University of Pennsylvania
ScholarlyCommons

Technical Reports (CIS) | Department of Computer & Information Science

January 2007

Graphical Models for Primarily Unsupervised Sequence Labeling

Neal Parikh
University of Pennsylvania, neal@nparikh.org

Mark Dredze
University of Pennsylvania, mdredze@seas.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/cis_reports

Recommended Citation
Parikh, Neal and Dredze, Mark, "Graphical Models for Primarily Unsupervised Sequence Labeling" (2007). *Technical Reports (CIS)*. Paper 638.
http://repository.upenn.edu/cis_reports/638

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-07-18.

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_reports/638
For more information, please contact repository@pobox.upenn.edu.

Graphical Models for Primarily Unsupervised Sequence Labeling

Abstract
Most models used in natural language processing must be trained on large corpora of labeled text. This tutorial explores a "primarily unsupervised" approach (based on graphical models) that augments a corpus of unlabeled text with some form of prior domain knowledge, but does not require any fully labeled examples. We survey probabilistic graphical models for (supervised) classification and sequence labeling and then present the prototype-driven approach of Haghighi and Klein (2006) to sequence labeling in detail, including a discussion of the theory and implementation of both conditional random fields and prototype learning. We show experimental results for English part of speech tagging.

Comments
University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-07-18.

This technical report is available at ScholarlyCommons: http://repository.upenn.edu/cis_reports/638

Graphical Models for Primarily Unsupervised Sequence Labeling*

Neal Parikh NEAL@NPARIKH.ORG
Mark Dredze MDREDZE@SEAS.UPENN.EDU
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104

Abstract

Most models used in natural language processing must be trained on large corpora of labeled text. This tutorial explores a "primarily unsupervised" approach (based on graphical models) that augments a corpus of unlabeled text with some form of prior domain knowledge, but does not require any fully labeled examples. We survey probabilistic graphical models for (supervised) classification and sequence labeling and then present the prototype-driven approach of Haghighi and Klein (2006) to sequence labeling in detail, including a discussion of the theory and implementation of both conditional random fields and prototype learning. We show experimental results for English part of speech tagging.

1. Introduction

A primary goal of the field of natural language processing is to discover linguistic structure in unstructured text. This includes a wide range of applications, such as syntactic parsing, machine translation, and word sense disambiguation. Many of these problems are inherently sequential in that each datum is a sequence of words (such as a sentence) rather than a single word, so algorithms can assign structure to sentences by assigning a label to each word in the sentence. Such tasks are referred to as *sequence labeling*.

In *NP chunking*, for example, noun phrases such as "the tall mountain" are identified (Tjong et al., 2000). The field of information extraction contains many labeling tasks, such as finding named entities (called *named entity recognition*; see Figure 1 for an example[1]) or finding contact information in emails (McCallum, November 2005; Minkov et al., 2005). Labeling tasks also often appear in the field of computational biology, though gene sequences rather than sentences are labeled (Culotta et al., April 2005; Vinson et al., 2007).

This tutorial focuses on the part of speech (POS) tagging problem, where words are labeled with their parts of speech, such as NOUN, VERB, or ADJECTIVE (Charniak, 1997; Ratnaparkhi, 1996). Part of speech tagging is a useful preprocessing step for numerous applications, such as parsing or document classification. See Figure 1 for an example of tagged text.

The standard approach to these tasks is to use statistical techniques from the field of machine learning (Manning and Schutze, 1999; Jurafsky and Martin, 2006), which can be divided into two broad categories based on the nature of the available training data:

1. **Supervised learning** relies on a corpus of *labeled* text containing training examples like the one in Figure 1. The model learns to label new sentences by finding parameters that assign high probability to the labelings found in the training data. For this reason, model behavior is greatly influenced by

*. University of Pennsylvania Technical Report MS-CIS-07-18.
1. B represents the beginning of a named entity, I represents being inside, and O represents being outside.

NER	B	I	O	O	O	O	O	O	O	O	O	O
POS	NNP	NNP	,	CD	NNS	JJ	,	MD	VB	DT	NN	.
	Pierre	Vinken	,	61	years	old	,	will	join	the	board	.

Figure 1: A sentence labeled with parts of speech (POS) and named entity labels (NER).

the size and quality of the training corpus. While supervised approaches can perform exceedingly well when good data is available, designing a corpus requires substantial expertise and resources.

2. **Unsupervised learning** uses only *unlabeled* text and groups text based on perceived patterns in the data. The clear advantage of this approach is that large amounts of unlabeled text are often freely available, and these methods can be used when labeled data is unavailable. However, since no supervision is provided, the patterns found by the model may not match the user's intentions. For example, the model could arbitrarily decide to group words by first letter. To avoid such problems, specifying model structure appropriate to the task is crucial. Performance is primarily determined by model structure, and designing an appropriate model is a difficult task requiring expertise and effort.

Given this dichotomy, it is natural to ask whether a middle ground exists that can profitably use both labeled and unlabeled data. Such approaches are referred to as *semi-supervised*, and they hope to achieve the performance of supervised algorithms without substantial data requirements by augmenting a small amount of labeled data with a large amount of unlabeled data.

A crucial step in semi-supervised learning is to define a model that describes the unlabeled data well. Despite significant research into such models, it has been difficult to find a method that works well across multiple tasks without significant engineering for each domain (Mann and McCallum, 2007; Ando and Zhang, 2005; Chapelle et al., 2006; Sindhwani and Keerthi, 2006; Joachims, 1999; Jiao et al., 2006; Grandvalet and Bengio, 2004; Nigam et al., 2000). See Zhu (2006) for a survey of semi-supervised learning.

Explicitly, in supervised learning, the training data takes the form $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$ (text with labels), while in unsupervised learning, $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ (text without labels). There are N training instances where $\mathbf{x}^{(i)}$ is the ith sentence (a sequence of words) and $\mathbf{y}^{(i)}$ is the correct labeling for that sentence, such as in Figure 1. In semi-supervised learning, $\mathcal{D} = \{\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N},\ \{\mathbf{x}^{(j)}\}_{j=1}^{M}\}$, so there are N labeled instances and M unlabeled instances. In practice, more unlabeled data is provided because it is usually easy to obtain, so $N \ll M$. In fact, one could give just a single example for each label, in which case N is as small as the number of labels (Mann and McCallum, 2007).

Traditional semi-supervised learning requires some labeled data, but a new set of techniques has recently emerged that use some prior knowledge of the domain to push the model in the right direction but that do not require any labeled examples. This paper discusses one such technique, *prototype learning* (Haghighi and Klein, 2006), in which prior knowledge is encoded by providing several examples, or prototypes, of each label. For example, one could give two or three words for each part of speech but not provide any complete labeled sentences. These methods go by many names that set them apart from the standard semi-supervised setting, including "primarily unsupervised" (our preference), "unsupervised with prior information," "weakly supervised," and "minimally supervised." In this setting, $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ as in the unsupervised case.

The purpose of this tutorial is to explain the structure and implementation of graphical models for sequence labeling, and to clarify dense material on prototype learning where details are scarce. Section 2 surveys graphical models with a focus on conditional random fields. Prototype learning uses a kind of graphical model rarely used in natural language processing, and understanding the different classes of graphical models helps in appreciating this choice. Section 3 describes prototype-driven learning, with an emphasis on algorithms needed for unsupervised inference and parameter estimation.

2. Graphical Models

A *statistical model* $\mathcal{P}$ is a family of probability distributions

$$\mathcal{P} = \{p_\theta \mid \theta \in \Theta\}, \tag{1}$$

where θ is a *parameter vector*, the set Θ is a *parameter space*, and p_θ is a probability distribution function on a probability space. The parameter space Θ is usually considered to be a subset of $\mathbb{R}^n$, so the parameters of the model will be real numbers. In natural language processing, it is assumed that observed (labeled or unlabeled) text is generated from a probability distribution, and training a model involves finding which of the p_θ most likely produced a given set of data. Model training is often called *parameter estimation* because it requires finding the best estimates of the parameters θ.

