GENERALIZED EXPECTATION CRITERIA FOR LIGHTLY SUPERVISED LEARNING

A Thesis Presented

by

GREGORY DRUCK

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2011

Computer Science

GENERALIZED EXPECTATION CRITERIA FOR LIGHTLY SUPERVISED LEARNING

A Thesis Presented

by

GREGORY DRUCK

Approved as to style and content by:

Andrew McCallum, Chair

James Allan, Member

David Smith, Member

Andrew Cohen, Member

Xiaojin Zhu, Member

Andrew Barto, Department Chair
Computer Science

To Mom, Dad, Shannon, and Grandma Ruth

ACKNOWLEDGMENTS

I am very grateful to my adviser Andrew McCallum for his guidance and support. Andrew's abundant energy and enthusiasm, his ability to instantly contribute insightful comments to any discussion, and his endless stream of interesting research ideas have been fundamental to my work. I am also grateful to the rest of my dissertation committee, James Allan, Andrew Cohen, David Smith, and Xiaojin (Jerry) Zhu, for providing thoughtful and helpful comments throughout the process.

I have collaborated with and learned from many others, including Kedar Bellare, Sameer Singh, Burr Settles, Kuzman Ganchev, João Graça, Mukund Narasimhan, Paul Viola, Xuerui Wang, and Gerome Miklau. I am especially grateful to Gideon Mann and Chris Pal, who acted as unofficial second advisers early in my career. I have also benefited from discussions and interactions with others at UMass, including Hanna Wallach, Fernando Diaz, Charles Sutton, Aron Culotta, Martin Allen, Pallika Kanani, David Mimno, Mike Wick, Karl Schultz, Limin Yao, Sebastian Riedel, Laura Dietz, and Tim Vieira.

I am fortunate to have loving and supportive friends and family. Thanks Mom, Dad, Shannon, Grandma Ruth, and the Kathermans. Thanks Naomi, the love of my life, for making me happy, and for being patient with me during the writing of this document.

Finally, I would like to acknowledge my sources of funding. This work was supported in part by the Center for Intelligent Information Retrieval, in part by Defense Advanced Research Projects Agency (DARPA) Machine Reading Program under Air Force Research Laboratory (AFRL) prime contract no. FA8750-09-C-0181, in part by

the Central Intelligence Agency, the National Security Agency and National Science Foundation under NSF grant #IIS-0326249, in part by NSF grant #CNS-0958392, in part by DARPA through the Department of the Interior, NBC, Acquisition Services Division, under contract #NBCHD030010, and in part by UPenn NSF medium IIS-0803847.

ABSTRACT

GENERALIZED EXPECTATION CRITERIA FOR LIGHTLY SUPERVISED LEARNING

SEPTEMBER 2011

GREGORY DRUCK

B.Sc., JOHNS HOPKINS UNIVERSITY

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew McCallum

Machine learning has facilitated many recent advances in natural language processing and information extraction. Unfortunately, most machine learning methods rely on costly labeled data, which impedes their application to new problems. Even in the absence of labeled data we often have a wealth of prior knowledge about these problems. For example, we may know which labels particular words are likely to indicate for a sequence labeling task, or we may have linguistic knowledge suggesting probable dependencies for syntactic analysis. This thesis focuses on incorporating such prior knowledge into learning, with the goal of reducing annotation effort for information extraction and natural language processing tasks.

We advocate constraints on expectations as a flexible and interpretable language for encoding prior knowledge. We focus on the development of Generalized Expectation (GE), a method for learning with expectation constraints and unlabeled data. We

explore the various flexibilities afforded by GE criteria, derive efficient algorithms for GE training, and relate GE to other methods for incorporating prior knowledge into learning. We then use GE to develop lightly supervised approaches to text classification, dependency parsing, sequence labeling, and entity resolution that yield accurate models for these tasks with minimal human effort. We also consider the incorporation of GE into interactive training systems that actively solicit prior knowledge from the user and assist the user in evaluating and analyzing model predictions.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Machine learning has facilitated many recent advances in natural language processing and information extraction. Most successful applications use *supervised* machine learning, in which the learning algorithm is provided with labeled data, or data that is annotated with correct outputs. Unfortunately, annotating data often requires a substantial amount of human effort. Annotation can be incredibly time-consuming, especially when outputs are complex structures like sequences or trees. Often annotation also requires special expertise such as an understanding of English dependency syntax for parsing or knowledge of conferences and journals for information extraction from the scientific literature. Finally, models trained using data from one domain, for instance text from newspapers, often perform poorly when applied to another, for instance text on the web. This suggests that it may be necessary to obtain new labeled data for every domain of interest.

Motivated by these challenges, there has been much interest in learning algorithms that leverage unlabeled data. In contrast to labeled data, unlabeled data is often easy to obtain at very low cost. For example, while annotating research papers is challenging, a large corpus of unlabeled research papers can be downloaded from the web. Two prominent paradigms for learning with unlabeled data are *unsupervised* and *semi-supervised* learning. Unsupervised learning employs unlabeled data only, while semi-supervised learning uses a combination of labeled and unlabeled data. Learning with unlabeled data presents numerous challenges, and both paradigms typically rely on data or modeling assumptions that may be violated in difficult problems.

Even in the absence of labeled data, however, we often have a wealth of prior knowledge about information extraction and natural language processing tasks of interest. Examples of such prior knowledge include:

- **text classification**: Most documents that contain the word *senate* should be labeled with the *politics* class.
- **dependency parsing**: *Nouns* are often dependents of *verbs*.
- **information extraction** (from citations): The word *ACM* should usually be part of a *journal* or a *conference*.
- **entity resolution**: Mentions of entities that have high string overlap are likely to refer to the same entity.

Clearly learning should be able to benefit from such information.

However, most existing approaches to incorporating prior knowledge into learning do so in a way that is unintuitive. For example, encoding prior knowledge by modifying the model or specifying a prior on parameters requires the practitioner to "speak" in a language that is difficult to interpret. Additionally, prior knowledge may not straightforwardly map into a labeling of the unlabeled data, meaning that a reduction to a supervised or semi-supervised learning problem may not be feasible.

How should prior knowledge be encoded? Our prior knowledge is typically about output variables, though it may not provide the values of any particular output variables. Instead, it provides *properties* of the distribution over latent output variables. For example, knowing that documents that contain *senate* should usually be labeled *politics* tells us that the label distribution in those documents should be skewed toward *politics*, but it does not tell us which documents are *politics* and which are not. The notion of a property of a distribution can be formalized as preferences on the value of an *expectation* under the distribution. For example, one might specify that in

expectation 90% of documents that contain the word *senate* should be labeled *politics*. Such *expectation constraints* provide a flexible and interpretable language for encoding prior knowledge. Learning algorithms can make use of expectation constraints by encouraging model predictions for unlabeled data to satisfy them.

This thesis focuses on the development of Generalized Expectation (GE), a framework for learning with expectation constraints and unlabeled data, its application to leveraging prior knowledge, and its incorporation into "human-in-the-loop" training systems, with the goal of reducing annotation effort for information extraction and natural language processing problems.

1.1 Contributions

The majority of this thesis focuses on the development of the Generalized Expectation (GE) framework.

- **Generalized Expectation Criteria** (Chapter 3): We provide a thorough exposition of the Generalized Expectation (GE) framework, including exploration of the various flexibilities afforded through the selection of constraint features and score functions, discussion of the relationship between GE and related methods, and development of improved methods for GE parameter estimation for arbitrary Conditional Random Fields (CRFs). These methods are then used to develop efficient GE training algorithms for tree-structured CRFs including linear chain CRFs (Chapter 6).

- **GE for Lightly Supervised Text Classification** (Chapter 4): We use GE to leverage prior knowledge about the label distribution for documents that contain particular words. Specifically, light supervision is provided in the form of "labeled input features" that denote that a word, such as *puck*, typically indicates a particular label, such as *hockey*. Experiments demonstrate that GE

is preferable to other methods for learning with labeled input features, and that, given limited annotation time, having an annotator label input features rather than complete documents typically yields a more accurate classifier.

- **GE for Lightly Supervised Dependency Parsing** (Chapter 5): We develop an efficient algorithm for GE training of CRFs that model distributions over trees. We then use GE to leverage linguistic prior knowledge (e.g. a *noun*'s parent is often a *verb*) in lightly supervised non-projective dependency parsing. This method outperforms complex "unsupervised" methods with a small number of intuitive constraints.

- **GE with Approximate Inference using MCMC** (Chapter 7): Exact GE training is intractable for large or loopy models, or when constraints consider more variables than model features do. To address these cases we explore using MCMC methods to approximate expectations and covariances for GE training. We conduct experiments on sequence labeling tasks, where a comparison with exact GE is possible, and additionally apply this method to entity resolution, where GE provides high accuracy with two simple constraints.

- **Compensating for Noise** (Chapter 8): We demonstrate that if the constraints are known to be noisy or imprecise, and the particular type of noise or imprecision can be characterized, then the accuracy of GE training can be increased by designing a "noise-tolerant" GE objective.

- **Empirical Comparison with Posterior Regularization** (Chapter 8): We provide an empirical comparison of GE and Posterior Regularization (PR), a related method that can be viewed as an approximation to GE. We find that although the two methods often give similar performance, PR often takes many more passes through the data to converge than GE.

In an effort to assist the practitioner in providing more useful and precise supervision, we also explore the use of interaction between the system and the practitioner.

- **Active Learning by Labeling Input Features** (Chapter 9): We develop an active learning algorithm in which the system asks the user to label input features rather than instances. We select input features for labeling by approximating the expected resulting reduction in model uncertainty. Experiments in sequence labeling demonstrate that the proposed active learning method outperforms passive learning with labeled input features as well as traditional active learning with labeled instances.

- **Toward Interactive Training and Evaluation** (Chapter 10): We envision a novel interactive training paradigm in which a practitioner provides light supervision, evaluates performance, and performs error analysis in a closed loop. We take first steps toward developing this paradigm by solving problems that can be cast as selecting small, representative samples of the data for the practitioner to inspect. To select such samples, we propose an approach that uses model predictions to perform stratified sampling. We evaluate the approach on classification and sequence labeling tasks, and find that accuracy evaluation effort can be reduced by as much as 53% when compared to random sampling.

1.2 Declaration of Previous Work

- The Generalized Expectation framework was first published under the name *Expectation Regularization* (XR) by Mann and McCallum [71]. I began working on the framework shortly after the submission of that paper. A version of Chapter 4 was published as (Druck, Mann, and McCallum, 2008) [25]. A version of Chapter 5 was published as (Druck, Mann, and McCallum, 2009) [26]. My particular contributions to the development of GE include generalizing the

framework to arbitrary CRFs, different types of constraints, and different score functions, developing improved algorithms for GE training, connecting GE to related methods, and applying GE to a variety of problems.

- The covariance computation algorithm in Chapter 5 appeared in a technical report (Druck and Smith, 2009) [29].
- The work in Chapter 7 was done in collaboration with Sameer Singh.
- A version of Chapter 9 was published as (Druck, Settles, and McCallum, 2009) [28].
- A version of the approximation to GE discussed in Chapter 3.3 was published as (Bellare, Druck, and McCallum, 2009) [5].
- A version of Chapter 10 is to appear as (Druck and McCallum, 2011) [27].

CHAPTER 2

BACKGROUND

In this section we review the background material necessary to understand this thesis. We start by reviewing undirected graphical models, including inference and parameter estimation, focusing on Conditional Random Fields [59]. We then discuss methods for learning with unlabeled data. Finally, we motivate using prior knowledge as light supervision, and describe the limitations of previous approaches.

2.1 Probabilistic Models

In this thesis, we focus on probability distributions over discrete random variables. We separate these random variables into *input* (or *evidence*) and *output* (or *query*) variables. Input variables are always observed. Output variables are the variables we aim to predict. We denote input and output random variables as $\mathbf{X} = \{X_1, \ldots, X_n\}$ and $\mathbf{Y} = \{Y_1, \ldots, Y_m\}$, and realizations of those variables as $\mathbf{x} = \{x_1, \ldots, x_n\}$ and $\mathbf{y} = \{y_1, \ldots, y_m\}$, respectively. Input variables take values in the set $\mathcal{X}$, while output variables take values in the set $\mathcal{Y}$. We often refer to an assignment to an output variable as a *label*. The *joint* probability distribution of the input and output variables is denoted $p(\mathbf{X}, \mathbf{Y})$. Recall that a probability distribution must satisfy $\forall_{\mathbf{x}} \forall_{\mathbf{y}}\ 0 \leq p(\mathbf{X}=\mathbf{x}, \mathbf{Y}=\mathbf{y}) \leq 1$ and $\sum_{\mathbf{x},\mathbf{y}} p(\mathbf{X}=\mathbf{x}, \mathbf{Y}=\mathbf{y}) = 1$. To simplify notation, we use the shorthand $p(\mathbf{X}=\mathbf{x}, \mathbf{Y}=\mathbf{y}) \equiv p(\mathbf{x}, \mathbf{y})$.

Naively modeling the probability distribution of a large number of variables is intractable. For example, the joint distribution of n binary-valued random variables contains 2^n elements. This affects both computational and statistical efficiency. For-

tunately, tractability can be obtained by assuming *conditional independence* among some variables. Variable X is conditionally independent of Y given Z if and only if $p(X, Y|Z) = p(X|Z)p(Y|Z)$.

2.1.1 Undirected Graphical Models

Graphical models are probabilistic models whose conditional independence assumptions are specified by a graph. In this thesis, we focus on *undirected graphical models*, also known as *Markov random fields*. An undirected graphical model defines a family of probability distributions that decompose into the product of *factors* $\Psi_a(\mathbf{x}_a, \mathbf{y}_a)$ that each provide a non-negative score for a subset of the variables a.

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \prod_{a \in \mathcal{F}} \Psi_a(\mathbf{x}_a, \mathbf{y}_a) \tag{2.1}$$

To ensure that $p(\mathbf{x}, \mathbf{y})$ is a valid probability distribution, the score $\prod_{a \in \mathcal{F}} \Psi_a(\mathbf{x}_a, \mathbf{y}_a)$ is normalized by the *partition function* Z, which is the sum of the scores of all $(\mathbf{x}, \mathbf{y})$.

$$Z = \sum_{\mathbf{x}, \mathbf{y}} \prod_{a \in \mathcal{F}} \Psi_a(\mathbf{x}_a, \mathbf{x}_a) \tag{2.2}$$

The corresponding undirected graph $\mathcal{G} = (V, E)$ for this family of distributions contains a vertex for each variable, $V = (\mathbf{X}, \mathbf{Y})$, and an edge between every pair of variables that participate in the same factor. The undirected graph specifies that random variables $\mathbf{X}_a$ are conditionally independent of $\mathbf{X}_b$ given $\mathbf{X}_s$, $\mathbf{X}_a \perp\!\!\!\perp \mathbf{X}_b \mid \mathbf{X}_s$, if every path from any node in $\mathbf{X}_a$ to any node in $\mathbf{X}_b$ includes a node in $\mathbf{X}_s$.

Unfortunately, generating the corresponding undirected graph for $p(\mathbf{x}, \mathbf{y})$ can introduce ambiguity about the precise factorization. A *factor graph* provides an unambiguous graphical representation of $p(\mathbf{x}, \mathbf{y})$ [57]. A factor graph is a bipartite graph $\mathcal{G} = (V, F, E)$, where V are vertices for random variables, F are special vertices for factors, and E are edges between variables and factors. Other types of edges are not

permitted. Factors F represent Ψ_a, and edges between a variable X_i and a factor Ψ_a specify that the variable participates in that factor, $X_i \in \mathbf{X}_a$.

In this thesis, we use a *log-linear* parametrization of the factors

$$\Psi_a(\mathbf{x}_a, \mathbf{y}_a) = \exp\Big(\boldsymbol{\theta}_a \cdot \mathbf{f}_a(\mathbf{x}_a, \mathbf{y}_a)\Big), \tag{2.3}$$

where $\boldsymbol{\theta}_a$ are *parameters* of the model, and $\mathbf{f}_a$ is a vector of *model feature functions*. Each feature function examines the variables $\mathbf{x}_a$ and $\mathbf{y}_a$ and returns a real value. Parameters $\boldsymbol{\theta}_a$ are often tied across multiple factors. To simplify notation, we stack all model features into a single column vector, where f_i returns 0 if it does not apply to a particular factor a. Consequently, we drop the a subscripts on $\mathbf{f}$ and $\boldsymbol{\theta}$ and assume that a is implicitly an argument to $\mathbf{f}$.

$$\Psi_a(\mathbf{x}_a, \mathbf{y}_a) = \exp\Big(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}_a, \mathbf{y}_a)\Big) \tag{2.4}$$

2.1.2 Generative and Discriminative Models

Models of the joint distribution of input and output variables, $p(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})$, are called *generative* models. In many applications, the aim is to compute probabilities of latent output variables conditioned on observed input variables. For example, we often want to find the maximum probability output variable assignment $\mathbf{y}^*$ given the observed input variables.

$$\mathbf{y}^* = \operatorname*{argmax}_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) \tag{2.5}$$

A *discriminative* model models the conditional distribution $p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta})$ directly. Intuitively, discriminative models do not "waste effort" modeling the marginal distribution of input variables $p(\mathbf{x})$ when they will always be observed. Theoretical analysis suggests that generative estimation is more sensitive to model misspecification than discriminative estimation [65]. In practice, discriminative models often yield higher

accuracy than generative models, and provide other practical advantages discussed in the next section.

2.1.3 Conditional Random Fields

Conditional Random Fields[1] (CRFs) [59] are discriminative, log-linear MRFs. A CRF defines a conditional probability distribution

$$p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) = \frac{1}{Z(\mathbf{x};\boldsymbol{\theta})} \prod_{a \in \mathcal{F}} \exp\left(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{y}_a, \mathbf{x}_a)\right). \tag{2.6}$$

Note that the partition function is now specific to the input variables $\mathbf{x}$.

$$Z(\mathbf{x};\boldsymbol{\theta}) = \sum_{\mathbf{y}} \prod_{a \in \mathcal{F}} \exp\left(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{y}_a, \mathbf{x}_a)\right) \tag{2.7}$$

In addition to focusing modeling effort on the query of interest, CRFs have two important practical advantages over generative, log-linear MRFs. First, the fact that $\mathbf{x}$ is always observed allows the features $\mathbf{f}$ to consider any part of $\mathbf{x}$ without increasing the complexity of inference. As a result, we use $\mathbf{x}$ rather than $\mathbf{x}_a$ for the remainder of this thesis. Second, for applications in natural language processing and information extraction, computing $Z(\mathbf{x};\boldsymbol{\theta})$ is often much easier than computing $Z(\boldsymbol{\theta})$. For example, computing $Z(\boldsymbol{\theta})$ might require summing over all possible English sentences and labelings of those sentences, whereas computing $Z(\mathbf{x};\boldsymbol{\theta})$ would require summing only over the set of all possible labelings for an observed sentence.

2.1.3.1 Exact Inference in CRFs

We next discuss inference in CRFs. Three inference problems of interest are 1) computing the probability of a particular output $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$, 2) computing marginal

[1] In the literature the term CRF is often used to refer specifically to a linear chain CRF. In this thesis CRF refers to an arbitrarily structured, conditional, log-linear Markov random field.

probability distributions for factors $p(\mathbf{y}_a|\mathbf{x};\boldsymbol{\theta}) = \sum_{\mathbf{y}_{\_a}} p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$, where $\mathbf{y}_{\_a}$ denotes an assignment to all output variables other than those in a, and 3) computing the maximum probability output $\mathbf{y}^* = \text{argmax}_{\mathbf{y}}\, p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$.

If the factor graph for the CRF is a tree, i.e. it is acyclic, then we can solve all three problems using message passing algorithms. In *Belief propagation* (BP), originally proposed by Pearl [86], a variable Y_i receives beliefs about what its value should be from its neighbors. This algorithm is also called the *sum-product algorithm*, and can be applied directly to factor graphs [57]. When the factor graph is a tree, BP can compute all factor marginals exactly in $O(n|\mathcal{Y}|^T)$ time, where n is the number of output variables and T is the maximum number of output variables that participate in a factor. BP also computes $Z(\mathbf{x};\boldsymbol{\theta})$, so it solves inference problems 1 and 2 above. Inference problem 3 can be solved with the *max-product algorithm*, which is identical to sum-product with each $\sum$ replaced by max. These message passing algorithms are often generalizations of well-known inference algorithms for particular model structures. For example, the Viterbi and Forward Backward algorithms for chain models [92] are instances of max-product and sum-product, respectively.

Factor graphs that contain cycles are often called *loopy* factor graphs. The *junction tree algorithm* [61] performs exact inference in any factor graph by running BP in a new acyclic factor graph with cluster variables that represent multiple original variables. Unfortunately running BP in this graph is often intractable, as a cluster variable that combines m original variables takes values in a set of size $|\mathcal{Y}|^m$. As a result, approximate inference algorithms are typically applied.

2.1.3.2 Approximate Inference in CRFs

There are many approximate inference algorithms for factor graphs. *Variational methods* [121] select the best approximation to the factor graph of interest from a family of factor graphs where exact inference is tractable. Inference is then performed

in the tractable model. *Loopy belief propagation* [79], which is BP applied to a loopy factor graph, can be viewed as a variational method.

An alternative approach is to estimate quantities of interest from samples drawn from $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$. For example, samples could be used to estimate marginal distributions $p(\mathbf{y}_a|\mathbf{x};\boldsymbol{\theta})$ by counting the number of occurrences of each assignment to $\mathbf{Y}_a$ in the sample and normalizing. Though we cannot in general sample from $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$ directly, *Markov Chain Monte Carlo (MCMC)* [2] methods can be used to mimic samples from $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$. MCMC methods collect samples while exploring the space of output variable assignments using a Markov chain. Specifically, each state in the Markov chain represents an assignment to the output variables. MCMC methods work when the invariant distribution of the Markov chain is $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$. This holds if the Markov chain is *irreducible* and *aperiodic*. We next review two prominent MCMC methods.

The *Metropolis-Hastings (MH)* algorithm considers new states sampled from a *proposal distribution*, denoted q. Specifically, a candidate state is first sampled from $q(\mathbf{y}'|\mathbf{y})$, where $\mathbf{y}$ and $\mathbf{y}'$ are current and candidate states, respectively. The candidate $\mathbf{y}'$ is *accepted*, or selected as the next state in the Markov chain, with probability

$$\mathcal{A}(\mathbf{y},\mathbf{y}') = \min\left(1, \frac{p(\mathbf{y}')q(\mathbf{y}|\mathbf{y}')}{p(\mathbf{y})q(\mathbf{y}'|\mathbf{y})}\right). \tag{2.8}$$

Otherwise it is *rejected*, and a new candidate state is generated.

The *Gibbs sampler* is a special case of MH in which the proposal distribution is

$$q(\mathbf{y}'|\mathbf{y}) = \begin{cases} p(y'_j|\mathbf{y}_{\text{-}j},\mathbf{x};\boldsymbol{\theta}) & \text{if } \mathbf{y}'_{\text{-}j} = \mathbf{y}_{\text{-}j} \\ 0 & otherwise, \end{cases} \tag{2.9}$$

where j is incremented (or reset) after each sample from q. Note that with this proposal distribution $\mathcal{A}(\mathbf{y},\mathbf{y}') = 1$, so $\mathbf{y}'$ is always accepted. Therefore the algorithm consists of iteratively sampling a new assignment to each output variable, conditioned

on the current assignments to all other output variables. Gibbs sampling is applicable to models where sampling from $p(y_j|\mathbf{y}_{-j}, \mathbf{x}; \boldsymbol{\theta})$ is tractable.

2.1.3.3 Supervised Parameter Estimation for CRFs

We next discuss estimating parameters $\boldsymbol{\theta}$ for CRFs. Recall that $\mathbf{x}$ and $\mathbf{y}$ denote assignments to all variables. Often data consists of N *independent and identically distributed instances* (iid) instances. In this case $p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) = \prod_{i=1}^{N} p(\mathbf{y}^i|\mathbf{x}^i; \boldsymbol{\theta})$.

In *supervised* parameter estimation, all input and output variables in the training data are observed. We refer to such training data as *labeled data*. The standard method for estimating parameters in CRFs is to maximize the log-likelihood of the labeled data, $\log p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta})$. To reduce overfitting to the training data, the likelihood is augmented with a prior on parameters $p(\boldsymbol{\theta})$. The complete log-likelihood is

$$\mathcal{L}(\boldsymbol{\theta}) = \log p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}). \tag{2.10}$$

The parameters that maximize $\mathcal{L}(\boldsymbol{\theta})$ are selected.

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmax}_{\boldsymbol{\theta}} \ \log p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}). \tag{2.11}$$

Note that the log-likelihood of the labeled data is

$$\log p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}) - \log \sum_{\mathbf{y}} \exp\big(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y})\big). \tag{2.12}$$

Therefore, the partial derivative of $\mathcal{L}(\boldsymbol{\theta})$ with respect to a parameter θ_j is

$$\frac{\partial}{\partial \theta_j} \mathcal{L}(\boldsymbol{\theta}) = f_j(\mathbf{x}, \mathbf{y}) - \mathrm{E}_{p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta})}[f_j(\mathbf{x}, \mathbf{y})] + \frac{\partial}{\partial \theta_j} \log p(\boldsymbol{\theta}), \tag{2.13}$$

where $f_j(\mathbf{x}, \mathbf{y}) = \sum_{a \in \mathcal{F}} f_j(\mathbf{x}, \mathbf{y}_a)$. In words, the partial derivative is the difference between f_j computed on the labeled data and the model expectation of f_j. Ignoring

the prior, the partial derivative is 0 when the model expectation matches the labeled data. The first term in Equation 2.13, $f_j(\mathbf{x}, \mathbf{y})$, is easily computed using the labeled data. The second term, $\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[f_j(\mathbf{x}, \mathbf{y})]$, requires inference. Note that

$$\begin{aligned}
\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[f_j(\mathbf{x}, \mathbf{y})] &= \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) f_j(\mathbf{x}, \mathbf{y}) \\
&= \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) \sum_{a \in \mathcal{F}} f_j(\mathbf{x}, \mathbf{y}_a) \\
&= \sum_{a \in \mathcal{F}} \sum_{\mathbf{y}_a} p(\mathbf{y}_a|\mathbf{x};\boldsymbol{\theta}) f_j(\mathbf{x}, \mathbf{y}_a) \sum_{\mathbf{y}_{-a}} p(\mathbf{y}_{-a}|\mathbf{x};\boldsymbol{\theta}) \\
&= \sum_{a \in \mathcal{F}} \sum_{\mathbf{y}_a} p(\mathbf{y}_a|\mathbf{x};\boldsymbol{\theta}) f_j(\mathbf{x}, \mathbf{y}_a), \qquad (2.14)
\end{aligned}$$

where the last step follows because $\sum_{\mathbf{y}_{-a}} p(\mathbf{y}_{-a}|\mathbf{x};\boldsymbol{\theta}) = 1$. Equation 2.14 shows that computing the expectations of the model features requires marginal distributions over the factors. We discussed computing factor marginals in Section 2.1.3.1.

Unfortunately, as shown in Equation 2.13, the partial derivative with respect to θ_j includes the other parameters $\boldsymbol{\theta}$. Therefore, there is no closed form solution to this optimization problem. Instead, numerical optimization is used. It can be shown that $\mathcal{L}(\boldsymbol{\theta})$ is concave in $\boldsymbol{\theta}$, and consequently simple gradient ascent can provide globally optimal parameters. Second-order optimization methods, specifically L-BFGS [68], have been shown to converge much faster than gradient ascent and other numerical methods for maximizing likelihood in CRFs [70].

A disadvantage of the *batch* parameter estimation methods described thus far is that they require inference over all variables in each iteration of numerical optimization. In some applications, the factor graph contains a number of disconnected components. For example, often the training data consists of a number of iid instances. In this case *online* parameter estimation can be applied. In online learning, parameters are updated after performing inference on each instance. A standard method for online learning is *stochastic gradient ascent*, in which the parameters are

updated in the direction of the gradient for each instance. For CRFs, this online parameter estimation method generally converges faster than batch methods [120].

Finally, we discuss the prior $p(\boldsymbol{\theta})$. Typical choices for $p(\boldsymbol{\theta})$ are zero-mean Gaussian or Laplace priors. Ignoring constant terms, these priors are defined as

$$\text{Gaussian}/L_2^2: \quad \log p(\boldsymbol{\theta}; \sigma) = -\frac{\|\boldsymbol{\theta}\|_2^2}{2\sigma^2} \tag{2.15}$$

$$\text{Laplace}/L_1: \quad \log p(\boldsymbol{\theta}; b) = -\frac{\|\boldsymbol{\theta}\|_1}{b}. \tag{2.16}$$

These priors can also be interpreted as regularization terms that penalize "large" parameter vectors according to some norm. Intuitively, L_2^2 regularization strongly penalizes large parameter values, preferring a solution where many parameters have moderate values. In contrast, an L_1 penalty is stronger than an L_2^2 penalty when θ_i is close to 0, and consequently L_1 regularization pushes more parameter values to 0, while tolerating a few large parameter values. This encourages sparsity in $\boldsymbol{\theta}$. However, note that $\|\boldsymbol{\theta}\|_1$ is not differentiable everywhere, and consequently requires the use of different numerical optimization algorithms [1].

2.1.3.4 Maximum Entropy Estimation

Recall that the entropy H of a probability distribution $p(\mathbf{x})$ is

$$\mathrm{H}(p(\mathbf{x})) = -\sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x}). \tag{2.17}$$

Log-linear models can also be derived from the *principle of maximum entropy* [8]. The principle states that of all possible distributions that encode some testable information about the observed random variables, we should choose the distribution with maximum entropy. Estimation using the principle of maximum entropy is often referred to as *maximum entropy estimation*.

Suppose that we have labeled data $(\mathbf{x}, \mathbf{y})$ and features $\mathbf{f}$, and we would like to find a model $p(\mathbf{y}|\mathbf{x})$ that matches the value of $\mathbf{f}$ computed on the labeled data in expectation. Let $\mathcal{C}$ be the set of all probability distributions that satisfy this constraint.

$$\mathcal{C} \equiv \left\{ p \in \Delta \mid \mathbf{f}(\mathbf{x}, \mathbf{y}) = \mathrm{E}_{p(\mathbf{y}|\mathbf{x})}[\mathbf{f}(\mathbf{x}, \mathbf{y})] \right\} \tag{2.18}$$

In general there may be many distributions in $\mathcal{C}$. Applying the principle of maximum entropy results in the following *primal* optimization problem over $p \in \mathcal{C}$

$$\max_{p \in \mathcal{C}} \ \mathrm{H}(p(\mathbf{y}|\mathbf{x})) = \max_{p \in \mathcal{C}} - \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \log p(\mathbf{y}|\mathbf{x}). \tag{2.19}$$

It can be shown [30, 87] that the corresponding *dual* optimization problem is

$$\max_{\boldsymbol{\theta}} \ \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}) - \log \sum_{\mathbf{y}} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y})) = \max_{\boldsymbol{\theta}} \ \log p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}). \tag{2.20}$$

Note that the dual problem is maximum likelihood in a CRF with factorization and parameterization specified by features $\mathbf{f}$ used in the primal problem.

With *generalized maximum entropy* estimation [30], we may penalize the difference between $\mathbf{f}(\mathbf{x}, \mathbf{y})$ and $\mathrm{E}_{p(\mathbf{y}|\mathbf{x})}[\mathbf{f}(\mathbf{x}, \mathbf{y})]$, rather than require equality. Mathematically this results in parameter regularization in the dual problem. For example, an L_2^2 penalty in the primal problem yields L_2^2 parameter regularization in the dual problem. Additional discussion of this method is provided in Sections 3.3.1 and 3.3.2.

2.1.4 Logistic Regression

A *logistic regression* model, also known as a *maximum entropy classifier*, models the probability of a single output variable y as

$$p(y|\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{Z(\mathbf{x}; \boldsymbol{\theta})} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y)). \tag{2.21}$$

Note that a logistic regression model can be viewed as a degenerate CRF where there are no dependencies among output variables. This allows us to use the term CRF to stand in for both structured and unstructured log-linear models.

2.2 Learning with Unlabeled Data

Unfortunately, obtaining labeled data is often challenging. Annotation is time-consuming, especially when output variables are structured. For many tasks, annotation also requires special expertise, for example knowledge of English syntax for parsing or the names of conferences and journals for information extraction from research papers. Consequently, annotation is often expensive. In addition, current machine learning methods are not robust to shifts in *domain*. For example, a sentiment polarity classifier trained using reviews of books may perform substantially worse when applied to reviews of movies [10]. This implies that it may be necessary to obtain new labeled data for each task-domain pair of interest. Even with new annotation platforms based on crowdsourcing like Amazon Mechanical Turk[2], this approach does not scale to the multitude of diverse problems we would like to solve.

As a result, there has been much interest in learning with *unlabeled* data. In contrast to labeled data, large amounts of unlabeled data can be obtained at very low cost. For example, while obtaining English sentences annotated with syntax is difficult, large numbers of unannotated English sentences can be obtained from the web. In the following sections, we survey strategies for learning with unlabeled data.

2.2.1 Unsupervised Learning

In contrast to supervised learning methods, which use only labeled data to estimate parameters, *unsupervised learning* methods use only unlabeled data. That is, in this setting we observe input variables $\mathbf{x}$ but do not observe output variables.

[2]http://www.mturk.com

Purely unsupervised learning with discriminative models is not straightforward. Marginalizing out the unobserved output variables $\mathbf{y}$ shows that unlabeled data has no effect on the likelihood for a discriminative model.

$$\log \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) = 0 \tag{2.22}$$

The predominant approach to unsupervised learning is to use a generative model. Because the output variables $\mathbf{y}$ are unobserved, a natural parameter estimation objective function is the *marginal likelihood* of the observed input variables.

$$\mathcal{L}_M(\boldsymbol{\theta}) = \log p(\mathbf{x};\boldsymbol{\theta}) = \log \sum_{\mathbf{y}} p(\mathbf{x},\mathbf{y};\boldsymbol{\theta}) \tag{2.23}$$

The *Expectation Maximization* (EM) algorithm [23] is often used to optimize $\mathcal{L}_M(\boldsymbol{\theta})$. EM can be viewed as the following optimization problem [81]:

$$\max_{q,\boldsymbol{\theta}} \; F(q,\boldsymbol{\theta}) = \max_{q,\boldsymbol{\theta}} \; -\mathrm{D}_{KL}\big(q(\mathbf{y}|\mathbf{x})||p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})\big) + \mathcal{L}_M(\boldsymbol{\theta}) \tag{2.24}$$

The EM algorithm alternates between *Expectation* and *Maximization* steps, which correspond to optimizing $F(q,\boldsymbol{\theta})$ with respect to q and $\boldsymbol{\theta}$, respectively.

$$\mathbf{E-Step}\colon \; \hat{q}^{t+1} = \operatorname*{argmax}_{q} F(q,\hat{\boldsymbol{\theta}}^t) = p(\mathbf{y}|\mathbf{x};\hat{\boldsymbol{\theta}}^t) \tag{2.25}$$

$$\mathbf{M-Step}\colon \; \hat{\boldsymbol{\theta}}^{t+1} = \operatorname*{argmax}_{\boldsymbol{\theta}} F(\hat{q}^{t+1},\boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{\mathbf{y}} \hat{q}^{t+1}(\mathbf{y}|\mathbf{x}) \log p(\mathbf{x},\mathbf{y};\boldsymbol{\theta}) \tag{2.26}$$

In the E-step, $\hat{q}^{t+1} = p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}^t)$ because there are no restrictions on the form of q, and this solution has zero KL divergence. The M-step is maximum likelihood estimation with the latent variables $\mathbf{y}$ "filled in" by $\hat{q}^{t+1}$. If computing the posterior $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$ is intractable, then a variational approximation can be used [121].

An alternative approach to unsupervised learning with generative models is to use MCMC methods to estimate parameters [2]. For example, this approach is often used for Latent Dirichlet Allocation (LDA) [9], a generative model of documents.

Unsupervised learning is often used to discover latent structure in data. For example, LDA can discover latent *topics*, or distributions over words, that are often semantically coherent and useful for understanding a document collection.

However, when the goal is to make predictions for a particular task, unsupervised learning is often challenging. The true process that generated the unlabeled data is typically unknown, or difficult to model tractably, and as a result the model $p(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})$ approximates the true process. The learning algorithm then implicitly assumes that the model is correct. Therefore, maximizing $\mathcal{L}_M(\boldsymbol{\theta})$ may not yield posterior distributions $p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta})$ that are relevant to the task of interest. Successful applications of unsupervised learning to solve concrete tasks rely on carefully designed models and learning algorithms [46, 89].

2.2.2 Semi-Supervised Learning

Semi-supervised learning compromises between supervised and unsupervised learning by leveraging both labeled and unlabeled data. As in unsupervised learning, assumptions are required to incorporate unlabeled data into parameter estimation. Common assumptions are the *cluster assumption*, the *low-density separation assumption*, the *manifold assumption*, and the *multi-view assumption*. Below we discuss examples of prominent methods that make each of these assumptions, and situations in which these assumptions are violated, which may cause these methods to fail. There has been much additional work in semi-supervised learning. For more information the reader is referred to [16, 128].

As with unsupervised learning, probabilistic semi-supervised learning is more straightforward with a generative model. Generative semi-supervised learning meth-

ods make the *cluster assumption*, which informally states that the values of the latent output variables inferred by the model will be correlated with the true labels. In semi-supervised learning, we have a mixture of observed and latent output variables, which we denote as $\mathbf{y}_L$ and $\mathbf{y}_U$ (for labeled and unlabeled), respectively. We estimate parameters by maximizing

$$\mathcal{L}_M(\boldsymbol{\theta}) = \log p(\mathbf{x}, \mathbf{y}_L; \boldsymbol{\theta}) = \log \sum_{\mathbf{y}_U} p(\mathbf{x}, \mathbf{y}_L, \mathbf{y}_U; \boldsymbol{\theta}) \tag{2.27}$$

Either the EM algorithm or direct gradient-based optimization can be applied to this objective function. Nigam et al. [83] apply EM to semi-supervised text classification with a naive Bayes model and obtain significant improvements over supervised learning. However, note that semi-supervised learning by maximizing marginal likelihood may fail when the model is misspecified, violating the cluster assumption [18, 77].

Note that if there are dependencies between the observed and latent output variables, then we can also perform semi-supervised learning in a CRF by optimizing the marginal likelihood of observed output variables conditioned on the input variables.

$$\mathcal{L}_{MD}(\boldsymbol{\theta}) = \log p(\mathbf{y}_L|\mathbf{x}; \boldsymbol{\theta}) = \log \sum_{\mathbf{y}_U} p(\mathbf{y}_L, \mathbf{y}_U|\mathbf{x}; \boldsymbol{\theta}) \tag{2.28}$$

The partial derivative of $\mathcal{L}_{MD}(\boldsymbol{\theta})$ is the difference of two expectations. The first is the model expectation over the latent output variables of f_j. The second is the expectation over all output variables of f_j.

$$\frac{\partial}{\partial \theta_j}\mathcal{L}_{MD}(\boldsymbol{\theta}) = \mathrm{E}_{p(\mathbf{y}_U|\mathbf{y}_L,\mathbf{x};\boldsymbol{\theta})}[f_j(\mathbf{y}_L, \mathbf{y}_U, \mathbf{x})] - E_{p(\mathbf{y}_L,\mathbf{y}_U|\mathbf{x};\boldsymbol{\theta})}[f_j(\mathbf{y}_L, \mathbf{y}_U, \mathbf{x})] + \frac{\partial}{\partial \theta_j} \log p(\boldsymbol{\theta}).$$

Training with $\mathcal{L}_{MD}(\boldsymbol{\theta})$ can be performed using the expected gradient algorithm [98]. Note, however, that with iid instances where each instance is either fully labeled or fully unlabeled, this reduces to supervised parameter estimation.

An alternative formulation of the cluster assumption that is more natural for discriminative training is the *low-density separation assumption*, which states that the decision boundary should lie in a region of low-density. Examples of methods that make this assumption include *Entropy Regularization* (ER) [41]. Entropy regularization augments the likelihood with a term that encourages low entropy or peaked distributions over latent variables.

$$\mathcal{O}_{ER}(\boldsymbol{\theta}) = \log p(\mathbf{y}_L|\mathbf{x};\boldsymbol{\theta}) - \lambda \mathrm{H}(p(\mathbf{y}_U|\mathbf{x};\boldsymbol{\theta})) \tag{2.29}$$

Either direct optimization or an EM-like algorithm can be used to maximize $\mathcal{O}_{ER}(\boldsymbol{\theta})$ [41]. *Transductive Support Vector Machines* [51], which aim to find the maximum margin decision boundary that separates both the labeled training and unlabeled test instances, also make a low density separation assumption. In general, these methods may fail in difficult problems with high class overlap.

Another class of methods assume that the data lies on a low-dimensional manifold, and that instances that are close on the manifold should have the same output. Graph-based semi-supervised learning methods [4, 129] build a graph in which nodes are instances and edge weights denote similarity between instances. A labeling function is then learned that matches the labels of labeled instances and varies smoothly over the graph. Because the graph is typically constructed so that it only preserves local distances, these methods are said to make the *manifold assumption*. These methods may fail when the data does not lie on a low-dimensional manifold, or when data sparsity prevents the discovery of the correct manifold structure.

Multi-view semi-supervised learning methods assume that there are multiple independent views, i.e. multiple sets of features, and may additionally assume that each view is sufficient to provide accurate predictions. *Co-training* [11] encourages predictions of models that use each view to agree on unlabeled data. This effectively reduces the hypothesis space and consequently permits learning with less labeled data. How-

ever, problems of interest may not have multiple views, and one view in isolation may be insufficient for learning.

As a result of the above assumptions, semi-supervised learning methods tend to be delicate and require data-specific tuning. Mann and McCallum [71] note that there have been very few published papers that describe a successful application of semi-supervised learning.

2.2.3 Lightly Supervised Learning

Even in the absence of labeled data we often have a wealth of prior knowledge about information extraction and natural language processing tasks of interest. Examples of such prior knowledge include:

- **text classification**: Most documents that contain the word *senate* should be labeled with the *politics* class.
- **dependency parsing**: *Nouns* are often dependents of *verbs*.
- **information extraction** (from citations): The word *ACM* should usually be part of a *journal* or a *conference*.
- **entity resolution**: Mentions of entities that have high string overlap are likely to refer to the same entity.

An emerging paradigm that we refer to as *lightly supervised learning* aims to leverage such prior knowledge in learning with unlabeled data. In this section we provide a survey of different classes of methods for lightly supervised learning. We then illustrate the limitations of these methods for incorporating general prior knowledge, motivating the methods we develop in Chapter 3.

Note that in this section we focus on broad classes of approaches. More detailed descriptions of lightly supervised approaches to particular tasks are provided in subsequent chapters. Methods that are closely related to GE are described in Section 3.3.

2.2.4 Labeling Data Approach

A simple approach is to use prior knowledge to label data, essentially reducing a lightly supervised learning problem to a supervised or semi-supervised learning problem. This can often be viewed as using a rule-based system to perform annotation.

A prominent method that takes this approach is *prototype-driven learning* [44, 45]. This method uses prototypes, for example a list of words that are highly indicative of each label, to partially label unlabeled data. Parameters of a generative model are then estimated to maximize the marginal likelihood of the partial labeling and the input variables. To improve accuracy, this method also uses cluster features that help supervision to propagate from the prototypes to other related contexts.

Other related approaches include using prior knowledge to induce a soft-labeling of the unlabeled data [24, 67, 101], or to label the unlabeled data and assign confidence scores that can be incorporated into learning [125]. We discuss these methods in more detail in Section 4.3.

The limitation of this approach is that in general it is often unclear how to convert prior knowledge into a labeling. Consider our prior knowledge about the word *ACM*, described above. For any particular occurrence of the word *ACM*, it is not clear whether it should be labeled *journal* or *conference*. In fact, the occurrence could be labeled with a different label, as the knowledge only *usually* applies.

2.2.5 Bayesian Approach

Another possible approach is to use prior knowledge to specify a prior on model parameters $p(\boldsymbol{\theta})$. For example, Dayanik, et al. [21] propose several methods that use known associations between words and labels to specify prior distributions on the parameters of a logistic regression model for text classification.

The limitation of this approach is that our prior knowledge is often not about parameter values. Parameter values are difficult to interpret as a result of complex

interactions within the model. Consequently, designing a prior on parameters that achieves a particular desired effect is challenging.

2.2.6 Modeling Approach

Finally, one can attempt to incorporate prior knowledge directly into the model. Many successful applications of unsupervised learning can be viewed as taking this approach, as they often involve complex generative models whose specification is based on prior knowledge about the task [45, 89].

Unfortunately, understanding the relationship between the structure of the model and the resulting distribution over latent output variables can be challenging. Consequently it can be difficult to obtain a particular desired effect, especially for a practitioner who is not a machine learning expert.

CHAPTER 3

GENERALIZED EXPECTATION CRITERIA

As discussed in Chapters 1 and 2, obtaining labeled data sets for each task and domain of interest is not feasible. While labeled data may be limited, unlabeled data is typically easy to obtain. In addition, we have an abundant amount of prior knowledge about most tasks of interest. However, most existing approaches to learning with unlabeled data do not take advantage of such information, and approaches that do require the practitioner to incorporate prior knowledge in an unintuitive way.

In this chapter, we propose constraints on model expectations as a more natural, declarative language for encoding prior knowledge. We then describe Generalized Expectation (GE), a flexible framework for specifying and learning with preferences about model expectations. We also discuss connections between GE and related frameworks.

3.1 Expectations

Suppose we would like to build a system to extract information from citations of research papers such as the *authors*, *title*, *journal*, etc. We have a wealth of prior knowledge about this task. For example, we know that "the word *ACM* should *usually* be part of a *conference* or a *journal*." We aim to develop methods for learning with unlabeled data that leverage this information.

The methods described in Section 2.2.3 require prior knowledge to be encoded in terms of model structures or parameter values, which can be difficult to interpret. In actuality, our prior knowledge is about output variables. However, as also discussed

in Section 2.2.3, we do not know the values of particular output variables, as it is not always clear how to label data with our prior knowledge. For example, consider our prior knowledge about the word *ACM*. For any particular occurrence of *ACM*, we do not know whether it should be labeled *conference*, *journal*, or something else.

However, if given a distribution over the latent output variables, we can evaluate how well it respects our prior knowledge. For example, given one distribution where *ACM* is usually part of a *conference* or a *journal*, and another where *ACM* is usually part of a *title*, we know that the first distribution is preferable. That is, our prior knowledge tells us desirable properties of the distribution over latent output variables.

We formalize this idea using *expectations*. The expectation of a feature $\phi(\mathbf{x}, \mathbf{y})$ under a conditional distribution p is

$$\mathrm{E}_{p(\mathbf{y}|\mathbf{x})}[\phi(\mathbf{x}, \mathbf{y})] = \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})\phi(\mathbf{x}, \mathbf{y}). \tag{3.1}$$

We call ϕ a *constraint feature* to distinguish it from a *model feature* f.

In order to incorporate our prior knowledge into learning we specifically evaluate properties of the model distribution over latent output variables

$$\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\phi(\mathbf{x}, \mathbf{y})] = \mathrm{E}_{\boldsymbol{\theta}}[\phi] = \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})\phi(\mathbf{x}, \mathbf{y}), \tag{3.2}$$

where $\mathrm{E}_{\boldsymbol{\theta}}[\phi]$ is shorthand. We describe how to specify preferences about the values of these expectations in Section 3.2.

