $(3 * 3 * 20)$ elements. The drawback to this formulation is that only the current feature, x_t, influences the class transition. It is often more useful to use a window of input features rather than a single input feature ($w_t = \langle x_{t-1}, x_t, x_{t+1} \rangle$ instead of $w_t = \langle x_t \rangle$). Given a window size d, there are F^d possible feature combinations. This means, using a window of input features, λ will be of size $(K * K * F^d)$.

Window size	1	3	5	7
λ table size	180	72000	$28.8 * 10^6$	$11.52 * 10^9$

Table 2: The λ weight table size as the window of input features increase.

As seen in table 2, λ grows rapidly when increasing the input window size for a problem with 3 classes and 20 features. There are two main disadvantages to a large λ. First, the CRF can become computationally infeasible, both in time and memory, if the weight table grows too large. Second, with too many weights there will not be enough data to accurately learn each value. The CRF can overfit the training data and lose the ability to generalize to new data.

The core problem is that of feature selection and combination. We want the ability to express *any* informative combination of features in our input window, yet we do not want to express *every* combination. In other words, we need a compact way to represent the important areas of the λ table. Our solution to this problem is a variant of CRF that uses regression trees to learn the equivalent of the λ weights.

CHAPTER 2

A TREE BOOSTED CRF

We introduce a new implementation of the conditional random field which uses gradient descent and boosted regression trees to train the classifier. This approach retains the benefits of the table-based CRF while the regression trees support a method for feature selection and combination. The regression tree CRF can make more accurate predications, as the trees help prevent overfitting to the training data. However, the primary motivation for the new algorithm is the dramatic computational speed-up when compared to an equivalent table-based CRF.

Window size	1	3	5	7
Table-based CRF	0.04	0.66	41.2	1505
Regression Tree CRF	0.8	1.11	1.2	1.4

Table 3: Training itertation run time (in seconds) for table and tree CRFs.

Table 3 compares the average run time per training iteration of the table-based CRF and the regression tree CRF for various feature window sizes. The CRFs were trained on a reduced version of the protein secondary structure dataset (section 3.3) which uses only five feature values, as opposed to the original 20 amino acids. The regression trees were limited to thirty leaves. The run times were recorded on a 2.0 GHz Intel Celeron machine.

2.1 Algorithm Overview

The gradient boosted regression tree CRF is adapted from the $L_K\_TreeBoost$ algorithm by Friedman [8]. The fundamental idea is that our potential functions are now represented by a set of regression trees F_y,

$$M_t(y_t, y_{t-1}|w_t) = \exp F_{y_t}(y_{t-1}, w_t).$$

Table 4: Pseudo-Code for Training a Boosted Regression Tree CRF

```
Let M be number of iterations
Let S be number of training sequences
Let T_s be the length of sequence s
Let K be number of classes

F_q, 0(w) = 0, q = 1, K
For m = 1 to M :
    Find P_s(Y_s|X_s) using Forward-Backward,   s = 1, S
    For q = 1 to K :
        For p = 1 to K :
            For s = 1 to S :
                For t = 1 to T_s :
                    ψ_{q,p,s,t} = [q = y_t, p = y_{t-1}] − P_s(q, p|w_t)
                    Generate a target example for H_q : (⟨w_{s,t}⟩ , ψ_{q,p,s,t})
                End For
            End For
        End For
        Fit the regression tree H_q to all the target examples (⟨w⟩ , ψ_q)
        F_{q,m}(w) = F_{q,m-1}(w) + H_q(w)
    End For
End For
```

At every iteration, we grow new trees to correct for the error of the existing regression trees and add them into the classifier. This is the boosting process [9].

As with the regular CRF, the potentials are used to calculate probabilities of classes and transitions over a sequence. With these probabilities, we can find the gradients ψ of our potential functions, F_y. In the original CRF, the gradients $\partial\lambda$ are used to update the weight table λ. However, for the tree CRF we will fit a new set of regression trees H_y to the gradients ψ. H_y consists of one regression tree for each class. The trees, H_y, can be thought of as the representing the corrections for our current trees F_y. The update for F_y is carried out by adding the outputs of H_y,

$$F_y = F_y + H_y.$$

The update concludes one training iteration, which can be repeated to further lower the error of the probabilities calculated on the training data. Table 4 gives the pseudo-code for training a regression tree CRF.

Figure 5 shows the $\partial\lambda_y$ tables and the corresponding H_y regression trees for a three class problem with three possible feature values. The $\tilde{H}_y$ coloumn depicts the table equivalent of the H_y regression trees. Essentially, the H_y trees are a method to compactly generalize $\partial\lambda_y$ tables. The larger we grow the regression trees, the more closely they will resemble $\partial\lambda_y$.

2.2 Regression Trees

A regression tree predicts continuous numbers given some input features [2]. The trees are grown according to a set of examples of the form $(\langle x_1, x_2, \ldots, x_n \rangle, \psi)$, where ψ is a real valued target that represents a functional gradient.