In a modeling task, each entity (such as a word or label) is assigned a random variable, and the p_θ then represent joint probability distributions over all the random variables. Joint probability distributions require parameters exponential in the number of random variables in the model, so it is necessary to make assumptions of independence. For example, we could assume the Markov property, which states that a future state is independent of past states given its immediate predecessors. In part of speech tagging, this says that the label of the last word in the sentence depends only on the penultimate label. In general, assumptions that random variables only depend on "nearby" variables are referred to as *Markov assumptions*.

Graphical models provide a convenient and powerful way of representing and exploiting such independence assumptions. A graphical model is a statistical model where the joint distributions p_θ factorize according to an underlying graph, such that nodes in the graph are random variables. The idea is to represent a complex distribution over a large number of random variables as a product of local functions that each depend on only a small number of related variables.[2] This framework subsumes a very wide variety of models from many different fields. For introductions to graphical models, see Jordan (1999, 2003, 2004); Murphy (2001); Heckerman (1995); Bishop (2006).

We will follow Sutton and McCallum (2006) and emphasize the following distinctions: directed vs. undirected, generative vs. discriminative, and classification vs. sequence models.

2.1 Bayesian networks and Markov random fields

Graphical models are called directed or undirected depending on whether or not the underlying graph is directed. When graphical models are taken to represent a family of joint (rather than conditional) distributions, directed models are also called *Bayesian networks* and undirected models are called *Markov random fields*. Directed and undirected models have different factorization properties with different advantages.

Consider probability distributions over sets of random variables $\{X, Y\}$, where X is a set of input variables observed from data, and Y is a set of output variables that the model must predict. The graphical model posits some relationship between members of X and members of Y according to a graph G. The model describes some distribution $p(\mathbf{y}, \mathbf{x})$ for some assignments $\mathbf{x}, \mathbf{y}$ of the random variables X and Y. For sequence problems, $\mathbf{x}$ is the observed sequence of words and $\mathbf{y}$ is the label sequence that must be predicted. In this tutorial, all variables are assumed to be discrete, which is appropriate for labeling problems.

Definition 1 (Bayesian network)**.** Let $G = (V, E)$ be a directed graph. Then a Bayesian network is a family of distributions that factorize as

$$p(\mathbf{y}, \mathbf{x}) = \prod_{v \in V} p(v|\pi(v)), \tag{2}$$

2. The precise nature of the relationship between independence assumptions and factorization is outside the scope of this tutorial; see Jordan (2003) for a discussion.

where $\pi(v)$ denotes the parents of the node v in G. Note that the local functions $p(v|\pi(v))$ are themselves conditional probability distributions. This factorization is equivalent to the assumption that nodes are conditionally independent of nondescendants given their parents.

Definition 2 (Markov random field)**.** Let H be an undirected graph. Given a set of maximal cliques $\mathcal{A} = \{A_i\}$ in H, a Markov random field (MRF) is a family of distributions that factor as

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_{A \in \mathcal{A}} \psi_A(\mathbf{y}_A, \mathbf{x}_A), \tag{3}$$

where

$$Z = \sum_{\mathbf{x}} \sum_{\mathbf{y}} \prod_{A \in \mathcal{A}} \psi_A(\mathbf{y}_A, \mathbf{x}_A) \tag{4}$$

is called the *partition function*, a global normalization constant that ensures the above distribution sums to 1. Here the *clique potentials* ψ_A are arbitrary functions $V^n \to \mathbb{R}^+$, from sets of nodes to nonnegative reals. This factorization is equivalent to the assumption that nodes are conditionally independent of all other nodes given their neighbors.

Note that the partition function Z sums over all possible input sequences and labelings, which in the case of part of speech tagging corresponds to all possible sequences of words (not only grammatical sentences) and all possible sequences of part of speech tags.

Directed graphical models are appealing when a natural conditional relationship exists between entities, such as symptoms depending on diseases. Undirected models are appropriate when no such directionality can be assigned, but are also useful because they allow for more flexible local structure with clique potentials that are not probability distributions. The cost of this added representational power is that the normalizer Z must be computed to ensure a coherent global probability distribution; in directed models, $Z = 1$ because it sums over a probability distribution. Computing Z is often the key difficulty in using undirected models, and approximations are needed when computing the true value is intractable.

2.1.1 Feature Functions and Exponential Representations

In the definition of undirected models, clique potentials on maximal cliques were allowed to take an arbitrary form. By ranging over all possible potential functions on the maximal cliques, we obtain all the probability distributions that respect the Markov random field structure on the graph. In practice, one does not want to work with arbitrary, fully-parametrized cliques because inference is exponential in the size of the cliques and because estimating a huge number of parameters requires large amounts of data. For this reason, reduced parametrizations of clique potentials are used. Our presentation follows Jordan (2003).

Consider the problem of building a model that assigns high probability to strings that respect the orthographic rules of English and low probability to strings that do not. Let each position in the string be represented by a multinomial random variable taking on one of 26 values, so for strings of length five there will be five nodes and a sample space of order $26^5 \approx 12$ million. Suppose we want to assign high probability to strings ending in *-ing*. This requires having a clique of size three in the model (and a potential function on three nodes), but having a fully parametrized potential on three nodes would require $26^3 \approx 18$K parameters, an undesirably large number. It is inefficient to assign values to all these 26^3 possibilities just to be able to assign a high value to the *-ing* configuration. The natural solution is to use a reduced parametrization where there is some elementary parameter called a *feature* that turns on or off depending on whether the configuration *-ing* appears or not. Thus there will be a relatively small set of elementary features available, and each clique potential will use the same set of parameters as building blocks.

In the *-ing* case, create a binary feature $f_{\text{ing}}(\mathbf{x})$ that is 1 if $\mathbf{x}$ ends in *-ing* and is 0 otherwise. Associate a parameter θ_{ing} that the model can use to vary the numerical weight that should be accorded this feature. In

other words, θ_{ing} specifies how important the model thinks it is for a word to end in *-ing*. It is also possible and useful to make a feature for every word type observed in the training data, so known words get more weight in the model. Since the features are a function of the configuration of the relevant clique, they can be called *feature functions*.

The remaining question is what form the parametrized clique potentials will take. There are many possible choices, but the most common is to use the exponential or log-linear representation

$$\psi_A(\mathbf{y}_A, \mathbf{x}_A) = \exp\left\{\sum_k \theta_{Ak} f_{Ak}(\mathbf{y_A}, \mathbf{x_A})\right\} \tag{5}$$

for parameters θ and feature functions $\{f_{Ak}\}$. Given this, (3) can be rewritten

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \exp\left\{\sum_A \sum_k \theta_{Ak} f_{Ak}(\mathbf{y_A}, \mathbf{x_A})\right\} \tag{6}$$

since $\prod \exp = \exp \sum$.

This form is used because it ensures that the probability distributions over the graph will be an *exponential family*, which is any class of distributions that can be written

$$p(x|\eta) = h(x) \exp\left\{\eta^T T(x) - A(\eta)\right\} \tag{7}$$

for a parameter vector η. Most commonly used classes of distributions form an exponential family, such as the Gaussian, Dirichlet, multinomial, Bernoulli, and Poisson families. See Jordan (2003) for a discussion of (7), including its origin and what the variables and functions in the definition represent. For our purposes, the important point is that this form has many convenient algebraic and statistical properties, and that by ensuring our distributions form an exponential family, we can make use of powerful general results that will ease inference and estimation.

See Jordan (2004) for an introduction to directed and undirected models, and see Pearl (1988) and Jordan (2003) for in-depth discussions, including full explanations of the relationship between factorization and independence assumptions. See Jordan (2003) for a detailed treatment of exponential families and features.

2.2 Classification Models

Classification models are one of the simplest classes of graphical models, so they are a natural first example. A classification problem is one where we wish to predict a single variable y, which takes values in a finite, unordered set, given some input data $\mathbf{x} = (x_1, \ldots, x_K)$. A simple example is spam classification: $y \in \{\text{spam, not spam}\}$, and $\mathbf{x}$ is a feature vector generated from an email (Sahami et al., 1998). See (Sebastiani, 1999, 2002; Langford, 2005) for other examples of document classification.