As with model features, the constraint features typically decompose into the sum of local constraint features that consider subsets of output variables a. We denote the set of variable subsets as $\mathcal{F}$.

$$\phi(\mathbf{x}, \mathbf{y}) = \sum_{a \in \mathcal{F}} \phi(\mathbf{x}, \mathbf{y}_a) \tag{3.3}$$

Therefore, the expectation of the constraint feature can be written

$$\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\phi(\mathbf{x},\mathbf{y})] = \sum_{a\in\mathcal{F}} \sum_{\mathbf{y}_a} p(\mathbf{y}_a|\mathbf{x};\boldsymbol{\theta})\phi(\mathbf{x},\mathbf{y}_a), \tag{3.4}$$

where $p(\mathbf{y}_a|\mathbf{x};\boldsymbol{\theta})$ are marginal probabilities of output variables in subset a.

The constraint features, like model features, are arbitrary functions of the variables that return a real value. The selection of the particular subsets of variables to sum over $\mathcal{F}$, the number of output variables that the local constraint feature takes as input $|a|$, and the attributes of variables the local constraint features compute collectively provide the flexibility to define many different types of expectations. For example, the constraint feature might only consider a particular subset of variables. If the data consists of multiple iid samples, then this type of constraint feature can be used to define an *instance-specific* expectation, rather than a *corpus* expectation.

It is important to note that there need not be any correspondence between constraint and model feature functions. The constraint features may examine different aspects of the variables or take different subsets of variables as input, and there may be more or fewer constraint features than model features. This flexibility can be leveraged to train a feature-rich model with a small number of expectation constraints, or to train a simple model with more expressive expectation constraints.

3.1.1 Input Feature Label Distributions

In this section we describe a constraint feature that is used extensively in this thesis. Note, however, that the learning methods developed in the remainder of this chapter apply to arbitrary constraint features.

Following the information extraction example from the previous section, suppose we would like to evaluate the expected label distribution for *ACM*. This information extraction problem can be cast as sequence labeling, so input and output variables are arranged into sequences. For the purposes of this example, we assume that all

citations are concatenated into a single sequence of length n. We first define a local *input feature* whose value is 1 when the token at some position j in the input sequence is *ACM*.

$$q_{ACM}(\mathbf{x}, j) = 1_{\{x_j = ACM\}} \tag{3.5}$$

The indicator function $1_{\{P\}}$ returns 1 if the predicate P is true, and 0 otherwise.

In this example, we are interested in individual output variables, so each subset a contains a single index. We define a local constraint feature whose value is 1 when the token at j is *ACM*, and the label y is *journal*.

$$\phi_{journal,ACM}(\mathbf{x}, y, j) = 1_{\{y=journal\}} q_{ACM}(\mathbf{x}, j) \tag{3.6}$$

The complete constraint feature, which sums the local constraint feature over all positions in the sequence is

$$\phi_{journal,ACM}(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{n} \phi_{journal,ACM}(\mathbf{x}, y_j, j). \tag{3.7}$$

The expectation of this constraint feature is

$$\begin{aligned} \mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\phi_{journal,ACM}(\mathbf{x}, \mathbf{y})] &= \sum_{j=1}^{n} \sum_{y_j} p(y_j|\mathbf{x};\boldsymbol{\theta}) \phi_{journal,ACM}(\mathbf{x}, y_j, j) \\ &= \sum_{j=1}^{n} p(Y_j = journal|\mathbf{x};\boldsymbol{\theta}) q_{ACM}(\mathbf{x}, j), \end{aligned} \tag{3.8}$$

where $p(y_j|\mathbf{x};\boldsymbol{\theta})$ is the marginal of the jth label. Notice that because $\phi_{journal,ACM}$ returns 0 for any token that is not *ACM* or any label that is not *journal*, the expectation will simply be the sum of the marginal probabilities of *journal* for each

ACM token. We can make this expectation easier to interpret by normalizing by the number of *ACM* tokens. We modify the constraint feature as follows

$$\phi_{journal,ACM}(\mathbf{x}, \mathbf{y}) = \frac{1}{c_{q_{ACM}}} \sum_{j=1}^{n} \phi_{journal,ACM}(\mathbf{x}, y_j, j) \tag{3.9}$$

$$c_{q_{ACM}} = \sum_{j=1}^{n} q_{ACM}(\mathbf{x}, j). \tag{3.10}$$

The expectation of $\phi_{journal,ACM}(\mathbf{x}, \mathbf{y})$ is now a probability. By defining similar constraint features for all other labels, we can compute a label distribution for *ACM*.

We refer to this expectation as an *input feature label distribution*. We use this expectation throughout this thesis to encode prior knowledge about the assignments to output variables that particular input features suggest.

Generalizing, we may use an arbitrary binary input feature $q(\mathbf{x}, a)$, arbitrary, fixed-size subsets a of the output variables, and an indicator function that returns 1 for a particular assignment $\boldsymbol{\ell}$ to $\mathbf{Y}_a$. The general constraint feature is

$$\phi_{\boldsymbol{\ell},q}(\mathbf{x}, \mathbf{y}) = \frac{1}{c_q} \sum_{a \in \mathcal{F}} 1_{\{\mathbf{y}_a = \boldsymbol{\ell}\}} q(\mathbf{x}, a). \tag{3.11}$$

The expectation of this constraint feature is

$$\begin{aligned} \mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\phi_{\boldsymbol{\ell},q}(\mathbf{x}, \mathbf{y})] &= \frac{1}{c_q} \sum_{a \in \mathcal{F}} \sum_{\mathbf{y}_a} p(\mathbf{y}_a|\mathbf{x}; \boldsymbol{\theta}) 1_{\{\mathbf{y}_a = \boldsymbol{\ell}\}} q(\mathbf{x}, a) \\ &= \frac{1}{c_q} \sum_{a \in \mathcal{F}} p(\mathbf{Y}_a = \boldsymbol{\ell}|\mathbf{x}; \boldsymbol{\theta}) q(\mathbf{x}, a). \end{aligned} \tag{3.12}$$

As above, we define a constraint feature for each possible $\boldsymbol{\ell}$. To simplify notation, we stack all k constraint features into a vector

$$\boldsymbol{\phi}(\mathbf{x}, \mathbf{y}) = \begin{pmatrix} \phi_{\boldsymbol{\ell}_0}(\mathbf{x}, \mathbf{y}) \\ \vdots \\ \phi_{\boldsymbol{\ell}_k}(\mathbf{x}, \mathbf{y}) \end{pmatrix}. \tag{3.13}$$

Note that we may define a *default* q that always returns 1, $q(\mathbf{x}, a) = 1$. In this case we obtain an overall label distribution for the subsets in $\mathcal{F}$.

3.2 Generalized Expectation Criteria

This section describes the method we use to specify preferences about model expectations. This method involves defining an objective function that has a larger value when model expectations better respect our preferences, and estimating parameters to maximize that objective function. Note that in addition to the terminology introduced below, in this thesis we often refer to preferences about model expectations as *constraints*. Importantly, we do not mean "hard" constraints that must be satisfied.

In this section we assume that constraint features are stacked into a column vector $\boldsymbol{\phi}$. We make no assumptions about the functional form of the constraint features.

Generalized Expectation (GE) Criteria are terms in a parameter estimation objective function that express preferences on the value of a model expectation of some function. These preferences are expressed through a *score function* S that takes as input the expectations of the constraint features and returns a real value.

$$\mathrm{S}(\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\boldsymbol{\phi}(\mathbf{x}, \mathbf{y})]) \tag{3.14}$$

A higher score is preferable to a lower score.

In general S could be an arbitrary function. In this thesis, we focus on score functions that encourage model expectations of the constraint features to be close to, or within some range of, a target expectation vector $\tilde{\boldsymbol{\phi}}$. It is often convenient to view this type of score function as the negative of a *penalty function*. Target expectations $\tilde{\boldsymbol{\phi}}$ could be provided directly by a user, or automatically derived from a more coarse-grained target (see Chapters 4 and 9, and Section 8.2). Target expectations could also be estimated from data, though note that an advantage of using prior knowledge is

that estimation error can be reduced or eliminated. That is, it may take a substantial amount of labeled data to estimate a target expectation that is as accurate as our prior expectation, especially if the constraint feature occurs infrequently.

We next provide several concrete examples of score functions.

- **α-Divergence Penalty:** If the expectation and target vectors are probability distributions, as when using input feature label distributions, then we can penalize divergence from the targets using an *α-divergence* [78]. For simplicity of notation, below we assume that each element of the constraint feature vector corresponds to an element of the distribution. With multiple constraints the complete score function is the sum of the score functions for each constraint.

$$\lim_{\boldsymbol{\alpha}\to\mathbf{1}} \textbf{ / KL Div}: \ \mathrm{S}_{KL}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) = -\mathrm{D}_{KL}\Big(\tilde{\boldsymbol{\phi}} \,||\, \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\Big) = \tilde{\boldsymbol{\phi}}^{\mathrm{T}} \log(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) - \tilde{\boldsymbol{\phi}}^{\mathrm{T}} \log(\tilde{\boldsymbol{\phi}}) \tag{3.15}$$

$$\lim_{\boldsymbol{\alpha}\to\mathbf{0}} \textbf{ / Rev. KL Div}: \ \mathrm{S}_{RKL}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) = -\mathrm{D}_{KL}\Big(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}] \,||\, \tilde{\boldsymbol{\phi}}\Big) = \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]^{\mathrm{T}} \log(\tilde{\boldsymbol{\phi}}) - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]^{\mathrm{T}} \log(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) \tag{3.16}$$

$$\boldsymbol{\alpha}\mathbf{=0.5} \textbf{ / Hellinger Dist}: \ \mathrm{S}_{H}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) = -\mathrm{D}_{H}\Big(\tilde{\boldsymbol{\phi}} \,||\, \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\Big) = -2\Big\|\sqrt{\tilde{\boldsymbol{\phi}}} - \sqrt{\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]}\Big\|_2^2, \tag{3.17}$$

  where log is element-wise. We refer to S_{RKL} as the *reverse KL divergence*.

- **Norm Penalty:** An alternative is to penalize the difference between the model and target expectations using a norm. Examples include:

$$\textbf{L}_\mathbf{2}^\mathbf{2}\textbf{/Squared Err}: \ \mathrm{S}_{L_2^2}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) = -\left\|\tilde{\boldsymbol{\phi}} - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\right\|_2^2 \tag{3.18}$$

$$\textbf{L}_\mathbf{2}\textbf{/Euclidean Dist}: \ \mathrm{S}_{L_2}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) = -\left\|\tilde{\boldsymbol{\phi}} - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\right\|_2 \tag{3.19}$$

$$\textbf{L}_\mathbf{1}\textbf{/Manhattan Dist}: \ \mathrm{S}_{L_1}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) = -\left\|\tilde{\boldsymbol{\phi}} - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\right\|_1 \tag{3.20}$$

- **Range Penalty:** Often target expectations will be estimated in a way that introduces noise. If we know something about the amount of noise in our estimate, we may use score functions that only aim to bring the model expectation within some range of the target expectation. We assume that we have lower $\tilde{\boldsymbol{\phi}}_l$ and upper $\tilde{\boldsymbol{\phi}}_u$ bounds rather than target expectations. Two possible score functions penalize distance from the target range using the L_2^2 or L_1 norm.

$$\textbf{Range } \boldsymbol{L_2^2}\textbf{:} \ \mathrm{S}_{L_2^2 R}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) = -\sum_i \begin{cases} \left(\tilde{\phi}_{li} - \mathrm{E}_{\boldsymbol{\theta}}[\phi_i]\right)^2 & \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] < \tilde{\phi}_{li} \\ \left(\tilde{\phi}_{ui} - \mathrm{E}_{\boldsymbol{\theta}}[\phi_i]\right)^2 & \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] > \tilde{\phi}_{ui} \\ 0 & \text{otherwise} \end{cases} \tag{3.21}$$

The L_1 range score function can be interpreted as a two-sided "hinge" penalty.

$$\textbf{Hinge}: \ \mathrm{S}_{L_1 R}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) = -\sum_i \begin{cases} \tilde{\phi}_{li} - \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] & \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] < \tilde{\phi}_{li} \\ \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] - \tilde{\phi}_{ui} & \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] > \tilde{\phi}_{ui} \\ 0 & \text{otherwise} \end{cases} \tag{3.22}$$

Figure 3.1 displays the different score functions. The expectation of the constraint feature is a distribution over two labels. The x-axis value is the probability of the first label. The target expectation is the uniform distribution. For range constraints, the target range is within ± 0.1. Each score function penalizes distance from the target expectation differently, as we discuss more in the following section. For empirical comparison of different score functions, see Section 8.1.

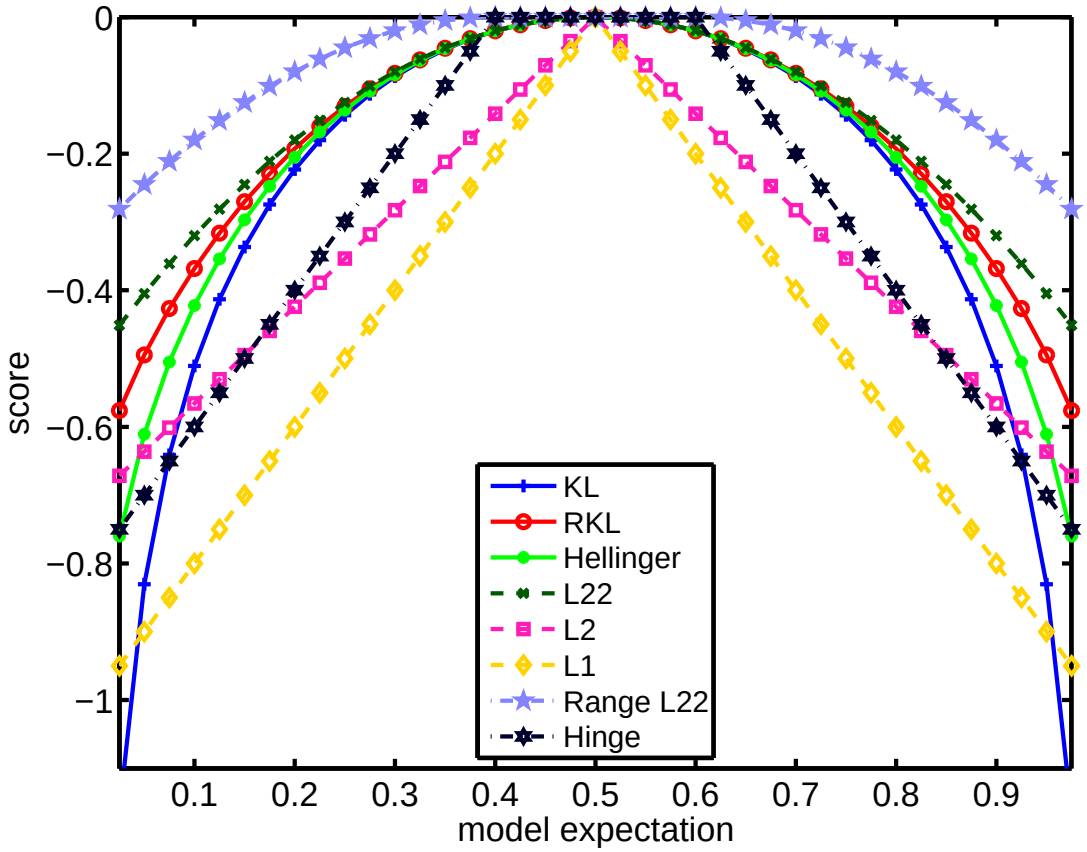


Figure 3.1: Score function values vs. the model expectation of the constraint feature for the first label when the target expectation is the uniform distribution.

3.2.1 GE Parameter Estimation for CRFs

We next discuss estimating CRF parameters using GE criteria. The application of GE to generative models is left to future work. The complete parameter estimation objective function contains three terms: the likelihood of available labeled data, the GE term, and the regularization term. The GE term only applies to unlabeled data.

$$\mathcal{O}(\boldsymbol{\theta}) = \log p(\mathbf{y}_L|\mathbf{x};\boldsymbol{\theta}) + \mathrm{S}(\mathrm{E}_{p(\mathbf{y}_U|\mathbf{x};\boldsymbol{\theta})}[\boldsymbol{\phi}(\mathbf{x},\mathbf{y}_U)]) + \log p(\boldsymbol{\theta}) \tag{3.23}$$

Note that the objective may include several GE terms that use different constraints.

In many applications in this thesis, the data consists of iid instances and no labeled data is available, so the objective function reduces to the GE and prior terms only.

$$\mathcal{O}(\boldsymbol{\theta}) = \mathrm{S}(\mathrm{E}_{p(\mathbf{y}_U|\mathbf{x};\boldsymbol{\theta})}[\boldsymbol{\phi}(\mathbf{x}, \mathbf{y}_U)]) + \log p(\boldsymbol{\theta}) \tag{3.24}$$

In this case, we often simplify notation by dropping U.

We maximize Equation 3.23 or 3.24 with gradient-based numerical optimization. We first show that the gradient of $\mathrm{S}(\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\boldsymbol{\phi}(\mathbf{x}, \mathbf{y})])$ is similar for all of the score functions described in the previous section.

The gradient for an $\boldsymbol{L_2^2}$ score function, $\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{S}_{L_2^2}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}])$, is

$$-\frac{\partial}{\partial \boldsymbol{\theta}} \Big(\tilde{\boldsymbol{\phi}} - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\Big)^{\mathrm{T}} \Big(\tilde{\boldsymbol{\phi}} - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\Big) = 2\Big(\tilde{\boldsymbol{\phi}} - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\Big)^{\mathrm{T}} \Big(\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\Big), \tag{3.25}$$

where $\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]$ will be a matrix of dimensionality $d(\boldsymbol{\phi}) \times d(\mathbf{f})$, where d is the dimensionality of a vector. Similarly, for an $\boldsymbol{L_2}$ score function, $\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{S}_{L_2}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}])$ is

$$-\frac{\partial}{\partial \boldsymbol{\theta}} \Big\| \tilde{\boldsymbol{\phi}} - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}] \Big\|_2 = \frac{(\tilde{\boldsymbol{\phi}} - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}])^{\mathrm{T}}}{\Big\| \tilde{\boldsymbol{\phi}} - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}] \Big\|_2} \Big(\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\Big). \tag{3.26}$$

The gradient of the $\boldsymbol{L_1}$ score function is not defined when $\tilde{\phi}_i = \mathrm{E}_{\boldsymbol{\theta}}[\phi_i]$, so we compute a subgradient. In practice, when using subgradient ascent, we need only choose one value in the set of subgradients. We choose 0.

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{S}_{L_1}(\mathrm{E}_{\boldsymbol{\theta}}[\phi_i]) = \begin{cases} \frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] & \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] < \tilde{\phi}_i \\ -\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] & \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] > \tilde{\phi}_i \\ 0 & \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] = \tilde{\phi}_i, \end{cases} \tag{3.27}$$

The gradient of the **KL divergence** score function, $\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{S}_{KL}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}])$, is

$$\begin{aligned} -\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{D}_{KL}\Big(\tilde{\boldsymbol{\phi}} \mid\mid \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\Big) &= \frac{\partial}{\partial \boldsymbol{\theta}} \Big(\tilde{\boldsymbol{\phi}}^{\mathrm{T}} \log \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}] - \tilde{\boldsymbol{\phi}}^{\mathrm{T}} \log \tilde{\boldsymbol{\phi}}\Big) \\ &= \frac{\partial}{\partial \boldsymbol{\theta}} \tilde{\boldsymbol{\phi}}^{\mathrm{T}} \log \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}] \\ &= \Big(\frac{\tilde{\boldsymbol{\phi}}}{\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]}\Big)^{\mathrm{T}} \Big(\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\Big) \end{aligned} \tag{3.28}$$

where $\frac{\tilde{\boldsymbol{\phi}}}{\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]}$ denotes element-wise division. The gradient of the **reverse KL divergence** score function, $\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{S}_{RKL}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}])$, is

$$\begin{aligned} -\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{D}_{RKL}\Big(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}] \mid\mid \tilde{\boldsymbol{\phi}}\Big) &= \frac{\partial}{\partial \boldsymbol{\theta}} \Big(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]^{\mathrm{T}} \log(\tilde{\boldsymbol{\phi}}) - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]^{\mathrm{T}} \log(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}])\Big) \\ &= \Big(\log \tilde{\boldsymbol{\phi}} - \mathbf{1} - \log(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}])\Big)^{\mathrm{T}} \Big(\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\Big) \end{aligned} \tag{3.29}$$

The gradient of the **Hellinger** distance score function, $\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{S}_{H}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}])$, is

$$\begin{aligned} -\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{D}_{H}\Big(\tilde{\boldsymbol{\phi}} \mid\mid \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\Big) &= -\frac{\partial}{\partial \boldsymbol{\theta}} 2 \Big\| \sqrt{\tilde{\boldsymbol{\phi}}} - \sqrt{\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]} \Big\|_2^2 \\ &= 2 \left(\frac{\sqrt{\tilde{\boldsymbol{\phi}}} - \sqrt{\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]}}{\sqrt{\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]}} \right)^{\mathrm{T}} \Big(\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\Big) \end{aligned} \tag{3.30}$$

The gradients (or subgradients) of **range** penalties are similar to their non-range equivalents except that the gradient is 0 in the interval $[\tilde{\phi}_{li}, \tilde{\phi}_{ui}]$.

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{S}_{L_2^2 R}(\mathrm{E}_{\boldsymbol{\theta}}[\phi_i]) = \begin{cases} 2\Big(\tilde{\phi}_{li} - \mathrm{E}_{\boldsymbol{\theta}}[\phi_i]\Big)\Big(\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\boldsymbol{\theta}}[\phi_i]\Big) & \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] < \tilde{\phi}_{li} \\ 2\Big(\tilde{\phi}_{ui} - \mathrm{E}_{\boldsymbol{\theta}}[\phi_i]\Big)\Big(\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\boldsymbol{\theta}}[\phi_i]\Big) & \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] > \tilde{\phi}_{ui} \\ 0 & \text{otherwise} \end{cases} \tag{3.31}$$

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{S}_{L_1 R}(\mathrm{E}_{\boldsymbol{\theta}}[\phi_i]) = \begin{cases} \Big(\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\boldsymbol{\theta}}[\phi_i]\Big) & \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] < \tilde{\phi}_{li} \\ -\Big(\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\boldsymbol{\theta}}[\phi_i]\Big) & \mathrm{E}_{\boldsymbol{\theta}}[\phi_i] > \tilde{\phi}_{ui} \\ 0 & \text{otherwise} \end{cases} \tag{3.32}$$

The $\boldsymbol{L_2^2}$ **range** score function is differentiable everywhere, whereas the $\boldsymbol{L_1}$ **range** / **hinge** score function is not differentiable at the boundaries of the zero-penalty region. As with the $\boldsymbol{L_1}$ score function, in practice at these boundaries we choose 0 from the set of possible subgradients.

The gradient of each of the above score functions is the product of a penalty-specific row vector and the matrix $\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]$. We next compute this matrix.

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\boldsymbol{\phi}(\mathbf{x},\mathbf{y})] &= \sum_{\mathbf{y}} \boldsymbol{\phi}(\mathbf{x},\mathbf{y}) \frac{\partial}{\partial \boldsymbol{\theta}} p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) \\ &= \sum_{\mathbf{y}} \boldsymbol{\phi}(\mathbf{x},\mathbf{y}) \frac{\partial}{\partial \boldsymbol{\theta}} \Big(\frac{1}{Z(\mathbf{x};\boldsymbol{\theta})} \exp\left(\boldsymbol{\theta}^{\mathrm{T}} \mathbf{f}(\mathbf{x},\mathbf{y})\right) \Big) \end{aligned} \tag{3.33}$$

Using the *product rule*, $\frac{\partial}{\partial x}(uv) = u(\frac{\partial}{\partial x}v) + v(\frac{\partial}{\partial x}u)$, the gradient of the first term is

$$\begin{aligned} \frac{1}{Z(\mathbf{x};\boldsymbol{\theta})} \Big(\frac{\partial}{\partial \boldsymbol{\theta}} \exp\left(\boldsymbol{\theta}^{\mathrm{T}} \mathbf{f}(\mathbf{x},\mathbf{y})\right) \Big) &= \frac{1}{Z(\mathbf{x};\boldsymbol{\theta})} \exp\left(\boldsymbol{\theta}^{\mathrm{T}} \mathbf{f}(\mathbf{x},\mathbf{y})\right) \mathbf{f}(\mathbf{x},\mathbf{y})^{\mathrm{T}} \\ &= p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) \mathbf{f}(\mathbf{x},\mathbf{y})^{\mathrm{T}}. \end{aligned} \tag{3.34}$$

The gradient of the second term is

$$
\begin{aligned}
&\exp\left(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{f}(\mathbf{x},\mathbf{y})\right)\left(\frac{\partial}{\partial\boldsymbol{\theta}}\frac{1}{Z(\mathbf{x};\boldsymbol{\theta})}\right)\\
&=-\exp\left(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{f}(\mathbf{x},\mathbf{y})\right)\frac{1}{Z(\mathbf{x};\boldsymbol{\theta})^2}\sum_{\mathbf{y}'}\exp\left(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{f}(\mathbf{x},\mathbf{y}')\right)\mathbf{f}(\mathbf{x},\mathbf{y}')^{\mathrm{T}}\\
&=-p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})\sum_{\mathbf{y}'}p(\mathbf{y}'|\mathbf{x};\boldsymbol{\theta})\mathbf{f}(\mathbf{x},\mathbf{y}')^{\mathrm{T}}. \qquad (3.35)
\end{aligned}
$$

Therefore, $\frac{\partial}{\partial\boldsymbol{\theta}}\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\boldsymbol{\phi}(\mathbf{x},\mathbf{y})]$ is

$$
\begin{aligned}
\frac{\partial}{\partial\boldsymbol{\theta}}\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}] =&\sum_{\mathbf{y}}p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x},\mathbf{y})\mathbf{f}(\mathbf{x},\mathbf{y})^{\mathrm{T}}\\
&-\Big(\sum_{\mathbf{y}}p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x},\mathbf{y})\Big)\Big(\sum_{\mathbf{y}}p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})\mathbf{f}(\mathbf{x},\mathbf{y})^{\mathrm{T}}\Big)\\
=&\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\boldsymbol{\phi}(\mathbf{x},\mathbf{y})\mathbf{f}(\mathbf{x},\mathbf{y})^{\mathrm{T}}]-\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\boldsymbol{\phi}(\mathbf{x},\mathbf{y})]\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\mathbf{f}(\mathbf{x},\mathbf{y})^{\mathrm{T}}]\\
=&\mathrm{COV}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}\Big(\boldsymbol{\phi}(\mathbf{x},\mathbf{y}),\mathbf{f}(\mathbf{x},\mathbf{y})\Big). \qquad (3.36)
\end{aligned}
$$

Equation 3.36 shows that the gradient of the model expectation of the constraint feature vector is the *covariance matrix* between constraint and model features.

The gradient provides intuition for how estimating parameters with GE works. If the score function is not already maximized, GE updates parameters for model features according to their covariance with constraint features. The score functions differ in the first term of the gradient, which we denote by $\mathbf{u}$. For example, for L_2^2, $\mathbf{u}=2(\tilde{\boldsymbol{\phi}}-\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}])$. The complete gradient for any of the score functions is

$$
\frac{\partial}{\partial\boldsymbol{\theta}}\mathrm{S}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}])=\mathbf{u}^{\mathrm{T}}\Big(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}\mathbf{f}^{\mathrm{T}}]-\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\mathrm{E}_{\boldsymbol{\theta}}[\mathbf{f}^{\mathrm{T}}]\Big). \qquad (3.37)
$$

Different score functions emphasize closing the gap between model expectations and target expectations in different ways. For example, KL divergence, in the direction used in S_{KL}, is said to be *inclusive*, as it forces $\mathrm{E}_{\boldsymbol{\theta}}[\phi_i]>0$ when $\tilde{\phi}_i>0$. This is apparent in Equation 3.28, as u_i is large when $\tilde{\phi}_i>0$ and $\mathrm{E}_{\boldsymbol{\theta}}[\phi_i]$ is close to 0. The

L_2^2 score function emphasizes constraint features i with $\mathrm{E}_{\boldsymbol{\theta}}[\phi_i] \ll \tilde{\phi}_i$ or $\mathrm{E}_{\boldsymbol{\theta}}[\phi_i] \gg \tilde{\phi}_i$ with $2(\tilde{\phi}_i - \mathrm{E}_{\boldsymbol{\theta}}[\phi_i])$. The L_1 score function treats all constraint features the same if $\mathrm{E}_{\boldsymbol{\theta}}[\phi_i] \neq \tilde{\phi}_i$. For empirical comparison of different score functions, see Section 8.1.

While Equation 3.37 provides intuition, we do not need to explicitly compute and store the covariance matrix. Note that we can rearrange this equation by moving $\mathbf{u}$ inside the expectations.

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{S}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) = \mathrm{E}_{\boldsymbol{\theta}}[\mathbf{u}^{\mathrm{T}} \boldsymbol{\phi} \mathbf{f}^{\mathrm{T}}] - \mathrm{E}_{\boldsymbol{\theta}}[\mathbf{u}^{\mathrm{T}} \boldsymbol{\phi}] \mathrm{E}_{\boldsymbol{\theta}}[\mathbf{f}^{\mathrm{T}}] \tag{3.38}$$

Note that $\mathbf{u}^{\mathrm{T}} \boldsymbol{\phi}$ is a scalar. If we define the *composite constraint feature* as $\phi'(\mathbf{x}, \mathbf{y}) \equiv \mathbf{u}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}, \mathbf{y})$, then the gradient may be rewritten as the covariance between the composite constraint feature and the model features.

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{S}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) = \mathrm{E}_{\boldsymbol{\theta}}[\phi' \mathbf{f}^{\mathrm{T}}] - \mathrm{E}_{\boldsymbol{\theta}}[\phi'] \mathrm{E}_{\boldsymbol{\theta}}[\mathbf{f}^{\mathrm{T}}] \tag{3.39}$$

The immediate benefit is that we never need to store the full covariance matrix.

To compute Equation 3.39 we must compute three expectations. The difficulty of computing these expectations depends on both the model and the way in which the constraint features decompose into local constraint features. In general, if the constraint and model features decompose in the same way, then standard inference techniques can be used to compute $\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\phi']$ and $\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\mathbf{f}]$. Computing the first term of Equation 3.39 can be more challenging.

$$\begin{aligned} \mathrm{E}_{\boldsymbol{\theta}}[\phi' \mathbf{f}^{\mathrm{T}}] &= \mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\phi'(\mathbf{x}, \mathbf{y}) \mathbf{f}(\mathbf{x}, \mathbf{y})^{\mathrm{T}}] \\ &= \sum_{a \in \mathcal{F}} \sum_{a' \in \mathcal{F}} p(\mathbf{y}_a, \mathbf{y}'_a | \mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, \mathbf{y}_a) \mathbf{f}(\mathbf{x}, \mathbf{y}'_a)^{\mathrm{T}} \end{aligned} \tag{3.40}$$

Equation 3.40 shows that, naively, we need to compute marginal distributions over all variables that participate in a and a'. As we saw in Section 2.1.3.3, standard parameter estimation only require marginal distributions over a.

We show that it is actually possible to compute Equation 3.40 efficiently for logistic regression models (Chapter 4), linear chain CRFs, and tree-structured CRFs in general (Chapter 6). We show that it is possible to compute Equation 3.40 in polynomial time for CRFs that model distributions over trees (Chapter 5). Finally, in some cases approximate inference may be necessary. Consequently, we also consider using MCMC methods to approximate Equation 3.40 (Chapter 7).

3.2.2 GE Gradient with iid Instances

When the data consists of N iid instances the model probability simplifies to

$$p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) = \prod_{i=1}^{N} p(\mathbf{y}^i|\mathbf{x}^i;\boldsymbol{\theta}), \tag{3.41}$$

where i indexes the instances. Assuming that the constraint features also decompose over instances, the GE gradient may be simplified.

$$\begin{aligned}
\frac{\partial}{\partial\boldsymbol{\theta}}\mathrm{S}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) &= \mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\phi'(\mathbf{x},\mathbf{y})\mathbf{f}(\mathbf{x},\mathbf{y})^{\mathrm{T}}] - \mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\phi'(\mathbf{x},\mathbf{y})]\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\mathbf{f}(\mathbf{x},\mathbf{y})^{\mathrm{T}}] \\
&= \sum_{i=1}^{N}\sum_{\mathbf{y}^i}\sum_{j=1}^{N}\sum_{\mathbf{y}^j} p(\mathbf{y}^i|\mathbf{x}^i;\boldsymbol{\theta})p(\mathbf{y}^j|\mathbf{x}^j;\boldsymbol{\theta})[\phi(\mathbf{x}^i,\mathbf{y}^i)\mathbf{f}(\mathbf{x}^j,\mathbf{y}^j)^{\mathrm{T}}] \\
&\quad - \sum_{i=1}^{N}\sum_{\mathbf{y}^i} p(\mathbf{y}^i|\mathbf{x}^i;\boldsymbol{\theta})[\phi(\mathbf{x}^i,\mathbf{y}^i)] \sum_{j=1}^{N}\sum_{\mathbf{y}^j} p(\mathbf{y}^j|\mathbf{x}^j;\boldsymbol{\theta})[\mathbf{f}(\mathbf{x}^j,\mathbf{y}^j)^{\mathrm{T}}] \\
&= \sum_{i=1}^{N}\sum_{\mathbf{y}^i} p(\mathbf{y}^i|\mathbf{x}^i;\boldsymbol{\theta})[\phi(\mathbf{x}^i,\mathbf{y}^i)\mathbf{f}(\mathbf{x}^i,\mathbf{y}^i)^{\mathrm{T}}] \\
&\quad - \sum_{\mathbf{y}^i} p(\mathbf{y}^i|\mathbf{x}^i;\boldsymbol{\theta})[\phi(\mathbf{x}^i,\mathbf{y}^i)] \sum_{\mathbf{y}^i} p(\mathbf{y}^i|\mathbf{x}^i;\boldsymbol{\theta})[\mathbf{f}(\mathbf{x}^i,\mathbf{y}^i)^{\mathrm{T}}] \\
&= \sum_{i=1}^{N} \mathrm{COV}_{p(\mathbf{y}^i|\mathbf{x}^i;\boldsymbol{\theta})}\Big(\phi(\mathbf{x}^i,\mathbf{y}^i), \mathbf{f}(\mathbf{x}^i,\mathbf{y}^i)\Big)
\end{aligned} \tag{3.42}$$

Equation 3.42 shows that with iid instances the covariance over the entire data set is equivalent to the sum of per-instance covariances. Intuitively, the "diagonal" terms among pairs of different, independent instances in the covariance cancel out.

Algorithm 1 Computing the GE value and gradient

Input: unlabeled data $\mathbf{x}$, constraint features $\boldsymbol{\phi}$, score function S
Output: value v, gradient **grad**
// compute constraint feature expectations
$\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}] = \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x},\mathbf{y})$
// compute GE value
$v = \mathrm{S}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) + \log p(\boldsymbol{\theta})$
// compute GE gradient
$\mathbf{grad} = \mathrm{E}_{\boldsymbol{\theta}}[\phi'\mathbf{f}^{\mathrm{T}}] - \mathrm{E}_{\boldsymbol{\theta}}[\phi']\mathrm{E}_{\boldsymbol{\theta}}[\mathbf{f}^{\mathrm{T}}] + \frac{\partial}{\partial\boldsymbol{\theta}} \log p(\boldsymbol{\theta})$

3.2.3 Generic GE Value and Gradient Computation Algorithm

We next describe a general algorithm for estimating parameters with GE. Parameter estimation with GE is driven by a numerical optimizer. In order to decide how to adjust the parameters, the optimizer asks the GE implementation for the value and gradient of the GE objective function with the current parameters. Algorithm 1 shows that the method for computing the value and gradient can be described as three distinct steps. Note that this algorithm assumes no labeled data.

First, the expectations of the constraint features are computed. If it is possible to perform exact inference in $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$, and constraint features factor in the same way as model features, this step can be performed efficiently. Second, the value v is computed by summing the score function, computed using $\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]$, and the parameter regularization term. Finally, the gradient **grad** is computed by computing the covariance between the composite constraint feature and the model features. Note that computing the composite constraint feature requires the constraint feature expectations $\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]$ computed in the first step.

3.2.4 GE Optimization Notes

In standard maximum entropy estimation, reviewed in Section 2.1.3.4, we know that there is a labeling of the data that satisfies the equality expectation constraints: the labeling that was used to estimate the target values. Note that there may not be a solution that exactly matches the target expectations in the applications in this thesis, since the target expectations are typically estimated in a way that introduces

noise. However, GE training is still feasible in these scenarios as it encourages but does not require the model to match the target expectations.

As with most objective functions that include latent variables, Equation 3.23 is not concave in the parameters $\boldsymbol{\theta}$. It is well known that $\log p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$ is concave in $\boldsymbol{\theta}$, but $\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]$ is not concave in $\boldsymbol{\theta}$. It is still possible to use standard convex optimization techniques to maximize Equation 3.23, though of course we are not guaranteed to find the global optimum. We use two optimization algorithms in this thesis: *Subgradient Ascent*, for score functions with points of non-differentiability, and a limited memory quasi-Newton optimizer *L-BFGS* [68]. When using L-BFGS, we use the standard, though not often published, trick of resetting the history and re-starting optimization upon convergence. In our experience, this improves accuracy with non-convex functions. In general, we do not find it necessary to try many different initializations of the parameters, though higher accuracy may be obtained by doing so. Unless noted, experiments in this thesis begin GE training with parameters initialized to zero.

3.2.5 Temperature

In general, there may be multiple ways to match the target expectations. For example, suppose we have four documents, and we would like the model to expect that 75% are label 0. This constraint could be perfectly satisfied in a number of different ways. For example, one solution is to label any three documents as 0, and the other as 1. Alternatively, we could label any two documents with label 1, and label the other documents as 50% label 0, 50% label 1. In some cases the first solution may be preferable to the second.

To address this, we may modify model probabilities with a *temperature* T.

$$p_T(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) \propto \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x},\mathbf{y})/T) \tag{3.43}$$

If $T < 1$, the log score scores for each $\mathbf{y}$ are multiplied by a constant > 1. These log scores are then exponentiated. Consequently, using $T < 1$ makes the model probability distribution more peaked. As $T \to 0$, p_T approaches a distribution where all mass is placed on the maximum probability output.

Using a temperature $T < 1$ in GE training discourages solutions in which target expectations are matched with very flat distributions. Empirically, using a temperature has provided accuracy improvements [71]. We provide additional discussion and experiments using a temperature in Chapter 7.

3.3 Related Methods

In this section we discuss other frameworks that are related to GE.

3.3.1 Posterior Regularization

As we show in this thesis, in many cases of interest the GE gradient, specifically Equation 3.39, can be computed efficiently. However, for some models computing the GE gradient requires inference algorithms that have higher time complexity than standard inference algorithms. In this section we discuss an approximation to the GE objective function that avoids the covariance computation.

We define an *auxiliary distribution*, denoted by q, over latent output variables $q(\mathbf{y}_U|\mathbf{x})$. The following objective function is a function of both the model parameters $\boldsymbol{\theta}$ and the auxiliary distribution q.

$$\mathcal{O}(\boldsymbol{\theta}, q) = \mathcal{L}(\boldsymbol{\theta}) - \mathrm{D}_{KL}\big(q(\mathbf{y}_U|\mathbf{x})||p(\mathbf{y}_U|\mathbf{x};\boldsymbol{\theta})\big) + \mathrm{S}\big(\mathrm{E}_{q(\mathbf{y}_U|\mathbf{x})}[\boldsymbol{\phi}(\mathbf{x}, \mathbf{y}_U)]\big) \tag{3.44}$$

In this objective, the score function S evaluates expectations of the constraint features under the auxiliary model, and the model and auxiliary distributions over latent variables are encouraged to be "close" as measured by KL divergence. This objective is an approximation to Equation 3.23.

To optimize this objective function, we perform *block coordinate ascent*, alternating between optimizing $\mathcal{O}(\boldsymbol{\theta}, q)$ with respect to q and with respect to $\boldsymbol{\theta}$. These optimization steps can be interpreted as projections.

Information-Projection: Maximizing Equation 3.44 with respect to q gives

$$\begin{aligned}\max_q \ \mathcal{O}(\boldsymbol{\theta}, q) &= \max_q \ \mathcal{L}(\boldsymbol{\theta}) - \mathrm{D}_{KL}\big(q(\mathbf{y}_U|\mathbf{x})||p(\mathbf{y}_U|\mathbf{x};\boldsymbol{\theta})\big) + \mathrm{S}\big(\mathrm{E}_{q(\mathbf{y}_U|\mathbf{x})}[\boldsymbol{\phi}(\mathbf{x},\mathbf{y}_U)]\big) \\ &= \min_q \ \mathrm{D}_{KL}\big(q(\mathbf{y}_U|\mathbf{x})||p(\mathbf{y}_U|\mathbf{x};\boldsymbol{\theta})\big) - \mathrm{S}\big(\mathrm{E}_{q(\mathbf{y}_U|\mathbf{x})}[\boldsymbol{\phi}(\mathbf{x},\mathbf{y}_U)]\big) \qquad (3.45)\end{aligned}$$

If S is convex in the constraint feature expectations, then this is a *Generalized Maximum Entropy* problem [30]. The problem is a generalization of maximum entropy in two ways. First, we minimize the KL divergence from a base distribution, rather than maximize entropy. Note that this is equivalent to maximizing entropy if the base distribution is uniform. Second, rather than satisfy equality constraints, we instead minimize a penalty function of model expectations (the negative of the score function). In the terminology of information geometry, this problem is also called an *information projection*.

As in maximum entropy estimation, the optimization problem in Equation 3.45 is hard to solve in the primal form, but it is easy to solve the dual problem. The dual problem is similar to the dual of maximum entropy, but here the base distribution appears in the log-partition function, and there is regularization on model parameters. For example, with an L_2^2 score function, the dual is

$$\operatorname*{argmax}_{\boldsymbol{\lambda}} \quad \boldsymbol{\lambda}^{\mathrm{T}}\tilde{\boldsymbol{\phi}} - \log \sum_{\mathbf{y}_U} \exp\left(\boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x},\mathbf{y}_U) + \log p(\mathbf{y}_U|\mathbf{x};\boldsymbol{\theta})\right) - \left\|\boldsymbol{\lambda}\right\|_2^2.$$

Moment-Projection: Maximizing Equation 3.44 with respect to $\boldsymbol{\theta}$ gives

$$\begin{aligned}
\max_{\boldsymbol{\theta}} \ \mathcal{O}(\boldsymbol{\theta}, q) &= \max_{\boldsymbol{\theta}} \ \mathcal{L}(\boldsymbol{\theta}) - \mathrm{D}_{KL}\big(q(\mathbf{y}_U|\mathbf{x})||p(\mathbf{y}_U|\mathbf{x};\boldsymbol{\theta})\big) + \mathrm{S}\big(\mathrm{E}_{q(\mathbf{y}_U|\mathbf{x})}[\boldsymbol{\phi}(\mathbf{x},\mathbf{y}_U)]\big) \\
&= \max_{\boldsymbol{\theta}} \ \mathcal{L}(\boldsymbol{\theta}) - \sum_{\mathbf{y}_U} q(\mathbf{y}_U|\mathbf{x}) \log q(\mathbf{y}_U\mathbf{x}) + \sum_{\mathbf{y}_U} q(\mathbf{y}_U|\mathbf{x}) \log p(\mathbf{y}_U|\mathbf{x};\boldsymbol{\theta}) \\
&= \max_{\boldsymbol{\theta}} \ \mathcal{L}(\boldsymbol{\theta}) + \sum_{\mathbf{y}_U} q(\mathbf{y}_U|\mathbf{x}) \log p(\mathbf{y}_U|\mathbf{x};\boldsymbol{\theta}) \qquad (3.46)
\end{aligned}$$

In this problem, called the *moment projection*, the score function is a constant. Consequently, this is a maximum likelihood problem with the hidden variables $\mathbf{y}_U$ "filled in" by the auxiliary distribution q.

Intuitively, this method is similar to GE, but instead of trying to directly enforce the expectation constraints by adjusting $\boldsymbol{\theta}$, we introduce a distribution q to match the expectation constraints, and encourage q and p to be similar. The optimization alternates between trying to find a q that is close to p but also respects the constraints, and trying to p close to q that also respects $\mathcal{L}(\boldsymbol{\theta})$.

This method, called *Alternating Projections (AP)* in [5], is equivalent to the penalty version of *Posterior Regularization (PR)*. PR was first described as an EM algorithm that constrains the posterior distribution in the E-step [39]. PR was subsequently generalized to use soft constraints with a penalty formulation similar to GE and AP, and to address training of discriminative models [36].

Concretely, the E- and M-steps in PR are equivalent to the I- and M-projections in AP. Ganchev et al. formulate the penalty objective slightly differently, using the norm of slack variables [36], but this is equivalent to the generalized maximum entropy formulation used by AP.

There have been numerous applications of PR, including incorporating structural constraints into word alignment models [39], agreement constraints in a multi-view setting [35], "noisy labeled data" into dependency grammar induction [34], and posterior sparsity into part-of-speech induction [40]. In contrast, in this thesis we are

primarily concerned with input feature label distribution constraints. We compare GE and PR empirically in Section 8.3.

3.3.2 Generalized Maximum Entropy / Maximum Likelihood

The most straightforward method for learning with expectation constraints is generalized maximum entropy [30], previously discussed in Sections 2.1.3.4 and 3.3.1. With a uniform base distribution and no labeled data, the primal problem is

$$\max_{p} \ \mathrm{H}(p(\mathbf{y}_U|\mathbf{x})) + \mathrm{S}(\mathrm{E}_{p(\mathbf{y}_U|\mathbf{x})}[\boldsymbol{\phi}(\mathbf{x}, \mathbf{y}_U)]).$$

It is convenient to solve this problem in its dual form. The precise dual objective depends on S, but in all cases the parametrization of p is determined by the constraint features $\boldsymbol{\phi}$. Specifically, the model and constraint features are identical, $\boldsymbol{\phi} = \mathbf{f}$.

In contrast, GE, PR, and related methods allow $\boldsymbol{\phi} \neq \mathbf{f}$. We suggest two settings in which this flexibility can be leveraged. First, we often know about the existence of additional unconstrained features. For example, in text classification, we may only have input feature label distribution constraints for a few words, but given unlabeled data we know about the existence of many other words that may be relevant to the classification task. Allowing $\boldsymbol{\phi} \neq \mathbf{f}$ enables learning parameters for these features by including them in $\mathbf{f}$. This thesis focuses on training feature-rich models with a small number of expectation constraints. Empirical comparisons between GE and generalized maximum entropy in this setting are provided in Sections 7.3 and 8.3.

Alternatively, model features could be less expressive than constraint features, which enables leveraging rich prior knowledge while retaining a tractable model. For example, we could train a logistic regression model for a sequence labeling task using additional expectation constraints that encourage correct transition marginals. Note, however, that in general the model will not be capable of matching targets for more expressive constraints exactly. Consider the following example. Suppose there are two

binary variables: Y_0 and Y_1, and we have expectation constraints that specify that the transition distribution should be $p(Y_0=0, Y_1=0)=0.6$ and $p(Y_0=1, Y_1=1) = 0.4$. Maximum entropy estimation would create a dependency between Y_0 and Y_1, and the targets could be matched exactly. However, if we use GE with $p(y_0, y_1) = p(y_0)p(y_1)$, then the target transition distribution cannot be matched exactly. Instead, the global maximum of a GE objective with a KL divergence penalty and no regularization would result in marginals of $p(Y_0 = 0) = p(Y_1 = 0) = 0.6$, yielding a transition distribution of $p(Y_0 = 0, Y_1 = 0) = 0.36$, $p(Y_0 = 0, Y_1 = 0) = 0.16$, $p(Y_0 = 1, Y_1 = 0) = 0.24$, $p(Y_0 = 0, Y_1 = 1) = 0.24$. Note that the marginals $p(y_0)$ and $p(y_1)$ are correct, but that 48% of transitions the model expects are intended to be disallowed by the constraints.