The regression tree starts as a single leaf node whose output, γ, is the average of all the ψ targets,

$$\gamma = \frac{\sum_{i=1}^{C} \psi_i}{C}.$$

C is the number of examples in the data set. To grow the tree, a leaf node is selected to become a new split. The new split node will test some feature from the feature window, x_k. Two leaf nodes are added as children of the split node, one representing the case where $x_k = true$ and the other where $x_k = false$. We divide the dataset according to x_k. The output of the 'true' leaf node will be the average of the targets for the 'true' dataset,

$$\gamma_{x_k=true} = \frac{\sum_{i=1}^{C_{x_k=true}} \psi_{i,x_k=true}}{C_{x_k=true}}.$$

Conversely, the output of the 'false' leaf node is the average of the targets for the 'false' dataset,

$$\gamma_{x_k=false} = \frac{\sum_{i=1}^{C_{x_k=false}} \psi_{i,x_k=false}}{C_{x_k=false}}.$$

To choose which leaf node to split, the regression tree will greedily choose the split that minimizes the squared error on the training data. The squared error for leaf l is found by summing the squared difference between l's output and the targets of the training data classified by l. Let σ_l be the squared error at l:

$$\sigma_l = \sum_{i=1}^{C_l} (\psi_{l,i} - \gamma_l)^2.$$

For every possible leaf node and every feature, we search for the greatest difference of squared error between the candidate splitting node and its children. We will split at the

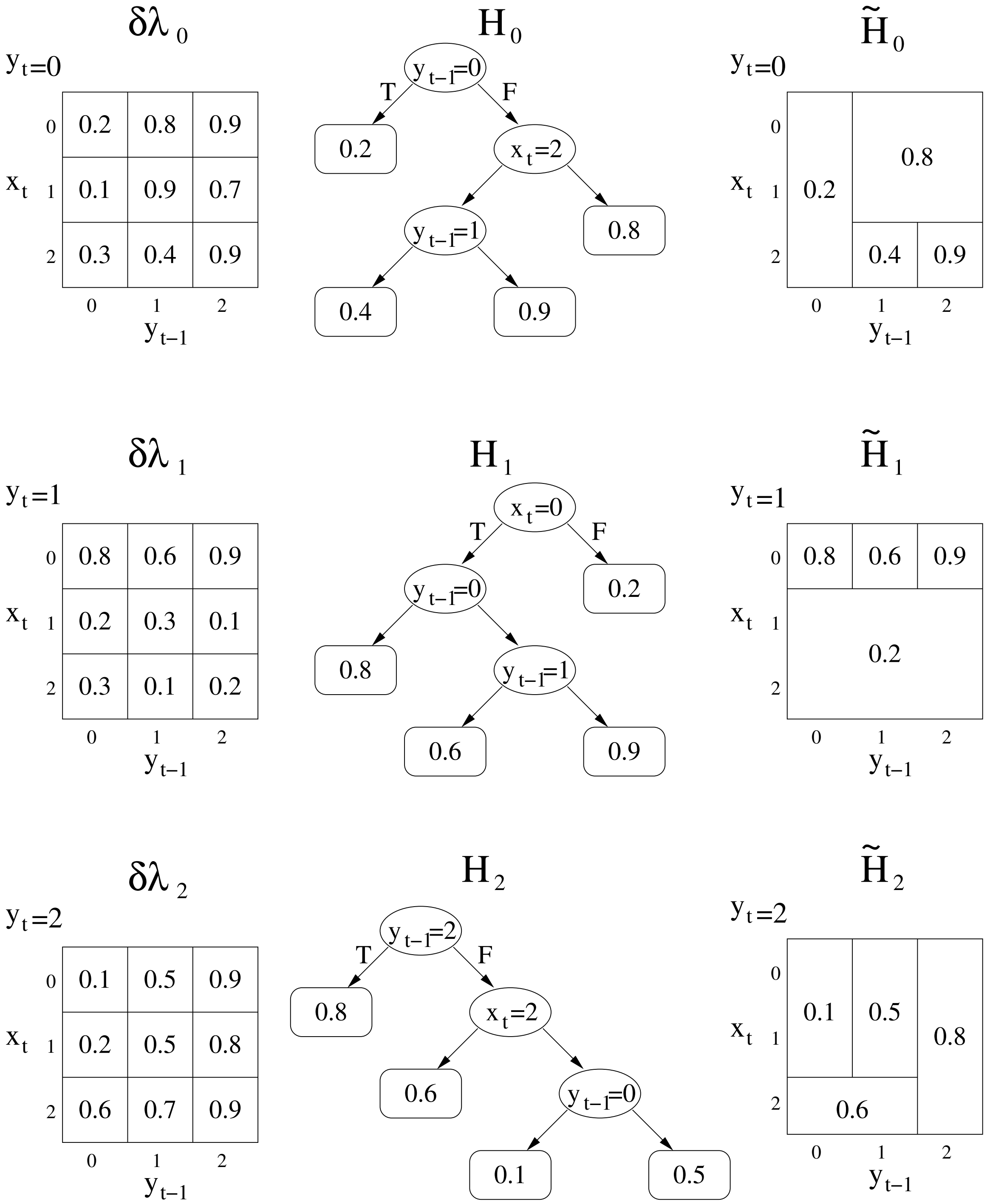

Figure 5: Table vs. Tree Representation

leaf l with the splitting feature x_i that maximizes:

$$\max_{l,x_i} \sigma_l - (\sigma_{l,x_i=true} + \sigma_{l,x_i=false}).$$

We determine the size of the regression trees by setting a limit on the number of allowed leaf nodes. There is a tradeoff in choosing tree sizes. The larger a tree, the more expressive it is. A CRF with large regression trees will learn faster, but runs the risk of overfitting if the tree is too big. Small regression trees narrow the hypothesis space for the CRF, which can help the CRF generalize better to test data.

2.3 Gradient Derivation

We have discussed growing regression trees to fit a set of target gradients, and how to update our CRF's potential functions. Before this happens, however, we must find the target gradients.

The conditional model of the boosted regression tree CRF is unchanged from equation 2. Our potential functions are represented by regression trees:

$$M_t(y_t, y_{t-1}|w_t) = \exp F_{y_t}(y_{t-1}, w_t).$$

Equation 2 can be rewritten:

$$P(Y|X) = \frac{exp\left[\sum_t F_{y_t}(y_{t-1}, w_t)\right]}{Z}$$

The goal is to train the CRF to maximize the log likelihood of all sequences in a training data set, S:

$$\sum_{i \in S} \log P(Y_i, w_i).$$

To find a target ψ, we must know the gradient of the log likelihood with respect to some $F_q(p.w)$:

$$\frac{\partial}{\partial F_q(p,w)} \sum_i \log P(Y_i, X_i)$$

Since targets are generated for each sequence independently, so we can abandon the i subscripts.

The log likelihood for a sequence is:

$$\log P(Y|X) = \sum_t F_{y_t}(y_{t-1}, w_t) - \log Z.$$

Recalling that Z is found with the forward-backward algorithm:

$$\begin{aligned} Z &= \sum_{i \in K} \alpha_t(i)\beta_t(i) \\ &= \sum_{i,j \in K} \alpha_{t-1}(i) M_t(j, i|w_t)\beta_t(j) \end{aligned}$$

Using substitution:

$$\frac{\partial}{\partial F_q(p,w)} \log P(Y|X) = \frac{\partial}{\partial F_q(p,w)} \sum_t F_{y_t}(y_{t-1}, w_t) - \log Z \tag{5}$$

2.3.1 Case 1: $F_q(p,w)$ appears in both terms

In this case, $F_q(p,w)$ appears within the term $\sum_t F_{y_t}(y_{t-1}, w_t)$ from equation 5, as well as in $\log Z$. In other words, the transition from class p to class q appears in the training sequence.

When differentiating the first term, $F_q(p,w)$ goes to 1 while all other terms in the summation act as constants and are dropped. This gives us:

$$\frac{\partial}{\partial F_q(p,w)} \log P(Y,X) = 1 - \frac{\partial}{\partial F_q(p,w)} \log Z$$

Differentiating $\log Z$ gives us:

$$\frac{\partial}{\partial F_q(p,w)} \log P(Y,X) = 1 - \frac{1}{Z} \frac{\partial}{\partial F_q(p,w)} Z. \tag{6}$$

We are now interested in the derivative of Z with respect to $F_q(p,w)$. We rewrite Z using its forward-backward equivalent,

$$\frac{\partial}{\partial F_q(p,w)} Z = \frac{\partial}{\partial F_q(p,w)} \sum_{i,j \in K} \alpha_{t-1}(j)[\exp F_i(j, w_t)]\beta_t(i).$$

$F_q(p,w)$ appears in Z, so we can seperate out the terms:

$$\begin{aligned} \frac{\partial}{\partial F_q(p,w)} Z &= \frac{\partial}{\partial F_q(p,w)} \sum_{i \neq q, j \neq p} \alpha_{t-1}(j)[\exp F_i(j, w_t)]\beta_t(i) + \\ &\qquad \alpha_{t-1}(p)[\exp F_q(p, w_t)]\beta_t(q) \end{aligned}$$

The first term consists entirely of constants, while the second term differentiates to itself:

$$\frac{\partial}{\partial F_q(p,w)} Z = \alpha_{t-1}(p)[\exp F_q(p,w_t)]\beta_t(q).$$

Substituting back into equation 6, we have

$$\frac{\partial}{\partial F_q(p,w)} \log P(Y,X) = 1 - \frac{\alpha_{t-1}(p)[\exp F_q(p,w_t)]\beta_t(q)}{Z} \tag{7}$$

2.3.2 Case 2: $F_q(p,w)$ appears only in Z

This case is identical to Case 1, except that the entire summation in first term is a constant and can be ignored. This gives us:

$$\frac{\partial}{\partial F_q(p,w)} \log P(Y,X) = -\frac{\alpha_{t-1}(p)[\exp F_q(p,w_t)]\beta_t(q)}{Z} \tag{8}$$

2.3.3 Generating example targets

Combining equations 7 and 8, we generate gradient targets:

$$\psi_{q,p,w_t} = [q = y_t, p = y_{t-1}] - \frac{\alpha_{t-1}(p)[\exp F_q(p,w_t)]\beta_t(q)}{Z}$$

$[q = y_t, p = y_{t-1}]$ is an indicator function that evaluates to 1 if q and p are in the true training sequence, 0 if otherwise. A ψ target is generated for every q and p at every time step t for every training sequence.