A standard model used in computational linguistics is the *naïve Bayes classifier*, which makes the strong simplifying assumption that all the features x_i are conditionally independent given the class label y. It models the joint probability distribution $p(y, \mathbf{x})$ in the following manner:

$$p(y, \mathbf{x}) = p(y) \cdot p(\mathbf{x}|y) = p(y) \prod_{k=1}^{K} p(x_k|y), \tag{8}$$

where the second equality follows from the conditional independence assumption. To classify some unknown $\mathbf{x}$ as one of some set $C = \{c_i\}$, one checks to see which $p(y = c_i|\mathbf{x})$ is largest. The naive Bayes model is a simple example of a Bayesian network, as shown in Figure 2(a).

(a) Naive Bayes (b) Maximum Entropy

Figure 2: Simple classification models.

Another standard classifier is the *maximum entropy classifier* (known as *logistic regression* in the statistics community) (Nigam et al., 1999). This model assumes that the conditional probability of the class label $p(y|\mathbf{x})$ is a log-linear function of the features. This leads to the model form

$$p(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{\theta_y + \sum_{i=1}^{N} \theta_{y,i} x_i\right\}, \tag{9}$$

where $Z(\mathbf{x})$ is an *instance-specific* normalization constant (due to conditioning on $\mathbf{x}$). Said differently, it assumes that the log probability $\log p(y|\mathbf{x})$ is a linear function of the features. The term θ_y is a *bias weight* that behaves like the prior distribution $p(y)$ in the naive Bayes model: it pushes the model closer to some labels before even looking at the specific $\mathbf{x}$, which is appropriate when all labels do not occur equally often.

Figure 2 shows graphical representations of naïve Bayes and maximum entropy. Note that naïve Bayes is directed and maximum entropy is undirected because the local functions in naïve Bayes are conditional probability distributions, while in maximum entropy they are log-linear and must be explicitly normalized. Though maximum entropy is undirected, it is a conditional random field rather than a Markov random field because it models a conditional distribution.

The large grey rectangle here indicates that maximum entropy globally conditions on the entire observation, rather than breaking each feature into a separate clique as in naive Bayes; it does not make any independence assumptions about the data. The black square nodes are *factors*; in general, instead of having cliques in a graph and defining potentials on cliques, we can add factor nodes, connect clique nodes to factors, and then define the potentials over factors. This is equivalent but easier to visualize. In the case of the maximum entropy model above, the factor node is entirely unnecessary, but is included because it mirrors how conditional random fields are drawn.

Classification models are not the primary focus of this tutorial, but they are useful to keep in mind due to their simplicity. The sequence models presented below build on this foundation: classifiers are sequence models for sequences of length 1. Note the similarities, for example, between (9) and (5), or (8) and (10). It is not a coincidence that the clique potentials in undirected models are chosen to look like an unnormalized maximum entropy model, and in fact, clique potentials are often assumed to include a bias weight θ_A as in (9), but this was omitted for readability. In some sense, hidden Markov models are a generalization of naïve Bayes to sequences, and conditional random fields are a generalization of maximum entropy to sequences. See Mitchell (2006) for a thorough introduction and discussion of naïve Bayes and maximum entropy classifiers.

2.3 Generative and Discriminative Models

Probabilistic models can be divided into two groups: *generative* and *discriminative* models. Roughly speaking, generative models model the joint distribution $p(\mathbf{y}, \mathbf{x})$, while discriminative models model the conditional distribution $p(\mathbf{y}|\mathbf{x})$. Intuitively, generative models fully describe the data, while discriminative models describe the differences between classes without saying anything about the classes themselves.

Generative models are called generative because they specify how to generate new data: if $p(\mathbf{y}, \mathbf{x})$ were modeled directly, then sample from it; if it were modeled in two pieces as $p(\mathbf{x}|\mathbf{y})$ and $p(\mathbf{y})$, first sample a class from $p(\mathbf{y})$ and then generate data based on this class using $p(\mathbf{x}|\mathbf{y})$. In the case of spam classification with naïve Bayes, this would involve deciding whether a given message is spam or not, then sampling words from the relevant distribution. Clearly, one can obtain $p(\mathbf{y}|\mathbf{x})$ from $p(\mathbf{y}, \mathbf{x})$ or from $p(\mathbf{x}|\mathbf{y})$ and $p(\mathbf{y})$, but this is circuitous since the goal is only to discriminate between choices of $\mathbf{y}$ to assign to a given $\mathbf{x}$, which discriminative methods do directly.

Modeling the full joint distribution can be difficult when the data is highly structured, as it involves modeling the distribution of the data and accounting for correlated features of the input. On the other hand, directly modeling the conditional distribution is sufficient for classification tasks, and because the model no longer needs to account for complex dependencies among input variables, richer features of the input can be used to aid classification performance.

Complicated dependencies between input features can be handled in two primary ways in generative models: either one must undertake a difficult modeling task (what is the relationship between different features?) or make powerful simplifying independence assumptions as naïve Bayes does (features are conditionally independent given the class label). The former can lead to intractable models, but the latter generally hurts performance. Discriminative models are better equipped to handle such situations: they make no claims about the form of $p(\mathbf{x})$, and can focus all their modeling power on capturing $p(\mathbf{y}|\mathbf{x})$. In settings other than classification and labeling, such as *topic modeling* (Blei et al., 2003; Steyvers and Griffiths, 2006), generative models can be a natural choice.

A major reason for introducing naïve Bayes and maximum entropy classifiers is that they are the canonical examples of generative and discriminative models respectively, as one models a joint distribution and the other models a conditional distribution directly. Naïve Bayes learns $p(\mathbf{x}|y)$ and $p(y)$, from which it can get $p(y|\mathbf{x})$; maximum entropy, on the other hand, models the distribution of interest directly and does not need to make independence assumptions about $p(\mathbf{x}|y)$.[3] Because it can cope with complex interactions between features, maximum entropy tends to outperform naïve Bayes in practice (Genkin et al., 2004, 2006).

See Mitchell (2006) for an excellent case study in generative and discriminative models based on naïve Bayes and maximum entropy classifiers. For a broader discussion, see Jordan (2003).

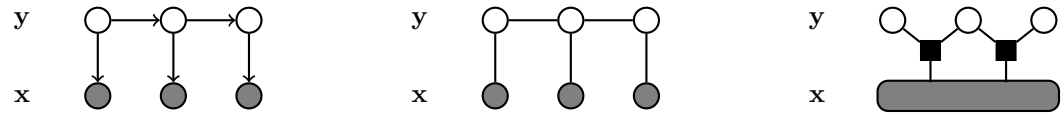
2.4 Sequence Models

Sequence models extend classifiers to the setting where the input $\mathbf{x}$ is a sequence of words and the output $\mathbf{y}$ is an entire sequence of labels. The goal of the model is to label an entire sentence at a time, which includes tasks like part of speech tagging that are our primary concern. We discuss both generative and discriminative sequence models.

A *hidden Markov model* (HMM) is a type of Bayesian network that models a sequence of observations $\mathbf{x} = (x_t)$ by assuming the existence of an underlying sequence of states (labels) $\mathbf{y} = (y_t)$ that have the structure of a Markov chain (Rabiner, 1989). For language tasks, each x_t is the (feature vector of the) word at position t in a given sentence, and each y_t is the corresponding label (such as a part of speech tag).[4] An HMM assumes that each state depends only on the previous state (the Markov property) and that each observation depends only on the current state:

$$p(\mathbf{y}, \mathbf{x}) = p(y_0) \prod_{t=1}^{T} p(y_t|y_{t-1}) p(x_t|y_t), \tag{10}$$

3. Though we will not explore this aspect here, this is in fact the *only* difference between naive Bayes and maximum entropy; they form a *generative-discriminative pair*, in the sense of Ng and Jordan (2002).
4. In a mild abuse of notation, feature vectors of words are not written in bold here as they were for classifiers to avoid introducing unnecessary new notation. In sequence models, each x_t is a feature vector but y_t is a single label.



(a) Hidden Markov Model (b) Linear-chain MRF (c) Linear-chain CRF

Figure 3: Sequence models.

where $p(y_0)$ is the prior distribution over labels, $p(y_t|y_{t-1})$ is the *transition probability* of moving from one state to another, and $p(x_t|y_t)$ is the *emission probability* of seeing an observation x_t while in state y_t. Note that this factorization follows from (2). In part of speech tagging, the emission probability of *computer* in state NOUN should be high because *computer* is a noun, but the transition probability of DETERMINER to . should be low because sentences rarely end with determiners. See Figure 3(a) for a graphical representation.