It may be possible to develop alternatives to GE that are based on maximum entropy. For example, if labeled and unlabeled data are available, we could specify a mixture of constraints from labeled data and constraints from prior knowledge, and apply them to the unlabeled data. However, as above this method is not capable of learning parameters for unconstrained features that appear in unlabeled data. To address this, it may be possible to develop a maximum entropy method that uses an additional assumption about the parametrization of the model. Though the additional assumption conflicts with the principle of choosing the simplest model that is capable of satisfying the constraints, it would allow $\mathbf{f} \neq \boldsymbol{\phi}$, enabling the inclusion of extra model features. Note, however, that maximizing entropy may discourage the use of the extra model features. We leave exploration of these directions to future work.

Though generalized maximum entropy and GE provide different solutions in the settings explored in this thesis, we also describe a case in which they are equivalent. Suppose that there is no parameter regularization in either objective function, the score functions are maximized when $\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\boldsymbol{\phi}(\mathbf{x}, \mathbf{y})] = \tilde{\boldsymbol{\phi}}$, and GE uses a model with only the constraint features, $\mathbf{f} = \boldsymbol{\phi}$. In this case, the maximum entropy solution $\boldsymbol{\theta}_{me}$ is a global maximum of the GE objective function. As discussed in Section 2.1.3.4,

maximum entropy is equivalent to maximum likelihood for CRFs. Therefore, the maximum likelihood solution is a global maximum of the GE objective when target expectations $\tilde{\boldsymbol{\phi}}$ are estimated from labeled data and the above conditions are satisfied.

3.3.3 Learning from Measurements

Liang et al. take a Bayesian approach to learning with *measurements* [66], which are equivalent to target expectations $\tilde{\boldsymbol{\phi}}$ in this thesis. Note that unlike the approaches described in Section 2.2.3, this approach explicitly models the observation of prior knowledge about feature functions. Liang et al. model the probability of observed and latent output variables, $\mathbf{y}_L$ and $\mathbf{y}_U$, and observed target expectations, $\tilde{\boldsymbol{\phi}}$, as

$$p(\mathbf{y}_L, \mathbf{y}_U, \tilde{\boldsymbol{\phi}}|\mathbf{x}; \boldsymbol{\theta}) = p(\mathbf{y}_L|\mathbf{x}; \boldsymbol{\theta}) p(\mathbf{y}_U|\mathbf{x}; \boldsymbol{\theta}) p_N\big(\tilde{\boldsymbol{\phi}}|\boldsymbol{\phi}(\mathbf{x}, \mathbf{y}_U)\big). \tag{3.47}$$

It is assumed that $\mathbf{y}_L$ and $\mathbf{y}_U$ are conditionally independent given the input variables $\mathbf{x}$. Target expectations $\tilde{\boldsymbol{\phi}}$ are modeled as noisy observations of the true values according to some noise model P_N. The marginal likelihood of observed variables under this model is

$$\mathcal{L}_{mm}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \log \sum_{\mathbf{y}_U} p(\mathbf{y}_U|\mathbf{x}; \boldsymbol{\theta}) p_N\big(\tilde{\boldsymbol{\phi}}|\boldsymbol{\phi}(\mathbf{x}, \mathbf{y}_U)\big) \tag{3.48}$$

Computing the second term is intractable, as it does not decompose into expectations over smaller subsets of variables. However, Ganchev et al. [36] show that if we make the approximation

$$\sum_{\mathbf{y}_U} p(\mathbf{y}_U|\mathbf{x}; \boldsymbol{\theta}) p_N(\tilde{\boldsymbol{\phi}}|\boldsymbol{\phi}(\mathbf{x}, \mathbf{y}_U)) = \mathrm{E}_{\boldsymbol{\theta}}[p_N(\tilde{\boldsymbol{\phi}}|\boldsymbol{\phi})] \approx p_N\big(\tilde{\boldsymbol{\phi}}|\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\big), \tag{3.49}$$

then, with an appropriate noise model, we recover GE. For example, if we assume $\tilde{\boldsymbol{\phi}} \sim \mathcal{N}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}], 0.5I)$, then, ignoring constants with respect to $\boldsymbol{\theta}$, we recover GE with

an L_2^2 score function.

$$\begin{aligned}\mathcal{L}_{mm}(\boldsymbol{\theta}) &\approx \mathcal{L}(\theta) + p_N\big(\tilde{\boldsymbol{\phi}}|\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\big) \\ &= \mathcal{L}(\theta) - \Big(\tilde{\boldsymbol{\phi}} - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\Big)^{\mathrm{T}}\Big(\tilde{\boldsymbol{\phi}} - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\Big) \\ &= \mathcal{L}(\theta) + \mathrm{S}_{L_2^2}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) \end{aligned} \tag{3.50}$$

Liang et al. [66] make a different approximation to Equation 3.48 that combines variational inference and Jensen's inequality. The resulting objective function is equivalent to PR [36].

3.3.4 Constraint-Driven Learning

Constraint-driven learning (CODL) [15] incorporates instance-specific constraints into inference using hand set penalties. Incorporating the penalties may break standard inference algorithms such as Viterbi, so approximate inference, such as beam search, is used instead. Unlabeled data is incorporated into learning with an EM-like algorithm. The modified E-step consists of using the penalty-augmented inference procedure to generate the n-best assignments to the latent output variables. The modified M-step uses the n-best lists to re-estimate model parameters. CODL can be viewed as an approximation to PR because it uses an n-best list rather than the full posterior. Unlike other related methods, CODL requires the practitioner to specify difficult-to-interpret penalties, rather than expectation constraints.

3.3.5 Constraint-Driven SampleRank

SampleRank is a supervised parameter estimation method that performs parameter updates during MCMC inference [124]. Specifically, each pair of consecutive samples is compared, and a parameter update is made if the ranking of the model scores of the samples disagrees with the ranking implied by the labeled data. For further details consult [124].

Constraint-Driven SampleRank (CDSR) extends SampleRank to leverage constraints and unlabeled data [110]. In this algorithm the two samples are compared with a function that penalizes constraint violation with user-specified penalties $\boldsymbol{\lambda}$. In particular, each sample is evaluated using

$$\mathcal{F}_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{y}) = \lambda^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}), \tag{3.51}$$

where Singh et al. [110] define ϕ_i to return -1 if the constraint is violated, 1 if it is satisfied and 0 if it does not apply. Because CDSR updates parameters during sampling, this method is very efficient. However, unlike GE and related methods, it is not clear how CDSR can enforce that a constraint should hold in expectation, and consequently the method is best suited to hard, per-instance constraints.

3.3.6 Other Related Methods

GE is related to the *method of moments* [38], but differs in that it uses target expectations from prior knowledge rather than sample moments, maximizes a score function instead of requiring equality, and allows more or fewer moments than parameters. More recent work in statistics that provides some (but not all) of these flexibilities includes work that suggests using prior knowledge to set moments and allows the use of more (but not fewer) moments than parameters [37], and work that allows the use of auxiliary moment constraints to improve the efficiency of fully observed parameter estimation [47].

Quadrianto, et al. present a method for learning a conditional model for a test set with known label proportions given only n ($n \geq |\mathcal{Y}|$) sets of observations, each with known label proportions [91]. The method estimates feature expectations by solving a system of equations that relates the known empirical expectations in each set, the known label proportions, and the true expectations on the test set. However, this

method requires knowing the label proportions on the test set, and does not address structured problems or noisy label proportions.

In later work, Quadrianto et al. propose a transductive learning algorithm that encourages the model to have similar training and testing data feature expectations [90]. The method uses a GE-like term in the objective function that penalizes large divergence between the expectations. The objective is optimized in an online fashion by comparing the expectations of pairs of points from the training and test set. This suggests it may be possible to devise an online algorithm for GE, though this is left to future work.

Coupled semi-supervised learning [14] is a method that enforces coupling constraints between different learning tasks in a multi-task setting. The proposed algorithms are conceptually similar to PR or CODL: they involve predicting values for latent variables and adjusting the predictions to take into account the coupling constraints (E-step), and using the adjusted predictions to update the model (M-step).

CHAPTER 4

GE FOR LIGHTLY SUPERVISED TEXT CLASSIFICATION

In this chapter, we use GE to develop a lightly supervised text classification method that leverages known relationships between input features (words) and labels. Extensive experiments demonstrate that this approach typically outperforms other methods for learning with such prior knowledge, and that, given limited annotation time, having an annotator "label" input features rather than complete documents typically yields a more accurate classifier.

4.1 GE for Logistic Regression

The goal of classification is to predict the value of an output variable y^i, often referred to as a *label* or a *class*, for an input $\mathbf{x}^i$. If we assume that all output variables are independent, $\forall_{i,j:i\neq j}\ y_i \perp\!\!\!\perp y_j$, then the conditional probability of a set of n instances can be simplified to

$$p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) = \prod_{i=1}^{N} p(y^i|\mathbf{x}^i;\boldsymbol{\theta}). \tag{4.1}$$

As discussed in Section 2.1.4, with independent output variables a CRF is equivalent to a *multinomial logistic regression* model or a *maximum entropy classifier*.

$$p(y^i|\mathbf{x}^i;\boldsymbol{\theta}) = \frac{1}{Z(\mathbf{x}^i;\boldsymbol{\theta})} \exp\left(\boldsymbol{\theta}\cdot\mathbf{f}(\mathbf{x}^i, y^i)\right) \tag{4.2}$$

Inference in a maximum entropy classifier is straightforward. It follows from Equation 4.1 that expectations of model features decompose into the sum of instance-

specific expectations. We can compute instance-specific partition functions $Z(\mathbf{x}^i; \boldsymbol{\theta})$ or the maximum probability label for $\mathbf{x}^i$ in $O(|\mathcal{Y}|)$ time, where $|\mathcal{Y}|$ is the number of possible labels, by computing the sum or the max of the scores of each label.

Computing the gradient of a GE term is also straightforward. As discussed in Section 3.2.2, with iid instances the covariance decomposes into the sum of instance-specific covariances. The first term of Equation 3.39 is

$$\mathrm{E}_{\boldsymbol{\theta}}[\phi' \mathbf{f}^{\mathrm{T}}] = \sum_{i=1}^{N} \sum_{y^i} p(y^i|\mathbf{x}^i; \boldsymbol{\theta}) \phi'(y^i, \mathbf{x}^i) \mathbf{f}(y^i, \mathbf{x}^i)^{\mathrm{T}}. \tag{4.3}$$

Equation 4.3 shows that computing the GE gradient requires only $p(y^i|\mathbf{x}^i; \theta)$. Therefore no additional inference algorithms are needed, and both supervised and GE training have time complexity of $O(N|\mathcal{Y}|)$. In practice GE is slower than supervised training because computing the GE gradient requires two passes over the data.

4.2 Learning with Labeled Features using GE

In this chapter we focus on *text classification*, in which the task is to assign a label to a text document. Example text classification problems include email, webpage, and sentiment polarity classification. Even if we do not have labeled data for these tasks, we often have prior knowledge that we can use as light supervision: knowledge about the distribution over labels for the set of documents that contain a particular word. It is unrealistic to assume that users can provide a precise estimate of this distribution directly. Instead, we propose an approach in which the user provides list of labels that comprise most of the probability mass. For example, rather than provide a distribution over labels for documents that contain the word "game", the user could instead provide a list of labels related to sports, for example *baseball* and *hockey*. We refer to this unit of supervision as a *labeled input feature*.

We would like to estimate parameters using only a set of labeled input features and unlabeled data. In this chapter we take the following simple approach. First, we convert each labeled input feature into a target distribution using a heuristic. Through this conversion we provide a labeled input feature with a precise meaning. Then, we use GE to encourage the model to match these target distributions. Although this method may introduce noise, the experiments in this chapter demonstrate that precise estimates of the target distributions are not necessarily required.

4.2.1 Estimating Target Distributions

Let q be an input feature, L_q be its set of labels, and $\tilde{\boldsymbol{\phi}}_q$ be the target distribution for q, a vector of dimensionality $|\mathcal{Y}|$. We use two methods for converting labeled input features (q, L_q) into target distributions $\tilde{\boldsymbol{\phi}}_q$.

- *Schapire distributions:* As proposed by Schapire et al. [101], we use a simple heuristic in which a majority of the probability mass for an input feature is distributed uniformly among its user-specified labels, and the remaining probability mass is distributed uniformly among the other labels. Specifically, we assign probabilities

$$\tilde{\phi}_{qy} = \begin{cases} p_{maj}/|L_q| & y \in L_q \\ (1 - p_{maj})/(|\mathcal{Y}| - |L_q|) & y \notin L_q, \end{cases} \tag{4.4}$$

  where p_{maj} is the *majority probability*. In this chapter we use $p_{maj} = 0.9$.

- *Feature-voted distributions:* Alternatively, we use the labeled input features to vote on labels for the unlabeled instances. For each labeled input feature that is present in an instance, it contributes a vote for each of its labels. We then

normalize the vote totals to get a distribution over labels for each instance i, $p_v(y|\mathbf{x}^i)$. Finally, we estimate target distributions with this soft-labeled data.

$$\tilde{\phi}_{qy} = \sum_{i=1}^{N} \mathrm{E}_{p_v(y|\mathbf{x}^i)}[\phi_{qy}(\mathbf{x}^i, y)] \tag{4.5}$$

4.2.2 Objective Function

We encourage the model to match the target distributions using GE. The constraint feature for this application is

$$\phi_{qy'}(\mathbf{x}, y) = \frac{1}{c_q} \mathbf{1}_{\{y=y'\}} q(\mathbf{x}). \tag{4.6}$$

As discussed in Section 3.1.1, the expectation of $\phi_{qy'}$ is the probability that instances with q are labeled y' by the model. We use a KL divergence score function S_{KL}. The complete objective function, including a Gaussian prior on parameters, is

$$\mathcal{O}(\boldsymbol{\theta}) = \sum_{q} \tilde{\boldsymbol{\phi}}_q^{\mathrm{T}} \log(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}_q]) - \tilde{\boldsymbol{\phi}}_q^{\mathrm{T}} \log(\tilde{\boldsymbol{\phi}}_q) - \frac{1}{2\sigma^2} \left\| \boldsymbol{\theta} \right\|_2^2, \tag{4.7}$$

where $\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}_q] = \sum_{i=1}^{N} \sum_{y^i} p(y^i|\mathbf{x}^i; \boldsymbol{\theta}) \boldsymbol{\phi}_q(\mathbf{x}^i, y^i)$. While in general it may be beneficial to tune σ^2, in the following experiments we simply use $\sigma^2 = 1$.

4.2.3 Selecting Input Features to Label

We consider both scenarios in which 1) the user supplies labeled input features directly, and 2) the user is provided with a set of candidate input features to label. Importantly, in scenario 2) we give the user the option to skip labeling an input feature if they are unsure or the input feature is uninformative. For example, the user might skip labeling if presented the word "the". We propose the following method to select input features to label for scenario 2).

We could select candidate input features randomly or by frequency. However, the random method may select uninformative or infrequent features, while the frequency method would select common, uninformative features. Instead we perform unsupervised input feature clustering and select the most prominent input features in each cluster. Specifically, we use latent Dirichlet allocation (LDA) [9], a widely used topic model. For each LDA topic t_i, we sort input features q by $p(q|t_i)$ and choose the top f input features. There is no guarantee that the candidate input features selected by this heuristic are relevant to the learning task of interest. However, in practice this method is preferable to the alternatives. We refer to this method as *LDA features*.

4.3 Related Work

Several other methods for incorporating labeled input features into text classification have been proposed. Many of these methods convert labeled input features into labeled instances. Liu et al. [67] use human annotators to label features that are highly predictive of unsupervised instance clustering assignments. The unlabeled instances are soft-labeled according to their cosine similarity with pseudo-instances that only contain labeled input features, and this soft-labeled data is used as initialization for the expectation maximization (EM) algorithm. Schapire, Rochery, and Gupta [101] use hand-crafted rules based on relevant input features to label instances, and modify AdaBoost to choose weak learners that both fit the labeled training data and the soft-labeled data. Wu and Srihari [125] use labeled input features to assign labels and confidence scores to unlabeled instances, which are then used in conjunction with labeled data during training. We compare with the methods of Schapire, Rochery, and Gupta [101] and Wu and Srihari [125] in Sections 4.4.4 and 4.4.5, respectively. As discussed in Section 2.2.3, GE is preferable to these methods because the mapping between labeled input features and labels is automatically determined.

Dayanik et al. [21] propose several methods that use labeled input features to specify prior distributions on the parameters of a logistic regression model. As discussed in Section 2.2.3, GE does not require the practitioner to make statements about difficult-to-interpret parameter values.

Some recent work has addressed active learning by labeling input features. Raghavan, Madani, and Jones [94] interleave feedback on instances and input features in an algorithm called *tandem learning*. They show that incorporating feedback on input features can significantly accelerate active learning. Experiments also demonstrate that humans can provide accurate information about input features, and that it takes five times as long to label instances as to label input features. Raghavan and Allan [93] provide additional methods for training SVMs with labeled input features related to those described above, including scaling the parameters of labeled input features, creating specially-weighted pseudo-instances containing only labeled input features, and soft-labeling unlabeled instances. We compare with tandem learning in Section 4.4.6.

4.3.1 Recent Work

In work published after the work presented in this chapter, Sindhwani and Melville [108] propose a graph-based *co-regularization* approach to leveraging labeled input features and instances. In this method a bipartite graph of instances and features is constructed in which a weighted undirected edge between a feature and an instance indicates the frequency of the feature in the instance. A labeling function f is learned on this graph such that labeled instances and features get their correct labels, and f is smooth over the graph. The smoothness condition incorporates unlabeled instances and features. Sindhwani et al. [109] compare co-regularization with GE on binary problems. If the best value of a hyperparameter of co-regularization is selected, co-regularization outperforms GE in 6 of 10 cases. Note that we do not tune

hyperparameters in the experiments that follow. Additionally, there is no discussion of the application of co-regularization to multi-class problems.

As in the work of Dayanik et al. [21], Settles [104] proposes a method for using labeled input features to specify priors on model parameters. Specifically, labeled input features are used to set the concentration parameters of Dirichlet priors on multinomial distributions for naive Bayes models. The EM algorithm is then used to incorporate unlabeled data. Settles [104, 105] reports that this method significantly outperforms GE in 5 of 8 cases with 10 labeled input features per class, but that GE often performs better with 30 labeled input features per class. Note that when complete input feature label distributions are available, it is more natural to incorporate this information into learning with GE than with difficult-to-interpret hyperparameter values. Additionally, we conjecture that GE training would outperform an extension of this method in complex structured prediction tasks.

4.4 Experiments

In this section we present experiments with the proposed method for learning with labeled input features using both real and simulated users.

4.4.1 Simulated User

To simulate scenario 1), in which the user both selects the input features and labels them, we use the following method to select input features.

Ideally, a selected input feature should be both highly predictive of some subset of labels, and occur frequently enough to have an impact. We simulate the selection of such input features by revealing the labels of unlabeled instances, which simulates human background knowledge. Input features are then selected in descending order of their predictive power, as measured by their mutual information with the class label. We refer to this method as *oracle features*.

We use the following method to simulate the labeling of an input feature.

We simulate deciding whether to accept, rather than skip, an input feature by computing its mutual information with the class label, and accepting if the mutual information is $> \alpha$. In the following experiments, α is the mean of the mutual information values of the top M most predictive features, where $M = 100|\mathcal{Y}|$, or 100 times the total number of labels. If accepted, the oracle labels an input feature with the label it occurs with most often, and any other label that it occurs with at least half as often. We note that because M is typically small relative to the total number of input features, the oracle is somewhat conservative in the input features it accepts. This simulates a scenario in which the user only knows about the most prominent and important input features. We refer to this as the *oracle labeler* method.

Example labeled input features obtained using the *oracle features* method and the *oracle labeler* are provided in Table 4.1.

4.4.2 Experimental Setup

We evaluate the effectiveness of GE on six text classification data sets. For all data sets, instances correspond to documents and features are word counts. For the tasks in which a single instance can be assigned multiple labels, we split the task into $\mathcal{Y}$ one vs. all binary learning tasks, where $\mathcal{Y}$ is the number of labels. For other data sets, we use multi-class classification. We describe the data sets below.

- **reuters21578:**[1] A standard text categorization data set in which task is to assign categories to news articles. We use the ModApte split and evaluate on the top 10 most frequent classes, as in [125] (9603 training instances, 3299 testing instances).

[1] http://kdd.ics.uci.edu/

input feature	label
bad	*negative*
unfunny	*negative*
terrific	*positive*
minute	*negative*
poor	*negative*
awful	*negative*
idiotic	*negative*
ridiculous	*negative*
lame	*negative*
fantastic	*positive*
ludicrous	*negative*
performances	*positive*
family	*positive*
laughable	*negative*
portrayal	*positive*
waste	*negative*
hilarious	*positive*
embarrassing	*negative*
excellent	*positive*
worst	*negative*

input feature	labels
handouts	*course*
areas	*faculty*
teaching	*course*, *faculty*
problem	*course*
grade	*course*
description	*course*
final	*course*
phone	*faculty*, *student*
exam	*course*
material	*course*
advisor	*student*
department	*faculty*, *student*
selected	*faculty*
received	*faculty*, *student*
professor	*faculty*
science	*faculty*, *student*
required	*course*
ta	*course*
week	*course*
readings	*course*

Table 4.1: Randomly selected labeled input features obtained using *oracle features* and the *oracle labeler* for the **movie** (left) and **webkb** (right) data sets.

- **20 newsgroups**[2] The task is to classify messages according to the newsgroup to which they were posted. We use both the entire data set (20 classes, 20,000 instances) and binary subsets (2,000 instances).
- **movie**[3] The Polarity v2.0 data set, in which the task is to classify the sentiment of movie reviews as positive or negative (2,000 instances).
- **sraa**[2] The task is to classify messages about real and model automobiles and aviation with the appropriate newsgroup (4 classes, 73,218 instances).

[2]http://www.cs.umass.edu/∼mccallum/code-data.html

[3]http://www.cs.cornell.edu/People/pabo/movie-review-data/

- **webkb**[4] The task is to classify university webpages as *student*, *course*, *faculty*, or *project* (4,199 instances).
- **industry sector**[2] The task is to classify webpages according to a hierarchy of industrial sectors (4,582 instances). We use binary subsets, and the top level categories (7 classes).

For data sets without a standard test/train split, we randomly split the data such that 75% is used as training data, and the remaining 25% is reserved for testing. We use 10 such random splits and report the mean of the results. For experiments that do not use labeled instances we simulate unlabeled data by hiding labels of all instances. Experiments with GE never include labeled instances.

4.4.3 Comparison with Baselines

We first compare GE with several baseline methods, described below.

- **feature voting:** Use the labeled input features to vote on the classification.
- **feature labeling:** Use the labeled input features to vote on labels for the unlabeled instances and train a supervised model on this data. We do not include instances without a labeled input feature, and use hard class assignments, which provided significantly better results in our experiments.
- **labeled only:** Use GE to estimate parameters of a model with a reduced feature set that only includes labeled input features ($\mathbf{f} = \boldsymbol{\phi}$).

For these simulated user experiments we use the *oracle labeler* and select the top $25|\mathcal{Y}|$ input features (according to *oracle features* or *LDA features*) for labeling. Data sets **med-space**, **ibm-mac**, and **baseball-hockey** are subsets of the **20 newsgroups** data set; **healthcare-financial** is a subset of the **industry sector** data set.

[4] http://www.cs.cmu.edu/∼webkb

We run experiments comparing the above baselines with GE and provide the results in Tables 4.2, 4.3, 4.4 and 4.5. The parenthesized number with each data set indicates the mean number of input features labeled by the oracle labeler. The results presented in Tables 4.2 and 4.3 are obtained using *oracle features* and *Schapire distributions*. This simulates a scenario in which there is a domain expert who can suggest and label relevant input features. We also run experiments using *LDA features* and *Schapire distributions*, which simulates a scenario in which some candidate input features are presented to the labeler. The results are presented in Tables 4.4 and 4.5. We evaluate the methods with macro-averaged F_1, the mean of the F_1 for each label. GE attains the highest macro-F_1, in 7 of the 9 data sets using *oracle features*, and 7 of 9 using *LDA features*. Results marked with a * indicate that GE performs significantly better under a two-tailed paired t-test with significance level $\alpha = 0.05$.

We also perform experiments to determine the effectiveness of GE in relation to semi-supervised training with labeled documents. We use entropy regularization (ER) [41], a discriminative semi-supervised learning method that aims to minimize the uncertainty of predictions on unlabeled data. This method was discussed previously in Section 2.2.2. ER introduces a tuning parameter λ that controls the weight of the regularizer relative to the data likelihood. We set $\lambda = 0.2$, a value that provided the best mean results across all data sets, and perform training with a deterministic annealing procedure. We report the number of instances at which the performance of GE and the instance learning method are statistically indistinguishable. Raghavan, et al. [94] perform a thorough user study in which they conclude that it is five times faster to label an input feature than to label a document. We use this result to present estimated speed-ups using GE over entropy regularization. We note that in the computation of this estimated speed-up, we consider the number of input features presented to the labeler, including those that are skipped. Since we expect skipping

	Learning with Labeled Input Features			
data set	**feat. voting**	**feat. label**	**labeled only**	**GE**
movie (43.7 of 50)	0.763*	0.766*	0.772*	**0.797**
sraa (97.5 of 100)	0.630*	0.596*	0.585*	**0.651**
webkb (88.8 of 100)	0.496*	0.477*	0.745*	**0.774**
med-space (50.0 of 50)	0.907*	0.932*	0.930*	**0.952**
ibm-mac (43.7 of 50)	0.853	**0.864**	0.861	0.855
baseball-hockey (50 of 50)	0.925*	0.927*	0.939*	**0.954**
20 newsgroups (494.4 of 500)	0.554*	0.560*	0.643*	**0.704**
financial-healthcare (50 of 50)	**0.653**	0.443*	0.539*	0.583
sector.top (163.9 of 175)	0.664*	0.657*	0.719*	**0.730**

Table 4.2: Macro-averaged F_1 for methods that use labeled input features. Candidate input features are selected using *oracle features*. A * indicates that GE performs significantly better using a two-tailed paired t-test with significance level $\alpha = 0.05$.

an input feature to be faster than labeling an input feature, the estimates in Table 2 are likely to be conservative.

Each of the baselines demonstrates an important point about GE. **Feature voting** uses the domain knowledge only, whereas GE uses this information to constrain model predictions on unlabeled data, and in the process learns about co-occurring features without labels. **Labeled only** again demonstrates the importance of incorporating these co-occurring features without labels. Finally, suppose that an input feature q has two labels, y_0 and y_1. The **feature labeling** method interprets this to mean that a document with q should have a uniform distribution over y_0 and y_1. In contrast, GE does not specify a distribution for individual documents, permitting a more reasonable solution in which half of documents with q receive label y_0, and half receive label y_1.

4.4.4 Comparison with Schapire, Rochery, and Gupta [2002]

In this experiment, we compare GE with *boosting with prior knowledge* [101]. Boosting with prior knowledge aims to maximize the conditional log likelihood of both labeled instances and instances classified using a hand-crafted model. The hand-crafted model classifies instances using the product of label probabilities for

	Labeled Instances Required	
data set	**sup. + ER**	**est. speed-up**
movie (43.7 of 50)	150	15.0
sraa (97.5 of 100)	160	8.0
webkb (88.8 of 100)	70	3.5
med-space (50.0 of 50)	90	9.0
ibm-mac (43.7 of 50)	110	11.0
baseball-hockey (50 of 50)	200	20.0
20 newsgroups (494.4 of 500)	650	6.5
financial-healthcare (50 of 50)	50	5.0
sector.top (163.9 of 175)	140	4.0

Table 4.3: The number of labeled instances at which semi-supervised training becomes statistically indistinguishable from GE, and the estimated speed-up if labeling an input feature is 5 times faster than labeling a document.

features, which are estimated from labeled input features using the *Schapire distributions* heuristic. Schapire et al. provide 138 labeled input features for the **20 newsgroups** data set. For comparison, we use the same labeled input features and use the *Schapire distributions* heuristic to estimate target distributions. We note that the experiments in [101] use n-gram features, whereas we use only unigram features. Comparing using the domain knowledge only, GE gives approximately a 15% absolute error reduction from 64% error ([101] Figure 3) to 49% error. Furthermore, the boosting method requires the domain knowledge and between 400 and 800 labeled documents for boosting with prior knowledge to match the accuracy of GE, which uses no labeled documents.

4.4.5 Comparison with Wu and Srihari [2004]

Next, we compare GE with a method for leveraging labeled input features using *Weighted Margin Support Vector Machines* (WMSVMs) [125]. Wu and Srihari provide a few input features associated with each of the top 10 most frequent classes in the ModApte split of the **Reuters21578** data set. With WMSVMs, a macro-average break-even-point of around 0.53 is obtained using only this domain knowledge, and a

	Learning with Labeled Input Features			
data set	**feat. voting**	**feat. label**	**labeled only**	**GE**
movie (4.6 of 50)	0.616	0.608	0.607*	**0.623**
sraa (29.5 of 100)	**0.577**	0.526*	0.520*	0.559
webkb (17.5 of 100)	0.514*	0.513*	0.593*	**0.615**
med-space (14.3 of 50)	0.857*	0.862*	0.867*	**0.927**
ibm-mac (10.4 of 50)	0.740*	**0.817**	0.762*	**0.817**
baseball-hockey (10.8 of 50)	0.779*	0.840*	0.853*	**0.915**
20 newsgroups (269.6 of 500)	0.493*	0.514*	0.585*	**0.667**
financial-healthcare (9.4 of 50)	0.552*	0.456*	**0.595**	0.588
sector.top (50.7 of 175)	0.538*	0.534*	0.544*	**0.596**

Table 4.4: Same Figure 4.2 as above, but candidate input features are selected using *LDA features*.

	Labeled Instances Required	
data set	**sup. + ER**	**est. speed-up**
movie (4.6 of 50)	20	2.0
sraa (29.5 of 100)	80	4.0
webkb (17.5 of 100)	20	1.0
med-space (14.3 of 50)	40	4.0
ibm-mac (10.4 of 50)	50	5.0
baseball-hockey (10.8 of 50)	40	4.0
20 newsgroups (269.6 of 500)	300	3.0
financial-healthcare (9.4 of 50)	50	5.0
sector.top (50.7 of 175)	60	1.7

Table 4.5: Same as Figure 4.3, but candidate input features are selected using *LDA features*.

macro-average break-even-point of around 0.60 is obtained using domain knowledge and 16 labeled examples ([125] Figure 3). Using the same domain knowledge, *feature-voted distributions*, and no labeled documents, GE attains a break-even-point of 0.630.

4.4.6 Comparison with Raghavan [2007]

We also provide an informal comparison with *tandem learning* [93], an active learning algorithm that incorporates labeled instances and input features into learning with Support Vector Machines. We call the comparison informal because tandem

learning is quite different from GE. Importantly, GE uses neither active learning nor labeled documents. In the referenced experiments, tandem learning uses a total of 12 labeled documents, and shows at most 100 input features to the annotator. Both input features and instances are actively selected to reduce uncertainty. Conversely, we use a static list of input features, chosen before learning begins using an unsupervised method. We compare performance on the **20 newsgroups** data set. We use a one vs. all setup for better comparison. Raghavan et al. report macro-F_1 of 0.354 ([93] Table 3). With 100 candidate features selected using *LDA features*, target distributions estimated using *association-voted-distributions*, and the *oracle labeler*, we attain macro-F_1 of 0.477, averaged over 10 random splits of the data.

4.4.7 User Experiments

Finally, we conduct annotation experiments in which we time three users as they label 100 documents and 100 input features for binary classification tasks. The candidate input features are selected using *LDA features*. The input features are presented one at a time, and the user can choose a single label for the input feature or choose to discard the input feature. After the users finish labeling input features, they label documents, again with the option to choose a label for the document or to skip the document if it appears ambiguous. We prefer this ordering (labeling input features followed by documents) in order to give maximum benefit to the traditional document labeling method. We choose documents to present to the user with uncertainty sampling: after each instance is labeled, the instance with the most uncertain classification under the current model is selected for labeling. We do this to ensure that the labeled instances are beneficial. The list of candidate input features is static.

First, we are interested in the accuracy of the human annotators. Table 4.6 shows the labeling precision and recall for different annotators. For input feature labeling, performance is measured using the *oracle labeler* as ground truth; for document la-

	doc. labeling		feat. labeling	
user + data set	prec	rec	prec	rec
1 ibm-mac	0.90	0.58	0.80	1.00
1 med-space	0.95	0.86	0.73	1.00
1 baseball-hockey	0.98	0.84	0.52	0.92
2 ibm-mac	0.92	0.37	0.50	0.80
2 med-space	0.98	0.80	0.52	0.96
2 baseball-hockey	0.96	0.71	0.41	1.00
3 ibm-mac	0.91	0.75	0.86	1.00
3 med-space	0.99	0.75	0.67	1.00
3 baseball-hockey	0.96	0.83	0.54	1.00
Overall mean	0.95	0.72	0.62	0.96

Table 4.6: User labeling performance. Input feature labeling performance is with respect to the *oracle labeler*.

beling, performance is measured using the true labels. The labelers provided precise labels for documents, but also discarded many documents. The labelers were able to both correctly identify and label almost all input features that the *oracle labeler* considers relevant, as indicated by the high recall values. The users also labeled many additional input features that are actually relevant. For example, all users correctly identified the names of several baseball teams (*braves*, *cubs*, and *jays*) and a hockey team (*flyers*), but these input features are not labeled by the simulated user. We defined the simulated user to be conservative, only choosing to label input features that are almost certainly relevant. Consequently, the precision values are artificially low. The users did make some mistakes when labeling input features, however, as discussed in more detail in Section 8.1. User 2 had the most trouble selecting and labeling input features. We suspect that this indicates insufficient familiarity with the learning tasks. This suggests that future experiments should involve an opportunity to look through the data before annotation. However, it does not seem unreasonable to assume that the annotators are familiar with the task they are trying to solve.

Figures 4.1 and 4.2 show the accuracy of two trained systems over time. The first uses the labeled input features and unlabeled instances with GE. Target distri-

med: blood, cancer, care, disease, doctor, doctors, drugs, health, medical, medicine, pain, patients, vitamin, yeast **space**: earth, launch, mars, mission, moon, nasa, orbit, planet, satellite, shuttle, sky, space, universe
ibm: hp, dos, ibm **mac**: apple, mac
baseball: ball, baseball, braves, cubs, hit, hitter, jays, pitching, runs **hockey**: flyers, goal, hockey, leafs, nhl, period, shots

Table 4.7: Input features that all three users labeled.

butions are estimated using *Schapire distributions* with $q_{maj} = 0.9$. The second uses entropy regularization (ER) [41] (in this experiment we use direct maximization and weighting parameter $\gamma = 0.01$) with the labeled and unlabeled instances. Annotating input features yields large accuracy improvements for the same amount of time. On average across all experiments, labeling input features is 3.7 times faster than labeling documents, and the models trained with GE have 1.0% higher final accuracy. Note that the point at which the GE curve changes from a dotted line into dots indicates the point at which the user had processed all 100 input features.

When the annotator is accurate, the results with input feature labeling can be quite striking. For example, consider the results of User 1 for the **ibm vs. mac** classification task. The accuracy of the GE system after 30 seconds of feature labeling is better than the accuracy of the ER system after 12 minutes of document labeling, a 24x speed-up. As another example, User 3 achieves accuracy of 90% on the **baseball vs. hockey** task after 90 seconds with the GE system, at which point the ER system accuracy is around 50%.

Notice that the ER system gives erratic performance, with large accuracy jumps in consecutive 30 second intervals. This reinforces our earlier assertions about the delicateness of traditional semi-supervised learning methods.

4.5 Conclusion and Future Work

In this chapter, we applied GE to train logistic regression models using labeled input features. Experiments show that GE outperforms other methods for learning with labeled input features and that labeling input features rather than labeling documents provides a more efficient way to train a document classifier.

This chapter suggests many directions for additional research, many of which are explored in later chapters.

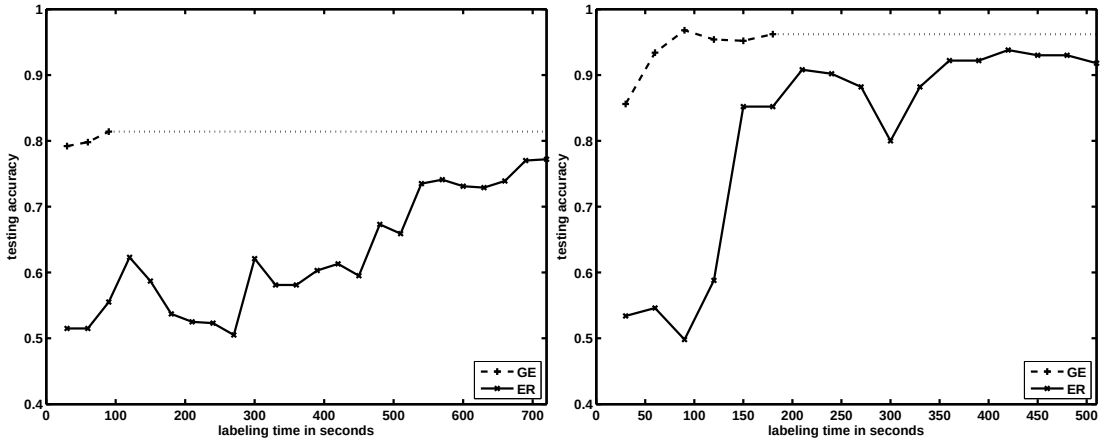
- The target distributions used in this chapter are very noisy, as the targets are set using simple heuristics. Additionally, input features may be labeled incorrectly. We explore strategies for compensating for noise in Section 8.1.

- In the experiments in Section 4.4.7, active learning is used when labeling documents but not when labeling input features. We develop an active learning method in which users label features in Chapter 9.

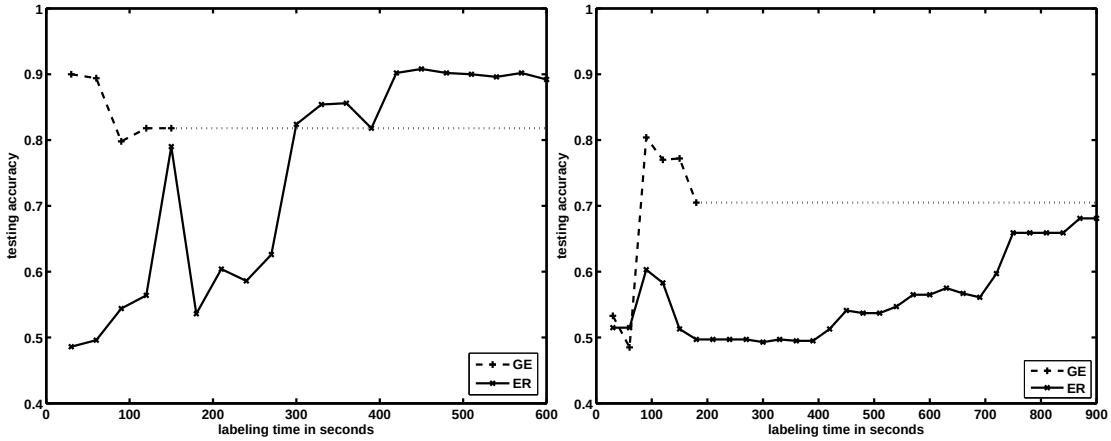
- We develop methods for efficiently evaluating the accuracy of classifiers trained using labeled input features in Chapter 10.

Additional conclusions and discussion are provided in Chapter 12.

User 1: ibm-mac, med-space



User 1: baseball-hockey, User2: ibm-mac



User 2: med-space, baseball-hockey

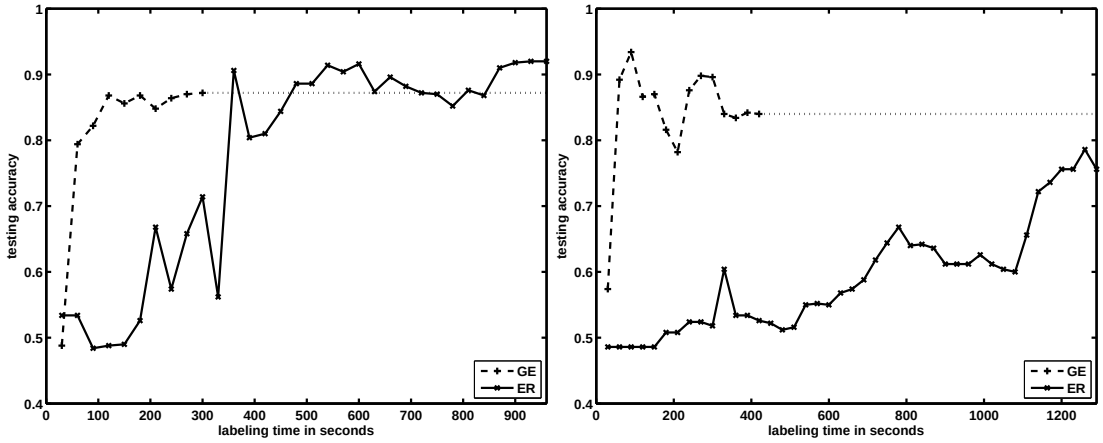
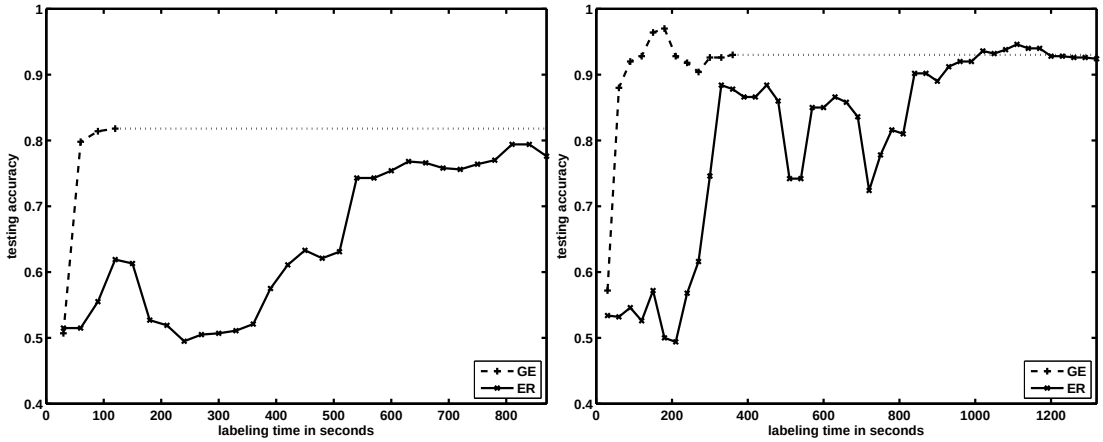


Figure 4.1: Accuracy vs. time for the GE and ER systems with Users 1 and 2. In most cases, GE gives better accuracy given the same amount of annotation time.

User 3: ibm-mac, med-space



User 3: baseball-hockey

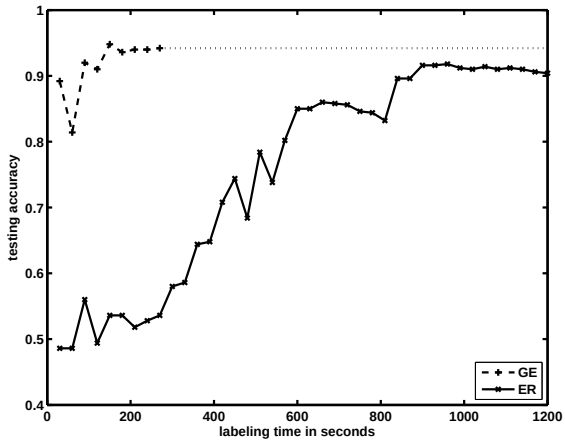


Figure 4.2: Accuracy vs. time for the GE and ER systems with User 3. In most cases, GE gives better accuracy given the same amount of annotation time.

CHAPTER 5

GE FOR LIGHTLY SUPERVISED DEPENDENCY PARSING

We next apply GE to lightly supervised non-projective dependency parsing. In *dependency parsing* the task is to output a *dependency tree* for a given input sentence. A dependency tree represents syntactic dependencies among lexical elements in the input sentence. A syntactic dependency is represented by directed edge from a *parent*, or *head*, to a *child*, or *modifier*. Multiple criteria for determining whether a dependency holds between a pair of lexical elements, as well as determining the direction of the dependency, have been proposed [84]. A *projective* dependency tree can be drawn such that there are no crossing edges when nodes are ordered as they appear in the sentence. In this chapter we consider the more general *non-projective* case, in which any valid tree is permitted. In *labeled dependency parsing*, in addition to predicting the tree, a parser must predict a label for each edge. For simplicity, in this chapter we focus on the unlabeled case. Additional background on dependency parsing is provided in [75, 76, 84, 112].

In this chapter, we derive novel algorithms for GE training of CRF models of non-projective dependency trees. We use the resulting method to leverage directly expressed linguistic prior knowledge (e.g. a *noun*'s parent is often a *verb*) to train dependency parsers with no labeled data. This is an important problem because annotating data for syntactic parsing is incredibly time-consuming. In a comparison with two prominent "unsupervised" learning methods that require indirect biasing toward the correct syntactic structure, we show that GE can attain better accuracy with as few as 20 intuitive constraints.

5.1 Tree Conditional Random Fields

We first define a CRF $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$ that models unlabeled, non-projective dependency trees. We choose non-projective parsing because it is the more general case. The tree $\mathbf{y}$ is represented as a vector of the same length as the sentence $\mathbf{x}$, where y_i is the index of the parent of lexical element x_i. The probability of a tree $\mathbf{y}$ given sentence $\mathbf{x}$ is

$$p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) = \frac{1}{Z(\mathbf{x};\boldsymbol{\theta})} \exp\Big(\sum_{i=1}^{n} \boldsymbol{\theta} \cdot \mathbf{f}(x_i, x_{y_i}, \mathbf{x})\Big), \tag{5.1}$$

where $\mathbf{f}$ are *edge-factored* feature functions that consider the child lexical element, or input (word, tag, or other feature), the parent input, and the rest of the sentence. This factorization implies that dependency decisions are independent conditioned on the input sentence $\mathbf{x}$ if $\mathbf{y}$ is a tree. Computing $Z(\mathbf{x};\boldsymbol{\theta})$ and the edge expectations needed for parameter estimation require summing over all possible trees for $\mathbf{x}$.

By relating the sum of the scores of all possible trees to counting the number of spanning trees in a graph, it can be shown that $Z(\mathbf{x};\boldsymbol{\theta})$ is the determinant of the *Kirchoff matrix*, $\mathbf{K}_{\mathbf{x};\boldsymbol{\theta}} \in \mathbb{R}^{n\times n}$, which is constructed using the scores of possible edges [55, 76, 112]. The score of edge $j \to i$, $s_{\mathbf{x};\boldsymbol{\theta}}(i,j)$, is defined

$$s_{\mathbf{x};\boldsymbol{\theta}}(i,j) = \exp\Big(\boldsymbol{\theta} \cdot \mathbf{f}(x_i, x_j, \mathbf{x})\Big). \tag{5.2}$$

We define $s_{\mathbf{x};\boldsymbol{\theta}}(i,i) = 0$. Note that index 0 corresponds to the special *root* symbol. The Kirchoff matrix $\mathbf{K}_{\mathbf{x};\boldsymbol{\theta}}$ is defined

$$[\mathbf{K}_{\mathbf{x};\boldsymbol{\theta}}]_{j,i} = \begin{cases} \sum_{k\in\{0,\ldots,n\}:k\neq i} s_{\mathbf{x};\boldsymbol{\theta}}(i,k) & : i = j \\ -s_{\mathbf{x};\boldsymbol{\theta}}(i,j) & : i \neq j. \end{cases}$$

In general we use the notation $[\mathbf{A}]_{i,j}$ to denote the value at row i and column j in matrix $\mathbf{A}$. Computing the determinant $|\mathbf{K}_{\mathbf{x};\boldsymbol{\theta}}| = Z(\mathbf{x};\boldsymbol{\theta})$ takes $O(n^3)$ time, where n is the length of the sentence.