2.4 Approximation

Although the regression tree CRF is robust for large numbers of features, it can suffer computationally from problems with a large number of classes. The forward-backward algorithm runs in $O(L * K^2)$, where L is the length of a sequence and K is the number of classes. If S is the number of sequences, when generating gradient targets we compute $L * S * K^2$ unique ψ targets. Either one of these steps can be very costly when given a large K, such as the Nettalk problem with 127 classes.

It is also important to note that we have been treating the CRF as a first order model. We have only modeled the relationship between y_t and y_{t-1}. However, it is often desirable to use a larger number of recurrent features $(y_{t-1}, y_{t-2}, y_{t-3}, \ldots)$.

Consider a second order model, where y_t is influenced by y_{t-1} and y_{t-2}. Calculating probabilities with the forward-backward algorithm and generating gradients work in nearly the same way as the first order model. The forward pass of the forward-backward algorithm for a first order model:

$$\alpha_t(y) = \sum_{i \in K} \alpha_{t-1}(i) M_t(i, y | w_t).$$

The major difference for the second order model is that every reference to a class (such as y or i in the previous equation) symbolizes a pair of actual classes. The forward pass for a second order model:

$$\alpha_t(\langle j, k \rangle) = \sum_{i \in K} \alpha_{t-1}(\langle j, i \rangle) * M_t(\langle j, k \rangle, \langle i, j \rangle | w_t).$$

If we made a trelice for a second order model, each column would have 2^K nodes. Every node would represent a pair of classes $\langle y_{t-1}, y_t \rangle$. A sequence through the trelice might look something like this: $\langle j, i \rangle \rightarrow \langle i, l \rangle \rightarrow \langle l, k \rangle \rightarrow \langle k, j \rangle \rightarrow \langle j, l \rangle$.

An interesting side effect is that while there are 2^K nodes in each column of the trelice, a node can only have K possible predecessors. Using '$*$' as a wildcard symbol, we can express this as: $\langle *, i \rangle \rightarrow \langle i, l \rangle$.

Even with the limitation on transition edges in the trelice, the costs of the forward-backward and Viterbi algorithms are high. For a second order CRF, the time complexity is $O(L{*}K{*}2^K)$ for either algorithm. With an n-th order model, the complexity is $O(L{*}K{*}n^K)$.

Obviously, it is impractical to use the full forward-backward or Viterbi algorithms when using many recurrent features. This led us to implement approximations for both algorithms using a beam search. At every time step through the trelice, only the nodes with the b highest potential scores (or α values) are kept. All other nodes in the column are deleted. The outgoing potentials from the b best nodes are used to find the next set of b nodes, and so forth across the trelice. This can be thought of as a process for choosing a subset of the trelice. With the forward-backward algorithm, the forward pass chooses the subset of nodes and the backward pass simply calculates over this subset. This reduces the time complexity of Viterbi and forward-backward to $O(L * K * b)$, where b is size of the beam search. The complexity stays the same, regardless of the number of recurrent features. Also, the beam approximation reduces the number of target gradients generated to $L * S * b^2$.

This approximation method allows us to apply the CRF to problems with many classes, and use an arbitrary number of recurrent features. Unfortunately, the beam search has shown to have adverse effects on performance. This is discussed further, along with results of the approximation on the Nettalk problem, in section 3.1.

CHAPTER 3

TESTS AND RESULTS

In this chapter, we describe some sequential learning problems and the performance of the tree boosted conditional random field on these tasks. It should be noted that the parameters of the tree CRF, such as tree size or the size of feature windows, have been chosen empirically. To make an unbiased comparison of learning methods, these parameters must be chosen without knowledge of the effect on the test sets. This can be accomplished using hold out sets, or cross validation. The results in this chapter should be considered as the performance potential of the regression tree CRF if suitable parameters are chosen.

3.1 Nettalk

A difficult machine learning problem, the Nettalk task is to map English text into speech. Each letter in a word is labeled with a stress and a phoneme. An English word, given as a sequence of letters, must be translated into a sequence of phoneme-stress pairs. The difficulty of the problem comes from the large number of classes. There are 127 possible classifications, representing all valid pairs of phonemes and stresses.

For Nettalk, we are most interested in pronouncing entire words correctly. This means we used Viterbi classifications to maximize the probability of the sequences we predict. Recurrent features carry useful information, so we used a 15-letter feature window, along with 7 recurrent features. Approximation for the forward-backward and Viterbi algorithms was required for the large number of recurrent features. The training set and test sets each consisted of 1000 words.