HMMs are generative models whose discriminative counterpart are conditional random fields. As in the case of naïve Bayes and maximum entropy, conditional random fields make fewer assumptions about the input and usually have superior performance. Despite this, prototype-driven learning uses a generative model for reasons discussed later. Conditional random fields are undirected models and can be graphically represented as in Figure 3(c).

It is worth pointing out that each feature function in the CRF can depend on observations from *any* time step, which is why the argument $\mathbf{x_t}$ in the definition is written in bold. The vector $\mathbf{x_t}$ represents "all the input features that are relevant at time t," not simply the features at position t of the input sequence; each $\mathbf{x_t}$ could be the entire input $\mathbf{x}$ or a local set of inputs like $\{x_{t-1}, x_t, x_{t+1}\}$. To indicate this, Figure 3(c) shows three connected nodes at each time step, one of which is the entire observation sequence (shaded); in the other two graphs, edges between labels are not connected to observations.

Definition 3 (linear-chain conditional random field)**.** Let Y, X be random vectors, let $\Theta = \{\theta_k\} \in \mathbb{R}^K$ be a parameter vector, and let $\{f_k(y, y', \mathbf{x_t})\}_{k=1}^K$ be a set of real-valued feature functions. Suppose the potentials are as in (5). A linear-chain conditional random field (CRF) is a distribution $p(\mathbf{y}|\mathbf{x})$ of the form

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \psi(y_t, y_{t-1}, \mathbf{x_t}) = \frac{1}{Z(\mathbf{x})} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y_{t-1}, y_t, \mathbf{x_t})\right\}, \tag{11}$$

where $Z(\mathbf{x})$ is an instance-specific normalization function

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y_{t-1}, y_t, \mathbf{x_t})\right\}. \tag{12}$$

We present only a few remarks about the definition here; for a thorough discussion, see Sutton and McCallum (2006) and Wallach (2004). CRFs were first introduced by Lafferty et al. (2001).

1. The normalization constant $Z(\mathbf{x})$ is instance-specific as in maximum entropy, which is expected since the maximum entropy classifier is a conditional random field for sequences of length 1. This normalizer contains a sum over all possible label sequences, which is exponential, but a dynamic programming algorithm called *forward-backward* computes this efficiently. The quantity $Z(\mathbf{x})$ is not as difficult to compute as the global normalizer Z in Markov random fields, which is *not* instance-specific and contains a sum over all possible $\mathbf{x}$ as well as $\mathbf{y}$. That said, in CRFs a separate $Z(\mathbf{x})$ must be computed for each $\mathbf{x}$, while in MRFs it can be computed once and used globally.

2. If a joint distribution $p(\mathbf{y}, \mathbf{x})$ factors like a hidden Markov model, then the associated conditional distribution $p(\mathbf{y}|\mathbf{x})$ is a linear-chain conditional random field with degenerate choices of feature functions. The general definition simply allows for arbitrary feature functions which will not correspond to local functions that are themselves probability distributions, as they must be in (2), and is the source of the added representational power. An example of more complex feature functions are input-dependent transition scores, which are commonly used in text applications. HMMs are time homogeneous, so representing this would be impossible.

Other than the fact that all CRF features can depend on the input data, the only practical difference between training and using conditional random fields (discriminative) and Markov random fields (generative) is that the normalizer is not observation dependent in the latter case, so life is harder.

Figure 3 shows graphical representations of the three main sequence models. The goal is to infer the white hidden nodes (labels) given the shaded observed nodes (input tokens). The box surrounding all the observations in the CRF indicates that we *globally* condition on as many observations $\mathbf{x_t}$ as we like. The black square nodes (*factors*) in Figure 3(c) indicate that the CRF feature functions are in the form $f(y_{t+1}, y_t, \mathbf{x_t})$ (*i.e.*, a function of three items rather than two, one of which is $\mathbf{x_t}$).

Hereafter, *conditional random field* and *Markov random field* will refer to the linear case. The rest of this section describes how to train and use CRFs with a labeled corpus.

2.5 Parameter estimation in conditional random fields

Parameter estimation for conditional random fields is closely related to estimation in Markov random fields, and the discussion below will serve as the foundation for Section 3's presentation of training the MRF used in prototype learning.

Let $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^N$ be independent and identically distributed training data, where each instance $\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_T^{(i)})$ is a sequence of inputs and each $\mathbf{y}^{(i)} = (y_1^{(i)}, \ldots, y_T^{(i)})$ is a sequence of labels. Explicitly, $\mathbf{x}^{(i)}$ is the ith sentence in the training data, and $x_j^{(i)}$ is the *feature vector* of the jth word in that sentence. Recall that each position in the sequence, each word, will be represented by an entire vector of features that capture properties of the word, such as whether or not it is capitalized or if it is a number.

Estimating the model parameters $\{\theta_k\}_{k=1}^K$ is generally done by penalized maximum likelihood estimation. Since the model represents a conditional distribution, the *conditional log likelihood* is used, just as in training maximum entropy classifiers:

$$\ell(\theta) = \sum_{i=1}^{N} \log p(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}), \tag{13}$$

which becomes

$$\ell(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} \theta_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x_t}^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)}) \tag{14}$$

when we substitute in the CRF definition from (11). As *regularization* helps avoid overfitting (Hastie et al., 2001), we place a diagonal Gaussian prior $N(0, \sigma^2 I)$ on the parameters θ_k. This gives the following regularized log likelihood as an objective function to maximize:

$$\ell(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} \theta_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x_t}^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)}) - \sum_{k=1}^{K} \frac{\theta_k^2}{2\sigma^2}. \tag{15}$$

The regularization parameter σ^2 is often taken to be 1, and the accuracy of the final model is usually not sensitive to changes in σ^2.

This objective function cannot be maximized in closed form, so numerical optimization algorithms are used. A popular choice is limited-memory BFGS (L-BFGS) (Liu and Nocedal, 1989), a drop-in replacement for standard conjugate gradient that converges more quickly. Gradient methods require the ability to evaluate the gradient of the objective function. The partial derivatives of the conditional log likelihood are

$$\frac{\partial \ell}{\partial \theta_k} = \mathbb{E}_{\tilde{\mathbf{x}},\tilde{\mathbf{y}}} f_k - \mathbb{E}_{\mathbf{y},\tilde{\mathbf{x}}} f_k - \sum_{k=1}^{K} \frac{\theta_k}{\sigma^2} \tag{16}$$

$$= \sum_{i=1}^{N} \sum_{t=1}^{T} f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x_t}^{(i)}) - \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{y,y'} f_k(y, y', \mathbf{x_t}^{(i)}) \cdot p(y, y'|\mathbf{x}^{(i)}) - \sum_{k=1}^{K} \frac{\theta_k}{\sigma^2}, \tag{17}$$

where $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ are empirical values from the training data. Explicitly, the first term $\mathbb{E}_{\tilde{\mathbf{x}},\tilde{\mathbf{y}}} f_k$ is the expected value of feature f_k under the empirical distribution

$$\tilde{p}(\mathbf{y}, \mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\{\mathbf{y}=\mathbf{y}^{(i)}\}} \mathbf{1}_{\{\mathbf{x}=\mathbf{x}^{(i)}\}}, \tag{18}$$

where $\mathbf{1}_P$ is an indicator function that is 1 when P is true and 0 otherwise. Since all training instances are equally likely in $\tilde{p}$, no probability weights are needed in this expectation, and it can be computed by counting the number of times feature f_k appears across all instances. This justifies the substitution

$$\mathbb{E}_{\tilde{\mathbf{x}},\tilde{\mathbf{y}}} f_k = \sum_{i=1}^{N} \sum_{t=1}^{T} f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x_t}^{(i)}). \tag{19}$$

The second expectation arises from the derivative of the $\log Z(\mathbf{x})$ term in (14), so it should incorporate a sum over possible labelings $\mathbf{y}$ but keep $\mathbf{x}$ fixed to the values in the training data. This term is the expected value of feature f_k under the model distribution $p(\mathbf{y}|\mathbf{x})\tilde{p}(\mathbf{x})$, or the expected number of times f_k should occur given a fixed corpus of sentences and a conditional probability distribution $p(\mathbf{y}|\mathbf{x})$ over label sequences for each sentence. Here, we must count the number of times f_k occurs in all possible label sequences $\mathbf{y}$ for each fixed $\mathbf{x}$, *but each occurrence must be weighted by the probability of seeing this* $\mathbf{y}$ *for that* $\mathbf{x}$.