One method for computing the marginal probability of a particular edge $k \to i$ (i.e. $y_i = k$), is to set the score of any edge $k' \to i$ such that $k' \neq k$ to 0. The determinant of the resulting modified Kirchoff matrix $\mathbf{K}^{k \to i}_{\mathbf{x};\boldsymbol{\theta}}$ is then the sum of the scores of all trees that include the edge $k \to i$. The marginal $p(y_i = k|\mathbf{x};\boldsymbol{\theta})$ can be computed by dividing this score by $Z(\mathbf{x};\boldsymbol{\theta})$ [76]. There are $O(n^2)$ possible edges in each sentence, so computing edge expectations with this algorithm takes $O(n^5)$ time.

Smith and Smith [112] describe a more efficient algorithm that can compute edge expectations in $O(n^3)$ time using the inverse of the Kirchoff matrix $\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}$. This algorithm is derived by computing the gradient of the log partition function directly [112].

$$\frac{\partial \log Z(\mathbf{x};\boldsymbol{\theta})}{\partial \theta_m} = \sum_{i=1}^{n} \sum_{j=0}^{n} s_{\mathbf{x};\boldsymbol{\theta}}(i,j) f_m(x_i, x_j, \mathbf{x}) \big([\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,i} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,j} \big) \tag{5.3}$$

Equation 5.3 computes edge expectations. The edge marginals are

$$p(y_i = k|\mathbf{x};\boldsymbol{\theta}) = s_{\mathbf{x};\boldsymbol{\theta}}(i,k) \big([\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,i} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,k} \big). \tag{5.4}$$

5.2 GE for Tree CRFs

In this section we derive an inference algorithm to compute covariance in a Tree CRF, allowing us to apply GE. For discussion of an alternative algorithm, see the last paragraph of Section 6.4. Because we assume that trees for different sentences are independent, covariance decomposes into the sum of per-instance covariances, as discussed in Section 3.2.2. If constraint features are edge-factored, the covariance for one sentence, using the composite constraint feature ϕ' discussed in Section 3.2.1, is

$$\begin{aligned}\mathrm{COV}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}\Big(\phi'(\mathbf{x},\mathbf{y}),\mathbf{f}(\mathbf{x},\mathbf{y})\Big) &= \sum_{y_i,y_j} p(y_i,y_j|\mathbf{x};\boldsymbol{\theta})\phi'(x_i,x_{y_i},\mathbf{x})\mathbf{f}(x_j,x_{y_j},\mathbf{x})^{\mathrm{T}} \\ &\quad - \Big(\sum_{y_i} p(y_i|\mathbf{x};\boldsymbol{\theta})\phi'(x_i,x_{y_i},\mathbf{x})\Big) \\ &\quad \times \Big(\sum_{y_j} p(y_j|\mathbf{x};\boldsymbol{\theta})\mathbf{f}(x_j,x_{y_j},\mathbf{x})^{\mathrm{T}}\Big). \end{aligned} \tag{5.5}$$

The second term of the covariance can be computed using the edge marginals $p(y_i|\mathbf{x};\boldsymbol{\theta})$. The first term of the covariance is more difficult to compute because it requires the marginal probability of two edges $p(y_i,y_j|\mathbf{x};\boldsymbol{\theta})$ occurring in the same tree.

We proceed by computing the second partial derivative of the log partition function. The first partial derivative of the log partition function is an expectation, so the second derivative is the covariance, as we saw in Section 3.2.1. We will use this fact to derive an $O(n^4)$ algorithm for computing the covariance. Note that although we derive the covariance between features that appear in the model, the resulting expression can also be used to compute the model predicted covariance between two arbitrary edge-factored features (that may or may not be in the model).

The second partial derivative of the log partition function with respect to θ_m and θ_n (the parameters for feature functions f_n and f_m) is

$$\begin{aligned}\frac{\partial^2 \log Z(\mathbf{x};\boldsymbol{\theta})}{\partial\theta_m\partial\theta_n} &= \sum_{k=1}^{n}\sum_{\ell=0}^{n} \frac{\partial^2 \log Z(\mathbf{x};\boldsymbol{\theta})}{\partial\theta_m\partial s_{\mathbf{x};\boldsymbol{\theta}}(k,\ell)} \frac{\partial s_{\mathbf{x};\boldsymbol{\theta}}(k,\ell)}{\partial\theta_n} \\ &= \sum_{k=1}^{n}\sum_{\ell=0}^{n} s_{\mathbf{x};\boldsymbol{\theta}}(k,\ell) f_n(x_k,x_\ell,\mathbf{x}) \frac{\partial^2 \log Z(\mathbf{x};\boldsymbol{\theta})}{\partial\theta_m\partial s_{\mathbf{x};\boldsymbol{\theta}}(k,\ell)}. \end{aligned} \tag{5.6}$$

If $j \to i$ and $\ell \to k$ are not the same edge, $i \neq k \ \vee j \neq \ell$, then

$$\frac{\partial^2 \log Z(\mathbf{x};\boldsymbol{\theta})}{\partial\theta_m\partial s_{\mathbf{x};\boldsymbol{\theta}}(k,\ell)} = \sum_{i=1}^{n}\sum_{j=0}^{n} s_{\mathbf{x};\boldsymbol{\theta}}(i,j) f_m(x_i,x_j,\mathbf{x}) \frac{\partial}{\partial s_{\mathbf{x};\boldsymbol{\theta}}(k,\ell)} \big([\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,i} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,j}\big). \tag{5.7}$$

For an arbitrary matrix $\mathbf{A}$, the derivative with respect to t of its inverse $\mathbf{A}^{-1}$ is

$$\frac{\partial \mathbf{A}^{-1}}{\partial t} = -\mathbf{A}^{-1}\frac{\partial \mathbf{A}}{\partial t}\mathbf{A}^{-1}. \tag{5.8}$$

The derivative of $\mathbf{A}$ with respect to a single cell in $[\mathbf{A}]_{k,l}$ is a matrix with a 1 at k, ℓ and zeros elsewhere. Therefore, the derivative of a cell in the inverse $[\mathbf{A}^{-1}]_{i,j}$ with respect to cell $[\mathbf{A}]_{k,\ell}$ is the product of two cells in the inverse.

$$\frac{\partial [\mathbf{A}^{-1}]_{i,j}}{\partial [\mathbf{A}]_{k,\ell}} = -[\mathbf{A}^{-1}]_{i,k}[\mathbf{A}^{-1}]_{\ell,j}. \tag{5.9}$$

Using this fact, we have

$$\begin{aligned}
\frac{[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,i}}{\partial s_{\mathbf{x};\boldsymbol{\theta}}(k,\ell)} &= [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,\ell}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,i} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,k}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,i}, \\
\frac{[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,j}}{\partial s_{\mathbf{x};\boldsymbol{\theta}}(k,\ell)} &= [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,\ell}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,j} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,k}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,j}.
\end{aligned} \tag{5.10}$$

Putting these terms together, we have (when $i \neq k \ \vee j \neq \ell$)

$$\begin{aligned}
\frac{\partial}{\partial s_{\mathbf{x};\boldsymbol{\theta}}(k,\ell)}\left([\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,i} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,j}\right) =& [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,\ell}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,i} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,k}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,i} \\
& - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,\ell}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,j} + [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,k}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,j}
\end{aligned} \tag{5.11}$$

$$\begin{aligned}
\frac{\partial^2 \log Z(\mathbf{x};\boldsymbol{\theta})}{\partial \theta_m \partial s_{\mathbf{x};\boldsymbol{\theta}}(k,\ell)} =& \sum_{i=1}^{n}\sum_{j=0}^{n} s_{\mathbf{x};\boldsymbol{\theta}}(i,j) f_m(x_i, x_j, \mathbf{x}) \\
& \times \Big([\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,\ell}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,i} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,k}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,i} \\
& - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,\ell}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,j} + [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,k}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,j} \Big).
\end{aligned} \tag{5.12}$$

If $j \to i$ and $\ell \to k$ are the same edge, $i = k \ \wedge j = \ell$, then

$$\begin{aligned}
\frac{\partial^2 \log Z(\mathbf{x};\boldsymbol{\theta})}{\partial\theta_m \partial s_{\mathbf{x};\boldsymbol{\theta}}(i,j)} =& \sum_{i=1}^{n}\sum_{j=0}^{n} \big([\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,i} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,j}\big) \frac{\partial}{\partial s_{\mathbf{x};\boldsymbol{\theta}}(i,j)} s_{\mathbf{x};\boldsymbol{\theta}}(i,j) f_m(x_i, x_j, \mathbf{x}) \\
&+ s_{\mathbf{x};\boldsymbol{\theta}}(i,j) f_m(x_i, x_j, \mathbf{x}) \frac{\partial}{\partial s_{\mathbf{x};\boldsymbol{\theta}}(i,j)} \big([\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,i} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,j}\big). \\
=& \sum_{i=1}^{n}\sum_{j=0}^{n} f_m(x_i, x_j, \mathbf{x}) \big([\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,i} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,j}\big) \\
&- s_{\mathbf{x};\boldsymbol{\theta}}(i,j) f_m(x_i, x_j, \mathbf{x}) ([\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,i} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,j})^2 \qquad (5.13)
\end{aligned}$$

Substituting back into Equation 5.6 gives us the covariance

$$\begin{aligned}
\frac{\partial^2 \log Z(\mathbf{x};\boldsymbol{\theta})}{\partial\theta_m \partial\theta_n} =& \sum_{k=1}^{n}\sum_{\ell=0}^{n}\sum_{i=1}^{n}\sum_{j=0}^{n} s_{\mathbf{x};\boldsymbol{\theta}}(k,\ell) f_n(x_k, x_\ell, \mathbf{x}) s_{\mathbf{x};\boldsymbol{\theta}}(i,j) f_m(x_i, x_j, \mathbf{x}) \\
&\times \Big([\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,\ell} [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,i} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,k} [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,i} \\
&\quad - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,\ell} [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,j} + [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,k} [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,j} \Big) \\
&+ \sum_{i=1}^{n}\sum_{j=0}^{n} s_{\mathbf{x};\boldsymbol{\theta}}(i,j) f_n(x_i, x_j, \mathbf{x}) f_m(x_i, x_j, \mathbf{x}) \\
&\times \big([\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,i} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,j} \big), \qquad (5.14)
\end{aligned}$$

where the last two lines accounts for the extra term introduced when $i = k \ \wedge j = \ell$.

Equation 5.14 computes the covariance between two edge-factored feature functions f_m and f_n. Notice that this computation only requires cells from the inverse Kirchoff matrix. Computing the inverse Kirchoff matrix takes $O(n^3)$ time for each sentence. The time required to compute the complete covariance is then $O(n^4)$, the time required to consider all possible pairs of edges. Note that Posterior Regularization [36], described in Section 3.3.1, would take $O(n^3)$ time for a tree CRF. As we discuss in Section 6.4, the GE gradient could also be computed in $O(n^3)$ time, though there may be reasons to prefer the above $O(n^4)$ algorithm.

Finally, we provide an expression for computing the two edge marginal itself. The covariance can also be written as

$$\begin{aligned}
&\mathrm{E}_{p(y_i=j,y_k=\ell|\mathbf{x};\theta)}[f_m(x_i,x_j,\mathbf{x})f_n(x_k,x_\ell,\mathbf{x})] \\
&\quad - \mathrm{E}_{p(y_i=j|\mathbf{x};\theta)}[f_m(x_i,x_j,\mathbf{x})]\mathrm{E}_{p(y_k=\ell|\mathbf{x};\theta)}[f_n(x_k,x_\ell,\mathbf{x})]. \qquad (5.15)
\end{aligned}$$

Therefore, the two edge marginal is

$$\begin{aligned}
p(y_i=j,y_k=\ell|\mathbf{x},\theta) =& [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,\ell}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,i} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,k}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,i} \\
&-[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,\ell}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,j} + [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,k}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,j} \\
&+[\mathbf{K}_{\mathbf{x};\boldsymbol{\theta}}]_{i,i}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,k} - [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,i}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,\ell} \\
&-[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,j}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,k} + [\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{i,j}[\mathbf{K}^{-1}_{\mathbf{x};\boldsymbol{\theta}}]_{k,\ell}. \qquad (5.16)
\end{aligned}$$

5.3 Lightly Supervised Dependency Parsing with GE

In the remainder of this chapter we use GE to train dependency parsers with naturally encoded linguistic insights. For example, we know that "in English, when a *determiner* is directly to the left of a *noun*, the *noun* is usually the parent of the *determiner*." We use edge-factored constraint features $\boldsymbol{\phi}$ that are the product of a predicate q that considers the head, modifier, and rest of the sentence, and a term that normalizes by the total number of *possible* edges for which q fires, c_q.

$$\phi_q(x_i,x_j,\mathbf{x}) = \frac{1}{c_q} q(x_i,x_j,\mathbf{x}) \qquad (5.17)$$

The expectation $\mathrm{E}_\theta[\phi_q]$ is then the probability that an edge is present when q fires. We encourage the model to match target expectations $\tilde{\boldsymbol{\phi}}$ with an L_2^2 score function, $\mathrm{S}_{L_2^2}$. The complete objective function also includes a Gaussian prior on parameters.

$$\mathcal{O}(\boldsymbol{\theta}) = -\left\|\tilde{\boldsymbol{\phi}} - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\right\|_2^2 - \frac{1}{2\sigma^2}\left\|\boldsymbol{\theta}\right\|_2^2 \qquad (5.18)$$

In the following experiments we use $\sigma^2 = 10$. We next discuss the selection of constraint features and estimation of targets.

5.3.1 Constraints from Linguistic Prior Knowledge

In this thesis, we use constraints derived from several basic types of linguistic knowledge. One simple form of linguistic knowledge is the set of possible parent tags for a given child tag. This type of constraint was used in the development of a rule-based dependency parser [22]. Additional information can be obtained from small grammar fragments. Haghighi and Klein [44] provide a list of prototype phrase structure rules that can be augmented with dependencies and used to define constraints involving parent and child tags, surrounding or interposing tags, direction, and distance. Finally there are well known hypotheses about the direction and distance of attachments that can be used to define constraints. Eisner and Smith [32] use the fact that short attachments are more common to improve unsupervised parsing accuracy.

5.3.2 Simulated User Constraints

For some experiments that follow we use "simulated user" constraints that are estimated from labeled data. This simulation involves choosing constraint features (motivated by the linguistic knowledge described above) and estimating target expectations. Simulated constraints used in this thesis consider three simple statistics of candidate constraint functions: count c_q, edge count e_q, and edge probability $p(e|q)$.

$$c_q = \sum_{i=1}^{N} \sum_{j} \sum_{k} q(x_j^i, x_k^i, \mathbf{x})$$

$$e_q = \sum_{i=1}^{N} \sum_{j} q(x_j^i, x_{y_j^i}^i, \mathbf{x})$$

$$p(e|q) = \frac{e_q}{c_q}$$

Constraint functions are selected according to some combination of the above statistics. In some cases we additionally prune the candidate set by considering only certain predicates. To compute the target expectation, we simply use $\text{bin}(p(e|q))$, where bin

returns the closest value in the set $\{0, 0.1, 0.25, 0.5, 0.75, 1\}$. This can be viewed as specifying that q is *very indicative of edge*, *somewhat indicative of edge*, etc.

5.4 Related Work

This work is related to the *prototype-driven* grammar induction method of Haghighi and Klein [44], which uses prototype phrases to guide the EM algorithm in learning a PCFG. Direct comparison with this method is not possible because we are interested in dependency syntax rather than phrase structure syntax. As discussed in Section 2.2.3, the primary advantage of GE over prototype-driven learning is that it uses expectation constraints, allowing one to specify how often a constraint should hold. In contrast, prototype-driven learning uses hard constraints. Additionally prototype-driven grammar induction needs to be used in conjunction with other unsupervised methods (distributional similarity and CCM [53]) to attain reasonable accuracy, and is only evaluated on length 10 or less sentences with no lexical information. In contrast, GE uses only the provided constraints and unparsed sentences, and is used to train a feature-rich discriminative model.

Conventional semi-supervised learning requires parsed sentences. Kate and Mooney [52] and McClosky and Johnson [74] both use modified forms of self-training to bootstrap parsers from limited labeled data. Wang et al. [122] combine a structured loss on parsed sentences with a least squares loss on unlabeled sentences. Koo et al. [54] use a large unlabeled corpus to estimate cluster features which help the parser generalize with fewer examples. Smith and Eisner [111] apply *entropy regularization* to dependency parsing. The above methods can be applied to small seed corpora, but McDonald[1] has criticized such methods as working from an unrealistic premise, as a significant amount of the effort required to build a treebank comes in the first 100

[1] R. McDonald, personal communication, 2007

sentences (both because of the time it takes to create an appropriate rubric and to train annotators).

There are also a number of methods for unsupervised learning of dependency parsers. Klein and Manning [53] use a carefully initialized and structured generative model (DMV) in conjunction with the EM algorithm to get the first positive results on unsupervised dependency parsing. As empirical evidence of the sensitivity of DMV to initialization, Smith [113] (pg. 37) uses three different initializations, and only one, the method of Klein and Manning [53] , gives accuracy higher than 31% on the WSJ10 corpus (see Section 5.5). This initialization encodes the prior knowledge that long distance attachments are unlikely.

Smith and Eisner [114] develop *contrastive estimation* (CE), in which the model is encouraged to move probability mass away from implicit negative examples defined using a carefully chosen neighborhood function. For instance, Smith [113] (pg. 82) uses eight different neighborhood functions to estimate parameters for the DMV model. The best performing neighborhood function DEL1ORTRANS1 provides accuracy of 57.6% on WSJ10 (see Section 5.5). Another neighborhood, DEL1ORTRANS2, provides accuracy of 51.2%. The remaining six neighborhood functions provide accuracy below 50%. This demonstrates that constructing an appropriate neighborhood function can be delicate and challenging.

Smith and Eisner [115] propose *structural annealing* (SA), in which a strong bias for local dependency attachments is enforced early in learning, and then gradually relaxed. This method is sensitive to the annealing schedule. Smith [113] (pg. 136) use 10 annealing schedules in conjunction with three initializers. The best performing combination attains accuracy of 66.7% on WSJ10, but the worst attains accuracy of 32.5%.

Seginer [102] and Bod [12] approach unsupervised parsing by constructing novel syntactic models. The development and tuning of the above methods constitute the

encoding of prior domain knowledge about the desired syntactic structure. In contrast, our framework provides a straightforward and explicit method for incorporating prior knowledge.

Finally, Ganchev et al. [34] use *Posterior Regularization*, introduced in Section 3.3.1, to learn a projective target language parser using only a source language parser and word alignments. The constraints specify that at least some of edges projected from the source language must appear in target language parses.

5.4.1 Recent Work

In work that was published after the work presented in this chapter, Naseem et al. [80] present a related approach for incorporating universal linguistic knowledge into dependency grammar induction. We refer to this method as HDP-DEP. HDP-DEP uses posterior regularization, introduced in Section 3.3.1, rather than GE, to estimate parameters of a complex generative model, rather than a simple conditional model. Specifically, the model is similar to DMV but encodes richer context information, generates words rather than part-of-speech tags, and learns to refine syntactic categories. The constraints used by Naseem et al. [80] also take a different form. They provide a set of rules and specify that some percentage of edges p must be an instance of one of these rules. HDP-DEP outperforms GE with the 20 constraints in Table 5.1, giving accuracy of 64.9% vs. 61.3%. We attribute this difference to the more complex and expressive generative model. Recall that part of the motivation for our work was to avoid the development of such models.

5.5 Comparison with Unsupervised Learning

In this section we compare GE training with methods for unsupervised parsing. We use the WSJ10 corpus (as processed by Smith [113]), which is comprised of English

sentences of ten words or fewer (after stripping punctuation) from the WSJ portion of the Penn Treebank. As in previous work sentences contain only part-of-speech tags.

We compare GE and supervised training of an edge-factored CRF with unsupervised learning of a DMV model [53] using EM and contrastive estimation (CE) [114]. We also report the accuracy of an attach-right baseline[2]. Finally, we report the accuracy of a constraint baseline that assigns a score to each possible edge that is the sum of the target expectations for all constraints on that edge. Possible edges without constraints receive a score of 0. These scores are used as input to the maximum spanning tree algorithm, which returns the best tree. Note that this is a strong baseline because it can handle uncertain constraints, and the tree constraint imposed by the MST algorithm helps information propagate across edges.

We note that there are considerable differences between the DMV and CRF models. The DMV model is more expressive than the CRF because it can model the arity of a head as well as sibling relationships. Because these features consider multiple edges, including them in the CRF model would make exact inference intractable [76]. However, the CRF may consider the distance between head and child, whereas DMV does not model distance. The CRF also models non-projective trees, which when evaluating on English is likely a disadvantage.

Consequently, we experiment with two sets of features for the CRF model. The first, *restricted* set includes features that consider the head and child tags of the dependency conjoined with the direction of the attachment, (*parent-POS,child-POS,direction*). With this feature set, the CRF model is less expressive than DMV. The second *full* set includes standard features for edge-factored dependency parsers [75], though still unlexicalized. The CRF cannot consider valency even with the *full* feature set, but this is balanced by the ability to use distance.

[2]The reported accuracies with the DMV model and the attach-right baseline are taken from [113].

feature	ex.	feature	ex.
MD → VB	1.00	NNS ← **VBD**	0.75
POS ← **NN**	0.75	PRP ← **VBD**	0.75
JJ ← **NNS**	0.75	**VBD** → TO	1.00
NNP ← **POS**	0.75	**VBD** → VBN	0.75
ROOT → MD	0.75	NNS ← **VBP**	0.75
ROOT → VBD	1.00	PRP ← **VBP**	0.75
ROOT → VBP	0.75	**VBP** → VBN	0.75
ROOT → VBZ	0.75	PRP ← **VBZ**	0.75
TO → VB	1.00	NN ← **VBZ**	0.75
VBN → IN	0.75	**VBZ** → VBN	0.75

Table 5.1: 20 constraints that give 61.3% accuracy on WSJ10. Tags are grouped according to heads, and are in the order they appear in the sentence, with the arrow pointing from head to modifier.

We generate constraints in two ways. First, we use constraints of the form (*parent-POS,child-POS,direction*) such that $c_q \geq 200$. We choose constraints in descending order of $p(e|q)$. The first 20 constraints selected using this method are displayed in Table 5.1.

Although the reader can verify that the constraints in Table 5.1 are reasonable, we additionally experiment with human-provided constraints. We use the prototype phrase-structure constraints provided by Haghighi and Klein [44], and with the aid of head-finding rules, extract 14 (*parent-pos,child-pos,direction*) constraints.[3] We then estimated target expectations for these constraints using our prior knowledge, without looking at the training data. We also created a second constraint set with an additional six constraints for tag pairs that were previously underrepresented.

5.5.1 Results

We present results varying the number of constraints in Figures 5.1 and 5.2. Figure 5.1 compares supervised and GE training of the CRF model, as well as the feature

[3]Because the CFG rules in [44] are "flattened" and in some cases do not generate appropriate dependency constraints, we only used a subset.

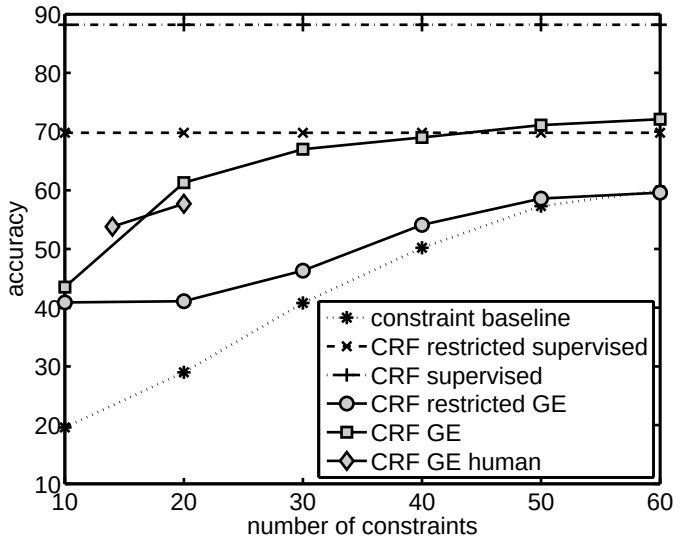


Figure 5.1: Comparison of the baseline and both GE and supervised training of the *restricted* and *full* CRF. Note that supervised training uses 5,301 parsed sentences. GE with human provided constraints closely matches the simulated results.

constraint baseline. First we note that GE training using the *full* feature set substantially outperforms the *restricted* feature set, despite the fact that the same set of constraints is used for both experiments. This result demonstrates GE's ability to learn about related but non-constrained features. GE training also outperforms the baseline[4].

We compare GE training of the CRF model with unsupervised learning of the DMV model in Figure 5.2[5]. Despite the fact that the *restricted* CRF is less expressive than DMV, GE training of this model outperforms EM with 30 constraints and CE with 50 constraints. GE training of the *full* CRF outperforms EM with 10 constraints and CE with 20 constraints (those displayed in Table 5.1). GE training of the *full*

[4]The baseline eventually matches the accuracy of the restricted CRF but this is understandable because GE's ability to bootstrap is greatly reduced with the restricted feature set.

[5]Klein and Manning [53] report 43.2% accuracy for DMV with EM on WSJ10. When jointly modeling constituency and dependencies, Klein and Manning [53] report accuracy of 47.5%. Seginer [102] and Bod [12] propose unsupervised phrase structure parsing methods that give better unlabeled F-scores than DMV with EM, but they do not report directed dependency accuracy.

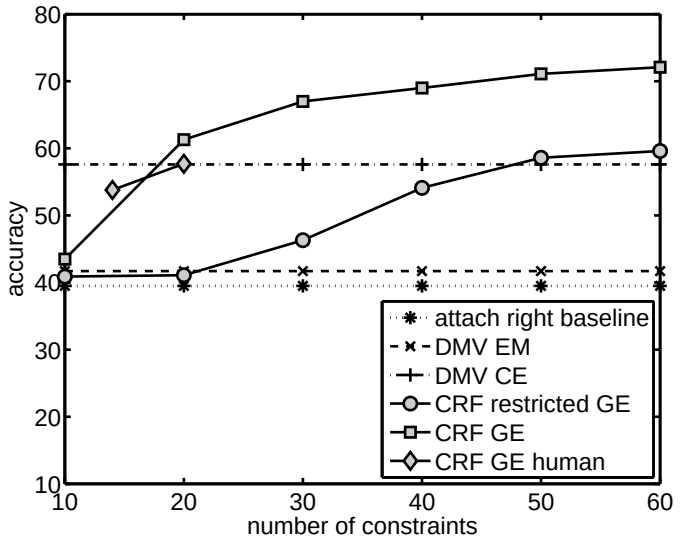


Figure 5.2: Comparison of GE training of the *restricted* and *full* CRFs with unsupervised learning of DMV. GE training of the *full* CRF outperforms CE with just 20 constraints. GE also matches CE with 20 human provided constraints.

CRF with the set of 14 constraints from [44], gives accuracy of 53.8%, which is above the interpolated simulated constraints curve (43.5% accuracy with 10 constraints, 61.3% accuracy with 20 constraints). With the 6 additional constraints, we obtain accuracy of 57.7% and match CE.

Recall that CE, EM, and the DMV model incorporate prior knowledge indirectly, and that the reported results are heavily-tuned ideal cases (see Section 5.4). In contrast, GE provides a method to directly encode intuitive linguistic insights.

Finally, note that structural annealing [115] provides 66.7% accuracy on WSJ10 when choosing the best performing annealing schedule [113]. As noted in Section 5.4 other annealing schedules provide accuracy as low as 32.5%. GE training of the *full* CRF attains accuracy of 67.0% with 30 constraints.

5.6 Experiments on Long Sentences

Unsupervised parsing methods are typically evaluated on short sentences, as in Section 5.5. In this section we show that GE can be used to train parsers for longer sentences that provide comparable accuracy to supervised training with tens to hundreds of parsed sentences. We use the standard train/test splits of the Spanish, Dutch, and Turkish data from the 2006 CoNLL Shared Task. We also use standard edge-factored feature templates [75][6]. We experiment with versions of the data sets in which we remove sentences that are longer than 20 words and 60 words.

For these experiments, we use a simulated user constraint selection method motivated by the linguistic prior knowledge described in Section 5.3.1. The first set of constraints specifies the most frequent head tag, attachment direction, and distance combinations for each child tag. We select constraints of the type (*parent-CPOS,child-CPOS,direction,distance*)[7]. We add constraints for every q such that $e_q > 100$ for max length 60 data sets, and $e_q > 10$ times for max length 20 data sets.

In some cases, the *possible parent* constraints described above will not be enough to provide high accuracy, because they do not consider other tags in the sentence [75]. Consequently, we experiment with adding an additional 25 *sequence* constraints (for what are often called "between" and "surrounding" features). The constraint feature selection method aims to choose such constraints that help to reduce uncertainty in the *possible parents* constraint set. Consequently, we consider sequence features q_s with $p(e|q_s) \geq 0.75$, and whose corresponding (*parent-CPOS,child-CPOS,direction,distance*) constraint q, has edge probability $p(e|q) \leq 0.25$. Among these candidates, we sort by c_{q_s}, and select the top 25.

---

[6]Typical feature processing uses only *supported* features, or those features that occur on at least one true edge in the training data. Because we assume that the data is unlabeled, we instead use features on all possible edges. This generates tens of millions features, so we prune those features that occur fewer than 10 total times, as in [111].

[7]For these experiments we use coarse-grained part-of-speech tags in constraints.

We compare with the constraint baseline described in Section 5.5. Additionally, we report the number of parsed sentences required for supervised CRF training (averaged over 5 random splits) to match the accuracy of GE training using the *possible parents + sequence* constraint set.

The results are provided in Table 5.2. We first observe that GE always outperforms the baseline, especially on parent decisions for which there are no constraints (not reported in Table 5.2, but for example 53.8% vs. 20.5% on Turkish 20). Second, we note that accuracy is always improved by adding *sequence* constraints. Importantly, we observe that GE gives comparable performance to supervised training with tens or hundreds of parsed sentences. These parsed sentences provide a tremendous amount of information to the model, as for example in 20 Spanish length $\leq$ 60 sentences, a total of 1,630,466 features are observed, 330,856 of them unique. In contrast, the constraint-based methods are provided at most a few hundred constraints. When comparing the human costs of parsing sentences and specifying constraints, remember that parsing sentences requires the development of detailed annotation guidelines, which can be extremely time-consuming (see also the discussion is Section 5.4).

	possible parent constraints		+ sequence constraints		complete trees
	baseline	GE	baseline	GE	
dutch 20	69.5	70.7	69.8	**71.8**	80-160
dutch 60	66.5	69.3	66.7	**69.8**	40-80
spanish 20	70.0	73.2	71.2	**75.8**	40-80
spanish 60	62.1	66.2	62.7	**66.9**	20-40
turkish 20	66.3	71.8	67.1	**72.9**	80-160
turkish 60	62.1	65.5	62.3	**66.6**	20-40

Table 5.2: Experiments on Dutch, Spanish, and Turkish with maximum sentence lengths of 20 and 60. Observe that GE outperforms the baseline, adding *sequence* constraints improves accuracy, and accuracy with GE training is comparable to supervised training with tens to hundreds of parsed sentences.

Finally, we experiment with iteratively adding constraints. We sort constraints with $c_q > 50$ by $p(e|q)$, and ensure that 50% are (*parent-CPOS,child-CPOS,direction,distance*)

parent tag	true	predicted
det.	0.005	0.005
adv.	0.018	0.013
conj.	0.012	0.001
pron.	0.011	0.009
verb	0.355	0.405
adj.	0.067	0.075
punc.	0.031	0.013
noun	0.276	0.272
prep.	0.181	0.165

direction	true	predicted
right	0.621	0.598
left	0.339	0.362
distance	true	predicted
1	0.495	0.564
2	0.194	0.206
3	0.066	0.050
4	0.042	0.037
5	0.028	0.031
6-10	0.069	0.033
> 10	0.066	0.039

feature (distance)	false pos. occ.
verb $\rightarrow$ punc. (>10)	1183
noun $\rightarrow$ prep. (1)	1139
adj. $\rightarrow$ prep. (1)	855
verb $\rightarrow$ verb (6-10)	756
verb $\rightarrow$ verb (>10)	569
noun $\leftarrow$ punc. (1)	512
verb $\leftarrow$ punc. (2)	509
prep. $\leftarrow$ punc. (1)	476
verb $\rightarrow$ punc. (4)	427
verb $\rightarrow$ prep. (1)	422

Table 5.3: Error analysis for GE training with *possible parent + sequence* constraints on Spanish 60 data. On the top left, the predicted and true distribution over parent coarse part-of-speech tags. On the top right, the predicted and true distributions over attachment directions and distances. On the bottom, common features on false positive edges.

constraints and 50% are *sequence* constraints. Figure 5.3, which displays results for Spanish 60, shows that GE outperforms the baseline more soundly than above, and that adding constraints continues to increase accuracy.

5.7 Error Analysis

In this section, we analyze the errors of the model learned with the *possible parent + sequence* constraints on the Spanish 60 data. In Table 5.3, we present four types of analysis. First, we present the predicted and true distributions over coarse-grained parent part of speech tags. We can see that verb is being predicted as a parent tag more often then it should be, while most other tags are predicted less often than

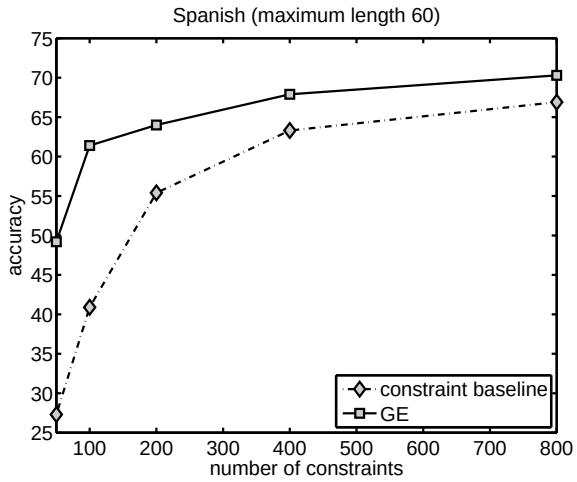


Figure 5.3: Comparing GE training of a CRF and constraint baseline while increasing the number of simulated user constraints.

they should be. Next, we show the predicted and true distributions over attachment direction and distance. From this we see that the model is often incorrectly predicting left attachments, and is predicting too many short attachments. Finally, we show the most common parent-child tag with direction and distance features that occur on false positive edges. From this table, we see that many errors concern the attachments of punctuation. The second line indicates a prepositional phrase attachment ambiguity.

This analysis could also be performed by a linguist by looking at predicted trees for selected sentences. Once errors are identified, GE constraints could be added to address these problems.

5.8 Conclusion and Future Work

In this chapter we developed a method for GE training of CRFs that model distributions over trees. We applied this method to lightly supervised non-projective dependency parsing, leveraging linguistic prior knowledge.

There are several directions for future work.

- User specified target expectations are likely to be imprecise. Additional experiments that aim to compensate for this imprecision appear in Section 8.1.
- GE could also be applied to projective models. The resulting training method would be more efficient (see Section 6.4), and would be suited to applications to English, which is mostly projective.
- Applying GE to models with valence or constituency could mitigate some of the errors observed in Section 5.7.
- Additional experiments could use the *universal rules* of [80]. The *universal rules* have higher coverage than the prior knowledge used in this chapter, which should improve accuracy.

Additional conclusions and discussion are provided in Chapter 12.

CHAPTER 6

GE FOR SEQUENCE LABELING AND TREE-STRUCTURED CRFS

In this chapter we discuss the application of GE to linear chain CRFs. We first review the algorithm of Mann and McCallum [72] for computing the GE gradient when constraint features are zeroth-order, and derive a similar algorithm for first-order constraint features. We show that these algorithms can be made more efficient by using the composite constraint feature, introduced in Section 3.2.1. We then provide empirical results that demonstrate the efficiency advantages of using the composite constraint feature. Finally, we generalize this algorithm to enable efficient GE training in any tree-structured CRF.

6.1 Linear Chain CRFs

In this chapter input and output variables are arranged into iid sequences $\mathbf{x}^i$ and $\mathbf{y}^i$. An nth-order linear-chain CRF [59] models output sequences with an nth-order Markov assumption

$$y^i_{j+n+1} \perp\!\!\!\perp y^i_j \mid y^i_k \ : \ j+1 \leq k \leq j+n. \tag{6.1}$$

We assume a first-order linear-chain CRF for the remainder of this chapter. The probability of a particular output sequence $\mathbf{y}$ for the ith input sequence is

$$p(\mathbf{y}|\mathbf{x}^i;\boldsymbol{\theta}) = \frac{1}{Z(\mathbf{x}^i;\boldsymbol{\theta})} \exp\Big(\sum_{j=1}^{n^i} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^i, y_j, y_{j+1}, j)\Big), \tag{6.2}$$

where n^i is the sequence length. Model features $\mathbf{f}$ consider the entire input sequence and two consecutive output variables. To simply notation, for the remainder of this chapter we drop the sequence index i and the position argument j to $\mathbf{f}$.

Although inference in a linear-chain CRF requires summing over all possible output sequences, dynamic programming can be applied, yielding polynomial time algorithms. The most probable output sequence can be computed with the Viterbi algorithm, and transition marginals can be computed with the forward-backward algorithm [59]. Both algorithms take $O(n|\mathcal{Y}|^2)$ time, where $|\mathcal{Y}|$ is the number of labels. Additional background on linear chain CRFs can obtained in [116].

6.2 GE for Linear Chain CRFs

In this section we derive algorithms for GE training of linear chain CRFs. If constraint features consider one label, or are zeroth-order, then the second term of Equation 3.36 is easy to compute using the forward backward algorithm. As in Chapter 5, the first term of Equation 3.36 is more difficult to compute.

$$\mathbf{u}^{\mathrm{T}}\Big(\sum_{i=1}^{n}\sum_{y_i}\sum_{y_{i+1}}\Big[\sum_{j=1}^{n}\sum_{y_j} p(y_i, y_{i+1}, y_j|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_j)\Big]\mathbf{f}(y_i, y_{i+1}, \mathbf{x})^{\mathrm{T}}\Big) \tag{6.3}$$

We first review the algorithm proposed by Mann and McCallum [72] for computing the bracketed quantity in Equation 6.3.

6.2.1 Gradient Computation for Zeroth-Order Constraint Features

The original algorithm of Mann and McCallum [72] for computing the bracketed quantity in Equation 6.3 performs the computation once for each constraint feature. We describe an equivalent version that performs the computation for all constraint features simultaneously, storing a vector in each cell of the dynamic programming table. This version is more amenable to the improvement in Section 6.2.3.

We would like to compute

$$\sum_{j=1}^{n}\sum_{y_j} p(y_i, y_{i+1}, y_j|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_j)$$
$$= \sum_{j=1}^{i}\sum_{y_j} p(y_i, y_{i+1}, y_j|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_j) + \sum_{j=i+1}^{n}\sum_{y_j} p(y_i, y_{i+1}, y_j|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_j) \quad (6.4)$$

We can rewrite the first term of Equation 6.4 as follows.

$$\sum_{j=1}^{i}\sum_{y_j} p(y_i, y_{i+1}, y_j|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_j)$$
$$= p(y_i, y_{i+1}|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_i) + \sum_{j=1}^{i-1}\sum_{y_j} p(y_i, y_{i+1}, y_j|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_j)$$
$$= p(y_i, y_{i+1}|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_i) + \sum_{y_{i-1}}\sum_{j=1}^{i-1}\sum_{y_j} p(y_{i-1}, y_i, y_{i+1}, y_j|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_j)$$
$$= p(y_i, y_{i+1}|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_i) + p(y_{i+1}|y_i, \mathbf{x};\boldsymbol{\theta})\sum_{y_{i-1}}\Big(\sum_{j=1}^{i-1}\sum_{y_j} p(y_{i-1}, y_i, y_j|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_j)\Big) \quad (6.5)$$

Therefore, we can compute the first term of Equation 6.4 using dynamic programming with the recursion

$$\alpha(y_0, y_1, 0) \equiv p(y_0, y_1|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_0) \quad (6.6)$$
$$\alpha(y_i, y_{i+1}, i) \equiv p(y_i, y_{i+1}|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_i) + p(y_{i+1}|y_i, \mathbf{x};\boldsymbol{\theta})\sum_{y_{i-1}}\alpha(y_{i-1}, y_i, i-1) \quad (6.7)$$

We can compute the second term of Equation 6.4 of similarly.

$$\sum_{j=i+1}^{n} \sum_{y_j} p(y_i, y_{i+1}, y_j|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_j)$$
$$= p(y_i, y_{i+1}|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_{i+1}) + \sum_{j=i+2}^{n} \sum_{y_j} p(y_i, y_{i+1}, y_j|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_j)$$
$$= p(y_i, y_{i+1}|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_{i+1}) + \sum_{y_{i+2}} \sum_{j=i+2}^{n} \sum_{y_j} p(y_i, y_{i+1}, y_{i+2}, y_j|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_j)$$
$$= p(y_i, y_{i+1}|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_{i+1}) + p(y_i|y_{i+1}, \mathbf{x};\boldsymbol{\theta}) \sum_{y_{i+2}} \Big(\sum_{j=i+2}^{n} \sum_{y_j} p(y_{i+1}, y_{i+2}, y_j|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_j) \Big) \tag{6.8}$$

Therefore the "backward" recursion is

$$\beta(y_{n-1}, y_n, n-1) \equiv p(y_{n-1}, y_n|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_n) \tag{6.9}$$
$$\beta(y_i, y_{i+1}, i) \equiv p(y_i, y_{i+1}|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_{i+1}) + p(y_i|y_{i+1}, \mathbf{x};\boldsymbol{\theta}) \sum_{y_{i+2}} \beta(y_{i+1}, y_{i+2}, i+1). \tag{6.10}$$

For any y_i, y_{i+1}, i, Equation 6.4 can be computed

$$\sum_{j=1}^{n} \sum_{y_j} p(y_i, y_{i+1}, y_j|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x}, y_j) = \alpha(y_i, y_{i+1}, i) + \beta(y_i, y_{i+1}, i). \tag{6.11}$$

Note that α and β are dense vectors that store partial expectations of the constraint features. Therefore, the time complexity of the zeroth-order algorithm is $O(d(\boldsymbol{\phi})n|\mathcal{Y}|^2)$, where $d(\boldsymbol{\phi})$ is the dimensionality of $\boldsymbol{\phi}$, or the total number of constraint features. Note that if we have constraints for some set of input features of size c and all labels, $d(\boldsymbol{\phi}) = c|\mathcal{Y}|$, so the algorithm is $O(cn|\mathcal{Y}|^3)$.

6.2.2 Gradient Computation for First-Order Constraint Features

In this section we derive an algorithm for computing the GE gradient when constraint features are first-order $\boldsymbol{\phi}(\mathbf{x}, y_j, y_{j+1})$. In this case we want to compute

$$\sum_{j=1}^{n} \sum_{y_j, y_{j+1}} p(y_i, y_{i+1}, y_j, y_{j+1} | \mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_j, y_{j+1}) \tag{6.12}$$

$$= \sum_{j=1}^{i} \sum_{y_j, y_{j+1}} p(y_i, y_{i+1}, y_j, y_{j+1} | \mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_j, y_{j+1})$$
$$+ \sum_{j=i+1}^{n} \sum_{y_j, y_{j+1}} p(y_i, y_{i+1}, y_j, y_{j+1} | \mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_j, y_{j+1}). \tag{6.13}$$

Proceeding as in the previous section, we can write the first term of Equation 6.13 as

$$\sum_{j=1}^{i} \sum_{y_j, y_{j+1}} p(y_i, y_{i+1}, y_j, y_{j+1} | \mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_j, y_{j+1}) \tag{6.14}$$

$$= p(y_i, y_{i+1} | \mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_i, y_{i+1})$$
$$+ \sum_{j=1}^{i-1} \sum_{y_j, y_{j+1}} p(y_i, y_{i+1}, y_j, y_{j+1} | \mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_j, y_{j+1}) \tag{6.15}$$

$$= p(y_i, y_{i+1} | \mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_i, y_{i+1})$$
$$+ \sum_{y_{i-1}} \sum_{j=1}^{i-1} \sum_{y_j, y_{j+1}} p(y_{i-1}, y_i, y_{i+1}, y_j, y_{j+1} | \mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_j, y_{j+1}) \tag{6.16}$$

$$= p(y_i, y_{i+1} | \mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_i, y_{i+1})$$
$$+ p(y_{i+1} | y_i, \mathbf{x}; \boldsymbol{\theta}) \sum_{y_{i-1}} \sum_{j=1}^{i-1} \sum_{y_j, y_{j+1}} p(y_{i-1}, y_i, y_j, y_{j+1} | \mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_j, y_{j+1}). \tag{6.17}$$

Therefore, we can compute the first term of Equation 6.13 using dynamic programming with the recursion

$$\alpha(y_0, y_1, 0) \equiv p(y_0, y_1 | \mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_0, y_1) \tag{6.18}$$

$$\alpha(y_i, y_{i+1}, i) \equiv p(y_i, y_{i+1} | \mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_i, y_{i+1})$$
$$+ p(y_{i+1} | y_i, \mathbf{x}; \boldsymbol{\theta}) \sum_{y_{i-1}} \alpha(y_{i-1}, y_i, i-1). \tag{6.19}$$

We can write the second term of Equation 6.13 as

$$\sum_{j=i+1}^{n} \sum_{y_j, y_{j+1}} p(y_i, y_{i+1}, y_j, y_{j+1}|\mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_j, y_{j+1}) \tag{6.20}$$

$$= \sum_{y_{i+2}} p(y_i, y_{i+1}, y_{i+2}|\mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_{i+1}, y_{i+2}) + \sum_{j=i+2}^{n} \sum_{y_j, y_{j+1}} p(y_i, y_{i+1}, y_j, y_{j+1}|\mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_j, y_{j+1}) \tag{6.21}$$

$$= p(y_i|y_{i+1}, \mathbf{x}; \boldsymbol{\theta}) \sum_{y_{i+2}} p(y_{i+1}, y_{i+2}|\mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_{i+1}, y_{i+2}) + \sum_{y_{i+2}} \sum_{j=i+2}^{n} \sum_{y_j, y_{j+1}} p(y_i, y_{i+1}, y_{i+2}, y_j, y_{j+1}|\mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_j, y_{j+1}) \tag{6.22}$$

$$= p(y_i|y_{i+1}, \mathbf{x}; \boldsymbol{\theta}) \sum_{y_{i+2}} p(y_{i+1}, y_{i+2}|\mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_{i+1}, y_{i+2}) + p(y_i|y_{i+1}, \mathbf{x}; \boldsymbol{\theta}) \sum_{y_{i+2}} \sum_{j=i+2}^{n} \sum_{y_j, y_{j+1}} p(y_{i+1}, y_{i+2}, y_j, y_{j+1}|\mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_j, y_{j+1}). \tag{6.23}$$

Therefore the "backward" recursion is

$$\beta(y_{n-1}, y_n, n-1) \equiv 0 \tag{6.24}$$

$$\beta(y_{n-2}, y_{n-1}, n-2) \equiv p(y_{n-2}|y_{n-1}, \mathbf{x}; \boldsymbol{\theta}) \sum_{y_n} p(y_{n-1}, y_n|\mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_{n-1}, y_n) \tag{6.25}$$

$$\beta(y_i, y_{i+1}, i) \equiv p(y_i|y_{i+1}, \mathbf{x}; \boldsymbol{\theta}) \sum_{y_{i+2}} p(y_{i+1}, y_{i+2}|\mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_{i+1}, y_{i+2}) + p(y_i|y_{i+1}, \mathbf{x}; \boldsymbol{\theta}) \sum_{y_{i+2}} \beta(y_{i+1}, y_{i+2}, i+1) \tag{6.26}$$

For any y_i, y_{i+1}, i, Equation 6.13 can be computed

$$\sum_{j=1}^{i} \sum_{y_j, y_{j+1}} p(y_i, y_{i+1}, y_j, y_{j+1}|\mathbf{x}; \boldsymbol{\theta}) \boldsymbol{\phi}(\mathbf{x}, y_j, y_{j+1}) = \alpha(y_i, y_{i+1}, i) + \beta(y_i, y_{i+1}, i). \tag{6.27}$$

The time complexity of the first-order algorithm is $O(d(\boldsymbol{\phi})n|\mathcal{Y}|^2)$. Note that if we have constraints for some set of input features of size c and all transitions, $d(\boldsymbol{\phi}) = c|\mathcal{Y}|^2$, so the algorithm is $O(cn|\mathcal{Y}|^4)$.