Bakiri and Dietterich used recurrent decision trees with error correcting output coding to correctly label 37% of the words in a 1000 word corpus [1]. Although we test on a separate 1000 word corpus, the CRF produces competitive results.

Figure 6 shows the results of a regression tree CRF and a regression tree RSW with

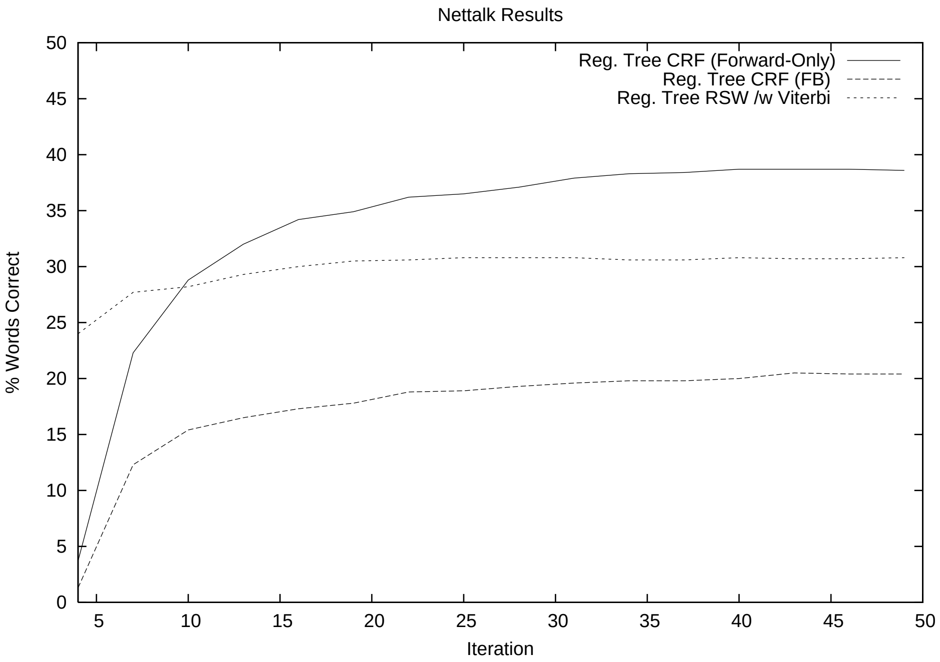

Figure 6: Nettalk Results

Viterbi classification. We limited the regression trees to 50 leaves. For the CRF, we used a small forward-backward beam search of 3. The Viterbi classification used a beam search of 10.

A surprising result from our Nettalk experiments is that calculating the training probabilities with only the forward pass of the forward-backward algorithm was much more successful than using both passes. This correlates to the plots of 'FB' and 'Forward-Only' CRFs. We believe this is a side effect of the beam search approximation on the forward-backward algorithm.

Consider the resultant sub-trelice following the forward pass of the beam search forward-backward algorithm. A situation can occur where a class node q has no children that survived within beam width at the next time step. When this happens, the β_t value for q will be zero. Assuming q is not the true state at time t,

$$\begin{aligned}\psi_{q,p,w_t} &= [q = y_t, p = y_{t-1}] - \frac{\alpha_{t-1}(p)[\exp F_q(p, w_t)]\beta_t(q)}{Z} \\ &= 0 - \frac{0}{Z} = 0\end{aligned}$$

We treat $F_q(p, w_t)$ as though it has a gradient of of 0, regardless of its value. For the training sequence, this is fine since q's children are not in the true path. However, we are not generating any training gradients for $F_q(p, w_t)$. For a test sequence, q's children may not be eliminated and a poorly trained $F_q(p, w_t)$ could lead to poor performance. The 'Forward-Only' CRF from figure 6 treats every β_t value as 1, therefore avoiding this problem. The pitfall with the 'Forward-Only CRF' is that it trains using sequential information in exclusively the forward direction.

Another drawback of our forward-backward approximation is that if the true state is dropped during the forward pass, it cannot be corrected. Regardless of the information gained in the backward pass, the states that were not contained in the forward pass beam width are lost. This biases the algorithm toward sequential information in the forward direction.

3.2 Artificial Dataset

The artificial dataset was created using a ten state hidden Markov model. The hidden states were linked in a ring pattern. Each state has a 70% chance of transitioning to the next state in the ring:

$$P(y_t = 1 | y_{t-1} = 0) = 0.7$$

$$P(y_t = 2 | y_{t-1} = 1) = 0.7$$

$$P(y_t = 3 | y_{t-1} = 2) = 0.7$$

$$\ldots$$

The other 30% transistion probability is evenly distributed over the remaining nine states.

There are ten features, one correlated to each state. A state has a 70% chance of generating its corresponding feature:

$$P(x_t = 0 | y_t = 0) = 0.7$$

$$P(x_t = 1 | y_t = 1) = 0.7$$

$$P(x_t = 2 | y_t = 2) = 0.7$$

$$\ldots$$

As before, the remaining 30% probability is evenly distributed over the other nine features. All sequences were of length ten. 500 sequences were generated for the training data, and 500 for the testing data.