The expectation is computed in the following manner. For each $\mathbf{x} \in \mathcal{D}$, for each position t in the sentence, and for all choices of labels y, y' for y_t, y_{t-1}, count the number of times f_k appears at t and weight by $p(y, y'|\mathbf{x})$, the probability of seeing this label configuration in the first place:

$$\mathbb{E}_{\mathbf{y},\tilde{\mathbf{x}}} f_k = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{y,y'} f_k(y, y', \mathbf{x_t}^{(i)}) \cdot p(y, y'|\mathbf{x}^{(i)}). \tag{20}$$

A few final remarks are necessary:

1. That the partial derivatives of the log likelihood are a difference of feature expectations is a general property of exponential families (Sutton and McCallum, 2006; Jordan, 2003). There is no simple intuition for why this is the case, but it does imply that the two expectations are equal at the unregularized maximum likelihood solution, which is satisfying because it means that expected feature counts suggested by the model match the ones in the data. Conceptually, maximum likelihood estimation finds a model that best explains the given data. This form of the gradient is one key implication of assuming the clique potential form in (5), and is a common theme in the literature.

2. The parameter estimation problem contains within it two key *inference* problems: computing $Z(\mathbf{x})$, which is needed in the likelihood; and computing the marginals on edges $p(y, y'|\mathbf{x}^{(i)})$, which is needed in the gradient. The link between estimation and inference is a deep theme in graphical models.

3. The function $\ell(\theta)$ is concave, so every local maximum is a global maximum. The addition of the regularizer makes ℓ *strictly* concave, so there is a unique global maximum that L-BFGS will find. Concavity of ℓ follows from the convexity of functions of the form $\log \sum_i \exp x_i$, so strict concavity of the objective function is a second key implication of (5). The objective in the prototype model will *not* be concave, as is often the case with unsupervised models, and this causes serious difficulties.

4. The first expectation in the gradient of the log likelihood only needs to be computed once, as it does not depend on any model parameters. All the other terms must be recomputed every time the parameters are modified during training.

5. Training CRFs is an area of study in its own right, as efficient methods for large datasets allow new applications (Wallach, 2002; Dietterich et al., 2004; Qi et al., 2005; Sutton and McCallum, 2007).

2.6 Inference in conditional random fields

The first inference problems of interest, computing $Z(\mathbf{x})$ and $p(y, y'|\mathbf{x})$, can be solved efficiently with a variant of the standard *forward-backward algorithm* for HMMs (Rabiner, 1989). Define *forward variables*

$$\alpha_t(j) := \sum_{i \in S} \psi_t(j, i, \mathbf{x_t}) \alpha_{t-1}(i), \tag{21}$$

with initialization $\alpha_1(j) = \psi_1(j, y_0, \mathbf{x_1})$ and where $\psi_t(j, i, \mathbf{x_t})$ is shorthand for $\psi(y_t = j, y_{t-1} = i, \mathbf{x_t})$. The variable $\alpha_t(j)$ captures the (unnormalized) "probability" of seeing the observations up to time t and then landing in state j. This can be computed recursively by looking at all source states i of j and summing over the cost of getting to i (the $\alpha_{t-1}(i)$) weighted by the cost of getting from i to j (the $\psi_t(j, i, \mathbf{x_t})$). Note that $\psi_t(y, y', \mathbf{x})$ incorporates what in an HMM would be two quantities: the transition probability of moving between states and the emission probability of emitting an observation from the destination.

It follows that

$$Z(\mathbf{x}) = \sum_{i \in S} \alpha_T(i),$$

where T is the end of the sequence, so only the forward pass is needed to compute the normalizer.

Similarly, define *backward variables*

$$\beta_t(i) := \sum_{j \in S} \psi_{t+1}(j, i, \mathbf{x_{t+1}}) \beta_{t+1}(i), \tag{22}$$

with initialization $\beta_T(i) = 1$. The variable $\beta_t(i)$ captures the (unnormalized) probability of being in state i at time t and seeing the observations from $t+1$ to the end of the sequence. This can be computed recursively by looking at all possible destinations j of i, and summing over the cost of being in j at time $t+1$ and seeing all subsequent observations, weighted by the cost of getting from i to j.

Both forward and backward passes are needed to compute the marginals $p(y, y'|\mathbf{x})$:

$$\xi_t(i, j) := p(y_t = i, y_{t+1} = j|\mathbf{x}) = \frac{\alpha_t(i) \cdot \psi_{t+1}(j, i, \mathbf{x_t}) \cdot \beta_{t+1}(j)}{Z(\mathbf{x})}. \tag{23}$$

This represents the probability of having an $i \to j$ transition at time t given the *entire* sequence of observations. Note that normalization is needed to ensure that the probability distribution sums to 1.

Usually, probabilities of being in a given state i at time t are also computed:

$$\gamma_t(i) := p(y_t = i|\mathbf{x}) = \frac{\alpha_t(i) \cdot \beta_t(i)}{Z(\mathbf{x})}. \tag{24}$$

Parameters representing the probability of starting or ending in various states are usually included in the model, in which case $\gamma_0(i)$ or $\gamma_T(i)$ would be used instead of $\xi_t(i,j)$ in the second term of (16). In this case, these probabilities are used to initialize α_0 and β_T rather than the defaults given.

Forward-backward solves these inference problems in polynomial time. The remaining inference task is that of labeling an unseen instance, which is done by finding the Viterbi decoding $\mathbf{y}^* = \operatorname{argmax}_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$, but as in HMMs, this only involves replacing summation with maximization in forward-backward.

The probabilities computed by the algorithm will be so small that in practice they will underflow a computer. For this reason, the forward-backward algorithm is usually implemented in log space, in which case the forward pass is

$$\log \alpha_t(j) = \bigoplus_{i \in S} \left(\log \psi_t(j, i, \mathbf{x_t}) + \log \alpha_{t-1}(i) \right), \tag{25}$$

where the other computations are modified similarly. Here $a \oplus b = \log(\exp(a) + \exp(b))$; in other words, products become sums, and sums are done with the $\oplus$ operator. For numerical stability, notice that $a \oplus b$ can be computed using the identity

$$a \oplus b = a + \log(1 + \exp(b - a)) = b + \log(1 + \exp(a - b)), \tag{26}$$

where the version of the identity with the smaller exponent is used at any given time.

3. Prototype-Driven Learning

Prototype-driven learning (Haghighi and Klein, 2006) is a primarily unsupervised approach where prototypical examples are specified for each target label but full sentences are not labeled. A Markov random field is used to learn how to label parts of speech from the Penn Treebank. This model achieved high performance for an unsupervised technique, so we discuss the approach in detail.

The key to prototype learning is an efficient injection of prior knowledge into the model. For example, *the* could be a prototype for DETERMINER. Distributional similarity (Schutze, 1995) is used to link words to similar prototypes, so if *turnip* is a prototype for NOUN, and the word *radish* is linked to *turnip* because they appear in similar contexts, then the model will push *radish* closer to NOUN. These links are encoded as features in a linear-chain Markov random field, which is trained in an unsupervised fashion. In practice, only two or three prototypes for each label are needed.

Haghighi and Klein (2006) report that adding prototype features provides substantial error rate reductions on several sequence labeling tasks, such as raising per-position accuracy in the case of English part of speech tagging with three prototypes per tag from 41.3% to 80.5% on the Penn Treebank. They also discuss Chinese POS tagging and an information extraction task, but we focus on English POS tagging here.

3.1 Distributional Similarity

Given a set of prototype features, distributional similarity is used to spread the prototypes to other words in the corpus. Distributional similarity is a well known method for grouping words based on local context. Words that appear in similar contexts are similar, so seeing "I eat apples" and "I eat pears" suggests that apples and pears are related because both appear after "eat." This allows prior knowledge to spread from a few dozen words to the entire corpus.