6.2.3 Improved Algorithm using the Composite Constraint Feature

In Section 3.2.1 we introduced the *composite constraint feature* ϕ'.

$$\phi'(\mathbf{x}, \mathbf{y}) \equiv \mathbf{u}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}) \tag{6.28}$$

Substituting the composite constraint feature into Equation 6.3 yields

$$\sum_{i=1}^{n} \sum_{y_i} \sum_{y_{i+1}} \Big[\sum_{j=1}^{n} \sum_{y_j} p(y_i, y_{i+1}, y_j | \mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, y_j) \Big] \mathbf{f}(y_i, y_{i+1}, \mathbf{x})^{\mathrm{T}}. \tag{6.29}$$

We can use the algorithm of Section 6.2.1 to compute Equation 6.29. Note that using the composite constraint feature dramatically reduces the time complexity of this algorithm. Previously, a dense vector of dimensionality $d(\boldsymbol{\phi})$ was stored in each cell of the dynamic programming table. In contrast, with the composite constraint feature we only need to store a scalar in each cell. If we assume that the constraint features are cached, computing the composite constraint feature takes $O(d_s(\boldsymbol{\phi}))$ time, where $d_s(\boldsymbol{\phi})$ is the maximum number of constraint features that are non-zero for a particular $\mathbf{x}$, j, and y_j[1]. Importantly, note that $d_s(\boldsymbol{\phi})$ does not depend on $|\mathcal{Y}|$. Therefore, the total time complexity of the algorithm of Section 6.2.1 is reduced to $O(d_s(\boldsymbol{\phi})n|\mathcal{Y}|^2)$. Additionally, the time complexity of the algorithm of Section 6.2.2, for first-order constraint features, is *also* $O(d_s(\boldsymbol{\phi})n|\mathcal{Y}|^2)$ with the composite constraint feature.

There are several practical implications of this result. In both cases GE training is much more efficient. Increasing the number of constraint features has much

[1]In the worst case $d_s(\boldsymbol{\phi}) = d(\boldsymbol{\phi})$, and using the composite constraint feature does not improve efficiency. However, $d_s(\boldsymbol{\phi}) = d(\boldsymbol{\phi})$ requires every constraint feature to fire for each $\mathbf{x}$, j, and y_j, which is unlikely to happen in practice. In the applications explored in this thesis, typically $d(\boldsymbol{\phi}) >> d_s(\boldsymbol{\phi})$.

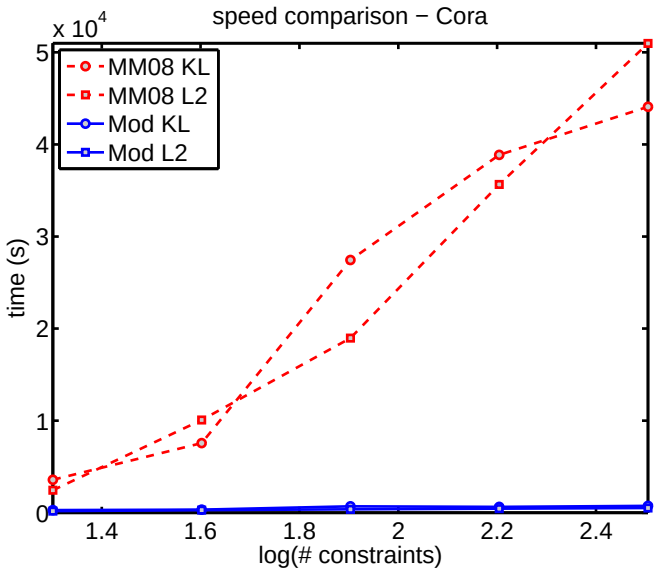


Figure 6.1: Comparison of GE training time (in seconds) for a linear chain CRF with zeroth-order constraints using the algorithm of Section 6.2.1 with (Mod) and without (MM08) the composite constraint feature.

less impact on the running time, as $d_s(\phi)$ typically grows very slowly with $d(\phi)$ in the sequence labeling problems addressed in this thesis. For example, at most one constraint feature can fire at each position when using constraints on label distributions for words. Using first-order constraint features was previously not feasible, but with the improved algorithm it is in theory no less efficient than using zeroth-order constraint features.

6.3 Empirical Efficiency Comparison

We conduct an experiment to compare training times using the algorithm of Section 6.2.1 with and without the composite constraint feature. Recall that previous work did not utilize the composite constraint feature [72].

We use the same 5 80:20 splits of the *Cora references* data set as in Section 9.4, with the same model features. We select $m = \{20, 40, 80, 160, 320\}$ input features that have the highest mutual information with the label at positions where they fire, and

add zeroth-order input feature label distribution constraints with target values set to their true values in the training data. Timing results with both L_2^2 and KL divergence score functions are presented in Figure 6.1. Using the composite constraint feature significantly reduces the training time and the speed with which it grows with m.

6.4 Generalization to Tree-Structured CRFs

The previous section shows that it is possible to compute the GE gradient for linear chain CRFs in $O\big((d_s(\mathbf{f}) + d_s(\boldsymbol{\phi}))n|\mathcal{Y}|^2\big)$ time. In this section we generalize this algorithm. In general, we would like to compute

$$\begin{aligned}
\mathrm{E}_{\boldsymbol{\theta}}[\phi' \mathbf{f}^{\mathrm{T}}] &= \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) \phi'(\mathbf{x},\mathbf{y}) \mathbf{f}(\mathbf{x},\mathbf{y})^{\mathrm{T}} \\
&= \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) \phi'(\mathbf{x},\mathbf{y}) \sum_{a \in \mathcal{F}} \mathbf{f}(\mathbf{x},\mathbf{y}_a)^{\mathrm{T}} \\
&= \sum_{a \in \mathcal{F}} \sum_{\mathbf{y}_a} \mathbf{f}(\mathbf{x},\mathbf{y}_a)^{\mathrm{T}} \sum_{\mathbf{y}_{_a}} p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) \phi'(\mathbf{x},\mathbf{y}),
\end{aligned} \tag{6.30}$$

where $\mathbf{y}_{\_a}$ denotes an assignment to all variables other than those in a. Suppose that the factor graph for $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$ is a tree and that the constraint features factor in the same way as the model features. In this case we can compute Equation 6.30 efficiently using dynamic programming. Suppose that n variables participate in a. We denote assignments to those variables as $y_{a1}, \ldots, y_{an}$. Because the factor graph for $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$ is a tree, we can split $\mathbf{y}_{\_a}$ into assignments to variables that participate in subtrees rooted at each y_{ai}, denoted $\mathbf{y}_{\pi(ai)}$. Note that $\mathbf{y}_{\pi(ai)}$ excludes y_{ai}. We would like to compute the inner sum of Equation 6.30

$$
\begin{aligned}
&\sum_{\mathbf{y}_{\text{-}a}} p(y_{a1}, \ldots, y_{an}, \mathbf{y}_{\pi(a1)}, \ldots, \mathbf{y}_{\pi(an)} | \mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, y_{a1}, \ldots, y_{an}, \mathbf{y}_{\pi(a1)}, \ldots, \mathbf{y}_{\pi(an)}) \\
&\quad = \sum_{\mathbf{y}_{\text{-}a}} p(y_{a1}, \ldots, y_{an}, \mathbf{y}_{\pi(a1)}, \ldots, \mathbf{y}_{\pi(an)} | \mathbf{x}; \boldsymbol{\theta}) \Big(\phi'(\mathbf{x}, \mathbf{y}_a) + \sum_{i=1}^{n} \phi'(\mathbf{x}, y_{ai}, \mathbf{y}_{\pi(ai)}) \Big) \\
&\quad = p(\mathbf{y}_a | \mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, \mathbf{y}_a) + \sum_{i=1}^{n} \sum_{\mathbf{y}_{\pi(ai)}} p(y_{a1}, \ldots, y_{an}, \mathbf{y}_{\pi(ai)} | \mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, y_{ai}, \mathbf{y}_{\pi(ai)}) \\
&\quad = p(\mathbf{y}_a | \mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, \mathbf{y}_a) + \sum_{i=1}^{n} p(\mathbf{y}_{a\text{-}i} | y_{ai}, \mathbf{x}; \boldsymbol{\theta}) \sum_{\mathbf{y}_{\pi(ai)}} p(y_{ai}, \mathbf{y}_{\pi(ai)} | \mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, y_{ai}, \mathbf{y}_{\pi(ai)}),
\end{aligned}
\tag{6.31}
$$

where $\mathbf{y}_{a\text{-}i}$ denotes an assignment to all variables in a except Y_{ai}. The first step follows because we assume ϕ' decomposes in the same way as the model. The second step follows because all variables that appear in subtrees other than i marginalize out. The third step follows because $\mathbf{y}_{a\text{-}i} \perp\!\!\!\perp \mathbf{y}_{\pi(ai)} \mid y_{ai}$.

Equation 6.31 shows that the expectation of ϕ' over all outputs with $\mathbf{Y}_a = \mathbf{y}_a$ can be decomposed into the sum of a term that depends only on the factor marginal $p(\mathbf{y}_a | \mathbf{x}; \boldsymbol{\theta})$, and the sum of the expectations of ϕ' in subtrees rooted at the variables that participate in a. We can compute these expectations with dynamic programming.

Proceeding from Equation 6.31, suppose that we would like to compute the expectation of the subtree rooted at Y_{ai} with $Y_{ai} = y_{ai}$. Specifically, this subtree includes $\mathbf{Y}_a$ and $\{\mathbf{Y}_{\pi(aj)} : 1 \leq j \leq n \wedge j \neq i\}$. Marginalizing over $\mathbf{y}_{a\text{-}i}$, we have

$$
\sum_{\mathbf{y}_{a\text{-}i}} p(\mathbf{y}_a | \mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, \mathbf{y}_a) + \sum_{1 \leq j \leq n : j \neq i}^{n} p(\mathbf{y}_{a\text{-}j} | y_{aj}, \mathbf{x}; \boldsymbol{\theta}) \sum_{\mathbf{y}_{\pi(aj)}} p(y_{aj}, \mathbf{y}_{\pi(aj)} | \mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, y_{aj}, \mathbf{y}_{\pi(aj)}).
\tag{6.32}
$$

It is convenient to specify this procedure as a message passing algorithm. The form of the algorithm is similar to the sum product algorithm [57]. The messages and an expression for computing $\sum_{\mathbf{y}_{\text{-}a}} p(\mathbf{y} | \mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, \mathbf{y})$ are presented in Table 6.1. As with the sum product algorithm, a root is first selected. Then, the algorithm consists

message from variables to factors: $\mu_{Y_{ai} \to a}(y_{ai}) = \sum_{a' \in N(Y_{ai}) \backslash a} \mu_{a' \to Y_{ai}}(y_{ai})$
message from leaf variables to factors (follows from above): $\mu_{Y_{ai} \to a}(y_{ai}) = 0$
message from factors to variables: $\mu_{a \to Y_{ai}}(y_{ai}) = \sum_{\mathbf{y}_a : Y_{ai} = y_{ai}} p(\mathbf{y}_a \vert \mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, \mathbf{y}_a) + \sum_{1 \leq j \leq \vert a \vert : j \neq i} p(\mathbf{y}_{a\_j} \vert y_{aj}, \mathbf{x}; \boldsymbol{\theta}) \mu_{Y_{aj} \to a}(y_{aj})$
message from unary leaf factors to variables (follows from above): $\mu_{a \to Y_{ai}}(y_{ai}) = p(y_{ai} \vert \mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, y_{ai})$
expectation with variables in factor a assigned: $\sum_{\mathbf{y}_{\_a}} p(\mathbf{y} \vert \mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, \mathbf{y}) = p(\mathbf{y}_a \vert \mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, \mathbf{y}_a) + \sum_{j=1}^{\vert a \vert} p(\mathbf{y}_{a\_j} \vert y_{aj}, \mathbf{x}; \boldsymbol{\theta}) \mu_{Y_{aj} \to a}(y_{aj})$

Table 6.1: Messaging passing algorithm to compute $\sum_{\mathbf{y}_{\_a}} p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, \mathbf{y})$. $N(Y)$ denotes the set of factors that take Y as input. $A \backslash a$ denotes the set A with a removed.

of an upward pass, in which messages are passed from the leaves to the root, followed by a downward pass, in which messages are passed from root to the leaves.

Note that the algorithm takes factor marginals as input. It can be viewed as a generalization of the algorithms of Sections 6.2.1 and 6.2.2. After the messages are computed, the gradient can be computed with Equation 6.30 and the expression for $\sum_{\mathbf{y}_{\_a}} p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) \phi'(\mathbf{x}, \mathbf{y})$ in Table 6.1. The time complexity of the complete algorithm is $O((d_s(\mathbf{f}) + d_s(\boldsymbol{\phi}))n|\mathcal{Y}|^T)$, where T is the maximum number of output variables that participate in a factor. Consequently, when the CRF is tree-structured and constraint features factor as model features, GE training has the same time complexity in n and $\mathcal{Y}$ as computing expectations. Note however, that there is a constant factor difference in the run time, as this algorithm requires two passes of dynamic programming.

A generalized algorithm can also be derived using a *hypergraph* representation. Li and Eisner [64] provide efficient algorithms for computing covariances on hyper-

graphs[2]. A weighted hypergraph $HG = (V, E)$ consists of a set of vertices V and a set of weighted hyperedges E. A hyperedge links a set of antecedent nodes to a consequent node. For example, a hypergraph that represents a lattice for a linear chain CRF contains $n|\mathcal{Y}|$ vertices, one for each possible label at each position in the sequence, and $n|\mathcal{Y}|^2$ hyperedges, one for each possible pair of consecutive labels in the sequence. The weight of each edge is its score $\exp(\boldsymbol{\theta} \cdot \mathbf{f}(y_i, y_{i+1}, \mathbf{x}))$. Hypergraphs are convenient for representing all possible derivations for phrase structure parsing.

For models whose inference problems can be represented with a hypergraph, performing inference can be cast as a *semiring* parsing problem. A semiring is an algebraic structure equipped with multiplicative and additive operations and identities. Li and Eisner [64] define first and second-order expectation semirings, and algorithms that use these semirings to compute expectations and covariances on hypergraphs. They then propose a "lifting trick" ([64], Section 4.3) that allows the use of a variant of the *inside outside* algorithm to compute covariances more efficiently.

Any tree-structured factor graph can be encoded as a hypergraph in which there are vertices for each possible assignment to each variable, and hyperedges among all nodes for variables that participate in the same factor a with weight $\exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{y}_a, \mathbf{x}))$.

The general message passing algorithm and the hypergraph algorithm are equivalent for tree-structured models and conceptually related, but there are several small differences. For instance, the message passing algorithm uses marginals, whereas the hypergraph algorithm uses unnormalized scores, and subsequently normalizes.

In Chapter 5 we applied GE to tree CRFs. To ensure a valid tree, the factor graph for this model contains a factor that touches each parent index variable. Consequently, naively using sum product or the message passing algorithm presented in this section would take exponential time in the sentence length n. In Chapter 5 we

[2]Pauls et al. [85] concurrently developed a similar algorithm.

derived an $O(n^4)$ algorithm for computing the gradient that uses matrix inversion. Instead, we could use symbolic differentiation in reverse mode [43], as known as back-propagation, to compute the gradient in $O(n^3)$ time. This would involve applying symbolic differentiation to the determinant computation. Our method may be preferable, however, because it leverages standard, well-studied matrix algorithms that are both fast and numerically accurate. Though it requires a sum over all pairs of edges, pushing the time complexity to $O(n^4)$, only a few arithmetic operations are required for each pair. We leave an empirical comparison of the algorithms to future work.

CHAPTER 7

MCMC GE AND LIGHTLY SUPERVISED ENTITY RESOLUTION

In previous chapters we developed exact algorithms for GE training. In this chapter we approximate the expectations and covariances required for GE training using MCMC. We compare exact and MCMC GE training of linear chain CRFs, and use MCMC GE to train an entity resolution model for which exact inference is intractable. We also explore the use of temperature in MCMC GE training, and illustrate the contrast between MCMC GE training with and without iid instances.

7.1 MCMC GE

Thus far we have applied GE to CRFs for which exact inference is tractable: logistic regression models (Chapter 4), tree CRFs (Chapter 5), and tree-structured CRFs (Chapter 6). We showed that if constraint and model features factor in the same way, exact GE training of these models is tractable.

However, exact inference is intractable for many models of interest, including *entity* or *coreference resolution* models. In coreference resolution the task is to determine the set of *mentions* that refer to the same real-word *entity*. For instance, given citations of research papers, we aim to predict sets of citations that all refer to the same paper. Note that we do not know how many true entities exist in the data.

Additionally, in some settings we may prefer to use constraint features that do not factor in the same way as model features. For example, when extracting information from citations of research papers, we know that at most one segment of each type,

i.e. *title*, should be present in each citation. Incorporating this knowledge into the model directly would make exact inference intractable. Instead, we may enforce this constraint at training time, retaining tractable inference at test time.

To address these cases, we approximate the expectations and covariances required for GE training by sampling outputs from the model $\mathbf{y} \sim p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$. The approximate expectations of the constraint features are

$$\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\boldsymbol{\phi}(\mathbf{x},\mathbf{y})] = \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x},\mathbf{y}) \approx \frac{1}{|S|}\sum_{\mathbf{y}\in S}\boldsymbol{\phi}(\mathbf{x},\mathbf{y}), \tag{7.1}$$

where S is the set of samples. By the law of large numbers, this estimate converges almost surely to the true expectation as $|S| \to \infty$. This approximation can be used in place of the true expectation in any of the score functions described in Section 3.2.

The approximation of the gradient (expanded from Equation 3.39) is

$$\begin{aligned}\frac{\partial}{\partial\boldsymbol{\theta}}\mathrm{S}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]) &= \mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\phi'(\mathbf{x},\mathbf{y})\mathbf{f}(\mathbf{x},\mathbf{y})^{\mathrm{T}}] - \mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\phi'(\mathbf{x},\mathbf{y})]\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[\mathbf{f}(\mathbf{x},\mathbf{y})^{\mathrm{T}}] \\ &\approx \frac{1}{|S|}\sum_{\mathbf{y}\in S}\phi'(\mathbf{x},\mathbf{y})\mathbf{f}(\mathbf{x},\mathbf{y})^{\mathrm{T}} - \frac{1}{|S|}\sum_{\mathbf{y}\in S}\phi'(\mathbf{x},\mathbf{y})\frac{1}{|S|}\sum_{\mathbf{y}\in S}\mathbf{f}(\mathbf{x},\mathbf{y})^{\mathrm{T}}.\end{aligned} \tag{7.2}$$

In this thesis, sampling is performed using Markov chain Monte Carlo (MCMC) methods. We briefly reviewed MCMC methods in Section 2.1.3.2. We discuss the details of specific sampling methods in Sections 7.2 and 7.3.

7.1.1 Temperature

We next discuss the use of a temperature in MCMC GE. As described in Section 3.2.5, a variant of GE training modifies model probabilities with a temperature T. This method is motivated by the fact that a label distribution constraint with target $[0.6, 0.4]$ could be satisfied with parameters $\boldsymbol{\theta}'$ that assign a label distribution of $[0.6, 0.4]$ to each variable. As $T \to 0$, however, the label distribution with $\boldsymbol{\theta}'$ approaches $[1.0, 0.0]$. Consequently a temperature $T < 1$ discourages such solutions.

In MCMC sampling, a temperature can be used to concentrate samples in high probability regions [2]. Specifically, as $T \to 0$, transitions in the Markov chain that reduce the probability of the output variable assignments become less likely.

In MCMC GE, the above notions of temperature are identical. Consequently, there are two potential benefits to using a temperature $T < 1$ in MCMC GE. First, it may increase the accuracy of the trained model. Second, it may reduce the number of samples necessary to obtain an accurate approximation, as only samples around the mode of the distribution are required. However, it is well known that MCMC methods have difficulty transitioning between modes, and this problem is exacerbated when using a temperature $T < 1$, as the probability distribution is more peaked. We next evaluate these issues empirically.

7.2 Comparing MCMC GE to Exact GE

We first compare MCMC GE and exact GE for training linear chain CRFs. If constraint features factor in the same way as model features, exact GE is tractable for this model (see Chapter 6).

We use *Gibbs sampling*, reviewed in Section 2.1.3.2, to sample from $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$. Each *iteration* involves sampling for each output variable, indexed by j, in each sequence, indexed by i, a new assignment y_j^i conditioned on the current assignments to its neighbors, $y_j^i \sim p_T(y_j^i|y_{j-1}^i, y_{j+1}^i, \mathbf{x};\boldsymbol{\theta})$. Output variable assignments are initialized randomly. We begin with 100 "burn-in" iterations, and then collect a sample every 10 iterations. In the following experiments, we vary the number of samples in order to evaluate the tradeoff between efficiency and accuracy. We additionally evaluate the effect of temperature by varying T.

We use two sequence labeling tasks for the following experiments. The *apartments* information extraction data set and its feature representation are described

in Section 9.4. The *CoNLL03* named entity recognition data set and its feature representation are described in Section 8.2.

Input feature label distribution constraints for these experiments are selected automatically using labeled data. We select 100 input features with count ≥ 3 that have the highest mutual information with the label variable at positions where they fire. We estimate target distributions by *binning* the true distributions estimated from labeled data. For example, with 6 bins, each probability is estimated as the closest value in $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$. To compensate for rounding error we then re-normalize to obtain the coarsened target distribution. With 101 bins, the targets are very close to their true values.

Note that we compare the accuracy of models produced by different training methods, rather than the approximation error. In some cases MCMC GE actually provides higher accuracy than exact GE. This is possible because there is not necessarily a direct relationship between the GE objective and the accuracy of the model.

Figure 7.1 displays *apartments* results using 6 and 101 bin constraints, and temperatures $T \in \{0.1, 0.5, 1\}$. In all cases, using 25 samples is sufficient to obtain within 1% test token accuracy of exact GE. With 101 bin constraints and 10 samples, MCMC GE with $T = 0.1$ provides lower accuracy than MCMC GE with $T \in \{0.5, 1\}$. However, this method outperforms all other methods with 25 samples. With 6 bin constraints and $T = 0.1$, using only 10 samples is sufficient to outperform all other methods. These results confirm that using a temperature $T < 1$ can be beneficial.

Figure 7.2 displays *CoNLL03* results. With 6 bin constraints, 25 samples is always sufficient to get within 0.01 segment F_1 of exact GE. With more precise constraints, more samples are required, though for $T \in \{1, 0.5\}$, 100 samples is more than sufficient to get within 0.01 segment F_1 of exact GE. However, we obtain erratic results with MCMC GE with $T = 0.1$ and 101 bin constraints.

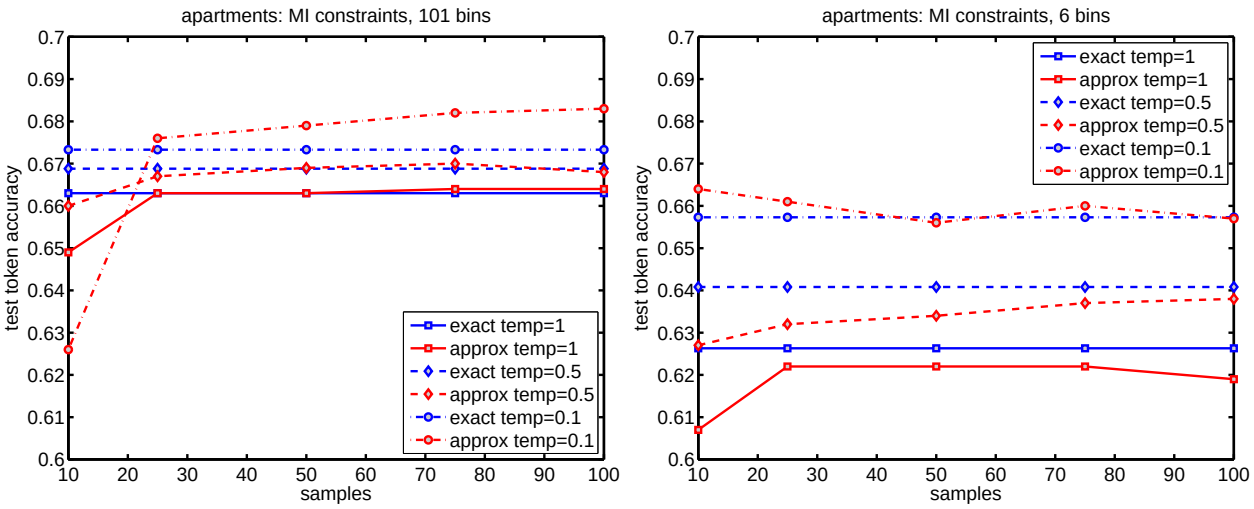


Figure 7.1: *Apartments* experiments using exact and MCMC GE.

Analysis reveals that the inconsistent results obtained with $T = 0.1$ are a result of MCMC being unable to transition between modes. With $T = 0.1$ it is often the case the sampler always yields the same output for a particular sequence. When all sampled outputs for a sequence are identical, the covariance is 0, and hence the gradient is 0 and optimization terminates. When this occurs, however, the true covariance, as computed by exact GE, is non-zero.

In summary, we found that MCMC GE typically only requires a small number of samples to produce an accurate model in this setting. Though using a low temperature is sometimes beneficial, it can also result in erratic performance, suggesting the investigation of *annealing* procedures in which T varies over time. We leave further investigation of temperature in MCMC GE to future work.

In the above experiments the data consists of iid instances. In practice, models where approximate inference is required often have connected factor graphs. We next compare MCMC GE and exact GE in this setting.

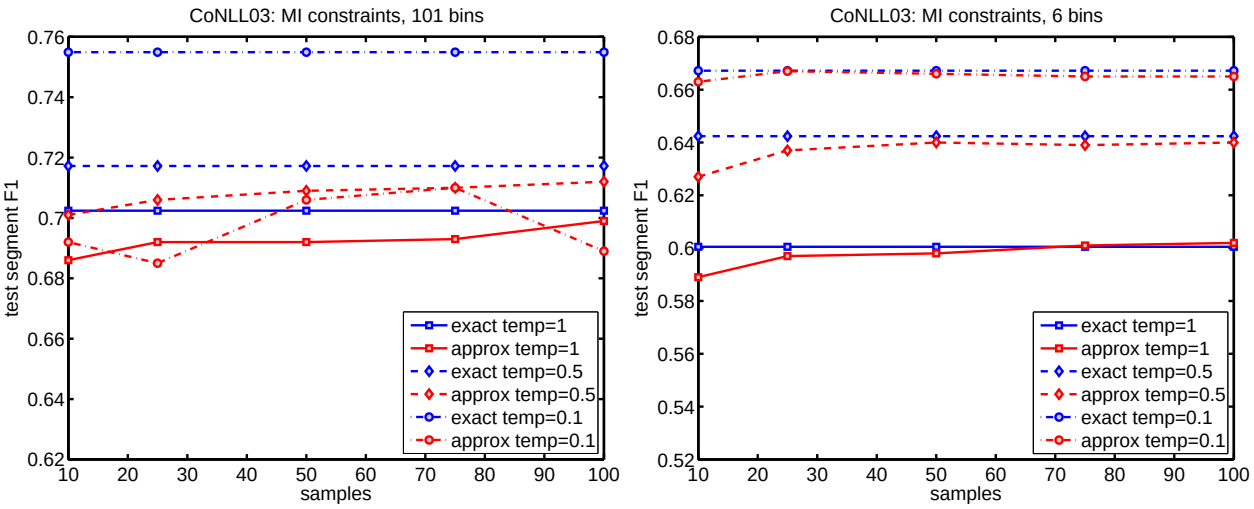


Figure 7.2: *CoNLL03* experiments using exact and MCMC GE.

7.2.1 Comparison using Connected Factor Graphs

The experiments of Section 7.2 suggest that MCMC GE only requires a small number of samples to produce an accurate model. However, the experiments of Section 7.2 are conducted with data that consists of iid instances. Large, fully connected factor graphs are advantageous for many problems of interest. In this section we illustrate additional challenges with MCMC GE when the factor graph is connected.

As described in Section 3.2.2, with iid instances the covariance is the sum of per-instance covariances

$$\sum_i \mathrm{E}_{p(\mathbf{y}^i|\mathbf{x}^i;\theta)}[\boldsymbol{\phi}(\mathbf{x}^i,\mathbf{y}^i)\mathbf{f}(\mathbf{x}^i,\mathbf{y}^i)^{\mathrm{T}}] - \mathrm{E}_{p(\mathbf{y}^i|\mathbf{x}^i;\theta)}[\boldsymbol{\phi}(\mathbf{x}^i,\mathbf{y}^i)]\mathrm{E}_{p(\mathbf{y}^i|\mathbf{x}^i;\theta)}[\mathbf{f}(\mathbf{x}^i,\mathbf{y}^i)^{\mathrm{T}}]. \quad (7.3)$$

Suppose that a constraint feature ϕ and a model feature f never fire in the same instance. Then, by Equation 7.3, their covariance is 0, which means that a constraint on ϕ will have no effect on the parameter for f. This also holds when approximating Equation 7.3, regardless of the particular set of samples S.

However if we add dependencies between instances to the model, it is possible for the covariance of any f and ϕ to be non-zero. That is, we expect the covariance matrix

for the iid setting to be less dense, or to have more zero entries. This suggests that additional samples may be required to accurately estimate covariances in connected models. An alternative perspective on this problem is that when the number of samples is small, it is possible that large, spurious covariances are estimated for pairs of model and constraint features that happen to co-occur unusually often. This problem is mitigated with iid instances, as many covariances must be zero.

To test this hypothesis, we conduct an experiment that is identical to the experiment of Section 7.2, except that we concatenate the *apartments* data into a single sequence by adding a dependency between the final output variable of one sequence and the initial output variable of the next. Figure 7.3 displays the results of this experiment. Note that with 101 bins the exact GE results are very similar to those in Figure 7.1. With 6 bins the exact results are slightly lower but comparable. With a connected model, even using 200 samples is insufficient to get within 1% test token accuracy of exact GE, while with iid instances only 25 samples are required. We expect this problem to be especially pronounced early in training when model probabilities are close to uniform. Consequently, in the experiment in Section 7.3 we attempt to mitigate this problem with parameter initialization.

7.3 Lightly Supervised Entity Resolution

In this section we apply MCMC GE to lightly supervised entity resolution. This is an important application because annotating entity resolution data can be incredibly time-consuming. For example, determining whether two mentions refer to the same entity may require conducting research on the web. Successful unsupervised approaches to the related *noun-phrase coreference* problem rely on complex generative models [46, 89]. In this section we perform entity resolution with a standard discriminative model and two simple GE constraints.

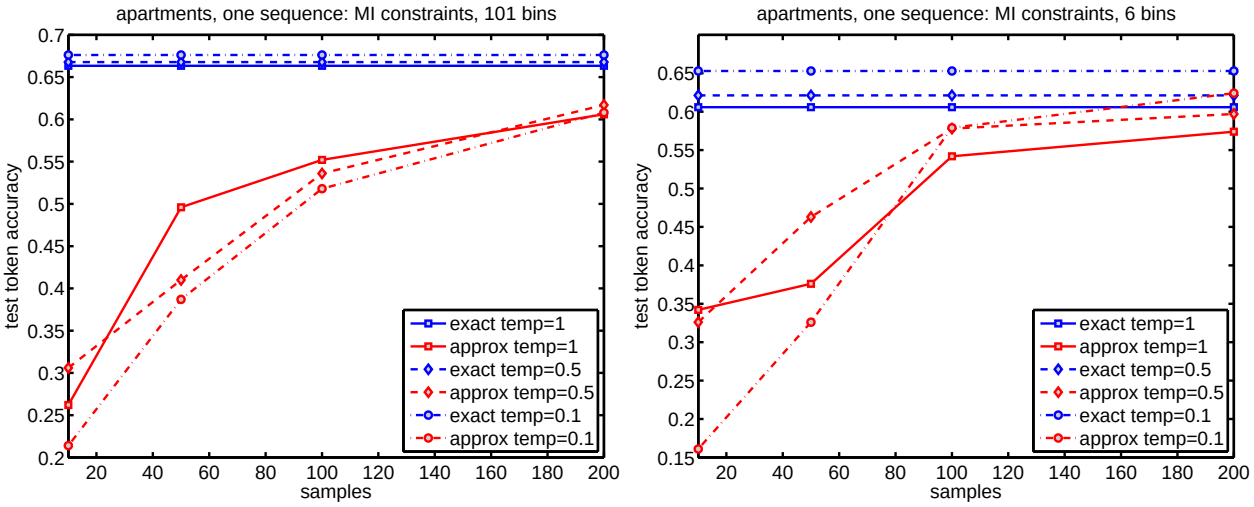


Figure 7.3: *Apartments* experiments using exact and MCMC GE with a connected model.

In this section the task is to predict, given citations of research papers, the sets of citations that refer to the same paper. We use the Cora entity resolution data set[1]. In this data set each citation is segmented into *title*, *author*, and *venue*. We conduct separate experiments on each of the three splits of the data set. There are 377, 471, and 447 mentions, and 43, 40, and 51 true entities in splits 0, 1, and 2.

We model the conditional distribution of entities given mentions with a CRF model. Specifically, the input variables $\mathbf{x}$ correspond to mentions. Each output variable represents an entity. An assignment to output variable i, y_i, is a subset of mentions that refer to entity i. Therefore, the domain of each output variable is the power set of the set of mentions. With n mentions, the size of the domain is 2^n. Therefore, exact inference is intractable for practical entity resolution tasks.

In this experiment model features are defined over pairs of mentions and include various measures of string similarity. We use the *similar title* and *similar venue* features of Poon and Domingos [88], as well as conjunctions of these features. Ad-

[1]Available at http://alchemy.cs.washington.edu/

If the string match is $\geq 80\%$, then the citations are a match 85% of the time.
If the string match is $\leq 20\%$, then the citations are not a match 99% of the time.

Table 7.1: Two simple GE constraints for entity resolution.

ditional features include the count, proportion, discretized count, and discretized proportion of token matches in the two citation strings, as well as separate versions of these features for the author, title, and venue strings. Examples include *the number of matching tokens in the two author strings* and *whether the number of matching tokens in the two venue strings is* > 4. The bins used for count discretization are $\{0, 1, 2, \leq 4, \leq 8, \geq 8\}$ and the bins used for proportion discretization are $\{0, \leq 0.2, \leq 0.4, \leq 0.6, \leq 0.8, \geq 0.8\}$. Finally there are exact string match features, and a default feature that is present for all pairs of mentions. In total there are 68 features.

We use two simple constraints, displayed in Table 7.1. In words, the constraints say that "two citations that have a high string overlap usually refer to the same entity", and "two mentions that have very lower string overlap almost never refer to the same entity." Target distributions are estimated using a combination of domain knowledge and examination of the mentions.

We use an L_2^2 GE score function to incorporate the constraints. Experiments in Section 7.2.1 show that a large number of samples may be required to get accurate covariance estimates when the graphical model is connected. To mitigate this problem, we initialize model parameters using *generalized maximum entropy* (ME) estimation (see Sections 2.1.3.4 and 3.3.2). Specifically, we run five iterations of generalized maximum entropy training with the constraints and an L_2^2 score function. The resulting parameters are used to initialize GE estimation. The motivation for this initialization is to avoid sampling when the model probabilities are close to uniform. We also use generalized maximum entropy as a baseline. For both methods we use Gradient

Ascent with a learning rate of 0.1 for optimization, and per-parameter learning rates as advocated by Lowd and Dominos [69]. Specifically, we divide the gradient for each model feature by its count in the unlabeled data. Both objectives include a zero-mean Gaussian prior on parameters $p(\boldsymbol{\theta})$ with $\sigma^2 = 10$.

We compare with *constraint-driven SampleRank* (CDSR), previously described in Section 3.3.5. Unlike GE and ME, which alternate between sampling and gradient steps, CDSR adjusts the parameters during sampling. Specifically, it adjusts the parameters so that the model gives higher score to output variable assignments that satisfy the constraints. Note that this method cannot incorporate the knowledge that a constraint should hold 85% of the time. We use the objective function in Section 3.3.5 with $\lambda = 10^{10}$. This should result in hard enforcement of the constraints.

Sampling is performed using the Metropolis-Hastings (MH) algorithm, reviewed in Section 2.1.3.2. The MH proposal distribution randomly chooses a mention, and with probability p_{move} proposes moving it to another entity. With probability $1 - p_{move}$, it instead proposes creating a new singleton entity containing the mention. Here $p_{move} = 0.8$. For GE and ME, we use 500 samples, with 1000 jumps between each collected sample. For CDSR, we use a total of 100,000 jumps. To make predictions, we use MH with simulated annealing, so that the final state in the Markov chain is likely to be the maximum probability output. Specifically, 50,000 jumps are taken with each of the following temperatures in sequence $\{1.0, 0.1, 0.01, 0.001, 0.0001\}$.

We evaluate using pairwise precision, recall, and F_1. We say that two mentions *match* in an output $\mathbf{y}$ if they refer to the same entity. Let c be the number of pairs of correctly predicted matches, p be the number of pairs of matches the model predicts, and t be the true number of pairs of matches. Then the pairwise precision is $P = c/p$, the pairwise recall is $R = c/t$, and the pairwise F_1 is the harmonic mean of precision and recall $F_1 = 2PR/(P + R)$. We also evaluate with B^3 precision, recall, and F_1 [3]. This metric is similar to the pairwise metrics, but rewards models for

correctly predicting entities with few mentions. For a mention indexed by i, let c_i be the number of correctly predicted matches with i (including i itself), p_i be the total number of matches the model predicts for i , and t_i be the true number of matches for i. Then the precision and recall for i are defined as above. The final metrics are computed by summing the per mention metrics and normalizing.

7.3.1 Results

Table 7.2 displays the results of the experiment. GE substantially outperforms the ME baseline in all metrics. Recall that maximum entropy estimation results in a model that only has parameters for constraint features. In contrast, GE is able to estimate parameters for the 68 model features described above. The difference in performance between ME and GE demonstrates that this is beneficial.

CDSR outperforms GE in terms of pairwise F_1 on all three data sets. GE outperforms CDSR in terms of B^3 F_1 on two of the three data sets. GE always provides better recall, while CDSR always provides better precision.

It is somewhat surprising that the two methods provide comparable results. Note that CDSR is not capable of explicitly encouraging the model to match target expectations. Instead, prior knowledge must encoded as difficult-to-interpret penalties in the function that compares pairs of samples. Intuitively, one would expect setting $\lambda = 10^{10}$ to result in hard enforcement of the constraints, but this is not what we observe. Note that we have verified that changing the target expectations from 85% and 99% to 100% and 100% for GE training substantially reduces pairwise F_1 and B^3, as expected. We hypothesize that GE would outperform CDSR on other problems where designing an appropriate penalty is more challenging, where the target expectations are less peaked, or where matching the targets precisely is required to obtain high accuracy.

	Pairwise F_1			B^3		
Method (Split)	P	R	F1	P	R	F1
ME (0)	83.46	66.07	73.75	76.51	63.76	69.56
CDSR (0)	**95.85**	98.92	**97.36**	**95.50**	98.08	96.77
GE (0)	94.82	**100.00**	97.34	94.26	**100.00**	**97.05**
ME (1)	80.82	60.69	69.32	81.59	67.68	73.99
CDSR (1)	**99.18**	91.07	**94.95**	**98.08**	92.36	**95.14**
GE (2)	87.46	**99.42**	93.06	88.92	**98.66**	93.54
ME (2)	67.06	60.90	63.83	68.07	61.78	64.78
CDSR (2)	**88.94**	91.89	**90.39**	**91.71**	92.89	92.29
GE (2)	82.48	**99.18**	90.07	86.80	**98.82**	**92.42**

Table 7.2: Entity resolution experiments on the Cora data set.

7.4 Conclusion and Future Work

In this chapter we approximated the expectations and covariances required for GE training using MCMC. We conducted sequence labeling experiments that demonstrate that with iid instances, a small number of samples is sufficient to match the accuracy of exact GE. In the non-iid setting, however, more samples are required to accurately estimate covariances. We then applied MCMC GE to lightly supervised entity resolution, obtaining accurate models with two simple constraints.

Directions for future work include additional comparison of GE and CDSR, as discussed in Section 7.3.1, and investigation of annealing methods and more complex sampling schemes in general, in order to reduce the number of samples MCMC GE requires in the non-iid setting.

Additional conclusions and discussion are provided in Chapter 12.

CHAPTER 8

ADDITIONAL EMPIRICAL ANALYSIS

In previous chapters we demonstrated that it is often possible to train accurate models with minimal human effort using the GE framework. In this chapter we explore directions for improving these results by either compensating for noise in the target expectations, or increasing the precision of the target expectations.

Additionally, we provide an empirical comparison with Posterior Regularization [36].

8.1 Compensating for Noise

In this thesis, input feature label distributions are estimated using procedures that introduce noise. In this section we develop methods to compensate for noise.

As discussed in detail in Chapter 3.2, defining a GE objective function requires selecting a score function. In this thesis score functions assign a lower score when model expectations are "far" from target expectations $\tilde{\phi}$. Consequently we can view score functions as the negative of constraint violation *penalties*. Different penalties can be used to emphasize closing the gap between model and target expectations in different ways. We first propose methods for defining noise-tolerant penalties.

- **target range:** Rather than encouraging the model to match precise target expectations, we may instead encourage the model expectation to be within some *target range*. This can be achieved by specifying a penalty function that is 0 if the model expectation is within the target range. Examples include the *hinge* and L_2^2 *range* score functions introduced in Chapter 3.2.

- **penalty shape:** Another way to address noise is to vary the shape of the penalty. Different shapes can help to compensate for different types of noise. If most target expectations are accurate, but a few target expectations are highly inaccurate, an L_1 penalty is appropriate. An L_1 penalty encourages most constraints to be matched exactly, i.e. to incur zero penalty, but may allow some constraints to be highly violated. In contrast an L_2^2 penalty discourages substantial violation of any constraint, which is more appropriate for constraints with consistent noise or imprecision. The KL divergence penalty highly encourages constraint features with target probability ≥ 0 to have model expectation ≥ 0. See Figure 3.1 to compare penalties visually.

- **compound constraint features:** We also consider noise-tolerant constraint features. Rather than specifying separate constraints with individual target expectations or ranges, it is possible to combine multiple constraint features into a single compound constraint feature.

  In this section we focus on cases in which prior knowledge can be encoded as a noisy rule-based system. Specifically, the rule-based system r takes $\mathbf{x}$ as input and provides, for subsets of output variables $\mathbf{Y}_a$, sets of allowable assignments to $\mathbf{Y}_a$. When the rule-based system has no information about a particular output variable, it returns the empty set $\emptyset$. For example, for dependency parsing, the rule-based system could provide a list of probable parents for each token in the input sentence.

  We define two constraint features using this rule-based system that vary in the way they are normalized. The *rule set precision* constraint feature is defined as

$$\phi_{rp}(\mathbf{x}, \mathbf{y}) = \frac{1}{c_r} \sum_{a \in \mathcal{F}} 1_{\{\mathbf{y}_a \in r(\mathbf{x})_a\}} \tag{8.1}$$

$$c_r = \sum_{a \in \mathcal{F}} 1_{\{r(\mathbf{x})_a \neq \emptyset\}}, \tag{8.2}$$

where $r(\mathbf{x})_a$ is the set of allowable output variable assignments to a. The expectation of $\phi_{rp}(\mathbf{x}, \mathbf{y})$ is the expected probability that the model agrees with the rule-based system r, given that r has a prediction. A similar constraint feature is used in [34].

Alternatively, we define the *rule set recall* constraint feature as

$$\phi_{rr}(\mathbf{x}, \mathbf{y}) = \frac{1}{|\mathcal{F}|} \sum_{a \in \mathcal{F}} 1_{\{\mathbf{y}_a \in r(\mathbf{x})_a\}}. \tag{8.3}$$

The expectation of $\phi_{rr}(\mathbf{x}, \mathbf{y})$ is the expected probability that model predictions are supported by the rule-based system. A similar constraint feature is used in [80].

Compound constraint features allow us to avoid specifying individual target expectations. For example, rather than specifying a target for each dependency parsing rule like JJ $\leftarrow$ NNS, we can instead specify the expected precision or recall of the entire rule set. However, using a compound constraint makes learning more challenging, as there is less specific guidance about how violation of the compound constraint should be addressed.

8.1.1 Document Classification Experiment

We first conduct an experiment using the labeled input features provided by users in Section 4.4.7. The resulting constraints are noisy in two ways: 1) the targets are set using a simple heuristic that assigns probability 0.9 to the label provided by the user, and 2) the users occasionally label input features incorrectly. Table 8.1 displays results using either a *KL* divergence score function, an L_1 score function, or a *Hinge* score function with a penalty of 0 when the model expectation is ≥ 0.9. For L_1 and *Hinge* score functions we use subgradient ascent for optimization, while for *KL* score functions we use L-BFGS. In six of nine cases, using either an L_1 or a *Hinge*

score function outperforms using a *KL* score function. Using a *hinge* score function outperforms using an L_1 score function in five of nine cases.

Table 8.1 also displays statistics of the noise present in each set of constraints. Let the *true expectation* be the value of $\boldsymbol{\phi}$ on labeled data, $\tilde{\boldsymbol{\phi}}_L = \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}_L)$. We define noise as the absolute difference between the true and target expectations, $|\tilde{\boldsymbol{\phi}}_L - \tilde{\boldsymbol{\phi}}|$. Note that the largest improvements with an L_1 or *Hinge* score function are obtained with constraints that have the largest *maximum noise* values.

To develop additional intuition, consider the *baseball-hockey* data set and User 1. Table 8.2 displays label distributions for two input features, *pitching* and *devils*, according to the true distribution, the target distribution, and distributions from logistic regression models estimated using *KL*, L_1, and *Hinge* score functions.

The following example illustrates the benefit of using a target range. Note that although the target probability of *baseball* for *pitching* is 0.9, the true value is 1.0. The model estimated with a *KL* divergence score function matches the target closely. However, the model estimated with a *Hinge* score function, which only encourages the probability of *baseball* to be ≥ 0.9, instead expects 93.1% of documents that contain *pitching* to be labeled *baseball*.

The following example illustrates the benefit of using an L_1-based penalty. Note that the word *devils* is incorrectly labeled by User 1 as *baseball*. While the model estimated with a *KL* score function does not match the target closely, it still expects 54.6% of documents that contain *devils* to be labeled *baseball*. The model estimated with an L_1 score function matches the target much less closely, expecting 76.4% of documents that contain *devils* to be labeled *hockey*.