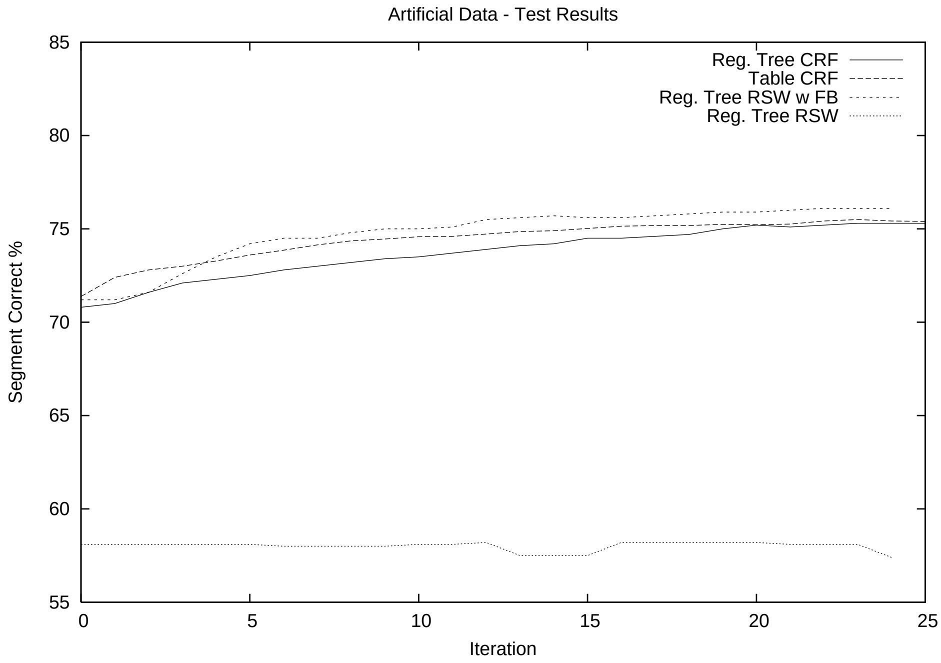

Figure 7: Artificial Dataset Results

The model was simple enough to allow us to use the full forward-backward calculations and experiment with beam width approximation. The problem was also small enough to apply the table CRF. In figure 7 the regression tree CRF and the table CRF show nearly identical performance once they are trained. This was expected, as we put no limit on the size of the regression trees. When growing the regression trees to their full size, they will represent gradients at the same detail as the table CRF. The recurrent sliding windows using the forward-backward algorithm for classification do slightly better than the CRFs. The RSW without the forward-backward algorithm performs very badly. This is likely because the RSW cannot use information from both directions over the sequence and sequential influence is stong in this problem.

Experiments with regression tree CRFs using beam width approximation show characteristics in common with the Nettalk results. In these experiments, we used Viterbi classifications to predict the most likely sequences. The percentages reported are an average over fifty training iterations. The regression tree CRF with the full forward-backward algorithm (beam width of 10) performed the best, with 11% error. Lowering the beam width to 5 severely reduces the CRF's performance to 6.5%. As in the Nettalk results, using only the forward pass of the forward-backward algorithm improves performance of the beam approximation to 8.9%.

	Full FB	FB Beam=5	F-Only Beam=5
% Seq. Correct	11.0	6.5	8.9

Table 5: Average Sequences Correct using Beam Approximation on Artificial Data

3.3 Protein Secondary Structure Prediction

Using a linear sequence of amino acids, we must predict, for every amino acid, which protein secondary structure the acid appears in [11]. There are twenty amino acids and three possible secondary structures: alpha-helix, beta-sheet, and random-coil.