For each unique word, collect a context vector of the counts of the 500 most frequently occurring words conjoined with a direction and distance. Explicitly, first extract the 500 most commonly occurring words in the corpus; second, for any given word w_i at position i in the corpus, look at the two words before and after and if any of these are in the top 500 words, update w_i's context vector accordingly. This will produce a $|V| \times 2000$ matrix $\mathbf{A}$, where if the `(table, the-2)` entry is 5, it means *table* appeared two positions to the right of *the* 5 times. Here V is the vocabulary.

Find the singular value decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ and use the dot product between (normalized) left singular vectors (rows of $\mathbf{U}$) as a measure of distributional similarity. For each word w, the set of prototype words with similarity exceeding a fixed threshold is S_w, the prototypes for w. For each $z \in S_w$, add a feature PROTO $= z$ for each occurrence of w. For example, one might hope to add features like PROTO $=$ *said* to each occurrence of *reported* or *explained*. Each word is limited to have similarity features from its 5 most similar prototypes. Each prototype word is also its own prototype, so locking a prototype to a label pushes all similar words towards that label.

Because of the way prior knowledge is spread through the corpus, a generative model is used. Discriminative models have generally outperformed generative models, but these results usually come from the fully supervised setting. This model learns by leaning heavily on relationships between words; it is the only way that limited prior information in the form of a prototype list could be helpful in tagging non-prototype words. Distributionally similar words tend to have similar labels, and so a generative model that directly leverages these dependencies is a natural choice when this is the best information available. It would be conceivable to use a hidden Markov model, but the representation of an undirected model is more convenient and flexible because of the exponential representation.

3.2 Approach

Given an instance $\mathbf{x} = (x_1, \ldots, x_T)$ of words, we would like to predict the correct sequence of labels $\mathbf{y} = (y_1, \ldots, y_T)$. We construct a generative model $p(\mathbf{y}, \mathbf{x}|\theta)$ and maximize the log likelihood of the data

$$\mathcal{L}(\theta; \mathcal{D}) = \sum_{i=1}^{N} \log p(\mathbf{x}^{(i)}|\theta) = \sum_{i=1}^{N} \log \sum_{\mathbf{y}} p(\mathbf{y}, \mathbf{x}^{(i)}|\theta), \tag{27}$$

where $\mathbf{x}^{(i)}$ denotes training instance i and $\mathcal{D}$ denotes the training data as before. Note that the objective function here uses a *data* likelihood $p(x|\theta)$, not the conditional likelihood $p(y|x, \theta)$ as in CRFs. Crucially, this objective function will *not* be concave, so global maxima are not reachable, and the choice of initial parameters will determine which local maximum will be reached. Non-concave models are common in unsupervised learning and intelligent model initialization can determine performance.

3.2.1 Model Structure

The generative model is a linear-chain Markov random field as in Figure 3(b), given by

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} \psi(y_{t-1}, y_t)\psi(y_t, x_t), \tag{28}$$

where ψ is a clique potential taking the exponential form in (5). For part of speech tagging, the model is given second-order Markov structure by making the current state depend on the previous two states (a trigram model). In practice, this is implemented by making each state y a pair of labels like (DT,NN), which represents being in NN currently and having been in DT previously. With second order states, each state can only transition to states where the intermediate labels match, so the model can move from (X, Y) to (Y, Z) but not to (W, Z). If there are 45 labels, as in the Penn Treebank, there will be $45^2 \approx 2000$ states in the model with 45 destinations each.

There are two kinds of cliques in the model: "transitions" $\psi(y_{t-1}, y_t)$ and "emissions" $\psi(y_t, x_t)$. For transition edges like (DT,NN) — (NN,VBD), the *only* active feature is (DT,NN,VBD), an identifier for the transition. This means that $\psi(i, j)$ does not vary at different positions in the sequence, like HMMs but unlike CRFs. There will be $45^3 \approx 91000$ parameters for transition edges. The active features on an emission edge look like

$$(\{ \text{word=Table, suffix=le, capitalized} \}, (\text{DT,NN})); \tag{29}$$

in other words, the usual feature vector together with the relevant state. (Here the *word=Table* feature can be used because *Table* is (in this example) a word from the training data; unknown words could only have active features like *capitalized* or suffixes.) The feature vectors do not contain information about words at other positions in the sentence. If there are 10000 unique features in the training corpus, each of the 45^2 states will have a vector of 10000 parameters for each of these features, yielding roughly 20 million total parameters for the model.

3.3 Parameter Estimation and Inference

The primary difficulty is training the model. As before, numerical optimization (L-BFGS) is used[5], which requires the ability to compute $\mathcal{L}(\theta; \mathcal{D})$ and its gradient $\nabla\mathcal{L}$. Let

$$\text{score}(\mathbf{x}, \mathbf{y}) := \prod_{t=1}^{T} \psi(y_{t-1}, y_t)\psi(y_t, x_t); \tag{30}$$

5. In this situation, L-BFGS must be used instead of other algorithms like conjugate gradient, which will likely not converge even after many hundreds of iterations.

then computing $\mathcal{L}(\theta; \mathcal{D})$ involves computing

$$p(\mathbf{x}|\theta) = \sum_{\mathbf{y}} p(\mathbf{y}, \mathbf{x}|\theta) = \frac{1}{Z} \sum_{\mathbf{y}} \text{score}(\mathbf{x}, \mathbf{y}) \tag{31}$$

for each training instance $\mathbf{x}$. Except for Z, this expression is identical to the CRF normalizer $Z(\mathbf{x}) = \sum_{\mathbf{y}} \text{score}(\mathbf{x}, \mathbf{y})$, which can be computed efficiently. The normalizer is given by

$$Z = \sum_{\mathbf{x}} \sum_{\mathbf{y}} \text{score}(\mathbf{x}, \mathbf{y}), \tag{32}$$

where the sum over $\mathbf{x}$ ranges over all possible sequences of words (not just grammatical sentences). This sum must account for sequences of arbitrary length, but since extremely long sentences are rare, Z can be approximated by only accounting for sequences up to some fixed length L. This gives a lower bound

$$\hat{Z} = \sum_{\ell=1}^{L} \hat{Z}_\ell = \sum_{\ell=1}^{L} \sum_{\mathbf{x}:|\mathbf{x}|=\ell} \sum_{\mathbf{y}} \text{score}(\mathbf{x}, \mathbf{y}), \tag{33}$$

where $\hat{Z}_\ell$ is the normalizer for fixed length sequences of length ℓ and can be efficiently computed using dynamic programming as described below. See Smith and Eisner (2005) for more motivation of this approximation, which is a special case of a general approach called *contrastive estimation*.

The key idea in computing $\hat{Z}_\ell$ is that the same kind of caching can be used from standard forward-backward, but now instead of accounting for an emission of a fixed word $\psi(x_t, y_t)$ at time t, we must account for the emission of *all possible words*. Define a new forward recursion

$$\alpha_t(j) := \sum_{x \in V} \sum_{i \in S} \psi(i, j)\psi(j, x) \cdot \alpha_{t-1}(i) = \sum_{i \in S} \psi(i, j) \left(\sum_{x \in V} \psi(j, x) \right) \alpha_{t-1}(i) \tag{34}$$

where V is the vocabulary and the *transition cost* $\psi(i, j) \sum_x \psi(j, x)$ is now constant for fixed i, j across all times t, as all time-dependent parameters have been removed. The only new element is the presence of $\sum_x$, which only needs to be computed once for each i, j pair, as $\psi(i, j)$ no longer depends on x. Then

$$\hat{Z}_\ell = \sum_{i \in S} \alpha_\ell(i), \tag{35}$$

where, as in a CRF, computing the normalizer requires only the forward pass. This alteration is sufficient to compute the entire likelihood. (We omit a proof that this definition of α is correct.)