8.1.2 Dependency Parsing Experiment

We also conduct dependency parsing experiments with the two sets of user provided constraints used in the experiments of Section 5.5. The target distributions for

(user) data set	noise			accuracy		
	mean	**stddev**	**max**	**KL**	**L_1**	**Hinge**
(1) baseball-hockey	0.131	0.190	0.900	86.2	91.2	**92.2**
(1) ibm-mac	0.097	0.107	0.304	**85.4**	84.4	84.4
(1) med-space	0.088	0.072	0.341	**91.6**	90.6	90.6
(2) baseball-hockey	0.160	0.164	0.536	87.4	94.0	**94.4**
(2) ibm-mac	0.262	0.264	0.792	72.3	**78.0**	**78.0**
(2) med-space	0.171	0.185	0.840	84.0	87.4	**89.2**
(3) baseball-hockey	0.072	0.032	0.137	**94.2**	93.6	93.8
(3) ibm-mac	0.071	0.094	0.304	85.0	**85.4**	**85.4**
(3) med-space	0.114	0.099	0.670	89.6	90.4	**91.4**

Table 8.1: Results on 20 Newsgroups subsets using the constraints provided by users in Section 4.4.7 while varying the score function. Using an L_1 or *Hinge* score function can help compensate for noise in the target expectations.

method	**baseball**	**hockey**	**method**	**baseball**	**hockey**
pitching: true	1.000	0.000	*devils*: true	0.000	1.000
pitching: target	0.900	0.100	*devils*: target	0.900	0.100
pitching: GE KL	0.888	0.112	*devils*: GE KL	0.546	0.454
pitching: GE L1	0.922	0.078	*devils*: GE L1	**0.236**	**0.764**
pitching: GE Hinge	**0.931**	**0.069**	*devils*: GE Hinge	0.250	0.750

Table 8.2: True, target, and model distributions for the words *pitching* and *devils* using the *baseball-hockey* data set and the constraints from User 1. Bold indicates the model expectations that are closest to the true distributions.

these constraints are set to a value in $\{0, 0.1, 0.25, 0.5, 0.75, 1\}$ by the user. Consequently, it is likely that the target distributions are imprecise.

Table 8.3 compares L_1 and L_2^2 score functions with both target expectations and target ranges, as well as rule-based constraints. Specifically, target ranges are set to $[\tilde{\boldsymbol{\phi}}{-}\epsilon, \tilde{\boldsymbol{\phi}}{+}\epsilon]$, so that the penalty is zero within ϵ of the original target expectation. The rule-based system provides a set of candidate parents for each child. This candidate set includes any parent such that a constraint feature fires on the resulting edge.

Table 8.4 shows that the mean noise value is generally higher in these experiments than in the experiments of Section 8.1.1. However, relative to the mean, the variance is lower. As a result, an L_1 score function is less appropriate. Empirically an L_1 score

score function	14 constraints	20 constraints
L_2^2	0.537	0.574
L_2^2 range ($\epsilon = 0.1$)	**0.540**	**0.576**
L_2^2 range ($\epsilon = 0.2$)	**0.546**	**0.579**
L_1	0.530	0.568
L_1 range ($\epsilon = 0.1$)	0.543	0.572
L_1 range ($\epsilon = 0.2$)	0.524	0.571
rule set (precision=0.9)	**0.575**	**0.620**
rule set (recall=0.9)	**0.574**	**0.614**

Table 8.3: Dependency parsing accuracy on WSJ10 with user-provided constraints from Section 5.5 using different GE score functions. Bold denotes an improvement over L_2^2, used in Section 5.5. Underline denotes the best performing method overall.

	14 constraints	20 constraints
noise mean	0.260	0.182
noise stddev	0.141	0.170
noise max	0.492	0.723

Table 8.4: Statistics of the noise in the user-provided constraints from Section 5.5.

function decreases parsing accuracy when compared to an L_2^2 score function. Using a target range with an L_2^2 score function provides a small accuracy improvement. Using rule set constraints outperforms all other methods. In this application, the reduction in noise provided by the rule set constraints outweighs the additional ambiguity.

8.1.3 Summary

In this section we showed that it is possible to compensate for noisy target expectations by selecting appropriate score functions and constraint features. However, experiments suggest that the appropriate method depends on the particular application and characteristics of the noise, making it difficult to provide a general recommendation.

8.2 Varying Target Expectation Precision

The input feature label distribution constraints used in this thesis are imprecise. For example, in Chapter 4 the target distributions are automatically generated from labeled input features using simple heuristics. In this section, we investigate the effect of the precision of the target expectations on GE training, with the aim of understanding the extent to which improving precision would improve accuracy.

We focus on input feature label distribution constraints. We use the following procedure to automatically vary the precision of the target distributions. First, we estimate the true target distributions using labeled data. Then, we *bin* each probability by mapping it to the closest value in a fixed set of possible probabilities. We vary the precision by varying the number possible probabilities or bins. For example, with 6 bins, each probability is mapped to the closest value in $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$. To compensate for rounding error we renormalize to obtain a new target distribution.

We conduct experiments on two sequence labeling tasks: CoNLL03 named entity recognition (NER) and Cora citation extraction. Model features for Cora are those used in Section 9.4. The unlabeled data set contains 350 instances, and the test set contains 150. Model features for CoNLL03 include standard regular expressions such as *starts with a capital letter*, character prefixes and suffixes up to length 4, the token at positions in $\{-2, -1, 0, 1, 2\}$, conjunctions of tokens at $\{-1\&0, 0\&1, -1\&1\}$, and the Wikipedia lexicon and Brown cluster features used in [95]. We use the standard training set and evaluate on *testb*.

We use 101, 11, 6, and 3 bins. Note that with 101 bins, the target probabilities are very close to their true values. GE training of a first-order linear chain CRF is performed using a KL divergence score function and a Gaussian prior on parameters with $\sigma^2 = 10$. We add zeroth-order constraints for non-context input features in descending order of *mutual information* with the label at positions where they fire.

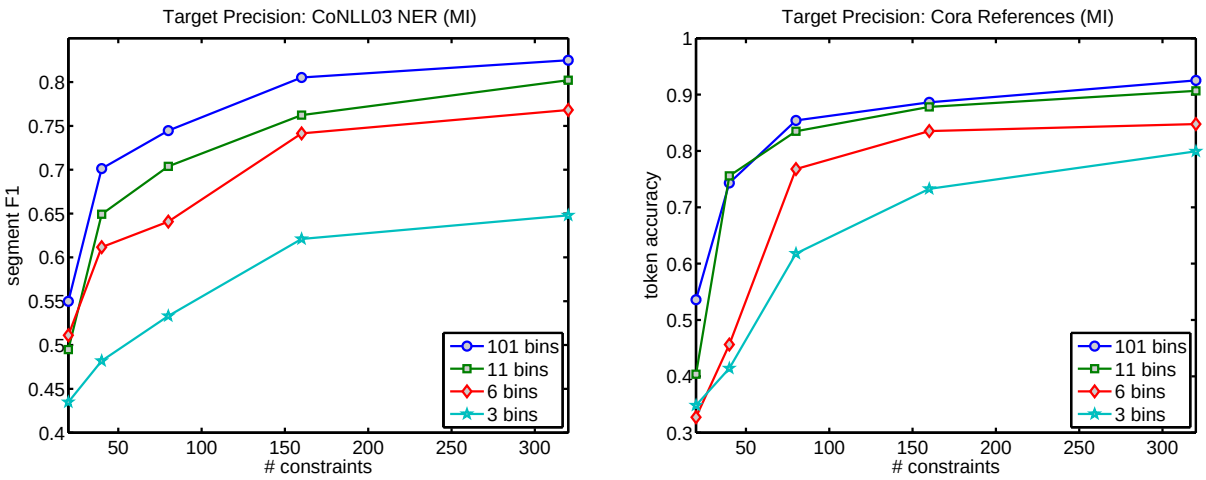


Figure 8.1: More precise constraints yield more accurate CoNLL03 NER.

Figure 8.1 displays learning curves using target distributions with different levels of precision. We observe that increasing the precision of the targets can substantially improve model accuracy. For example, on CoNLL03, 40 constraints using 101 bins yields segment F_1 of 70.1, while 40 constraints using 3 bins yields segment F_1 of 48.9. Note that 40 constraints using 6 bins yields segment F_1 of 61.7, demonstrating that smaller increases in precision can still provide a substantial improvement. Similar trends are evident in the Cora results.

These results suggest that developing methods for obtaining even slightly more precise target expectations is likely to beneficial.

8.3 Empirical Comparison with Posterior Regularization

In this section we compare GE with Posterior Regularization (PR) [36], discussed in Section 3.3.1. We first review mathematical and computational differences.

Recall that PR can be viewed as an approximation to GE. Concretely, given a GE objective function

$$\mathcal{O}_{GE}(\boldsymbol{\theta}) = \mathrm{S}\big(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\big) + \log p(\boldsymbol{\theta}), \tag{8.4}$$

where $\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}] = \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})\boldsymbol{\phi}(\mathbf{x},\mathbf{y})$, a corresponding PR objective function is

$$\mathcal{O}_{PR}(\boldsymbol{\theta}, q) = -\mathrm{D}_{KL}\big(q||p_{\boldsymbol{\theta}}\big) + \mathrm{S}\big(\mathrm{E}_q[\boldsymbol{\phi}]\big) + \log p(\boldsymbol{\theta}) \tag{8.5}$$

$$\mathrm{D}_{KL}\big(q||p_{\boldsymbol{\theta}}\big) = \sum_{\mathbf{y}} q(\mathbf{y}|\mathbf{x}) \log \frac{q(\mathbf{y}|\mathbf{x})}{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}. \tag{8.6}$$

PR encourages q to maximize the score function while encouraging q and $p_{\boldsymbol{\theta}}$ to be close. When $\mathrm{D}_{KL}\big(q||p_{\boldsymbol{\theta}}\big) = 0$, which implies $q(\mathbf{y}|\mathbf{x}) = p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$, the PR objective is identical to the GE objective. However, GE and PR may yield different solutions. A local maximum of the PR objective may have $q(\mathbf{y}|\mathbf{x}) \neq p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$, for example because $\log p(\boldsymbol{\theta})$ discourages equality, or because $p_{\boldsymbol{\theta}}$ cannot represent q. Even if the final PR solution satisfies $q(\mathbf{y}|\mathbf{x}) = p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$, differences in the GE and PR optimization algorithms may result in different parameters $\boldsymbol{\theta}$.

As described in Section 3.3.1, the PR objective can be maximized with a block coordinate ascent algorithm that only requires expectations of model and constraint features, rather than their covariance. Therefore, PR training is tractable if inference in $p_{\boldsymbol{\theta}}$ is tractable and constraint features factor in the same way as model features. Section 6.4 shows that GE training of tree-structured CRFs is tractable. As a result, for tree-structured CRFs the time complexity of both GE and PR is $O\big((d_s(\mathbf{f}) + d_s(\boldsymbol{\phi}))n|\mathcal{Y}|^T\big)$. However, this analysis does not account for the number of iterations of optimization. Note that in PR training, both the E- and M-step require multiple gradient computations. We compare optimization time in the following experiments.

8.3.1 Experimental Setup

tasks: We conduct experiments on sequence labeling tasks. Specifically, we use the *CoNLL03* named entity recognition and *Cora* citation extraction data sets, as described in Section 8.2. The model $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$ is a first-order linear chain CRF.

constraints: We use zeroth-order input feature label distribution constraints. Input features are selected in descending order of *mutual information* with the label

at positions where they fire. As in Section 8.2, we vary the precision of the target distributions by binning the true targets. We use either 101 or 6 bins.

objective functions: For both GE and PR we use an L_2^2 score function and a Gaussian prior on parameters. For PR, this yields L_2^2 regularization of the dual form of q in the modified E-step [5, 36]. The complete objective functions are

$$\mathcal{O}_{GE}(\boldsymbol{\theta}) = -\left\|\tilde{\boldsymbol{\phi}} - \mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}]\right\|_2^2 - \frac{1}{2\sigma^2}\left\|\boldsymbol{\theta}\right\|_2^2 \tag{8.7}$$

$$\mathcal{O}_{PR}(\boldsymbol{\theta}, q) = -\mathrm{D}_{KL}\big(q||p_{\boldsymbol{\theta}}\big) - \alpha\left\|\tilde{\boldsymbol{\phi}} - \mathrm{E}_q[\boldsymbol{\phi}]\right\|_2^2 - \frac{1}{2\sigma^2}\left\|\boldsymbol{\theta}\right\|_2^2, \tag{8.8}$$

where α in the PR objective is a hyperparameter that determines the importance of the score function relative to the KL divergence and regularization terms, and σ^2 is a hyperparameter that controls the strength of the regularization. We additionally compare with generalized maximum entropy, described in Section 3.3.2.

tuning hyperparameters: Labeled data is unavailable in the lightly supervised learning settings explored in this thesis. As a result, standard strategies for tuning hyperparameters such as using a held-out development set or performing cross-validation are not applicable. Elsewhere in this thesis, we use "default" hyperparameter values, for example $\sigma^2 = 1$ for GE training of logistic regression models and $\sigma^2 = 10$ for GE training of structured models. However, insightful selection of σ^2 can improve accuracy. Additionally, we find that careful selection of PR hyperparameters α and σ^2 is sometimes required to obtain an accurate model.

To avoid drawing conclusions that may be based on poor hyperparameter values, we use a development set for tuning. Specifically, we use a 60:20:20 unlabeled/development/test split. Because PR has two hyperparameters to tune whereas GE has one, we allow GE training to consider the same number of possible values of σ^2 as the total number of pairs of hyperparameters (σ^2, α) considered by PR training.

PR optimization: Because we use corpus constraints, we do not have target expectations for individual instances. Consequently, we use batch optimization. We begin L-BFGS optimization with the parameters from the previous round in each step. We consider both optimizing until convergence in each E-step and M-step, and a variant in which we perform one iteration of optimization in each step. Note that an iteration involves line optimization, which requires multiple gradient computations. The initial step size for line optimization in an E- or M-step is the first accepted step size from the previous E- or M-step.

stopping criteria: For PR training we run 250 rounds, where a round is an E-step followed by an M-step. For PR training with one iteration of optimization per step we run 1000 rounds. For GE and ME we use standard stopping criteria.

8.3.2 Results

Figures 8.2 and 8.3 display results for *Cora* and *CoNLL03* with different constraint sets. Each plot compares GE, PR, PR with one iteration of optimization per step (PR1), and generalized maximum entropy (ME). The x-axis is the number of *passes* through the data. For ME, the number of passes is the total number of gradient computations. For PR, the number of passes is the total number of gradient computations in both the E- and M-steps. For GE, the number of passes is twice the total number of gradient computations, to account for the need to compute constraint feature expectations before computing the covariance (see Algorithm 1).

First, note that both GE and PR substantially outperform ME, demonstrating the benefit of including additional features in $p_{\boldsymbol{\theta}}$ (see Section 3.3.2). Next, note that GE and PR often provide similar final results. This can also be observed in Table 8.5, which displays the token accuracies or segment F_1 values for the final models, as well as the *overlap* of the models trained with GE and PR. The overlap is the percentage

data set	# constr.	# bins	accuracy / F_1				GE overlap	
			ME	PR	PR1	GE	PR	PR1
Cora	20	101	0.341	0.562	0.649	**0.679**	0.759	0.918
	80	101	0.428	0.857	**0.898**	0.893	0.900	0.932
	320	101	0.476	0.928	**0.936**	0.931	0.970	0.963
	20	6	0.342	0.390	0.477	**0.506**	0.804	0.936
	80	6	0.419	0.747	0.759	**0.775**	0.907	0.943
	320	6	0.461	**0.865**	0.850	0.863	0.935	0.956
CoNLL03	20	101	0.275	0.545	0.654	**0.668**	0.941	0.981
	80	101	0.578	0.756	0.777	**0.783**	0.981	0.991
	320	101	0.725	0.834	**0.836**	0.835	0.992	0.992
	20	6	0.270	0.496	0.495	**0.520**	0.960	0.956
	80	6	0.535	**0.655**	0.630	0.645	0.984	0.978
	320	6	0.654	0.754	0.768	**0.769**	0.984	0.994

Table 8.5: Final token accuracy (*Cora*) / segment F_1 (*CoNLL03*) for ME, PR, PR1, and GE, as well as the overlap in the predictions of models trained with GE and PR.

of tokens for which two models make the same prediction. The overlap of models trained with GE and PR is typically over 90%.

However, Figures 8.2 and 8.3 show that PR often requires more passes through the data than GE. For example, in the *Cora* experiment with 20 constraints and 6 bins, GE reaches 50.6% token accuracy after 54 gradient computations or 108 passes. After 5,983 passes, PR1 provides token accuracy of 47.7%. Similar trends are evident in the *CoNLL03* results. We note that GE training also takes much less time, but we prefer to avoid comparisons that may be influenced by implementation details.

Finally, we note that typical hyperparameter values selected for PR training are large values ($\sim$100-10000) for α and small values ($\sim$0.1-1) for σ^2. Using a small σ^2 slows the rate at which information is transferred from q to p. We observe that models trained with PR tend to generalize better when q and p are coupled gradually.

8.3.3 Discussion

In previous work [5] and other unpublished experiments, we often found that GE training provided higher accuracy than PR training. The previous experiments

suggest that the two methods provide similar accuracy when 1) hyperparameters are carefully selected and 2) PR training is permitted to take up to thousands of passes.

There is a growing body of evidence that suggests that direct gradient-based optimization can be preferable to EM. Salakhutdinov et al. [98] show that EM exhibits slow convergence when the data is not well-clustered, and provide a direct gradient-based alternative that converges faster than EM in these situations. Berg-Kirkpatrick et al. [7] show that direct-gradient based optimization outperforms EM for training feature-rich generative models. Recall that PR uses a modified EM algorithm for optimization, while GE uses direct gradient-based optimization, and that PR can be viewed as an approximation to GE. While the aforementioned papers consider generative models and we consider discriminative models, and our objective is not to maximize likelihood but to satisfy constraints, we observe similar trends. Exploring this connection is a direction for future work.

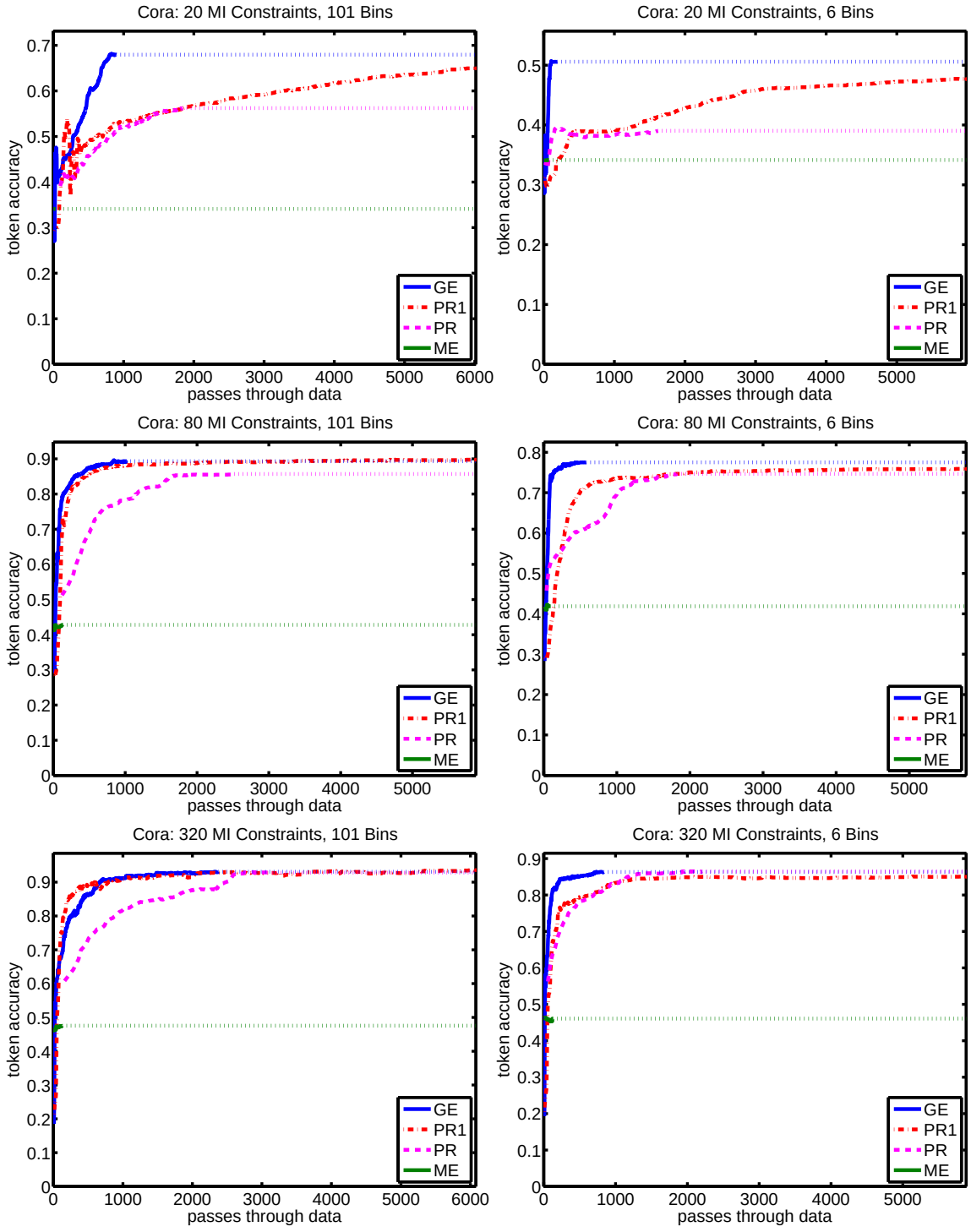


Figure 8.2: Comparison of convergence of GE, PR, and ME on *Cora*.

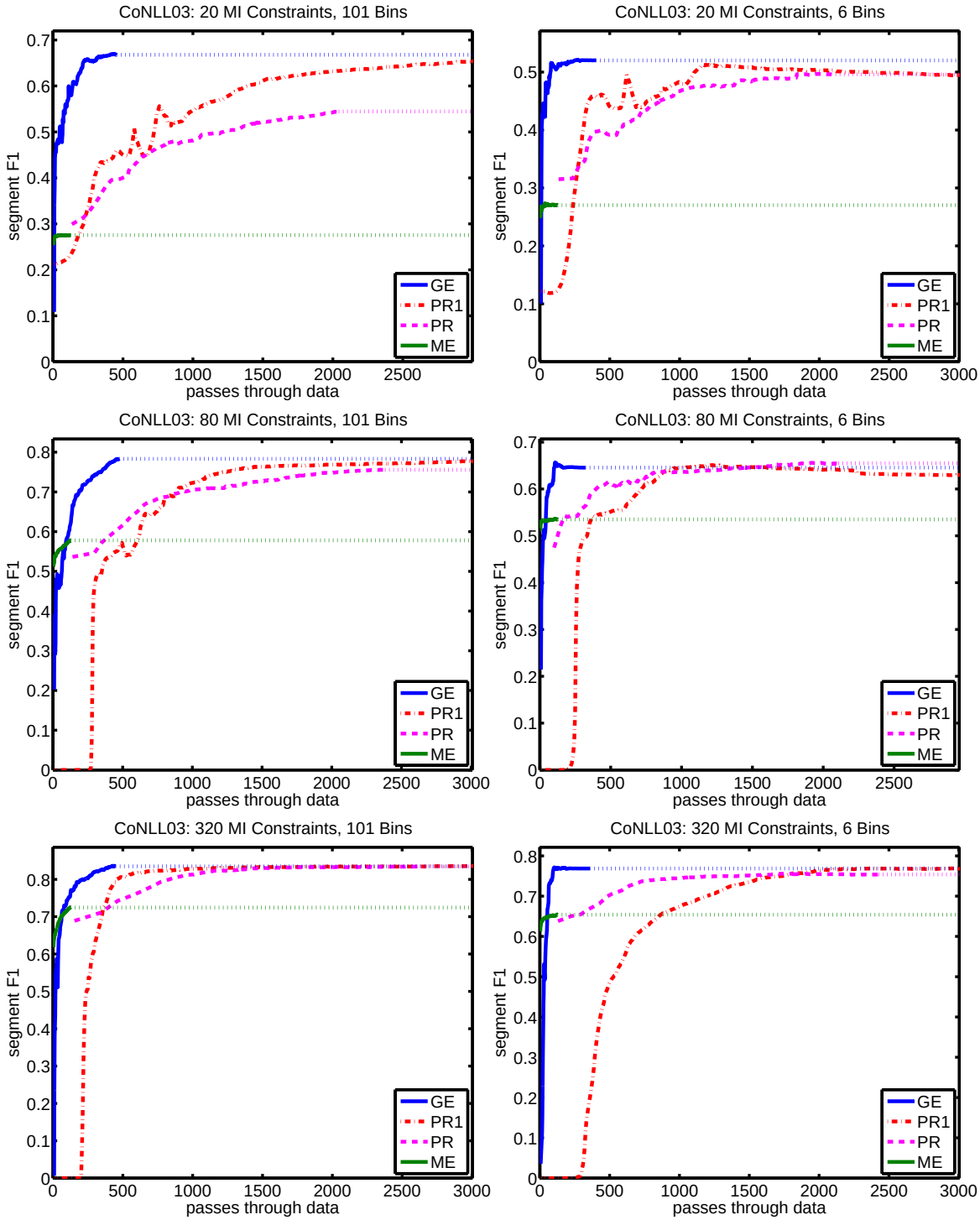


Figure 8.3: Comparison of convergence of GE, PR, and ME on *CoNLL03*.

CHAPTER 9

ACTIVE LEARNING BY LABELING INPUT FEATURES

In the setting investigated in previous chapters, constraints are provided before learning begins. In this chapter, we develop an active learning method that solicits targets for the candidate constraints that are expected to be most informative. We focus on the case in which users provide labels for input features for sequence labeling tasks, and show that the proposed active learning method outperforms passive learning with labeled input features as well as traditional active learning with labeled instances. An example of two iterations of active input feature labeling are presented in Table 9.1.

9.1 Active Learning Background

In traditional active learning, surveyed in [103], the machine selects instances for the user to label, with the goal of prioritizing the labeling of the instances that would most improve the model. In this section we review work in active learning relevant to this thesis.

In *pool-based active learning* [17], the machine selects a query from a pool of candidates, where each query represents an unlabeled instance. Most methods for selecting queries in pool-based active learning aim to reduce model uncertainty. *Uncertainty sampling* is a straightforward active learning method that selects the instance whose predictions are most uncertain [63]. For probabilistic models, the most uncertain instance is the one whose predicted label distribution has the highest entropy. Query

accuracy: 46.5% → 60.5%	
input feature	label
*PHONE**	*contact*
call	*contact*
deposit	*rent*
month	*rent*
pets	*restrictions*
lease	*rent*
appointment	*contact*
parking	*features*
*EMAIL**	*contact*
information	*contact*

accuracy: 60.5% → 67.1%	
input feature	label
water	*utilities*
close	*neighborhood*
garbage	*utilities*
included	*utilities, features*
shopping	*neighborhood*
bart	*neighborhood*
downtown	*neighborhood*
*TIME**	*contact*
bath	*size*

Table 9.1: Two iterations of active input feature labeling. Each table shows the input features labeled, and the resulting change in accuracy. Note that *included* was labeled as *utilities* and *features*, and that $*$ denotes a regular expression feature.

by committee (QBC) methods instead maintain a committee of possible models, and select instances that maximize disagreement among the committee [107].

The previously described methods consider the uncertainty of instances in isolation, though the addition of a new labeled instance will change the uncertainties of other instances as well. Methods that account for this include those that aim to minimize future expected error [97]. However, in general these approaches are computationally intractable, as naively they typically require re-training the model for every possible response to each query.

Most active learning work has focused on classification rather than structured output problems. Active learning was first applied to structured NLP problems by Thompson et al. [117] and Hwa [49]. Settles and Craven [106] provide the first large-scale empirical comparison of active learning methods for sequence labeling tasks. Additionally they suggest augmenting uncertainty-based query selection metrics with a component that depends on the density in the neighborhood of the instance.

Some work has shown that it can be beneficial to combine active and semi-supervised learning by using the pool of unlabeled instances during training [73, 130].

9.2 Active Learning by Labeling Input Features

In this chapter the model is a first-order linear-chain CRF. We consider queries for labels for input features $q_k(\mathbf{x}, i)$. We subsequently convert an input feature with a set of labels L into a target distribution by assigning probability $1/|L|$ for each $l \in L$ and probability 0 for each $l \notin L$[1]. We estimate parameters with a KL divergence GE score function and a Gaussian prior with $\sigma^2 = 10$. To compute the gradient, we use the algorithm described in Chapter 6.

Algorithm 2 presents a *pool-based* active learning algorithm [62] that solicits labels for input features. The novel components of the algorithm are an option to skip a query and the notion that skipping and labeling have different costs. The option to skip is important when using feature queries because a user may not know how to label some input features. In each iteration the model is retrained using the *train* procedure, which takes as input a set of labeled input features $\mathcal{C}$ and unlabeled data $\mathcal{D} = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$. Then, while the iteration cost c is less than the maximum cost c_{max}, the query that maximizes the query selection metric η is selected. The *accept* function determines whether the labeler will label input feature q. If q is labeled, it is added to the set of labeled input features $\mathcal{C}$, and the label cost c_{label} is added to c. Otherwise, the skip cost c_{skip} is added to c. This process continues for m iterations.

9.2.1 Feature query selection methods

In this section we propose feature query selection methods. Queries with higher scores according to scoring functions η are considered more useful. It is important to note these are not *feature selection* methods since we are determining the input features for which supervisory feedback will be most helpful to the model, rather than

[1]In Section 4 we used 0.9 rather than 1. The value 1 was originally selected to improve efficiency [28], but this choice does not change the time complexity when using the more efficient algorithm of Chapter 6. It is possible that better performance could be obtained using a value < 1.

Algorithm 2 Active Learning by Labeling Input Features

Input: unlabeled data set $\mathcal{D} = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$, initial feature constraints $\mathcal{C}$,
label cost c_{label}, skip cost c_{skip}, max cost per iteration c_{max}, max iterations m
Output: model parameters θ
for $i = 1$ **to** m **do**
  $\theta = \text{train}(\mathcal{D}, \mathcal{C})$
  $c = 0$
  while $c < c_{max}$ **do**
    $q = \text{argmax}_{q_k}\, \eta(q_k, \theta, \mathcal{C}, \mathcal{D})$
    if accept(q) **then**
      $\mathcal{C} = \mathcal{C} \cup \text{label}(q)$
      $c = c + c_{label}$
    else
      $c = c + c_{skip}$
    end if
  end while
end for
$\theta = \text{train}(\mathcal{D}, \mathcal{C})$

determining which features will be part of the model. Note that below we use $\eta(q_k)$ as shorthand for $\eta(q_k, \theta, \mathcal{C}, \mathcal{D})$.

We aim to select queries that are expected to provide the largest reduction in model uncertainty. We denote possible responses to a query q_k with $\tilde{\boldsymbol{\phi}}_k$. The **Expected Information Gain (EIG)** of a query is the expectation of the reduction in model uncertainty over all possible responses. Mathematically, EIG is

$$\eta_{EIG}(q_k) = \mathrm{E}_{p(\tilde{\boldsymbol{\phi}}_k|q_k)}\Big[\sum_{i=1}^{N} \mathrm{H}\big(p(\mathbf{y}|\mathbf{x}^i;\boldsymbol{\theta})\big) - \mathrm{H}\big(p(\mathbf{y}|\mathbf{x}^i;\boldsymbol{\theta}_{\tilde{\boldsymbol{\phi}}_k})\big)\Big],$$

where $\boldsymbol{\theta}_{\tilde{\boldsymbol{\phi}}_k}$ are the new model parameters if the response to q_k is $\tilde{\boldsymbol{\phi}}_k$. Unfortunately, this method is computationally intractable. Computing η_{EIG} requires retraining the model for each possible candidate-query-response pair. This is prohibitively expensive for structured output models. Computing the expectation over possible responses is also challenging, as in this thesis users may provide a set of labels for a query, and more generally $\tilde{\boldsymbol{\phi}}_k$ could be a label distribution.

To obtain a tractable strategy, we make the following assumptions: *1) the addition of a constraint for q_k only affects the uncertainty of output variables at positions where*

q_k fires, and *2) the addition of the constraint eliminates any uncertainty about those variables*. With these assumptions, $\eta_{EIG}(q_k)$ becomes the **Total Uncertainty (TU)**, the sum of the marginal entropies at the positions where the input feature fires.

$$\eta_{TU}(q_k) = \sum_{i=1}^{N} \sum_{j=1}^{n} q_k(\mathbf{x}^i, j) \mathrm{H}\big(p(y_j|\mathbf{x}^i;\theta)\big)$$

Note that while these assumptions yield a tractable method, they are often violated. In particular, consider assumption 2. In practice, input features are often skipped, so in many cases a candidate input feature will have no effect on uncertainty if selected. Two types of input features that will often be skipped are frequent input features such as stopwords, and very infrequent input features. We discourage the selection of these input features by down-weighting η_{TU} for frequent input features, while avoiding a bias for infrequent input features. This method, called **weighted uncertainty (WU)**, scales the mean uncertainty by the logarithm of the count.

$$\eta_{WU}(q_k) = \log(c_k) \frac{\eta_{TU}(q_k)}{c_k}.$$

Finally, we also suggest an uncertainty-based metric called **diverse uncertainty (DU)** that encourages diversity among queries by multiplying TU by the mean dissimilarity between the input feature and previously labeled features. For sequence labeling tasks, we can measure the relatedness using distributional similarity

$$\eta_{DU}(q_k) = \eta_{TU}(q_k) \frac{1}{|\mathcal{C}|} \sum_{q_j \in \mathcal{C}} 1 - \mathrm{sim}(q_k, q_j).$$

where $\mathrm{sim}(q_k, q_j)$ returns the cosine similarity between context vectors of words occurring in a window of ± 3.

We contrast the notion of uncertainty described above with another type of uncertainty: the entropy of the predicted label distribution for the input feature, or **expectation uncertainty (EU)**.

$$\eta_{EU}(q_k) = \mathrm{H}(\mathrm{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}_k])$$

Note that $\eta_{EU}(q_k)$ will be large if either the model is uncertain when q_k fires, or the marginal when q_k fires is low entropy, but q_k appears with many different labels. In other words, non-discriminative q_k whose true label distribution is close to uniform will always have large $\eta_{EU}(q_k)$.

The methods described above require the model to be retrained between iterations. To verify that this is necessary, we compare against query selection methods that only consider the previously labeled input features. First, we consider a feature query selection method called **coverage (cov)** that aims to select input features that are dissimilar from existing labeled input features, increasing the labeled input features' "coverage" of the feature space. In order to compensate for choosing very infrequent input features, we scale coverage by the count of the input feature.

$$\eta_{cov}(q_k) = c_k \frac{1}{|\mathcal{C}|} \sum_{q_j \in \mathcal{C}} 1 - \mathrm{sim}(q_k, q_j)$$

Motivated by the feature query selection method of Tandem Learning [93] (see Section 9.3 for further discussion), we consider a feature query selection metric **similarity (sim)** that is the maximum similarity to a labeled input feature, weighted by the occurrence count of the input feature in the corpus.

$$\eta_{sim}(q_k) = c_k \max_{q_j \in \mathcal{C}} \mathrm{sim}(q_k, q_j)$$

Input features similar to those already labeled are likely to be discriminative, and therefore likely to be labeled (rather than skipped). However, insufficient diversity

may also result in an inaccurate model, suggesting that *coverage* should select more useful queries than *similarity*.

Finally, we compare with several passive baselines. **Random (rand)** assigns scores to input features randomly. **Frequency (freq)** scores input features using their frequency in the training data.

$$\eta_{freq}(q_k) = \sum_{i=1}^{N} \sum_{j=1}^{n} q_k(\mathbf{x}, j)$$

Top LDA (LDA) selects the top words from 50 topics learned from the training data using latent Dirichlet allocation (LDA) [9]. More specifically, the words w generated by each topic t are ranked using the conditional probability $p(w|t)$. The word input feature is assigned its maximum rank across all topics.

$$\eta_{LDA}(q_k) = \max_{t} \ \text{rank}_{\text{LDA}}(q_k, t)$$

This method will select useful input features if the topics discovered are relevant to the task. A similar method was previously introduced in Section 4.2.3.

9.3 Related Work

In standard active learning, surveyed in Section 9.1, the learner requests instance labels. This section reviews work in active learning that uses other types of queries.

Tandem Learning [93], previously discussed in Section 4.3, is an active learning algorithm that alternates between querying the user for instance labels and input feature labels. Input feature queries are selected according to their co-occurrence with previously labeled input features and prominent model features. As shown in Section 4.4.6, GE is preferable to the methods Tandem Learning uses to learn with labeled input features. We discuss the potential for mixing feature and instance

queries in Section 9.6. In order to evaluate the feature query selection methodology of Tandem Learning, we propose a method motivated[2] by it in Section 9.2.1. This method, abbreviated **sim**, performs poorly in the experiments in Section 9.4.

In addition to a method for learning with *Measurements*, discussed in Section 3.3.3, Liang et al. [66] propose a method for actively soliciting measurements based on Bayesian experimental design. Selected measurement queries are those that maximize the expected utility over all possible responses to the query. This intractable computation can be simplified by sampling measurement observations, yielding a method similar to the EIG approach proposed here. This method requires computing the expected change in utility for every sample, which is intractable for real world data sets and structured output problems. Liang et al. use this method on synthetic data but switch to simpler uncertainty-based query selection method for experiments on real data [66].

Sindhwani et al. [109] present a method for interleaving instance and input feature label queries. They find that selecting input feature queries using a method closely related to what we call *expectation uncertainty* above performs poorly, as an uncertain feature may be non-discriminative. Instead they choose input feature queries according to model *certainty*. The risk of this approach is that it may only reinforce existing correct predictions, rather than address errors. Sindhwani et al. also propose an approach based on transductive experimental design that selects input feature and instance queries that minimize the variance in test set predictions. In experiments considering only feature queries, certainty and transductive experimental design outperformed random and uncertainty sampling. It is not clear how the training method of Sindhwani et al. would generalize to structured output spaces, and consequently a direct comparison is not possible.

---

[2]The query selection method in Tandem Learning [93] requires a stack that is modified between queries within each iteration. Here query scores are only updated after each iteration of labeling.

In recent work, Settles [104] develops an interface for interactive training in which users can label both instances and features. The interface organizes candidate features into columns, and each column contains features that are predicted to be indicative of a particular label. This method is related to previously described methods that select features that are expected to be predictive [93, 109]. In contrast, in this chapter we use only labeled features, and advocate uncertainty based query selection. Settles [104] does not compare with alternative query selection methods. In this chapter we find that uncertainty based methods outperform others. The interface itself is related to the interface we develop in Section 9.5 in that multiple queries are displayed to the user simultaneously, and feature queries are organized to facilitate labeling.

9.4 Sequence Labeling Experiments

In this section we present experiments with a simulated labeler. When presented an instance query, the labeler simply provides the true labels. When presented a feature query, the labeler first decides whether to skip. We have found that real users are more likely to label input features that are relevant for only a few labels. Therefore, the labeler chooses to label an input feature if the entropy of its label distribution is ≤ 0.7. If accepted, the labeler uses the labeling rule introduced in Section 4.4.1: label the input feature with the maximum probability label, as well as any label whose probability is at least half as large. Examples of labeled input features provided by this method are displayed in Table 9.1.

To estimate the effort of different labeling actions, we perform preliminary experiments in which we observe users labeling data for ten minutes. In these experiments it took an average of 4 seconds to label an input feature, 2 seconds to skip an input feature, and 0.7 seconds to label a token. We setup experiments such that each iteration simulates one minute of labeling by setting $c_{max} = 60$, $c_{skip} = 2$ and $c_{label} = 4$. For instance active learning, we use Algorithm 1 but without the skip option, and

set $c_{label} = 0.7$. We perform $m = 10$ iterations, so the entire experiment simulates 10 minutes of annotation time. For efficiency, we only consider the 500 most frequent unlabeled input features in each iteration. To start, ten randomly selected seed labeled input features are provided.

We use **random (rand)** selection, standard **uncertainty sampling (US)** (using sequence entropy) and **information density (ID)** [106] to select instance queries. When learning with instances we use Entropy Regularization (ER) [50], discussed in Section 2.2.2, to leverage unlabeled instances[3]. We balance the ER term with the data likelihood by choosing the best ER weight in $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$ multiplied by $\frac{\#labeled}{\#unlabeled}$ for each data set and query selection method. Seed instances are provided such that the simulated labeling time is equivalent to labeling 10 input features.

In addition to GE, we also use maximum marginal likelihood training, discussed in Section 2.2.2, for learning with labeled input features. Specifically, we use the labeled input features to obtain a partial labeling of the unlabeled data. When there are multiple choices for a token's label, we randomly sample one.

We evaluate these methods on two sequence labeling tasks. The *apartments* task involves segmenting 300 apartment classified ads into 11 fields including *features*, *rent*, *neighborhood*, and *contact*. We use the same feature processing as [45], with the addition of context features in a window of ± 3. The *Cora references* task is to extract 13 BibTex fields such as *author* and *booktitle* from 500 research paper references. We use a standard set of word, regular expressions, and lexicon features, as well as context features in a window of ± 3. All results are averaged over ten random 80:20 splits of the data.

[3]Results using self-training instead of ER are similar.

9.4.1 Results

Table 9.2 presents mean (across all iterations) and final token accuracy[4] results. On the *apartments* task, GE methods greatly outperform MML[5] and ER methods. Each uncertainty-based GE method also outperforms all passive GE methods. On the *Cora* task, only GE with *weighted uncertainty* significantly outperforms ER and passive GE methods in terms of mean accuracy, but all uncertainty-based GE methods provide higher final accuracy. This suggests that on the *cora* task, active GE methods are performing better in later iterations. Figure 9.1, which compares the learning curves of the best performing methods of each type, shows this phenomenon. Further analysis reveals that the uncertainty-based methods are choosing frequent input features that are more likely to be skipped than those selected randomly in early iterations.

We next compare with the results of related methods. We cannot make claims about statistical significance, but the results illustrate the competitiveness of our method. The 74.6% final accuracy on *apartments* is comparable to any result obtained by Haghighi and Klein [45] (the best is 74.1%), comparable to the *supervised* HMM results reported by Grenager et al. [42] (74.4%), and comparable to the results reported by Mann and McCallum [72] with GE with more accurate sampled label distributions and 10 labeled examples. Chang et al. [15] only obtain better results than 88.2% on *Cora* when using 300 labeled examples (two hours of estimated annotation time), 5000 additional unlabeled examples, and extra test time inference constraints. Note that obtaining these results required only 10 simulated minutes of annotation time, and that the method is provided no information about the label transition matrix.

---

[4]Punctuation tokens are ignored when computing token accuracy.

[5]Only the best performing MML active learning method is shown.

method	**apartments**		**cora**	
	mean	**final**	**mean**	**final**
ER rand	47.9	50.5	74.8	80.4
ER US	50.3	55.6	74.9	82.0
ER ID	50.4	57.2	75.1	82.5
MML rand	47.7	51.2	58.6	64.6
MML WU	57.6	60.8	61.0	66.2
GE rand	59.0	64.8*	77.6*	83.7
GE freq	66.5*	71.6*	68.6	79.8
GE LDA	65.7*	71.4*	74.9	85.0
GE cov	67.5*†	72.4*	70.7	84.1
GE sim	57.8	65.9*	67.1	79.2
GE EU	67.9*†	72.4*	72.9	83.2
GE TU	70.1*†	**73.6***†	76.9	**88.2***†
GE WU	**71.6***†	**74.6***†	**80.3***†	**88.1***†
GE DU	70.5*†	**74.4***†	78.4*	**87.5***†

Table 9.2: Mean and final token accuracy results for all methods. A * denotes that a GE method significantly outperforms all non-GE methods. A † denotes that an active GE method significantly outperforms all passive GE methods. Bold entries denote that the method significantly outperformed all non-bold entries. *Italics* denotes a passive method. Significance is assessed using a paired t-test with significance level $\alpha = 0.05$.

9.5 Sequence Labeling Experiments with Users

Another advantage of feature queries is that input feature names are concise enough to be browsed, rather than considered individually. This allows the design of improved interfaces that can further increase the speed of active learning. We built a prototype interface that allows the user to quickly browse many candidate input features. The input features are split into groups of five input features each. Each group contains input features that are related, as measured by distributional similarity. The input features within each group are sorted according to the active learning metric. This interface, displayed in Figure 9.3, may be useful because input features in the same group are likely to have the same label.

We conduct three types of experiments. First, a user labels instances selected by *information density*, and models are trained using ER. The instance labeling interface

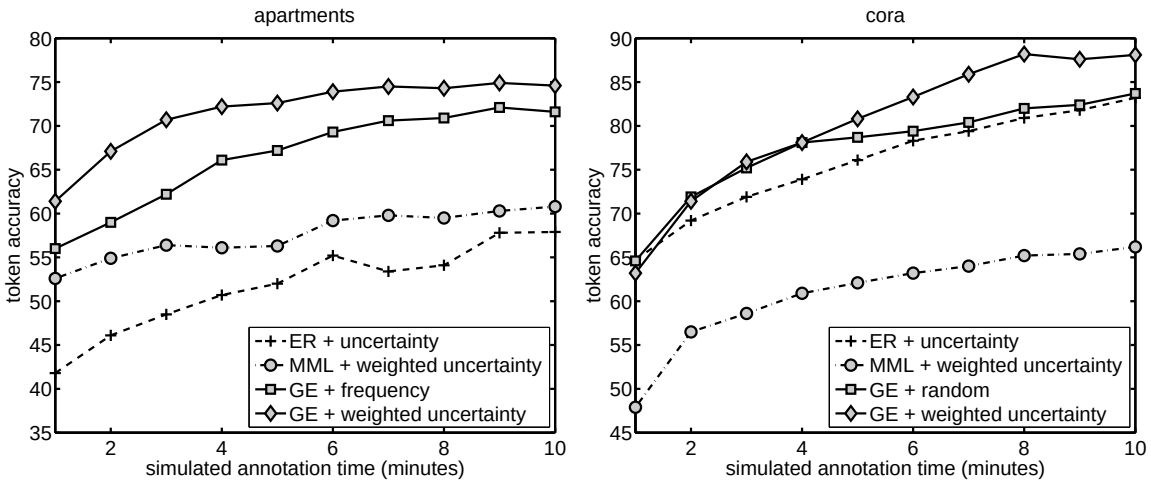


Figure 9.1: Token accuracy vs. time for best performing ER, MML, passive GE, and active GE methods.

allows the user to label tokens quickly by extending the current selection one token at a time and only requiring a single keystroke to label an entire segment. Second, the user labels input features presented one-at-a-time by *weighted uncertainty*, and models are trained using GE. To aid the user in understanding the function of the input feature quickly, we provide several examples of the input feature occurring in context and the model's current predicted label distribution for the input feature. Finally, the user labels input features organized using the grid interface described in the previous paragraph. *Weighted uncertainty* is used to sort feature queries within each group, and GE is used to train models. Each iteration of labeling lasts two minutes, and there are five iterations. Retraining with ER between iterations takes an average of 5 minutes on *cora* and 3 minutes on *apartments*. With GE, the retraining times are on average 6 minutes on *cora* and 4 minutes on *apartments*. Consequently, even when viewed with *total time*, rather than *annotation time*, active learning by labeling input features is beneficial. In future work, users could continue to label features selected according to the old model during retraining.