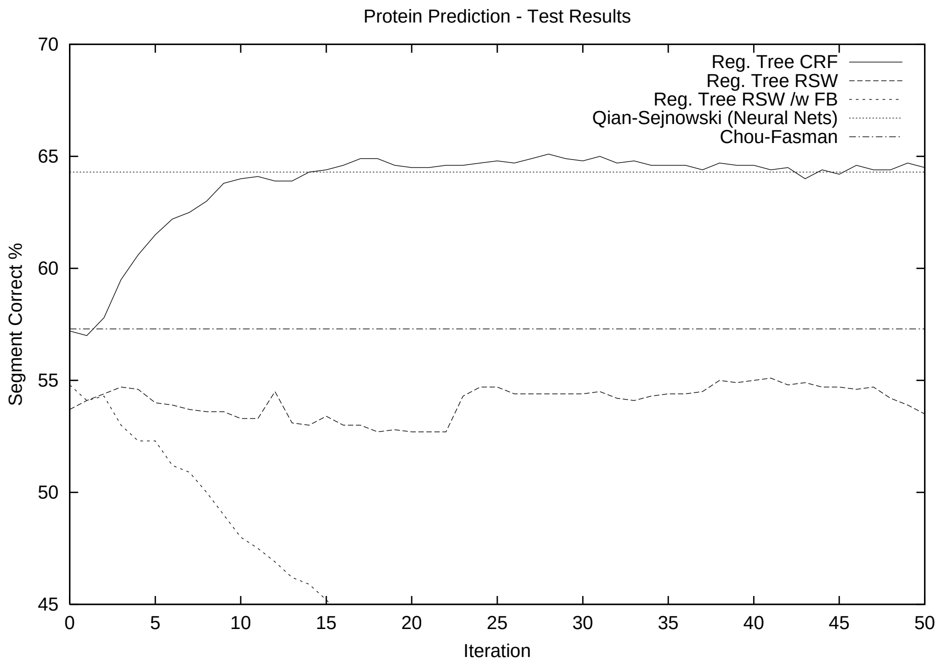

Figure 8: Protein Results

The dataset has been frequently used to compare the effectiveness of various algorithms.

One of the earliest approaches was the Chou-Fasman algorithm [3], a conglomeration of hand written rules. Qian and Sejonowski [15] designed a neural network for protein secondary structure prediction. Qian and Sejonowski achieved an average performance of 64.3% structures correctly classified. They also argue that, according to the dataset, significant improvement in performance is not possible.

As there are only three classes, we could afford to use the full forward-backward algorithm on this problem. Empirically, we found there is no benefit in using more than one recurrent feature. The tree boosted CRF and the recurrent sliding window methods used a feature window of size 11. Shown in figure 8, the tree CRF competes well against the previous best-known algorithm from Qian and Sejonowski. Both RSW methods show poor results. The RSW using forward-backward classification does especially poorly.

	Reg. Tree CRF	Table CRF
Train Set	64.8	80.4
Test Set	61.3	51.7

Table 6: Table vs. Tree CRF on Protein Prediction

The table based CRF is not practical for the 11 acid window of the previous experiment. To compare classification performance, we tested tree and table CRFs using a 3 acid window. Table 6 shows the results from both algorithms after ten iterations, when they had fit the data. The table based CRF fits the training data well, but generalizes poorly on the test set. This is not surprising, as representing so many feature combinations can lead to overfitting. The regression tree CRF, limited to trees with 25 leaves, does a much better job on the test set. Small regression trees help the tree CRF avoid overfitting, as is evident by the similar results on the training and testing sets.

3.4 Frequently-Asked-Question Parsing

The Frequently Asked Questions (FAQ) dataset is a collection of seven Usenet multipart FAQs from the internet. Each multi-part FAQ consists of four to seven separate FAQ files. A FAQ file is represented as a single sequence of labels. Every line of a FAQ is labeled as either part of the header, a question, an answer, or part of the FAQ's tail. We are interested in maximizing the number of lines we correctly label. This means classifying with the forward-backward algorithm. There are twenty boolean features, capturing information such as the current line ends with a '?'. This makes 2^{20} possible feature combinations, too many for the table-based CRF to represent.

	Reg. Tree CRF	MEMM
% Seg. Correct	91.2	86.7

Table 7: Segment Prediction on FAQ data

For our first experiment (table 7) the CRF is trained on all but one sequence from a FAQ collection, and then tested on the held out sequence. Every sequence will be held out once, and the entire process is repeated for every FAQ collection. The dataset used by McCallum, Freitag, and Pereira [13] may have some minor differences to that used by the CRF. Nonetheless, the CRF compares favorably to the results of the MEMM for overall segment prediction accuracy.

	Reg. Tree CRF	Reg. Tree RSW	Reg. Tree RSW /w FB
% Seg. Correct	96.3	54.7	91.6

Table 8: Segment Prediction on AI-Neural FAQ

In our second experiment (table 8) we focus on the AI-Neural FAQ collection. We train on three sequences, and test on the four remaining AI-Neural FAQ sequences. We compare the regression tree CRF to RSWs with and without forward-backward classification. Again, the RSW with forward-backward makes better use of sequential information and outperforms the regular RSW. The CRF achieves the best performance.

CHAPTER 4

CONCLUSIONS

The conditional random field is a powerful approach to sequential supervised learning. The CRF's main advantage is its ability to use a variety of arbitrary, non-independent features [12]. The CRF matched or outperformed other SSL classifiers in domains such as part-of-speech tagging [10], noun phrase segmentation [18], and table extraction from documents [14]. However, the CRF's ability to express any input feature, or their combination, can result in massive feature sets. Without a method to choose only informative feature combinations, the CRF is too computationally expensive for many SSL tasks.

This thesis has shown that using gradient boosted regression trees with conditional random fields provides a method for selecting feature combinations, and a process for efficient training. CRF run time results illustrate the drastic reduction in the computational burden for training CRFs. We also show that using regression trees to select feature conjunctions helps avoid overfitting and increases prediction performance.

Both the table based and tree based CRFs suffer when implemented as high order models, or when applied to tasks with many classes. The forward-backward and Viterbi algorithms are very sensitive to recurrent features and the number of classes. Often, we must use some sort of approximation for these dynamic programs. The thesis shows that beam width approximation for these algorithms decreases performance significantly. While the regression tree CRF is effective at coping with large potential feature sets, it provides no benefit for dealing with large class sets. This is a hurdle for the CRF becoming a successful general purpose SSL algorithm.