The derivatives of $\mathcal{L}$ with respect to each θ_j are again given by a difference of feature expectations:

$$\frac{\partial \mathcal{L}}{\partial \theta_j} = \sum_{\mathbf{x} \in \mathcal{D}} \left(\mathbb{E}_{\mathbf{y}, \tilde{\mathbf{x}}} f_j - \mathbb{E}_{\mathbf{x}, \mathbf{y}} f_j \right) \tag{36}$$

The first expectation is identical to the second expectation in (16) and can be computed as before. The second expectation is the expectation of feature f_j under the model's joint distribution over all $\mathbf{x}, \mathbf{y}$ pairs and is difficult to compute. Again, it is assumed that sentences longer than L have negligible probability mass, so expectations are computed for each fixed ℓ and mixed according to the distribution $p(|\mathbf{x}|)$:

$$\mathbb{E}_{\mathbf{x}, \mathbf{y}} f_j = \mathbb{E}_{p(|\mathbf{x}|)} \left[\mathbb{E}_{\mathbf{x}, \mathbf{y}|\ell} f_j \right] = \sum_{\ell=1}^{L} p(|\mathbf{x}| = \ell) \cdot \mathbb{E}_{\mathbf{x}, \mathbf{y}|\ell} f_j \tag{37}$$

where $p(|\mathbf{x}| = \ell) = \hat{Z}_\ell / \hat{Z}$. The final question that must be answered, then, is how to compute $\mathbb{E}_{\mathbf{x}, \mathbf{y}|\ell} f_j$ for fixed ℓ, and for this a backward recursion is needed.

A complication in the backward case is that in CRFs, $\beta_t(i)$ accounts for observations from $t+1$ up to some fixed T; here, L cannot be used in place of T because this would account for too many observations when $\ell < L$. This means a whole set of L backward recursions is needed. Define

$$\beta_t^\ell(i) := \sum_{j \in S} \psi(i,j) \left(\sum_{x \in V} \psi(j,x) \right) \beta_{t+1}^\ell(j), \tag{38}$$

where t ranges from 1 to ℓ and β_ℓ^ℓ should be initialized with state final costs. At time t in the backward pass, instead of updating β_t based on values of β_{t+1}, we update β_t^ℓ from β_{t+1}^ℓ for all $\ell \geq t$. Despite this added complication, the transition costs used are the same as in α and can be reused.

The probability of being in state i at time t given all possible observation sequences of length ℓ

$$\gamma_t^\ell(i) = \frac{\alpha_t(i) \cdot \beta_t^\ell(i)}{\hat{Z}_\ell} \tag{39}$$

is straightforward, but again L many are needed.

The ξ probabilities have an added complication. In all the other cases, it was possible to incorporate a sum over all observations as one quantity, which makes things very efficient to compute, but here it is necessary to compute them all separately. They must capture the probability of having an i — j edge at t and emitting a specific x from j, given all possible observations at other positions in a sequence of length ℓ. This is because feature counts at t depend on the specific x being emitted, and thus separate weights are needed for each word. In the CRF case, there was a fixed word present at each t, but here there are $|V|$, so

$$\xi_t^\ell(i,j,x) = \frac{\alpha_t(i) \cdot \psi(i,j)\psi(j,x) \cdot \beta_t^\ell(j)}{\hat{Z}_\ell} \tag{40}$$

depends on *five* parameters, though again it only makes sense to consider these for $\ell \geq t$.

It is exceedingly inefficient to compute all these probabilities separately (with $|V| = 10000$ and $L = 20$, there are about $10000 \times 45^2 \times 45 \times 20 \times 20 \approx 180$ billion unique values that must be recomputed in each L-BFGS iteration). The insight is that all the ξ's are unique, but they contain many common pieces, and since they are just being summed together, refactoring is possible:

$$\begin{aligned}
\mathbb{E}_{\mathbf{x},\mathbf{y}|\ell} f_k &= \sum_{t=1}^{\ell} \sum_{i,j} \sum_{x \in V} f_k \cdot \xi_t^\ell(i,j,x) && (41)\\
&= \sum_{t=1}^{\ell} \sum_{i,j} \sum_{x \in V} \frac{\alpha_t(i) \cdot \psi(i,j)\psi(j,x) \cdot \beta_t^\ell(j)}{\hat{Z}_\ell} \cdot f_k && (42)\\
&= \sum_{t=1}^{\ell} \sum_{i,j} \frac{\alpha_t(i) \cdot \psi(i,j) \cdot \beta_t^\ell(j)}{\hat{Z}_\ell} \cdot \left(\sum_{x} \psi(j,x) \cdot f_k \right), && (43)
\end{aligned}$$

where the inner sum over x can be cached (as one vector over all features for each state j), which gives very large speedups. The feature f_k is not factored out because its value implicitly depends on x. Other pieces can be moved as well, but pushing in $\sum_x$ and being able to *cache* this sum is the big improvement. Notice also that this inner sum does not depend on ℓ, so it can be reused for the other expectations as well. At time t in the backward pass, just as we update the $\beta_t^\ell(i)$ for all $\ell \geq t$, we also update $\mathbb{E}_{\mathbf{x},\mathbf{y}|\ell}$ for all $\ell \geq t$.

In (36), this expectation needs to be computed once for each training instance, but nothing in (41) depends on a given $\mathbf{x}$ (indeed, it sums over possible $\mathbf{x}$), so it can be computed once per maximizer iteration and then scaled up by N, the number of training instances.

	Tokens	
Setting	48K	193K
BASE	42.2	41.3
PROTO	79.1	80.5

Table 1: English POS tagging results from Haghighi and Klein (2006) measured by per-position accuracy.

We omitted the regularizer above, but a diagonal Gaussian prior is used again ($\sigma^2 = 0.5$), and the log likelihood and gradient must add this extra term. Given that computing the Viterbi labeling for unseen instances can be done in the same way as before, this completes our exposition of how to perform parameter estimation and inference for prototype learning. The computations required are significant and nontrivial to implement, but the model is quicker to train than it initially seems, and the dynamic programming algorithm is a relatively clean adaptation of standard forward-backward. From the description above, it should be clear how to adapt code for CRFs to implement this MRF model, at least in principle.

3.4 Implementation Details and Results

In this section, we describe some details of Haghighi and Klein's approach to English part of speech tagging.

The datasets used were either the first 48K tokens (2000 sentences) or 193K tokens (8000 sentences) of the Penn Treebank (Marcus et al., 1994). The model form used was the trigram tagger described above, and the following spelling features: exact word type, character suffixes up to length 3, *initial-capital*, *contains-hyphen*, and *contains-digit*. Again, the only features on transition edges were tag trigrams.

Prototypes were automatically extracted from the corpus by selecting for each label the top three occurring word types that were not given another label more often. This resulted in 116 prototypes for the 193K token setting. This method requires statistics from a labeled corpus, but it is also possible to build a prototype list by hand. The full prototype list used in their experiments is shown in Haghighi and Klein (2006).

Distributional similarity features were generated in the manner described above, and word frequency counts for the matrix $\mathbf{A}$ were collected from the entire WSJ portion of the Penn Treebank as well as the entire BLLIP corpus (Charniak et al., 2000). (Words that only occurred in the BLLIP corpus were ignored.) When performing the singular value decomposition, which can be done easily in Matlab, the top 250 singular vectors were used, so after the dimensionality reduction, the rows of $\mathbf{U}$ should be vectors of 250 elements each. The similarity threshold used to pick S_w for each w was 0.35, and only the top 5 most similar prototypes were used for words that had a large number of similar prototypes.

Before the prototype features are included, the problem is symmetric in the labels; without any prior information, the model does not know what any of these labels are, so they are all interchangeable. To break initial symmetry, it is necessary to randomly initialize all the weights, and Haghighi and Klein initialize the weights to be near one, with some random noise. Even when symmetry is broken, it is necessarily to map the model labels to target labels in order to evaluate model performance. The model may be able to tell, for example, that *the*, *a*, and *an* deserve to get the same label, but it has no way of knowing whether this label should be DT (determiner) or JJ (adjective). To avoid this problem, at evaluation time, the model's predicted labels are greedily permuted to maximize per-position accuracy on the dataset. Finally, because the results still depend on random initialization, reported performance numbers are averaged over 10 runs. In Table 1, BASE is the model without prototype features.

For reference, in our MALLET (McCallum, 2002) based implementation of this model, with $|V| = 10000$ and the maximum sequence length $L = 20$, it takes about 5 hours to complete one full L-BFGS iteration, of which about 1-1.5 hours are spent computing the ξ's and $\mathbb{E}_{\mathbf{x},\mathbf{y}}$; the rest of the modified forward-backward

algorithm runs in a few seconds. One should expect roughly on the order of 100 iterations of L-BFGS to get convergence.