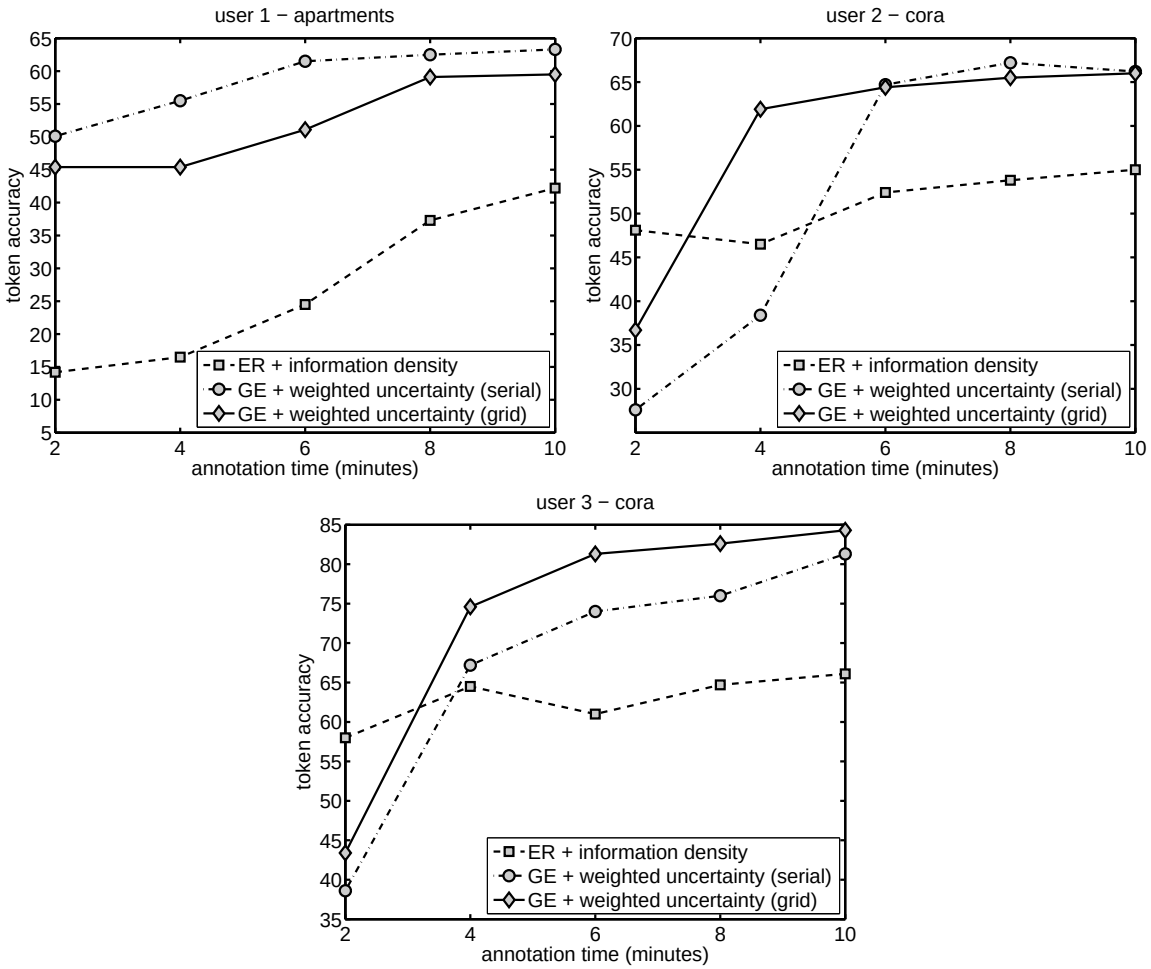


Figure 9.2: User experiments with instance labeling and input feature labeling with the *serial* and *grid* interfaces.

Figure 9.2 displays the results. User 1 labeled *apartments* data, while Users 2 and 3 labeled *cora* data. User 1 was able to obtain much better results with input feature labeling than with instance labeling, but performed slightly worse with the grid interface than with the serial interface. User 1 commented that they found the label definitions for *apartments* to be imprecise, so the other experiments were conducted on the *cora* data. User 2 obtained better results with input feature labeling than instance labeling, and obtained higher mean accuracy with the grid interface. User 3 was much better at labeling input features than instances, and obtained better results with the grid interface.

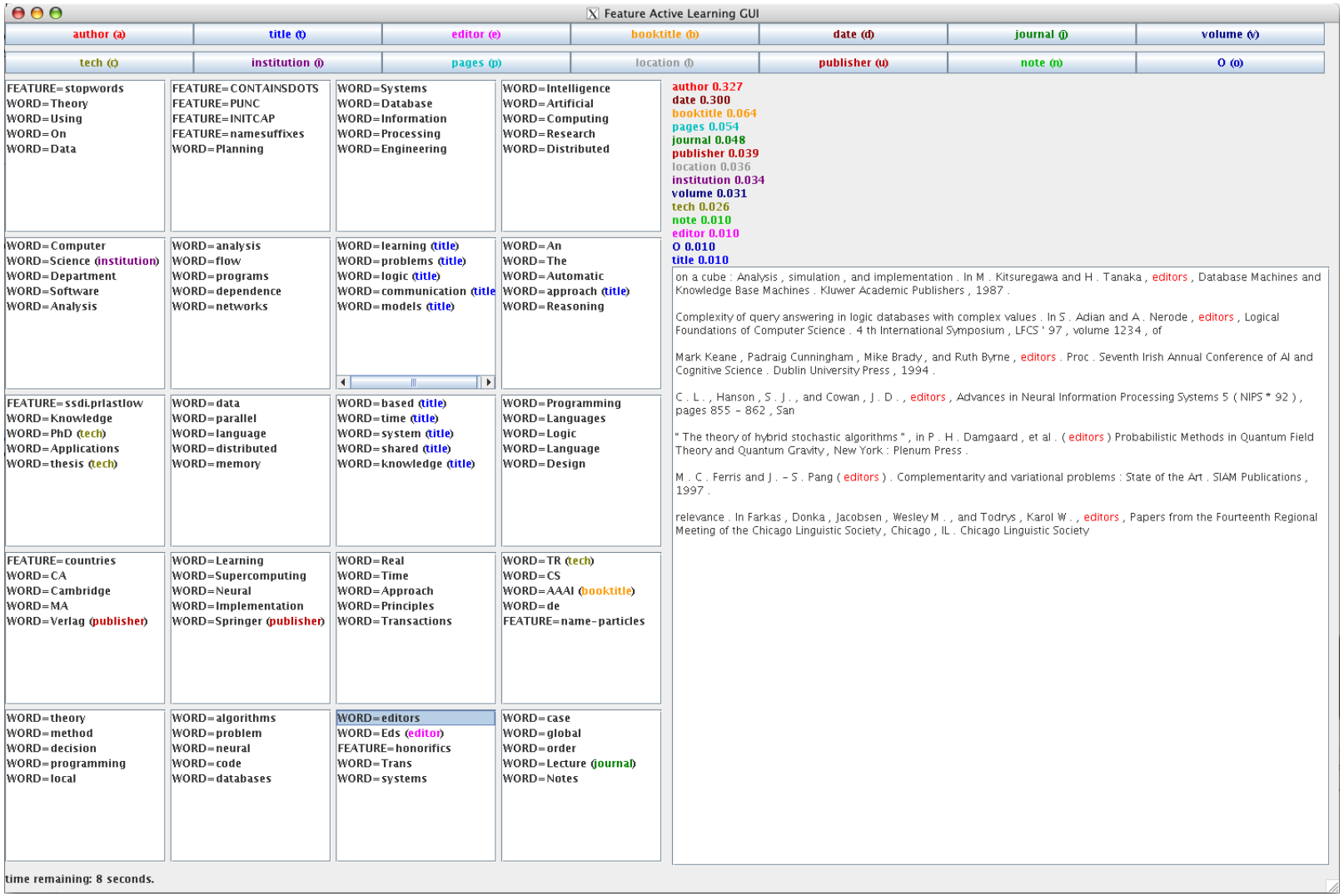


Figure 9.3: Grid input feature labeling interface. Boxes on the left contain groups of input features that appear in similar contexts. Input features in the same group often receive the same label. On the right, the model's current expectation and occurrences of the selected input feature in context are displayed.

9.6 Conclusion and Future Work

We proposed an active learning approach in which input features, rather than instances, are labeled. We presented an algorithm for active learning with input features and several feature query selection methods that approximate the expected reduction in model uncertainty of a feature query. In simulated experiments, active learning with labeled input features outperformed passive learning with labeled input features, and uncertainty-based feature query selection outperformed other baseline methods. In both simulated and real user experiments, active learning with input features outperformed passive and active learning with instances. Finally, we proposed a new labeling interface that leverages the conciseness of input feature queries. User experiments suggested that this grid interface can improve labeling efficiency.

In this chapter we focused on labeling input features. However, the proposed methods generalize to queries for expectation estimates of arbitrary functions, for example queries for the label distributions for input features, labels for instances (using a constraint feature that is non-zero only for a particular instance), partial labels for instances, and class priors. The uncertainty-based query selection methods described in Section 9.2.1 apply naturally to these new query types. Importantly this framework would allow principled mixing of different types of queries, instead of alternating between them as in Tandem Learning [93]. When mixing queries, it will be important to use different costs for different annotation types [119], and estimate the probability of obtaining a useful response to a query. This idea was also proposed by Liang et al. [66], but no experiments with mixed active learning were presented.

Additional conclusions and discussion are provided in Chapter 12.

CHAPTER 10

TOWARD INTERACTIVE TRAINING WITH GE

In the previous chapter we developed a method that actively solicits light supervision. In Section 9.5, we proposed and evaluated a method in which the system suggested a range of possible queries to answer, and the user could choose which action to take.

Building on this work, we envision an interactive training paradigm in which users perform evaluation, analyze errors, and specify and refine supervision in a closed loop. In contrast to active learning [103], in this paradigm the system aids the user in understanding what the model is predicting, and the user leverages this insight to direct the learning process. There are at least two benefits to building this paradigm around methods for learning with expectation constraints. First, expectation constraints provide a flexible and powerful language for users to give feedback to the system. Second, interactive analysis may enable users to specify more accurate and useful constraints than would be possible otherwise.

In this chapter, we focus on several key subproblems in our interactive training paradigm that can be cast as selecting a representative sample of the unlabeled data for the user to inspect. Specifically, we develop methods to assist the user in performing evaluation at multiple levels of granularity, and in specifying and refining constraints. Random sampling can be applied to these problems, but the resulting sample may not be representative. Instead, we develop sampling strategies based on stratified sampling [118], a method that has a long history in statistics [82]. In stratified sampling, points are grouped into strata, and random sampling is performed

within each stratum. If the statistic of interest varies across the strata, then stratified sampling yields a lower variance estimator than random sampling. To perform stratification, we use model expectations as a proxy for latent output variables. In classification and sequence labeling experiments, these sampling strategies reduce accuracy evaluation effort by as much as 53%, provide more reliable estimates of F_1 for rare labels, and aid in the specification and refinement of constraints.

We also briefly describe other problems in interactive training and discuss possible approaches.

10.1 Related Work

In contrast to active learning [103], in which the system queries the user, our interactive training paradigm allows the user to provide supervision based on error analysis, which may enable more efficient training. Our interactive training paradigm and the active learning method developed in Chapter 9.2 could also be combined, allowing a mixture of user-initiated and system-initiated interactions.

There is growing interest in interaction in machine learning, including some work in interactive training and model selection. Huang and Mitchell [48] develop methods for interactive clustering. Users may specify that a feature indicates cluster membership, that an instance is in a cluster, or that a cluster should be deleted, for example.

Roth and Small [96] propose a method that allows the user to interactively modify model features. They focus on modifying and constructing word lists, also known as gazetteers or lexicons, for NLP tasks. Instances selected for interaction are those that are incorrectly predicted and contain word lists that are expected to be imprecise. The user may then modify the word lists or create new word lists to correct the mistakes. Experiments demonstrate that this interaction is beneficial.

Zaidan et al. [127] propose soliciting *rationales*, an annotation that summarizes the user's labeling decision. For example, in text classification, an annotator may mark

a portion of a document as the justification for the label they assigned. Experiments show that users can provide rationales quickly, and that a generative approach for learning with both rationales and labels outperforms learning with labels only.

Culotta et al. [19] propose a method for interactively correcting errors of CRFs. When a user corrects a mistake, the most probable sequences with the corrected label are presented. If the correct labeling is present in that list, the user can choose it directly, avoiding correcting the other mistakes by hand. Candidate mistakes can also be selected for correction according to the expected propagation of corrections.

Fails and Olsen [33] present an interactive training system for pixel classification tasks. The system provides the user with an image, annotated with the current model predictions, and asks the user to correct the mistakes. They stress the need for fast learning algorithms in an interactive training system.

Ware et al. [123] develop a system for interactively building decision tree classifiers that allows users to visualize attribute values and draw polygons to separate different labels. In contrast, our goal is to combine user interaction and machine learning.

Settles [104] develops an interface for interactive training in which users can label both instances and features, previously discussed in Sections 4.3.1 and 9.3.

Kumar et al. [58] argue for an interactive classification framework in which annotation cost and both current and future utility are jointly optimized.

In contrast to the above approaches, the proposed paradigm is based on learning with expectation constraints, and assists the user in evaluation, analysis, and the specification and refinement of constraints.

Note that interactive machine learning often refers to learning directly from interactions with the world, for example based on rewards received for taking context-dependent actions [60]. Though we are interested in exploring this direction in future work, in this chapter learning is user-directed.

This chapter focuses on developing sampling methods for various subproblems in interactive training. These methods are based on stratified sampling, discussed in Section 10.2.1, and use model expectations as a proxy for latent output variables. Stratified sampling using model predictions has also been applied to address specific evaluation problems in information retrieval [126]. More closely related, Bennett and Carvalho [6] develop a confidence-based stratified sampling method for estimating classifier accuracy. In contrast, we apply stratified sampling to rapid evaluation of classifiers and structured models trained using expectation constraints. We propose an extension to the method of Bennett and Carvalho [6] that reduces error in this setting in Section 10.3.1. Additionally, we propose a general approach that is also applicable to other problems in interactive training. Alternatives to stratified sampling include methods based on importance sampling [99, 100]. We plan to consider these approaches in future work.

10.2 Selecting Representative Samples

In this chapter, we aim to develop methods to assist the user in evaluation, error analysis, and the specification and refinement of constraints. We cast these problems as instances of the following more general problem: selecting a sample of the data for the user to inspect. In this section we summarize our *stratified sampling* approach to this problem.

10.2.1 Stratified Sampling

We first review *stratified sampling* [118], a sampling method we use pervasively in this thesis. We assume an iid unlabeled dataset $\mathcal{D} = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$. We aim to compute the mean of a per-instance function $r(\mathbf{x}^j, \mathbf{y}^j)$ that considers both the input variables and the true values of the latent output variables. To simplify notation in this section we define $r^j = r(\mathbf{x}^j, \mathbf{y}^j)$. The true mean is $\bar{r}^* = \frac{1}{N}\sum_{j=1}^{N} r^j$. Because the

output variables $\mathbf{y}^j$ are latent, evaluating r is costly. Rather than evaluating r for each instance, we choose a sample S of size n and use it to estimate the mean of r.

The most basic strategy is *random sampling*, in which n instances are selected uniformly at random from $\mathcal{D}$. The estimate of the population mean is the sample mean $\hat{\bar{r}}_{rs} = \frac{1}{n}\sum_{j=1}^{n} r^j$. This estimator is unbiased, meaning that the expected value of $\hat{\bar{r}}_{rs}$ over all possible samples of size n is $\bar{r}^*$. However, when the sample size n is small, $\hat{\bar{r}}_{rs}$ has high variance. With replacement[1], the estimated variance of $\hat{\bar{r}}_{rs}$ is $\hat{V}ar(\hat{\bar{r}}_{rs}) = \frac{S^2}{n}$, where S^2 is the sample variance.

Stratified sampling has a long history in statistics [82]. In stratified sampling, instances are partitioned into m strata $\{\mathcal{D}_{s1}, \ldots, \mathcal{D}_{sm}\}$, where each $x^j \in \mathcal{D}$ appears in exactly one stratum. To obtain a complete sample of size n, for each stratum i, n_i instances are randomly sampled ($n = \sum_{i=1}^{m} n_i$). An estimate of the mean of r can be obtained using

$$\hat{\bar{r}}_{ss} = \sum_{i=1}^{m} \frac{N_i}{N} \frac{1}{n_i} \sum_{j=1}^{n_i} r_i^j = \sum_{i=1}^{m} W_i \hat{\bar{r}}_i,$$

where r_i^j is r for the jth instance in the ith stratum, $W_i = N_i/N$ is the *weight* of the ith stratum, and $\hat{\bar{r}}_i = \frac{1}{n_i}\sum_{j=1}^{n_i} r_i^j$ is the mean estimate for the ith stratum. This estimator is also unbiased. The estimated variance of $\hat{\bar{r}}_{ss}$ is

$$\hat{V}ar(\hat{\bar{r}}_{ss}) = \sum_{i=1}^{m} W_i^2 \hat{V}ar(\hat{\bar{r}}_i),$$

where $\hat{V}ar(\hat{\bar{r}}_i) = S_i^2/n_i$ is the estimated variance of $\hat{\bar{r}}_i$. A $1-\alpha$ confidence interval for $\hat{\bar{r}}_{ss}$ is $\hat{\bar{r}}_{ss} \pm z_{\alpha/2}\sqrt{\hat{V}ar(\hat{\bar{r}}_{ss})}$.

For a given sample size n, we consider two methods for choosing the number of samples from each stratum. With *proportional allocation*, the sampling proportions

[1]Without replacement $\hat{V}ar(\hat{\bar{r}}_{rs}) = (1 - \frac{n}{N})\frac{S^2}{n}$.

are equal to the weights, $n_i/n = W_i$. It can be shown that the true variance of the proportional allocation estimator is lower than the true variance of the random sampling estimator: $Var(\hat{\bar{r}}_{ss}) \leq Var(\hat{\bar{r}}_{rs})$. The difference of the variances is

$$Var(\hat{\bar{r}}_{rs}) - Var(\hat{\bar{r}}_{ss}) = \frac{1}{n}\sum_{i=1}^{m} W_i(\bar{r}_i^* - \bar{r}^*)^2. \tag{10.1}$$

Equation 10.1 has implications in stratification, as it shows that the variance reduction is larger when strata means $\bar{r}_i^*$ have high variance, or are very different from each other.

The second sample allocation method is *optimal allocation*. In optimal allocation, per-stratum sample sizes n_i are not proportional to the size of the stratum. Instead, the idea is to use more samples in strata where r has higher variance. Formally, suppose that the true standard deviations σ_i for each stratum are known. Optimal allocation assigns samples to each stratum using the following equation

$$n_i = n \times \frac{N_i\sigma_i}{\sum_{i'=1}^{m} N_{i'}\sigma_{i'}}.$$

Optimal allocation can outperform proportional allocation when the variance of r varies across different strata.

Because the r^j are latent, we must select proxy variables to stand in for them to perform stratified sampling. These proxy variables can be used for both stratification and estimating strata variances for optimal allocation.

10.2.2 Estimating Vector-Valued and Non-Linear Functions

In many applications of interest we need to estimate the means of several functions simultaneously, which we view as estimating the mean of a vector-valued function $\mathbf{r}$ instead of a scalar function r. Often a vector-valued function $\mathbf{r}$ is required because we are interested in estimating a *non-linear* function with the sample. A non-linear

function is one that cannot be computed as the mean of a function on individual instances in the sample. For example, accuracy is the mean of a function that evaluates correctness on each instance, but F_1 is non-linear, as computing it requires the number of correct, predicted, and true instances in the entire sample. Vector-valued and non-linear functions have implications in stratification, sample allocation, and estimation.

Stratification & Sample Allocation: As described in Section 10.2.1, for scalar r strata should be selected so that stratum means $\bar{r}_i^*$ are much different than the population mean $\bar{r}^*$. In the vector-valued $\mathbf{r}$ case, stratification is less straightforward. Intuitively, the stratification should be helpful for estimating all elements of $\mathbf{r}$, but note that for some non-linear functions certain estimates r_i may be more important than others. We simply stratify so that a composite variable, correlated with $\mathbf{r}$, varies across strata. Alternative methods for future study include multi-way [13] and clustering-based stratification. We also perform optimal allocation using the estimated standard deviation of a composite variable.

Estimation: A natural estimator of a non-linear function ω is $\hat{\omega} = \omega(\hat{\bar{\boldsymbol{r}}})$ [56]. Note that although $\hat{\bar{\boldsymbol{r}}}$ is an unbiased estimate of $\bar{\mathbf{r}}^*$, $\hat{\omega}$ is not necessarily an unbiased estimate of $\omega(\bar{\mathbf{r}}^*)$. Computing the variance of $\hat{\omega}$ is not straightforward, as we only obtain one value of ω from the sample. A general approach is to use re-sampling methods such as jackknifing, in which each instance is held out of the sample in turn, providing n different function values that can be used to estimate variance [31]. Jackknife estimates for stratified sampling have also been developed [56].

10.2.3 General Stratified Sampling Approach

For each stratified sampling method we propose throughout this thesis, we describe four components: the *target function*, the *stratification function*, the *stratification*

scheme, and the *sample allocation scheme*. We next describe these components and our general approach to designing them.

As described in the previous section, in some cases we are not interested in the mean estimates $\hat{\bar{\boldsymbol{r}}}$, but rather some non-linear function of the estimates $\omega(\hat{\bar{\boldsymbol{r}}})$. We refer to the non-linear function ω as the *target function*. The target function varies in different applications. In the linear case the target function is the identity function.

The *stratification function* s computes a proxy value for each instance that can be used for stratification and sample allocation. Though the output variables $\mathbf{y}$ are latent, if we have a trained model we do have some information about the values of these variables. Our general approach is to use current model predictions as a proxy for latent output variables $\mathbf{y}$. Specifically, a method we use pervasively is to stratify according to the *expectation* of some function of interest r' (often $r' = r$)

$$s(\mathbf{x}^j) = \mathrm{E}_{p(\mathbf{y}|\mathbf{x}^j;\boldsymbol{\theta})}[r'(\mathbf{x}^j, \mathbf{y})] = \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}^j;\boldsymbol{\theta}) r'(\mathbf{x}^j, \mathbf{y}).$$

This approach makes the reasonable assumption that model predictions are correlated with the true output variables. This implies that the improvement provided by stratified sampling is bounded by accuracy of the current model.

When estimating a vector of means $\hat{\bar{\boldsymbol{r}}}$, we take a simple approach and continue to use a scalar stratification function $s(\mathbf{x})$, essentially defining a composite variable. One simple strategy is to use the expectation of a single element of the vector: $r' = r_i$, $s(\mathbf{x}) = \mathrm{E}_{p(\mathbf{y}|\mathbf{x}^j;\boldsymbol{\theta})}[r_i(\mathbf{x}^j, \mathbf{y})]$.

The *stratification scheme* takes as input the values computed by the stratification function and defines the strata. We use two methods for defining strata, which we discuss in detail in the following sections.

Finally, the *sample allocation scheme* defines how samples are allocated to strata. If optimal allocation is used, rather than proportional allocation, then a *stratum variance estimator* that defines a procedure for estimating the variance of strata must

also be defined. We estimate within-stratum variances by using the sample estimates (for instances that have already been inspected), using model probabilities as a proxy for the latent output variables, or a combination. To use model probabilities, we advocate a simple strategy in which values for the latent output variables are sampled according to the model, r is computed, and the resulting sample variance estimates of r stand-in for true variances.

Note that, in general, inspecting individual instances to estimate a statistic could also yield labeled instances as a by-product. In future work we could perform training using both constraints and labeled instances.

10.3 Overall Evaluation

Suppose we have an initial model for the task we would like to solve. Our goal is to interactively improve this model. A natural first step is to come to an understanding of what the model is predicting and how accurate it is, in order to decide where to focus analysis and provide supervision. Importantly, we want to do this with minimal effort. In this section we apply stratified sampling to choose sets of instances (with model predictions) for manual inspection. For concreteness and to allow thorough validation, we focus on the task of using the sample to estimate a performance metric such as accuracy. However, we also suggest that with appropriate adjustments to the sampling scheme, viewing a sample of instances selected by this method would provide accurate representations of other properties of interest.

Note that the methods described here, though motivated by interactive training, could also be used in a non-interactive setting to evaluate lightly supervised learning. This is an important problem on its own, as in a lightly supervised setting there is typically no data available for evaluation.

10.3.1 Evaluating Classification Accuracy

We first estimate classification accuracy. Formally, we aim to estimate the mean of the correctness indicator function

$$r_c(\mathbf{x}, y) = 1_{\{\hat{y}=y\}},$$

where $\hat{y}$ is the predicted label, $\hat{y} = \text{argmax}_y\, p(y|\mathbf{x}; \boldsymbol{\theta})$, and $1_{\{p\}}$ returns 1 if the predicate p is true, and 0 otherwise. When computed with the true label, r_c returns 1 if the model is correct, and 0 otherwise. We next devise several stratified sampling schemes for estimating $\hat{\bar{r}}_c$.

Target Function: Classification accuracy is a linear function, as $\hat{\bar{r}}_c$ is an estimate of accuracy. Consequently the target function ω is simply the identity function $\omega(\hat{\bar{r}}_c) = \hat{\bar{r}}_c$.

Stratification Function: Equation 10.1 suggests that strata should be defined to maximize the variance in individual stratum accuracies. Because the stratum accuracies are unavailable, we instead stratify using the expectation of r_c.

$$\begin{aligned} s_c(\mathbf{x}^j) &= \mathrm{E}_{p(y|\mathbf{x}^j;\boldsymbol{\theta})}[r_c(\mathbf{x}^j, y)] \\ &= \sum_y p(y|\mathbf{x}^j; \boldsymbol{\theta}) 1_{\{\hat{y}=y\}} = p(\hat{y}|\mathbf{x}^j; \boldsymbol{\theta}) \end{aligned} \tag{10.2}$$

Equation 10.2 shows that the model expectation of r_c is the probability of the best label, or the model's *confidence* in its prediction. Note that, using our general approach, we recover the stratification function of Bennett and Carvalho [6]. When applied to other tasks in interactive training our approach yields different stratification functions.

Stratification Scheme: We use a *uniform size stratification scheme*. First, we sort unlabeled instances according to $s_c(\mathbf{x}^j)$. Then, we define strata by splitting the sorted list into m pieces, each containing the same number of instances.

Sample Allocation: Finally, we must allocate samples to the strata. With a uniform size stratification scheme, *proportional allocation* allocates n/m samples to each stratum. We additionally use *optimal allocation*, where the challenge is estimating the standard deviations in each stratum $\hat{\sigma}_i$.

Bennett and Carvalho [6] propose *online stratified sampling*, in which the $\hat{\sigma}_i$ are re-estimated using the observed values r_i^j after each sample. As the number of samples increases, the estimates become more accurate, and savings increase. However, a potential disadvantage of this approach is that many samples may be required to obtain estimates that are accurate enough to be beneficial. Because we are especially interested in evaluation with minimal effort, we propose two additional methods for estimating $\hat{\sigma}_i$ that leverage model predictions.

We can model each unobserved r_i^j in stratum i as a Bernoulli random variable c_i^j with $p_i^j = p(\hat{y}|\mathbf{x}^j;\boldsymbol{\theta})$. Summing all c_i^j for stratum i yields an expected accuracy random variable with a *Poisson Binomial* distribution[2]. We can use the variance of this distribution as an estimate of the stratum variance $\hat{\sigma}_i^2 = \sum_j p_i^j(1-p_i^j)$. This method prioritizes strata where there are expected to be a mix of correct and incorrect predictions over strata with expected accuracy near 0 or 1.

As n increases, we expect *online stratified sampling* to eventually provide better estimates than the above method. Consequently, we propose a novel compromise. In each iteration, we sample a correctness value for each unlabeled instance $c_i^j \sim p(\mathbf{y}^j|\mathbf{x}^j;\boldsymbol{\theta})$, and treat these values as pseudo-observations of r_i^j that are down-weighted by parameter α. We then estimate $\hat{\sigma}_i$ as in *online stratified sampling*, combining the true and pseudo-observations.

---

[2]The sum is a Binomial random variable if p is the same for each instance.

10.3.2 Classification Experiments

We compare approaches for evaluating the accuracy of document classifiers for binary subsets of *20 Newsgroups* as described in Section 4.4.2. The classifiers are trained with GE using constraints provided by users in Section 4.4.7.

We compare random sampling (*random*) and stratified sampling approaches that use confidence stratification with $m = 5$ strata and different sample allocation methods: proportional allocation (*pro conf*), optimal allocation using online variance estimation (*opt online*) [6], and optimal allocation using the combined confidence and online variance estimation method (*opt conf online*). We also conduct but do not display experiments with confidence-based optimal allocation. In general, *opt conf online* outperforms this method more as n increases. We begin stratified sampling by allocating two samples to each stratum. For the optimal allocation methods, we reestimate $\hat{\sigma}_i$ after each sample, and smooth the estimates with 10 pseudo-observations[3]; for *opt online* these pseudo-observations are uniform, while for *opt conf online* the pseudo-observations are sampled correctness values (i.e. $\alpha = 10/N_i$). To compute estimates of $\hat{\bar{r}}$, we reveal the true labels for instances in the sample. We run 1000 trials, and report the mean absolute accuracy estimation error.

Figures 10.1 and 10.2 display error vs. n. First, note that the stratified sampling approaches provide lower mean absolute error than random sampling. We assess significance with a Mann-Whitney U test, the non-parametric counterpart of an unpaired t-test. Of the 216 possible comparisons between random sampling and a stratified sampling method (9 tasks $\times$ 8 different sample sizes $\times$ 3 stratified sampling methods), stratified sampling provides significantly lower error (significance level $\alpha = 0.05$) in 209 cases. Random sampling never significantly outperforms stratified sampling. Attaining the same mean absolute error with random sampling would often require

[3]We initially smoothed with $1/\sqrt{n_i}$ pseudo-observations as in [6], but this significantly increased error for *opt online*.

significantly more effort. For example, in the *med-space* task with *User 1*'s constraints, a sample of size $n = 20$ using *opt conf online* gives error of 0.0406. *Random* attains comparable performance with 30 samples, giving an error of 0.0418. This is a 33% reduction in evaluation effort. We conclude that the accuracy of classifiers trained with GE can be estimated more efficiently using stratified sampling.

Opt conf online provides lower mean absolute error than any other method in 54 of the 72 cases (9 tasks $\times$ 8 sample sizes). Of the 162 reductions in these 54 cases, 133 are significant. *Opt conf online* significantly outperforms *opt online* 43 times, and is significantly outperformed by *opt online* only once. Error analysis reveals that *opt online* tends to overestimate the differences in variance between strata. Additional smoothing reduces error with large n, but increases error with small n, as *opt online* approaches *pro conf* as the amount of smoothing increases. We conclude that *opt conf online* is preferable for minimal effort evaluation.

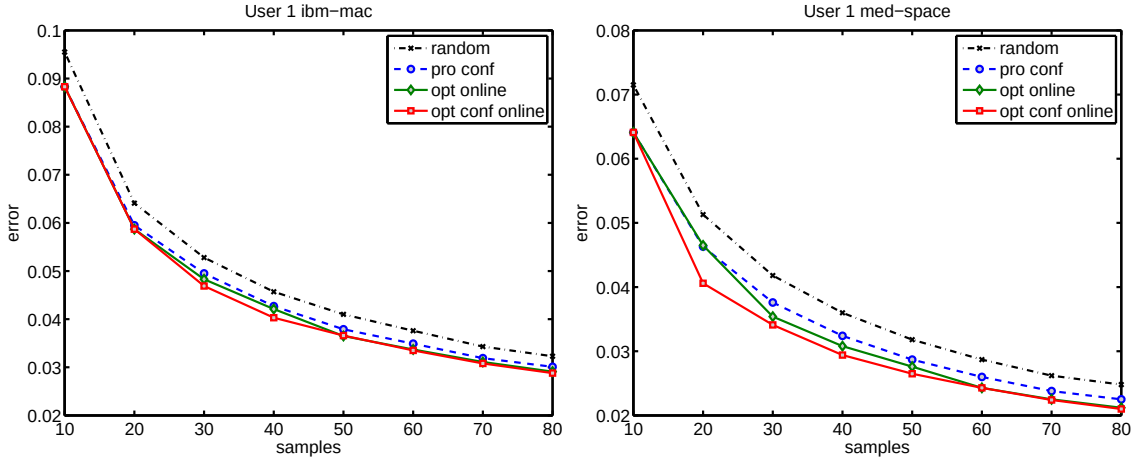
10.3.3 Estimating Sequence Token Accuracy

We next propose stratified sampling methods for evaluating token accuracy for sequence labeling. We estimate the mean of the vector-valued function $\mathbf{r}_{sc}$ defined

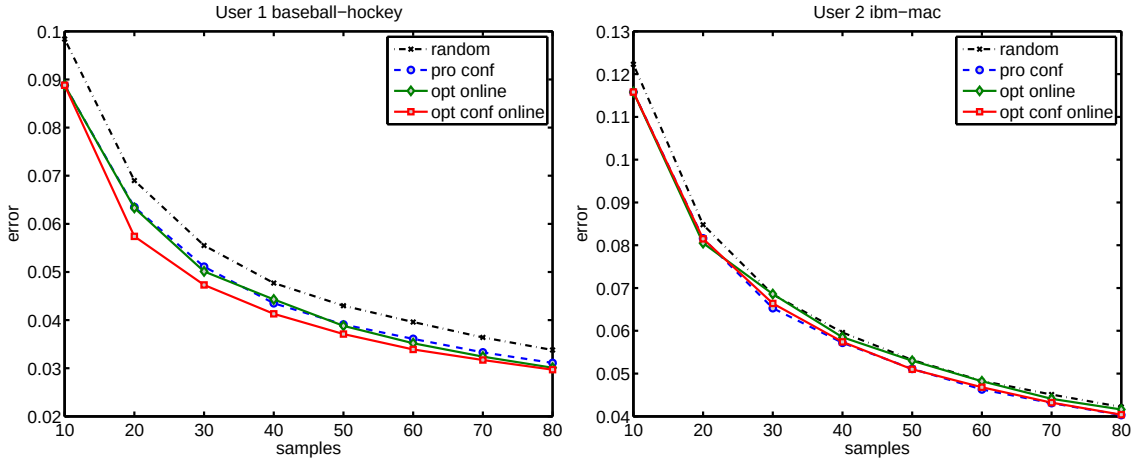
$$r_{sc0}(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^{T} 1_{\{\hat{y}_t = y_t\}}$$
$$r_{sc1}(\mathbf{x}, \mathbf{y}) = T,$$

where T is the length of the sequence and t indexes positions in the sequence. The function $\mathbf{r}_{sc}$ returns the number of correctly predicted labels in the first position, and the length in the second. We describe the components of several sampling schemes.

User 1: ibm-mac, med-space



User 1: baseball-hockey, User2: ibm-mac



User 2: med-space, baseball-hockey

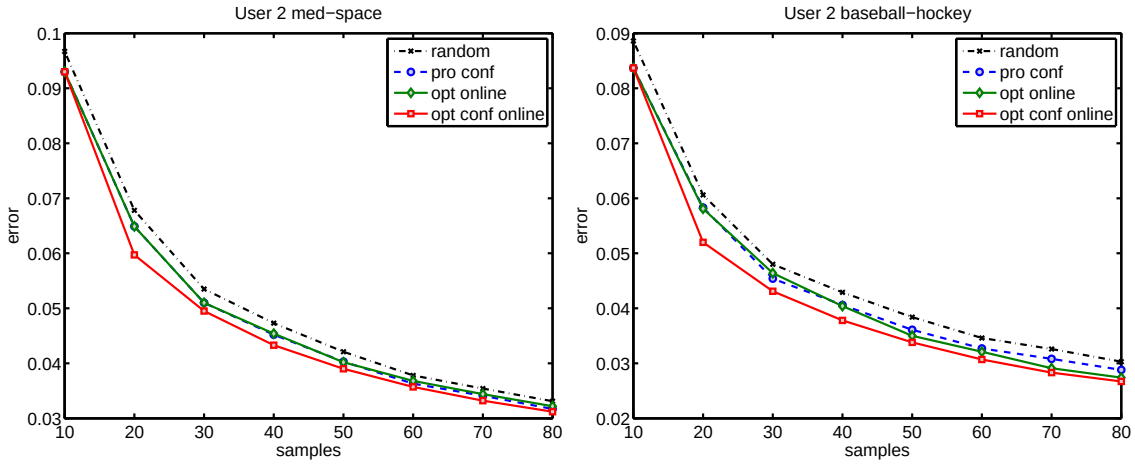


Figure 10.1: Stratified sampling methods provide classification accuracy estimates with lower error. *Opt conf online* typically outperforms the other methods.

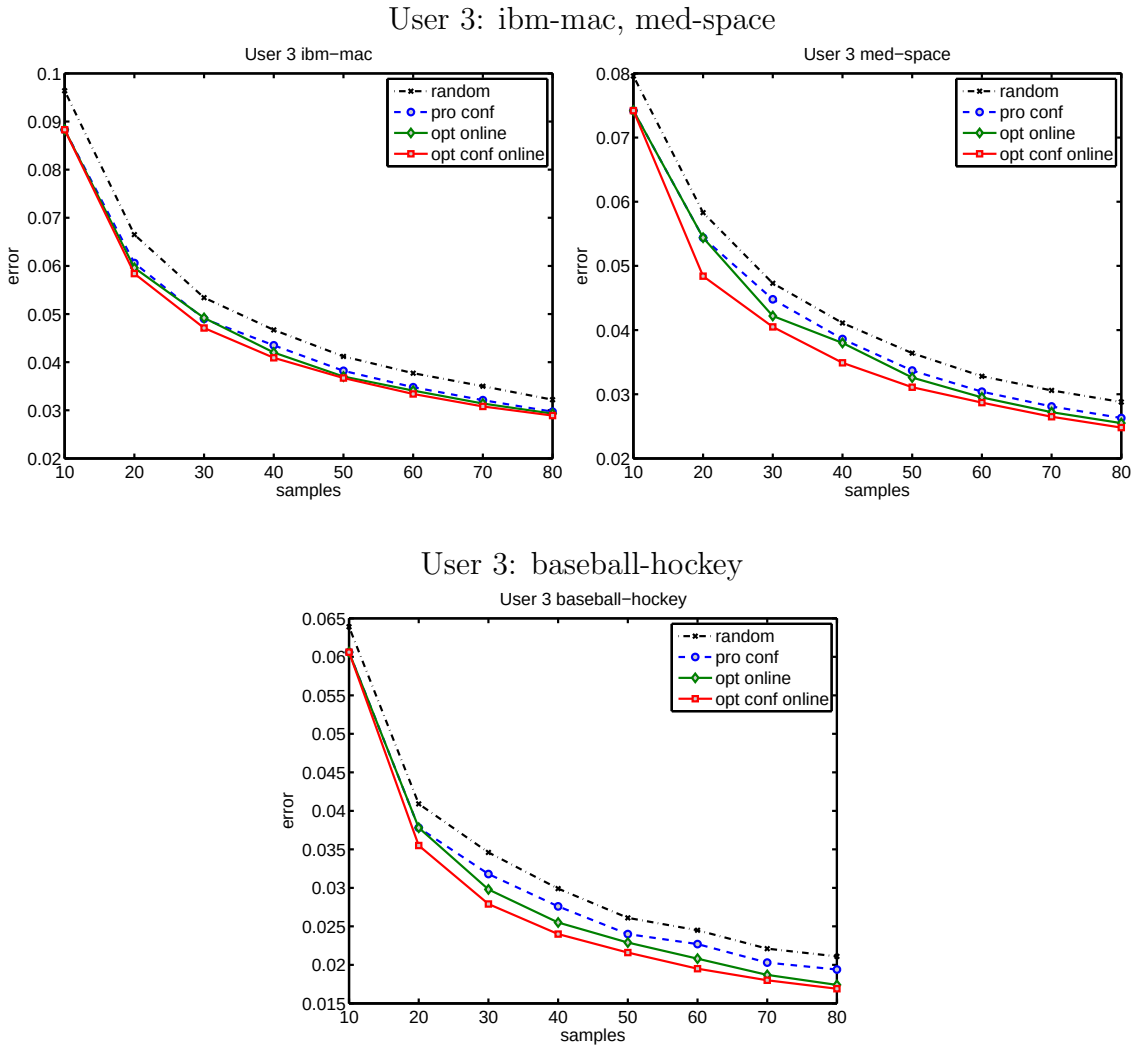


Figure 10.2: Stratified sampling methods provide classification accuracy estimates with lower error. *Opt conf online* typically outperforms the other methods.

Target Function: While instance accuracy and average token accuracy are linear, token accuracy is non-linear. Token accuracy is defined $\omega_{ta}(\hat{\hat{\boldsymbol{r}}}_{sc}) = \hat{\hat{r}}_{sc0}/\hat{\hat{r}}_{sc1}$[4].

Stratification Functions: We propose two stratification functions. The first is the expectation of r_{sc0}.

[4]We could use the population mean length in place of $\hat{\hat{r}}_{sc1}$. Stratified sampling provides even larger improvements with this estimator, but the estimates have higher error due to discrepancies between the population mean length and $\hat{\hat{r}}_{sc1}$. This may result in accuracy > 1, for example. Therefore, in this thesis we estimate all arguments to ω from S.

$$s_{exc}(\mathbf{x}^j) = \mathrm{E}_{p(\mathbf{y}|\mathbf{x}^j;\boldsymbol{\theta})}[r_{sc0}(\mathbf{x}^j, \mathbf{y})] = \sum_{t=1}^{T} p(\hat{y}_t|\mathbf{x}^j;\boldsymbol{\theta})$$

This can be interpreted as the expected number of correct tokens, where $p(\hat{y}_t|\mathbf{x}^j;\boldsymbol{\theta})$ is the marginal probability of the predicted label at position t. The stratification function s_{conf} is the expectation of a function that returns 1 only if the labeling of the entire sequence is correct.

$$s_{conf}(\mathbf{x}^j) = \mathrm{E}_{p(\mathbf{y}|\mathbf{x}^j;\boldsymbol{\theta})}[1_{\{\hat{\mathbf{y}}=\mathbf{y}\}}] = p(\hat{\mathbf{y}}|\mathbf{x}^j;\boldsymbol{\theta}).$$

The expectation is the probability of the predicted label sequence. This can be interpreted as model confidence.

Stratification Scheme: We use the *uniform size stratification scheme*, described in Section 10.3.1, for this task.

Sample Allocation: We use both proportional and optimal allocation. In optimal allocation, we allocate samples using estimates of the standard deviation of $r_{sc0}(\mathbf{x}^j, \mathbf{y})$. This is preferable to using the standard deviation of $1_{\{\hat{\mathbf{y}}=\mathbf{y}\}}$ in the applications we explore here, as complete instance accuracy is low. As in Section 10.3.1, we use both online estimation of the variance with the samples, and a combined method. In the combined method, pseudo-observations of token correctness are sampled according to the marginal probabilities of the predicted labels $p(\hat{y}_t|\mathbf{x}^j;\boldsymbol{\theta})$, and these pseudo-observations are combined with the labeled sample estimates, as described in Section 10.3.1.

10.3.4 Sequence Labeling Experiments

This experiment uses the *Cora* data set as described in Section 9.5. The model is trained with the constraints provided by User 3 in Section 9.5, and an additional constraint with weight 10 that specifies that 80% of transitions between labels should

be self-transitions. In this task each sequence is a complete citation. To enable precise sampling, we split citations into smaller subsequences. However, using very short subsequences would obscure helpful context, making the user's task more difficult. Consequently, we split citations into subsequences of maximum length 10.

The sampling strategies for this experiment are *random*, s_{conf} stratification with proportional allocation (*pro conf*), s_{exc} stratification with proportional allocation (*pro ex*), s_{conf} stratification with online optimal allocation (*opt online*), and s_{conf} stratification with combined optimal allocation (*opt ex online*). We do not report results with s_{exc} stratification and optimal allocation. The results are similar to *opt ex online*, but with higher error. For *opt online*, we used the same smoothing strategy as [6], which performed better than a fixed value of 10 for this task (the opposite was true in Section 10.3.2). We did not tune α for *opt ex online*, keeping $\alpha = 10/N_i$, giving an advantage to *opt online*. We use $m = 5$ strata, and conduct 1000 trials. To simulate rapid evaluation, we use 100-500 tokens ($\sim$10-50 subsequences).

Figure 10.3 displays the results. The x-axis is the total number of tokens evaluated. All stratified sampling methods significantly outperform random sampling (Mann Whitney U test with significance level $\alpha = 0.05$). *Opt ex online* outperforms all methods significantly at each point on the graph except for *pro conf* at 300, and *pro conf* and *opt online* at 500. At 200 tokens, the mean absolute error with *opt ex online* is 0.0309. *Random* sampling does not attain a comparable error of 0.0311 until $n = 425$. This is a savings of 225 tokens, or 53% of total evaluation effort.

In this experiment *pro conf* outperforms *pro ex*. Note that $s_{exc} \leq T$. Error analysis shows that s_{exc} yields strata that are more correlated with T than s_{conf}.

10.4 Fine-Grained Evaluation and Error Analysis

In Section 10.3, we found that stratified sampling methods can provide significantly more accurate estimates of overall performance than random sampling. One

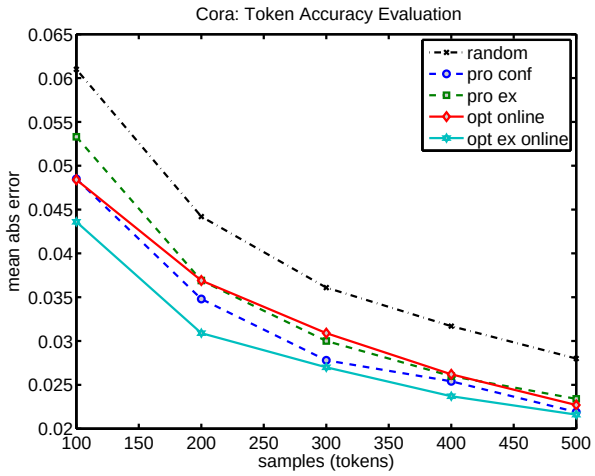


Figure 10.3: Stratified sampling methods significantly outperform random sampling for evaluating token accuracy on the *Cora* data set.

possible next step in interactive training is to drill down and perform fine-grained evaluation, with the goal of using this information to refine or provide new supervision. In this section, we propose a stratified sampling approach to fine-grained evaluation and error analysis. As an example error analysis task, we consider computing token F_1 for a particular label of interest ℓ for sequence labeling tasks. This is an important problem because in a particular application some labels may be more important than others. Additionally, awareness of low F_1 for ℓ provides a path for improving the model.

The function of interest, $\mathbf{r}_{f\ell}$, is defined as

$$r_{f\ell 0}(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^{T} 1_{\{y_t=\ell \wedge \hat{y}_t=\ell\}}$$

$$r_{f\ell 1}(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^{T} 1_{\{\hat{y}_t=\ell\}}$$

$$r_{f\ell 2}(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^{T} 1_{\{y_t=\ell\}}.$$

In words, the first element is the number of correctly predicted labels whose value is ℓ, the second element is the number of predictions of ℓ, and the third element is the number of true labels whose value is ℓ.

Target Function: The F_1 target function for label ℓ is

$$\omega_{F_1}(\hat{\hat{\boldsymbol{r}}}_{f\ell}) = \frac{2 \times \hat{\hat{r}}_{f\ell 0}/\hat{\hat{r}}_{f\ell 1} \times \hat{\hat{r}}_{f\ell 0}/\hat{\hat{r}}_{f\ell 2}}{\hat{\hat{r}}_{f\ell 0}/\hat{\hat{r}}_{f\ell 1} + \hat{\hat{r}}_{f\ell 0}/\hat{\hat{r}}_{f\ell 2}}.$$

Stratification Function: The stratification function is the expectation of $r_{f\ell 2}$

$$s_{f\ell 2}(\mathbf{x}^j) = \mathrm{E}_{p(\mathbf{y}|\mathbf{x}^j;\boldsymbol{\theta})}[r_{f\ell 2}(\mathbf{x}^j, \mathbf{y})] = \sum_{t=1}^{T} p(y_t = \ell|\mathbf{x}^j;\boldsymbol{\theta}).$$

This can be interpreted as the expected number of ℓ tokens.