4.1 Related Work

Andrew McCallum addresses the feature selection and combination problem using feature induction [12]. Features and conjunctions are iteratively added to the CRF's feature set.

Starting with no features in the CRF model, his algorithm creates a list of candidate features and their conjunctions. The numbers of conjunctions considered are limited to those with the highest gain. The gain of a feature is a measure of the improvement in log-likelihood the feature provides the CRF. All candidate features are tested, and those with the highest gain are added to the model. This process is repeated until the CRF has the desired number of features. The effectiveness of this algorithm has not been compared to the boosted regression tree CRF, but both methods provide the same benefits for the CRF.

4.2 Future Work

The foremost research question raised by this thesis is that of how to improve the forward-backward and Viterbi approximation. The beam search appears to be inadequate for a large set of classes. With effective approximation, the CRF would be robust to both large feature and class sets. The A^* algorithm is one possible solution for better approximations. However, this leads to another question: finding a decent heuristic.

More work is also needed in reducing the number of hand-tuned parameters used by the regression tree CRF. Instead of a user-defined threshold to limit the size of our regression trees, an automatic pruning technique could be implemented. Also, the scale or step-size of each gradient update is currently a user defined constant. Finding a step-size using a line search, or another method, could eliminate this parameter.

Finally, in this thesis we consider only two types of prediction tasks. For one task, we are concerned with predicting an entire sequence correctly. In the other, we care only about maximizing the probability of correctly classifying each segment. However, problems can have more complicated objectives. Misclassification of one class may be more costly than another. Predicting short sequences correctly may be more important than correctly predicting long sequences. More research is needed to find how to train conditional random fields for arbitrary objective functions.

REFERENCES

[1] Bakiri, G. and Dietterich, T. G., "Achieving high-accuracy text-to-speech with machine learning," in Data Mining Techniques in Speech Synthesis ((Ed.), R. I. D., ed.), New York, NY: Chapman and Hall, 2002.

[2] Breiman, L., Friedman, J., Olshen, R., and Stone, C., Classification and Regression Trees. Belmont, CA: Wadsworth, 1984.

[3] Chou, P. and Fasman, G., "Prediction of the secondary structure of proteins from their amino acid sequence," Advanced Enzymology, vol. 47, pp. 45–148, 1978.

[4] Della Pietra, S., Della Pietra, V., and Lafferty, J., "Inducing features of random fields," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 4, 1997.

[5] Dietterich, T. G., "Machine learning for sequential data: A review," Lecture Notes in Computer Science, Springer-Verlag, 2002.

[6] Dietterich, T. G., Hild, H., and Bakiri, G., "A comparative study of id3 and backpropagation for english text-to-speech mapping," in Proceedings of Machine Learning Conference, (Austin, TX.), pp. 24–31, 1990.

[7] Friedman, J., "Stochastic gradient boosting," tech. rep., Dept. of Statistics, Stanford University, 1999.

[8] Friedman, J., "Greedy function approximation: a gradient boosting machine," The Annals of Statistics, vol. 29, no. 5, pp. 1189–1232, 2001.

[9] Friedman, J., Hastie, T., and Tibshirani, R., "Additive logistic regression: A statistical view of boosting," The Annals of Statistics, vol. 28, no. 2, pp. 337–374, 2000.

[10] Lafferty, J., McCallum, A., and Pereira, F., "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in Proceedings of Int. Conf. on Machine Learning, Morgan Kaufmann, 2001.

[11] Maclin, R. and Shavlik, J., "Refining algorithms with knowledge-based neural networks: Improving the chou-fasman algorithm for protein folding," Computational Learning Theory and Natural Learning Systems, pp. 249–286, 1994.

[12] McCallum, A., "Efficently inducing features of conditional random fields.," in 19th Conference on Uncertainty in Artificial Intelligence, 2003.

[13] McCallum, A., Freitag, D., and Pereira, F., "Maximum entropy markov models for information extraction and segmentation," in Proceedings of Proceedings of Int. Conf. on Machine Learning, Morgan Kaufmann, 2000.

[14] Pinto, D., McCallum, A., Lee, X., and Croft, W. B., "Combining classifiers in text categorization," in Submitted to SIGIR '03: Proceedings of Twenty-sixth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2003.

[15] Quan, N. and Sejnowski, T., "Predicting the secondary structure of globular proteins using neural network models," Journal of Molecular Biology, vol. 202, pp. 865–884, 1988.

[16] Quinlan, J. R., "Bagging, boosting, and c4.5," in Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-96), (Portland, OR), AAAI Press, 1996.

[17] Rabiner, L., "A tutorial on hidden markov models and selected applicationsin speech recognition," Proceedings of the IEEE, vol. 77, no. 2, pp. 257–286, 1989.

[18] Sha, F. and Pereira, F., "Shallow parsing with conditional random fields," in Proceedings of Human Language Technology, NAACL, 2003.