An important practical consideration that applies to trigram taggers in general — but that is not explicitly part of the model — is the inclusion of a *start state*. Since the states here are pairs of labels, we add states (S,S) and (S,X) for each of the 45 labels X, such that every state sequence must begin with (S,S), the state (S,S) is only allowed to transition to (S,X), and the (S,X) transition to all (X,Y) as usual. This allows a natural representation of the fact that different labels are more or less likely to begin a sentence, and it is awkward to capture this using only the initial γ_0 probabilities here because each state implicitly accounts for some previous state that in this case does not exist. For example, if we want to say that sentences are likely to begin with determiners, which of the (X,DT) do we weight highly? By adding a start state, transition edges (S,S) — (S,X) with high weight indicate that label X is a popular way of starting a sentence, and the first word of each sentence is emitted from some (S,X) state, indicating that that word should be given a label of X. Explicitly, (S,S) is always at $t = 0$ in the lattice, and the (S,X) are always at $t = 1$; the other states appear from $t = 2$ to the end of the sequence.

4. Conclusion

We have surveyed the framework of graphical models and its application to sequence labeling tasks in natural language processing, including a detailed discussion of parameter estimation and inference in conditional random fields. We then discussed prototype-driven learning, a novel approach to primarily unsupervised sequence labeling, focusing on the details of model training that are not in Haghighi and Klein (2006). Though the Markov random field model is certainly not simple, and is rather difficult to actually implement, the mathematical description of the model is not mysterious. The approach needed to train the model and perform inference is shown to require a more complex but relatively clean adaptation of the standard algorithms for training conditional random fields. Surprisingly, the model is not much more time-consuming to train than a conditional random field.

Semi-supervised learning is intuitively appealing because it does not have the same crippling reliance on labeled corpora found in supervised learning, and does not try to blindly model data as in unsupervised learning. It is a natural compromise that fits many tasks, as we usually know a little but not enough to rely on. It has been an especially active area of recent research for all these reasons, but these approaches are often difficult to implement, reproduce, and scale to practical settings. Though prototypes provide a simple, compact, understandable, and declarative way of specifying prior information, the rest of the approach is not equally simple, and making such models more easily reproducible and scalable for practical applications is an area of active research.

Acknowledgements

We thank Fernando Pereira and Kedar Bellare who provided invaluable support throughout. Thanks also to Aria Haghighi for some useful comments.

References

Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research (JMLR)*, 6, 2005.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*, chapter 8. Springer, 2006.

David Blei, Andrew Ng, and Michael Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research (JMLR)*, 3, 2003.

Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien, editors. *Semi-supervised learning*. MIT Press, 2006.

Eugene Charniak. Statistical techniques for natural language parsing. *AI Magazine*, 18(4):33–44, 1997.

Eugene Charniak, Don Blaheta, Niyu Ge, Keith Hall, John Hale, and Mark Johnson. BLLIP 1987–89 WSJ corpus release 1. Linguistic Data Consortium, Philadelphia, PA, 2000.

Aron Culotta, David Kulp, and Andrew McCallum. Gene prediction with conditional random fields. Technical Report UM-CS-2005-028, University of Massachusetts, Amherst, April 2005.

Thomas G. Dietterich, Adam Ashenfelter, and Yaroslav Bulatov. Training conditional random fields via gradient tree boosting. In *Twenty-First International Conference on Machine Learning (ICML)*, 2004.

Alexander Genkin, David D. Lewis, and David Madigan. Large-scale Bayesian logistic regression for text categorization. Technical report, Department of Statistics, Rutgers University, 2004. `http://www.stat.rutgers.edu/ madigan/PAPERS/techno-06-09-18.pdf`.

Alexander Genkin, David D. Lewis, and David Madigan. Large-scale Bayesian logistic regression for text categorization. In *Technometrics*, 2006.

Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems (NIPS)*, volume 17, 2004.

Aria Haghighi and Dan Klein. Prototype-driven learning for sequence models. In *Proceedings of the Human Language Technology conference and the North American chapter of the Association for Computational Linguistics (HLT-NAACL)*, 2006.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, 2001.

David Heckerman. A tutorial on learning with bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, 1995.

Feng Jiao, Shaojun Wang, Chihoon Lee, Russ Greiner, and Dale Schuurman. Semi-supervised conditional random fields for improved sequence segmentation and labeling. In *Proceedings of the International Committee on Computational Linguistics and the Association for Computational Linguistics (COLING/ACL)*, 2006.

T Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1999.

Michael Jordan, editor. *Learning in Graphical Models*. MIT Press, Cambridge, MA, 1999.

Michael I. Jordan. Introduction to probabilistic graphical models. To appear, 2003.

Michael I. Jordan. Graphical models. In *Statistical Science (Special Issue on Bayesian Statistics)*, volume 19, pages 140–155, 2004.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall, 2nd edition, 2006.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pages 282–289, 2001.

John Langford. Tutorial on practical prediction theory for classification. *Journal of Machine Learning Research (JMLR)*, (6):273–306, March 2005.

Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. In *Mathematical Programming*, 1989.

Gideon Mann and Andrew McCallum. Simple, robust, scalable semi-supervised learning via expectation regularization. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2007.

Chris Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1994.

Andrew McCallum. Information extraction: Distilling structured data from unstructured text. *ACM Queue*, 3(9), November 2005.

Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

Einat Minkov, RC Wang, and William W Cohen. Extracting personal names from emails: Applying named entity recognition to informal text. In *Proceedings of the conference on Human Language Technology (HLT)*, 2005.

Tom Mitchell. Generative and discriminative classifiers: Naive bayes and logistic regression. http://www.cs.cmu.edu/t̃om/mlbook/NBayesLogReg.pdf, 2006.

Kevin Murphy. A introduction to graphical models. http://www.cs.ubc.ca/ murphyk/Bayes/bnintro.html, 2001.

Andrew Ng and Michael Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems (NIPS)*, volume 14, 2002.

Kamal Nigam, John Lafferty, and Andrew McCallum. Using maximum entropy for text classification. In *IJCAI-99 Workshop on Machine Learning for Information Filterting*, 1999.

Kamal Nigam, Andrew McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kauffman, 1988.

Yuan Qi, Martin Szummer, and Thomas P. Minka. Bayesian conditional random fields. In *The Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS 2005)*, 2005.

Lawrence Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–286, 1989.

Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 133–142, 1996.

Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail. In *AAAI'98 Workshop on Learning for Text Categorization*, Madison, Wisconsin, July 27 1998.

Hinrich Schutze. Distributional part-of-speech tagging. In *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*, March 1995.

Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1): 1–47, 2002.

Fabrizio Sebastiani. A tutorial on automated text categorisation. In *European Symposium on Telematics, Hypermedia and Artificial Intelligence (THAI)*, 1999.

Vikas Sindhwani and S. Sathiya Keerthi. Large scale semi-supervised learning svms. In *Proceedings of the ACM SIGIR Conference (SIGIR)*, 2006.

Noah A. Smith and Jason Eisner. Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of the Association for Computational Linguistics (ACL)*, 2005.

Mark Steyvers and Tom Griffiths. Probabilistic topic models. In T Landauer, D McNamara, S Dennis, and W Kintsch, editors, *Latent Semantic Analysis: A Road to Meaning*. Erlbaum, 2006.

Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2006.

Charles Sutton and Andrew McCallum. Piecewise pseudolikelihood for efficient training of conditional random fields. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2007.

Erik F. Tjong, Kim Sang, and Sabine Buchholz. Introduction to the CoNLL-2000 shared task: chunking. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning*, pages 127–132, 2000.

Jade Vinson, David DeCaprio, Matthew Pearson, Stacey Luoma, and James Galagan. Comparative gene prediction using conditional random fields. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

Hanna Wallach. Efficient training of conditional random fields. Master's thesis, Division of Informatics, University of Edinburgh, 2002.

Hanna M. Wallach. Conditional random fields: An introduction. Technical Report MS-CIS-04-21, Dept. of Computer and Information Science, University of Pennsylvania, 2004.

Xiaojin Zhu. Semi-supervised learning literature survey. Technical report, Computer Sciences, University of Wisconsin-Madison, 2006. http://www.cs.wisc.edu/∼jerryzhu/pub/ssl_survey.pdf.