Stratification Scheme: We again use the *uniform size stratification scheme*. However, we find that the density of $r_{f\ell 2}$ is less uniform than the density of the stratification functions used for overall evaluation. There are often a few $\mathbf{x}^j$ for which $s_{f\ell 2}$ is large, and many $\mathbf{x}^j$ for which $s_{f\ell 2}$ is very small. This occurs when ℓ is infrequent, for example.

Consequently, we also experiment with non-uniform size stratification. A standard method for determining stratum boundaries is the *cum* $\sqrt{F}$ *rule* [20], which aims to minimize within-stratum variance. Performing stratification according to the cum $\sqrt{F}$ rule first involves splitting the instances into k initial sorted classes, where C_i denotes the ith class. Next, the cumulative function c of the square root of the class frequencies is computed.

$$c(j) = \sum_{i=1}^{j} \sqrt{|C_i|}$$

The *stratum width* w is then computed as $w = c(k)/m$. Finally, the initial classes are grouped into strata of equal width w using the cum $\sqrt{F}$ values, $c(j)$.

Sample Allocation: We allocate two initial samples to each stratum (to enable computing $\hat{Var}(\hat{\bar{r}}_{ss})$ if needed), and allocate the remaining samples proportionally. When $n = 2m$, this can be viewed as optimal allocation with $\hat{\sigma}_i$ that are inversely proportional to the stratum sizes, $\hat{\sigma}_i = 1/N_i$. This scheme is appropriate because in this setting large strata are likely to have small $s_{f\ell 2}$ values, and consequently we expect few occurrences of ℓ in those strata.

10.4.1 Experiments

We use the same data set and initial model as in Section 10.3.4. Citations are again split into subsequences of maximum length 10. To simulate obtaining a rapid estimate of token F_1 for ℓ, we evaluate using $n = 10$ subsequences.

We compare *random* sampling, sampling proportionally from equal-sized strata using $s_{f\ell 2}$ (*ex uniform*), and sampling proportionally from strata determined by the cum $\sqrt{F}$ rule using $s_{f\ell 2}$ (*ex cum*$\sqrt{F}$). We use $m = 5$ strata. For the *cum* $\sqrt{F}$ rule, we use $k = 20$ initial classes that each cover a fixed-width segment of the range of $s_{f\ell 2}$ (i.e. one class is all $\mathbf{x}$ with $s_{f\ell 2}(\mathbf{x}) \in [0, 0.5]$). In some cases with $k = 20$ initial classes less than m of them contain instances — empirical evidence of the statement above that the density of $s_{f\ell 2}$ can be highly non-uniform. In this case we multiply k by 10 iteratively until we have at least $m + 1$ non-empty classes.

Table 10.1 reports the results. We evaluate both the mean absolute error in the F_1 estimate (F_1 *err*) and the percentage of wasted trials (*waste*). A wasted trial occurs when none of the 10 subsequences contain a true occurrence of the label of interest. In this case it is not possible to obtain a meaningful recall estimate[5]. Bold denotes that a method gives the lowest F_1 *err* or *waste*. A * in the F_1 *err* column denotes statistical significance (Mann Whitney U test with significance level $\alpha = 0.05$).

[5]Following standard conventions, recall is 1 if there are no true occurrences of ℓ in the sample, and precision is 1 if there are no predicted occurrences of ℓ in the sample.

	random		ex uniform		ex cum$\sqrt{F}$	
	F_1 err	waste	F_1 err	waste	F_1 err	waste
author	0.073	0.004	0.051	**0.000**	**0.041***	**0.000**
journal	0.273	0.069	0.221	0.006	**0.143***	**0.000**
note	0.843	0.843	0.822	0.822	**0.274***	**0.274**
booktitle	0.183	0.017	0.139	**0.000**	**0.105***	**0.000**
tech	0.392	0.501	0.363	0.364	**0.129***	**0.000**
volume	0.293	0.084	0.239	0.004	**0.091***	**0.000**
location	0.286	0.286	0.262	0.145	**0.093***	**0.000**
editor	0.510	0.605	0.468	0.488	**0.172***	**0.000**
institut.	0.351	0.427	0.339	0.257	**0.136***	**0.000**
title	0.079	**0.000**	0.056	**0.000**	**0.051**	**0.000**
date	0.123	0.006	0.080	**0.000**	**0.077**	**0.000**
pages	0.183	0.052	0.133	**0.000**	**0.081***	**0.000**
publisher	0.392	0.149	0.356	0.071	**0.157***	**0.000**

Table 10.1: Using $n = 10$ subsequences to evaluate token label F_1 for each *Cora* label. Stratified sampling provides more accurate F_1 estimates and avoids wasted samples.

The *ex cum*$\sqrt{F}$ method performs as well as or better than the other methods. In terms of F_1, it always significantly outperforms *random*, and significantly outperforms *ex uniform* in all cases except *title* and *date*. The *ex cum*$\sqrt{F}$ method also avoids a wasted sample in 1000 trials in all cases except for *note*. Using non-uniform strata with $s_{f\ell 2}$ typically results in a small stratum with instances that are very likely to contain ℓ. This greatly reduces waste. However, stratified sampling also samples other instances, ensuring that we obtain an unbiased estimate of $\bar{\mathbf{r}}^*$ even if model predictions are poorly correlated with the true labels. This illustrates the utility of stratified sampling for targeted evaluation.

10.5 Specifying New Constraints

Thus far we have focused on evaluation and analysis. We now shift our focus to improving the model. Users could specify new constraints manually, or select from candidate constraints [25, 28]. In this section we propose a new paradigm in which the user specifies constraints while inspecting data. For example, after

inspecting [*journal*: Transactions] [*title*: on Pattern Analysis] ..., the user may choose to add new constraints that specify that *Transactions* should almost always be labeled *journal*, and that transitions from *journal* to *title* are extremely unlikely. This paradigm can help the user specify more accurate constraints, find constraints that are particularly useful, and may suggest constraints to the user that they may not have considered otherwise. Note that constraints apply to the entire data set, and hence provide more supervision than labeling data [25, 28]. However, in future work we plan to allow both types of supervision. We focus on *targeted improvement* of the model. In this section we aim to improve token F_1 for a particular label ℓ.

We can view this as an estimation problem as follows. Based on their prior knowledge, the user has some set of candidate constraints that they are capable of specifying. For each instance, the function $\mathbf{r}$ simply returns the number of times each candidate constraint is applicable with respect to the targeted improvement task. Note that computing $\mathbf{r}$ is expensive, since the user must manually inspect instances. The user decides which constraints to add based on the mean candidate constraint estimate $\hat{\bar{\boldsymbol{r}}}$. When applying stratified sampling to improve label ℓ, we use the same sample allocation and cum $\sqrt{F}$ stratification method as in Section 10.4.

10.5.1 Experiments

We use the same data set and constraints as in Section 10.3.4. Initial models are trained with either the *full* set of 108 constraints or a random subsample of 52 constraints (*small*).

We simulate the specification of new constraints so that we can conduct a large number of trials. To simulate user prior knowledge, we use labeled data to define constraints as in previous work [25, 28, 66]. Candidate constraints include input feature label distribution constraints of the form used in [28] for input features that occur at least 10 times and have label distribution entropy ≤ 0.7. In addition, there are

candidate constraints that discourage unlikely label transitions[6]. In this experiment unlikely transitions are those that do not occur in the labeled data. Candidate constraints are applicable if they apply to a token with true or predicted label ℓ. Any constraint i with $\hat{\hat{r}}_i \geq 0$ is then added. For input feature label distributions, the target distribution is assigned using the "labeling" method used in [28].

We evaluate the targeted improvement of three moderately infrequent labels in the *Cora* data set: *institution*, *journal*, and *location*. We use $n = 10$ sub-sequences of length at most 10 to find constraints, and subsequently retrain the model with the augmented set of constraints. Results comparing *random* and *stratified* sampling with 100 trials are presented in Table 10.2. In all cases, stratified sampling yields significantly higher token F_1 for ℓ. The improvement is the result of finding additional applicable constraints. While specifying these constraints takes additional time, when ℓ is infrequent, we expect finding appropriate constraints to dominate the time required to specify them. As in Section 10.4, this method encourages the sample to contain a mix of correct predictions of ℓ, false positives, and false negatives. Other sampling strategies do not provide this coverage. For example, uncertainty sampling may miss true occurrences of ℓ, and certainty sampling method may miss false negatives.

10.6 Estimating Target Expectations

In this thesis, we often assign target distributions with simple heuristics. We know from Section 8.2 that as the target distributions become more precise, the resulting model becomes more accurate (c.f. [72]). Additionally, we know from Section 8.1 that

[6]The probability of taking any transition in the unlikely set is encouraged to be close to 0 using KL divergence. To balance this constraint with the others, its weight is set to the number of labeled feature constraints.

constraints	label	initial	random	stratified
small	institution	0.178	0.271	**0.448***
	journal	0.341	0.473	**0.634***
	location	0.466	0.580	**0.684***
full	institution	0.661	0.682	**0.717***
	journal	0.635	0.593	**0.671***
	location	0.655	0.685	**0.727***

Table 10.2: Using stratified sampling to find new constraints improves token F_1 for a label of interest.

users occasionally make mistakes when specifying constraints, and that such incorrect constraints can be detrimental to GE.

Incorrect constraints can be corrected and imprecise constraints can be refined by having the user view a few occurrences of the constraint feature ϕ in context. Mann and McCallum [72] found that this target estimation method gave higher accuracy than traditional sequence labeling with the same number of labels. In this section we use stratified sampling to ensure that the small number of occurrences considered by the user are representative. Note that these ideas could be applied to other expectation estimation problems.

The specific task we consider is estimating a distribution over labels for a particular input feature q. The function of interest returns a vector with the count of q with each label. For a classification task, the function $\mathbf{r}_q$ is

$$r_{q\ell}(\mathbf{x}, y) = 1_{\{y=\ell\}} q(\mathbf{x})$$

For a sequence labeling task, the function $\mathbf{r}_q$ is

$$r_{q\ell}(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^{T} 1_{\{y_t=\ell\}} q(\mathbf{x}, t)$$

Target Function: We aim to estimate the distribution over labels for each input feature. Consequently $\omega_{ex}(\hat{\tilde{\boldsymbol{r}}}_q)$ simply returns $\hat{\tilde{\boldsymbol{r}}}_q$ normalized to sum to 1.

Stratification Functions: We need only consider $\mathbf{x}^j$ where $q(\mathbf{x}^j)=1$, or $q(\mathbf{x}^j,t)=1$ for some t. For binary classification tasks, the stratification function is the expectation of r_{q0}, where 0 refers to one of the labels.

$$s_{q0}(\mathbf{x}^j) = \mathrm{E}_{p(y|\mathbf{x}^j;\boldsymbol{\theta})}[r_{q0}(\mathbf{x}^j,y)] = p(y=0|\mathbf{x}^j;\boldsymbol{\theta}),$$

where again we only consider $\mathbf{x}^j$ with $q(\mathbf{x}^j)=1$.

For a non-binary task we stratify according to the index of the most frequently predicted label ℓ_{max}.

$$\begin{aligned} s_{\ell_{max}}(\mathbf{x}^j) &= \mathrm{E}_{p(\mathbf{y}|\mathbf{x}^j;\boldsymbol{\theta})}[r_{q\ell_{max}}(\mathbf{x}^j,\mathbf{y})] \\ &= \sum_{t=1}^{T} p(y_t=\ell_{max}|\mathbf{x}^j;\boldsymbol{\theta})q(\mathbf{x}^j,t). \end{aligned}$$

In ongoing work we are developing improved, clustering-based methods for stratification for expectation estimation.

Stratification Scheme and Sample Allocation: We use the *cum* $\sqrt{F}$ *rule*, described in Section 10.4, for stratification, and the same sample allocation scheme.

10.6.1 Classification Experiments

For this experiment we use the same data sets and constraints as the experiments in Section 10.3.2. We use $m = 2$ strata, 4 samples per constraint, and repeat the experiment 1000 times with different random seeds. We evaluate using the mean absolute expectation estimation error (*err*), and the accuracy of the logistic regression model after re-training with the refined constraints (*rt acc*). Table 10.3 displays results comparing *random* sampling and stratified sampling with cum $\sqrt{F}$ rule stratification using s_{q0} (*stratified*). Stratified sampling always provides more accurate expectation estimates, and provides higher accuracy when the model is retrained with

constraint and data sets	random		stratified	
	err	rt acc	err	rt acc
(1) baseball-hockey	0.178	0.932	**0.113***	**0.948***
(1) ibm-mac	0.274	0.784	**0.244***	**0.790***
(1) med-space	0.139	0.921	**0.108***	**0.931***
(2) baseball-hockey	0.234	0.912	**0.163***	**0.932***
(2) ibm-mac	0.310	**0.782**	**0.303***	**0.782**
(2) med-space	0.205	0.912	**0.175***	**0.919***
(3) baseball-hockey	0.178	0.923	**0.119***	**0.942***
(3) ibm-mac	0.250	0.809	**0.226***	**0.816***
(3) med-space	0.112	0.922	**0.094***	**0.928***

Table 10.3: Stratified sampling provides lower error target expectation estimates, and higher accuracy when the classifier is retrained with the refined targets.

the refined constraints in all cases except one. Cases in which stratified sampling significantly outperforms random sampling are indicated with a *.

10.6.2 Sequence Labeling Experiments

Finally, we refine User 3's constraints for *Cora* with $n=4$ and $n=10$ samples of the constraint occurring in context. We use $m=2$ strata for $n=4$, $m=5$ strata for $n=10$, and conduct 100 trials. We use the same initial model as in Section 10.3.4, which has accuracy of 82.8%. Note that here strata with low $s_{\ell_{max}}$ values do not necessarily have low variance. Therefore, in this experiment we avoid over-stratifying, allowing $<m$ strata if there are fewer non-empty initial classes.

Using random sampling gives mean absolute expectation estimation error of 0.259 with $n=4$ and error of 0.171 with $n=10$. Using proportional allocation with s_{qmax} and cum $\sqrt{F}$ stratification gives error of 0.215 with $n=4$, a 17% error reduction, and error of 0.136 with $n=10$, a 20% error reduction. Retraining the model with the refined constraints obtained using *random* sampling gives accuracy of 82.5% with $n=4$ and accuracy of 86.3% with $n=10$, while retraining the model with refined constraints using stratified sampling gives statistically significantly higher accuracy of 84.0% with $n=4$ and accuracy of 87.2% with $n=10$. Random sampling requires

$n = 16$ samples per constraint to match the accuracy of stratified sampling with $n = 10$, a 37.5% reduction.

The user may instead use the sample to specify their own target expectation. In this case, we want the sample to include as many labels as possible, to remind the user of the input feature's uses. Random sampling with $n = 4$ finds 61.1% of the labels input features occur with, whereas stratified sampling finds 70.4%. With $n = 10$, random sampling finds 69.5%, whereas stratified sampling finds 82.9%.

We also conjecture that in applications where the target label distributions have higher entropy, reductions in target estimation error will yield larger improvements.

10.7 Conclusion and Future Work

In this chapter we proposed an interactive training paradigm, and focused on solving the sub-problem of selecting representative samples for user inspection. In this section we describe opportunities for future work in interactive training, propose possible approaches, and discuss key challenges.

10.7.1 Suggesting Refinements

The development of methods that automatically select constraints that require refinement or further examination is likely to be beneficial, as we have found that incorrect constraints can be detrimental. We propose two possible approaches. First, in various experiments we have observed that constraints whose model expectations are far from their target expectations after training are likely to be incorrect. This suggests selecting constraints that are least well satisfied for refinement. In addition, in some cases we can detect analytically that a pair of constraints cannot both be satisfied. As an extreme example, consider two constraints that are copies of each other but have different target expectations.

10.7.2 Contrasting Models

During interactive training it may be beneficial to assist the user in understanding how the model is changing with the addition or refinement of supervision. This may be useful for evaluating whether the model is improving, or evaluating whether some new supervision had the desired effect. This method would also be generally applicable when comparing two models in a non-interactive setting.

We propose that stratified sampling could be applied to this problem. Possible stratification metrics include the magnitude of the divergence between the predictions under each model, or the identity of the pair of predicted outputs.

A key challenge is the development of fast re-training methods that allow contrasting the models shortly after supervision is provided. In fact, fast re-training is a key challenge for interactive training in general, as it has been suggested that real-time interaction is important [33].

10.7.3 Analyzing Specific Errors

Another possible direction is developing methods to assist the user in analyzing predictions for a single example. These methods would provide insight into why the model is making particular incorrect predictions. In addition to the qualitative benefit of gaining insight into model predictions, knowing why the model is making a mistake can help one add the necessary constraints to fix the problem. We conjecture that this will allow users to provide precise supervision more quickly. Note that the proposed methods could be applicable to other, more traditional types of error analysis and model improvement paradigms, such as modifying the feature representation.

10.7.3.1 Possible Approaches

We first consider assisting the user in answering the question: *What does the model predict in other similar contexts?* We propose two methods.

- **Displaying expectations:** First, we can display expectations $\mathrm{E}_{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}[f(\mathbf{x},\mathbf{y})]$ of features f that fire within the region of interest. Both corpus and instance-specific expectations may be of interest.

- **Displaying similar contexts:** Second, we can display specific model predictions in similar contexts. For structured outputs, a context is a substructure, for example a subsequence in sequence labeling. We can measure the similarity between contexts using some distance between their expectation vectors. For example, in sequence labeling tasks a context may be a subsequence of length k starting at some position j. This context is represented by the vector $\mathrm{E}_{p(y_j,\ldots,y_{j+k}|\mathbf{x};\boldsymbol{\theta})}[\mathbf{f}(\mathbf{x},\mathbf{y})]$, and we can display model predictions for the m contexts that are closest to this vector. Alternatively, we could use stratified sampling to choose a representative sub-sample of contexts whose similarity is above some threshold.

We also propose methods to help the user in answering the question: *How does changing the input change model predictions?* This provides a tool for interactively obtaining intuition about model predictions without looking at parameter values, which are difficult to interpret.

As a quantitative measure of how modifying the input changes predictions, we use the *odds ratio*. Let $\mathbf{x}$ be the input and $\mathbf{x}'$ be the modified input. First, we consider the case in which we want to understand how changing the input changes the odds of two outputs $\mathbf{y}$ and $\mathbf{y}'$. Define the *odds* O and *odds ratio* OR as

$$O(\mathbf{y},\mathbf{y}',\mathbf{x},\boldsymbol{\theta}) = \frac{p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})}{p(\mathbf{y}'|\mathbf{x};\boldsymbol{\theta})} \tag{10.3}$$

$$OR(\mathbf{y},\mathbf{y}',\mathbf{x},\mathbf{x}',\boldsymbol{\theta}) = \frac{O(\mathbf{y},\mathbf{y}',\mathbf{x},\boldsymbol{\theta})}{O(\mathbf{y},\mathbf{y}',\mathbf{x}',\boldsymbol{\theta})}. \tag{10.4}$$

If p is a log-linear model, then Equation 10.3 can be simplified by canceling the partition functions

$$O(\mathbf{y}, \mathbf{y}', \mathbf{x}, \boldsymbol{\theta}) = \frac{s(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})}{s(\mathbf{x}, \mathbf{y}'; \boldsymbol{\theta})},$$

where $s(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) = \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}))$. In some cases we may also be interested in the marginal odds ratio $OR(\mathbf{y}_a, \mathbf{y}'_a, \mathbf{x}, \mathbf{x}', \boldsymbol{\theta})$, where a selects a subset of $\mathbf{y}$.

$$O(\mathbf{y}_a, \mathbf{y}'_a, \mathbf{x}, \boldsymbol{\theta}) = \frac{p(\mathbf{y}_a|\mathbf{x}; \boldsymbol{\theta})}{p(\mathbf{y}'_a|\mathbf{x}; \boldsymbol{\theta})} \tag{10.5}$$

$$OR(\mathbf{y}_a, \mathbf{y}'_a, \mathbf{x}, \mathbf{x}', \boldsymbol{\theta}) = \frac{O(\mathbf{y}_a, \mathbf{y}'_a, \mathbf{x}, \boldsymbol{\theta})}{O(\mathbf{y}_a, \mathbf{y}'_a, \mathbf{x}', \boldsymbol{\theta})}. \tag{10.6}$$

How should we choose the $\mathbf{x}$ and $\mathbf{y}$ arguments above? All arguments may be specified by the user, but we also propose some default methods. A simple method to set $\mathbf{y}$ and $\mathbf{y}'$ is to use the two most probable outputs. A simple strategy for choosing $\mathbf{x}'$, or how to modify the input, is to remove some portion of $\mathbf{x}$. For example, for a document classification task we can modify $\mathbf{x}$ by removing an individual word or an entire sentence from the document. For sequence labeling, we can consider "masking" a sub-sequence by replacing the word with a special OOV token. Viewing how model predictions change when removing or masking certain parts of the input can provide the user with intuition for how the model makes predictions.

However, we must exercise care so that we avoid modifying the input in a way that is very unlikely according to the observed data. Otherwise, we may encounter *interaction* issues. We describe interaction with an example. Suppose we have two features f_i and f_j that are highly correlated — i.e. they often fire for the same $(\mathbf{x}, \mathbf{y})$. Because the features often occur together, it is possible for f_i and f_j to have similar expectations but very different parameter values λ_i and λ_j. Supposing $\lambda_i >> \lambda_j$, zeroing f_j may appear to have little effect on model predictions. This result is misleading because f_j is as useful as f_i. Consequently, if we make an unlikely modification, we may not get an accurate estimate of the importance of the modified part of the original input.

We do not believe this is a limiting issue, as we can automatically detect unlikely modifications. When the user or system modifies the input, we can compute the mean feature covariance between the modified portion and the rest of the input, and notify the user if it exceeds some fixed threshold.

10.7.3.2 Pilot Experiment

In Section 10.7.3.3, we provide a case study using the proposed methods. In this section, we describe a quantitative pilot experiment. We use the *movie reviews* data set, as processed in Section 4.4.2, and conduct the experiment using Amazon Mechanical Turk[7]. The experiment tests whether displaying expectations can result in more useful constraints.

In *setting 1*, 10 users are provided 10 reviews that the model is predicting incorrectly and the true label for that review. They are then asked to select three words from the review that are strong indicators of the true label, and that they would expect to generalize beyond this particular review. In *setting 2* 10 users are again provided 10 reviews, but now the words in the review that have highest input feature label expectation for each label are colored red (positive) and blue (negative). Users are asked to choose words that are not already colored, if possible.

We use the words the users provide to create sets of labeled input feature constraints as in Chapter 4. We then compare the accuracy of the original model, trained with 1600 labeled instance, with models trained with labeled data and additional constraints. We choose hyperparameters, the Gaussian prior variance σ^2 and the weight of the GE term, using 5-fold cross validation. The accuracy of the original model is 85.5%. With the constraints provided by the users in *setting 1*, the mean accuracy improves to 86.0%. With the constraints provided by users in *setting 2*, the mean accuracy improves to 87.0%. This provides some initial evidence that displaying ex-

[7]https://www.mturk.com

pectations encourages the specification of more useful constraints. For additional evidence, see also the *grid interface* experiment in Section 9.4.1.

10.7.3.3 Case Study

In this section we conduct two case studies using the Cora data set as processed in Section 9.4. For simplicity, the questions we ask only consider one label, though the methods are not limited to this case.

- ***author* vs. *editor* error:** After generating a summary of model predictions, we find an example, Example 1 in Table 10.4, where an *editor* segment is being labeled *author*. "(Eds .)" are the tokens that should tell the model to label this segment *editor*. We first look at the marginal probability at the token "Eds". The most probable labels include $author = 0.62$, $date = 0.21$, and $editor = 0.15$.

  We next ask how the labeling of "Eds." would change if we modified the input by masking some neighboring words. We compute the odds ratio of *author* vs. *editor* and *date* vs. *editor*. In both cases *editor* has lower probability than the alternative, so a value < 1 indicates that masking this word decreases the odds of *editor*. We see from this analysis, presented in Table 10.4, that masking any token in the parenthesized block increases the odds of "Eds" being labeled *author*, though not necessarily by much. Masking the tokens within the parenthesis increases the odds of "Eds" being labeled *date*.

  Next, we analyze expectations of features that fire at this position. Relevant expectations are displayed in Table 10.4. We observe that the model does expect "Eds" to be labeled *editor*, but gives nearly as high expectation to *author*. We also observe that the model expects a *date* when the previous token is ")".

  Next, we analyze other tokens. At the token "Clinger", the marginal is $author = 0.82$, $editor = 0.15$. Relevant expectations that overlap with this position are

listed in Table 10.4. Here we observe that the model favors *author* for tokens with "Eds" in the right context. Additionally, the model expects tokens that appear in a last name list to almost always be *author*. The expectation of *editor* is less than 0.05. We also look at expectations of features at the first token, where we find that the model expects *author* to begin the sentence 98% of the time. Finally, to determine the extent of this problem, we then ask the system for a similar context, and obtain the context displayed in Table 10.4.

Based on the above analysis, we specify new constraints that encourage "Eds" and any tokens with "Eds" one or two tokens to the right to be labeled *editor*. Adding these constraints and re-training with GE was sufficient to change the model's predictions to *editor* for these segments. Not only did the proposed methods allow us to understand how the model makes predictions, but they also helped us to a choose a constraint to address a particular type of error. In previous work [15], where constraints were selected without the use of these methods, constraints on "Eds" were not included.

- **journal vs. booktitle error**: Another example in the summary had a *journal* segment mislabeled as a *booktitle*. This example, labeled Example 2, is displayed in Table 10.4. Intuition might lead one to believe that what is needed a constraint on the word "Transactions", which we expect to be highly indicative of the *journal*. Surprisingly, the model already expects the label of "Transactions" to be *journal* 90% of the time. In fact, the marginals across the entire segment give more probability to *journal* than *booktitle*, for example 59% vs. 40% for "Transactions". Masking any word in the segment with the exception of "on" decreases the odds of *journal*. If we look at the expectation of the default feature for a transition between *journal* and *pages*, we find the problem. This transition is very unlikely according to the model, with probability of 0.6% of *pages* given *journal*. According to the model, a *journal* token primarily transitions to

another *journal* token (78.4%) or to a *volume* token (19.1%). Consequently, we may add a constraint that specifies that transitions between *journal* and *pages*, though unlikely, are possible. Importantly, through analysis we find that our intuitions about model predictions were incorrect, and this insight allows us to specify a precise constraint to address the problem.

Example 1
[*author* Rees, J . and W . Clinger (Eds .) .] [*title* The revised 3 report on the algorithmic language Scheme .] [*journal* SIGPLAN Notices] [*volume* 21 (12) , 1986 ,] [*pages* 37 - 79 .]

Example 1 Related Context (centered on "Eds")
. . . [*note* desJardins ,] [*author* M . , & Gordon , D . F . (Eds .) .] . . .

Example 2
[*author* A . J . Bernstein .] [*title* Analysis of programs for parallel processing .] [*booktitle* IEEE Transactions on Electronic Computers ,] [*pages* pages 757 - 763 ,] [*date* October 1966 .]

OR vs. *editor* for "Eds"		
masked token	*author*	*date*
"Clinger"	3.13	1.33
"("	0.32	1.97
"Eds"	0.85	0.19
"."	0.69	0.51
")"	0.20	2.46

Expectations at "Eds"	
word	expectation
W=Eds	*editor*: 0.40 *author*: 0.38 *title*: 0.18
W=(@-1	*date*: 0.65 *volume*: 0.21 *booktitle*: 0.05

Expectations at "Clinger"	
word	expectation
W=Eds@2	*author*: 0.55 *editor*: 0.31 *title*: 0.18
LASTNAME	*author*: 0.87

Table 10.4: Case study. The upper tables contains the examples themselves, and a related context for Example 1. The lower table contain relevant answers to questions about changing the input and model expectations. Section 10.7.3.3 discusses this information and its implications in detail.

CHAPTER 11

LIMITATIONS AND KNOWN ISSUES

In this chapter, we briefly discuss the limitations of and known issues with the lightly supervised learning methods used in this thesis.

The proposed methods assume that users have knowledge of feature expectations. In tasks where the data is text, this assumption is often reasonable, as we have shown. However, this assumption may be unreasonable for other problems. For example, features in computer vision tasks are often complex transformations of small patches of pixels. It is unlikely that users will have strong intuitions about the expectations of such features. The proposed methods may not be applicable in such settings.

Models in this thesis use large numbers of features. This is advantageous because it allows GE to learn about many features, improving generalization, and because it provides more flexibility during optimization. GE may work less well with models with few features. In this case it may be beneficial to augment the model features, for example with additional cluster features as in [45].

The utility of a particular set of constraints depends on subtle interactions among them. For example, we have found cases in which a small set of carefully selected constraints can provide higher accuracy than a much larger set, in which particular combinations of constraints yield surprisingly low accuracy, and in which the addition of a few carefully selected constraints greatly improves the accuracy of a constraint set. These observations motivate the study of interactive training, the hope being that interaction will help users to construct more useful constraint sets. In practice, we have found that better accuracy is obtained when constraints are *balanced* among

labels and provide broad *coverage* of the feature space. Making precise statements about desirable properties of constraint sets is a challenging problem for future work.

This thesis used input feature label distribution constraints, which are normalized by the input feature count (see Section 3.1.1). This makes the expectations easier for the user to interpret, but additionally seems to be beneficial in numerical optimization. We have observed cases in which using counts instead of distributions with an L_2^2 score function yields low accuracy, as in this case frequent constraint features can dominate the objective function.

CHAPTER 12

CONCLUSION AND FUTURE WORK

The eventual goal of the line of research pursued in this thesis is to enable a domain expert to train an accurate model for a task of interest in a matter of hours rather than months. In this thesis we took several steps toward this goal. We contributed to the development of the Generalized Expectation framework, which allows a practitioner to incorporate their prior knowledge into learning in a natural and intuitive way. We applied GE to a variety of tasks, demonstrating that it is often possible to train accurate models with minimal human effort. We then took first steps toward developing interactive training systems that actively solicit prior knowledge and provide assistance in evaluating and refining the model.

We view the following research directions, several of which we have begun exploring in this thesis, as being important for making additional progress.

There are several possible directions for further developing the GE framework.

- This thesis focused exclusively on discriminative models, though GE could also be applied to generative models. For example, input feature label distribution constraints could be used to compensate for model misspecification, allowing a practitioner to directly guide latent structure discovery.

- There is much interest in the machine learning community in online learning. Online GE training is not straightforward, as making an update for a single instance requires computing expectations over the entire data set, and those expectations change with each parameter update. However, recent work sug-

gests that the development of an online version of GE may be possible [90]. Online GE could help to reduce latency in an interactive setting.

Constraints that are provided by users or derived from existing resources are often noisy, and noisy constraints reduce accuracy. We explored approaches for compensating for noise in Section 8.1, but there are many directions for future work.

- The ideal GE objective function seems to depend on both the application and the type of noise. It would be beneficial to have a better characterization of where particular objectives can be expected to work well.

- Rather than compensating for imprecision, it may be possible to obtain more precise constraints. Additional study is needed to determine the maximum level of precision that users can provide. Note that interactive systems can also facilitate this by aiding the user in specifying constraints.

- It may instead be possible to develop constraints that are more natural for users to specify, such as inequality constraints among pairs of constraint features.

Interactive training systems may help users provide more useful, precise supervision. We discuss future directions in active learning, including soliciting supervision at multiple granularities, in Section 9.6. We discuss future directions in interactive training, including developing methods for suggesting constraint refinement, contrasting models, and analyzing specific errors, in detail in Section 10.7. Finally, it is important to begin building and deploying prototype interactive training systems in order to identify additional directions for future work.

BIBLIOGRAPHY

[1] Galen Andrew and Jianfeng Gao. Scalable training of L1-regularized log-linear models. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 33–40, 2007.

[2] Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50:5–43, 2003.

[3] Amit Bagga and Breck Baldwin. Entity-based cross-document coreferencing using the vector space model. In *Proceedings of the 17th international conference on Computational linguistics - Volume 1*, pages 79–85, 1998.

[4] M. Belkin, V. Sindhwani, and P. Niyogi. Manifold regularization: a geometric framework for learning from examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

[5] K. Bellare, G. Druck, and A. McCallum. Alternating projections for learning with expectation constraints. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 35–42, 2009.

[6] Paul N. Bennett and Vitor R. Carvalho. Online stratified sampling: evaluating classifiers at web-scale. In *ACM Conference on Information and Knowledge Management (CIKM)*, pages 1581–1584, 2010.

[7] Taylor Berg-Kirkpatrick, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. Painless unsupervised learning with features. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 582–590, 2010.

[8] Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

[9] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[10] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Association for Computational Linguistics*, 2007.

[11] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *COLT: Proceedings of the Workshop on Computational Learning Theory*, 1998.

[12] Rens Bod. An all-subtrees approach to unsupervised parsing. In *Proceedings of the Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics (COLING-ACL)*, pages 865–872, 2006.

[13] D. J. Holmes C. J. Skinner and D. Holt. Multiple frame sampling for multivariate stratification. *International Statistical Review*, 62(3):333–347, 1994.

[14] Andrew Carlson. *Coupled Semi-Supervised Learning*. PhD thesis, Carnegie Mellon University, 2010.

[15] M. Chang, L. Ratinov, and D. Roth. Guiding semi-supervision with constraint-driven learning. In *Proc. of Meeting of Assoc. for Computational Linguistics*, pages 280–287, 2007.

[16] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, 2006.

[17] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[18] Fabio Cozman and Ira Cohen. Risks of semi-supervised learning: How unlabeled data can degrade performance of generative classifiers. In *Semi-Supervised Learning*. MIT Press, 2006.

[19] Aron Culotta, Trausti Kristjansson, Andrew McCallum, and Paul Viola. Corrective feedback and persistent learning for information extraction. *Artificial Intelligence*, 170(14):1101–1122, 2006.

[20] T. Dalenius and J.L. Hodges. Minimum variance stratification. *J. Amer. Statist. Ass.*, 30(219-229), 1959.

[21] Aynur Dayanik, David D. Lewis, David Madigan, Vladimir Menkov, and Alexander Genkin. Constructing informative prior distributions from domain knowledge in text classification. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 493–500, 2006.

[22] R. Debusmann, D. Duchier, A. Koller, M. Kuhlmann, G. Smolka, and S. Thater. A relational syntax-semantics interface based on dependency grammar. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, 2004.

[23] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society Ser. B*, 39, 1977.

[24] Doug Downey and Oren Etzioni. Look ma, no hands: Analyzing the monotonic feature abstraction for text classification. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

[25] Gregory Druck, Gideon Mann, and Andrew McCallum. Learning from labeled features using generalized expectation criteria. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2008.

[26] Gregory Druck, Gideon Mann, and Andrew McCallum. Semi-supervised learning of dependency parsers using generalized expectation criteria. In *Joint conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL-IJCNLP)*, 2009.

[27] Gregory Druck and Andrew McCallum. Toward interactive learning and evaluation. In *ACM Conference on Information and Knowledge Management (CIKM)*, 2011.

[28] Gregory Druck, Burr Settles, and Andrew McCallum. Active learning by labeling features. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP 2009)*, pages 81—90, 2009.

[29] Gregory Druck and David Smith. Computing conditional feature covariance in non-projective tree conditional random fields. Technical Report UM-CS-2009-060, University of Massachusetts, 2009.

[30] Miroslav Dudik, Steven J. Phillips, and Robert E. Schapire. Maximum entropy density estimation with generalized regularization and an application to species distribution modeling. *Journal of Machine Learning Research*, 8:1217–1260, 2007.

[31] Bradley Efron. *The Jackknife, the Bootstrap, and Other Resampling Plans*. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial Mathematics, 1987.

[32] J. Eisner and N.A. Smith. Parsing with soft and hard constraints on dependency length. In *12th International Conference on Parsing Technologies (IWPT)*, 2005.

[33] Jerry Alan Fails and Dan R. Olsen Jr. Interactive machine learning. In *IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces*, pages 39–45, 2003.

[34] Kuzman Ganchev, Jennifer Gillenwater, and Ben Taskar. Dependency grammar induction via bitext projection constraints. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 369–377, 2009.

[35] Kuzman Ganchev, Joao Graca, John Blitzer, and Ben Taskar. Multi-view learning over structured and non-identical outputs. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2008.

[36] Kuzman Ganchev, João Graça, Jennifer Gillenwater, and Ben Taskar. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*, 11:2001–2049, July 2010.

[37] Andrew Gelman. Method of moments using Monte Carlo simulation. *Journal of Computational and Graphical Statistics*, 4(1):36–54, 1995.

[38] Roger L. Berger George Casella. *Statistical inference*. Thomson Learning, 2nd edition, 2002.

[39] J. Graça, K. Ganchev, and B. Taskar. Expectation maximization and posterior constraints. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems (NIPS)*. MIT Press, 2008.

[40] Joao Graca, Kuzman Ganchev, Ben Taskar, and Fernando Pereira. Posterior vs parameter sparsity in latent variable models. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 664–672. 2009.

[41] Y. Grandvalet and Y. Bengio. Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2004.

[42] Trond Grenager, Dan Klein, and Christopher D. Manning. Unsupervised learning of field segmentation models for information extraction. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2005.

[43] Andreas Griewank and George F. Corliss, editors. *Automatic Differentiation of Algorithms*. Society for Industrial and Applied Mathematics, 1991.

[44] A. Haghighi and D. Klein. Prototype-driven grammar induction. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, 2006.

[45] Aria Haghighi and Dan Klein. Prototype-driven learning for sequence models. In *HTL-NAACL*, 2006.

[46] Aria Haghighi and Dan Klein. Coreference resolution in a modular, entity-centered model. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 385–393, 2010.

[47] Judith K. Hellerstein and Guido W. Imbens. Imposing moment restrictions from auxiliary data by weighting. *The Review of Economics and Statistics*, 81(1):1–14, 1999.

[48] Yifen Huang and Tom M. Mitchell. Text clustering with extended user feedback. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 413–420, 2006.

[49] Rebecca Hwa. Sample selection for statistical grammar induction. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora*, EMNLP '00, pages 45–52, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[50] Feng Jiao, Shaojun Wang, Chi-Hoon Lee, Russell Greiner, and Dale Schuurmans. Semi-supervised conditional random fields for improved sequence segmentation and labeling. In *Proceedings of the Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics (COLING-ACL)*, pages 209–216, 2006.

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

[52] R. J. Kate and R. J. Mooney. Semi-supervised learning for semantic parsing using support vector machines. In *HLT-NAACL (Short Papers)*, 2007.

[53] D. Klein and C. Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004.

[54] T. Koo, X. Carreras, and M. Collins. Simple semi-supervised dependency parsing. In *Proc. of Meeting of Assoc. for Computational Linguistics*, pages 595–603, 2008.

[55] Terry Koo, Amir Globerson, Xavier Carreras, and Michael Collins. Structured prediction models via the matrix-tree theorem. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 141–150, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[56] D. Krewski and J. N. K. Rao. Inference from stratified samples: Properties of the linearization, jackknife and balanced repeated replication methods. *The Annals of Statistics*, 9(5):1010–1019, 1981.

[57] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[58] Mohit Kumar, Rayid Ghani, Mohak Shah, Jaime G. Carbonell, and Alexander I. Rudnicky. Framework for interactive classification problems. In *ICML Workshop on Combining Learning Strategies to Reduce Label Cost*, 2011.

[59] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. Int'l. Conf. on Machine Learning*, pages 282–289, 2001.

[60] John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[61] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224, 1988.

[62] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 3–12, 1994.

[63] D.D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1994.

[64] Zhifei Li and Jason Eisner. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 40–51, Singapore, August 2009.

[65] P. Liang and M. I. Jordan. An asymptotic analysis of generative, discriminative, and pseudolikelihood estimators. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2008.

[66] P. Liang, M. I. Jordan, and D. Klein. Learning from measurements in exponential families. In *Proc. Int'l. Conf. on Machine Learning*, pages 641–648, 2009.

[67] B. Liu, X. Li, W. Lee, and P. Yu. Text classification by labeling words. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 2004.

[68] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Math. Programming*, 45(3, (Ser. B)):503–528, 1989.

[69] Daniel Lowd and Pedro Domingos. Efficient weight learning for markov logic networks. In *Proceedings of the 11th European conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD 2007, pages 200–211. Springer-Verlag, 2007.

[70] R. Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *In Sixth Conf. on Natural Language Learning*, pages 49–55, 2002.

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

[72] G. Mann and A. McCallum. Generalized expectation criteria for semi-supervised learning of conditional random fields. In *Proc. of Meeting of Assoc. for Computational Linguistics*, pages 870–878, 2008.

[73] A. McCallum and K. Nigam. Employing em in pool-based active learning for text classification. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1998.

[74] D. McClosky, E. Charniak, and M. Johnson. Effective self-training for parsing. In *Proceedings of the Conference on Human Language Technology and North American chapter of the Association for Computational Linguistics (HLT-NAACL 2006)*, 2006.

[75] Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 91–98, 2005.

[76] Ryan McDonald and Giorgio Satta. On the complexity of non-projective data-driven dependency parsing. In *Proc. of IWPT*, pages 121–132, 2007.

[77] Bernard Merialdo. Tagging english text with a probabilistic model. *Computational Linguistics*, 20(2):155–171, 1994.

[78] Tom Minka. Divergence measures and message passing. Technical Report MSR-TR-2005-173, Microsoft Research, 2005.

[79] K. P. Murphy, Y. Weiss, and M. Jordan. Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *Uncertainty in Artificial Intelligence*, pages 467–475, 1999.

[80] Tahira Naseem, Harr Chen, Regina Barzilay, and Mark Johnson. Using universal linguistic knowledge to guide grammar induction. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1234–1244, 2010.

[81] Radford Neal and Geoffrey Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In Michael I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. MIT Press, 1999.

[82] Jerzy Neyman. On the two different aspects of the representative method: The method of stratified sampling and the method of purposive selection. *Journal of the Royal Statistical Society*, 97(4):558–625, 1934.

[83] Kamal Nigam, Andrew McCallum, and Tom Mitchell. Semi-supervised text classification using em. In *Semi-Supervised Learning*. MIT Press, 2006.

[84] Joakim Nivre. Dependency grammar and dependency parsing. Technical Report MSI report 05133, Växjö University: School of Mathematics and Systems Engineering, 2005.

[85] Adam Pauls, John Denero, and Dan Klein. Consensus training for consensus decoding in machine translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1418–1427, 2009.

[86] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 2nd edition, 1988.

[87] Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, April 1997.

[88] Hoifung Poon and Pedro Domingos. Joint inference in information extraction. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 1*, pages 913–918, 2007.

[89] Hoifung Poon and Pedro Domingos. Joint unsupervised coreference resolution with markov logic. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 650–659, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[90] N. Quadrianto, J. Petterson, and A. Smola. Distribution matching for transduction. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Neural Information Processing Systems*, pages 1500–1508. MIT Press, 2009.

[91] Novi Quadrianto, Alex J. Smola, Tiberio S. Caetano, and Quoc V. Le. Estimating labels from label proportions. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2008.

[92] Lawrence Rabiner. A tutorial on HMM and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[93] Hema Raghavan and James Allan. An interactive algorithm for asking and incorporating feature feedback into support vector machines. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 79–86, 2007.

[94] Hema Raghavan, Omid Madani, and Rosie Jones. Active learning with feedback on features and instances. *Journal of Machine Learning Research*, 7:1655–1686, 2006.

[95] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *CoNLL '09: Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155, 2009.

[96] Dan Roth and Kevin Small. Interactive feature space construction using semantic information. In *CoNLL*, 2009.

[97] Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 441–448, 2001.

[98] Ruslan Salakhutdinov, Sam Roweis, and Zoubin Ghahramani. Optimization with EM and expectation-conjugate-gradient. In *Proceedings of the International Conference on Machine Learning*, volume 20, pages 672–679, 2003.

[99] Christoph Sawade, Niels Landwehr, Steffen Bickel, and Tobias Scheffer. Active risk estimation. In *Proceedings of the International Conference on Machine Learning*, 2010.

[100] Christoph Sawade, Niels Landwehr, and Tobias Scheffer. Active evaluation of f-measures. In *Advances in Neural Information Processing Systems*. 2010.

[101] R. Schapire, M. Rochery, M. Rahim, and N. Gupta. Incorporating prior knowledge into boosting. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2002.

[102] Yoav Seginer. Fast unsupervised incremental parsing. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 384–391, Prague, Czech Republic, 2007.

[103] Burr Settles. Active learning literature survey. Technical Report 1648, University of Wisconsin - Madison, 2009.

[104] Burr Settles. Closing the loop: Fast, interactive semi-supervised annotation with queries on features and instances. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1467–1478, 2011.

[105] Burr Settles. Addendum to emnlp 2011 paper "closing the loop...". 2012.

[106] Burr Settles and Mark Craven. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2008.

[107] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Computational Learning Theory*, pages 287–294, 1992.

[108] Vikas Sindhwani and Prem Melville. Document-word co-regularization for semi-supervised sentiment analysis. In *Proceedings of IEEE International Conference on Data Mining*, 2008.

[109] Vikas Sindhwani, Prem Melville, and Richard D. Lawrence. Uncertainty sampling and transductive experimental design for active dual supervision. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2009.

[110] Sameer Singh, Limin Yao, Sebastian Riedel, and Andrew McCallum. Constraint-driven rank-based learning for information extraction. In *Human Language Technologies: Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*, pages 729–732, 2010.

[111] David A. Smith and Jason Eisner. Bootstrapping feature-rich dependency parsers with entropic priors. In *EMNLP-CoNLL*, pages 667–677, 2007.

[112] David A. Smith and Noah A. Smith. Probabilistic models of nonprojective dependency trees. In *EMNLP-CoNLL*, pages 132–140, 2007.

[113] Noah A. Smith. *Novel Estimation Methods for Unsupervised Discovery of Latent Structure in Natural Language Text*. PhD thesis, Johns Hopkins University, 2006.

[114] Noah A. Smith and Jason Eisner. Contrastive estimation: training log-linear models on unlabeled data. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 354–362, 2005.

[115] Noah A. Smith and Jason Eisner. Annealing structural bias in multilingual weighted grammar induction. In *Proceedings of the Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics (COLING-ACL)*, pages 569–576, 2006.

[116] Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 2011. To appear.

[117] Cynthia A. Thompson, Mary Elaine Califf, and Raymond J. Mooney. Active learning for natural language parsing and information extraction. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML-99)*, pages 406–414, Bled, Slovenia, June 1999.

[118] Stephen K. Thompson. *Sampling*. Wiley-Interscience, 2002.

[119] Sudheendra Vijayanarasimhan and Kristen Grauman. Multi-level active prediction of useful image annotations for recognition. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

[120] S. V. N. Vishwanathan, Nicol N. Schraudolph, Mark W. Schmidt, and Kevin P. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 969–976, 2006.

[121] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1:1–305, January 2008.

[122] Qin Iris Wang, Dale Schuurmans, and Dekang Lin. Semi-supervised convex training for dependency parsing. In *Proceedings of ACL-08: HLT*, pages 532–540, 2008.

[123] Malcolm Ware, Eibe Frank, Geoffrey Holmes, Mark Hall, and Ian H. Witten. Interactive machine learning: letting users build classifiers. *Int. J. Hum.-Comput. Stud.*, 56(3):281–292, 2002.

[124] Michael Wick, Khashayar Rohanimanesh, Kedare Bellare, Aron Culotta, and Andrew McCallum. Samplerank: training factor graphs with atomic gradients. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2011.

[125] X. Wu and R. K. Srihari. Incorporating prior knowledge with weighted margin support vector machines. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2004.

[126] Emine Yilmaz, Evangelos Kanoulas, and Javed A. Aslam. A simple and efficient sampling method for estimating AP and NDCG. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 603–610, 2008.

[127] Omar F. Zaidan and Jason Eisner. Modeling annotators: A generative approach to learning from annotator rationales. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Honolulu, October 2008.

[128] X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005.

[129] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2003.

[130] Xiaojin Zhu, John Lafferty, and Zoubin Ghahramani. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2003.